Experimental and Computational Assessment of Tailings Binder Matrices for Construction Purposes in Cold Regions

Ali A. Mahmood

A Thesis

in

The Department

of

Building, Civil and Environmental Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy at

Concordia University

Montreal, Quebec, Canada

February 2012

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Ali A. Mahmood**

Entitled: **Experimental and Computational Assessment of Tailings Binder Matrices for Construction Purposes in Cold Regions**

and submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY (Civil Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. M.Y. Chen

_____ External Examiner
Dr. A. Foriero

_____ External to Program
Dr. A. Bulgak

_____ Examiner
Dr. J. Hadjinicolaou

_____ Examiner
Dr. A.M. Hanna

_____ Thesis Supervisor
Dr. M. Elektorowicz

Approved by _____
Dr. M. Elektorowicz, Graduate Program Director

February 16, 2012

_____
Dr. Robin A.L. Drew, Dean
Faculty of Engineering & Computer Science

Abstract

Experimental and Computational Assessment of Tailings Binder Matrices for Construction Purposes in Cold Regions

Ali A. Mahmood, Ph.D.
Concordia University, 2012

Mine tailings are the waste materials of the mining industry. They are typically disposed of in tailings ponds surrounded by tailings dams. This traditional method of disposal has caused severe environmental damage throughout the years. In this study a new approach of sustainable development of tailings is attempted. The study consisted of two phases – experimental and computational. In Phase 1, six different types of tailings are gathered from mines in Eastern Canada and subjected to a series of laboratory tests. Tailings were stabilized using different compositions of binder materials: Portland cement, slag, fly ash along with a new type of binder called Calsifrit. These experiments aimed at verifying the suitability of tailing-binder matrices as road construction material. Furthermore, weathering tests assessed feasibility of using the matrices in cold regions.

In Phase 2 a computational program was developed using the Discrete Element Method to support the engineer's decision with regards to the application of the binder tailing materials in construction.

Experimental results show that these tailings binder matrices passed the freezing/thawing durability and TCLP tests. In addition, these matrices sustained high compression loads. Using these results, a statistical equation is developed to predict the unconfined

compressive strength of the tailings binder matrices. Simulations show that the computer program developed was able to model successfully the unconfined compressive strength and freezing/thawing durability characteristics of the tailings binder matrices.

Dedication

I dedicate this thesis to the late civil engineers: my uncle Fouad Al-Talibi and my friend Mohamed-Amin Bahrul-Uloom.

Acknowledgement

I would like to express my sincere gratitude to my parents and my brother Ahmed for their great moral and sometimes financial support throughout the course of my study. I will never forget this.

Special gratitude is also extended to my thesis supervisor, Prof. Maria Elektorowicz, for her relentless efforts and patience in guiding me throughout the course of this research. Her comments and invaluable remarks and feedback were pivotal not only for my research, but for the whole course of my academic life within the university.

Special thanks is also extended to Dr. Michelle Nokken and Prof. Sabah Alkass for the support that both have provided. Dr. Nokken's laboratory facilities were instrumental in conducting the experiments of this research and the moral support that Prof. Alkass has granted me can only be surpassed by that of my thesis supervisor.

My sincere thanks and gratitude are also extended to my dearest friend and former roommate Mr. Ahmed Haddad. He is the single factor that made my life in Montreal pleasant and enjoyable.

I also would like to express my gratitude to the laboratory supervisors, Mr. Daniel Roy, Mr. Joseph Hrib, Mr. Luc Demers, Mr. Ronald Parisella and Mr. Lang Vo. Their support and cooperation are greatly appreciated.

Last but not least, I wish to thank the personnel from NovaFrit International, St. Laurence Cement, Lafarge North America Cement Company, Placer Dome mine, Noranda Inc.,

Louvicourt Mine, Newmont Canada Limited, Quebec Cartier mineral company, Gaspe copper mine and various others for providing the materials that made this research possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Introduction

Tailings are the waste materials (byproducts) of the mining industry. The tailings contain all other constituents of the ore but the extracted metal, among them heavy metals and other toxic substances that are either added to the tailings in the milling process or available with the ore before that (ICOLD 2003).

Most of the mill tailings mass-produced worldwide are dumped in large surface impoundments (Simieritsch et al. 2009). The embankments forming these impoundments are earth fill dams. Over the years these earth fill dams have had several serious failures some even fatal. An example of the disasters originating from tailings dams failures is the Merriespruit dam tailings failure that resulted in the deaths of 17 people and extensive damages to a residential township in South Africa in 1994 (Fourie and Papageorgiou 2001 and Fourie et al. 2001**)**. Another disaster took place in the Philippines in 2002 were tailings spilled into Mapanuepe Lake and eventually into the Saint Tomas river. Low lying villages were flooded with mine waste. At least 250 families were evacuated from that area.

Therefore, it is necessary to devise another approach for the storage and disposal of mine tailings with the aim of eliminating the hazards and risks associated with mine tailings dams. Research on mine tailings stems from the necessity to prevent the danger posed by mine tailings to the environment. This danger is characterized by the tendency of mine tailings to release acids and heavy metals once in contact with oxygen and water (Gautam et al. 2000).

Recently, there has been a shift in the way of thinking when dealing with mine tailings from considering these tailings as "waste materials" that need to be disposed of to "managing" these tailings. Researchers, in recent years, made several attempts to utilize these tailings in a sustainable manner beneficial economically. Researchers like Demers and Haile (2003), Zou and Sahito (2004) and Roy at al. (2007) attempted to investigate the possibility of using mine tailings as filler or construction materials with the addition of binders that include Portland cement. Portland cement was used since it produces stronger matrices than other inorganic binder systems, and they do it at lower mix ratios, which results in a smaller volume of waste requiring ultimate disposal (Weitzman et al. 1988). Portland cement, however, is relatively expensive, hence the need to use additives other than cement to reduce cost without compromising effectiveness. This effectiveness is specifically important in cold climates such as that of Canada.

Calsifrit™ is a special cement that was developed recently by NovaFrit International (NovaFrit International 2006). It is a totally amorphous material, a matrix of calcium and sodium fluoro-aluminosilicate. It is a homogenous solid substance possessing high

reactivity potential and shows cementitious properties when finely ground. No research to date has been published on the utilization of this product.

This research will investigate the applicability of using mine tailings - Calsifrit matrices as construction materials in cold climates thus attempting to eliminate the hazard associated with storage of tailings in ponds while economically benefiting from an unused widely available resource. Calsifrit™ will be used in this study as a cement replacement in addition to fly ash and slag.

1.2 Objectives

Development of a new sustainable approach to road construction using mining waste in cold climate by implementation of the following:

1. Formulation of tailings binder matrices and investigating their properties,

2. Investigation of feasibility of using formulated matrices for construction,

3. Determining the resistance of the matrices to weathering (freezing/thawing and wetting/drying),

4. Determining the predictive equation that governs the behavior of the tailings-binder matrices,

5. Development of a Discrete Element Method program as a tool for simulating the strength characteristics of the newly developed tailings-binder matrices,

6. Simulating the vulnerability to cold climate, through the freezing/thawing characteristics of these matrices, using the Discrete Element Method.

Chapter 2

LITERATURE REVIEW

2.1 Introduction

Mine tailings are the waste materials left after the extraction of the ore mineral. They are traditionally stored in tailings ponds located not far away from the mine opening (Simieritsch et al. 2009). This method of storage has created several environmental problems in the past and has prevented the use of a widely available resource.

In order to make use of mine tailings and reduce their bulky method of storage, Robinsky (1978) proposed the thickened tailings disposal (TTD) system for the disposal of mine tailings. He showed that by the process of thickening the tailings to heavy slurry before disposal, it is possible to create a self supporting deposit of tailings and to eliminate the typically used settling pond.

Robinsky (1978) argued that a major setback of conventional flat disposal ponds is that they cannot be reclaimed until the mining operation is terminated many years after being constructed. Such flat areas, since their initial operational life, collect and absorb precipitation, allowing seepage to pass through the deposited tailings, thereby dissolving undesirable metals and chemicals in the form of leachates into natural water bearing mediums. On the other hand, if allowed to dry, these flat areas are exposed to dusting.

Using the TTD system, rapid runoff from the sloping deposit prevents infiltration of precipitation and dusting is reduced to a great extent because the fines in the homogeneously dispersed tailings tend to bond the particles of the entire mix, thus preventing serious dusting. In addition, the TTD approach to disposal will permit the progressive reclamation of the area under consideration (Mahmood and Mulligan 2001).

Robinsky (1978) stated that the inherent problems associated with the operation of conventional tailings disposal schemes are eliminated using this new technique. Some of the problems that are eliminated or reduced are the following:

- Danger of failure of steep-sided tailings dams caused by earthquakes, blasting vibrations, or movement of heavy equipment;

- Erosion and undermining of the dams by seepage from the raised pond in the center of the tailings deposit area;

- Infiltration of undesirable tailings fluid into the underlying natural soil strata from the liquid pond on top of the tailings; in addition to commencement of reclamation only after the mining operation is terminated and the mine closed.

The elimination of the conventional pond on top of the deposit also provides a major environmental advantage; the hydrostatic head responsible for the seepage of process and rainwater through conventionally deposited tailings is eliminated. Another very important

environmental advantage is that the confining dams are eliminated, or extensively reduced in height which will, in turn, reduce or eliminate the problems of aging of tailings dams (Murray et al. 2000 and Vanicek 2002) and liquefaction (Mahmood and Mulligan 2002). Finally, implementation of the system may permit progressive reclamation in some topographical settings.

Robinsky (1978) concluded that since the conventional tailings pond has been eliminated, and without water to flow and wash out the tailings, a dam collapse becomes only a local problem and not an ecological disaster. Another advantage of the TTD system is that the system is capable of inhibiting acid drainage. The environmentally undesirable product is sulphuric acid. It is produced when air and water come into contact with sulphur-containing tailings. Once formed, it is capable of infiltrating the tailings and dissolving other undesirable metallic elements that eventually may seep through and contaminate the surrounding environment (Robinsky 1999).

2.2 Tailings thickening and hardening

The thickened surface tailings disposal concept has been applied at various mining facilities for several decades, including the Kidd Creek Mine in the northern region of Ontario and for managing the disposal of red muds produced by the alumina industry (Robinsky 1999).

A study by Haile et al. (2000) described the use of the thickened (paste) tailings concept with red mud tailings in tropical and temperate climates. Two such processes were described, one in Jamaica and the other in Ireland. They noted significant decrease in storage volume for the same weight of tailings post to dewatering. And they outlined the applicability of the technique in both climates, which was significant taking into consideration that the annual precipitation exceeds, to a large extent, the average annual evaporation in the climate of Ireland. The authors indicated, as well, that bench scale and pilot tests carried out by Alcan International Ltd. on copper and gold tailings showed the applicability of the technology to several types of mine tailings (Haile et al. 2000).

Also, researchers started modifying and improving this technique by including additives with the tailings to further solidify and strengthen them. For example, Peng and Lu (1998) investigated the effect of the sequence of addition, mixing, type and dosage of polymer addition on the overall performance and capacity of a tailings thickening process. The objective of this research study was to evaluate the effect of a dual polymer system on the sedimentation of copper flotation tailings under various operational conditions using both batch and continuous dynamic tests. The authors used a combination of two oppositely charged polymers. The anionic polymer was used to flocculate the large particles and the cationic polymer was then added to coagulate the negatively charged ultrafine particles ($< 45\ \mu m$). Peng and Lu (1998) concluded in their study that the best thickening and clarification performances were made when the dual polymer system was utilized.

As researchers started incorporating additives into the tailings mass for the purpose of strengthening and stabilizing it, the terms “hardened” and “solidified/stabilized” tailings came into existence.

Another example of this approach is the work by Zou and Li (1999) who studied direct solidification and strengthening of dilute tailings slurry. Two types of mine tailings were tested at water/binder ratios of up to 4.5 using specially-developed high-water binder. The main objective of their research was to develop a backfill technology that can operate without using the normally expensive dewatering process. The technology developed should use the total tailings mass and be applicable to all types of mine tailings. A special binder named HiFa Bond was used. HiFa Bond is a special cement with specific gravity ranging between 2.5 and 3.0 and has a typical particle size of 96% below 90 μm. Its main constituent is sulfate aluminate calcium. It was seen that the uniaxial compressive strength increased as the HiFa Bond/tailings ratio increased. The study showed also very rapid strength development; the 7-day strength was able to reach 80 to 94% of the 28-day strength (Zou and Huang 1997). These results proved that the high water tailings slurry could be solidified directly with sufficiently high early strength and that it was possible to use the total tailings slurry without the dewatering process in backfill. It was concluded that in order to achieve the best strengthening results, tailings should be solidified with pozzolanic additives and binders.

2.3 Solidification/stabilization

Stabilization is defined as a process where additives are mixed with waste to minimize the rate of contaminant migration from the waste and to reduce the toxicity of the waste (LaGrega et al. 1994). Thus stabilization may be described as a process by which contaminants are fully or partially bound by the addition of binders, supporting media or modifiers (Conner 1990). Likewise solidification is a process that employs additives by which the physical nature (measured by the engineering properties of strength, compressibility and permeability) of the waste is changed during the process (Pojacek 1979). Thus objectives of stabilization and solidification encompass reductions in waste toxicity and mobility in addition to an improvement in the engineering properties of the stabilized material (Cullinane et al. 1986).

With reference to the nature of the solidification chemicals used, solidification systems are of two basic types, inorganic or organic (Conner 1974). The most important inorganic solidification systems used are the following (Conner 1990):

1) Portland cement,

2) Lime/fly Ash,

3) Kiln dust (lime and cement),

4) Portland cement/fly ash,

5) Portland cement/lime,

6) Portland cement/sodium silicate.

2.3.1 Portland cement

Portland cement is made by heating together limestone and clay (or an alternate source of silica) at about 2700 degrees Fahrenheit, forming a mass called clinker (Conner 1990). A small amount of gypsum is added and the clinker is ground to a fine powder. Portland cement is basically a calcium silicate mixture containing predominantly tricalcium and dicalcium silicates, which are called in cement terminology C_3S and C_2S respectively. Smaller amounts of tricalcium aluminate and calcium aluminferrite with the approximate formulas C_3A and C_4AF, respectively, are also found in Portland cement (Double and Hellawell 1977). Tables 2-1 and 2-2 show the typical chemical and mineralogical compositions of Portland cement, respectively (Conner 1990).

Portland cement was used alone for many years for solidification of radioactive wastes and other wastes containing hazardous constituents, especially for subsequent ocean disposal (Conner 1990). In Japan, it is reported that in 1974 about 47,000 tons of sludges containing mercury were subjected to solidification by concrete before ocean disposal.

With the Portland cement process taking place, water in the waste reacts chemically with the cement to form hydrated silicate and aluminate compounds (Conner 1990).

In general, Portland cement systems produce stronger matrices than other inorganic binder systems, and they generally do it at lower mix ratios, which results in a smaller volume of waste requiring final disposal (Weitzman et al. 1988).

Table 2-1 Typical compositions of Portland cements: chemical composition (%) (After Conner 1990)

Cement Type	**Designation or Characteristics**	**C_3S**	**C_2S**	**C_3A**	**C_4AF**	**Free CaO**	**$CaSO_4$**	**Specific Gravity**	**Specific Surface Area cm^2/g**
I	Normal Portland-general use	45	27	11	8	0.5	3.1	3.17	3220
II	Moderate heat evolution	44	31	5	13	0.4	2.8		
III	High early strength	53	19	11	9	0.7	4		
IV	Low heat evolution	28	49	4	12	0.2	3.2		
V	Sulfate resistance	38	43	4	9	0.5	2.7		
	Rapid hardening	66	11	8	9			3.13	4340
	Super rapid hardening	68	5	9	8			3.14	5950
	Jet cement	52	0	22	5			3.04	5300

Table 2-2 Typical composition of Portland cements: mineral composition (%) (After Conner 1990)

Cement Type	**Designation or Characteristics**	**Ignition Loss**	**Insoluble Material**	**SiO_2**	**Al_2O_3**	**Fe_2O_3**	**CaO**	**MgO**	**SO_3**
I	Normal Portland-general use	0.6	0.1	22	5.1	3.2	65	1.4	1.6
II	Moderate heat evolution							2.5	
III	High early strength							2	
IV	Low heat evolution							1.8	
V	Sulfate resistance							1.9	
	Rapid hardening	0.9	0.2	21	4.9	2.8	66	1.1	2.5
	Super rapid hardening	0.9	0.1	19.7	5.1	2.7	65	2	3
	Jet cement	0.6	0.1	13.8	11.4	1.5	59	0.9	10

Although the concept of tailings thickening is relatively new, soil stabilization methods using cement and lime have been applied for several decades now. Hilt and Davidson (1960) discussed lime fixation in clays showing a decade of advancing technology in lime and Portland cement stabilization of clay soils (Petry and Little 2002). Prusinski (Little et al. 2000) recounts that since 1915 more than 140,000 km of equivalent 7.5 m wide pavement bases had been constructed from different types of cement-stabilized soils. Cement has been found effective in stabilizing a wide variety of soils, which includes granular materials, silts and clays (Petry and Little 2002).

Campbell et al. (1987) used seawater as a leachant in a sequential batch extraction procedure. Samples of cement based waste forms were cured for 3 years before testing. They correlated leaching with the macroscopic and microscopic (scanning electron microscope SEM) observations of the waste forms. It was shown that increased release of cadmium correlated well with the cracking of the solid, which was connected with the formation of crystalline microstructures, especially the expansive ettringite, due to sulfate attack. Total leaching over a 50-day period was about 1 percent of the total cadmium in the solid with no detectable lead.

The stabilization of contaminated soil from a Superfund site was studied by Barich et al. (1987). The Solid Waste Leaching Procedure (Conner 1990) was used to provide engineering estimates of leachates that might be expected from treated wastes subjected to land-filling on site. Several proprietary processes were compared: vitrification, cement-based and two other cement-based processes. These methods produced a one to two order of magnitude reduction in metal (specifically Cr, Pb, Cd, Zn) leachate concentrations over those of the untreated waste.

Shin et al. (1988) investigated a number of factors in cement-based chemical fixation and solidification design in a system that included the addition of sand, sand/cement, water/cement, sludge content and a dosage of “precipitating agent”, which was defined as an organic sulfide compound. They found that the sand/cement ratio had the greatest

effect on Cr^{+6} leaching, while compressive strength and Zn leaching were mostly effected by the water/cement ratio.

Ryan and Jasperse (1989) chronicled deep soil mixing using cement to improve the foundation soils for the Jackson lake dam in Wyoming. The authors found that up to 112 days and beyond there were significant strength gains in the soilcrete. Excellent in-situ curing conditions that included high pressure and a cool moist environment assisted in generating good results for this project.

A few years later, the same authors described two case studies detailing both methods of soil mixing; deep soil mixing and shallow soil mixing (Jasperse and Ryan 1992). The applicability of both technologies was demonstrated to major sites that contain contaminated soils and sludges in terms of the economic and practical advantages that favor this kind of treatment over offsite transport and disposal.

Adaska et al. (1992) illustrated remediation work for an oil refinery sludge basin. These oily sludges had high concentrations of certain types of metals and volatile organic compounds. The remediation method employed for this project consisted of in place stabilization of the sludge and contaminated underlying soil. Part of the remediation procedure included jet grouting (soilcrete), which is a technique that has been used in Japan and Europe for several years. This technique entails the mixture of native soil and cement slurry through the use of high pressure injection. Based on the results of their

work, 21% cement content was recommended to acquire the minimum required 28 day UCS of 242 *kPa*.

Aldridge and Naguib (1992) described two case histories of employing cement in shallow soil mixing for the purpose of solidifying and stabilizing in situ hazardous and non hazardous soils. They demonstrated the applicability of using this procedure to achieve the minimum required UCS strength.

Pamukcu and Hijazi (1992) conducted a laboratory testing program to determine the feasibility of stabilizing soils contaminated with fuel oil and transforming them into usable products. The authors showed that the addition of lime and cement produced marked improvement in the treated samples and the seven day cured samples of treated clay showed unconfined compressive strength almost double the strength of untreated clay samples.

Cement stabilized/solidified soil was also used for building coal retaining berms (Van Riessen and Hansen 1992). The review of five projects constructed between 1974 and 1982 indicated that this technique could be used to solve a number of technical requirements in an economical manner. Structure slopes of up to 55º were constructed with little modification. These structures have shown excellent performance over the years (Van Riessen and Hansen 1992). Cement contents, by weight, ranging from 8% to 10% provided the desired physical properties and required durability for these applications.

Andromalos and Ameel (1994) delineated stabilization work performed on the Geiger (C and M oil) superfund site, which is located near Charleston, South Carolina. Stabilization work on this site was done using a two pass injection process. The first pass involved the injection of a reagent solution to chemically fixate the metals within the contaminated soil mass and the second pass involved the injection of a cement based reagent to physically stabilize and solidify the contaminated soil mass into a cement soil. An extensive testing regime was employed throughout the program. It was concluded that in situ stabilization was effective in stabilizing the site for the contaminants of concern; specifically lead and chromium, and all strength and permeability criteria were met.

Andromalos et al. (2000) described a technique that illustrates the stabilization of loose and soft soils. Cohesive and cohesionless soils were treated under static loading. Three case histories of soil stabilization with soil-cement columns were described. The procedure was proven satisfactory for the three sites in question.

Horpibulsuk et al. (2005) described test results on cement admixed soft clays in in-situ deep mixing. Their aim was to identify the dominant parameters controlling strength development to arrive at a means of combining them to formulate appropriate phenomenological models. Characteristic tests, such as Atterberg limits, specific gravity, pH of the pore fluid and sodium chloride concentrations, were performed to determine the soil characteristics relevant to the investigation. Then unconfined compression tests and consolidated undrained and consolidated drained triaxial compression tests were

conducted on specimens cured for different periods of time. The authors, then, proposed an identity to calculate variations in the cement content required to reach the target specifications without altering the clay-water/cement ratio.

These studies proved the reliability and applicability of using Portland cement with different waste and soil types. Use of cement notably increased the soil and waste strength in these studies. The success in using Portland cement in these various studies with different types of waste materials advocates its use for the stabilization and solidification of mine tailing in this study.

Despite its favorable mechanical and chemical qualities, Portland cement is considered relatively costly. Hence, many research studies that investigated the solidification and stabilization of industrial waste have used industrial additives that are less expensive with favorable properties. These additives are usually added to Portland cement matrices to alleviate some of the cost and provide other complimentary mechanical properties. One such additive is fly ash.

2.3.2 Fly ash

More than 65 million metric tonnes of fly ash is generated in the US each year as a by-product of burning coal at electric power plants, making fly ash one of the most abundant and versatile of the industrial by-products (Collins and Ciesielski 1992). Fly ash is a pozzolan that is a siliceous or siliceous-aluminous material that acquires cementitious

properties when combined with an activator (lime, Portland cement, or kiln dust) in the presence of water. Many of the reactions are analogous to those of Portland cement, but in general these reactions are slower than those of cement and do not produce the exact same products in terms of chemical and physical properties (Conner 1990). Unconfined compressive strengths of fly ash range from 1.38 to 6.895 *MPa* with permeabilities ranging from 10^{-5} to 10^{-8} *cm/s* (Conner 1990).

Fly ash is classified according to its chemical composition into two classes F and C. The chemical requirements stipulated in ASTM C 618-08a (2008) for classifying fly ashes are shown in Table 2-3. Type C fly ash has self hardening properties (since it contains relatively large quantities of calcium hydroxide), hence the addition of water alone will produce a cementing reaction without the addition of lime (Conner 1990).

Table 2-3 Chemical Properties of fly ash

Chemical Composition	**Class F (%)**	**Class C (%)**
Silica (SiO_2)	35	35
Alumina Al_2O_3	20	20
Iron Oxide (Fe_2O_3)	6	6
Total SiO_2 +Al_2O_3+Fe_2O_3	70 min	50 min sulfur
Sulfur Trioxide (SO_3)	5 max	5 max
Calcium Oxide (CaO)	5	15
Magnesium Oxide (MgO)	5 max	5 max
Moisture Content	3 max	3 max
Loss on ignition	12 max	6 max
Available alkali as Na_2O_3	1.5 max	1.5 max

Fly ash has been recycled for many years as an engineering material because of its pozzolanic properties (Ghosh and Subbarao 1998). For example as early as 1914, Engineering News Record published research results verifying that Portland cement concrete can benefit from the addition of fly ash (FHWA 1995).

Although a majority of fly ash produced in the United States (for example) is currently landfilled, 22% of fly ash is used in a variety of beneficial applications (FHWA 1995). The increased costs and potential environmental hazards of landfilling have caused regulatory agencies to encourage more beneficial use of fly ash (Ghodrati et al. 1995). Consequently, new and innovative uses of fly ash are continuously being researched and developed (Torrey 1978).

Fly ash is often used in Portland cement concrete, stabilized road bases, structural fills and flowable fills. Fly ash is also utilized in soil stabilization, waste stabilization, asphalt mixes, cold recycled bituminous pavement (Cross and Fager 1995) and in grouts for concrete pavement subsealing. Fly ash has been utilized in soil stabilization to improve the mechanical properties of soils for more than 20 years (Ferguson 1993).

Leaching characteristics of fly ash mixed waste systems were studies by other researchers. Cote (1986) studied the leachate pH history of several chemical fixation and solidification systems over a period of approximately two years, and found that the fly ash containing processes had lower initial pH and the pH decreased faster as time proceeded. This is explained by the pozzolanic reactions that consume lime produced

from cement hydration. Concurrently the cement/fly ash process produced the lowest leaching of metals with time in the dynamic leaching test. Cote (1986) attributed this result to the moderation of pore water pH by the fly ash as well as to other effects such as adsorption and decreased porosity.

Northeim (1989) declared that the acid neutralization capacity of fly ash is very high and slow to be depleted. Later on, Cheng and Bishop (1990) explained that the penetration distance of the leaching front is highly dependent on acidity of the leaching solution while water/binder ratio had little or no effect on the penetration distance or rate of penetration. They determined the relationship among hydrogen ion in the leachant, the penetration depth of the leaching front and the alkalinity leached from the cement-based and lime-fly ash (type F) based matrices. Acids attack pozzolanic-based matrices through permeation of the pore structure and dissolution of ions.

Vipulanandan and Shenoy (1992) studied the properties of cement grouts and grouted sands used with additives. The effects of additives such as condensed silica fume, fly ash, clay calcium chloride sodium silicate on the cement grout (water to cement ratio of 1) were researched. The authors found that use of fly ash as a substitute for cement affected the initial and final setting times for the grout and caused substantial reduction in the bleeding.

Scanlon et al. (1995) used the Total Concentration Leaching Procedure test to characterize the hazardous potential of fly ash and found that the heavy metals were below the detection limits.

Robinson (1995) subjected lime/fly ash monoliths to the TCLP test for the hazardous metals chromium, cadmium and lead. He showed that all monoliths were below the detection limit for both lead and cadmium but one third of them were above the detection limit for chromium.

Valles (1996) modified the TCLP test from a batch to a semi-batch operation to analyze the leaching behavior of simulated evaporator bottoms/fly ash mixture. He observed the presence of a declining concentration trend after undergoing a maximum at approximately 24 hours. Chromium metal was the only detectable species in the modified TCLP test effluent.

Zou and Li (1999) found during the study of two types of mine tailings that using fly-ash to replace a portion of a new binder, called HiFa Bond, lead to mixture reinforcement and strength increase. It was shown that strength also increased when adding fly ash to these two tailings. An important aspect of using fly ash was to reduce the required quantity of HiFa Bond, thereby reducing the total cost of the binder.

Kumar et al. (2001) investigated the geotechnical properties of fine coal refuse stabilized with fly ash and lime. Geotechnical properties investigated in this study were:

compaction characteristics, triaxial shear strength, California Bearing Ratio (CBR) and permeability. Coal mine refuse used was taken from a selected mine in southern Illinois. They found that the addition of 10-30% of fly ash content to the coal mine refuse gave good compaction characteristics. This result was in general agreement with results of other investigations (Moulton et al. 1974 and Usmen 1986). It was also shown by performing grain size analysis on the coal refuse samples after compaction that no measurable degradation took place for the fine coal refuse particles. Also it was found that the addition of lime or mixtures of lime-fly ash had a beneficial effect on the shear strength of the fine coal refuse. It was ascertained that the shear strength parameters determined for the fine coal refuse were within the range reported in available literature.

Raouf and Nowier (2004) developed a simple and inexpensive method for the immobilization of the hazardous metals Cd, Pb and Fe. They used fossil fuel fly ash as the primary ingredient for the stabilization/solidification of Cd and Pb in compressed formulations that are characterized by a rigid structure and have relatively high durability and low leachability. The molded formulations were prepared by first mixing the hazardous waste solutions with the fossil fuel fly ash material at a ratio of 17%. Then the resulting pastes were pressed at the appropriate applied pressure using a manual hydraulic press. The compressed formulations showed good water resistance, compression strength and radiation resistance. The authors showed that loading fly ash formulations with Cd and Pb at higher initial contamination concentrations (up to 20,000 *mg/kg*) did not affect their leachability (Raouf and Nowier 2004).

These beneficial leaching characteristics of fly ash when added to waste systems have encouraged its use in the current study. Consequently, it is desired to investigate its leaching and mechanical effects when added to Portland cement in the binder-mine tailings matrices.

2.3.3 Slag

Another example of an industrial additive that is used in conjunction with Portland cement in waste systems is slag. Blast furnace slag is a nonmetallic coproduct produced during the production of iron. Granulated blast furnace slag is defined as the glassy granular material formed when molten blast furnace slag is rapidly cooled as by immersion in water, ASTM C 125 (2007) and ACI (1994). Its main constituents are silicates, aluminosilicates, and calcium-alumina-silicates (FHWA 2007).

Due to its high content of silica and alumina in an amorphous state, GGBFS shows pozzolanic behavior akin to that of natural pozzolans, fly ash and silica fume (Atis and Bilim 2007). Slag is added to Portland cement- waste matrices to reduce the total cost and to contribute other beneficial properties.

Blast-furnace slags have been widely used as ingredients in cement or concrete with potential hydraulicity from the perspective of effective use industrial byproducts (Bijen 1996). Erdogan (2003) reported that the use of granulated blast furnace slag as a mineral

admixture in Normal Portland cement (NPC) concrete mixes had started in South Africa in 1953 (Atis and Bilim 2007).

Oner and Akyuz (2007) conducted a laboratory investigation on the optimum ratio of ground granulated blast furnace slag (GGBFS) on the compressive strength of concrete. They showed that as the GGBFS content increased, the water to binder ratio decreased for the same workability indicating a positive effect of GGBS on workability. They also showed that the compressive strength of GGBFS concrete increased with the increase of the GGBFS content up to an optimum level after which the compressive strength began to decrease. The optimum level of GGBS content for maximizing strength found was at about 55-59% of the total binder content in the mix (Oner and Akyuz 2007).

Ann et al. (2008) used pulverized fuel ash (PFA) and ground granulated blast furnace slag (GGBFS) to compensate for the loss of strength and durability of concrete that contains recycled aggregate. Recycled aggregate from crushed concrete was used and PFA and GGBFS were partially replaced for cement in binder to enhance the concrete durability properties. It was found that 30% PFA and 65% GGBFS concretes increased the compressive strength to the level found in control specimens cast with natural granite gravel. Replacement with PFA and GGBFS was also effective in increasing the resistance to chloride ion penetrability into the concrete body. It was also found that after the onset of steel corrosion, the corrosion rate was significantly reduced by PFA and GGBFS, due to the restriction of cathodic reaction that normally needs a sufficient supply of oxygen and water (Ann et al. 2008). The positive effect of slag on workability, durability and

compressive strength of concrete has prompted its use as another additive in the present study.

2.3.4 Calsifrit™

It is shown from the previous research studies above that no researcher had attempted to investigate the effect of adding Calsifrit™ on the waste-binder system. Calsifrit™ is a totally amorphous material developed recently by novaFrit International. It is defined as a matrix of calcium and sodium fluoro-aluminosilicate. It is a homogenous solid substance possessing high reactivity potential and shows cementitious properties when ground to a fine powder (NovaFrit International 2006). Calsifrit™ is currently utilized as a concrete durability improver. A report published by the University of Sherbrook in Quebec stated that using 25% Calsifrit™ to substitute for Portland cement decreased the permeability of concrete to chloride ion by 50%. Calsifrit, it is explained, does this by promoting a discontinuous pore structure (NovaFrit International 2006). No research to date has been published on the utilization of this product in other industries. One of the objectives of this study is to investigate the effect of using Calsifrit™ on the overall mechanical and leaching properties of the tailings-binder matrices.

Calsifrit™ was chosen since it contains chemical components that aid and enhance binder materials setting. A component such as sodium silicate fills micropores densifying the mixture (Girard 2005), increasing pH and binds colloidal molecules together (Kazemian et al. 2010), thus, reducing the porosity of the matrix. This is important since the pH for

cement setting is highly alkaline. CaO adsorbs moisture and coagulates colloidal particles. Aluminum neutralizes charged particles. Calcium fluoride reacts with sulfate (or sulfuric acid) to form calcium sulfate, which is a coagulant. Iron helps in controlling the pH of the mixing environment (Boyer 2002). Additionally, Calsifrit™ contains carbon, which adsorbs organic matter, if found in the material (Amuda and Ibrahim 2006).

2.4 Mine tailings as construction materials

Success in the process and results of mine tailings hardening led researchers to further think the possibility of using these hardened tailings in the construction industry thereby utilizing a vastly available untapped resource and hindering, albeit to a certain extent, the harmful effects of unprocessed tailings on the environment. This success coupled with historical evidence that told of using tailings in the mine backfilling business further strengthened this new approach.

The earliest reported use of mine tailings as a construction material was its use as a mine backfill material. Use of mine tailings for backfilling started in the early years of the potash industry by use of dry and wet residues (Wasayo 2003). Different technologies were developed in the last 80 years. One of these, the slurry backfill technology is considered a very efficient backfill method, because of its combination of tailings and brine disposal. Since 1908, the slurry backfill technology has been successfully practiced

in the German potash industry, especially in flat-lying deposits of the South Harz potash district (Wasayo 2003).

Success in these early attempts led researchers to attempt a thorough investigation approach of mine backfilling using more advanced technologies and methods. Garand et al. (2000) discussed the effects of flocculent deposition of tailings sludge in the Bouchard-Hebert mine in northern Quebec. ASTM standard tests were performed on the tailings to characterize their particle size, specific gravity and Atterberg limits. Then a flocculent PERCOL E-10 was added to the tailings sludge before it was mechanically thickened in the paste backfill plant. These flocs showed grain size distribution larger than or similar to coarse silt. The authors stated that the beaches formed with these flocs could be used as competent foundation material for upstream raises (Garand et al. 2000). This study presented a modification on Robinsky's thickening principle (1978) by adding a flocculent to the tailings to enhance thickening and increase stability.

Another study by Benzaazoua et al. (2002) investigated in some detail the influence that several chemical factors have on the performance of mine sulphidic paste backfill. They used four samples from three different mines located in Canada, which were mixed with four types of binders and six different types of mixing water chemical characteristics. This was done as an attempt to simulate field conditions. It was found that the paste backfill matrix texture is directly related to its strength development. Also it was found that the mixing water is an important parameter that affects the quality of the paste backfill mass and that, in contrast to the Portland cement based binder, slag-based binder

hydration seems to be inhibited by the presence of soluble sulphates (Benzaazoua et al. 2002). The results of this study clearly demonstrated the inefficiency of choosing paste backfill mixtures without testing first the tailings and mixing water characteristics.

Theriault et al. (2003) described the surface disposal of paste tailings at the Bulyanhulu Gold mine in Tanzania. This mine is managed by a subsidiary of Barrick Gold Corporation. Goals of this process were, as stated by the authors, to conserve water, manage runoff, reduce risk and minimize containment dyke construction. The tailings slurry was dewatered before transportation to the paste plant where process water was added in the paste conditioner to produce a paste of the desired consistency. The authors affirmed that the cycling of the tailings deposition in thin layers has been successful in generating a stable paste stack. It was concluded that paste stack can be engineered to meet the required geotechnical and environmental objectives.

These earlier studies ascertained the applicability of using mine tailings as backfill materials and encouraged investigation of the use of mine tailings for other construction disciplines. One such discipline is exemplified by the work of Demers and Haile (2003) who described the stabilization of zinc tailings (called Jarosite). Jarosite was deposited in ponds sustained by clay dykes until 1998. Since then, Jarosite was thickened first using vacuum filters and then lime, cement and water were added to the thickened Jarosite to make a product that was termed Jarofix. Laboratory testing and stack modeling were performed to determine the feasibility of using the cured Jarofix to build containment dykes and service roads. Field tests were performed, as well, after which the authors

concluded that Jarofix is chemically inert. It was observed that cured Jarofix was an excellent fill material for raising the containment dykes. The research was carried out inside the tailings storage facility and limited testing was done in this case.

Another different example lies in the work of Zou and Sahito (2004) who studied the applicability of using a new type of binder termed HiFa Bond for shotcreting in underground support. The authors used mine tailings as aggregate in shotcrete. These tailings were mixed with sand, polymer and steel fibers. Test results showed that mine tailings have potential for shotcreting for underground support.

Meanwhile, Celik et al. (2006) investigated the effects of the mineralogical composition and chemical properties of gold tailings on the compressive strength of Ordinary Portland Cement. The authors used silica fume and 2 types of fly ash, with the tailings as Portland cement additives. It was shown that the compressive strength values of mortars prepared by these additives were acceptable as they were within the range of European standards. The authors presented the conclusion that the gold tailings could be used as an additive in Portland cement production. No weathering resistance or leaching testing had been done in this study.

Another study by Roy at al. (2007) investigated the use of gold tailings in making bricks. The authors mixed the tailings with 4 types of soil and Portland cement. The quality of bricks was characterized in terms of compressive strength, linear shrinkage and water absorption tests. The authors did a cost analysis as well. They showed that the quality of

these bricks was improved when mill tailings were mixed with the soils. The study showed that soil-tailings bricks passed the required criteria defined by means of compressive strength, linear shrinkage and water absorption. The cost of producing these tailings-soil bricks was shown to be less than that of traditional clay bricks. However, there were no leaching or weathering environmental tests conducted.

Swami et al. (2007) investigated the use of Kimberlite diamond mine tailings in road layers. For this purpose, the authors carried out a field and laboratory testing program. Physical tests including Proctor, CBR and unconfined compression were executed before and after stabilization with cement and bitumen. Chemical properties of the tailings were also determined before the laboratory and field experiments. The field experiments consisted of construction of a test road section using these tailings, which were evaluated for use in sub-base, base and wearing courses. It was shown that Kimberlite tailings could be used in cement bound sub-bases.

The study did present a new approach in the area of road construction. However, the testing regime implemented was not comprehensive and there were no weathering resistance or leaching tests conducted. These tests are considered essential in cold environments such as that of Canada. Also the experimental program lacked correlation with numerical analysis techniques.

Other researchers like Fall et al. (2005), Fall et al. (2008), Benzaazoua et al. (2008), Ercikdi et al. (2009), Fall and Pokharel (2010), Ercikdi et al. (2010) and Helinski et al.

(2010) investigated the use of mine tailings in the most traditional manner by using them as cemented paste backfill. Meanwhile, the research group of Yang et al. (2009) investigated the use of mine tailings in the glass-ceramic/ceramic tile industry.

2.4.1 Concluding remarks for mine tailings in construction

It appears from the above that although several research studies discussed the applicability of using mine tailings in several types of construction practices, none had comprehensively addressed the issue of using mine tailings as construction materials for temporary roads in cold climates. Simulation of the strength and weathering properties of these tailings using a powerful numerical technique was lacking in these studies, as well.

2.5 Numerical modeling

2.5.1 Introduction

Although granular media are a multiphase particulate system they have generally been modeled as a continuum. Problems occur with this assumption due to the known granular nature of soils and tailings and their multi-force multiphase nature. The problem is especially evident with non linear stress-strain behavior or when significant local cracks and yielding appear in compression tests.

Numerical methods are desired over experimental setups in modeling particulate media since the latter are more expensive to build and operate. Numerical modeling has advantages over physical testing: parametric studies can be performed with ease, non-destructive and obtrusive sample testing can be performed, boundary conditions can be controlled explicitly, sample reproducibility can be guaranteed, and tests can be stopped and restarted to suit a particular need (Khwaja 1996).

2.5.2 Continuum based numerical methods

Traditionally, continuum based numerical methods have been used qualitatively and quantitatively to analyze particulate and granular media. The most notable example in soil mechanics is the Finite Element Method and its various hybrids.

Although the finite element method was used to study discontinuous problems, it has certain disadvantages. For example this method is continuous in nature, which means that it cannot efficiently mimic the discontinuity property. Also, as the accuracy of the model is as good as the assumptions used, reproducing complex behavior with continuous methods requires complex constitutive models, containing sometimes dozens of parameters and/or internal variables in order to capture discontinuous behavior. This can improve results but on the other hand they are computationally intense and time consuming (Matar 2005).

Due to its particulate nature, soil can experience significant local deformations and bifurcations. Continuum based methods cannot compensate for such anomalies without incremental analysis involving non-linear elasto-plasticity based constitutive relations. Even so this approach is lacking in adequately modeling the most rudimentary behavior of soil, including non-linear deformations and local yielding, (Khwaja 1996).

2.5.3 The Discrete Element Method

It was not until 1979, that Cundall and Strack developed the Discrete Element Method, Cundall and Strack (1979a). The method models particulate media as a discrete collection of particles. This random collection of particles interacts through contact forces. The method calculates the displaced positions and rotations of these particles at discrete time steps. The DEM-simulation is started by first generating a model, which results in the random orientation of particles with assigned initial velocities. The forces and moments acting on each particle are computed form the initial data and the relevant physical laws and contact models. Generally, the simulation consists of three parts: the initialization, the explicit time stepping and post processing.

In DEM the microstructure of the system is modeled rather than using constitutive laws or complicated elements. Using DEM we can capture the changes in microstructure, change in shape and deformations, dynamics and forces within the system in real time and in detail. Compared to conventional continuum methods, the DEM uses fundamental and much fewer parameters when modeling discontinuous behavior (Matar 2005).

The method is capable of analyzing multiple interacting continuous, discontinuous or deformable interacting bodies undergoing large displacements and rotations. In the scheme developed by Cundall and Strack (1979a) the particles are not allowed to deform, instead they overlap and the method monitors each contact between particles and computes the new positions and orientations accordingly. Also since the algorithm can model dynamic stress propagation from particle to particle, it can be used to analyze dynamic as well as static soil behavior, (Ting et al. 1989). In addition, data evaluation at a macroscopic level can be achieved (Khwaja 1996).

Matar (2005) states that DEM is more suited for discontinuous materials for the following reasons:

1) Model is localized by nature, so there is currently no better tool to model discontinuous material than DEM,

2) Modeling discontinuous material is straightforward,

3) Any type of inter-particle forces can be incorporated,

4) Any particle shape can be considered,

5) Results obtained are more related to discontinuous materials than other methods,

6) It can be coupled with other methods to model continuous-discontinuous properties.

Cundall (2002) made the case that continuum methods, such as finite element, for rock and soil might be completely replaced by particle models, such as DEM, in 10-20 years.

2.5.3.1 General Discrete Element Method theory

The calculations performed in the discrete element method alternate between the application of Newton's second law to the discs and a force-displacement method at the contacts. Newton's second law gives the motion of a particle resulting from the forces acting on it. The force-displacement law is used to find contact forces from displacements (Cundall and Strack 1979a).

The DEM simulation consists of an assembly of particles, which has a certain shape contained in a bounded area. Those particles interact with each other under certain conditions based on a specific relational equation. The bounded area can be two or three dimensional. The DEM method involves the cyclic calculation of the inter-particle force between particles in contact.

The simulation starts by assuming some initial configuration of particle positions and then contact detection module finds which set of particles are interacting. Based on type

of interaction, forces are applied to these interacting particles. The simulation then proceeds by stepping in time. Velocities, orientations and positions are calculated based on Newton's second law of motion (Matar 2005).

Velocities and incremental displacements are calculated by integrating the equation of motion. The equation of motion for the center of mass of body (i) is given by:

$$m_i \frac{dvi}{dt} = \sum_{j=1}^{n} Fxij + mi\ g \quad \text{----------(2-1)}$$

Where:

m_i = Mass of body i

v_i = Velocity of body i

n= Total number of particles

F_{xij} = Inter-particle force between body i and j

g= Acceleration of gravity

The angular velocity is given by:

$$I_i \frac{d\omega i}{dt} = \sum_{j=1}^{n} L_{xij} \quad \text{----------(2-2)}$$

Where:

I_i = Inertia of body i

ω_i = Angular velocity of body i

L_{xij} = Angular momentum for body i and j

2.5.3.2 DEM applications in geotechnical engineering

After the pioneering work of Cundall and Strack (1979a) in developing the first discrete element scheme for analyzing particulate and granular media, researchers started evaluating and improving the technique.

Dobry and Ng (1992) performed a literature survey of publications which have used the DEM of compliant particles or blocks for simulations of granular media during the years from 1982-1992. Tables 2-4a-d show the 42 publications listed by the authors.

Table 2-4a Literature survey of DEM up to 1992 (After Dobry and Ng 1992)

Criterion		Bathurst and Rothenburg (1989)	Rothenburg and Bathurst (1992)	Cundall (1971)	Cundall (1974)	Cundall (1976)	Cundall and Strack (1979a)	Cundall and Strack (1979b)	Cundall and Strack (1979c)	Cundall and Strack (1983)	Cundall (1988)	Cundall and Strack (1989)
Boundaries	Periodic Space											
	Boundaries											
Dimensions	3D											
	2D											
Particle shape	Polygon											
	Rounded											
Particle size distribution	One size											
	Two sizes											
	Three sizes											
	Four to eight sizes											
Model size	≤ 100 grains											
	101 to 1000 grains											
	1001 to 8005 grains											
Contact law	Linear											
	Linear Pressure											
	Simplified Mindlin											
	Complete Mindlin											
Relaxation scheme	Special scheme											
	Newton's 2nd law											
Particle rotation	Rotation not allowed											
	Rotation allowed											
Type of equilibrium	Dynamic equilibrium											
	Static equilibrium											
Type of loading	Cyclic loading											
	Monotonic loading											
Type of problem	Soil mechanics											
	Rock mechanics											
	Grain flow											
	Engineering problem											

Table 2-4b Literature survey of DEM up to 1992 (After Dobry and Ng 1992)

Criterion		Huang and Lee(1989)	Ishibashi et al. (1989)	Iwashita et al. (1988)	Ishibashi and Hakuno (1989)	Kishino (1988)	Kiyama and Fujimura (1983)	Kuhn and Mitchell (1989)	Ng and Dobry (1988)	Ng (1989)	Oner (1984)	Petrakis et al. (1989)
Boundaries	Periodic Space		■						■	■		■
	Boundaries	■		■	■	■	■				■	
Dimensions	3D		■									
	2D	■		■	■	■	■		■	■	■	■
Particle	Polygon											
	Rounded	■	■	■	■	■	■	■	■	■	■	■
Particle size distribution	One size			■			■		■	■		■
	Two sizes		■						■	■		■
	Three sizes			■		■						
	Four to eight sizes			■	■						■	
Model size	≤ 100 grains								■	■		
	101 to 1000 grains					■	■	■	■	■	■	■
	1001 to 8005 grains			■								
Contact law	Linear			■	■	■	■				■	
	Linear Pressure Dependent		■									
	Simplified Mindlin											
	Complete Mindlin								■	■		■
Relaxation scheme	Special scheme					■						
	Newton's 2nd law	■	■	■	■		■		■	■	■	■
Particle rotation	Rotation not allowed								■	■		■
	Rotation allowed	■	■	■	■	■	■	■			■	
Type of equilibrium	Dynamic equilibrium				■		■					
	Static equilibrium	■	■	■		■		■	■	■		■
Type of loading	Cyclic loading			■		■			■	■	■	■
	Monotonic loading	■	■	■	■	■	■	■	■	■		■
Type of problem	Soil mechanics		■			■		■	■	■	■	■
	Rock mechanics											
	Grain flow											
	Engineering problem	■		■	■		■					

Table 2-4c Literature survey of DEM up to 1992 (After Dobry and Ng 1992)

Criterion		Petrakis and Dobry (1989)	Strack and Cundall (1978)	Strack and Cundall (1984)	Tarumi and Hakuno (1989)	Thornton and Randall (1988)	Ting et al. (1986)	Ting et al. (1987)	Ting and Corkum (1988a)	Ting and Corkum (1988b)	Ting and Corkum (1988c)	Ting et al. (1989)
Boundaries	Periodic Space	■		■								
	Boundaries		■			■	■	■	■	■	■	■
Dimensions	3D											
	2D	■	■			■	■	■	■	■	■	■
Particle shape	Polygon											
	Rounded	■	■	■		■	■	■	■	■	■	■
Particle size distribution	One size	■					■					
	Two sizes	■		■			■					
	Three sizes		■				■	■	■	■	■	■
	Four to eight sizes		■			■						
Model size	≤ 100 grains											
	101 to 1000 grains	■	■	■		■	■	■	■		■	■
	1001 to 8005 grains						■	■		■	■	■
Contact law	Linear		■	■			■	■	■	■	■	■
	Linear Pressure											
	Simplified Mindlin					■						
	Complete Mindlin	■										
Relaxation scheme	Special scheme											
	Newton's 2nd law	■	■	■		■	■	■	■	■	■	■
Particle rotation	Rotation not allowed	■						■	■	■	■	■
	Rotation allowed		■	■		■	■					
Type of equilibrium	Dynamic equilibrium					■					■	
	Static equilibrium	■	■	■			■	■	■	■	■	■
Type of loading	Cyclic loading	■		■	■							
	Monotonic loading	■	■			■	■	■	■	■	■	■
Type of problem	Soil mechanics	■	■	■	■	■	■	■	■	■		■
	Rock mechanics											
	Grain flow											
	Engineering problem						■	■		■	■	■

Table 2-4d Literature survey of DEM up to 1992 (After Dobry and Ng 1992)

Criterion		Uchida and Hakuno (1989)	Uemura and Hakuno (1987)	Uemura and Hakuno (1989)	Walton (1982)	Walton (1983)	Walton and Braun (1986)	Wu and Liu (1989)	Yamamoto and Hakuno (1989)	Zhang and Cundall (1986)
Boundaries	Periodic Space						■			■
	Boundaries	■	■	■	■	■			■	
Dimensions	3D									■
	2D	■		■	■	■	■		■	
Particle shape	Polygon				■	■				
	Rounded	■		■			■			■
Particle size distribution	One size		■				■			
	Two sizes									■
	Three sizes									
	Four to eight sizes		■		■	■				
Model size	≤ 100 grains						■			
	101 to 1000 grains									■
	1001 to 8005 grains		■	■	■	■			■	
Contact law	Linear	■	■	■	■	■	■			
	Linear Pressure									
	Simplified Mindlin									■
	Complete Mindlin									
Relaxation scheme	Special scheme									
	Newton's 2nd law	■	■	■	■	■	■	■		■
Particle rotation	Rotation not allowed									
	Rotation allowed	■	■	■	■	■	■	■		■
Type of equilibrium	Dynamic equilibrium	■			■	■	■			
	Static equilibrium		■	■				■	■	■
Type of loading	Cyclic loading		■							■
	Monotonic loading		■	■				■		■
Type of problem	Soil mechanics							■		■
	Rock mechanics				■					
	Grain flow	■				■	■			
	Engineering problem		■	■					■	

A major conference conducted by the American Society for Civil Engineers (ASCE) in 2002 in New Mexico, USA, presented several major trends in the DEM modeling and practice. Participants discussed and analyzed several topics in this field including the following:

1) Theory and algorithms: (Favier and Kremmer 2002, Feng and Owen 2002, (Hopkins 2002, Kuhn 2002, Ng 2002),

2) Model generation: (Boutt and McPherson 2002), (Feng et al. 2002), (Johnson and Williams 2002),

3) Simulation environments: (Hashash and Ghaboussi 2002), (Komodromos and Williams 2002), (Masala et al., 2002),

4) Solid continuum and discrete element methods (Bangash and Munjiza 2002), (Lin 2002), (Salami and Amini 2002), (Tavarez et al. 2002),

5) Fluid discrete element methods: (Cook et al. 2002), (Kawaguchi et al. 2002), (Mori et al. 2002), (Thornton and Kafui 2002),

6) Experimental validation: (McBride et al. 2002), (Murakami et al. 2002), (O'Sullivan et al. 2002),

7) Cohesive materials: (Anandarajah and Yao 2002), (Lechman et al. 2002), (Newson and Duliere 2002),

8) Granular mechanics: (Shen 2002), (Zhang 2002),

9) Powders and soils: (Cheng et al. 2002), (Nezami and Hashash 2002), (Zeghal et al. 2002),

10) Rock: (MacLaughlin and Clapp 2002), (Olson et al. 2002), (Potyondy 2002).

Table 2-5 shows some of the research studies conducted between the years 1992 and 2009.

Later on Potyondy and Cundall (2004) presented a numerical model for rock using the DEM. They showed that the model reproduced many features of rock behavior including elasticity, fracturing, acoustic emission, damage accumulation producing material anisotropy, hysteresis, dilation, post peak softening and strength increase with confinement.

Matar et al. (2007) modeled the evolution of particle subdivision in Montmorillonite clay using 3 dimensional DEM. They wrote a program using ANSI C++ that studied the swelling and swelling pressure response with various amounts of particle breakdown in a unidirectional swelling cell.

Table 2-5 Some DEM research studies conducted between 1992 and 2009

Criterion		Tavarez et al. (2002)	Zeghal et al. (2002)	O'Sullivan et al. (2002)	Sarracino et al. (2002)	McBride et al. (2002)	Tannant and Wang (2002)	Olson et al. (2002)	Matar et al. (2007)	Chen et al. (2009)	Potyondy and Cundall (2004)
Boundaries	Periodic Space										
	Boundaries	■	■	■	■		■	■	■		■
Dimensions	3D		■	■	■	■		■	■	■	■
	2D	■					■				
Particle shape	Polygon										
	Rounded	■	■	■	■	■	■	■		■	■
	Cuboid								■		
Particle size distribution	One size	■			■	■					
	Two sizes										
	Three sizes			■					■		
	Four to eight sizes		■				■	■		■	■
Model size	≤ 100 grains									■	
	101 to 1000 grains										
	More than 1000 grains		■				■		■		■
Contact law	Linear				■						
	Linear Pressure										
	Simplified Mindlin										
	Complete Mindlin										
Relaxation scheme	Special scheme										
	Newton's 2nd law	■	■		■		■				■
Particle rotation	Rotation not allowed										
	Rotation allowed	■			■	■					■
Type of equilibrium	Dynamic equilibrium		■		■			■		■	
	Static equilibrium	■									
Type of loading	Cyclic loading							■			■
	Monotonic loading	■	■	■			■				
Type of problem	Soil mechanics		■	■				■	■		
	Rock mechanics										■
	Grain flow				■	■				■	
	Engineering problem	■					■				

Chen et al. (2009) performed a 3 dimensional modeling of sinkhole repair using the DEM. They showed that the DEM was a reasonable method to investigate sinkhole repair procedure.

More recently, researchers used the DEM to model geosynthetic reinforced soils (Park and Lee, 2010), geotextile reinforced soils (Bhandari and Han, 2010) and geogrid reinforced soils (Han and Bhandari, 2009).

2.5.3.3 DEM applications in mining engineering

Most DEM applications dealt with either rock or soil materials. However a few researchers investigated the use of mine tailings as granular media.

McBride et al. (2002) performed an experimental and numerical study using a laboratory ball mill. In the experimental part they recorded the three dimensional trajectory of particles using an automated tracking technique and bi-planar X-ray filming. This was followed by 2 dimensional DEM modeling to determine the parameters needed for the 3 dimensional DEM simulation to avoid the extra computational overhead. Then 3 dimensional DEM simulation was performed. The methods presented provided rigorous benchmarking of DEM's predictive capabilities based on 3 dimensional trajectory data and the experimental measurement of material properties.

Sarracino et al. (2002) performed another experimental and numerical study of mills in their granular form. In the experimental work, they measured incident and final velocities, final angular velocity and the coefficient of restitution of tumbling mills. The experiments were then modeled numerically in order to find contact and damping models that best reproduced the experimental results.

These two research studies represent a noticeable incursion into the area of DEM modeling of tailings in their granular form. However, beyond the granular form simulation of mills, no investigation of hardened tailings simulation and analysis had been performed.

Another study by Tannant and Wang (2002) modeled using 2 dimensional Particle Flow Code (PFC), (Itasca, 1999) spray-on, rapid setting polymeric liner materials for underground rock support in Canadian mines. Two dimensional tensile and block punch tests were modeled. The authors showed that the PFC models of liners were capable of exhibiting many features in the field including: progressive de-bonding, liner bending and elongation, isolated zones of high tensile load, progressive liner rupture in tension and eventual liner failure. The study did not investigate the mechanical properties of the actual mining residues or the mine rock.

2.6 General conclusion

This shows that application of the DEM method to various rock and soil materials was found to be most promising and adequate. The limited amount of work done on simulating mine tailings also appears to be going on the right track with initial good results.

The DEM method seems to be the most adequate for particulate systems. It should be noted that the DEM has never been tested for strength and weathering characteristics. It also has never been tested on solidified tailings.

Thus from the above it appears that although several studies discussed the applicability of using mine tailings in several types of construction practices, none had comprehensively addressed the issue of using mine tailings as construction materials for temporary roads in cold climates.

It is also concluded that no research study had addressed the issue of DEM simulation of hardened mine tailings. The current study aims to address this by doing the following:

1) Determining the unconfined compressive strength and weathering characteristics of several types of Portland cement-fly ash-slag stabilized/solidified Canadian mine tailings matrices,

2) Investigating the suitability and effect of using Calsifrit on the properties of the mine tailings-binder matrices,

3) Determining the freezing/thawing and wetting/drying resistance of these tailings matrices,

4) Simulating the strength and weathering characteristics of the tailings matrices using the Discrete Element Method.

Chapter 3

METHODOLOGICAL APPROACH

3.1 Introduction

In order to assess the applicability of using mine tailings as construction materials, an experimental and computational program was devised to find their index properties and to assess their engineering properties with respect to the proposed function. The methodological approach is shown in Figure 3-1.

Investigations consist of two phases: experimental and computational. The Experimental Phase 1 is divided to 3 stages: characteristics of materials, formulation of tailing matrices and assessment of matrices usefulness for read construction material. Phase 2 consists of 3 stages: development of an adequate program predicting matrices usefulness for road material, verification of the program for strength tests and verification for weathering tests.

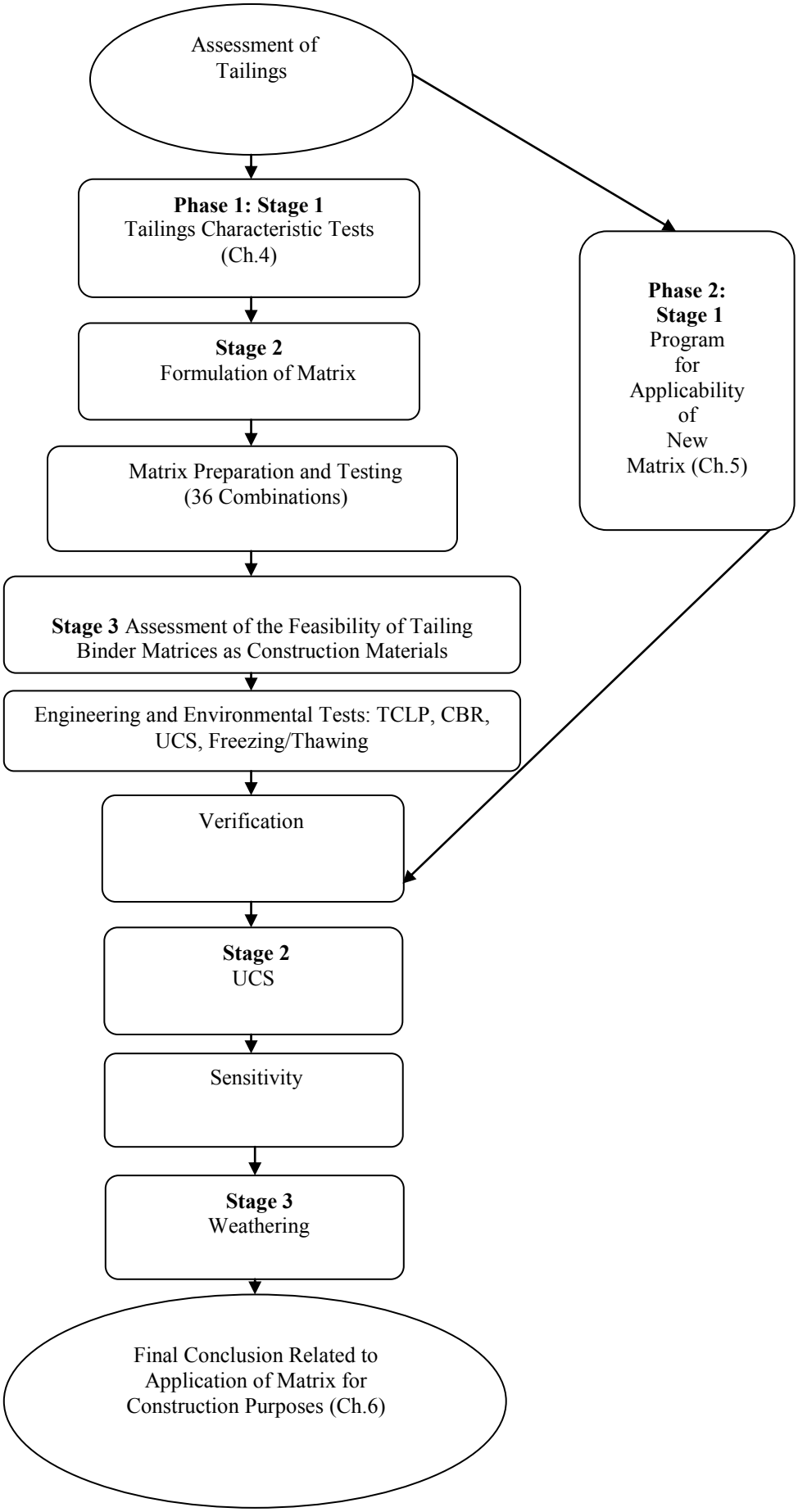


Figure 3-1 Experimental and computational methodology for tailings matrices assessment

3.2 Phase 1: Experimental investigations

3.2.1 Stage 1: Characteristics of materials

The following characteristic and engineering tests describe the different properties of the tailings and tailings-binder mixtures that need to be investigated for the assessment of the use of these tailings-binder mixtures as construction materials.

3.2.1.1 Binding materials

Portland cement

The Type I ordinary Portland Cement (OPC) has the chemical characteristics shown in Table 3-1 (Conner 1990).

Table 3-1 Chemical characteristics of type I ordinary Portland cement

Cement Type	Designation or Characteristics	C_3S	C_2S	C_3A	C_4AF	Free CaO	$CaSO_4$	Specific Gravity	Specific Surface Area cm^2/g
I	Normal Portland-general use	45	27	11	8	0.5	3.1	3.17	3220

The mineralogical composition of Type I ordinary Portland cement is shown in Table 3-2 (Conner 1990).

Table 3-2 Mineralogical composition of type I ordinary Portland cement

Cement Type	**Designation or Characteristics**	**Ignition Loss**	**Insoluble Material**	**SiO_2**	**Al_2O_3**	**Fe_2O_3**	**CaO**	**MgO**	**SO_3**
I	Normal Portland-general use	0.6	0.1	22	5.1	3.2	65	1.4	1.6

Fly ash

The chemical composition of fly ash type F used in this study is shown in Table 3-3, ASTM C 618-08a (2008). Unconfined compressive strengths of fly ash are in the range 1.38 to 6.895 *MPa* with permeabilities between 10^{-5} to 10^{-8} *cm/s*, (Conner 1990). This fly ash was obtained from St. Laurence Cement Company in Longueuil, Quebec.

Table 3-3 Chemical composition of type F fly ash

Chemical Composition	**Class F (%)**
Silica (SiO_2)	35
Alumina Al_2O_3	20
Iron Oxide (Fe_2O_3)	6
Total SiO_2 +Al_2O_3+Fe_2O_3	70 min
Sulfur Trioxide (SO_3)	5 max
Calcium Oxide (CaO)	5
Magnesium Oxide (MgO)	5 max
Moisture Content	3 max
Loss on ignition	12 max
Available alkali as Na_2O_3	1.5 max

Slag

Ground granulated blast furnace slag used in this work was obtained from Lafarge North America Cement Company in Montreal, Quebec under the trade name NewCem®. It is a light grey odorless powder produced in accordance with ASTM C 989 (2006). Its composition and physical and chemical properties are shown in Tables 3-4 and 3-5 respectively (Lafarge 2007).

Table 3-4 Composition of slag used

Component	**Percent (By Weight)**
Slag	100
Calcium Oxide	30-50
Magnesium Oxide	0-20
Crystalline Silica	< 1
Particulate Not Otherwise Regulated	-

Table 3-5 Physical and chemical properties of slag used

Characteristic	**Description**
Physical State	Solid (powder)
Appearance	Gray/black or brown/tan powder
Odor	None
Vapor Pressure	NA
Vapor Density	NA

Specific Gravity	2-3
Evaporation Rate	NA
pH (in water)	8-11
Boiling Point	> 1000 °C
Freezing Point	None, solid
Viscosity	None, solid
Solubility in Water	Negligible

Calsifrit™

CAlSiFrit™ is a totally amorphous siliceous material, a matrix of calcium and sodium fluoro-aluminosilicate. This homogeneous solid substance has a blackish grey color, possesses a high reactivity potential and shows cementitious properties when finely ground. This product was obtained from the manufacturer NovaFrit International. Tables 3-6 and 3-7 show its chemical composition and physical and chemical properties respectively (NovaFrit 2006).

Table 3-6 Chemical composition of Calsifrit™

Carbon	**Fe_2O_3**	**Na_2O**	**CaF_2**	**CaO**	**Al_2O_3**	**SiO_2**
3-5 %	2-3 %	10-14 %	16-20 %	5-8 %	18-22 %	37-43 %

Table 3-7 Physical and chemical properties of Calsifrit™

Characteristic	**Description**
Physical State and Appearance	Solid (particles < 10*mm*)
pH (1% soln/water)	6.5-7.5
Melting Point	800 °*C*
Specific Gravity	1.3
Solubility	No (Water)
Color	Grey-Blackish
% Moisture	< 10%
Dispersion	No (Water)

3.2.1.2 Tailings materials

Bench scale and pilot tests carried out by Alcan International Ltd. on copper and gold tailings had shown the applicability of the thickening technology to a variety of tailings (Haile et al. 2000) in addition to the alumina tailings that were investigated by the latter authors. Therefore in this study, six different mine tailings were selected and obtained to evaluate a wide range of tailings from different types of mines;

1. Musselwhite tailings: Placer Dome (gold) mine: Musselwhite, Ontario,
2. Noranda Tailings: Noranda Inc.: Brunswick (lead-zinc) mine, Bathurst, New Brunswick,

3. Louvicourt tailings: Louvicourt (copper-zinc) Mine, Val. d'Or, Quebec,
4. Golden Giant tailings: Newmont Canada Limited, Golden Giant (gold) Mine, Marathon, Ontario,
5. Mont Wright tailings: Quebec Cartier mineral company, Mont-Wright (iron) mine, Quebec
6. Copper tailings: Murdochville. Gaspe (copper) mine, Quebec.

The mineralogical composition for Noranda, Louvicourt, and Golden Giant tailings is shown in Table 3-8. Table 3-9 shows the mineralogical composition for Mont Wright Musselwhite and copper tailings.

Table 3-8 Mineralogical composition for Noranda, Louvicourt and Golden Giant tailings

Noranda		**Louvicourt**		**Golden Giant**	
Constituent	**Percentage (%)**	**Constituent**	**Percentage (%)**	**Constituent**	**Percentage (%)**
Iron Sulfide (Pyrite)	55	Silica	0.4-0.8	Silica (Crystalline, quartz)	20-50
Lead (Sulfide)	5	Chalcopyrite	1	Sulfur (as Sulfides)	3-6
Silica (Crystalline, Quartz)	15	Sphalerite	1	Iron	3-6
Zinc (Sulfide) (Zn.Fe.S)	15	Sulfide	35	Barium	2-5
Arsenic (Sulfide) (As.Fe.S)	0.5 (max)	carbonates	63	Aluminum	< 0.5
Copper (Sulfide)	0.3-0.5				

Table 3-9 Mineralogical composition for Mont Wright, Musselwhite and copper tailings

Mont Wright	**Musselwhite**	**Copper**
Sandstone (quartzite)	Quartz (SiO_2)	sulfide tailings embedded in limestone rock
Mica schist	Birnessite-syn (MnO_2)	
Amphibolite	Calcium manganese oxide hydrate ($Ca_2Mn_{14}O_{27}.xH_2O$)	
gabroïque Granite	Dannemorite $(Fe,Mg,Mn)_7Si_8O_{22}(OH)_2$	
gabroïque feldsphate		
Specular hematite		
Specular Magnetite		
quartz		
diopside		
Tremolite		
actinolite		
Gruenerite		

3.2.2 Laboratory tests for characterization of tailings

Laboratory tests on these tailings were carried out and the measured physical and chemical properties of the tailings are presented below.

The following chemical and physical characteristic tests will be performed on the tailings to determine their index properties. These Index properties will include particle size distribution, moisture content, organic content and specific gravity of the tailings. The following sections explain the procedures used to determine these properties.

3.2.2.1 Particle size analysis

Representative samples from each tailings type were chosen after removing any unusually big chunks of tailings. Dry sieving (gravity sieving) was performed according to ASTM D 422 (2002) after drying these tailings specimens in an electric oven at 105°*C* for 24 *hours*. For Golden Giant, Musselwhite, Copper and Louvicourt, the sieving sample was 100 *g* each, whereas for Noranda and Mont Wright tailings, 115 *g* of tailings was sieved for each type.

Hydrometer tests were performed, as well, as suggested by Bowles (1986), on the Louvicourt, Musselwhite and Golden Giant mine tailings since more than 12% by mass of these tailings was smaller than 75 *micrometers*. Hydrometer analysis was performed in accordance with ASTM D 421 and D 422 (2002) using a solution of 4% $NaPO_3$ (sodium

hexametaphosphate) made by mixing 20 *g* of the $NaPO_3$ powder with enough water to make 500 *milliliters*.

3.2.2.2 Moisture content for the tailings

Initial moisture content for each type of tailings was determined as defined by ASTM D 2216 (1998) and by taking representative samples from each tailings container. After determining the wet weight, these samples were placed in an oven at 105 ^{o}C for 24 *hours*, after which another weight measurement was taken. This procedure was continued until constant weight was achieved.

3.2.2.3 Specific gravity for the tailings

Specific gravity tests were performed according to ASTM D 854 (1998). A representative sample was obtained from each of the six tailings types. These samples were oven dried for 24 *hours* at 105°*C* prior to determining their specific gravity. After that 100 grams of these oven-dry samples were put in 500 *milliliters* pycnometers with distilled water added after that. An electrical vacuum pump was connected to the pycnometer in order to remove any air bubbles entrapped within the tailings-distilled water mixture. By gently rotating and agitating the pycnometer it was assured that all air bubbles were eliminated from within these samples before taking the weight measurements. After vacuuming, a thermometer was inserted in the pycnometer to determine the temperature of the mixture.

Weight determinations of the pycnometer with tailings and water and the pycnometer with water alone were made using an electronic balance sensitive to 0.01 *g*.

The following equation was used to determine the specific gravity of the tailings, ASTM D 854 (1998):

G_s at $T_b = M_o / [M_o + (M_a - M_b)]$ ----------(3-1)

Where:

M_o = mass of sample of oven-dry tailings, *g*,

M_a = mass of pycnometer filled with water at temperature T_b, *g*,

M_b = mass of pycnometer filled with water and tailings at temperature T_b, *g*,

T_b = temperature of the contents of the pycnometer when mass M_b was determined, *°C*.

It is customary to report specific gravity values at 20 *°C*, hence the following corrective equation was utilized for this purpose, ASTM D 854 (1998):

G_s at 20 *°C* = K x (G_s at T_b) ----------(3-2)

Where:

K = a correction factor found by dividing the density of water at temperature T_b by the density of water at 20 *°C*.

3.2.2.4 Metal content

The six tailings types detailed above were examined for a list of heavy metal contaminants normally encountered in the tailings leachate (Fahey et al. 2002, Yanful et al. 2000, Adu-Wusu et al. 2000 and Yanful and Verma 1999). This list includes copper, nickel, zinc, iron, chromium and lead. Atomic absorption analysis was performed to determine the exact amount of these heavy metals that each type of tailings contains.

Atomic Absorption Analysis for the tailings

Mine tailings were digested using an OI analytical microwave digester. The tailings were oven dried at 105*°C* in a gravimetric oven. Digestion vessels were loaded with one gram (0.01*g*) of the oven-dry tailings and 40 *ml* of trace metal grade nitric acid. One control vessel containing nitric acid only was included as a procedural blank. A four stage pressure program was used whereby internal vessel pressure ranked to 140 *psi* during a 30 *minute* period. Metal content was then determined with a Perkin-Elmer Analytical 100 atomic absorption spectrometer.

3.2.2.5 Organic content

The loss on ignition test was performed to determine the organic content of the tailings whereby a representative sample of about 10 *grams* of each type of tailings were oven dried for 24 hours at 105 *degrees Celsius*. Then it was placed in a furnace to be exposed at 550 degrees Celsius for 3 hours and the change in weight between the two drying periods was used as an indicator to determine the amount of organic matter present in the tailings mass (Skempton and Petley 1970).

3.2.3 Stage 2: Formulation and characteristics of tailings binder matrices

Since the tailings-binder solidified mixtures investigated possess different content than the tailings alone, they will be referred to throughout this study as"tailings matrices".

3.2.3.1 Formulation of cubic specimens

Cubic specimens 2.5 cm3

Initially, it was decided to use small cubes; 25-*mm* ones since they are easier to cast, require less sample, have considerable less mass than 50-*mm* cubes and are therefore easier to handle, and require less moisture curing space. In addition, the smaller cross sectional area allows higher compressive strengths to be reached by a testing machine that has a smaller load capacity, (Kosmatka et al. 2002). Other researchers used the same

size of cubes for compression testing (Bhatty et al. 1999). These preliminary tests were conducted with the aim of finding the strength efficiency of using Portland cement alone and in combination with a pozzolanic material as a binder, and for finding the optimum cement combinations for these tailings matrices.

Type I ordinary Portland cement (OPC) was mixed with each type of tailings using a mechanical mixer as outlined in ASTM C 305 (1999). This type of cement is usually the least expensive and the most commonly used in chemical fixation and solidification applications (Conner 1990). In general, Portland cement systems produce stronger matrices than other inorganic binder systems, and they do it at lower mix ratios, which results in a smaller volume of waste requiring ultimate disposal (Weitzman et al. 1988). In the first part of these tests it was desired to determine the effect of the change in binder ratio on the compressive strength. Therefore Portland cement was mixed in proportions ranging from 30% to 52.5% by wet weight of tailings. Ten different proportions that increase by 2.5% were used for each of the six tailings. Each mix was poured subsequently in three different wooden cubic moulds of 2.5 cm^3. These mixes were then left to cure at room temperature (23 $°C$) for 24 *hours* before placing them in the moisture chamber.

The moisture chamber was constructed for the storage of the cured specimens. Temperature inside the chamber was maintained at 23.5^oC and the humidity at 98% (Wang and Vipulanandan 1996; Diez et al. 1997; Zamorani et al. 1989). Having a pool of water at the bottom of the chamber ensured that humidity was maintained at this

prescribed level at all times. Weight of these cubes was measured before placement in the chamber in order to determine density. The water content was kept constant in these tests for each type of tailings.

The same procedure was repeated to determine the change in water content on the compressive strength of the cubes in which cement content was maintained constant.

Cubic specimens 5 cm3

Since compressive strength of mortar is typically measured by preparing 50-*mm* cubes and subjecting them to compression according to ASTM C 109/C 109 M (1999), (Mamlouk and Zaniewski 1999). Further strength testing was conducted on the tailings matrices using (5 X 5 X 5) *cm* metallic molds after curing periods of 1, 7 and 28 days in the same humid environment.

3.2.3.2 Preparation of cylindrical specimens

Cylindrical specimens of tailings matrices measuring (44 diameter x 74 height) *mm* were molded for this purpose in accordance with ASTM D 4842 (1996). These cylindrical specimens were, then, cured in the moisture chamber described above for 28 *days* in the case of the wetting and drying samples, and for 43 *days* for the freezing and thawing. Unconfined compression tests were conducted on these tailings matrices after subjecting them to 12 cycles of drying at 60 °*C* and wetting by being submerged in room

temperature distilled water. The remaining samples will be compressed after 12 cycles of freezing at -20 °*C* in a freezing cabinet and thawing in distilled water at room temperature (22 °*C*). Table 3-10 shows the codes implemented for the cylindrical specimens of the tailings matrices.

Table 3-10 Cylindrical tailings matrices specimen codes

Tailings Type	Weathering Type	Code	Specimen	Binder/ Tailings	OPC/ Binder (%)	Calsifrit/Binder (%)	Fly ash/Binder (%)	Slag/Binder (%)
Mont Wright	Wetting/ Drying	MC1	1, 2, 3, 5, 6	0.5	100	0	0	0
		MC3	1, 2, 3, 4, 5, 6	0.5	90	10	0	0
		MC6	1, 2, 3, 4, 5, 6	0.5	75	25	0	0
		MCF1	1, 2, 3, 4, 5, 6	0.5	75	0	25	0
		MCF3	1, 2, 3, 4, 5, 6	0.5	75	10	15	0
		MCF5	1, 2, 3, 4, 5, 6	0.5	75	20	5	0
		MCS1	1, 2, 3, 4, 5, 6	0.5	75	0	0	25
		MCS3	1, 2, 3, 4, 5, 6	0.5	75	10	0	15
		MCS5	1, 2, 3, 4, 5, 6	0.5	75	20	0	5
Musselwhite		MW1	1, 2, 3, 4, 5, 6	0.375	100	0	0	0
		MW3	1, 2, 3, 4, 5, 6	0.375	90	10	0	0
		MW6	1, 2, 3, 4, 5, 6	0.375	75	25	0	0
		MWF1	2, 3, 4, 5, 6	0.375	75	0	25	0
		MWF3	1, 2, 3, 4, 5, 6	0.375	75	10	15	0
		MWF5	1, 2, 3, 4, 5, 6	0.375	75	20	5	0
		MWS1	1, 2, 3, 4, 5, 6	0.375	75	0	0	25
		MWS3	1, 2, 3, 4, 5, 6	0.375	75	10	0	15
		MWS5	1, 2, 3, 4, 5, 6	0.375	75	20	0	5
Mont Wright	Freezing / Thawing	MC'1	1, 2, 3, 4, 5, 6	0.5	100	0	0	0
		MC'3	1, 3, 4, 5, 6	0.5	90	10	0	0
		MC'6	1, 2, 3, 4, 5, 6	0.5	75	25	0	0
		MCF'3	1, 2, 3, 4, 5, 6	0.5	75	10	15	0
		MCF'5	1, 2, 3, 4, 5, 6	0.5	75	20	5	0
		MCS'1	1, 2, 3, 4, 5, 6	0.5	75	0	0	25
		MCS'3	1, 2, 3, 4, 5, 6	0.5	75	10	0	15
		MCS'5	1, 2, 3, 4, 5, 6	0.5	75	20	0	5
Musselwhite		MW'1	1, 2, 3, 4, 5, 6	0.375	100	0	0	0
		MW'3	1, 2, 3, 4, 5, 6	0.375	90	10	0	0
		MW'6	1, 2, 3, 4, 5, 6	0.375	75	25	0	0
		MWF'1	2, 3, 4, 5, 6	0.375	75	0	25	0
		MWF'3	1, 2, 3, 4, 5, 6	0.375	75	10	15	0
		MWF'5	1, 2, 3, 4, 5, 6	0.375	75	20	5	0
		MWS'1	1, 2, 3, 4,	0.375	75	0	0	25

3.2.3.3 Characterization of matrices

Specific gravity of the tailings matrices

Specific gravity of the tailings matrices was determined, similar to the above, in accordance with ASTM D 854 (1998). Each mixture was broken down into powder consistency and dried at 60 °*C* before performing these tests. 20 grams of this dried powder was used for performing these tests (Stegemann and Cote 1991).

Moisture content of the tailings matrices

Stegemann and Cote (1991) had shown that for hardened wastes, moisture content can be found by considering the wet weight of the sample rather than the dry weight as is traditionally used for soils. This approach was used in determining the moisture content of the hardened tailings matrices. A representative sample from each mixture was selected for this purpose. All samples were broken down into powder consistency before being put to dry in an oven at 60 °*C*.

Porosity

Porosity (P) of the solidified tailings was calculated using the following equation (Stegemann and Cote 1991):

$P = 1- [BD (1-w)/(G_s d)]$ ----------(3-3)

Where:

BD = bulk density,

w = moisture content,

G_s = specific gravity,

d = density of water (= 1 g/cm^3).

3.2.4 Stage 3: Assessment of the feasibility of tailings binder matrices as road construction materials

The methodological approach to assess feasibility of the matrices usage as road construction materials included engineering tests such as uniaxial compressive testing on cubes, unconfined tests for cylinders, Modified Proctor tests, California Bearing Ratio (CBR). The engineering tests were complimented by environmental tests such as wetting/drying and freezing/thawing weathering resistance tests and Toxicity Characteristics Leaching Procedure (TCLP).

Analytical methods were also used to assess the feasibility of using the tailings binder matrices as construction materials. These analytical approaches included finding the layer

coefficients for the tailings binder matrices and determining the predictive equation that governs the behavior with respect components involved in the formulation of the matrices.

3.2.4.1 Engineering tests

Index property tests were followed by engineering tests to determine the relative resistance of these mixtures to physical degradation caused by stresses resulting from physical and weathering processes. The uniaxial and unconfined compression testing, California Bearing Ratio and freeze/thaw weathering resistance will determine these properties. Once that has been established, the Toxicity Characteristics Leaching Procedure (USEPA 1992) was performed to assess the tailings-binder combinations' capability to retain heavy metals.

Uniaxial compressive testing

The integrity of the tailings matrices was assessed by subjecting these matrices to one dimensional compression testing to determine their structural suitability for land disposal and load bearing. Determining the strength of the tailings block is the first part of ensuring that they are suitable environmentally for disposal and for construction thereafter.

A first attempt was made to measure the compression strength for all the tailings matrices. Uniaxial compression testing was performed by subjecting the 25 *mm* cubes to

one dimensional compression from a Com-Ten Industries™ uniaxial machine with a peak compression force of 8.896 *kN*, which applied a constant speed of descent. The peak force and peak deflection maintained during each test were recorded and used subsequently in calculations.

The speed of descent of the platen was kept constant at an average of 70 *mm/minute* and the test was continued until visual observation of failure of the cube. After which the machine was cleaned and made ready for another test.

Uniaxial compression testing was performed on the 50 *mm* cubes through the application of compression loads on these specimens in their "virgin" state; immediately after being cured in the moisture chamber for the specified curing periods. Uniaxial compression testing was performed here by the use of an Instron® 4400 R testing machine applying a constant speed of descent. The peak force and peak deflection maintained during each test were recorded using an Instron® Series IX data logging system.

Unconfined compression testing after weathering

It is known, that, solidified waste in the field undergoes continuous weathering cycles. Therefore, strength testing will not be complete without investigating the effect of weathering cycles on the compression strength.

Speed of descent of the platen was also kept constant here at 70 *mm/min*. with the exception of a few samples from of Mont Wright where it had to be changed to 50 *mm/min*. in order to increase the maximum load the machine can apply, since these samples had higher compression loads than the other matrices.

Modified Proctor test

The modified Proctor compaction test was performed to find the optimum moisture content and maximum dry density for the tailings. This test is necessary to show the laboratory compaction characteristics and their relation to field densities. In addition, these values will be used to perform the California Bearing Ratio test. This test was conducted in accordance with ASTM D-1557 (2007). The test was performed at Terratech, a division of SNC-Lavalin, in Montreal, Quebec.

Initially, both types of tailings; Mont Wright and Musselwhite, were oven dried to remove any traces of moisture present. Then they were spread on a large metal plate with an initial amount of water added and mixed thoroughly with the tailings. Then, a mechanical compactor was used to compact the tailings in a standard Proctor mold having a diameter of 101.6 *mm* and a height of 116.43 *mm*, to a minimum compaction degree of 92%. The tailings were compacted in 5 equal layers using the blows of the mechanical hammer dropped from a height of 457 *mm*. Tailings surface was made smooth flush with the top surface of the mold, after which the mold assembly was

weighed to determine the density of the mix. When that was done, a sample of approximately 100 *grams* was taken from the tailings for moisture content determination.

After that tailings were extruded and the whole operation repeated a few times until a maximum value of density was achieved and reduced with the addition of extra water.

California Bearing Ratio test (CBR)

Compaction characteristics of the tailings-binder combinations were investigated using the density-moisture content California Bearing Ratio test (ASTM D 1883 2007). This test is used to determine the relative quality of subgrade, subbase and base soils for pavements (Bowles 1986). The test was performed at Terratech, a division of SNC-Lavalin, in Montreal, Quebec.

Accordingly an amount of about 4.5 *kilograms* of each type of tailings was oven dried and then mixed with the appropriate amount of water to reach its optimum moisture content, as found by the modified Proctor test conducted earlier, and compacted in a standard CBR mold using the mechanical compactor. Tailings compaction was done in 5 equal layers, each compacted with 56 blows, inside the CBR mold using this hammer dropped from a distance of 457 *millimeters*, until a minimum compaction effort of 96% was achieved. One moisture content sample was taken before compaction to ascertain the value of the moisture content previously mixed. A surcharge of 4.54 *kilograms* was placed upon each sample before testing. Then load was applied from a standard triaxial

machine piston using a strain rate of 1.27 *mm/min*. Load and penetration data were recorded using the compression machine and two more moisture content samples were taken after the test was done. Stress and penetration data were recorded using a LabView™ data logging program and these data were then compared to those of a standard unit load required to obtain the same depth of penetration on a standard sample of crushed stone (Bowles 1986).

The CBR number is the ratio of the unit load required to affect a certain depth of penetration to the standard unit load required to obtain the same depth of penetration on a standard sample of crushed stone. Table 3-11 shows the standard unit load for a sample of crushed stone (Bowles 1986).

Table 3-11 CBR standard unit loads for a sample of crushed stone

Penetration (*mm*)	**Standard Unit Load (*MPa*)**
2.5	6.9
5	10.3
7.5	13.0
10.0	16.0
12.7	18.0

CBR = (test unit load/standard unit load) X 100 percent

The CBR number will be based on the load ratio for a penetration of 2.5 *mm* or 5 *mm*, whichever is higher.

3.2.4.2 Environmental tests

Freeze/thaw and wetting/drying weathering resistance

In cold climates repeated cycles of freezing and thawing can cause physical deterioration of the solidified waste matrix thus exposing it to contact with water and leaching mediums. Hence a method for measuring the solidified waste resistance to freezing and thawing was employed to determine its resistance to these environmental conditions according to ASTM D 4842 (1996). This test is intended for the evaluation of the freezing and thawing resistance of monolithic solidified wastes.

As such, small cylindrical specimens were molded in metallic molds measuring 44 *mm* diameter by 74 *mm* in height. After casting solidified tailings matrices in these molds, they were left to thaw in the moisture chamber for a period of 43 *days*. These samples were, then, subjected to 12 cycles of freezing at -20 *degrees Celsius* for 24 *hours*, followed by thawing in water at room temperature for 24 *hours*. The weight loss of the sample was measured and compared with that of a control specimen. The change in weight was used to determine the state of the structural integrity of the matrix. Six identical samples were used for each mixture; three test (denoted by T) and three control samples (denoted by C).

After removing the specimens from the freezing cabinet and the moisture chamber, 240 *milliliters* of distilled chilled water at 4 ºC was added to the frozen specimens. To the control specimens was added 240 *milliliters* of distilled water at room temperature (22 ºC). Then plastic wrap was placed on the beakers and all water covered specimens were stored at room temperature (22 ºC) for a period of approximately 23 *hours*.

Afterwards each specimen was transferred to another dry beaker prepared as before taking into account that all loosely attached particulates were removed by spraying them with distilled water from a wash bottle to the surface of the specimen. Water was left to drain into the beaker of origin (ASTM D 4842 1996). Then the specimens' weight loss was determined by measuring the amount of the solid residue in beakers by evaporating water in the oven. A total of 12 cycles was conducted in the prescribed manner.

The same cylindrical sample preparation procedure was repeated when performing the wetting/drying weathering resistance test, which was done according to ASTM D 4843-88 (2004). The major difference between these two tests was the temperatures involved. Wetting was performed by submerging the specimens in water at 23 *degrees Celsius* and drying was done in an oven at 60 *degrees Celsius*.

Toxicity Characteristics Leaching Procedure (TCLP)

This is a regulatory test designed to assist in waste classification designated as EPA 1311 (USEPA 1992). A sample of tailings matrix was crushed to powder consistency and then extracted with acetic acid for 18 *hours* at a 20:1 liquid to solid ratio with an initial pH of 5. This sample was agitated in a rotary extractor for 18 *hours* at 30 rpm and 22 *ºC*. After 18 *hours* of agitation, the sample was filtered through 0.6-0.8 *micrometer* glass fiber filter. Then metal content of the extract, for the series of heavy metals investigated in this research, was determined with the Perkin-Elmer Analyst 100 atomic absorption spectrometer. Table 3-12 shows the sample numbers and repetition for the main tests in this study.

Table 3-12 Sample numbers and repetition for the main tests

Type of Test	Number of Samples	Repetition	Total Number of Specimens
Specific gravity of tailings	6	3	18
Specific gravity of matrices	53	3	159
Uniaxial compression	210	3	630
Unconfined compression	36	6	216 - 4 = 212
Freezing/thawing	32	3	96
TCLP	17	3	51

3.2.4.3 Analytical assessment

Layer coefficients are parameters computed by using the AASHTO nomograph (AASHTO 1986). The thickness of any particular road layer depends to a significant level on the layer coefficients. Layer coefficients will be determined for the tailings matrices and compared with values used by transportation

Regression analysis will be performed on the tailings matrices strength results to determine the predictive equation that governs behavior of the system. This equation is a function of the binders and tailings used in the formulation of the matrices. Also statistical techniques will be used to determine the accuracy and suitability of the results of the conducted characteristic, engineering and environmental tests.

3.3 Phase 2: Computational investigations

Phase 2 is divided into 3 stages: stage 1 is the program development for the tailings binder matrices using computational techniques. Stage 2 is the execution and verification of the program modeling of unconfined compressive strength and sensitivity analysis while stage 3 is the verification of the program modeling of freezing/thawing test results. Phase 2 is described in detail in Chapter 5.

Chapter 4

PHASE 1: EXPERIMENTAL RESULTS AND DISCUSSION

Testing of the tailings and their binder matrices will commence by performing the characteristic (index) tests followed by the engineering tests. This chapter will first introduce the results of these tests followed by the discussion of the results.

4.1 Results

4.1.1 Stage 1: Characteristic tests

Characteristic tests describe the physical and chemical properties of the tailings and the tailingsmatrices.They include: particle size analysis, specific gravity, moisture content and organic content.

4.1.1.1 Particle size analysis

Particle size analysis was performed on the tailings following methods described in Section 3.2.2.1.

Figure 4-1 shows the particle size distribution curves for the tailings includingdry sieving for Copper, Noranda and Mont Wright tailings as well as both dry sieving and hydrometer tests for the Louvicourt, Musselwhite and Golden Giant tailings.

Table 4-1 outlines the physical characteristics of these tailings types where it can be seen that according to the Unified Soil Classification System (Das 2000), Musselwhite, Louvicourt, Golden Giant and Copper tailings are classified as sandy silts whereas Noranda tailings are considered well-graded silty sand and Mont Wright tailings are poorly-graded sand. Mont Wright tailings are coarse textured grayish purple silty sand with traces of gravel. Noranda tailings are grayish green gravely silty sand.

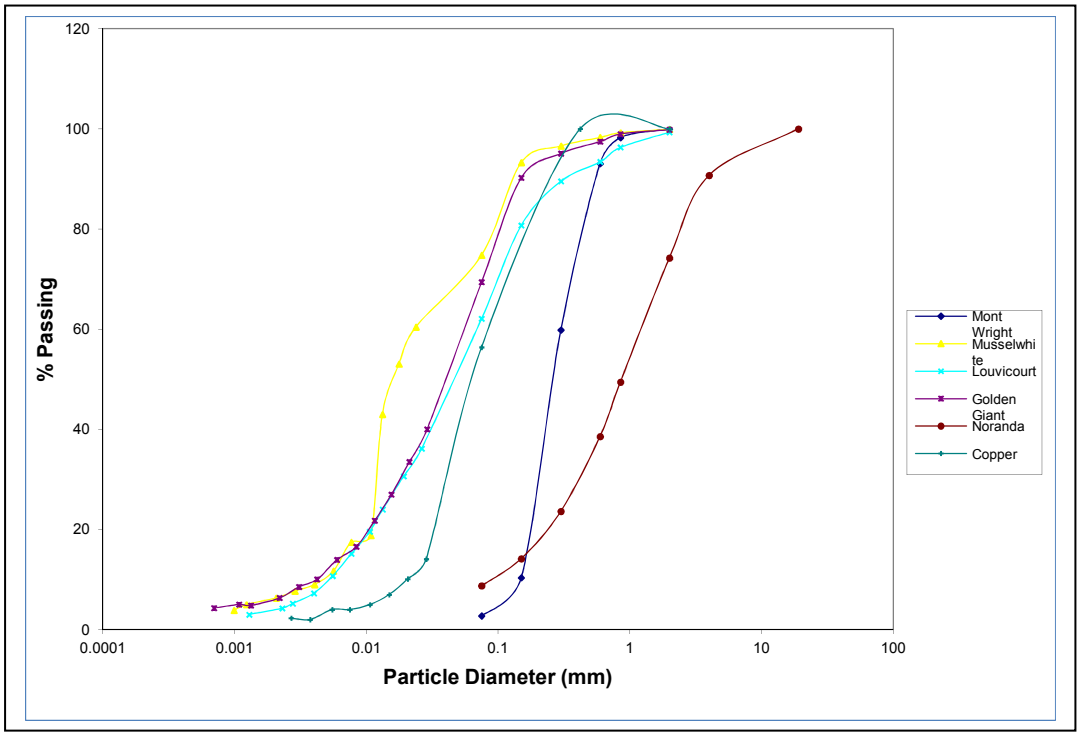


Figure 4-1Particle size distribution of the tailings

Table 4-1 Physical properties for the tailings

Type of Tailings	Noranda	Musselwhite	Golden Giant	Louvicourt	Copper	Mont Wright
D_{10} (*mm*)	0.063	0.0045	0.004	0.005	0.02	0.147
D_{50} (*mm*)	0.81	0.016	0.039	0.0464	0.0622	0.255
D_{60}(*mm*)	1.19	0.023	0.055	0.0697	0.0844	0.3
D_{30} (*mm*)	0.398	0.012	0.0178	0.018	0.04	0.2
C_u	18.89	5	13.75	13.94	4.22	2.04
C_z	2.11	1.39	1.44	0.93	0.948	0.91
$P_{4.75mm}$ (%)	92	100	100	100	100	100
$P_{0.075mm}$ (%)	8.59	74.9	69.69	62.05	56.9	2.05
Initial moisture content (%)	17.04	30.15	19.71	28.1	22.33	4.27
USCS	SW-SM	SM	SM	SM	SM	SP

Where:

D_{10} = diameter corresponding to 10% finer,

D_{50} = diameter corresponding to 50% finer,

D_{60} = diameter corresponding to 60% finer,

D_{30} = diameter corresponding to 30% finer,

C_u = uniformity coefficient = D_{60} / D_{10},

C_z= coefficient of gradation = D_{30}^2 /(D_{10} D_{60}),

$P_{4.75mm}$ (%) = percentage passing sieve no.4,

$P_{0.075mm}$ (%) = percentage passing ASTM sieve # 200,

SW-SM = well graded sand with silt,

SM = silty sand,

SP = poorly graded sand,

USCS = unified soil classification system.

4.1.1.2 Specific gravity of the tailings

Table 4-2 shows the specific gravities of the tailings samples measured and corrected for 20°*C*as described in Section 3.2.2.3. Specific gravity results shown below are within a range of 1.94-3.73, which is also found by other researchers(Mittal and Morgenstern (1975), Qiu and Sego (2001), Garand et al. (2000) Haile et al. (2000), Demers and Haile (2003) and Crowder et al. (2000)).

Table 4-2 Specific gravities of the tailings

Tailings Type	Copper	Mont Wright	Noranda	Golden Giant	Musselwhite	Louvicourt
SpecificGravity	3.4	2.76	3.68	2.97	3.26	3.33

4.1.1.3 Metal content

Results of the atomic absorption analysis for the tailings (Table 4-3) show that the materials still contain a high amount of metals particularly iron (up to 270 g/kg) and copper (1 g/kg).

Table 4-3 Results of the Atomic absorption analysis of the tailings

Tailings	**Cu** ***mg/kg***	**Fe** ***mg/kg***	**Cr*mg/kg***	**Zn** ***mg/kg***	**Pb*mg/kg***	**Ni** ***mg/kg***
Mont Wright	ND	3660	ND	ND	ND	ND
Golden Giant	32	38000	193.6	72	4	ND
Musselwhite	144	87200	67	36	180	ND
Louvicourt	884	49600	17.6	1280	200	ND
Noranda	408	76000	ND	1004	8760	ND
Copper	994	270100	215	4120	325	43

4.1.1.4 Organic content

Loss on ignition test for both Mont Wright and Musselwhite tailings revealed no indication of organic matter present in these tailings.

4.1.2 Stage 2: Matrix formulation

4.1.2.1Specific gravity of the tailings matrices

The specific gravity of 19 tailings matrices are showed in Tables 4-4, B-31, B-32 and B-33. These values were found using Equations (3-1) and (3-2). The values shown in this table are the average of three determinations corrected for a standard temperature of 20 °*C*. Values range between 2.48 and 3.53, which is slightly lower than the specific gravity for the tailings materials.

Table 4-4 Data range and Student t 95% confidence interval for the specific gravity tests for Mont Wright and Musselwhite matrices after 1, 7 and 28 curing days

Matrix	Mont Wright				Musselwhite			
	DataRange		Student T		DataRange		Student T	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC (1 *day*)	2.98	2.66	3.21	2.37	3.41	3.25	3.55	3.13
95% OPC, 5% CF (1 *day*)	3.16	2.71	3.48	2.36				
90% OPC, 10% CF (1 *day*)	2.83	2.68	2.94	2.57	3.76	2.66	4.56	1.82
75% OPC, 25% CF (1 *day*)	3.11	2.83	3.34	2.63	3.38	3.24	3.48	3.12
100% OPC (7 *days*)	3.33	2.94	3.65	2.66	3.43	3.23	3.58	3.09
95% OPC, 5% CF (7 *days*)	2.78	2.67	2.86	2.58				
90% OPC, 10% CF (7 *days*)	2.66	2.48	2.8	2.34	3.11	3	3.19	2.91
80% OPC, 20% CF (7 *days*)	3.53	3.15	3.81	2.87				
75% OPC, 25% CF (7 *days*)	3.43	3.12	3.65	2.85	3.23	3.04	3.36	2.85
100% OPC (28 *days*)	2.88	2.67	3.06	2.53	3.34	3.03	3.56	2.75
95% OPC, 5% CF (28 *days*)	3.03	2.82	3.18	2.65				
90% OPC, 10% CF (28 *days*)	2.9	2.81	2.97	2.73	3.26	3.22	3.29	3.19
75% OPC, 25% CF (28 *days*)	3.06	3.02	3.1	2.98	3.36	3.27	3.43	3.21
75% OPC, 25% FA (1 *day*)	3.05	2.87	3.22	2.73	3.23	3.04	3.39	2.9

75% OPC, 15% FA, 10% CF (1 *day*)	3.08	2.92	3.2	2.77	3.27	3.17	3.35	3.09
75% OPC, 5% FA, 20% CF (1 *day)*	3.18	2.93	3.44	2.74	3.23	3.09	3.33	2.96
75% OPC, 25% S (1 *day*)	2.92	2.69	3.09	2.51	3.37	3.08	3.58	2.82
75% OPC, 15% S, 10% CF (1 *day*)	2.93	2.89	2.97	2.85	3.24	3.05	3.39	2.92
75% OPC, 5% S, 20% CF (1 *day*)	3.04	2.97	3.1	2.91	3.12	3.04	3.19	2.96

4.1.2.2 Moisture content of the tailings matrices

Moisture content was measured following the procedure in Section 3.2.3.3. Tables B-6 and B-7 show values of the moisture content for the Mont Wright and Musselwhite tailings matrices, respectively. Generally, Mont Wright tailings matrices had higher moisture contents than the tailings material while Musselwhite had lower moisture content than its tailingsmaterial.

4.1.2.3 Bulk density

Tables B-1 to B-5 show the variation in average density with different binder contents. The general trend indicates an increase in density with the increase in Calsifrit replacement ratio. The increase in binder content (OPC, Calsifrit, Slag, Fly ash) increases the bulk densities because the binders filled the void space of the solidified specimens (Hills et. al (1993), Malviya and Chaudhary (2006)).

4.1.2.4 Porosity

Porosity values were calculated for the 5 cm^3 cubes with curing periods of 1, 7 and 28 *days*. Porosity values are presented in Tables 4-5 and 4-6 for the Mont Wright and Musselwhite tailings matrices respectively. Porosity for Mont Wright matrices ranged from 0.235 to 0.401 while porosity for Musselwhite matrices ranged from 0.32 to 0.427.

Table 4-5 Porosity values for Mont Wright tailings matrices

Mixture	Porosity
95% OPC, 5% CF (7*days*), Mont Wright	0.276
90% OPC, 10% CF (28 *days*), Mont Wright	0.302
90% OPC, 10% CF (7 *days*), Mont Wright	0.235
95% OPC, 5% CF (28 *days*) Mont Wright	0.327
80% OPC, 20% CF (7*days*), Mont Wright	0.401
100% OPC (7*days*), Mont Wright	0.363
100% OPC (1*day*), Mont Wright	0.306
75% OPC, 25% CF (1*day*) Mont Wright	0.336
95% OPC, 5% CF (1*day*), Mont Wright	0.340
90% OPC, 10% CF (1*day*), Mont Wright	0.288
100% OPC (28 *days*), Mont Wright	0.3
75% OPC, 25% CF (7*days*), Mont Wright	0.4
75% OPC, 25% CF (28 *days*), Mont Wright	0.344
75% OPC, 25% FA (1 *day*), Mont Wright	0.31
75% OPC, 15% FA, 10% CF (1*day*), Mont Wright	0.313
75% OPC, 25% S (1*day*), Mont Wright	0.27
75% OPC, 15% S, 10% CF (1*day*), Mont Wright	0.308
75% OPC, 5% FA, 20%	0.343

CF (1*day*), Mont Wright	
75% OPC, 5% S, 20% CF (1*day*), Mont Wright	0.322

Table 4-6 Porosity values for Musselwhite tailings matrices

Mixture	Porosity
90% OPC, 10% CF (1 *day*), Musselwhite	0.397
100% OPC (1 *day*), Musselwhite	0.425
75% OPC, 25% CF (1 *day*), Musselwhite	0.409
100% OPC (7 *days*), Musselwhite	0.427
90% OPC, 10% CF (7 *days*), Musselwhite	0.346
75% OPC, 25% CF (7 *days*), Musselwhite	0.349
100 OPC (28 *days*), Musselwhite	0.354
90% OPC, 10% CF (28 *days*), Musselwhite	0.368
75% OPC, 25% CF (28 *days*), Musselwhite	0.392
75% OPC, 25% FA (1 *day*), Musselwhite	0.325
75% OPC, 15% FA, 10% CF (1 *day*), Musselwhite	0.32
75% OPC, 25% S (1 *day*), Musselwhite	0.401
75% OPC, 15% S, 10% CF (1 *day*), Musselwhite	0.4
75% OPC, 5% FA, 20% CF (1 *day*), Musselwhite	0.352
75% OPC, 5% S, 20% CF (1*day*), Musselwhite	0.365

4.1.3 Stage 3: Assessment of the tailings binder matrices as construction material

4.1.3.1 Engineering tests

The engineering tests such as uniaxial and unconfined compressive tests, Modified Proctor, CBR, freezing/thawing and wetting/drying were conducted on the tailings alone and the tailings binder matrices.

Uniaxial compressive testing for 2.5 cm^3 cubes

Stage 1 tests were conducted with the aim of finding the strength efficiency of using Portland cement alone or in addition with a pozzolanic material for the solidification of these tailings, as well as finding the optimum cement combinations used for solidification. The advantage gained using these small cubes is that even a small change in parameters will affect the strength of the cubes.

Figures 4-2, 4-3 and 4-4 show the results of the tests (in *mega Pascals*) performed on the 2.5 cm^3 cubes. The uniaxial compressive strength corresponds to the maximum stress value observed during the compression test. Each value corresponds to the average of 3 determinations.

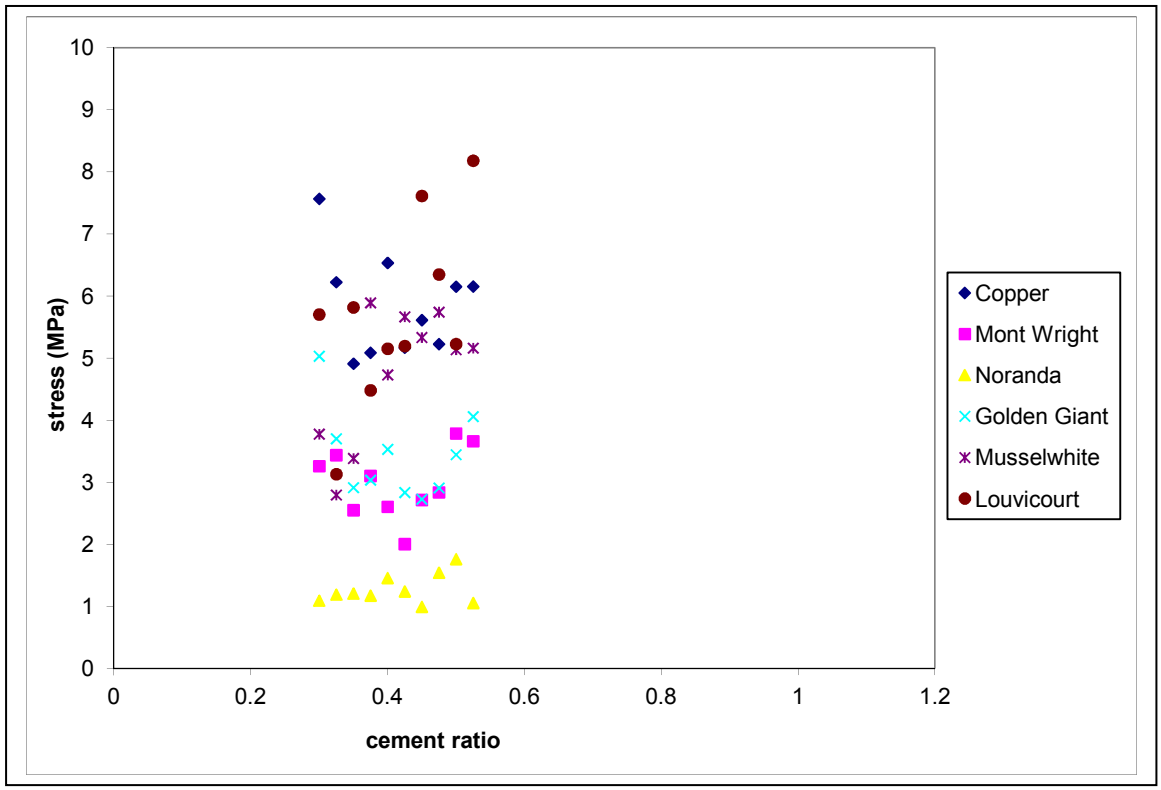


Figure 4-2 Uniaxial compressive strength versus cement ratio for all tailings matrices

It is seen in Figure 4-2 that there is a general increase in stress with increasing cement ratio for Louvicourt, Mont Wright, Noranda and Musselwhitetailings matrices whereas for Golden Giant and Copper tailings matrices no appreciable effect was noticed when varying the cement content. In addition Louvicourt and Copper tailings matrices maintained the highest compression strengthrecorded;8.18 and 7.56 *MPa*, respectively,whileNoranda matrices had the lowest stress; 1 *MPa*.

Then, cement content was maintained constant and the change in water content on the compressive strength of the cubes was determined.

When varying water/cement ratio, it is seen through Figure 4-3 that all the tailings matrices tested experienced a drastic decrease in stress by more than 50%, with the increase in water content.

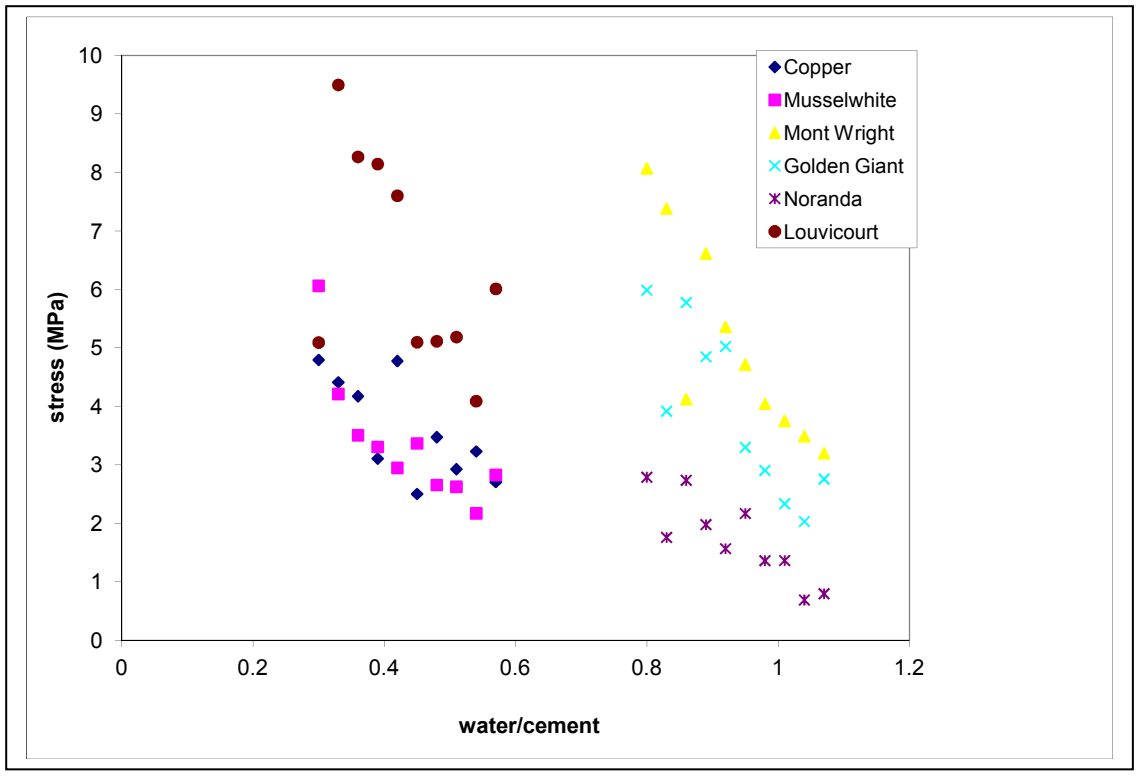


Figure 4-3 Stress vs. water/cement ratio ofthe tailings matrices

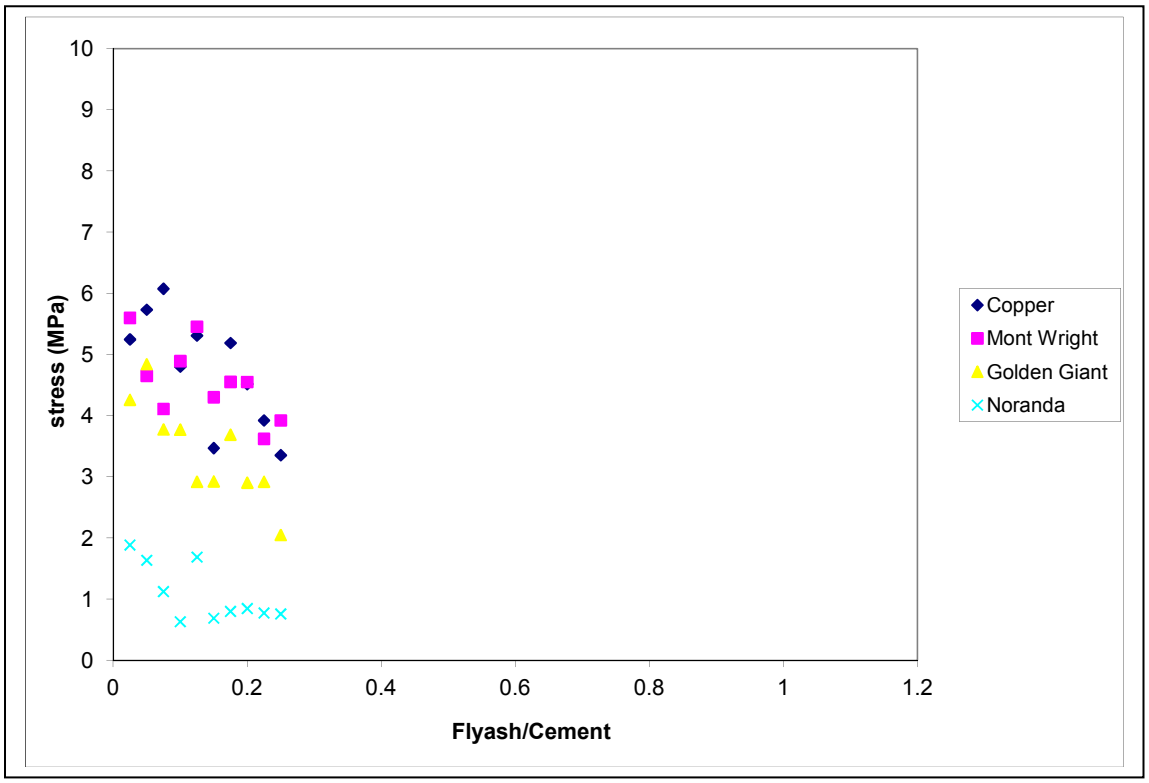


Figure 4-4 Uniaxial compressive strength versus fly ash/cement ratio for the tailings

Figure 4-4 shows that substituting part of the cement for fly ash has led to a generally decreasing trend of compressive strength with increasing fly ash content for Copper, Mont Wright, Golden Giant and Noranda tailings matrices. Compressive stress decreased by 36% for Copper, 30% for Mont Wright, 52% for Golden Giant and 60% for Noranda tailings, with the increase in fly ash content.

Uniaxial compressive testing for 5 cm^3 cubes

All the tailingstested,with the exception of Mont Wright, had relatively similar physical characteristics (Table 4-1), hence they were classified to silty sand. Mont Wright was classified to poorly-graded sand. Therefore, two of these tailings, namely

Musselwhiteand Mont Wright, were usedin further study to elucidate their strength characteristics using more compression and durability tests. Musselwhite was chosen as a representative of the silty sands and Mont Wright was chosen since it is the only tailings with poorly-graded sand.

Mont Wright was chosen since it was the only tailings in this combination classified as being poorly graded sand. Musselwhite, however was chosen as a representative of the remaining five other tailings since it has the highest percentage of fine particles as seen in Table 4-1 above.

Further strength testing was conducted on these two tailings using the (5 X 5 X 5) *cm* metallic molds in accordance with ASTM C109/ C 109 M (1999). Figures 4-5 to 4-8 show the results of these tests using different combinations of cement, flyash, slag and Calsifrit after curing periods of 1, 7 and 28 *days* in the same humid environment described in Section 3.2.3.1.

Compression stresses were applied on these specimens in their "virgin" state; immediately after being cured in the moisture chamber for different specified periods. All values shown are the average of 3 determinations.

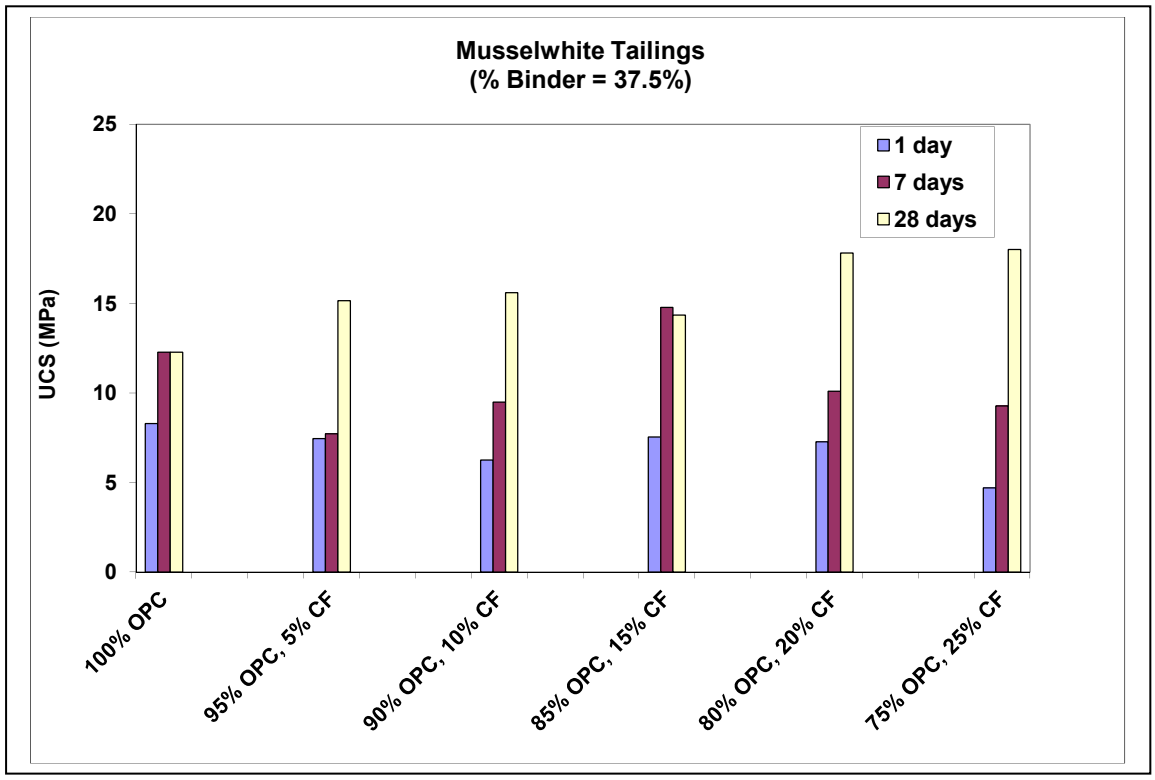


Figure 4-5 Uniaxial compressive strength versus binder ratio and binder content for Musselwhite tailings matrices for curing periods of 1, 7 and 28 days

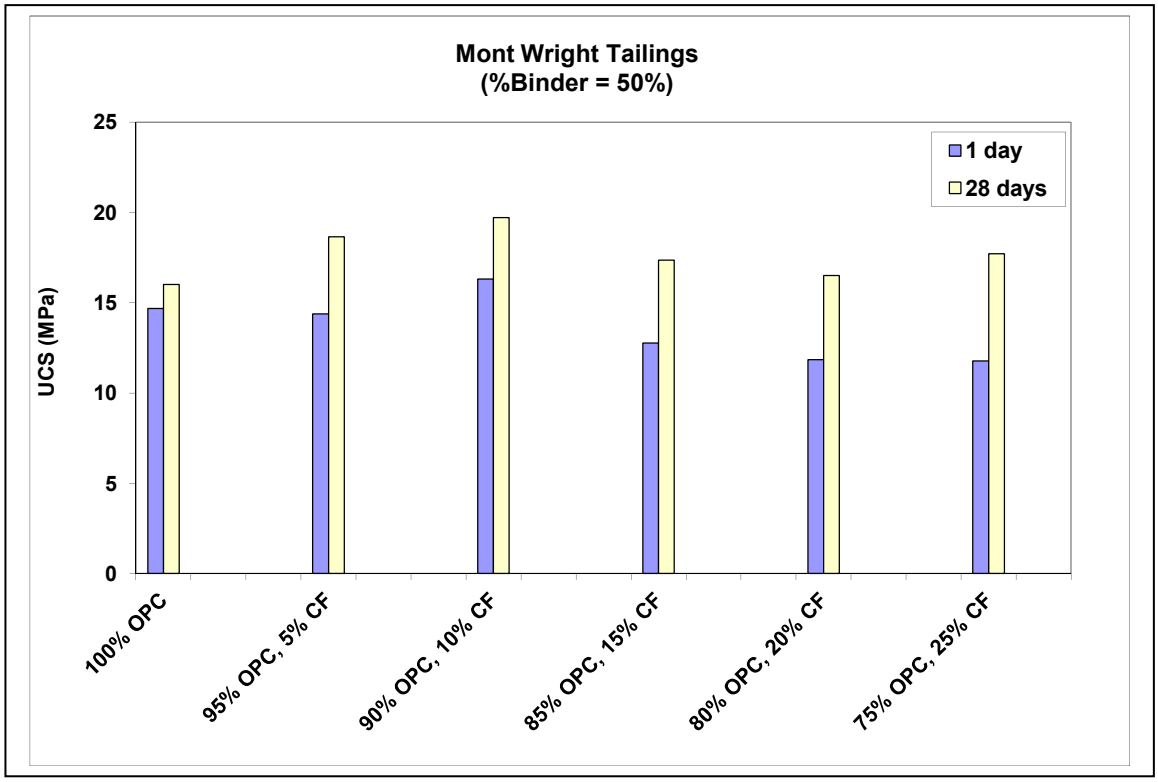


Figure 4-6 Uniaxial compressive strength versus binder ratio and binder content for Mont Wright tailings matrices for curing periods of 1 and 28 days

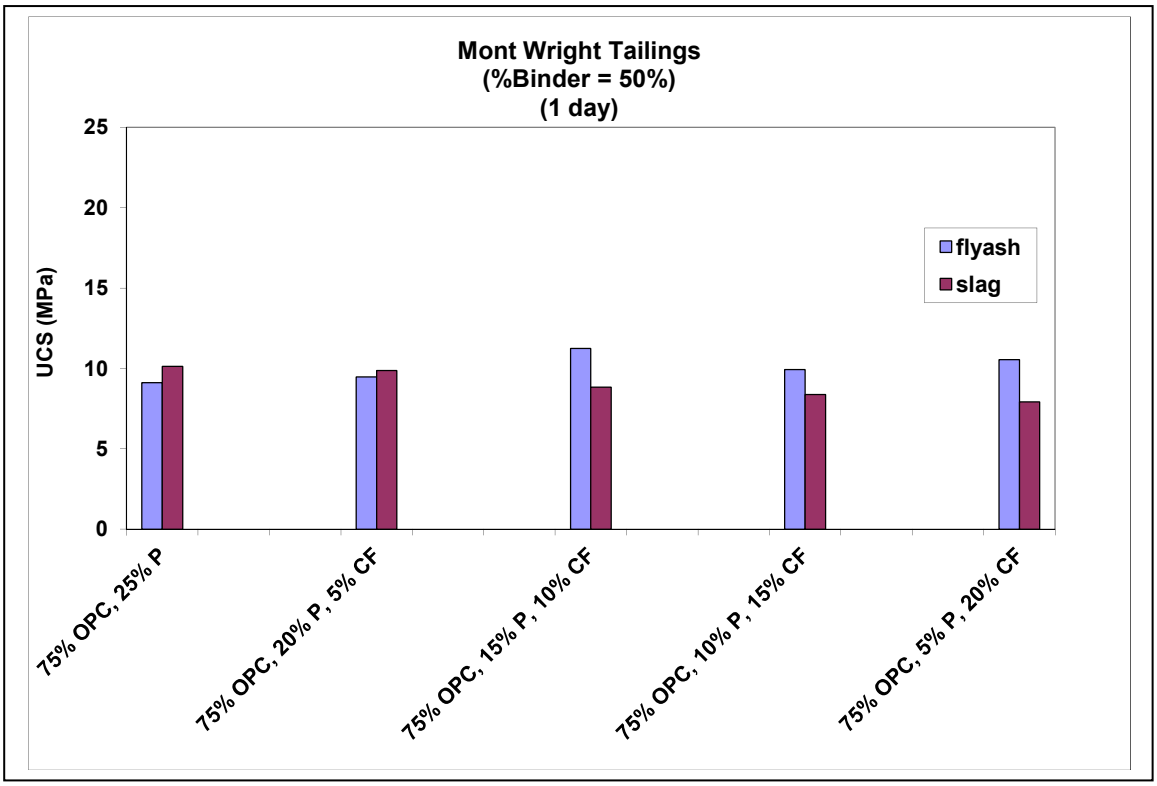


Figure 4-7 Uniaxial compressive strength versus binder ratio and binder content for Mont Wright tailings mixed with fly ash and slag for a curing period of 1 day

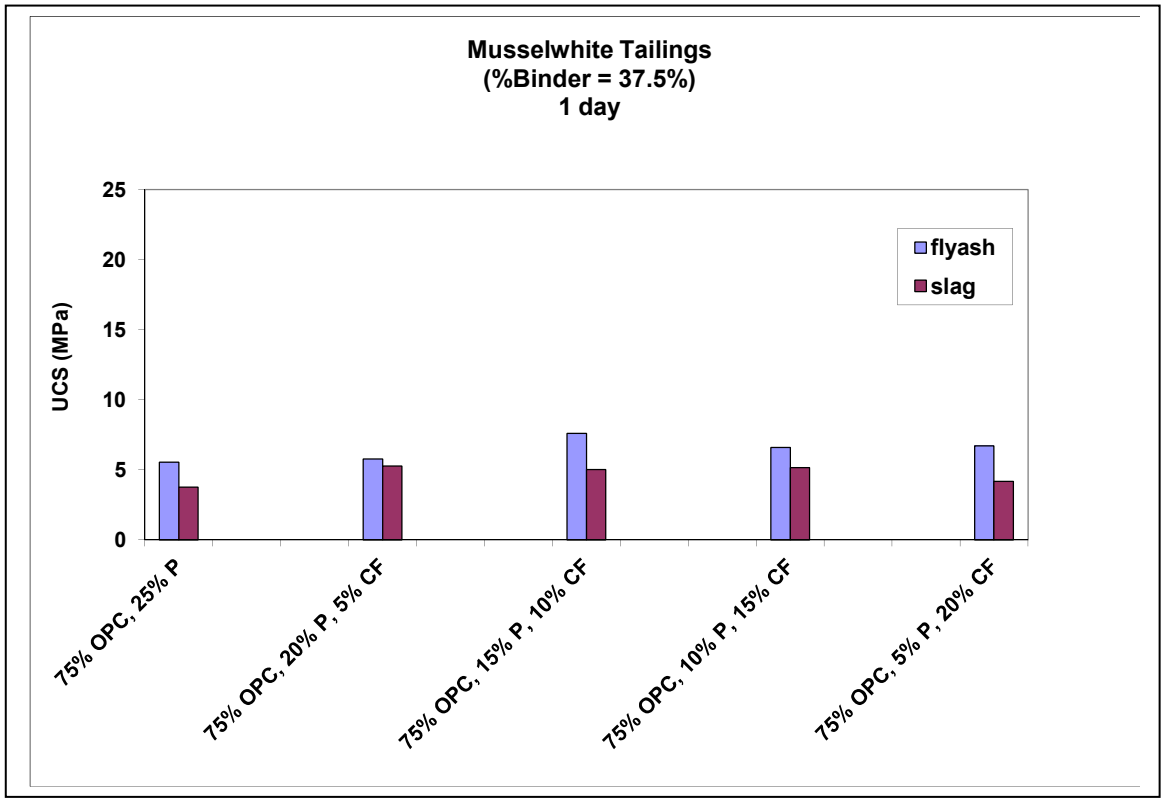


Figure 4-8 Uniaxial compressive strength versus binder ratio and binder content for Musselwhite tailings mixed with fly ash and slag for a curing period of 1 day

Unconfined compressive strength testing after weathering for cylinders

All cylindrical samples failed by having a diagonal crack along their length indicating the failure through the horizontal tension. The results of unconfined compressive strength testing for these tailings matrices after 12 cycles of wetting and dryingand the compression strength results for the same matrices after 12 cycles of freezing and thawing are presented in Figures A-1 to A-18 and Figures A-19 to A-36, respectively. Table 3-10 shows the codes implemented for these cylindrical specimens.

Modified Proctor test

Density values of this test are found taking into consideration that the volume of the test mold is 943.3 cm^3.

After determining the moisture content, the dry unit weight γ_d can be calculated as follows:

$\gamma_d = \gamma / (1 + w (\%)/100)$ ----------(4-1)

Where:

w (%) = percentage of moisture content.

The values of $\gamma_d$determined from Equation (4-1) were plotted against the corresponding moisture contents to obtain the maximum dry unit weight and the optimum moisture content for the tailings.

Also the 80% saturation line was drawn as a means of comparison and plotted beside the compaction curves. The following equation was used for this line:

$\gamma_s = G_s\gamma_w / (1 + (w\ G_s / S))$ ----------(4-2)

Where:

γ_s= 80% saturation unit weight,

γ_w= unit weight of water (= 9.8 kN/m^3),

G_s= specific gravity of tailings particles,

w = moisture content,

S = degree of saturation.

Figures 4-9 and 4-10 show such a compaction and the 80% saturation lines for Mont Wright and Musselwhite tailings respectively. It is shown here that Mont Wright tailings

had an optimum moisture content of 15.5% and a maximum dry density of 16.9 kN/m^3. Whereas Musselwhite had an optimum moisture content of 11.7% and a maximum dry density of 20.8kN/m^3. These values will be used subsequently when conducting the California Bearing Ratio test.

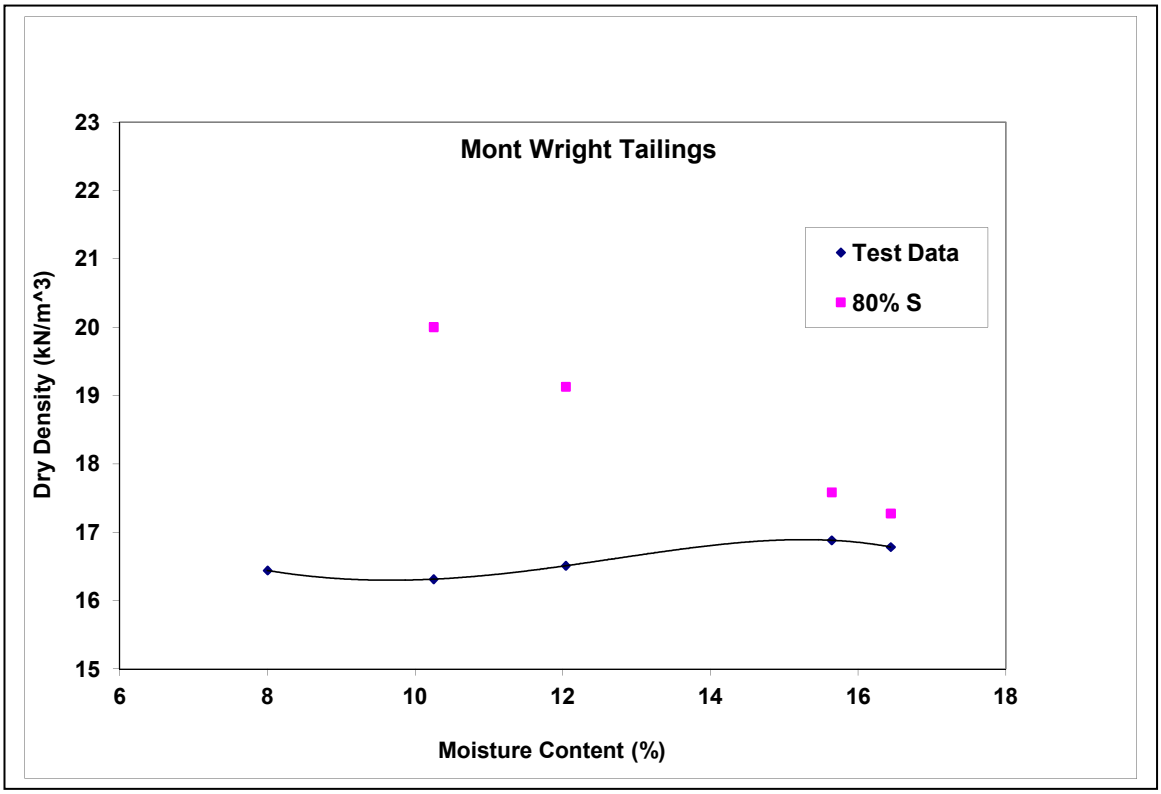


Figure 4-9 Modified Proctor test results and the 80% saturation line for Mont Wright tailings

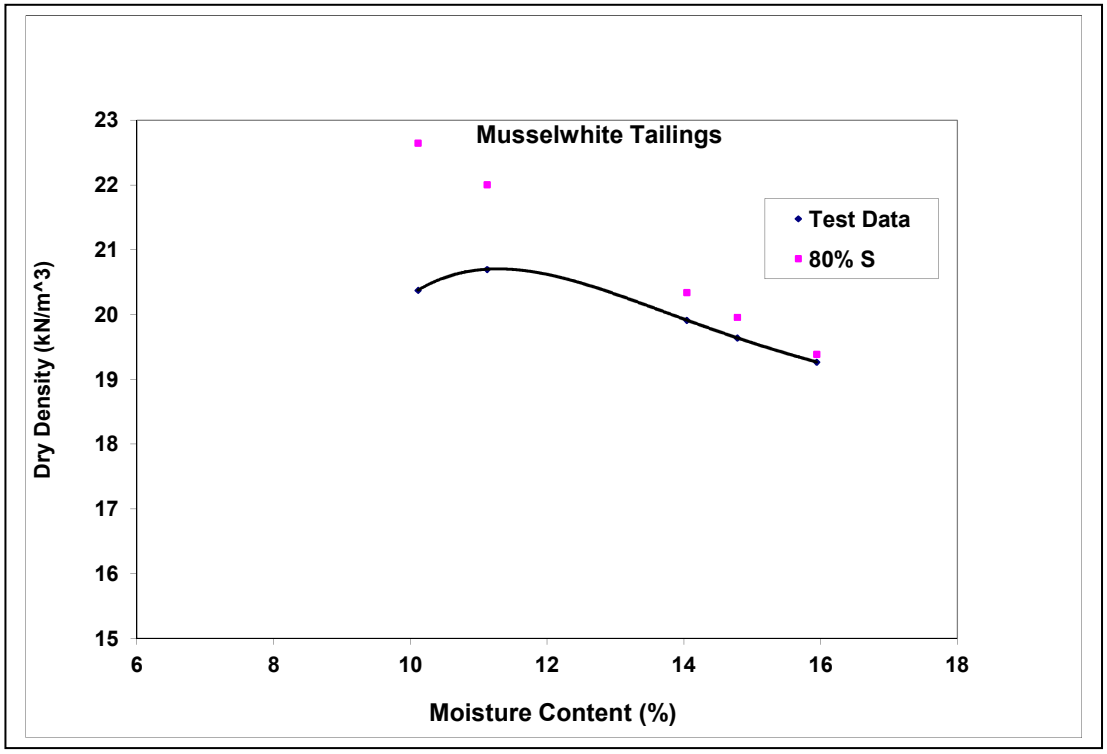


Figure 4-10 Modified Proctor test results and the 80% saturation line for Musselwhite tailings

California Bearing Ratio test

Figures 4-11 and 4-12 show the CBR test results for Mont Wright and Musselwhite tailings and their matrices respectively. Table 4-7 shows the moisture contents for both tailings before and after testing.

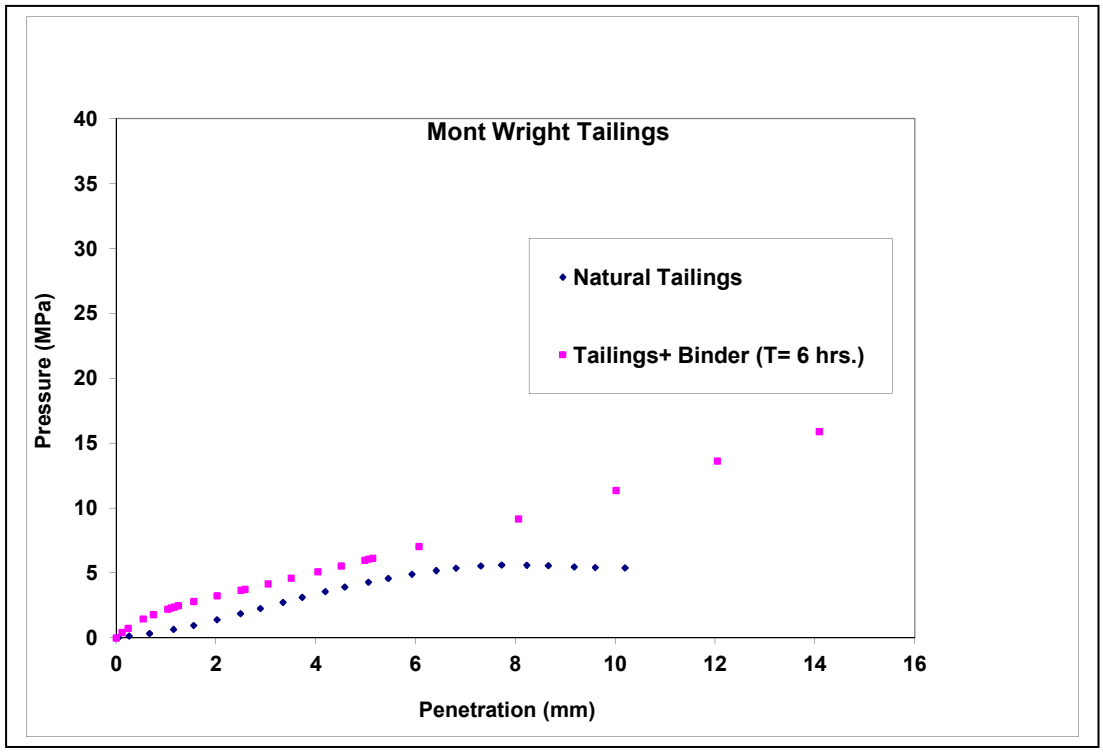


Figure 4-11 Pressure vs. penetration for the CBR test for Mont Wright tailings

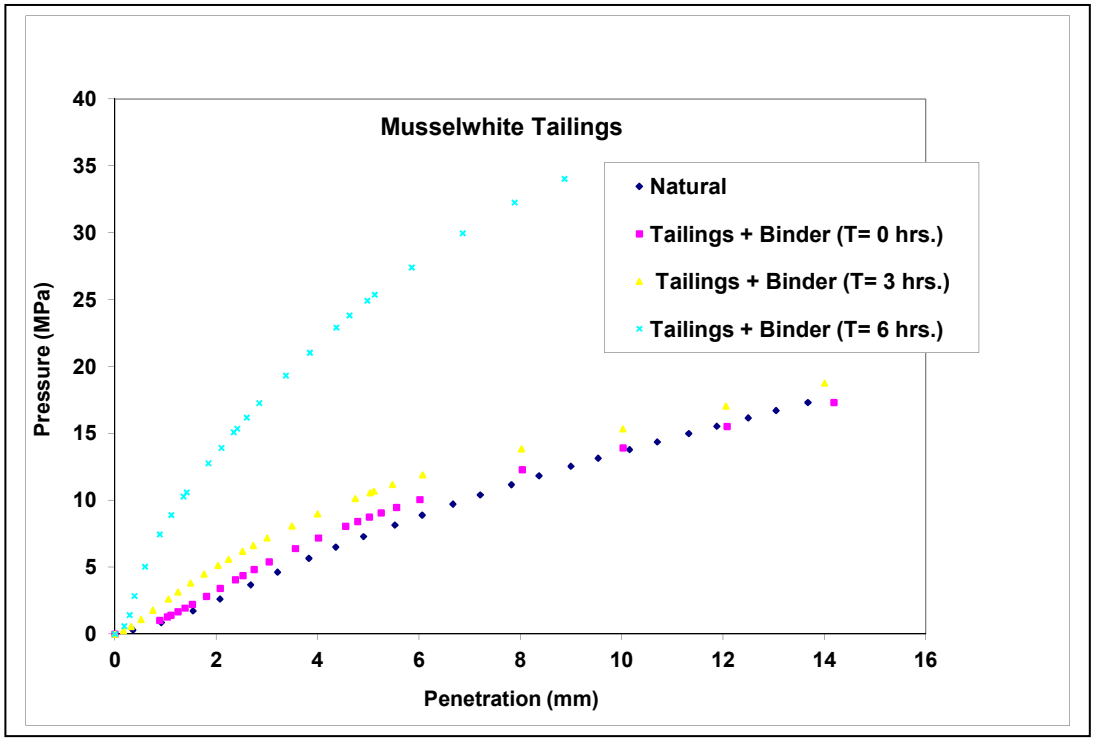


Figure 4-12 Pressure vs. penetration for the CBR test for Musselwhite tailings

Table 4-7 Moisture contents for both CBR samples before and after test

Tailings Type	**Moisture Content (%)**		
	Initial	After Test	
Mont Wright	15.6	13.5	15.3
Musselwhite	11.8	11.4	11.7

It can be found from these figures that stress values are higher for the 5.08*mm* than they are for the 2.54 *mm* penetration. Hence, based on a penetration of 5.08 *mm*, it can be concluded that Mont Wright tailings has a CBR index of 48.5 and Musselwhite tailings has a CBR index of 80.8.

4.1.3.2 Environmental tests

Freeze/thaw weathering resistance

Dry mass of the specimens is calculated as follows (ASTM D 4842 1996):

$M_s = (1-w/100)\, M_{sw}g$ ----------(4-3)

Where:

M_s= oven-dry mass of specimen in *grams*,

M_{sw}= initial mass of specimen in *grams*, and

w = moisture content.

Then the mass loss for each specimen is calculated after each cycle using the following equations:

$$W_{i,s,j} = T_{i,s,j} - B_{i,s,j}g \text{----------(4-4)}$$

Where:

$W_{i,s,j}$= mass loss of sample j during cycle i, in *grams*,

$T_{i,s,j}$= oven-dry mass of beaker and residue of sample j after after cycle i, in *grams*, and

$B_{i,s,j}$= oven-dry mass of beaker for sample j before cycle i, in *grams*.

And,

$$W_{i,c,j} = T_{i,c,j} - B_{i,c,j}g \text{----------(4-5)}$$

Where:

$W_{i,c,j}$= mass loss of control j during cycle i, in *grams,*

$T_{i,c,j}$= oven-dry mass of beaker and residue of control j after cycle i, in *grams*, and

$B_{i,c,j}$= oven-dry mass of beaker for control j before cycle i, in *grams*.

Subsequently the relative mass loss is calculated based on the ratio of the original mass of the oven dry sample, using the following equations:

$$R_{i,s,j} = W_{i,s,j} / M_{s,j}\% \quad \text{----------(4-6)}$$

Where:

$R_{i,s,j}$= relative mass loss of sample j during cycle i, %,

$W_{i,s,j}$= mass loss of sample j during cycle i, in *grams*, and

$M_{s,j}$= oven-dry mass of specimen j, in *grams*.

And,

$$R_{i,c,j} = W_{i,c,j} / M_{c,j}\% \quad \text{----------(4-7)}$$

Where:

$R_{i,c,j}$= relative mass loss of control j during cycle i, %,

$W_{i,c,j}$= mass loss of control j during cycle i, in *grams*, and

$M_{c,j}$= oven-dry mass of control j, in *grams*.

Figures 4-13to 4-16 show the freeze thaw weathering test results using different mixtures of cement, flyash, slag and Calsifrit, for both Mont Wright and Musselwhite tailings matrices.

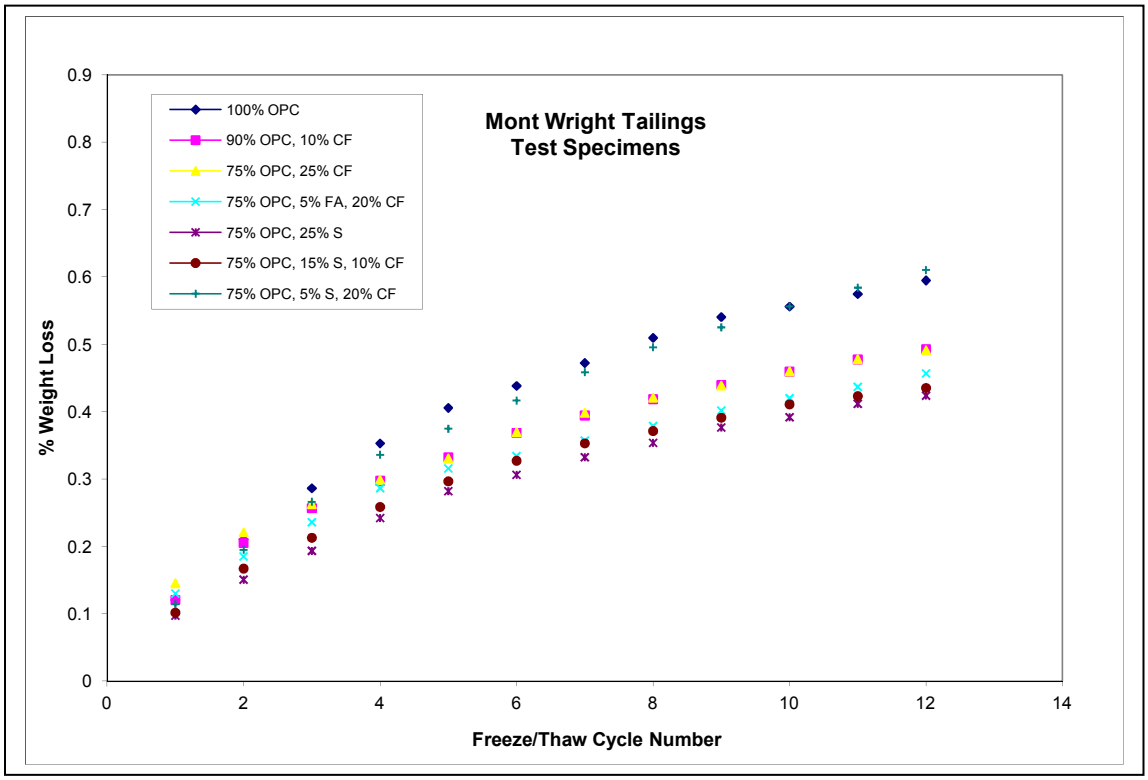


Figure 4-13 Weight loss after freezing and thawing test for Mont Wright test matrices

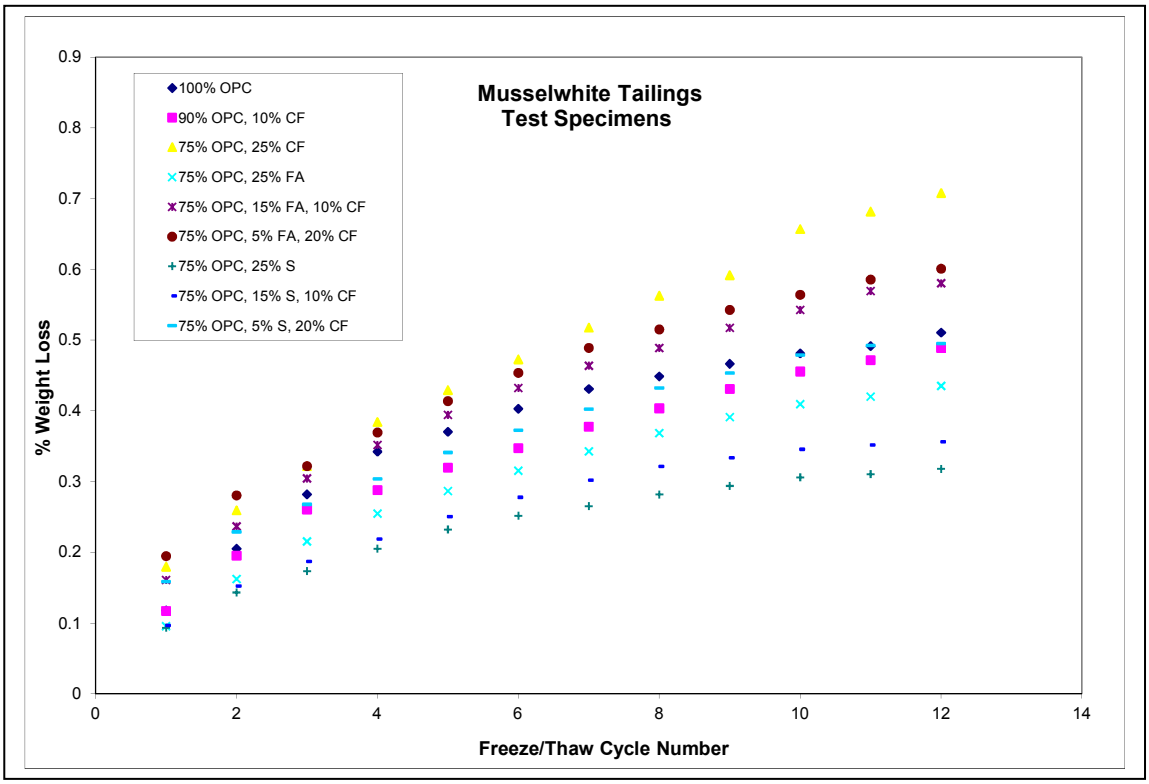


Figure 4-14 Weight loss after freezing and thawing test for Musselwhite test matrices

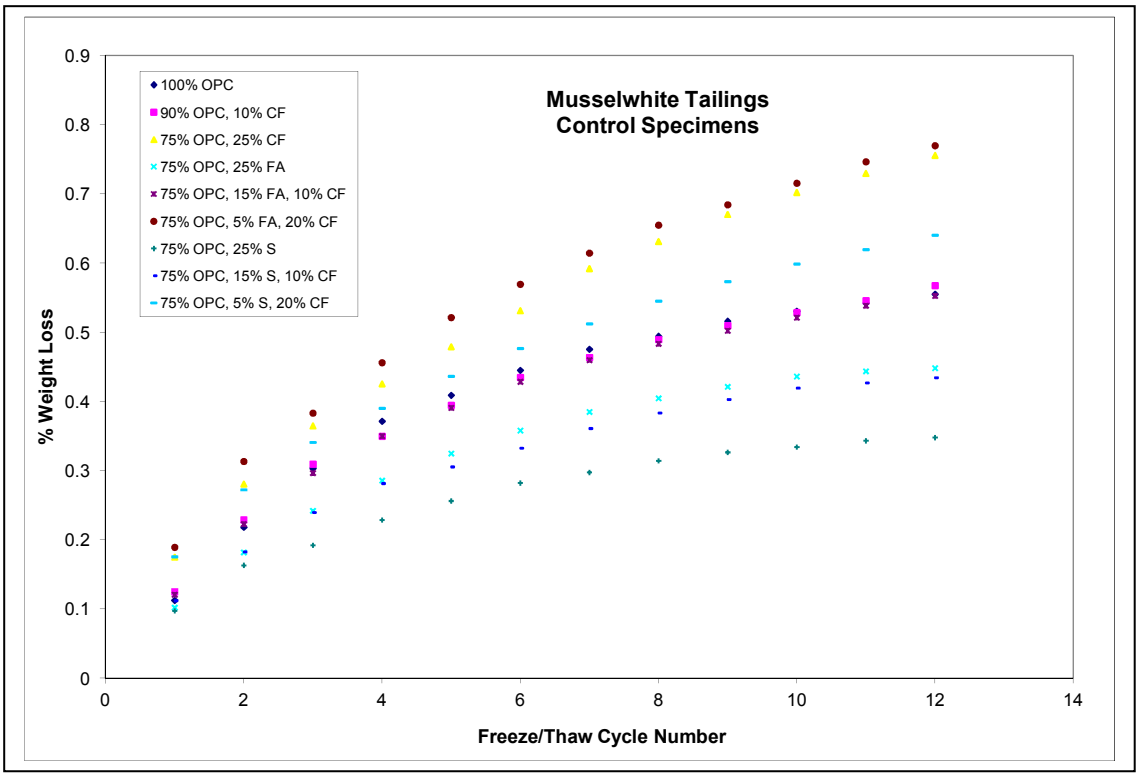


Figure 4-15 Weight loss after freezing and thawing test for Musselwhite control matrices

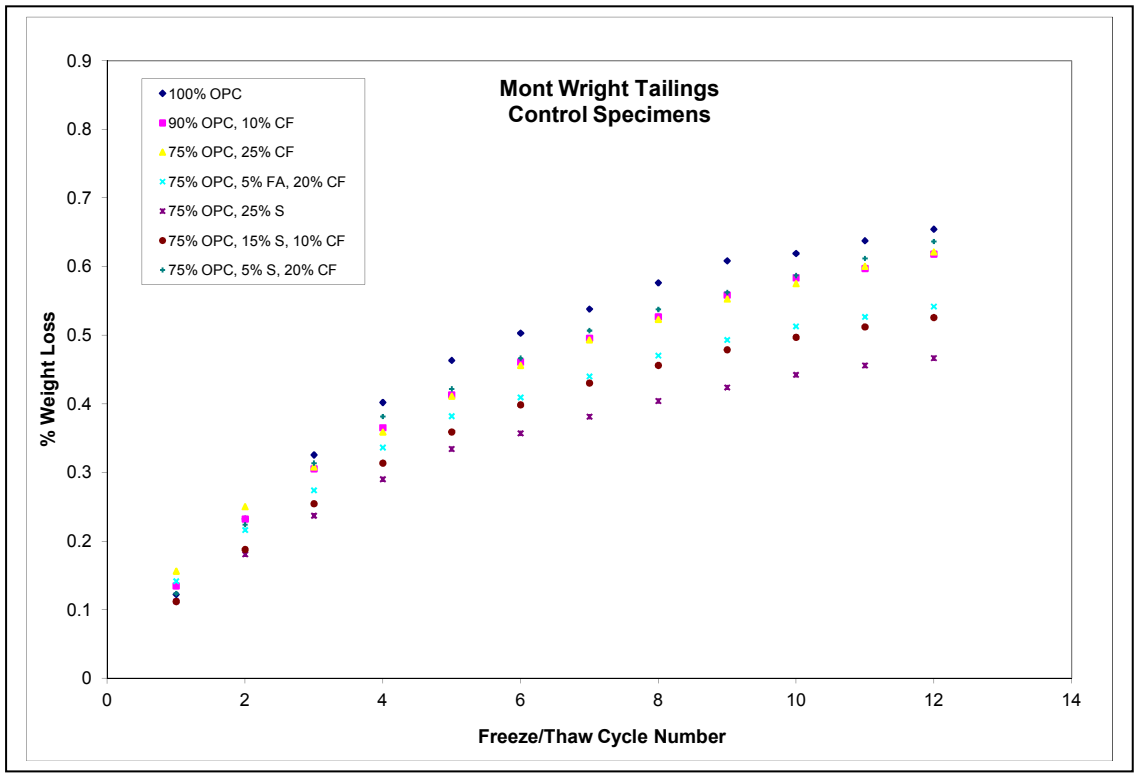


Figure 4-16 Weight loss after freezing and thawing test for Mont Wright control matrices

USEPA Toxicity Characteristics Leaching Procedure (TCLP)

Metal content was previously determined for the tailings specimens as explained in Section 4.1.1.3. After solidification of the tailings matrices, TCLP was conducted to find traces of either one of these heavy metals: Cr, Cu, Ni, Zn, Fe and Pb to determine the effect that solidification has on the new matrices. Results show that none of these metals was detected with the exception of Fe for some of the mixtures. Table 4-8 shows the results of the TCLP test on the hardened tailings matrices. The values shown are the average of 3 determinations.

Table 4-8 TCLP test results for Mont Wright and Musselwhite matrices

Mixture	Fe (mg/L)
100% OPC (28 *days*), Mont Wright	1.0
75% OPC, 5% FA, 20% CF (1*day*), Mont Wright	ND
75% OPC, 5% S, 20% CF (1*day*), Mont Wright	ND
75% OPC, 25% CF (1*day*) Mont Wright	ND
75% OPC, 25% S (1*day*), Mont Wright	ND
75% OPC, 25% CF (7*days*), Mont Wright	1.0
75% OPC, 25% FA (1 *day*), Mont	ND

Wright	

Mixture	Fe (mg/L)
75% OPC, 25% CF (28 *days*), Musselwhite	1.0
75% OPC, 25% FA (1 *day*), Musselwhite	ND
75% OPC, 5% S, 20% CF (1*day*), Musselwhite	ND
75% OPC, 25% S (1*day*), Musselwhite	1.0
100%OPC (28 *days*), Musselwhite	ND
100% OPC (1 *day*), Musselwhite	ND
75% OPC, 25% CF (7 *days*), Musselwhite	ND
100% OPC (7 *days*), Musselwhite	ND
75%OPC, 5% FA, 20% CF (1*day*), Musselwhite	1.0
75% OPC, 25% CF (1 *day*), Musselwhite	ND

ND = not detected.

4.2 Discussion

4.2.1 Stage 2

4.2.1.1 Effect of matrix combinations and impurities

It is seen in Figures A-1 to A-35 that Mont Wright tailings matrices have higher UCS values than their Musselwhite counterparts for the same mixtures. One reason for the increase of Mont Wright matrices strength is the increase in cement to tailings ratio as compared to Musselwhite: Mont Wright had a cement/wet tailings ratio of 0.5 compared to Musslwhite's 0.375. Another reason could be the presence of Zn in Musselwhite tailings. Previous studies have shown that Zn has a retarding effect in hydration of Portland cement (Trezza 2007).

For the wetting/drying case, strength increased for Musselwhite tailings matrices as 25% of the cement binder content was substituted with fly ash and a further increase took place as fly ash was replaced by 25% slag. The same effect was noticed with Mont Wright tailings matrices' control specimens.

Increasing the Calsifrit binder content for Mont Wright tailings matrices marginally reduced the strength for the cement only case. As for the case with fly ash and cement, partially replacing the fly ash content with Calsifrit kept the strength at almost the same previous levels. Partially replacing slag with Calsifrit increased marginally the strength for this combination.

For Musselwhite tailings matrices, increasing the Clasifrit content for the cement only case increased strength. Partially replacing fly ash with Calsifrit also increased the strength for Musselwhite tailings matrices. This effect, however, was noticed when partially replacing slag with Calsifrit.

For the freezing/thawing case, strength increased for Mont Wright tailings matrices as 25% of the cement binder content was replaced by slag. As for Musselwhite, replacing 25% of the cement binder content with either fly ash or slag marginally increased the strength of the specimens.

Strength increased for Mont Wright tailings matrices as 25% of the cement binder content was replaced with either slag or fly ash. Partially replacing fly ash with Calsifrit, in the presence of cement, increased the strength for Mont Wright tailings matrices. The same marginally increase in strength was noticed when slag was partially replaced by Calsifrit.

With Musselwhite tailings matrices, increasing the Calsifrit content marginally increased the strength in the presence of either fly ash or slag.

4.2.1.2 Porosity

Musselwhite matrices had higher porosity values than Mont Wright tailings matrices as can be seen in Tables 4-5 and 4-6. This is explained by the fact that Musselwhite, in comparison to Mont Wright, had higher initial water content as can be seen in Table 4-1.

Conner (1990) reported that as the water/cement ratio increased, the percentage of larger pores and therefore the permeability of the product increased. Increasing the percentage of Calsifrit in the matrices had generally led to a decrease in the porosity of the mixtures tested for Musselwhite tailings matrices. As for Mont Wright tailings matrices it seemed that the optimum Calsifrit percentage with the least porosity was 10% of the total percentage of binder used. A report submitted by the University of Sherbrook in Quebec established that using 25% Calsifrit as a Portland cement substitute decreased the permeability of concrete to chloride ion by 50%. Calsifrit does this by promoting a discontinuous pore structure (NovaFrit International 2006). Although pozzolans are accelerators of C_3S hydration (Massazza 1998), they themselves hydrate slowly, and pozzolanic cements tend to gain strength more slowly than Portland cement, unless an activator is used (Stegemann 2001). Using Calsifrit in addition to slag and fly ash reduced the permeability and improved the strength and durability of the final product in the longer term in accordance with other research studies on pozzolans (Stegemann 2001).Reduced permeability act to improve road layers performance since it will, in turn, reduce water infiltration to the ground water aquifer and hence more sustainable road construction.

4.2.2 Stage 3

4.2.2.1 Engineering tests

Effect of cement content

The scatter in the data of Figure 4-2 that shows UCS against cement ratio clearly demonstrates the effect that contaminants have on the hydration and strength characteristics of cement pastes and cement based products. This also suggests that cement content is associated with other products such as the heavy metals that compose the other part of the unhydrated mix of the strength specimens.Table 4-9 presentsthe optimum ratio of cement/wet tailings and water/cement, found through Figures4-2 and 4-3,respectively.

Table 4-9 Cement/wet tailings and water/cement ratios for the tailings

Tailings	Cement/Wet Tailings	Water/Cement
Copper	0.3	0.3
Mont Wright	0.5	0.8
Noranda	0.5	0.8
Golden Giant	0.3	0.8
Musselwhite	0.375	0.3
Louvicourt	0.525	0.3

Optimal Portland cement and water content values found above were used in making the samples for the wetting/drying and freezing/thawing tests. These values were used in preparing the specimens for the Mont Wright and Musselwhite tailings matrices.These optimum values will be used for the solidification of cement when combined with the other binders.

The US EPA requires a minimum compressive strength of 0.35 *MPa*for the stabilized/solidified waste to be safely disposed of in a landfill. This minimum guideline has been suggested to provide a stable foundation to be placed upon in a landfill(Malviya and Chaudhary2006). The unconfined compressive stress range as determined on these samples resembles that of other studies conducted in this regard (Demers and Haile 2003).All mixestested satisfy EPA landfill standards.

Field results confirm that using more cement with the same water/cement ratio showed insignificant strength gain, particularly considering the economics of cement expenditures. However stabilization using the same cement amount with lower water/cement ratios had significant strength gains (Ryan and Jasperse 1989), which is consistent with the results of this work shown in Figure 4-3.

Effect of water content

It is well known that strength decreases as matrix porosity increases, and that porosity increases as the water to cement ratio increases above an optimal value (Lea 1971 and Neville 1995). Figure 4-3 illustrates that strength decreases with increasing water content for Portland cement products containing metal contaminants. The same conclusion was reached by Stegemann (2001).

Modified Proctor tests

As expected, the Modified Proctor test for the Mont Wright tailings shows a one a half peak curve typical of poorly graded sand (Figure 4-9) and for Musselwhite tailings (Figure 4-10) shows a bell-shaped curve typical of silty sand (Das 2000). Although both tailings are characterized as sands, Musselwhite had a larger amount of fine particles (Table 4-1), finer than ASTM # 200 sieve, which when compacted in the Modified Proctor mold filled the voids among the coarser sand particles giving the mass an extra density for the same volume. This is in comparison to Mont Wright tailings that only have 2.05% of particles passing ASTM # 200 sieve. This had resulted in better compaction and higher density for Musselwhite tailings as evidenced in Figure 4-10. The maximum dry densities for Musselwhite and Mont Wright tailings were 20.8 kN/m^3 and 16.9 kN/m^3 respectively. The better compaction characteristics of Musselwhite tailings had left less void space for water hence the optimum moisture content of 11.7% for Musselwhite was smaller than that for Mont Wright, 15.5%, which had more voids to fill among the coarser sand particles. Both tailings reached 71% saturation at the optimum moisture content.

CBR tests

The CBR value of a construction material is oftenused as a benchmark in the assessment of its strength forpavement design. Materials that possess high CBR valuesare generally regarded as satisfactory for pavementconstruction. CBR tests were conducted to

investigate the performance of the stabilized samples. These tests were performed on the tailings alone and tailings matrices compacted in the standard CBR mould. The effects of additive mixtures on the CBR are given in Figures 4-11 and 4-12 for Mont Wright and Musselwhite tailings respectively.

When performing the CBR tests on the tailings without binder, Mont Wright achieved a CBR of 48.5 and Musselwhite had a CBR of 80.8, which are qualified as good and excellent base materials respectively Bowles (1986).

Robinson (2007) who did a study presentation on hydraulically bound materials at the British Lime Association seminar describes 20% of granulated blast-furnace slag and 1.5% lime activator to be used in slag bound mixtures. This fact was made with reference to more than 30 years of experience in France and the European continent. Another study by Richardson and Haynes (2001) found that compressive strength of slag bound mixtures increased sharply between 20% to 30% of slag content.

Since Calsifrit-OPC bound materials are not yet classified in standards, it was decided to follow slag bound materials specifications in performing the CBR test since they permit higher mixture content in comparison to cement bound and fly ash bound mixtures (Robinson 2007). This mixture more realistically emulates the binder percentages used in this study when performing the UCS test.

Therefore, adding 25% binder by weight composed of 90% Ordinary Portland Cement and 10% Calsifrit to both tailings had resulted in further increases in the CBR values. Mont Wright matrix achieved a CBR of 61.2, 6 *hours* after compaction and Musselwhite matrix had CBR values of 103.4, 116.7 and 279 at 0, 3 and 6 *hours* after compaction respectively. Mont Wright tailings matrix did not have any appreciable CBR value before 6 *hours* after compaction.

This can be explained by the addition of finer materials to the tailings causing an increase in cohesion. It also seems that adding cement and Calsifrit, which is a pozzolanic material, improved the packing of the tailings matrix at the interfaces with the soil grains resulting in a denser stabilized tailings matrix sample. Neville and Aitcin (1998) and Yarbasi et al. (2007) reported similar improved packing of soil particles leading to denser stabilized soil sample when using silica fume as an additive. Another reason for the increase in the CBR values when adding these additives is an increase in the maximum unit weight of stabilized samples and a decrease in the void ratio due to the addition of additive mixtures with fine particle size distribution (Yarbasi et al. 2007).

It was seen that adding Portland cement with Calsifrit had led to partial hydration of the tailings matrix mass with hydration improving and proceeding with time. For that reason testing was stopped at 6 *hours* after compaction and was not processed further. It was found through this test that CBR samples for both tailings matrices were hardened at 24 *hours* after compaction.

Ola (1975) observed that in cement stabilization thereis direct hydration of lime to form cementitious compounds.Bell(1976) attributed gains in CBR for materialstreated with cement to the cementation reactionproducts that formed, thus resulting in increasingstrength and improvement in compactability. Kezdi(1979) reported that in granular soils the cementationeffect is similar to that in concrete. He also stated thatthe greater thenumber and the larger the interparticlecontact surfaces, the stronger the effect of cementation.Hence, it can be concluded, that the sandy nature of the tailings and the well-graded character of Musselwhite tailings may have aided the cementation process and increased the CBR.

On the basis of the CBR values obtainedin this work, it is likely that cement-Calsifrittreated mine tailings matrix could be used as a road construction material, althoughthis would need to be carefully specified.

4.2.2.2 Environmental tests

Despite the fact that all engineering assessment tests confirmed the feasibility of using all matrices as road construction material, additional environmental tests were performed to define the potential risk these materials may pose for the environment. Environmental tests performed are the freeze/thaw durability tests and the TCLP.

Freeze thaw tests

ASTM stipulates that for a sample to fail the freeze and thaw test it has to lose 30% of its weight during or after the completion of this test (ASTM D 4842 1996). Accordingly, it can be seen from Figures 4-13 to 4-16 that all matrices tested for both Mont Wright and Musselwhite tailings matrices fared well in this test as the maximum cumulative weight loss did not exceed 0.8% during 12 cycles of freezing and thawing.

The figures also show that test and control specimens had similar weight loss values and characteristics, which is a good indication of rigidity of the mixtures and samples. They also show that adding Calsifrit to the cement-only specimens had a positive effect on durability for the Mont Wright testing and control specimens with weight loss decreasing with the addition of 10% and 20% Calsifrit to the cement-only mixtures. The same mixtures had less effect on the Musselwhite samples. On the other hand slag-OPC mixtures faired the best with the least weight loss out of all tailings matrices tested. The 5% Fly ash -20% Calsifrit -75% OPC mixture had less weight loss than the same matrix with slag for Mont Wright tailings. These mixtures had an opposite effect on Musselwhite matrix specimens.

The increase in durability of Mont Wright samples with addition of Calsifrit could be attributed to the improvement in the bond between the hydrated cement matrix and the tailings particles. This is due to the conversion of calcium hydroxide, which tends to form on the surface of aggregate particles, to calcium silicate hydrate (Toutanji et al. 2004).

Musselwhite tailings on the other hand had an amount of Zn and Pb that retarded hydration as discussed previously, preventing the creation of a stronger bond between the aggregate particles and the hydrated cement matrix. This could be the reason that Musselwhite tailings matrices generally suffered more weight loss than Mont Wright matrices.

Toxicity Characteristics Leaching Procedure (TCLP)

Since all samples met the limiting compressive strength of 0.35 *MPa*, TCLP tests were carried out for the crushed hardened matrices at 1, 7 and 28 *days* in order to determine the amounts of heavy metals that can be extracted from the samples. Results in Table 4-8 show high retention percentages of heavy metals in the hardened cement-pozzolan phases. The heavy metals with a high percentage of retention are Cr, Ni, Cu, Zn and Pb.

The US EPA specifies a regulatory level for Zn, Cr and Pb of 5 *mg/L* in the TCLP extract. For Cu the limit specified by EPA is 1 *mg/L* and for Fe 0.3 *mg/L*, USEPA, (2011).

As for the other metals, it specifies a value of 100 times the drinking water standards, LaGrega et al. (1994). The factor of 100 is to account for dilution into the environment.

Heavy metals including Cr and Ni were absent in the leachates by the addition of fly ash-Calsifrit or slag-Calsifrit. This may be due to the adsorption phenomenon of fly ash and

slag. Similar results for fly ash are reported by Singhal et al. (2008) and Zhang et al. (1999). All mixtures had lower concentrations than the EPA toxicity threshold levels for the range of metals tested with the exception of Fe for some of these mixtures. It is thought that the binding techniques investigated work on a lower Fe tailings concentration. The high initial Fe concentrations need be further investigated using higher binder concentrations.

The retention of heavy metal molecules in the hydrated Portland cement appears to be a combination of more than one process. It is suggested that some of those are ionic adsorption to the hydrated C-S-H in the hydrated cement paste, ionic incorporation into the crystalline network of some compounds of the hydrated cement such as sulfates in the ettringite hydrate and finally physical retention in the porous structure (Cheilas et al. 2007).

4.2.2.3 Analytical assessment

Statistical analysis

A quality program conducted by Stegemann et al. (2001) in which the products prepared by the University of Rome (Polettini et al. 2002), the University of Cantabria (FernándezOlmo et al., 2003a) and the University of Surrey (Gervais andOuki 2002, and Ouki and Hills 2002) were each tested in all three laboratories. The unconfined compression strength results from the quality control program (measured by ASTM C109

1992) have been used to calculate a pooled variance, which corresponds to an interlaboratory

Tailings Type	Copper	Mont Wright	Noranda	Golden Giant	Musselwhite	Louvicourt
Standard Deviation (*kPa*)	969.5	683.5	356.0	762.6	917.95	1534.6
Variance	939930.25	467172.25	126736	581558.76	842632.2	2354997.16

Table 4-10 Standard deviation and variance for the UCS tests for all cement only tests

standard deviation of 3430 *kPa*; measuring the UCS of separately prepared batches increased the interlaboratory standard deviation to 4120 *kPa* (Stegemann 2001).

Neville (1995) also reports that for high strength concrete a typical intra-laboratory standard deviation is in the range of 3500 to 5500 *kPa*. Table 4-10 shows the variance for the uniaxial compressive testing for the tailings matrices considered in this study.

A comparison between the values of standard deviation reported in the literature and cited above and the values found in this work (Table 4-10) indicates that compression values for this study have much less variability and the scatter is typical for cement paste mortars and therefore could be considered accurate and applicable for design and calculation purposes.

Student's t-distribution was used in estimating the mean of the samples for the 95% confidence interval. This distribution is usually used when the sample is small. The equation used is the following (Gosset 1908):

$X \pm t_c S/\sqrt{n}$ ----------(4-8)

Where:

X = mean value of parameter,

S = standard deviation of sample,

t_c= t score,

n = number of sample points.

It can be seen from Tables4-11 and 4-12for the cylindrical strength tests as well as Tables B-8 to B-12for the cube strength tests, that all sample data, testing and control, fall within the 95% confidence range as specified by this equation. The values shown in these tables are the average of three determinations corrected for a standard temperature of 20 *°C*. Table B-13 to B-17 (bulk density values), Table B-30 (TCLP) and Tables B-18 to B-29 (freezing/thawing results) show that experimental results are within student's t range.

When the sample size is increased to account for both testing and control sample in one mixture, the accuracy weakens but nevertheless most of the sample data fall within the 95% confidence interval. This is explained by the fact that testing and control samples followed different weathering patterns during the testing phase, hence the difference in their strength relative to each other.

Table 4-11 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite tailings matrices' cylindrical strength tests after wetting and drying

Matrix	Mont Wright				Musselwhite			
	DataRange		Student t		DataRange		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC	30.95	19.15	30.82	18.04	16.75	8.984	16.33	10.66
90% OPC, 10% CF	28.6	21.83	28.65	22.66	18.41	10.85	19.64	13.18
75% OPC, 25% CF	29.06	17.06	29.97	20.52	25.16	12.43	24.53	14.32
75% OPC, 25% FA	25.47	20.59	24.67	21.28	20.54	10.01	21.98	11.83
75% OPC, 15% FA, 10% CF	30.1	20.88	30.12	22.33	23.83	14.82	23.3	15.3
75% OPC, 5% FA, 20% CF	30.36	22.79	28.76	22.89	26.69	15.1	26.23	17.34
75% OPC, 25% S	32.48	26.35	31.32	25.93	23	16.03	22.83	16.74
75% OPC, 15% S, 10% CF	34	24.99	31.48	24.32	25.14	15.46	23.59	16.59
75% OPC, 5% S, 20% CF	36.64	26.3	33.39	25.54	20.55	13.87	19.84	15.15

Table 4-12 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite tailings matrices' cylindrical strength tests after freezing and thawing

Matrix	Mont Wright				Musselwhite			
	DataRange		Student t		DataRange		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC	26.88	20.73	25.86	21.15	18.64	12.08	19.16	13.97
90% OPC, 10% CF	30.19	20.4	30.76	21.86	25.15	16.01	24.8	17.09
75% OPC, 25% CF	31.3	23.38	30.42	23.76	26.27	19.4	26.23	20.43

75% OPC, 25% FA	29.43	18.28	27.02	17.84	19.57	16.48	19.26	16.3
75% OPC, 15% FA, 10% CF	30.05	22.4	30.07	24.06	20.65	12.23	19.71	13.72
75% OPC, 5% FA, 20% CF	33.08	23.4	32.31	23.7	18.39	14.57	18.37	14.94
75% OPC, 25% S	29.65	20.91	28.65	22.75	17.7	13.1	18.12	14.57
75% OPC, 15% S, 10% CF	28.73	26.49	28.64	27.01	21.35	10.26	21.77	11.71
75% OPC, 5% S, 20% CF	27.98	21.21	27.36	22.4	27.34	13.76	25.11	15.1

Layer coefficients

AASHTO highway design method requires knowledge of a structural number (SN), which is a function of several parameters including design traffic level, subgrade support, desired reliability and desired terminal serviceability, (AASHTO 1986). The equation relating the structural number to the layer coefficients is the following:

$$SN = a_1 D_1 + m_2 a_2 D_2 + m_3 a_3 D_3 \text{----------(4-9)}$$

Where:

SN = structural number;

a_1, a_2, a_3 = layer coefficients for the surface, base and subbase, respectively;

m_2,m_3=drainage coefficients of the base and subbase, respectively;

D_1,D_2,D_3= thickness of the surface, base and subbase layers, respectively.

Hence it appears from the equation above that the thickness of any particular layer depends to a significant level on the layer coefficients. Thus in order to assess the suitability of the tailings matrices for use in temporary access roads, for example, an analysis was made to determine the layer coefficient for use when designing an access road with tailings matrices as the base layer. Accordingly two different equations were used in this analysis; the first developed by the University of Missouri-Rolla (Richardson, 1996):

E_c=915.48 + 1314.9 q_u.....(4-10)

Where: E_c = Chord modulus of elasticity (*MPa*) = (σ_1-σ_2) / (ε_1-ε_2),

σ_1= 40% of ultimate strength (*MPa*),

σ_2 = stress at0.000050 *mm/mm* strain (*MPa*),

ε_1= strain at σ_1(*mm/mm*),

ε_2= 0.000050 (*mm/mm*),

q_u = the unconfined compressive strength (*MPa*).

And the second equation used was developed from data supplied by work performed by Felt and Abrams (1957), Reinhold (1955) and University of Missouri-Rolla (Richardson, 1996);

$$E_c = -34.367 + 2006.8\ (q_u)^{0.7784} \text{ ----------(4-11)}$$

Then layer coefficients are computed by using the AASHTO nomograph. The equation for the relationship of layer coefficients and modulus was derived from the nomograph and is the following (Richardson 1996):

$$a_2 = -2.7170 + 0.49711 \log E_c \text{ ----------(4-12)}$$

Where:

a_2 = layer coefficient.

Table 4-13 shows the cement-only layer coefficients for all tailings matrices used. The coefficients match reasonably well with values from 10 state departments of transportation reported in the literature, which range from 0.12 to 0.3 (ACI 1990), thus indicating that these mine tailings matrices are suitable structurally for road construction.

Other researchers such as Stegemann et al. 2000, Wang and Vipulanandan 2000 and Turner 1997, have performed unconfined compressive testing on cement hardened wastes containing heavy metals. Chord modulus values found for the weathered Musselwhite and Mont Wright cylindrical samples shown in Figures 4-17 to 4-20 showed that these values are consistent with their findings.

As expected compression strength values for the 5 *cm* cubes went higher as the curing period was increased from 1, 7 to 28 *days* and as the cement was allowed to gain more of

Table 4-13 Cement-only layer coefficients for all tailings used

Cement/Wet Tailings Ratio	**Layer Coefficient: Equation (4-10), Equation (4-11)**					
	Copper	**Mont Wright**	**Noranda**	**Golden Giant**	**Musselwhite**	**Louvicourt**
0.3	0.36, 0.34	0.21, 0.2	0.03, 0.01	0.29, 0.27	0.23, 0.22	0.31, 0.29
0.325	0.33, 0.31	0.21, 0.21	0.05, 0.03	0.23, 0.22	0.18, 0.17	0.2, 0.19
0.35	0.28, 0.27	0.16, 0.16	0.05, 0.03	0.19, 0.18	0.21, 0.21	0.31, 0.29
0.375	0.29, 0.27	0.2, 0.19	0.04, 0.02	0.19, 0.18	0.32, 0.3	0.26, 0.25
0.4	0.34, 0.31	0.17, 0.16	0.07, 0.06	0.22, 0.21	0.27, 0.26	0.29, 0.27
0.425	0.29, 0.27	0.12, 0.11	0.05, 0.03	0.18, 0.17	0.31, 0.29	0.29, 0.28
0.45	0.31, 0.29	0.17, 0.17	0.02, N/A	0.17, 0.17	0.3, 0.28	0.37, 0.34
0.475	0.29, 0.28	0.18, 0.17	0.08, 0.07,	0.19, 0.18	0.31, 0.29	0.33, 0.31
0.5	0.32, 0.3	0.23, 0.22	0.03, N/A	0.22, 0.21	0.29, 0.27	0.29, 0.28
0.525	0.32, 0.3	0.23, 0.22	0.05, 0.04	0.25, 0.23	0.29, 0.27	0.38, 0.35

its hydration products. Uniaxial compressive testing and chord modulus values for the cubes shown in Figures 4-21, 4-22 and4-23matched reasonably well with published values in the literature;FernándezOlmo et al. 2003a, Gervais and Ouki 2002, Polettini et al. 2001, Polettini et al. 2002, Stegemann et al. 2001 for the same type of cement hardened cubes containing heavy metals.

N/A = less than 0.01

It is shownin Table 4-13 that, in comparison to Equation (4-10), Equation (4-11)provides conservative values of the layer coefficient when used in Equation (4-12). Hence, it will be used for determining the layer coefficient for the other part of the tests and it is more generalized since it was developed by using data from 3 different institutions.

It is shown in Table 4-14 that using Calsifrit as an additive with Portland cement increased the values of the layer coefficients for both tailings matrices; Mont Wright and Musselwhite. Layer coefficients in this case were higher than the cement-alone layer coefficients; mixtures with 10% and 25% Calsifrit/cement ratio had higher layer coefficients in comparison with 100% cement samples for both wetting/drying and freezing/thawing samples.

Similarly, substituting Calsifrit for fly ash and slag resulted in higher values of layer coefficients; 10% and 20% Calsifrit replacement samples had higher layer coefficients when compared to the Portland cement-slag and the Portland cement-fly ash samples as can be seen in Table 4-14.

Table 4-14 Layer coefficients for Mont Wright and Musselwhite tailings matrices after weathering

	Binder Mixture for Matrix	Wetting/Drying		Freezing/Thawing	
		a_2	S.D	a_2	S.D
Mont Wright Tailings	100% OPC	0.53	0.036	0.53	0.016
	90% OPC, 10% CF	0.54	0.019	0.55	0.025
	75% OPC, 25% CF	0.54	0.034	0.55	0.02
	75% OPC, 25% FA	0.53	0.012	0.52	0.032
	75% OPC, 15% FA, 10% CF	0.55	0.025	0.55	0.018
	75% OPC, 5% FA, 20% CF	0.55	0.018	0.56	0.024
	75% OPC, 25% S	0.56	0.015	0.54	0.019
	75% OPC, 15% S, 10% CF	0.56	0.0197	0.56	0.005
	75% OPC, 5% S, 20% CF	0.57	0.020	0.54	0.016
Musselwhite Tailings	100% OPC	0.43	0.037	0.47	0.028
	90% OPC, 10% CF	0.47	0.036	0.51	0.030
	75% OPC, 25% CF	0.49	0.045	0.53	0.020
	75% OPC, 25% FA	0.47	0.048	0.48	0.013
	75% OPC, 15% FA, 10% CF	0.49	0.034	0.47	0.030

	75% OPC, 5% FA, 20% CF	0.51	0.035	0.47	0.017
	75% OPC, 25% S	0.5	0.025	0.47	0.019
	75% OPC, 15% S, 10% CF	0.50	0.028	0.47	0.053
	75% OPC, 5% S, 20% CF	0.48	0.022	0.5	0.041

S.D = standard deviation

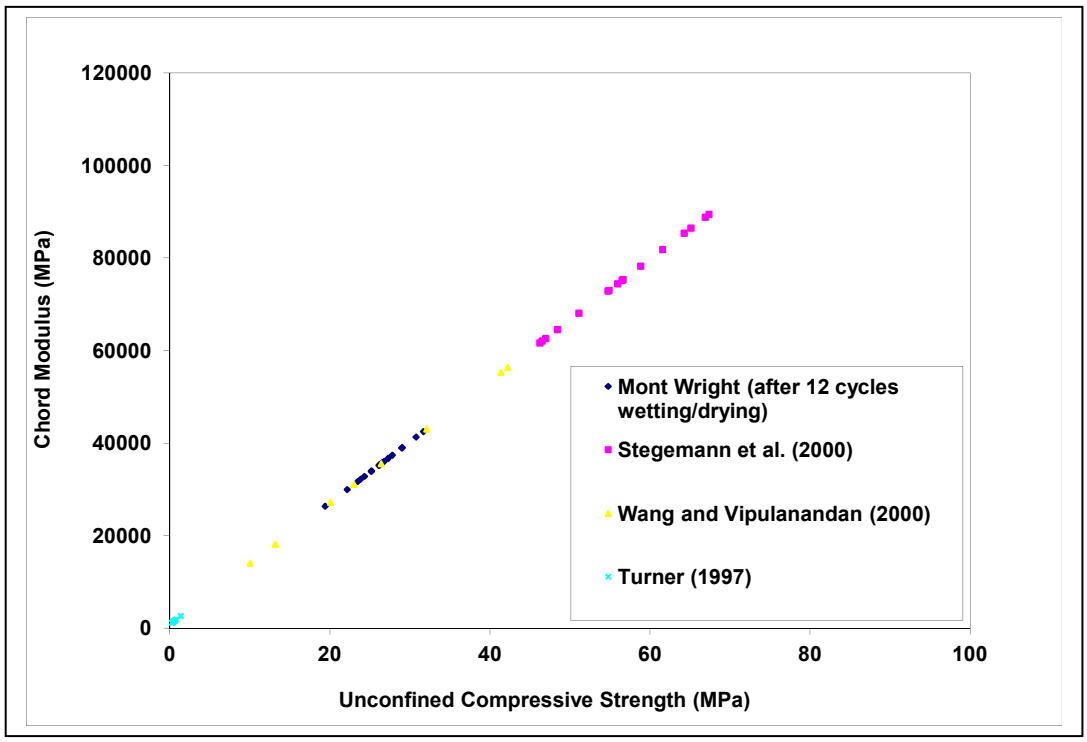


Figure 4-17 Chord modulus vs. unconfined compressive strength for Mont Wright tailings matrices after 12 cycles of wetting and drying with other values from the literature

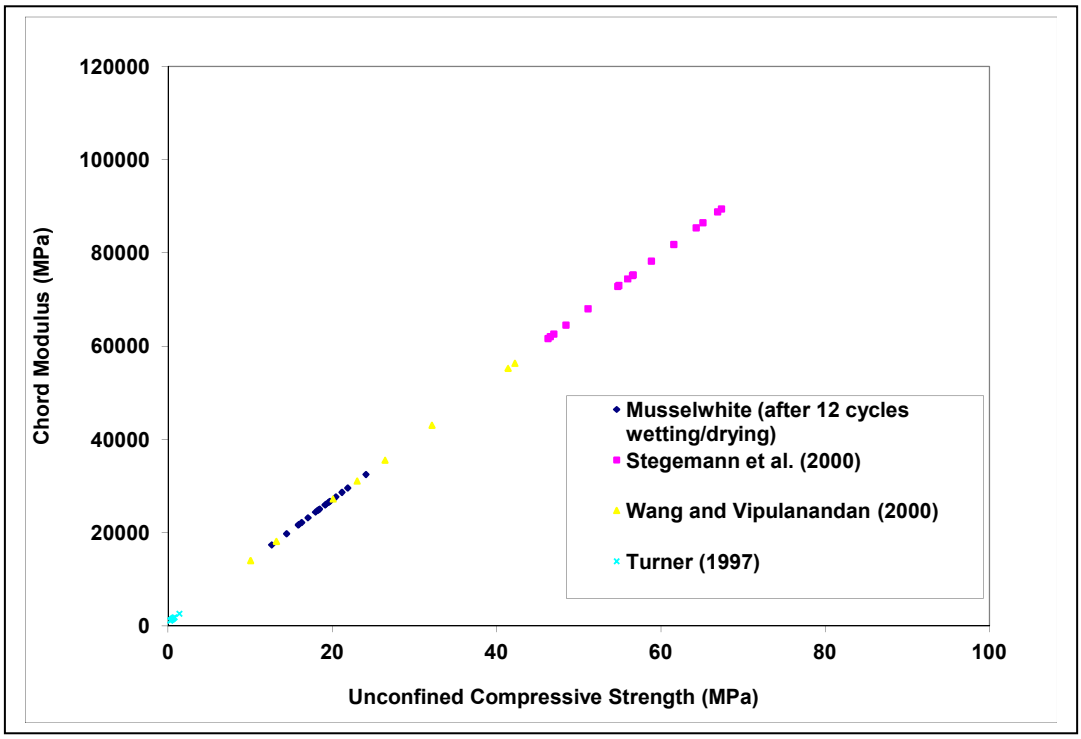


Figure 4-18 Chord modulus vs. unconfined compressive strength for Musselwhite tailings matrices after 12 cycles of wetting and drying with other values from the literature

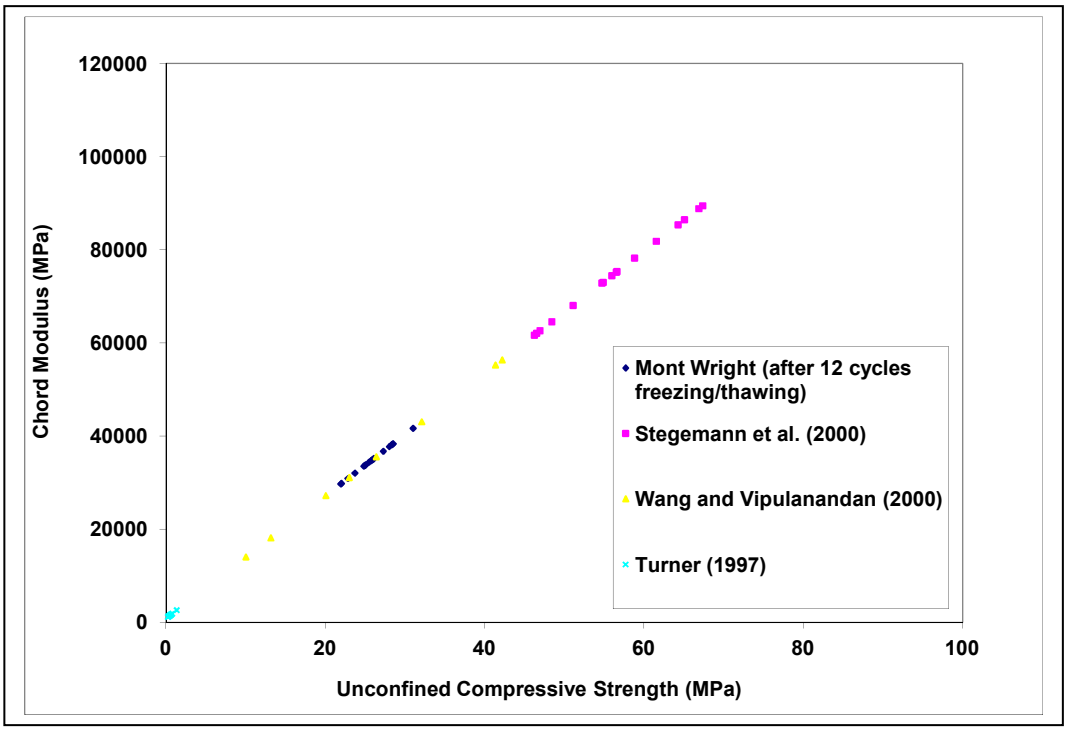


Figure 4-19 Chord modulus vs unconfined compressive strength for Mont Wright tailings matrices after 12 cycles of freezing and thawing with other values from the literature

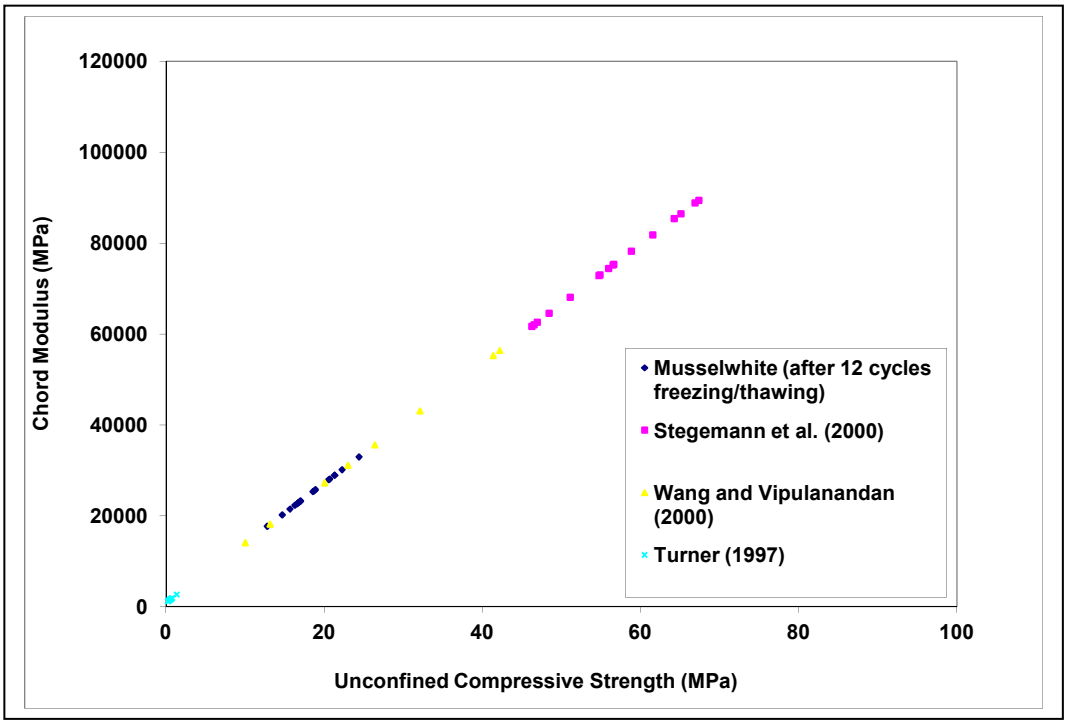


Figure 4-20 Chord modulus vs. unconfined compressive strength for Musselwhite tailings matrices after 12 cycles of freezing and thawing with other values from the literature

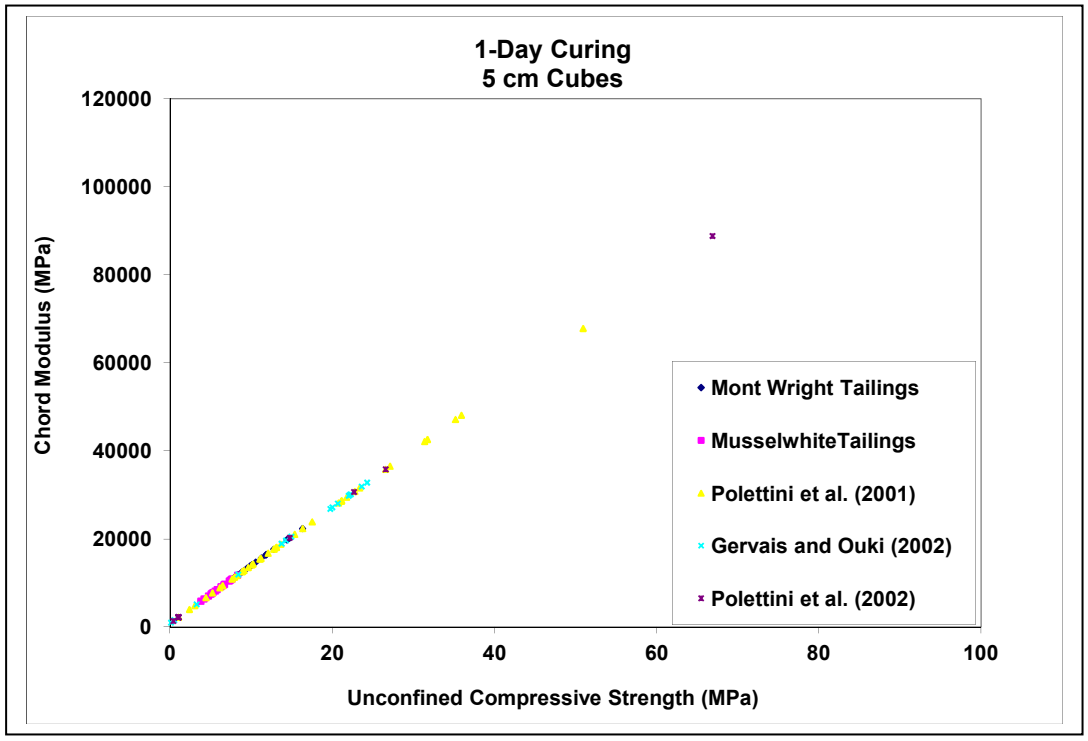


Figure 4-21 Chord modulus vs. unconfined compressive strength for Mont Wright and Musselwhite tailings matrices after 1 day curing with other values from the literature

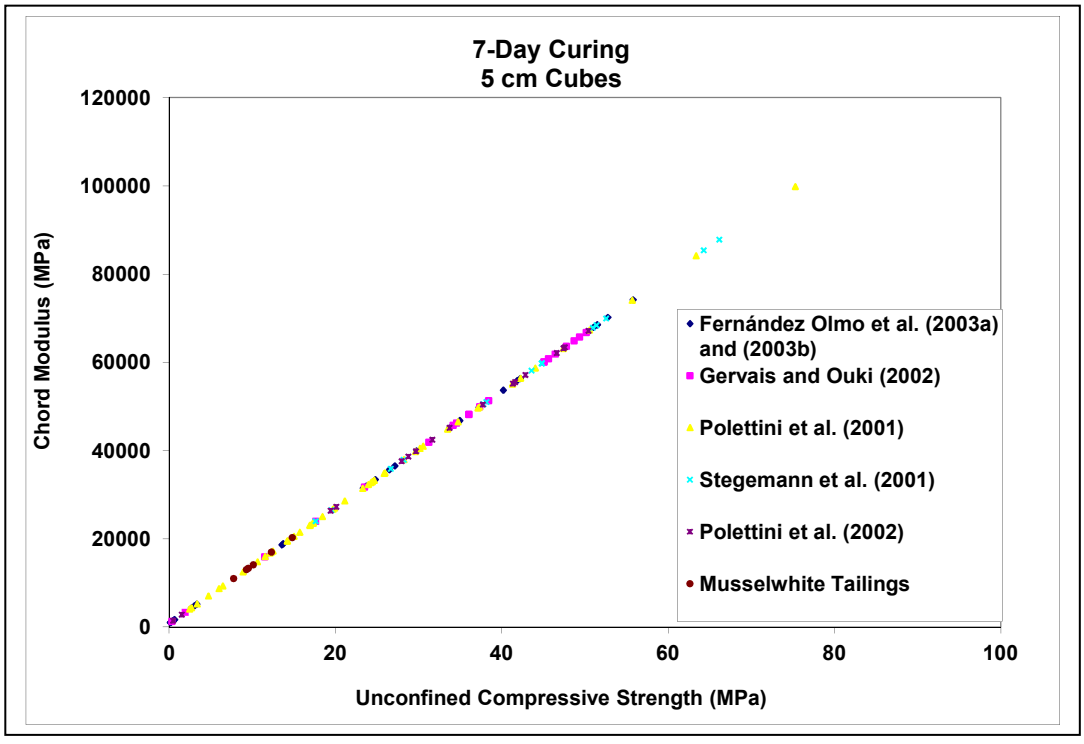


Figure 4-22 Chord modulus vs. unconfined compressive strength for Musselwhite tailings matrices after 7 day curing with other values from the literature

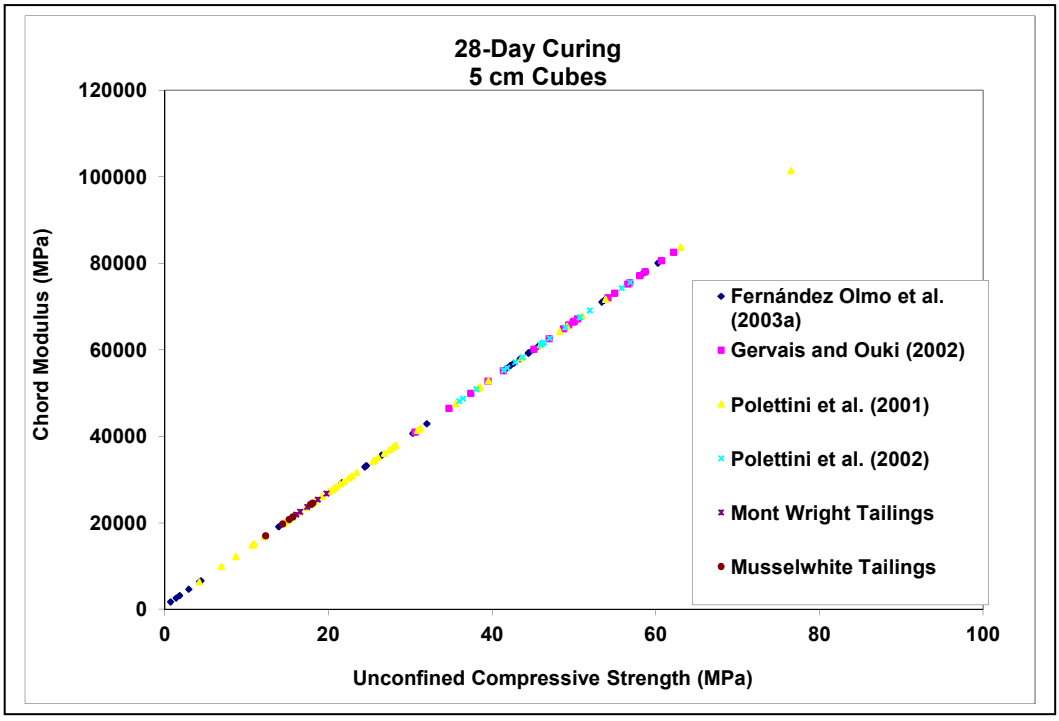


Figure 4-23 Chord modulus vs. unconfined compressive strength for Mont Wright and Musselwhite tailings matrices after 28 day curing with other values from the literature

For the 5 *cm* cube tests, it is seen in Figures4-5, 4-6, 4-7 and 4-8 that Mont Wright tests had higher strength values than Musselwhite for the 1 *day* cured cubes, these values however became closer for the 28 *days* curing period. The same observation can be seen for the fly ash and slag mixed specimens, where Mont Wright specimens had higher values than the same mixtures of Musselwhite for the 1 *day* curing period. This difference in strength values between the two tailings matrices might be due to the Musselwhite content of Zn and Pb. As water and cement are added toMusselwhite tailings, zinc hydroxide anions form and compete effectively with other anions for available adsorption sites to form a low permeability membrane coating cement particles(Asavapisit et al. 1997). Several researchers reported that Zn retarded the setting

through formation of this low permeability membrane of calcium hydroxyzincate ($CaZn_2(OH)_6.H_2O$) around the cement particles (Asavapisit et al. 1997; Li et al. 2001; Gervais and Ouki 2002; FernándezOlmo et al. 2003a), and Pb acted as a retardant agent by inhibiting the cement hydration, Salihoglu et al. (2007). The formation of a membrane around cement particles with the precipitation of calcium hydroxyzincate ($CaZn_2(OH)_6.H_2O$) can prevent water and ion transport needed for cement hydration. The retardation or suppression of cement hydration adversely affects the strength development. In the presence of lead, hydration occurs at a much slower rate than the reference (Thevenin and Pera, 1999), and the UCS values decrease. Coating of tri-calcium silicate $3CaO.SiO_2$ (C_3S) of cement with Pb or complexation of Pb (Gervais and Ouki, 2002; Thevenin and Pera, 1999; Palomo and Palacios, 2003) might have retarded the cement hydration.

The initial water content of the samples might have also been contributing to the decrease in the UCS values. The initial water content of 30.15% for Musselwhite plus the water/cement ratio of 0.3 might have been excessive for cement hydration especially as the Calsifrit ratio of the samples increased. Visual observation during the experiments revealed that increasing the amount of Calsifrit increased the fluidity of the mixes. Conner (1990) reported that as the water/cement ratio increased, the percentage of larger pores and therefore the permeability of the product increased. This may also explain the noticed decrease in strength between Musselwhite and Mont Wright matrices, Table 4-14.

Predictive equation

In the modeling of engineered systems a recurrent problem may arise that where the dependent variable or variables (Y) are known to depend on a combination of independent variables (x_1, x_2, ….., x_k) with no physical mechanism describing this dependence is available. According to conventional statistical practice such dependence can be best described using multiple regression analysis. Such analysis can be based on fitting either linear or polynomial equations, as described by Rumsey (2007).

Multiple regression models can be used both to generate predictions based on previously unseen data and to explain observational relationships. In multiple regression the general form of the model is:

$$Y = \beta_o + \beta_1 X_1 + \beta_2 X_2 + \ldots\ldots + \beta_k X_k + \varepsilon \quad \text{----------(4-13)}$$

Where β_o is the Y- intercept, β_i(where i=1 to k) is the true slope of the regression surface in the $X_i$direction, and ε is an error term. The regression equation can be readily solved using the method of least squares, which generates estimates for each of the beta (β) coefficients. These estimates are denoted by b_i.

The UCS data for the freezing/thawing and wetting/drying experiments were fitted using design expert®, which is a commercial software package used for the design of these

experiments. Multiple regression analysis was used to fit the data together with the purpose of finding a relevant predictive equation that fits the binders used with the UCS.

Table 4-15 shows the data being fitted into this model. Binder combinations are used as the factors of regression and the UCS is the response. Percentages of Calsifrit/binder, fly ash/binder, slag/binder, cement/binder, binder/tailings and type of weathering were the factors used in developing this model. However, only the factors indicating the percentages of Calsifrit/binder, slag/binder and binder/tailings were found statistically relevant.

Table 4-15 Data used in the regression model

Run	**Calsifrit/binder (%)**	**Fly ash/binder (%)**	**Slag/binder (%)**	**Cement/binder (%)**	**Binder/ tailings (%)**	**Weathering**	**UCS (*MPa*)**
1	0.00	0.00	0.00	100.00	50.00	1	24.43
2	10.00	0.00	0.00	90.00	50.00	1	25.65
3	25.00	0.00	0.00	75.00	50.00	1	25.24
4	0.00	25.00	0.00	75.00	50.00	1	22.97
5	10.00	15.00	0.00	75.00	50.00	1	26.22
6	20.00	5.00	0.00	75.00	50.00	1	25.83
7	0.00	0.00	25.00	75.00	50.00	1	28.62
8	10.00	0.00	15.00	75.00	50.00	1	27.9
9	20.00	0.00	5.00	75.00	50.00	1	29.47
10	0.00	0.00	0.00	100.00	37.50	1	13.5
11	10.00	0.00	0.00	90.00	37.50	1	16.41
12	25.00	0.00	0.00	75.00	37.50	1	19.42
13	0.00	25.00	0.00	75.00	37.50	1	16.91
14	10.00	15.00	0.00	75.00	37.50	1	19.3
15	20.00	5.00	0.00	75.00	37.50	1	21.79
16	0.00	0.00	25.00	75.00	37.50	1	19.79
17	10.00	0.00	15.00	75.00	37.50	1	20.09
18	20.00	0.00	5.00	75.00	37.50	1	17.5
19	0.00	0.00	0.00	100.00	50.00	2	23.51

20	10.00	0.00	0.00	90.00	50.00	2	26.31
21	25.00	0.00	0.00	75.00	50.00	2	27.1
22	20.00	5.00	0.00	75.00	50.00	2	28.00
23	0.00	0.00	25.00	75.00	50.00	2	25.7
24	10.00	0.00	15.00	75.00	50.00	2	27.83
25	20.00	0.00	5.00	75.00	50.00	2	24.88
26	0.00	0.00	0.00	100.00	37.50	2	16.57
27	10.00	0.00	0.00	90.00	37.50	2	20.94
28	25.00	0.00	0.00	75.00	37.50	2	23.33
29	0.00	25.00	0.00	75.00	37.50	2	17.78
30	10.00	15.00	0.00	75.00	37.50	2	16.71
31	20.00	5.00	0.00	75.00	37.50	2	16.65
32	0.00	0.00	25.00	75.00	37.50	2	16.35
33	10.00	0.00	15.00	75.00	37.50	2	16.74
34	20.00	0.00	5.00	75.00	37.50	2	20.11

The final equation describing the dependency of the UCS on the binder combinations was found to be the following:

$$UCS = - 6.856 + 0.14\ A + 0.092\ C + 0.6198\ E \quad \text{----------(4-14)}$$

Where:

A: Calsifrit/binder (%),

C: slag/binder (%),

E: binder/tailings (%).

It was found during the analysis that the effect of cement could not be associated directly with the UCS, however its effect was evident in conjunction with the other combinations. Table 4-16 shows the Analysis of Variance (ANOVA) for this model. The P values found are much less than the required 0.05 indicating that significance level has been reached.

Table 4-16Analysis of Variance for the regression model

Source	**Sum of Squares**	**df**	**Mean Square**	**F value**	**P value**
Model	580.11	3	193.37	58.93	< 0.0001
A-A	47.27	1	47.27	14.41	0.0007
C-C	18.76	1	18.76	5.72	0.0233
E-E	506.2	1	506.2	154.25	< 0.0001
Residual	98.45	30	3.28		

Equation (4-14) is especially applicable to the range of binder combinations and the 2 types of weathering used. If different binder combinations are required, beyond the range used above, it is best to base a new equation on the new binder combinations. This is done in order to retain a high numerical accuracy and closer proximity to experimental results.

The mathematical difference between the observed (y_i) and model predicted (y'_i) values of the dependent variable, for a given set of independent variables, specifically y_i-y'_i, is

the prediction error or residual.In order for a mathematical model to be valid, four assumptions regarding the model residuals must be met (Baxter et al. 2004):

- The residuals are normally distributed
- The residuals have a mean of zero
- The residuals have a constant variance
- The residuals are independent

One key assumption that must be fulfilled in order for a model to be valid is that the residuals are normally distributed. In order for this assumption to hold true a plot of the residuals should look like a random sample from a normal bell-shaped distribution. Histograms and normal scores plots are useful tools for detecting the normalcy of the residuals. In the case of the latter, a straight line plot is expected. Another assumption made in the development of the model is that all data points are collected with equal precision. In order for this assumption to hold true, a plot of the model residuals against model predictions will be free of obvious trends. The third assumption concerning the residuals relates to their independence with respect to time and the values of the model variables. For this assumption to be true, plots of residuals in time order, as well as plots of residuals against the values of the dependentand independent variables, should be free

of obvious trends. The fourth and final assumption suggests that a good model will have residuals that are scattered around a mean of zero (Baxter et al. 2004).

The first condition to meet is that the residuals must have a normal distribution with mean zero. Figure 4-24 shows how well the residuals match a normal distribution.The residuals fall on a straight line, which means the normality condition is met.

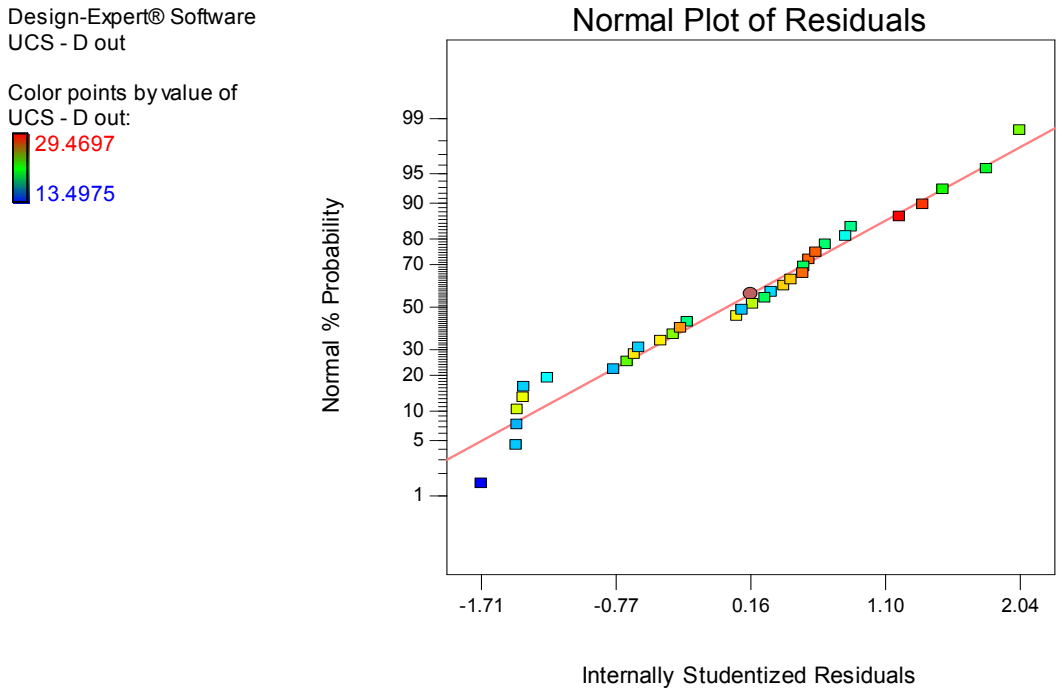


Figure 4-24 Normal probability plot of residuals

Figure 4-25 shows the residuals versus predicted UCS values. The horizontal line going across that plot is at zero as a marker. Residuals are centered around zero in a way that has no predictable pattern, with the same amount of variability around the horizontal line that crosses at zero as we move from left to right. This plot shows more residuals

hovering around zero, where the middle lump would be on a standard normal distribution plot, and fewer and fewer of the residuals as we go away from zero.Due to all of this, this plot confirms a normal distribution. The residuals average out at the zero line, which indicates that the mean of zero condition holds for the residuals in question.

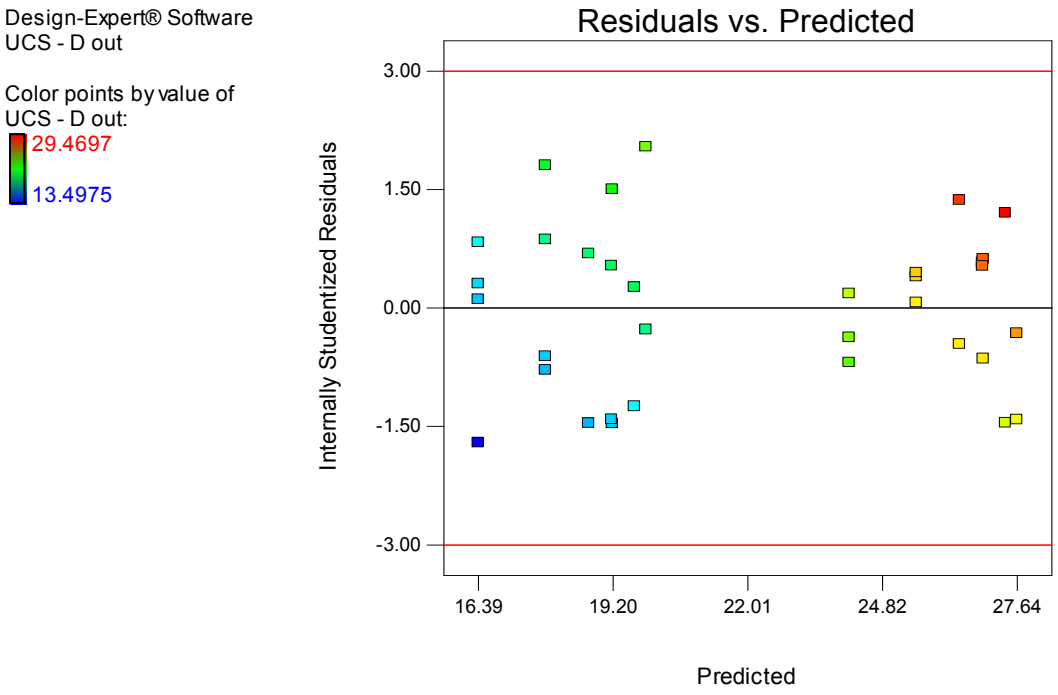


Figure 4-25 Residuals vs. predicted UCS values

Figure 4-25 shows no change in the amount of spread (variability)in the residuals around the horizontal zero line as we move from left to right or from right to left. This clearly indicates that condition three; the residuals have constant variance, has been met for this data set.

Figure 4-26 that shows the residuals plotted by observation order indicates no discernable pattern in these data. This indicates that condition four; the residuals are independent, has been met.

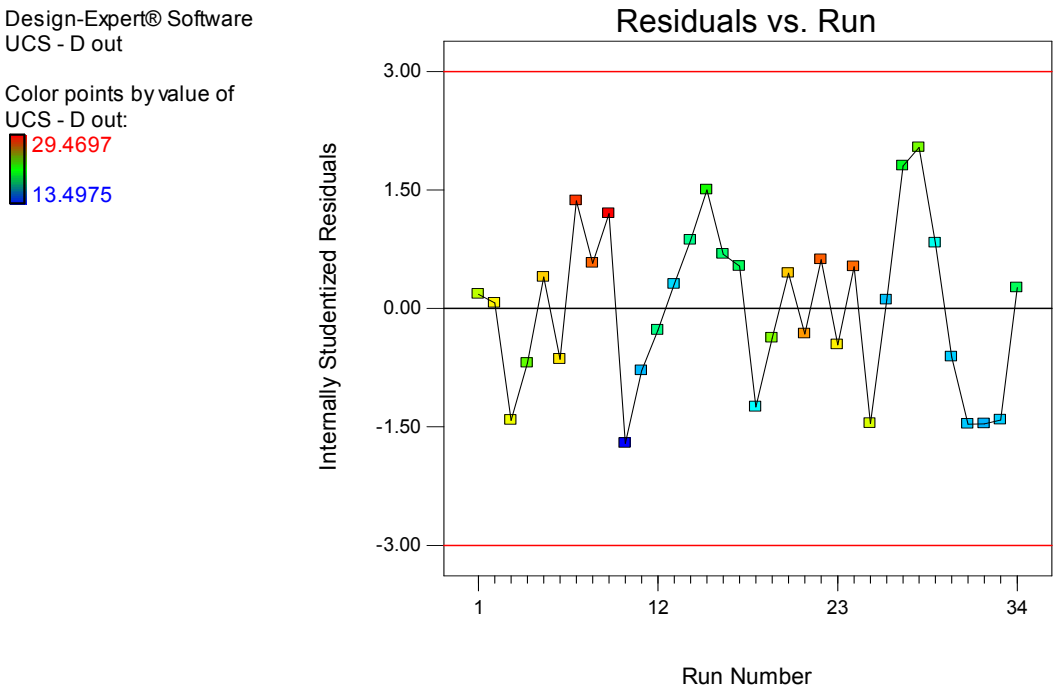


Figure 4-26 Residuals plotted by observation order

Plots of the model residuals against each of the independent variables were free of trends as can be seen in Figures 4-27 to 4-32.

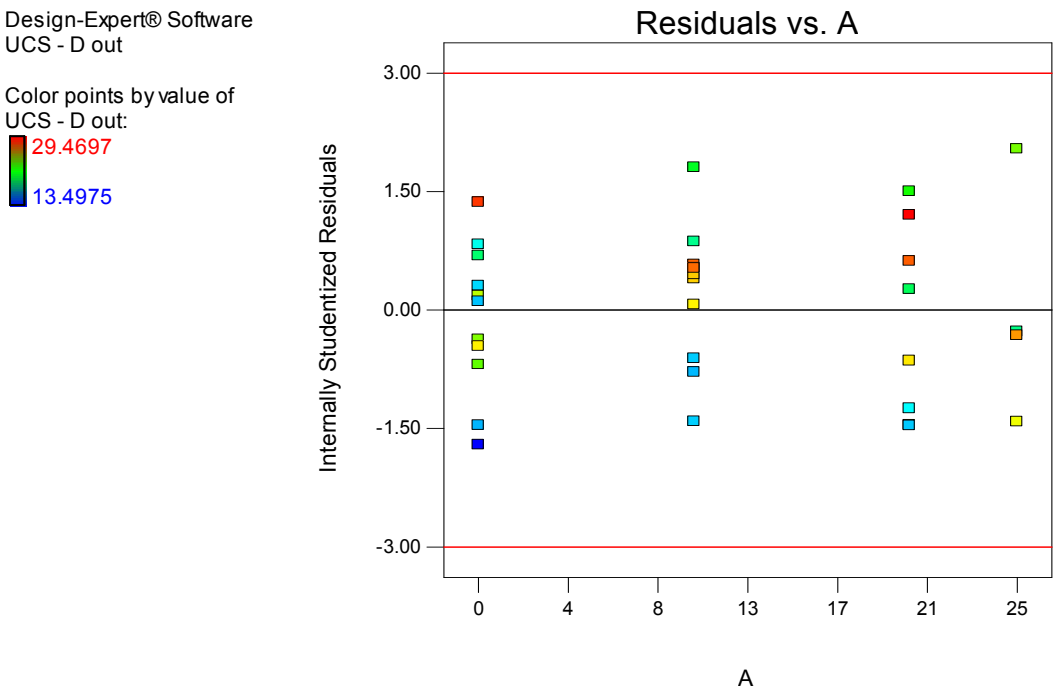


Figure 4-27 Residuals vs. Calsifrit/binder ratio

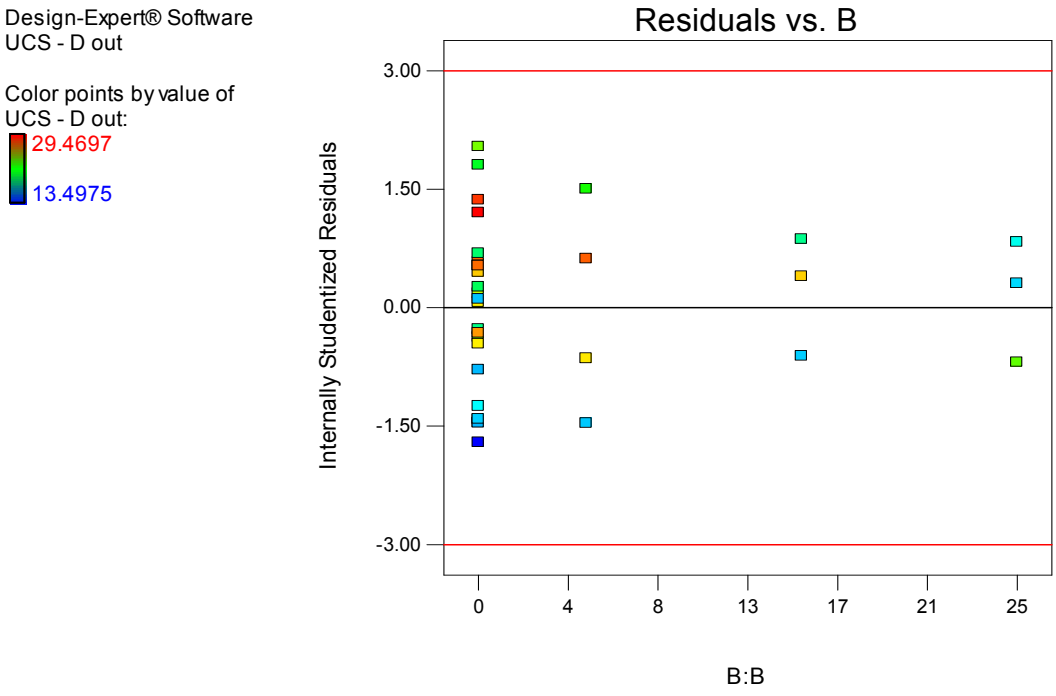


Figure 4-28 Residuals vs. fly ash/binder ratio

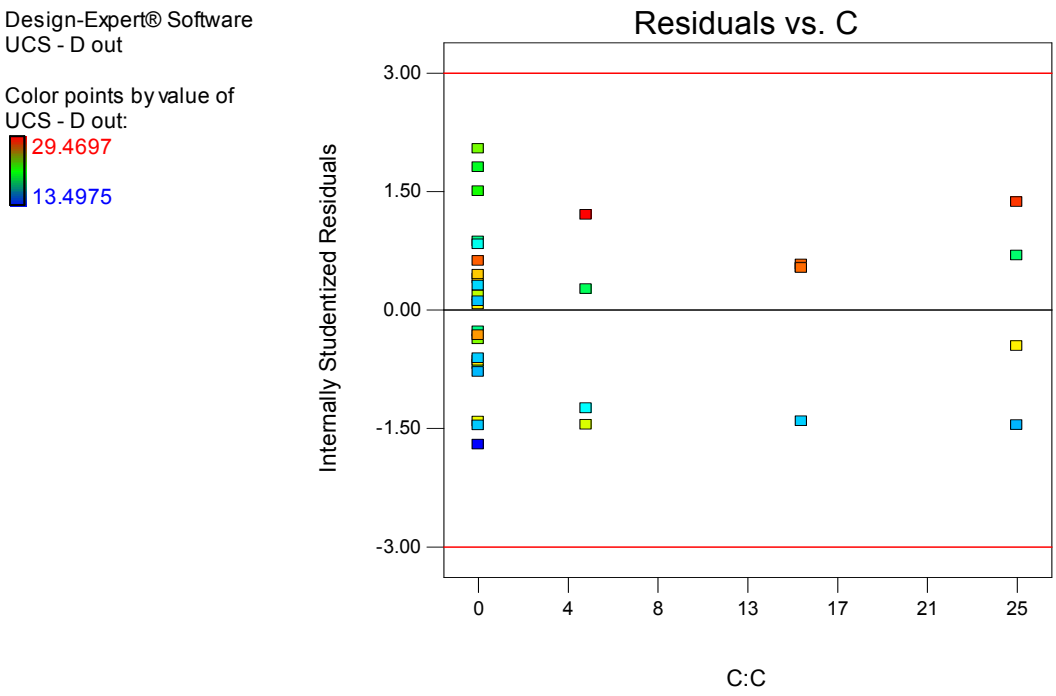


Figure 4-29 Residuals vs. slag/binder ratio

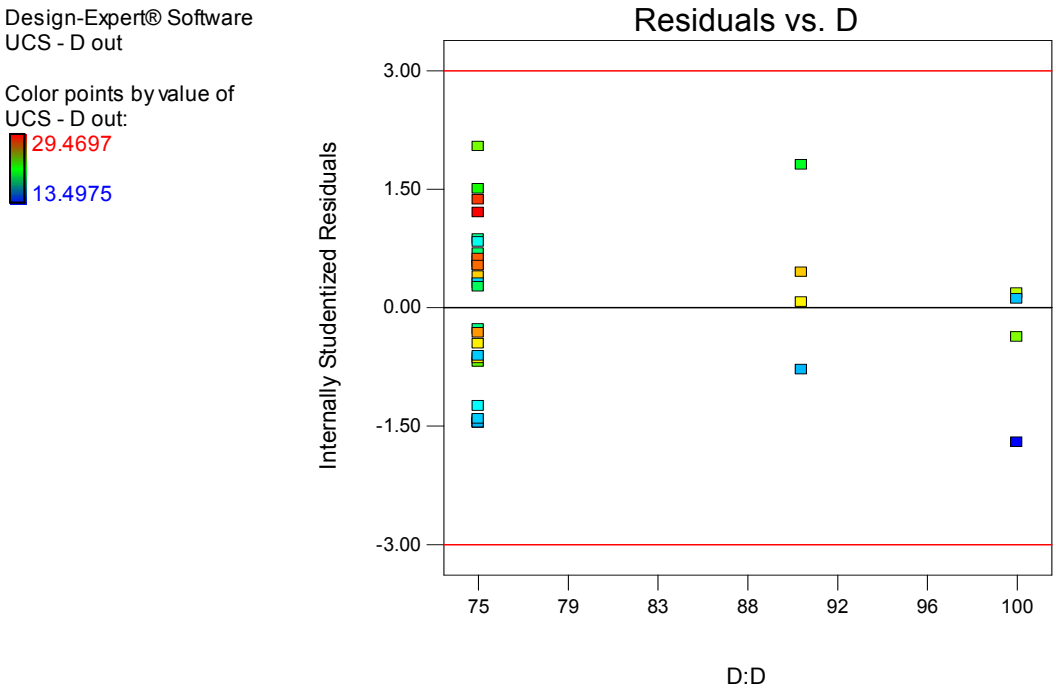


Figure 4-30 Residuals vs. cement/binder ratio

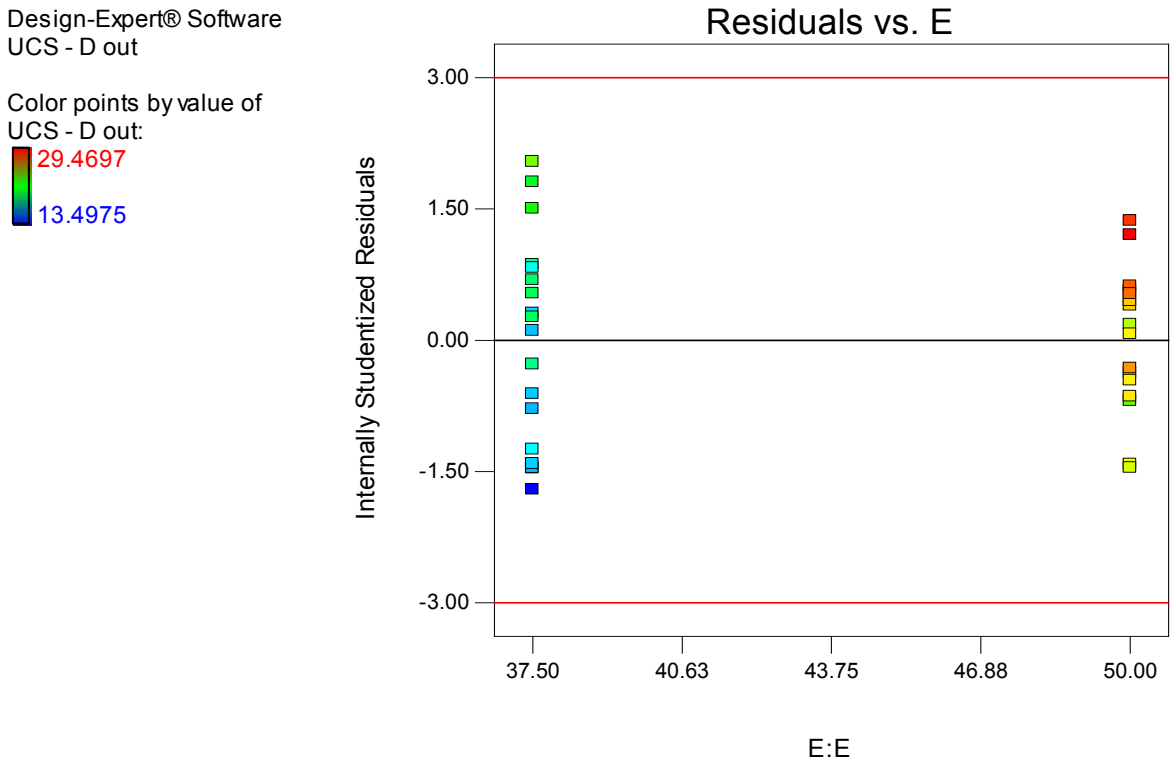


Figure 4-31 Residuals vs. binder/tailings ratio

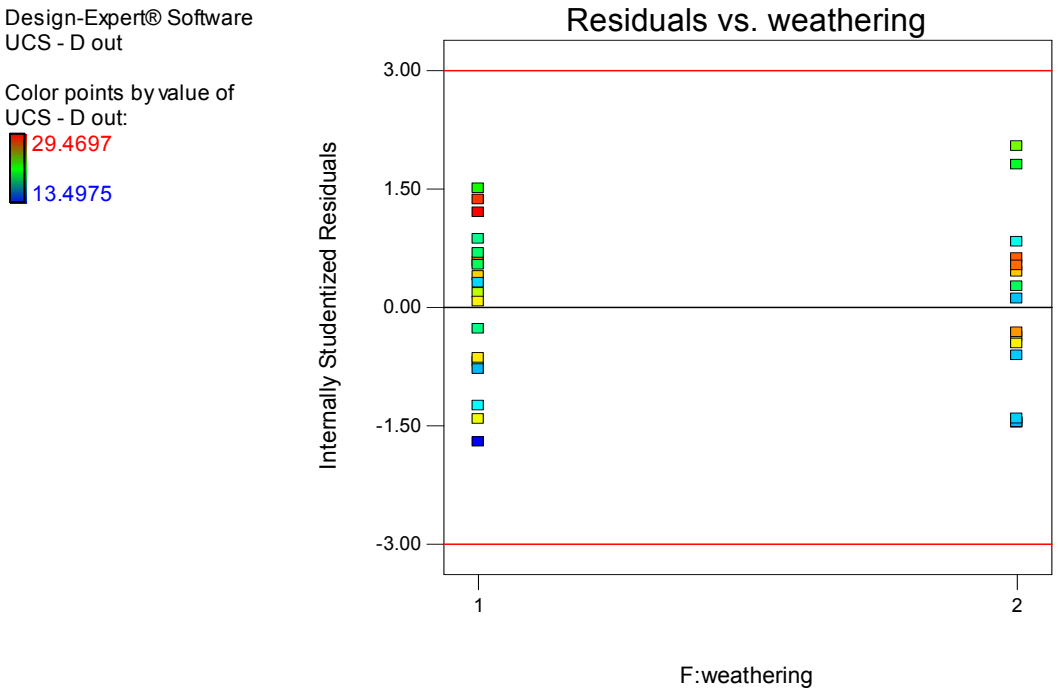


Figure 4-32 Residuals vs. weathering type; 1 for wetting/drying and 2 for freezing/thawing

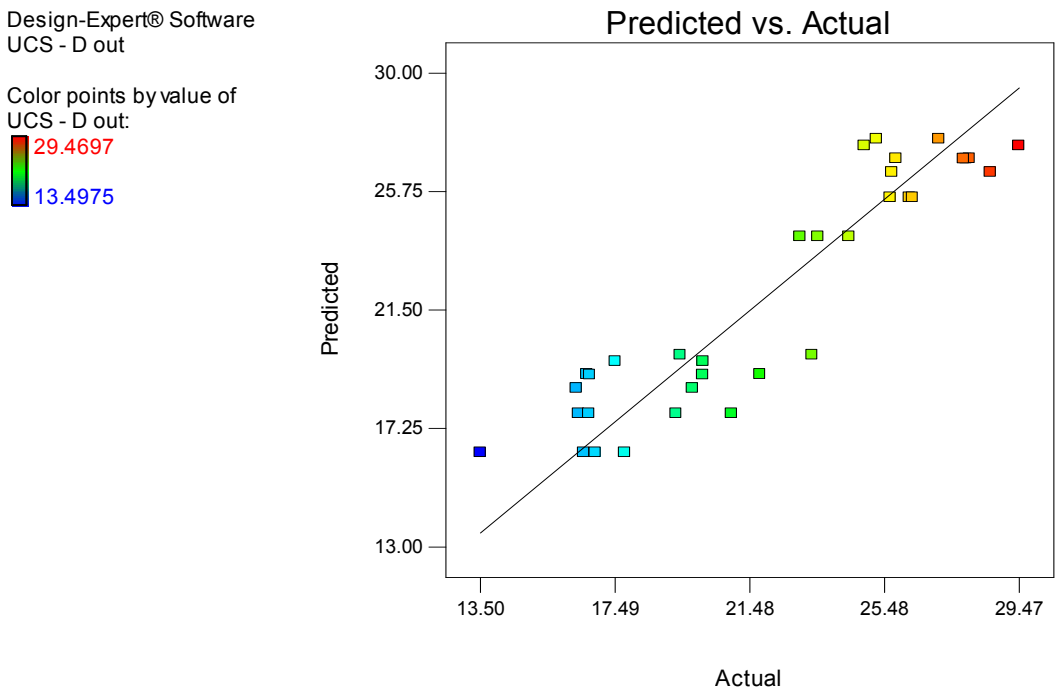


Figure 4-33 Predicted vs. actual UCS values

Predicted versus actual values as seen in Figure 4-33 show a good correlation between the two. All the above indicates that the multiple regression model suggested here is accurate and can be used for the prediction of UCS based on the variables used.

Cost Benefit Analysis (CBA)

A cost benefit analysis example on the use of these tailing matrices in construction is given here. This cost benefit analysis is implemented to ascertain the benefits involved in the use of these tailing matrices in the road construction industry. This analysis is primarily based on the environmental sustainability that the use of these matrices carries in comparison with the traditional, soil based methods.

The qualitative environmental benefits gained when tailings are used in construction versus their traditional storage method in ponds, werediscussed earlier in Chapter 2. This section attempts to quantitatively outline some of the benefits involved in the use of tailings matrices in construction, specifically road construction.

In the current study, it is thought that the following approach is the best for this study and will be adopted accordingly:

1) Identification of the materials,

2) Identification of costs and benefits,

3) Quantification of costs and benefits,

4) Summary and conclusions.

The two types of materials involved that make up the cost benefit analysis comparison are the following:

1) The tailing matrices including OPC, Calsifrit, fly ash and slag, previously identified in Chapter 3 then tested in Chapter 4,

2) Granular material normally used as sub-base in roads (Foth and Van Dyke 2003).

It is assumed in this analysis that two typical road sub-base sections are built that incorporate either one of these materials.The following is the identification of costs and benefits:

Benefits:

1) Tailings dusting is fully eliminated using this new technique. This dusting is considered a serious environmental incident prompting Hudson's Bay Mining and Smelting Company Limited to issue a control guidelines report for their tailings impoundment systems (HBMS 2007),

2) Failure of tailings dams with all the imposed hazards due to the traditional storage of tailings in ponds. A study by Rico et al. (2008) reports 147 worldwide tailings dam disasters. Once the solidified/stabilized tailings matrices are in-place, this hazard is eliminated, thereby saving lives and property that are otherwise prone to tailings dangers,

3) Heavy metal seepage to the ground water table is eliminated or much reduced by this approach of stabilizing/solidifying the tailings material,

4) Freeing up the space previously occupied by the tailings dam,

5) Freeing up resources and personnel connected with the supervision on the tailings dam,

6) Using up industrial wastes such as fly ash and slag,

7) Using Calsifrit, identified in Chapter 2.

Costs:

1) Trucks to transfer to the site the tailings from local mines in the area,

2) Machinery that spread and compact the tailings in the base course layer to the required thickness and density,

3) Specialized workforce,

4) Importation of OPC.

Cost benefit analysis attempts to put a monetary value on all items included in the analysis. However, this section is more focused on a conceptual quantitative analysis related to the environmental sustainability of the tailings matrices and their effects. Hence a procedure explained by Shutt (1997) and used and cited by Amjad (2005) will be used here whereby symbols will be given based on the importance priority of each cost or benefit item.

Amjad (2005) performed a cost benefit analysis for three potential alternatives for a road construction project through an industrial-residential town. He was able to demonstrate through the use of a combined conceptual-monetary approach the financial implication of each alternative and the best solution for the problem.

When analyzing environmental issues, it is often difficult to assign explicit monetary values to the gains associated with sustainability schemes. Therefore, based on that, and in order to quantify the un-quantifiable, it is decided here to assign an abbreviation for each of the following basic values: M for monetary benefits, S for increased safety, I for intangibles and T for time saving. These abbreviations denote annual flow of cost or benefit. Also, a plus sign (+) will be added to denote benefits and a minus (-) to denote costs. A duplication of the sign will indicate added importance.

Tables 4-17 and 4-18 show the benefits and costs associated with this approach for the tailings and sand matrices respectively.

Table 4-17 Cost Benefit Analysis of the use of tailings matrices

Factor	Benefits	Costs	Balance
Tailings dusting	S+ I+		S+ I+
Tailings dam failures	S++ M+ T+		S++ M+ T+
Heavy metal seepage to the ground water	S+ M+ T+		S+ M+ T+
Freeing up the space previously occupied by the tailings dam	M+		M+
Freeing up resources and personnel connected with the supervision on the tailings dam	M+ T+ I+		M+ T+ I+
Fly ash	S+	M-	S+ M-
Calsifrit	S+	M-	S+ M-
Slag	S+	M-	S+ M-
Trucks		M-	M-
Machinery		M-	M-
Workforce		M-	M-
On Site Delivery	M+		M+
Importing OPC		M-	M-

M: monetary benefits, S: increased safety, T: time saving, I: intangibles

Table 4-18 Cost Benefit Analysis of the use of sand matrices

Factor	Benefits	Costs	Balance
Fly ash	S+	M-	S+ M-
Calsifrit	S+	M-	S+ M-
Slag	S+	M-	S+ M-
Trucks		M-	M-
Machinery		M-	M-
Workforce		M-	M-
On Site Delivery		M-	M-
Importing OPC		M-	M-

M: monetary benefits, S: increased safety, T: time saving, I: intangibles

It is seen through Table 4-19 that the financial implications of using the sand matrices are higher than those of the tailings matrices. It can also be seen that financial benefits in terms of environmental sustainability are much higher for the tailings matrices. Consequently, it can be concluded that tailings matrices present a higher sustainability priority material for use in road sub-bases.

Table 4-19 Summary of findings and order of preference

Factor	**Tailings Matrices**	**Sand Matrices**	**Order of Preference**	
			Tailings Matrices	**Sand Matrices**
Tailings dusting	S+ I+	-	1	2
Tailings dam failures	S++ M+ T+	-	1	2
Heavy metal seepage to the ground water	S+ M+ T+	-	1	2
Freeing up the space previously occupied by the tailings dam	M+	-	1	2
Freeing up resources and personnel connected with the supervision on the tailings dam	M+ T+ I+	-	1	2
Fly ash	S+ M-	S+ M-	1	1
Calsifrit	S+ M-	S+ M-	1	1
Slag	S+ M-	S+ M-	1	1
Trucks	M-	M-	1	1
Machinery	M-	M-	1	1
Workforce	M-	M-	1	1
On Site Delivery	M+	M-	1	2
Importing OPC	M-	M-	1	1
Preference Totals			**13**	**19**
Overall Order of Preference			**1**	**2**

M: monetary benefits, S: increased safety, T: time saving, I: intangibles

The first part of this study (Phase 1) involved furnishing the experimental evidence confirming the usefulness of the mine tailings matrices for use as road construction materials and the application of the matrices to roads. It is now desired to optimize the whole material selection process to eliminate or extensively reduce the huge amount of experimental preparation work involved. Hence, it is necessary to devise a computational approach capable of performing the same task will much less experimental effort. This computational phase will be denoted as: Phase 2.

Chapter 5

PHASE 2: COMPUTATIONAL MODELING TOOL FOR THE ASSESSMENT OF TAILINGS BINDER MATRICES FOR CONSTRUCTION PURPOSES

5.1 Introduction

Phase 1 showed experimentally the feasibility of the application of tailingsbindermatrices as construction material.The objective in phase 2 is to design a program capable ofsupporting the engineer's decision with regards to the application of the binder materials in construction. It would substitute for extensive laboratory experiments required for designing various combinations of tailings-binder matrices. Thus, this chapter describes the computational tool used for the prediction and assessment of tailings binder matrices for construction purposes.

It was shown in Section 2.5.3 that the DEM can be applied for the assessment of the tailings binder matrix. Therefore, an algorithm and computational program was generated, using the DEM, to build a universal tool for tailings binder matrices assessment and usefulness for construction purposes. The tool is verified using a series of data generated during the experimental phase of this research.

Phase 2 consists of three stages. In Stage 1, development of an adequate code simulating matrix behavior is written; in Stage 2 verification of the computational tool will be performed using UCS experimental data and in stage 3 verification of the computational tool with regards to freezing/thawing experimental results will be attempted. This chapter, as it describes phase 2, shows the various steps and procedures applied to building and using this computational tool called Tailings-DEM™.

5.2 Stage 1: Program developmentfor solidified tailings materials

It is assumed that the solidified tailings cylinders are composed of several elements (particles) that are coherent and interact with each other once an external load is applied. That load being either a compression (uniaxial) load or a thermal load resulting from the change in temperatures in the freeze/thaw weathering test. Each solidified tailings element interacts with its neighboring elements at the contact points and the sum of the contact forces and moments is thus added. As it was justified in Section 2.5.3 the DEM is the most suitable method for solidified tailings.

5.2.1 Hypothesis for modeling stresses and strains

It is assumed that each solidified tailings element is composed of a tailings particle and its surrounding binder components. This is justified based on the small sizes of the tailings particles and their inter-particle void space. This is exemplified in Figure 5.1. All subsequent calculations will treat the two element components as one for numerical simplicity.

The coefficient of normal stiffness (k_n) is assumed equal to the slope of the linear portion of the stress versus strain diagrams of the uniaxial compression tests. It is further assumed for numerical simplicity and uniformity, that the coefficient of shear stiffness (k_s) is equal to the coefficient of normal stiffness for each tailings-binder matrix under investigation in this study:

$$k_n = k_s = k \quad \text{----------(5.1)}$$

This leads to the assumption that when forces at a certain contact point of any element become negative, separation between this element and its neighbor ensues and the following values will take effect:

When the normal force at the contact point (c) between 2 adjacent particles (F_n^c) becomes negative, a no-tension status takes place and normal and shear forces at that contact point cease to exist:

$F_n^c = 0$

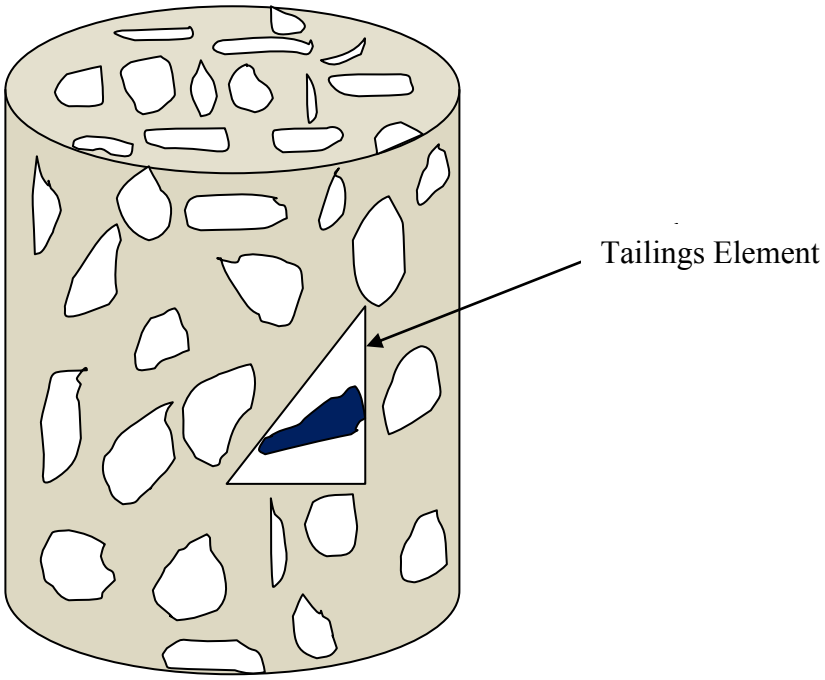


Figure 5.1 Tailings element shown within a cylindrical solidified tailings sample

$F_s^c = 0$

$D_n^c = 0$

$D_s^c = 0$

Where:

F_n^c= normal force at contact point c

F_s^c= shear force at contact point c

D_n^c= normal dashpot force at contact point c

D_s^c= shear dashpot force at contact point c

Physically, this is shown as cracks in the cylindrical specimen under uniaxial testing.

The mass of each particle is used in calculating the force of gravity that is added to the sum of external forces. The mass of a particle for a particular tailings matrix is calculated based on the specific gravity of the tailings particles and the surrounding binder materials. Since ordinary Portland cement composes most of the binder material its specific gravity represents that of the binder and is added to that of the tailings. The average size of the particles for a particular matrix is found from the particle size distribution curve of tailings, added to the particle a film of surrounding binder.

5.2.2Modeling of thermal stresses

The same assumptions above apply when modeling thermal stresses.Thermal stresses resulting from the change in temperatures of the freezing and thawing test ASTM D 4842 (1996),are calculated according to the following equation (Pytel and Singer 1987):

$$\sigma = E\,\alpha\,\Delta t \quad \text{----------(5.2)}$$

Where:

σ = thermal stress (*MPa*)

E = modulus of elasticity (*MPa*)

α = coefficient of thermal expansion (*m/m ºC*)

The United States Federal Highway Administration employs a value range of (18-20) x 10^{-6}/ *ºC* for the coefficient of thermal expansion of saturated cement pastes (FHWA 2011). The average of this range, 19 x 10^{-6}/ *ºC*, is used in calculating the thermal stresses in this study.

The modulus of elasticity (E) for each tailing matrix is assumed equal to the slope of the linear portion of the stress versus strain diagram of that matrix.

5.2.3 DEM stresses and strains

The method is illustrated here in 2 dimensions by using 2 adjacent blocks: i and j. Each block, as explained above, represents a tailings particle and its surrounding binder materials. The following equations, as reported by Cundall (1974),are used to compute the incremental displacements and rotations shown in Figure 5.2:

$$\Delta u^{c}_{y} = \Delta u^{i}_{y} - \Delta u^{j}_{y} + \Delta\theta^{i}(x^{c} - x^{i}) - \Delta\theta^{j}(x^{c} - x^{j}) \quad \text{----------(5.3 a)}$$

$$\Delta u^{c}_{x} = \Delta u^{i}_{x} - \Delta u^{j}_{x} + \Delta\theta^{i}(y^{c} - y^{i}) - \Delta\theta^{j}(y^{c} - y^{j}) \quad \text{----------(5.3 b)}$$

Where:

(x^{i},y^{i}) = global coordinates of block i centroid

(x^{j},y^{j}) = global coordinates of block j centroid

(x^{c},y^{c}) = global coordinates of contact point c

Δu^{c}_{y}= incremental y displacement at the contact point c

Δu^{c}_{x}= incremental x displacement at the contact point c

Δu^{i}_{y}= incremental y displacement at block i

Δu^j_y= incremental y displacement at block j

Δu^i_x= incremental x displacement at block i

Δu^j_x= incremental x displacement at block j

$\Delta\theta^i$= incremental rotation at block i

$\Delta\theta^j$= incremental rotation at block j

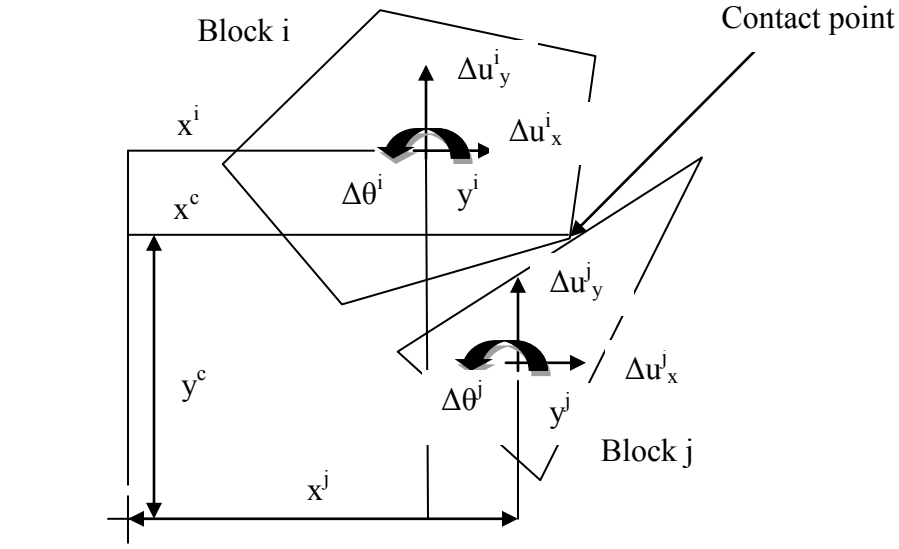


Figure 5.2 Graphical description of the incremental displacements and rotations of two adjacent blocks (reproduced after Cundall 1974)

Subsequently, the relative incremental normal and shear displacements (of i relative to j) are computed as follows:

$\Delta u^{ci}_s = \Delta u^c_x \cos\omega + \Delta u^c_y \sin\omega$ ----------(5.4 a)

$\Delta u^{ci}_n = \Delta u^c_y \cos\omega - \Delta u^c_x \sin\omega$ ----------(5.4 b)

Where:

Δu^{ci}_s= relative incremental shear displacement in block i

Δu^{ci}_n= relative incremental normal displacement in block i

ω = angle with the global x axis of the contact point plane

Then, normal and shear forces at the contact point c are calculated as follows:

F_n^c replaced by $F_n^c - \Delta u^c_n k_n$----------(5.5 a)

F_s^c replaced by $F_s^c + \Delta u^c_s k_s$----------(5.5 b)

Dashpot forces, which act in the same manner as F forces are calculated as follows:

$D_n^c = -\Delta u^c_n K_n$----------(5.6 a)

$D_s^c = \Delta u^c_s K_s$----------(5.6 b)

Where:

K_n= normal dashpot constant

K_s = shear dashpot constant

In this study, dashpot constants are assumed equal andare calculated according to the following equations:

$K_n = K_s = \kappa/\Delta t$----------(5.7 a)

Where: $\kappa = 2\sqrt{m.k}$ ----------(5.7 b)

m: mass of particle

k = stiffness coefficient

Contact forces found from the equations above are then resolved into their global X-Y directions as follows:

$F_y^{cj} = (F_s^c + D_s^c) \sin\omega - (F_n^c + D_n^c) \cos\omega$ ----------(5.8 a)

$F_x^{cj} = (F_s^c + D_s^c) \cos\omega + (F_n^c + D_n^c) \sin\omega$ ----------(5.8 b)

$F_y^{ci} = - F_y^{cj}$----------(5.8 c)

$F_x^{ci} = - F_x^{cj}$----------(5.8 d)

Where:

F_y^{cj}= y component of the force at block j part of contact point c

F_x^{cj}= x component of the force at block j part of contact point c

F_y^{ci}= y component of the force at block i part of contact point c

F_x^{ci}= x component of the force at block i part of contact point c

So total forces and moments acting on block iare found from the sum of the contributions of each contact. They are found as follows:

$F^i_{xsum} = \sum_c F^{ci}_x$----------(5.9 a)

$F^i_{ysum} = \sum_c F^{ci}_y + F^i_{ygrav}$----------(5.9 b)

$M^i_{sum} = \sum_c \{F_y^{ci}(x^c - x^i) - F_x^{ci}(y^c - y^i)\}$ ----------(5.9 c)

Where:

$\sum_c$means the summation is carried over all contact points for block i

F^i_{ygrav}= the gravity force acting on block i

F^i_{xsum}= sum of forces in the global x direction acting on block i

F^i_{ysum}= sum of forces in the global y direction acting on block i

M^i_{sum}= sum of moments acting on block i

Velocities are derived from forces by numerical integration. They are found as follows:

u'^i_y will be replaced by $u'^i_y + (F^i_{ysum}\Delta t)/m^i$ ----------(5.10 a)

u'^i_x will be replaced by $u'^i_x + (F^i_{xsum}\Delta t)/m^i$ ----------(5.10 b)

θ'^i will be replaced by $\theta'^i + (M^i_{sum}\Delta t)/ I^i$ ----------(5.10 c)

Where:

Δt = time increment

m^i= mass of block i

I^i= moment of inertia of block i

u'^i_y= y component of the linear velocity of block i

u'^i_x= x component of the linear velocity of block i

θ'^{i}= angular velocity of block i

Then, incremental and absolute displacements are derived from velocities by numerical integration as follows:

$\Delta u^{i}_{y} = u'^{i}_{y}.\Delta t$ ----------(5.11 a)

$\Delta u^{i}_{x} = u'^{i}_{x}.\Delta t$ ----------(5.11 b)

$\Delta \theta^{i} = \theta'^{i}.\Delta t$----------(5.11 c)

$u_y^i$is replaced by $u_y^i + \Delta u^i_y$ ----------(5.11 d)

$u_x^i$is replaced by $u_x^i + \Delta u^i_x$----------(5.11 e)

$\theta^i$is replaced by $\theta^i + \Delta\theta^i$----------(5.11 f)

The above equations are derived for block i. Similar equations are used for block j.

The time increment used should satisfy the following condition:

$\Delta t < 2 \times \sqrt{m/k}$ ----------(5.12)

Cundall (1974) showed that this condition gives numerical stability in terms of stable oscillatory for the previous equations.

5.3 Computer program (Tailings-DEM™)

As such, a computer program is written in the C++ language to analyze the solidified tailings matrices subjected to unconfined and thermal stresses. The computer program was written using C++ with the compiler Code Blocks.

The program starts by asking the user to enter the stiffness coefficient in *N/m* and the matrix mass factor in *kg*. After that the user is asked to enter the time increment in seconds as shown in the program flowchart with its various parts, Figure5.3. After entering the time increment, the program checks for its compatibility. If it is compatible, the running of the program starts, otherwise, the user is asked to re-enter the time increment.

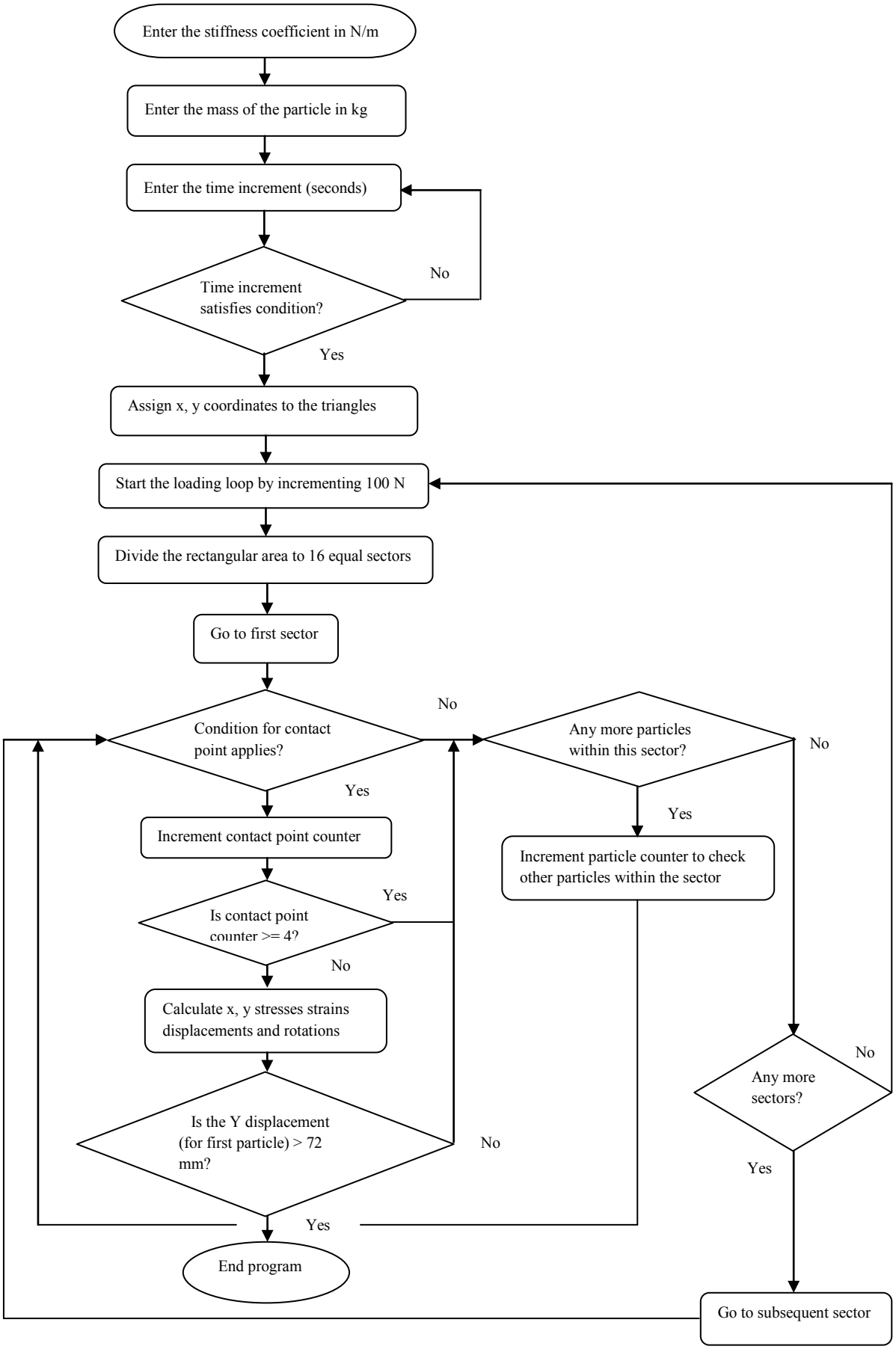


Figure 5.3 Flowchart showing the various steps within Tailings-DEM™

After entering the time increment, the program allocates x, y coordinates for the triangles that represent and compose the particles that occupy the cylindrical specimens of the tailings matrices. Subsequently, the program is executed by applying the load in Newtons and increasing it until failure takes place. Load is assumed to equally apply at the geometrical centroid on all particles within the rectangular area. In this instance failure is defined as the first upper particle reaching 7.2 *mm* of displacement. This number is the result of the particle reaching the bottom of the cylinder (74 *mm*) minus the assumed particle height, which is 2.12 *mm*. The assumptions for particle shape and dimensions are discussed in Section 5.3.1 below. The number was scaled by 10 to account for the correlation with experimental values.

The program divides the rectangular area of the specimen into 16 equal parts to facilitate finding the particles (triangles) that come into contact with each other. Each part is (11 x 18.5), (x, y) *mm*. In this case, a code snippet runs to find the particles in contact at each of the 16 parts of the rectangle. In each of these 16 parts, particles are assumed to be in contact once the following condition is satisfied:

$$(x_1 - x_i)^2 + (y_1 - y_i)^2 \leq 50\ mm \quad \text{----------(5.13)}$$

Where:

x_1, y_1 are the Cartesian coordinates of the particle under consideration, and

x_i, y_i are the coordinates of the particle to be checked for contact proximity

This 50 *mm* is the minimum distance at which credible results were obtained. Earlier attempts to run the program with smaller distances did not give any meaningful results.

5.3.1 Assumptions

Friedel and Murray (2010) who did monotonic and cyclic simple shear tests on undisturbed tailings samples found that the pre-cyclic shear strength of the tailings was relatively high when compared to other natural materials of similar grain size. They concluded that this was a result of the angular nature of the tailings particles and lack of clay minerals.

Paterson et al. (2004) performed a study to investigate the hydraulic transport considerations of copper tailings at the Southern Peru Copper Corporation. They foundusing an electron micrograph that coarse and fine tailings particles had similar particle shapes.

Vermeulen et al. (2002) who did research on the properties of South African gold tailingsfound that coarser tailings sands had angular and sub-rounded bulky but flattened particles. The finer slimes consisted mostly of thin and plate-like particles characteristic of clay particles.

It was shown previously in Section 4.1 that both Mont Wright and Musselwhite tailings consist primarily of sand as they are classified into poorly-graded and silty sands respectively. Additionally, from the above it seems that the closest approximation for a tailings particle is an angular sub-rounded shape. Based on that, it was decided in this study to useright isosceles triangles to model the shape characteristics of the tailings particles under consideration.

The following assumptions were included during the computing and the program execution parts to facilitate output and interpretation of results. These assumptions define the program limitations:

1) Based on the limited triangular geometry, it is assumed that each triangle will have a maximum of 4 contact points with its neighboring triangles. Hence when contact particle number 4 is found, the execution moves from the current loop to the next in sequence,

2) The compression force applied in the vertical direction is equally divided among all triangles even when changing location and orientation. It is applied at the centroid of these particles at all times,

3) Contact takes place between the tips of one triangle and either one of the adjacent triangle's3 boundaries,

4) The gravity force was excluded in these calculations,

5) These assumptions are used to model cylindrical specimens,

6) The contact force was assumed perpendicular on the triangle boundaries.

5.3.2 Layout of Tailings-DEM™

The program assumes the identity and shape of the particles involved in the interactions. These particles were chosen to be right isosceles triangles as explained above. In relation to the size of the assumed cylindrical specimens used (Chapter 3), size of these triangles was made 2.12 *mm* in height and 4.243 *mm* in base. Figures 5.4 and 5.5 show the numbering convention used in allocating the coordinates for the triangle heads and centroid and the sector numbering convention used, respectively. This procedure allows the allocation of a total of 112 particles.Appendix D shows the program code.

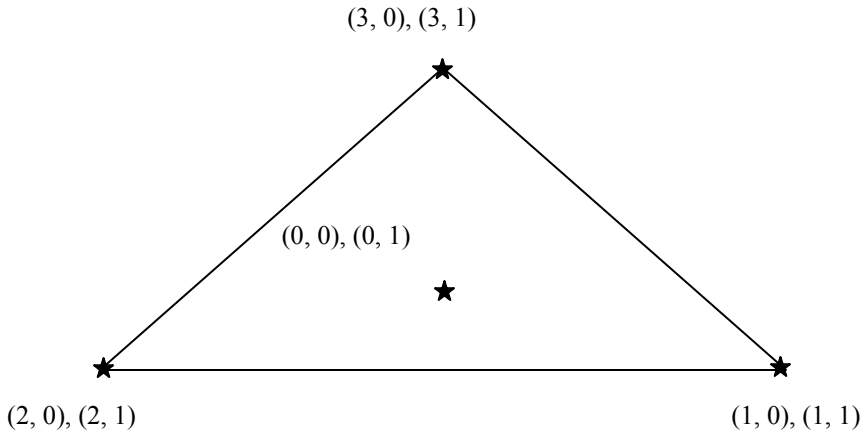


Figure 5.4Numbering convention used for the triangles' coordinates

1	**2**	**3**	**4**
5	**6**	**7**	**8**
9	**10**	**11**	**12**
13	**14**	**15**	**16**

Figure 5.5Numbering convention used for the sectors

5.4 Stage 2: Execution and verification of Tailings-DEM™modeling of unconfined compressive strength

The matrixmasscoefficient, used in the program, is computed through the following steps:

1) The unconfined compressive stress is found from the statistical predictive equation derived previously in Chapter 4. As shown in Chapter 4, the UCS is found by entering the following matrix mixture components (in percentage): Calsifrit/binder (A), slag/binder (C) and binder/tailings (E) into Equation (4-14) as follows:

   $$UCS = -6.856 + 0.14\ A + 0.092\ C + 0.6198\ E \quad \text{----------(4-14)}$$

   From the UCS above, it is possible to find the maximum compressive load (in *N*) for each of the matrix mixtures used in this equation.

2) Subsequently, the maximum load (in *N*) found experimentally (Chapter 4) for the same mixtures used in step (1) above, is used to find a correlation with a factor called here the matrix mass coefficient. This correlation is found by initially taking three values of the experimental maximum load and finding (by trial and error) their representative matrix mass coefficients (in *kg*) that give the closest match with the computational values. The three values represent the maximum and minimum data boundary shown in Figure 5.6 and an arbitrary middle point. Gradually other values for the matrix mass coefficient are

introduced into the program to find the closest match with the remaining experimental values. This procedure generated the following linear Equation(5.14) shown in Figure 5.6 relating the experimental maximum load and the matrix mass coefficient:

$$y = 2\,(10)^{9}x - 276.9 \quad \text{----------(5.14)}$$

Where: y is the experimental maximum load (in *N*) and x is the matrix mass coefficient (in *kg*).

The matrix mass coefficient is the major input used in the computer program to find the computational values including the computational maximum load for each tailings matrix mixture.

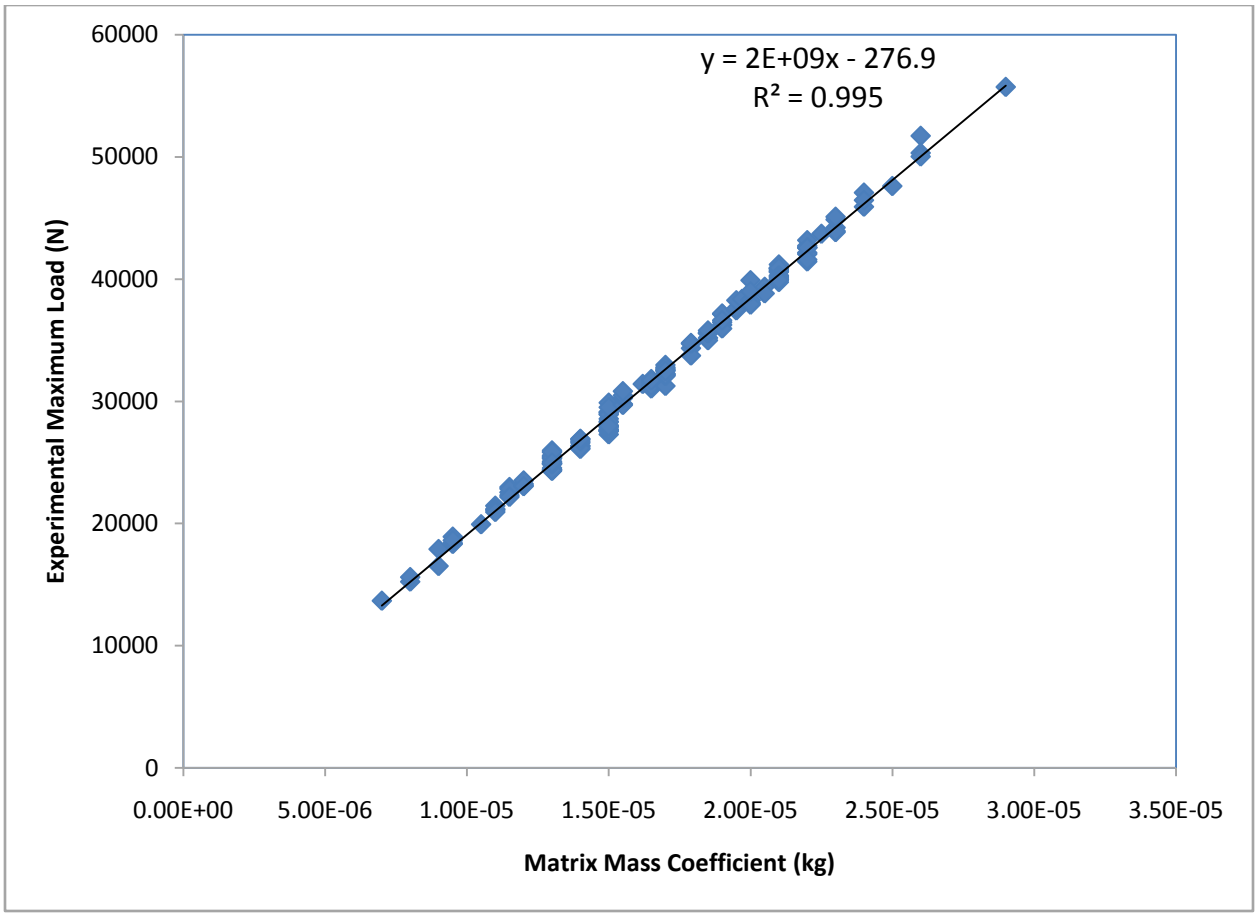


Figure 5.6 Correlation between the experimental maximum load and the matrix mass coefficient

3) The computational maximum load values were produced in step (2) as a byproduct of using the matrix mass coefficient. Hence, the same matrix mass coefficient now represents these new computational maximum load values and their corresponding experimental counterparts. Using regression techniques produced Equation (5.15) shown in Figure 5.7, that relates the newly found computational maximum loads and their respective matrix mass coefficients:

$$y = 2(10^9)\, x + 39.27 \quad \text{----------} \quad (5.15)$$

Where: y is the computational maximum load (in *N*) and x is the matrix mass coefficient (in *kg*).

4) Equation (5.15) is used to find the matrix mass coefficient when the statistical UCS value is found from Equation (4-14). This matrix mass coefficient is the major input used in the computer program to get the corresponding computational maximum load value.

5) Computational maximum load values found in step (4) are then compared with their experimental counterparts.

   Table 5.1 shows the values generated from the procedure explained in steps 1-5.

6) Figure 5.8 shows the correlation between values of the experimental (Chapter 4) and computational maximum loads found using the procedure explained above.

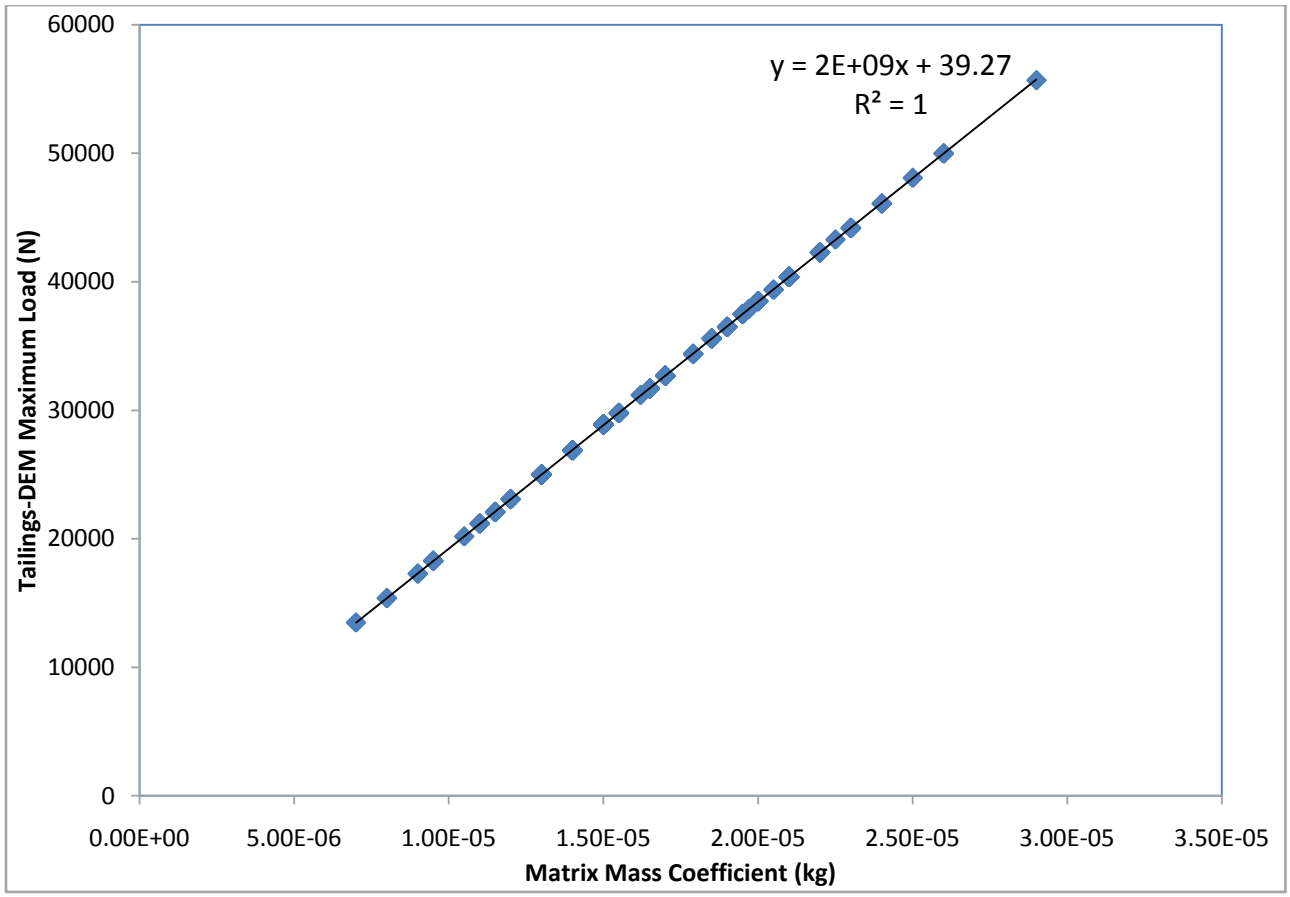


Figure 5.7 Correlation between the computational maximum load and the matrix mass coefficient

Subsequently, the user is asked to enter the time increment in seconds. This time increment is part of the original Equations (5.3) to (5.12). Equation (5.12) checks for its compatibility with the value $\sqrt{m/k}$. If the time increment is more than or equals the latter value, the user is asked to re-enter the time increment until compatibility is assured.

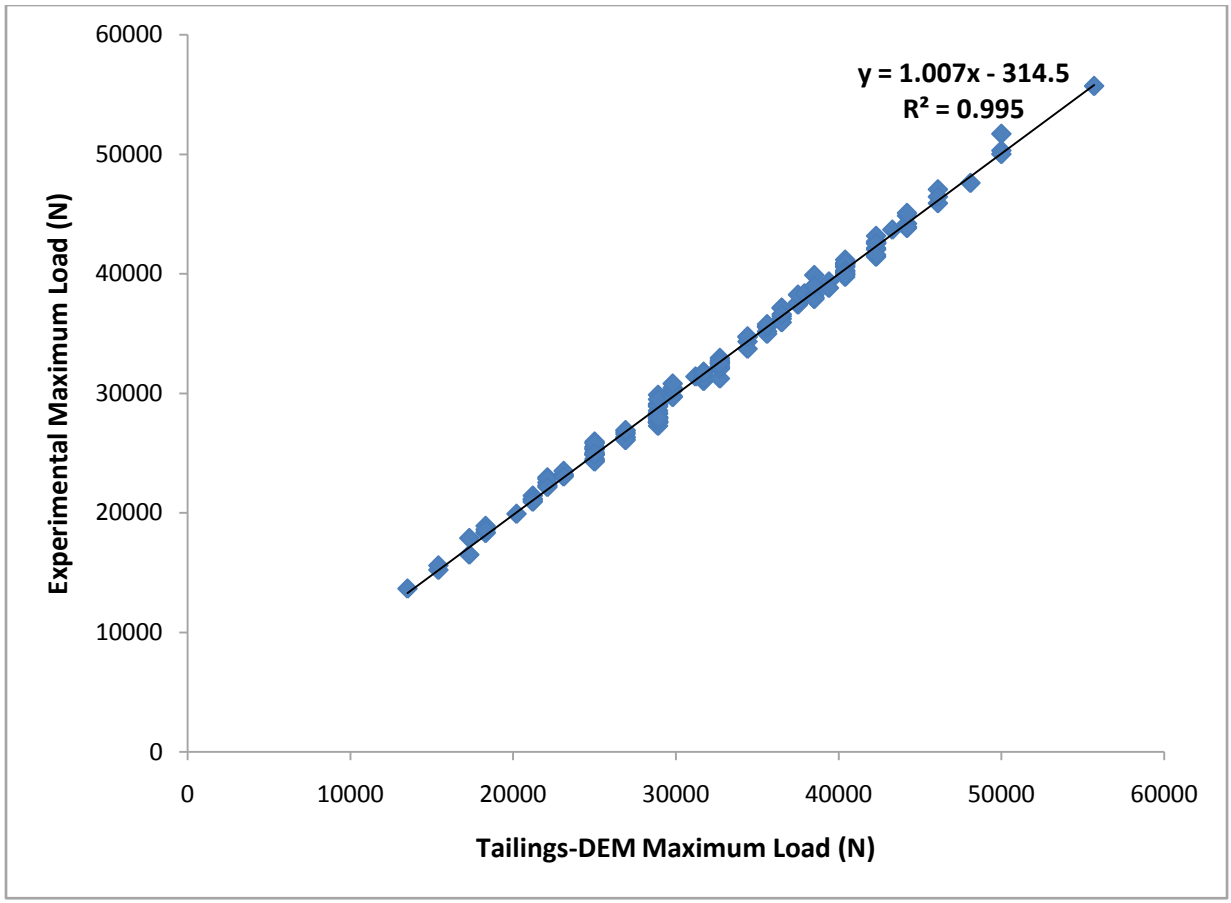


Figure 5.8Correlation between the experimental maximum loadvs. theTailings-DEM maximum load

Subsequent to the input operations, the triangular particles are arranged in a grid like fashion and dispersed uniformly throughout the rectangular area that represents the cylinder under compression.

Table 5.1Numerical procedure used in calculating the matrix mass coefficient

Stat. Eq. Max. load (N)	Weight (kg) from Eq. (5.15)	Max. Load from Program Run (N)	Difference (%)	Exp. Max Load (N)	Difference (%)
36696.49	1.84E-05	35200	4.08	37146.57	5.24
38825.24	1.94E-05	37300	3.93	39001.62	4.36
42018.35	2.1E-05	40400	3.85	38378.2	-5.27
36696.49	1.83E-05	35200	4.08	34926.59	-0.78
38825.24	1.94E-05	37300	3.93	39868.32	6.44
40953.98	2.05E-05	40400	1.35	39275.31	-2.86
40193.71	2.01E-05	38500	4.21	43517.59	11.53
40923.57	2.04E-05	38500	5.92	42422.81	9.25
41653.42	2.08E-05	40400	3.01	44810.04	9.84
24916.18	1.24E-05	23900	4.08	20527.17	-16.43
27044.92	1.35E-05	26000	3.86	24951.91	-4.20
30238.04	1.51E-05	29000	4.09	29528.71	1.79
24916.18	1.24E-05	23900	4.08	25712.18	7.05
27044.92	1.35E-05	26000	3.86	29346.25	11.40
29173.67	1.46E-05	28100	3.68	33132.37	15.19
28413.4	1.42E-05	27300	3.92	30091.31	9.28
29143.25	1.46E-05	28100	3.58	30547.47	8.0120059
29873.11	1.49E-05	28900	3.26	26609.29	-8.61
36696.49	1.83E-05	35200	4.08	35747.68	1.53
38825.24	1.94E-05	37300	3.93	40005.17	6.76
42018.35	2.1E-05	40400	3.85	41206.39	1.96
40953.98	2.05E-05	40400	1.35	42574.86	5.11
40193.71	2.01E-05	38500	4.21	39077.64	1.48
40923.57	2.04E-05	38500	5.92	42316.37	9.02
41653.42	2.08E-05	40400	3.01	37830.81	-6.79
24916.18	1.24E-05	23900	4.08	25195.2	5.14
27044.92	1.35E-05	26000	3.863653	31839.92	18.34
30238.04	1.51E-05	29000	4.09	35473.99	18.25
24916.18	1.24E-05	23900	4.08	27035.04	11.6
27044.92	1.35E-05	26000	3.86	25408.07	-2.33
29173.67	1.46E-05	28100	3.68	25316.84	-10.99
28413.4	1.42E-05	27300	3.92	24860.68	-9.81
29143.25	1.46E-05	28100	3.58	25453.69	-10.4
29873.11	1.49E-05	28900	3.26	30577.88	5.49

After the particles arrangement,the program loading loop starts with increments of 100 *N*. It is thought that this amount furnishes sufficient data points for graph plotting and data manipulation.

At the start of loading, the rectangular area gets divided into 16 sectors equal in area to facilitate contact detection among the particles. A counter for each sector stores the number of particles dispersed in it. Then execution moves to the first sector to check for contact detection among particles. The first sector particle is taken and checked for contact with other particles. Once contact is found for this particle, its contact meter is increasedby one up to a maximum of 4 contact points with 4 other particles. It is thought that the limited geometry of these triangles does not permit interaction with more than 4 particles without encountering serious computational errors.Equation (5.13) above shows the numerical condition that checks for contact among neighboring particles. Initial trials have shown that this is the minimum distance possible for reasonable results.

This contact is the mechanism by which stresses and strains are activated through the use of Equations (5.3) to (5.12). After checking the first particle for contact, the sector particle counter is increased and the procedure repeated for the remaining sector particles.

Then execution moves from the first sector to the second sector and the whole contact procedure is repeated again until all sectors are checked. After that, execution moves to the loading stage again and another increment of 100 *N* is applied. This continues until a displacement of 72 *mm* in

the vertical direction is reached. This displacement is scaled by 10 to account for the differences in the experimental and computational modeling and to account for the 2-D approximation. When this displacement is reached, the specimen failure is assumed to take place and the program halts.The program is capable of finding x, y stresses and strains for its member particles.

5.4.1 Computational and experimental results

It is necessary to verify the accuracy and reliability of the proposed computational model using laboratory-approved experimental results. Hence the UCS results of Chapter 4 were used in the verification and analysis of the computational data found using this program.

Cylindrical specimens of the Mont Wright and Musselwhite tailings matrices were subjected to wetting/drying and freezing/thawing weathering resistance tests as explained in Chapter 3.

After undergoing the weathering tests, thematrices specimens of both tailingswere tested under uniform conditions to find their strength in the UCS apparatus. Experimental results of this test were discussed in Chapter 4. Figures A-1 to A-18 show the results of UCS testing for these tailings-binder matrices after 12 cycles of wetting and drying. Figures A-19 to A-35 show the compression strength results for the same matrices after 12 cycles of freezing and thawing. All values shown are the average of 3 determinations.

Figures 5.9 to 5.25 show the UCS experimental and computational results comparison for Mont Wright and Musselwhite tailingsmatrices for the matrix binder combinations shown in Chapter 3. The experimental specimens had gone through the freezing/thawingweathering cycles before undergoing the UCS compression testing. Figures C-1 toC-16show the experimental and computational UCS results for both tailings for the same matrix binder combinations after being subjectedto the wetting/dryingweathering cycles.The following shows these matrix binder combinationsin groups of figures:

1) Figures 5.9and 5.17show the 100% OPC,

2) Figures5.10and 5.18show the 90% OPC, 10% Calsifrit,

3) Figures 5.11 and 5.19 show the 75% OPC with 25% Calsifrit,

4) Figures5.12and 5.20show the 75% OPC with 25% fly ash,

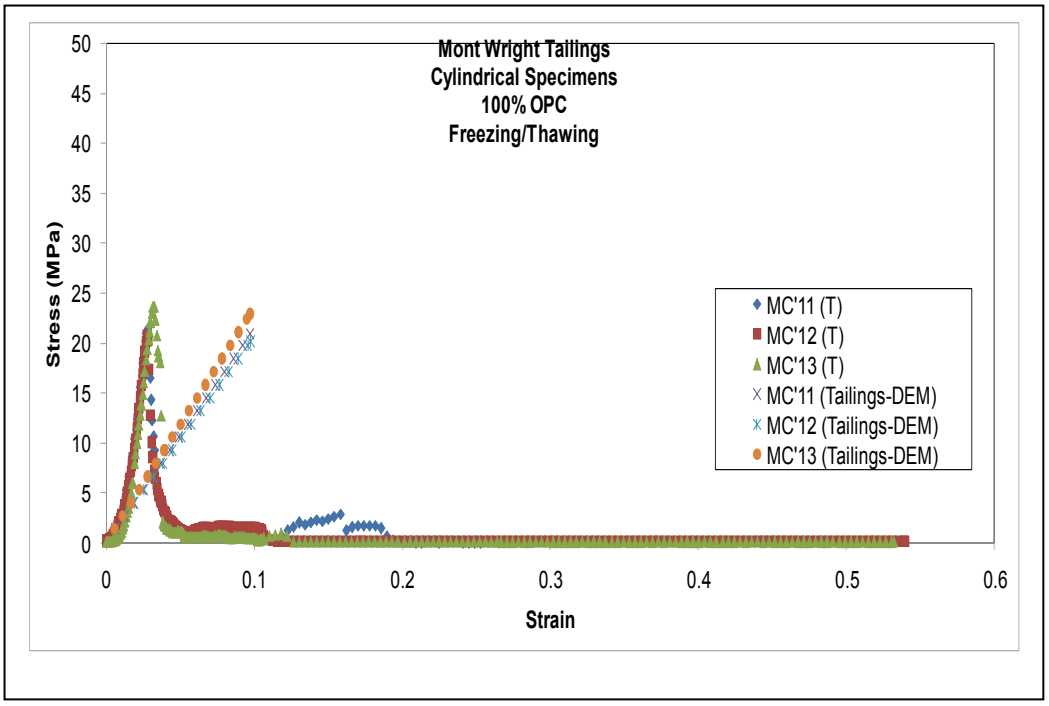
5) Figures5.13and 5.21show the 75% OPC, 15% fly ash and 10% Calsifrit,

6) Figure5.22shows the 75% OPC, 5% fly ash and 15% Calsifrit,

7) Figures5.14and 5.23show the 75% OPCand 25% slag,

8) Figures5.15and 5.24show the 75% OPC, 15% slag and 10% Calsifrit,

9) Figures 5.16and 5.25show the 75% OPC, 5% slag and 20% Calsifrit.



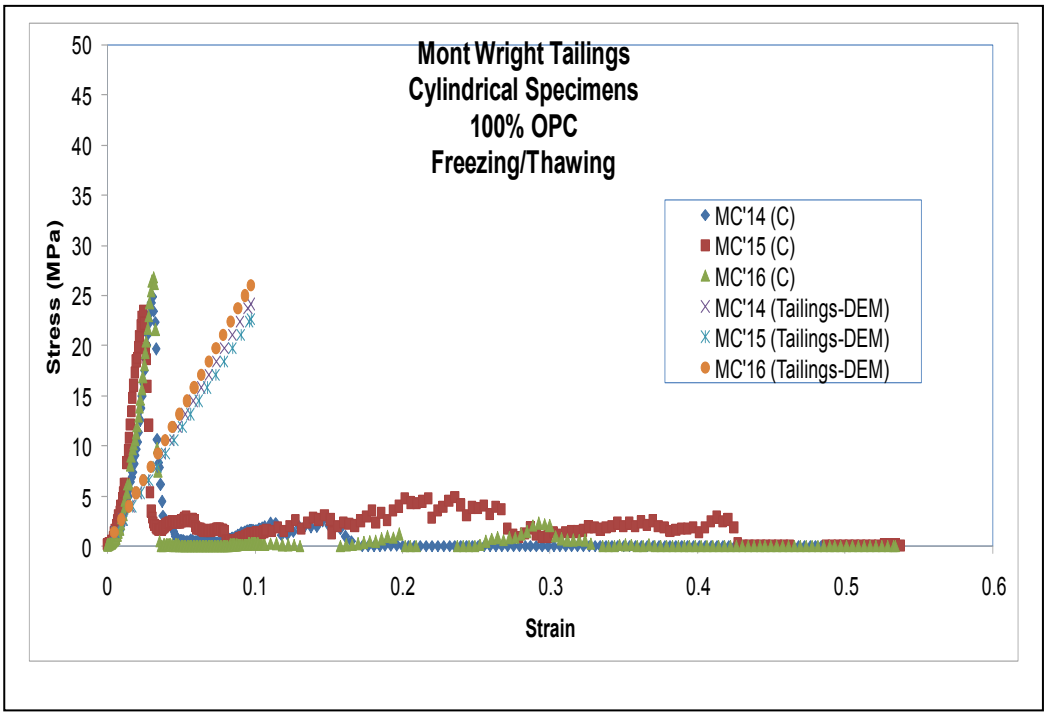
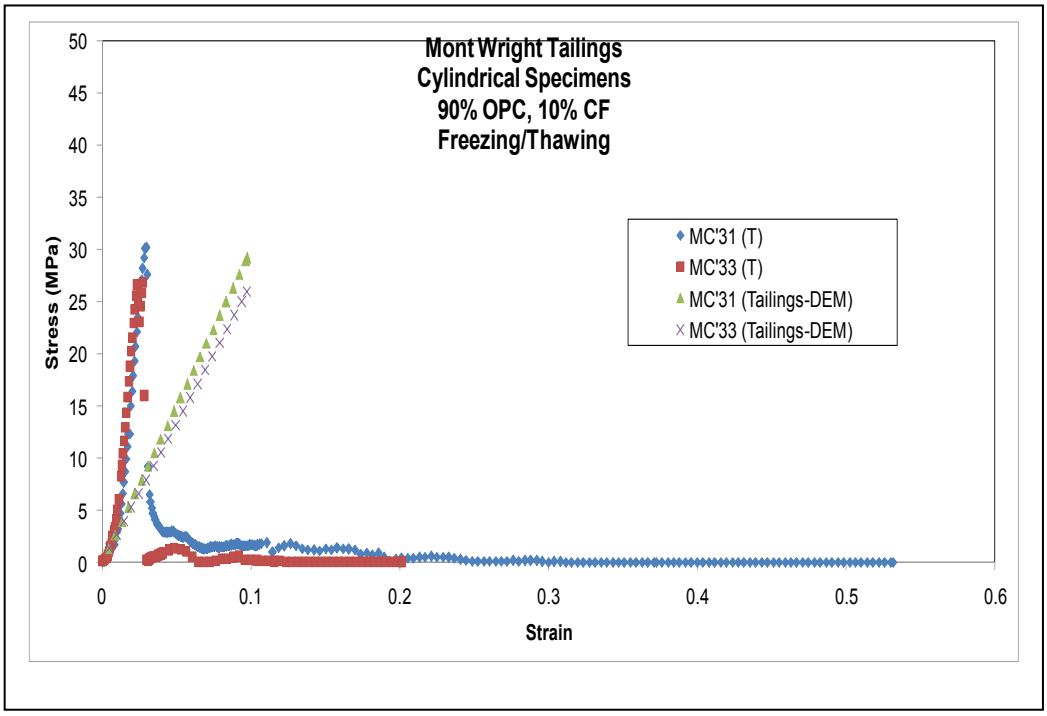


Figure 5.9 Computational and experimental UCS values for Mont Wright samples MC'11-MC'16 after freezing/thawing



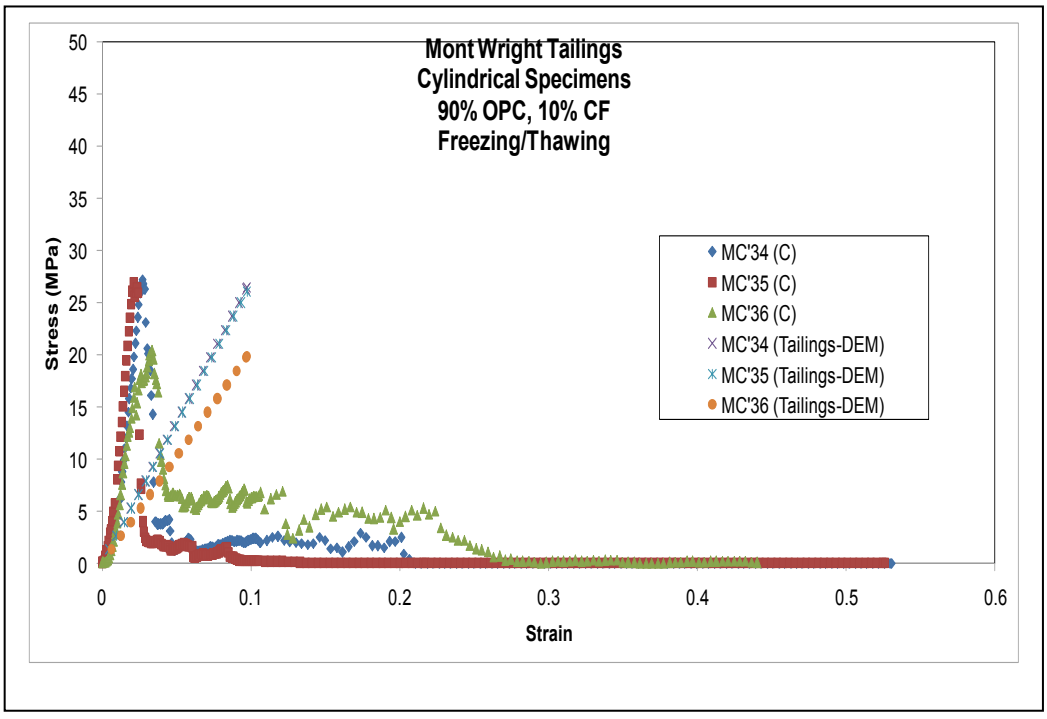
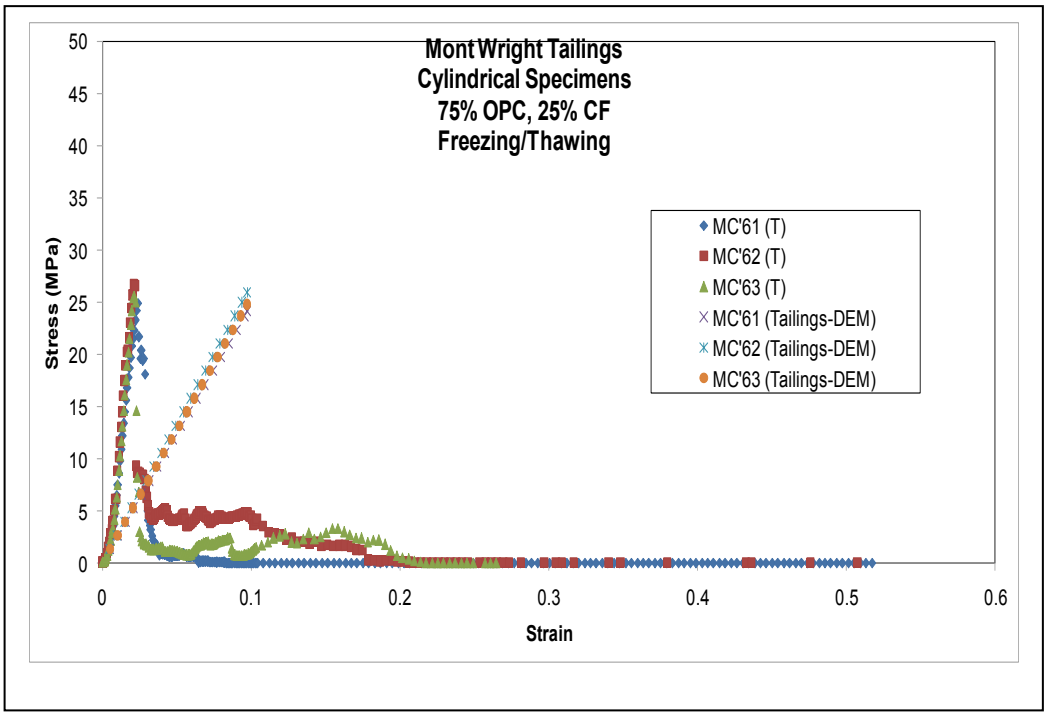


Figure 5.10 Computational and experimental UCS values for Mont Wright samples MC'31-MC'36 after freezing/thawing



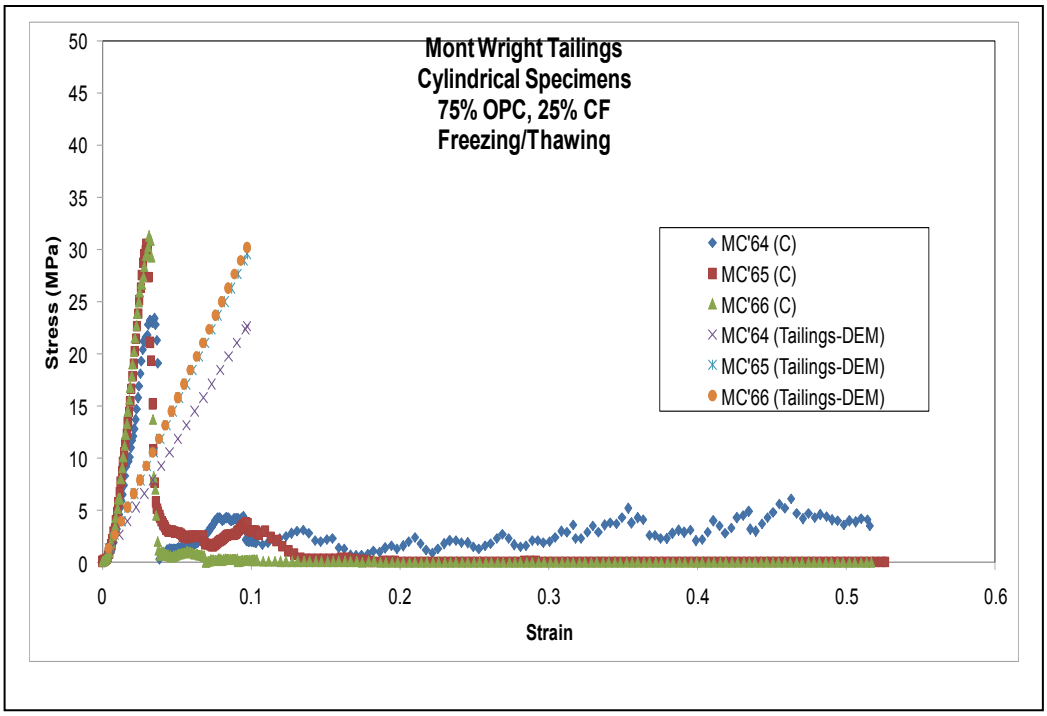
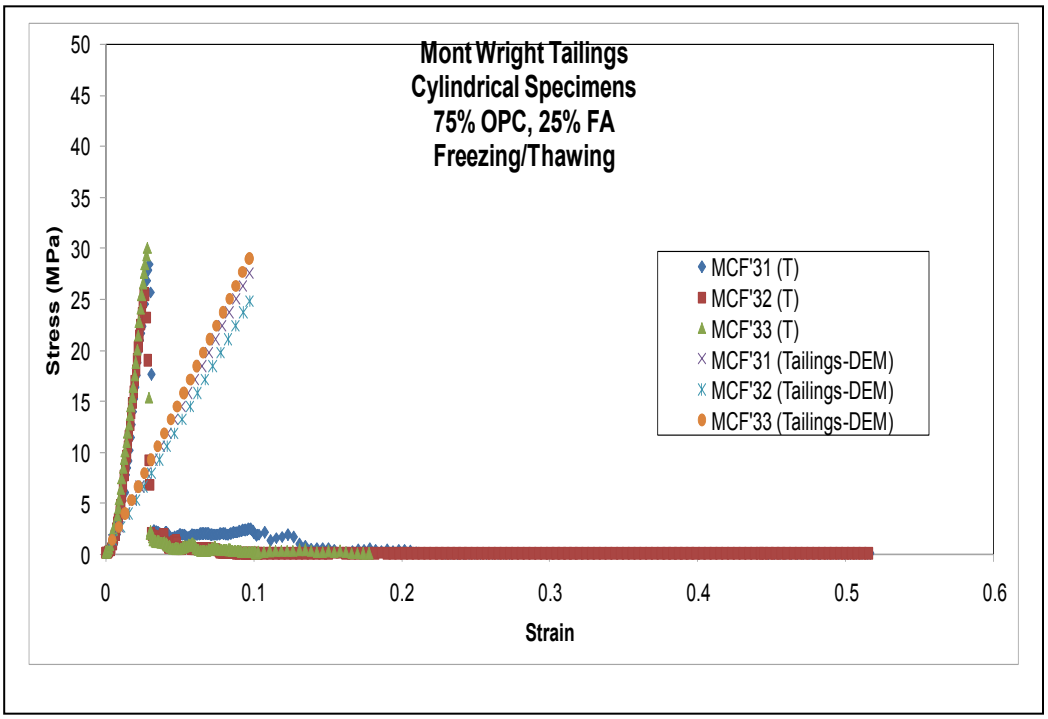


Figure 5.11 Computational and experimental UCS values for Mont Wright samples MC'61-MC'66 after freezing/thawing



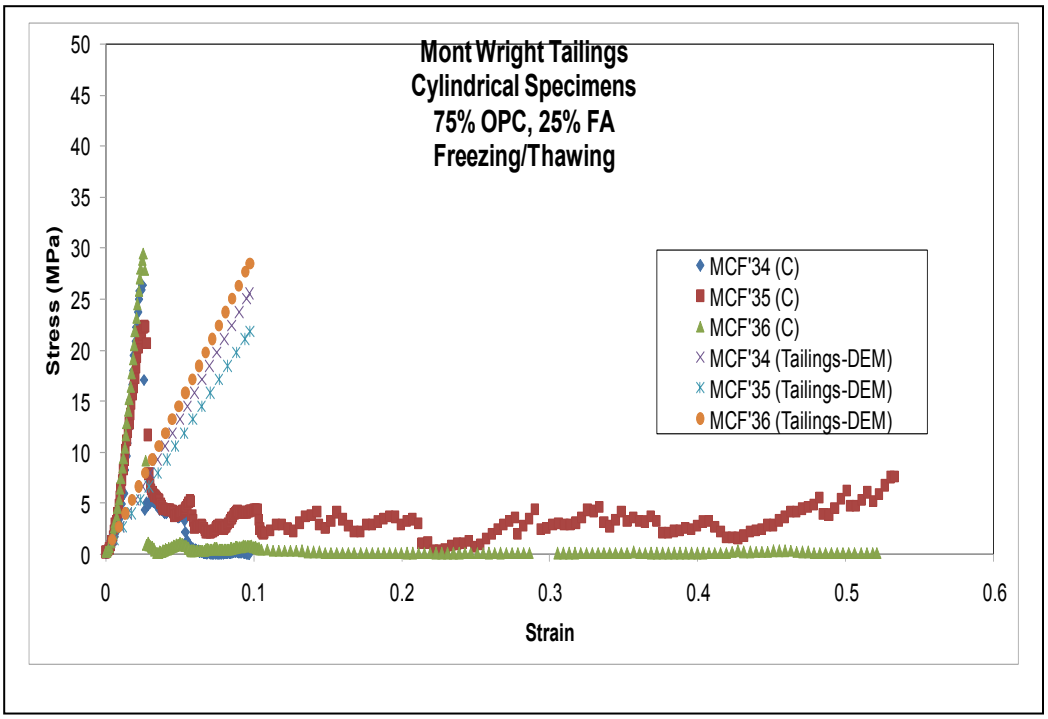
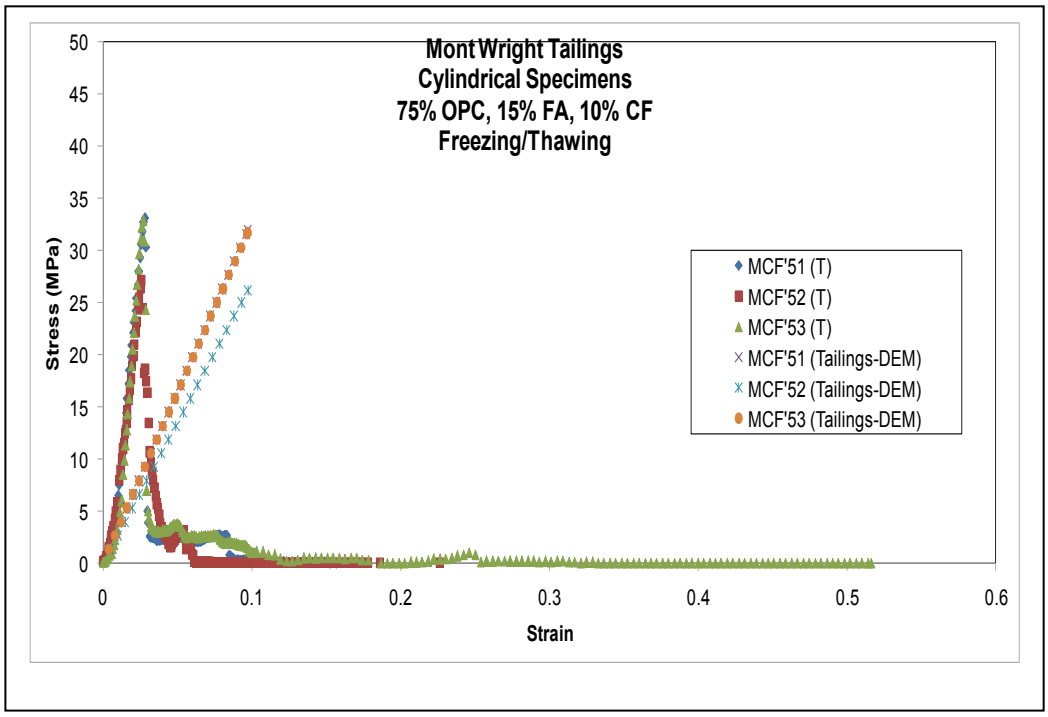


Figure 5.12 Computational and experimental UCS values for Mont Wright samples MCF'31-MC'36 after freezing/thawing



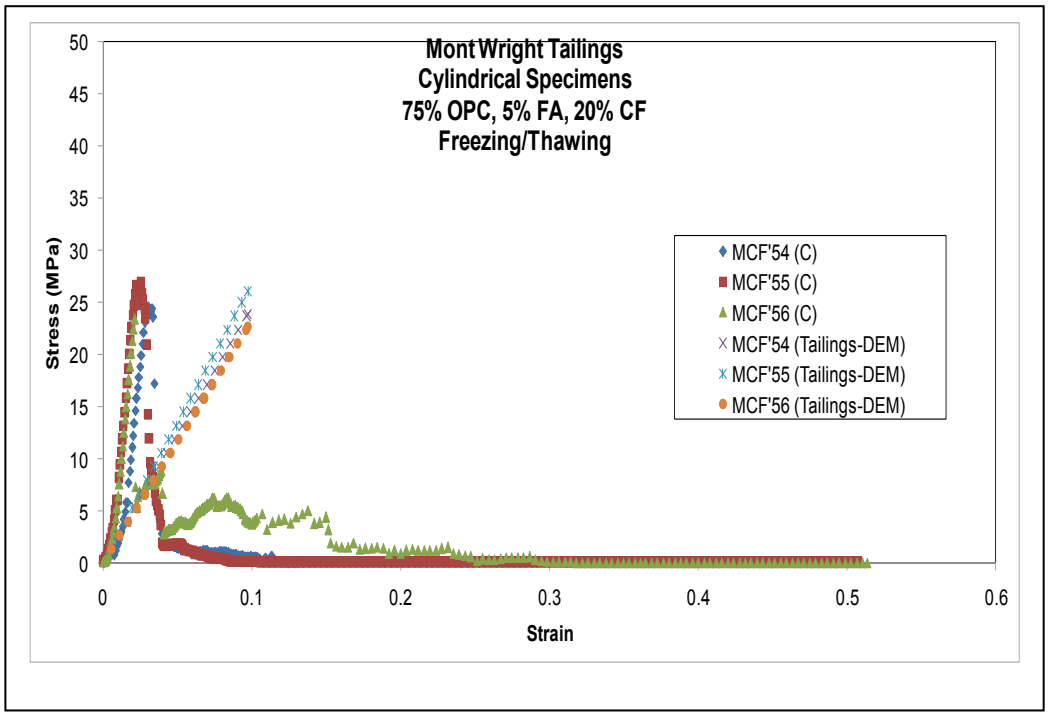
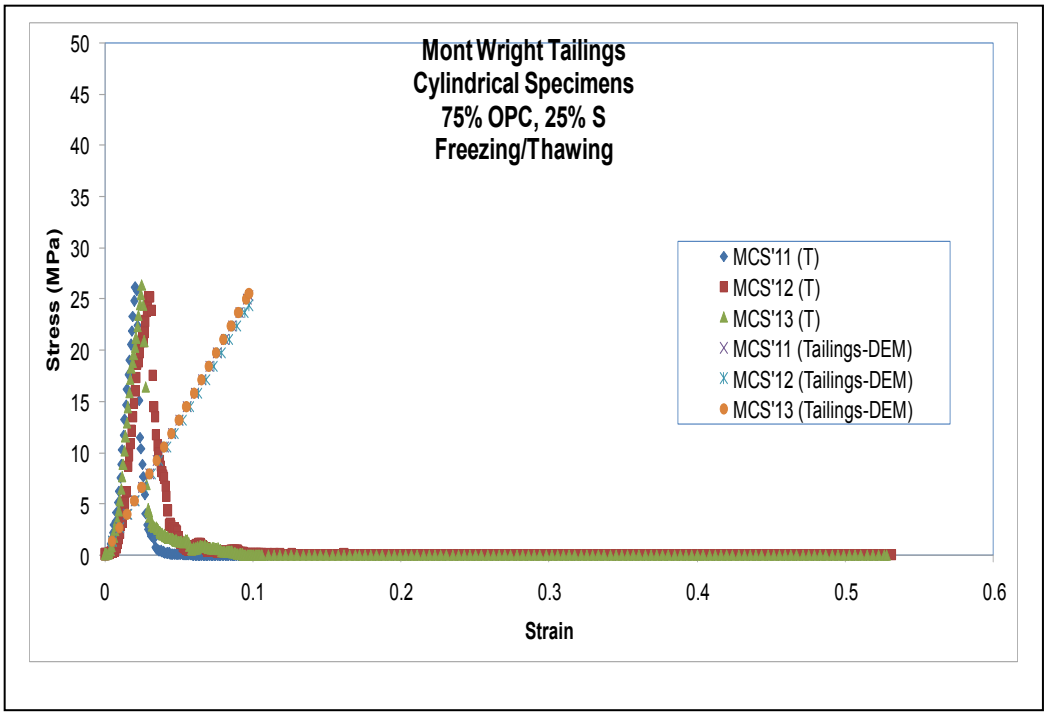


Figure 5.13 Computational and experimental UCS values for Mont Wright samples MCF'51-MCF'56 after freezing/thawing



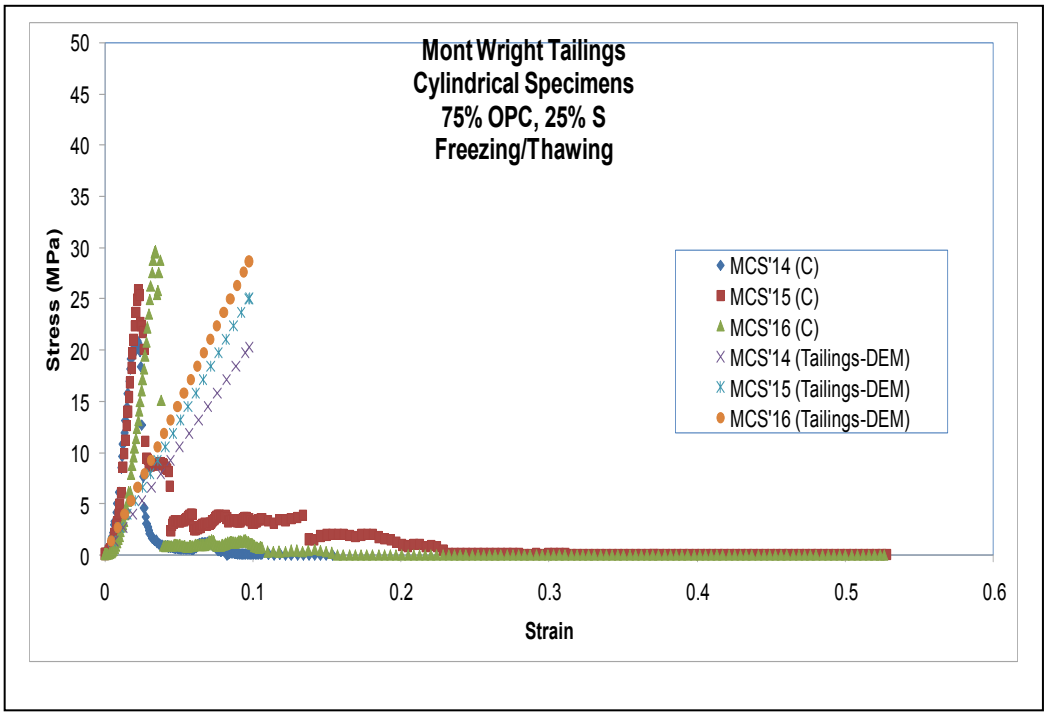
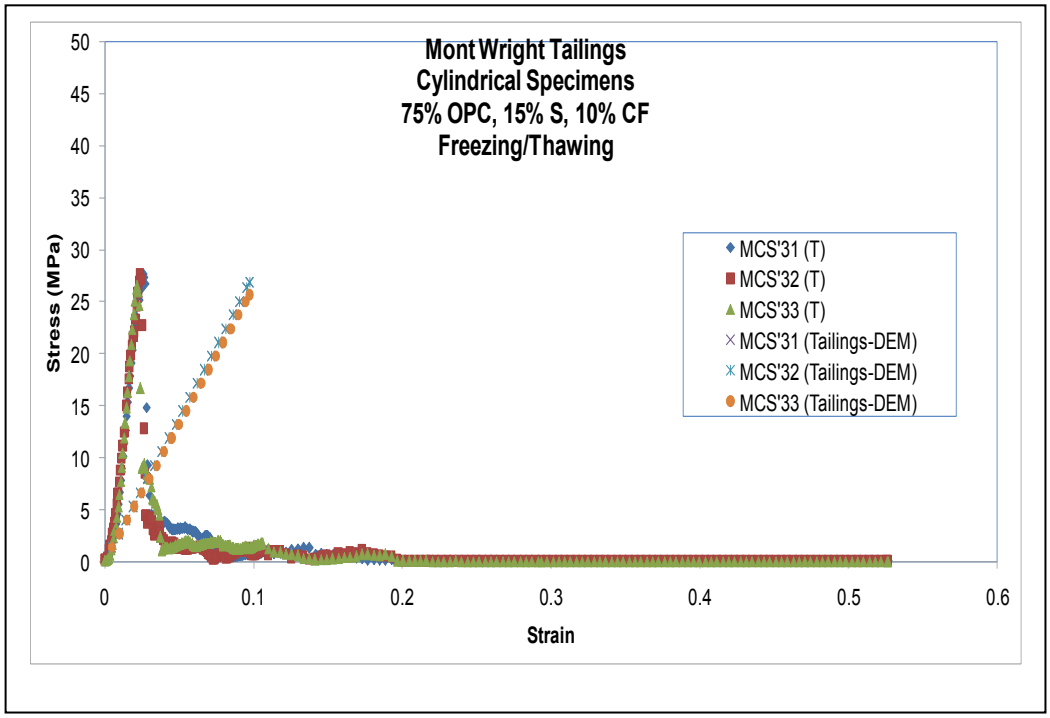


Figure 5.14 Computational and experimental UCS values for Mont Wright samples MCS'11-MCS'16 after freezing/thawing



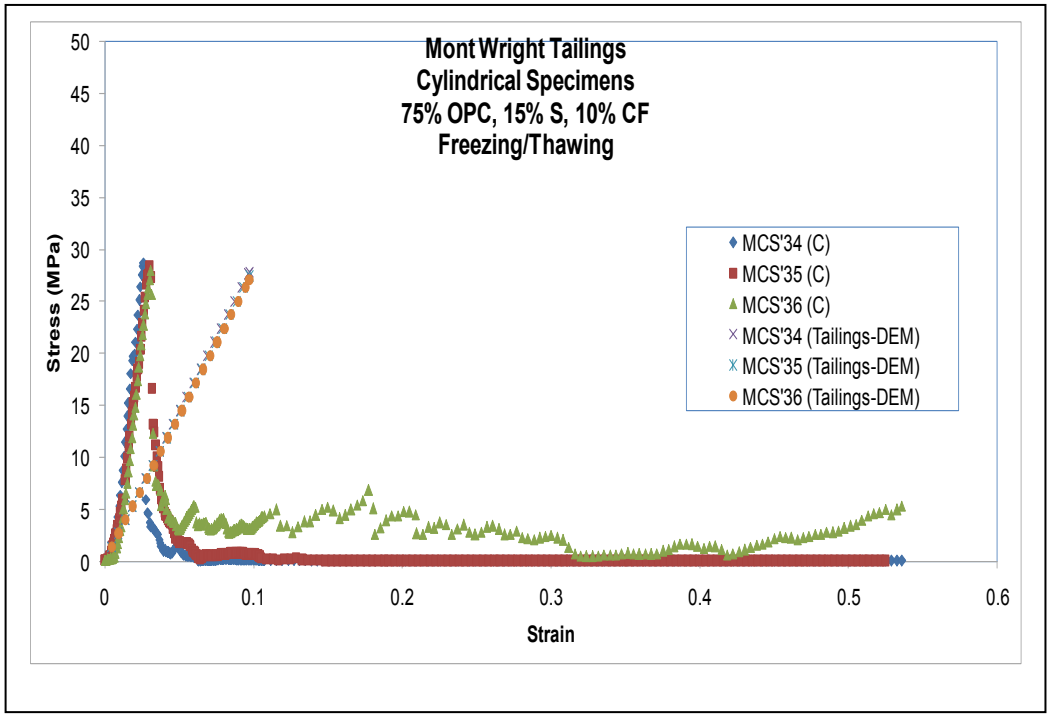
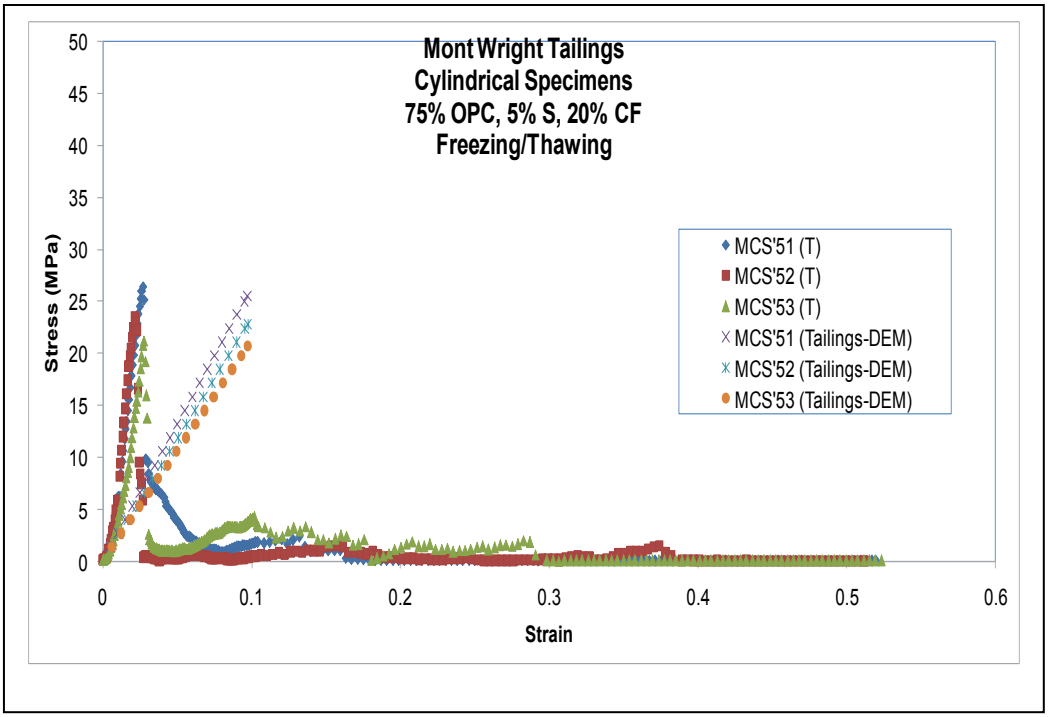


Figure 5.15 Computational and experimental UCS values for Mont Wright samples MCS'31-MCS'36 after freezing/thawing



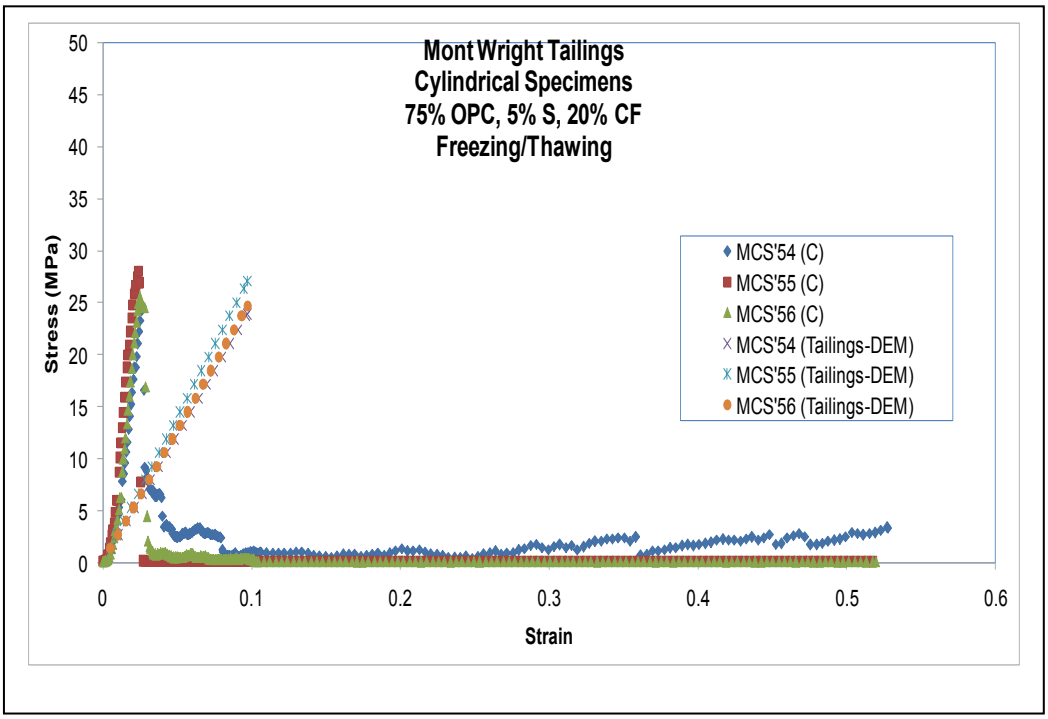
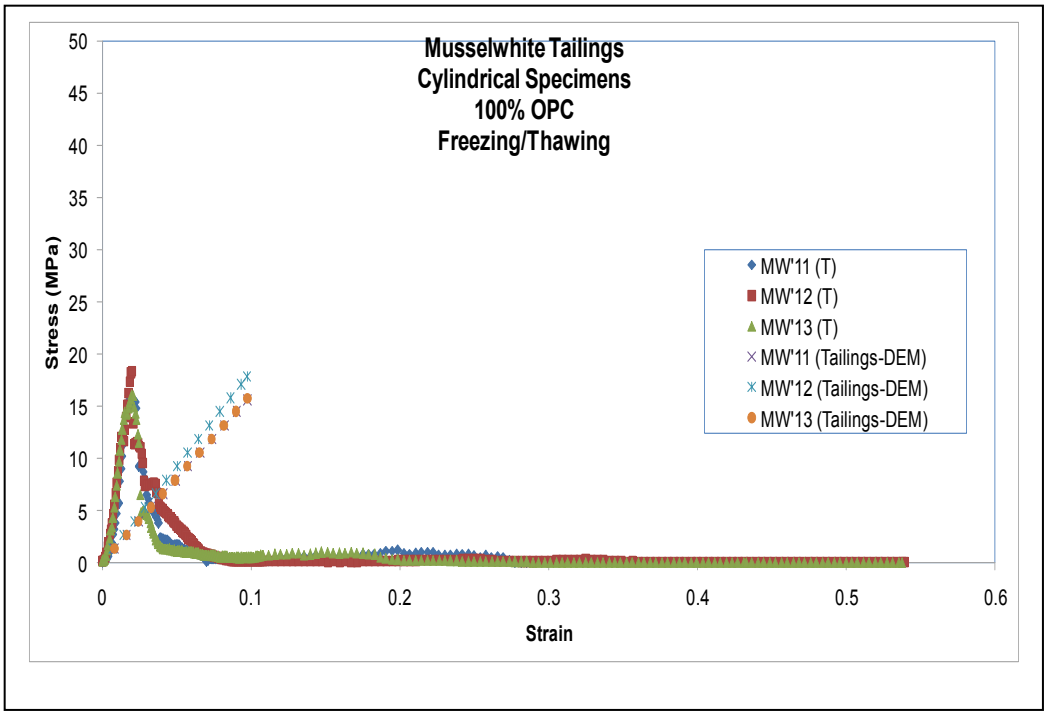


Figure 5.16 Computational and experimental UCS values for Mont Wright samples MCS'51-MCS'56 after freezing/thawing



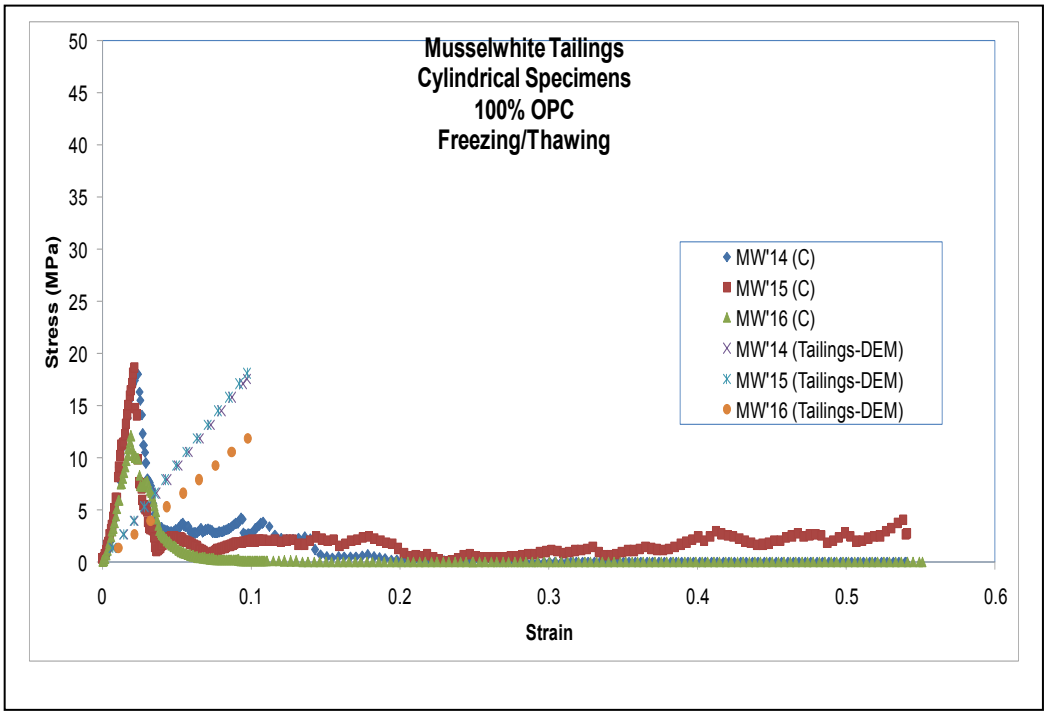
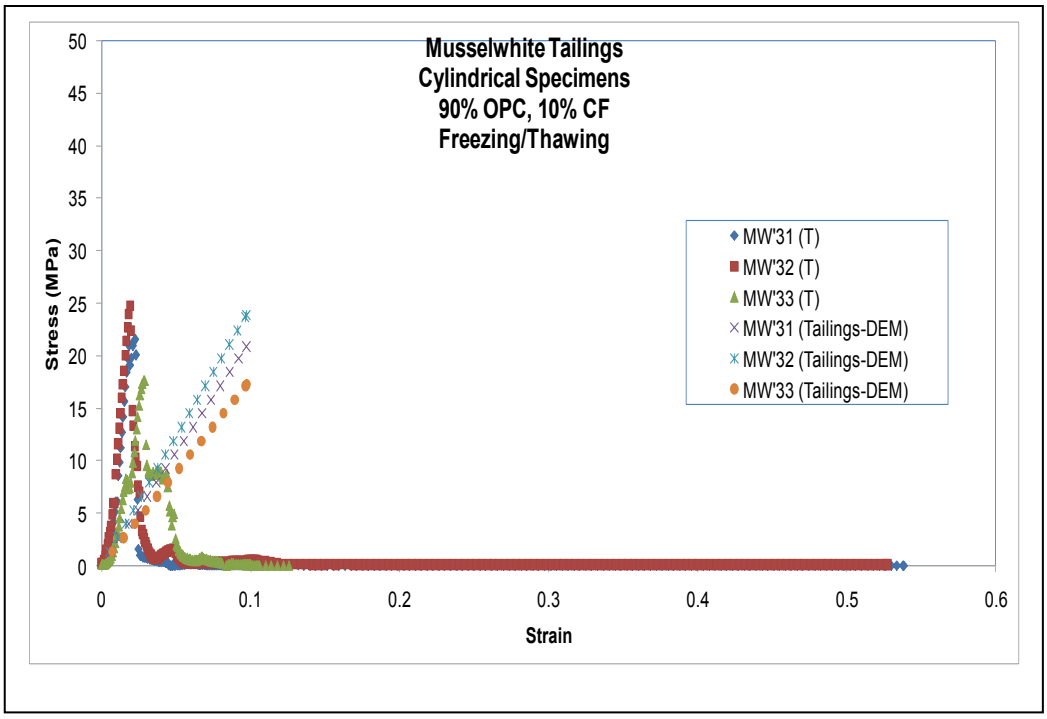


Figure 5.17 Computational and experimental UCS values for Musselwhite samples MW'11-MW'16 after freezing/thawing



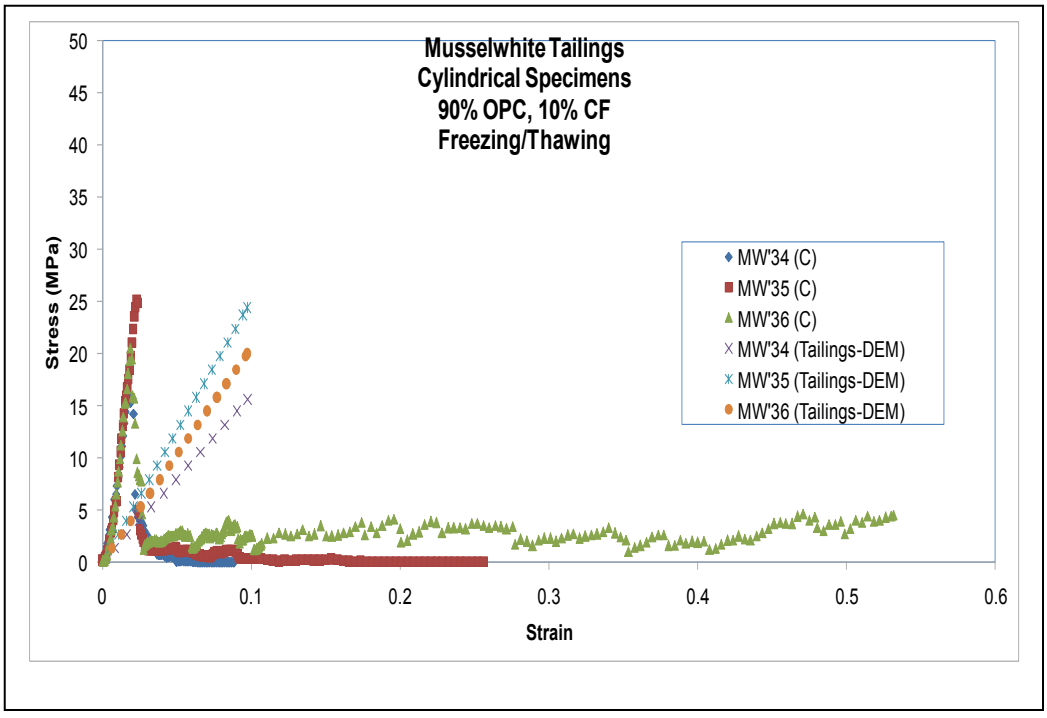
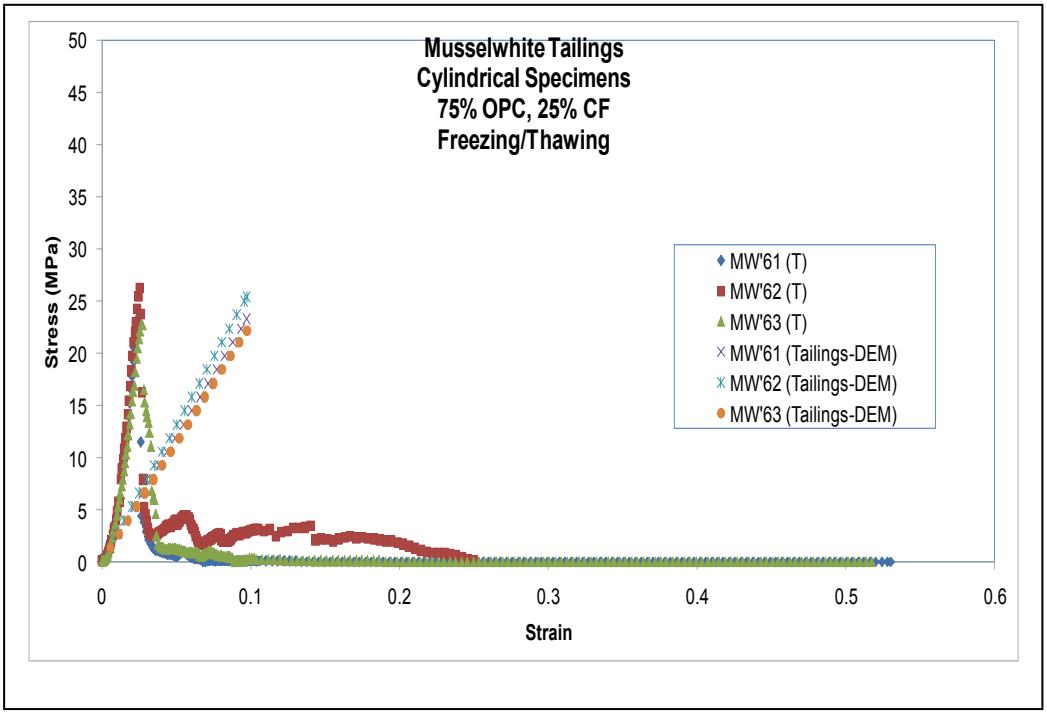


Figure 5.18 Computational and experimental UCS values for Musselwhite samples MW'31-MW'36 after freezing/thawing



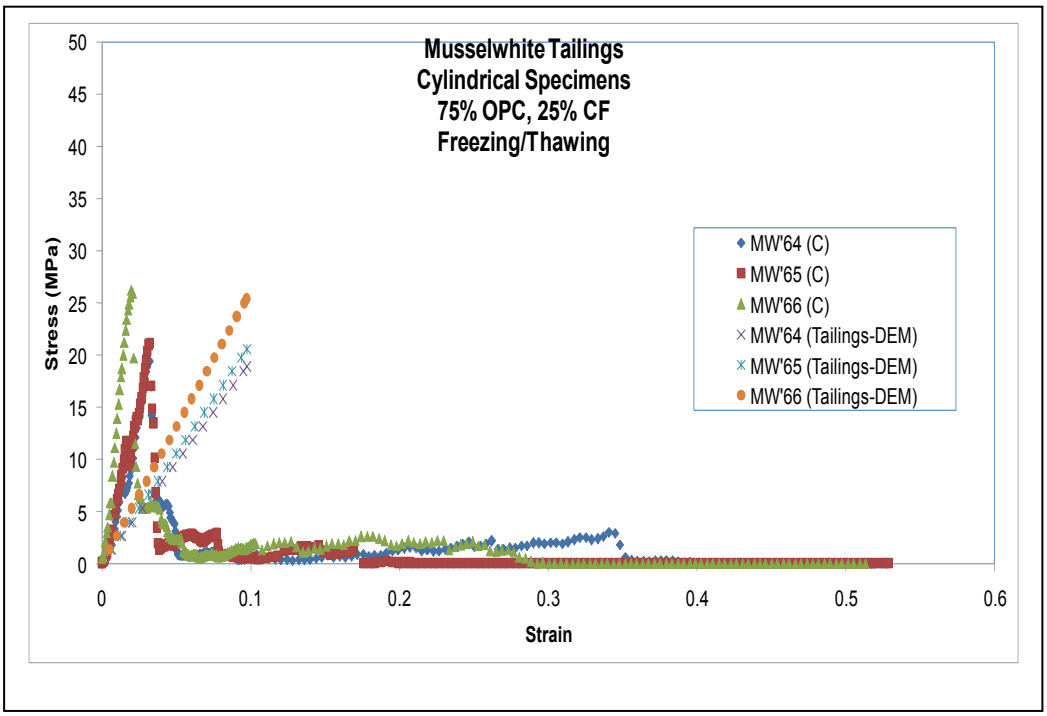
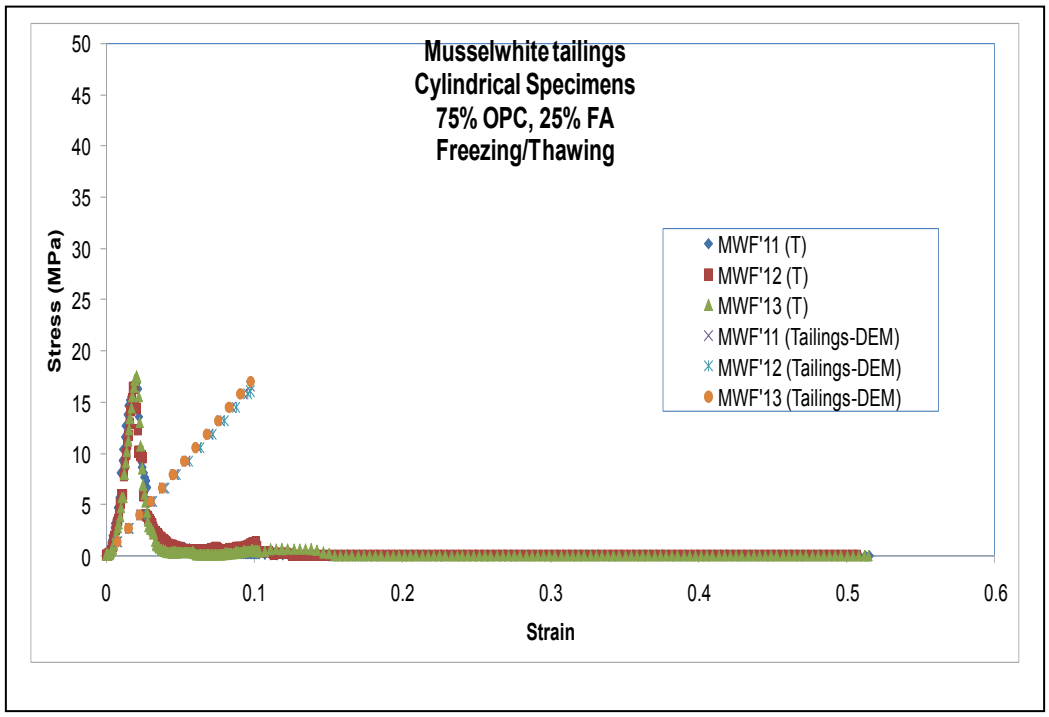


Figure 5.19 Computational and experimental UCS values for Musselwhite samples MW'61-MW'66 after freezing/thawing



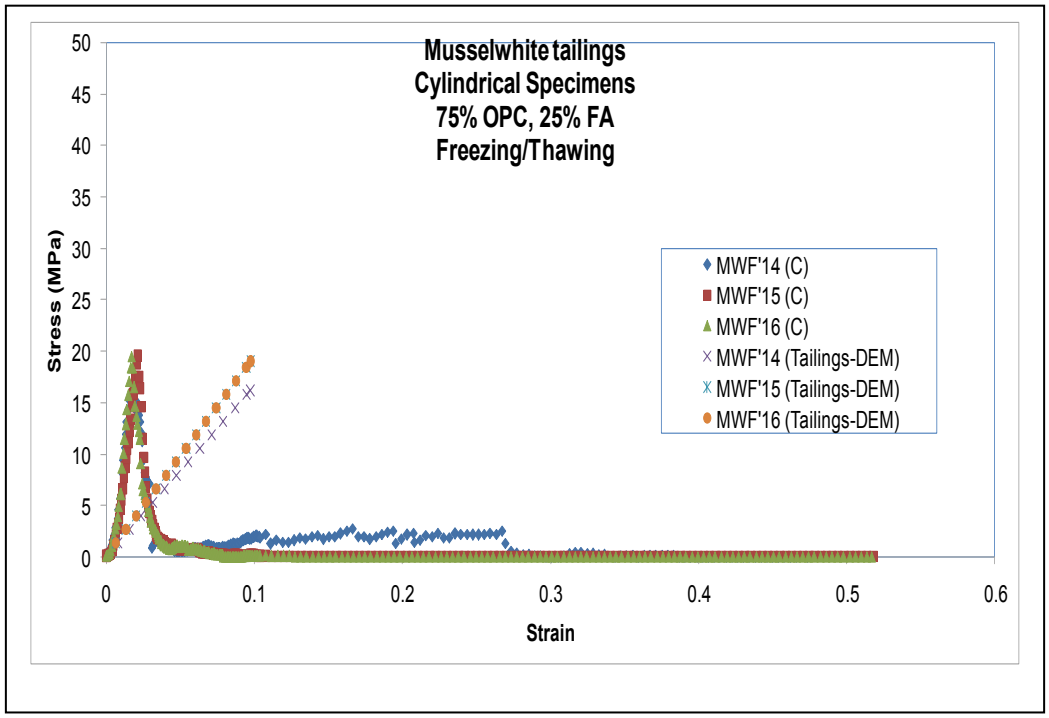
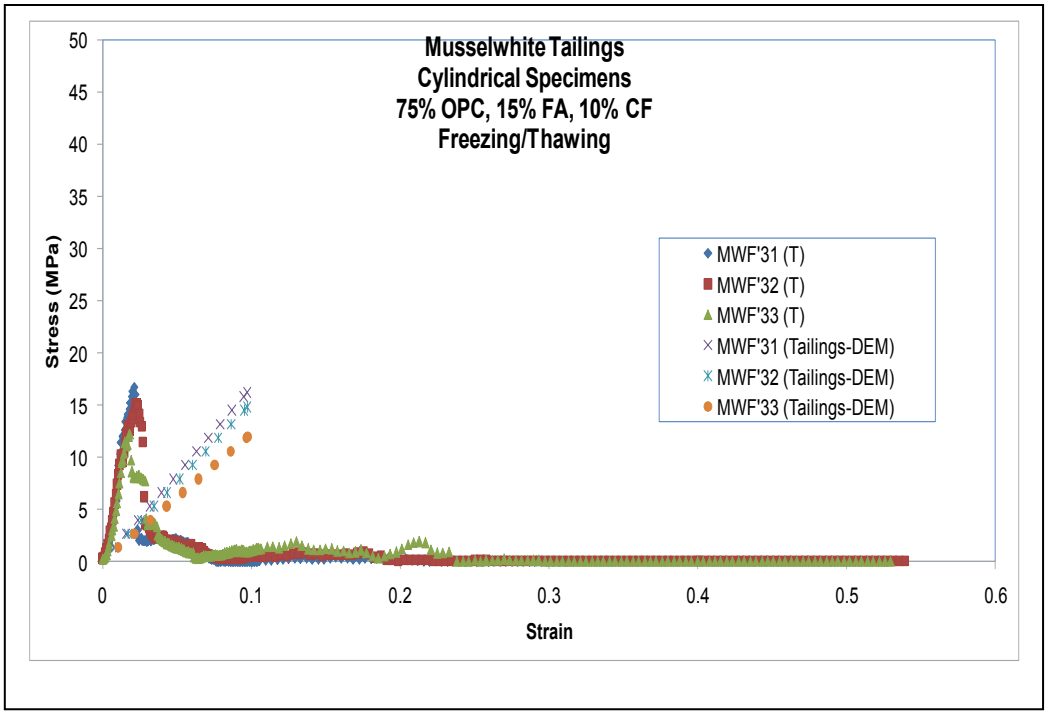


Figure 5.20 Computational and experimental UCS values for Musselwhite samples MWF'11-MWF'16 after freezing/thawing



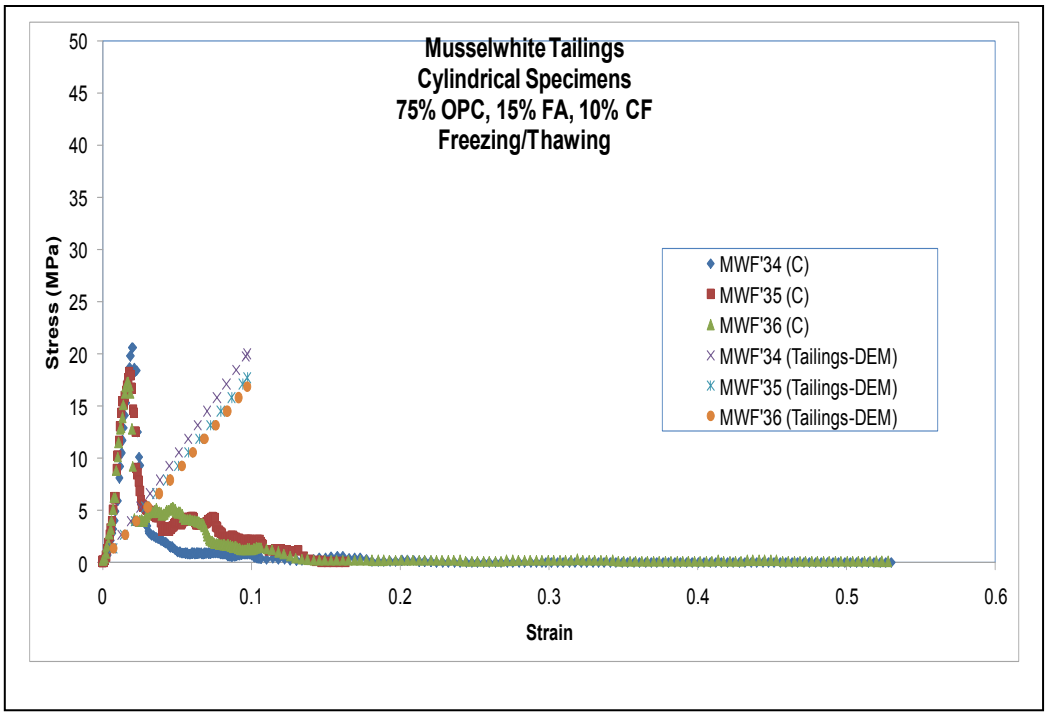
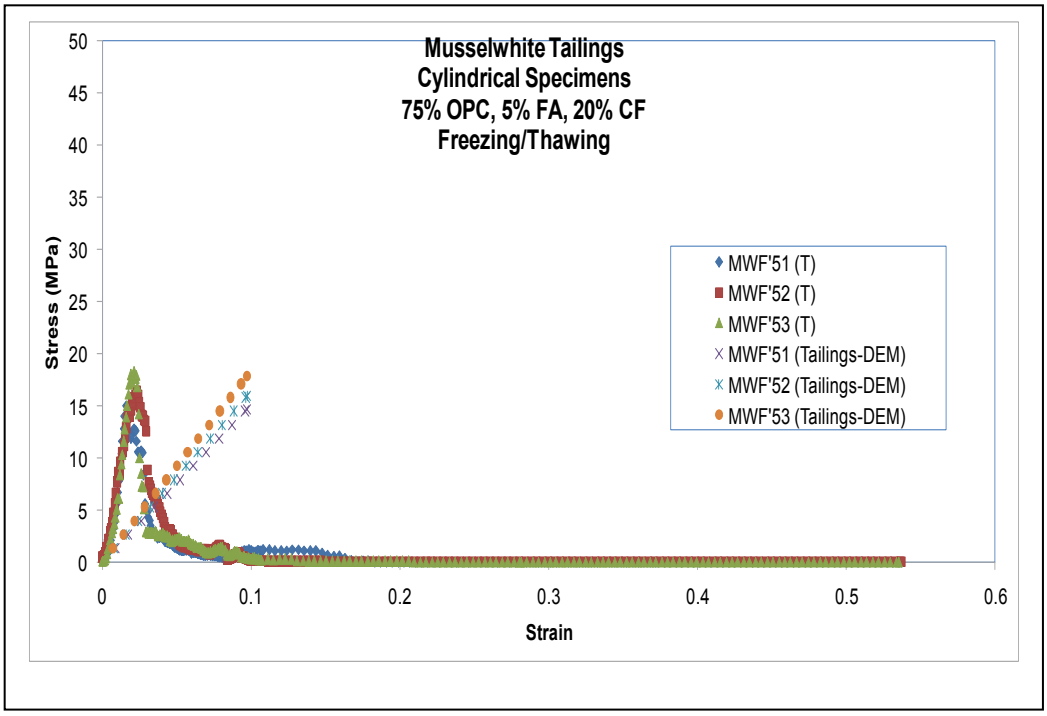


Figure 5.21 Computational and experimental UCS values for Musselwhite samples MWF'31-MWF'36 after freezing/thawing



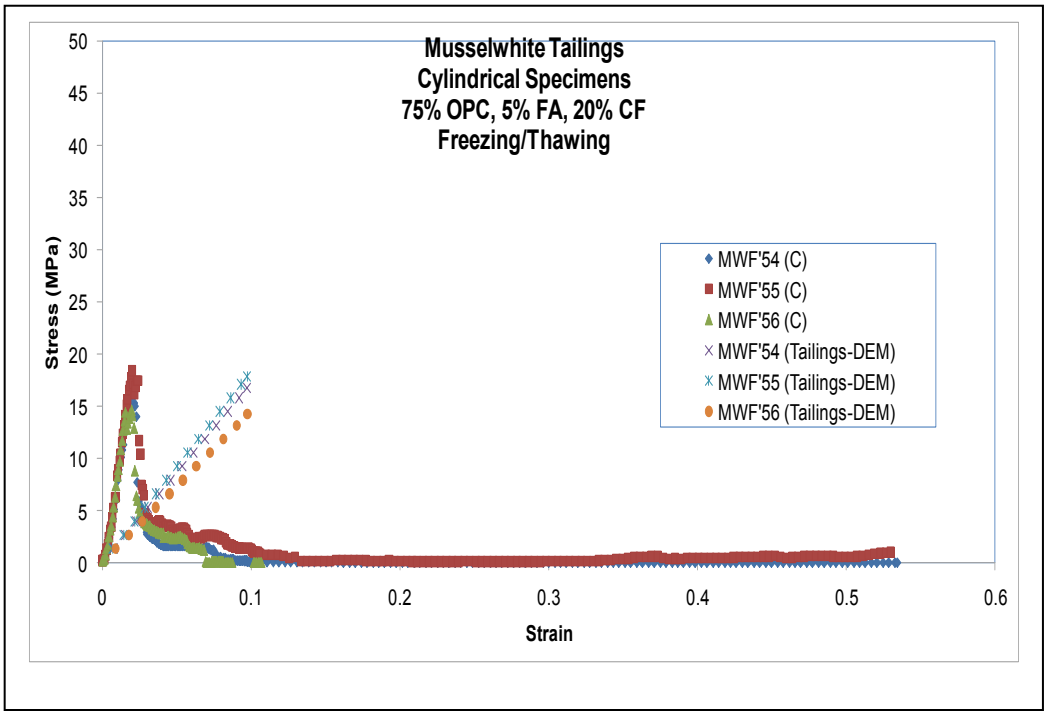
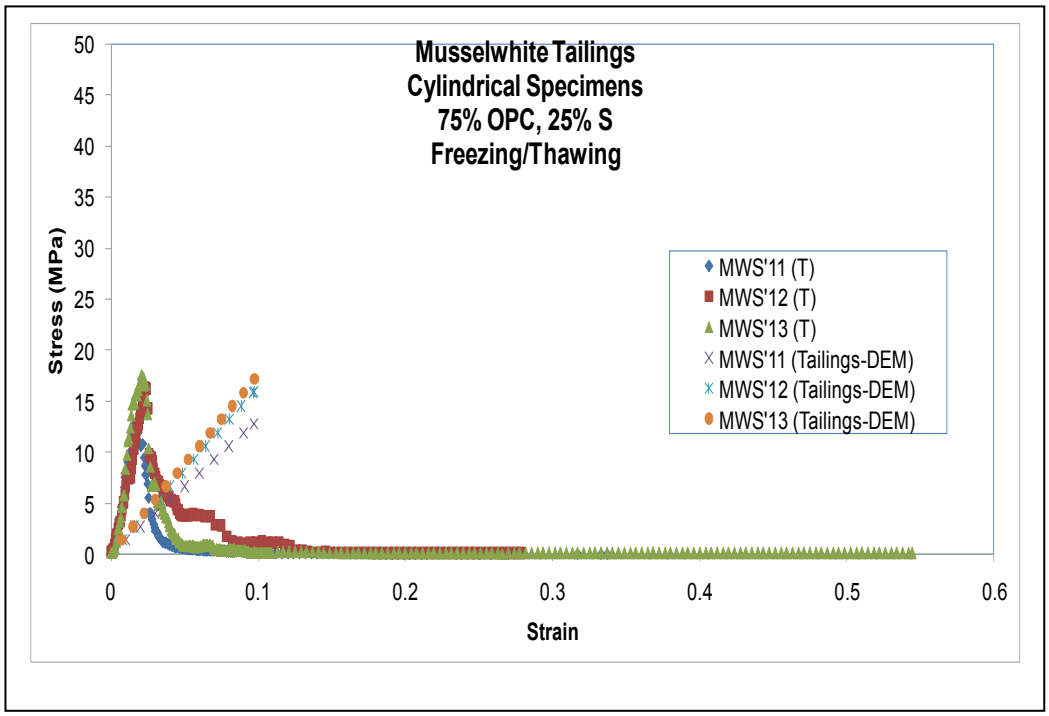


Figure 5.22 Computational and experimental UCS values for Musselwhite samples MWF'51-MWF'56 after freezing/thawing



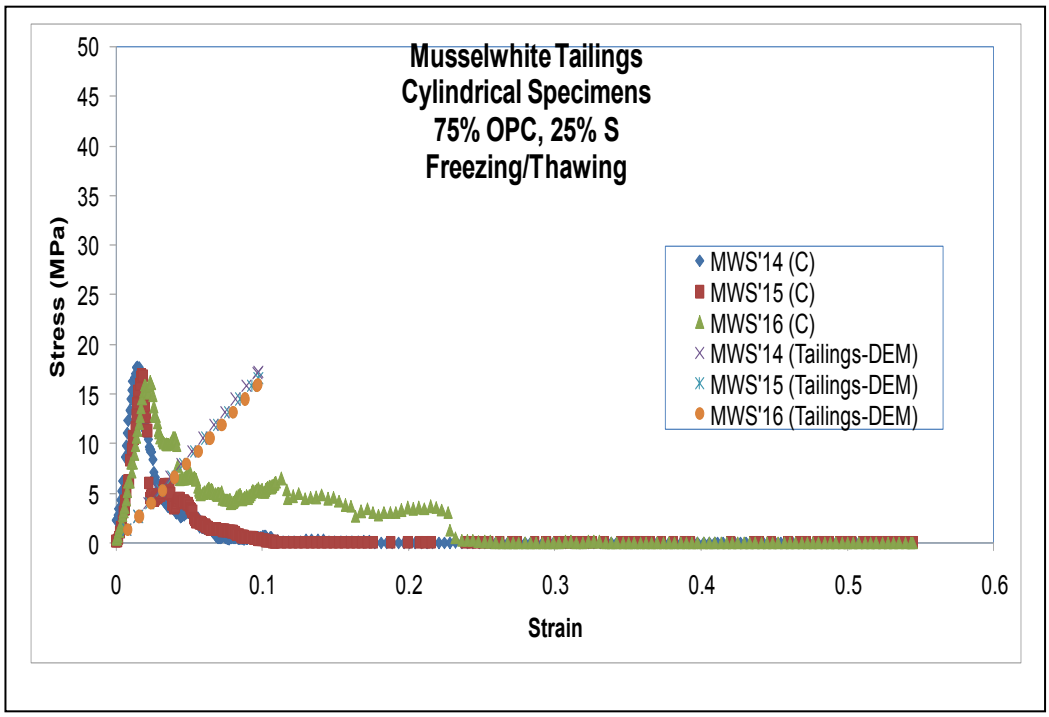
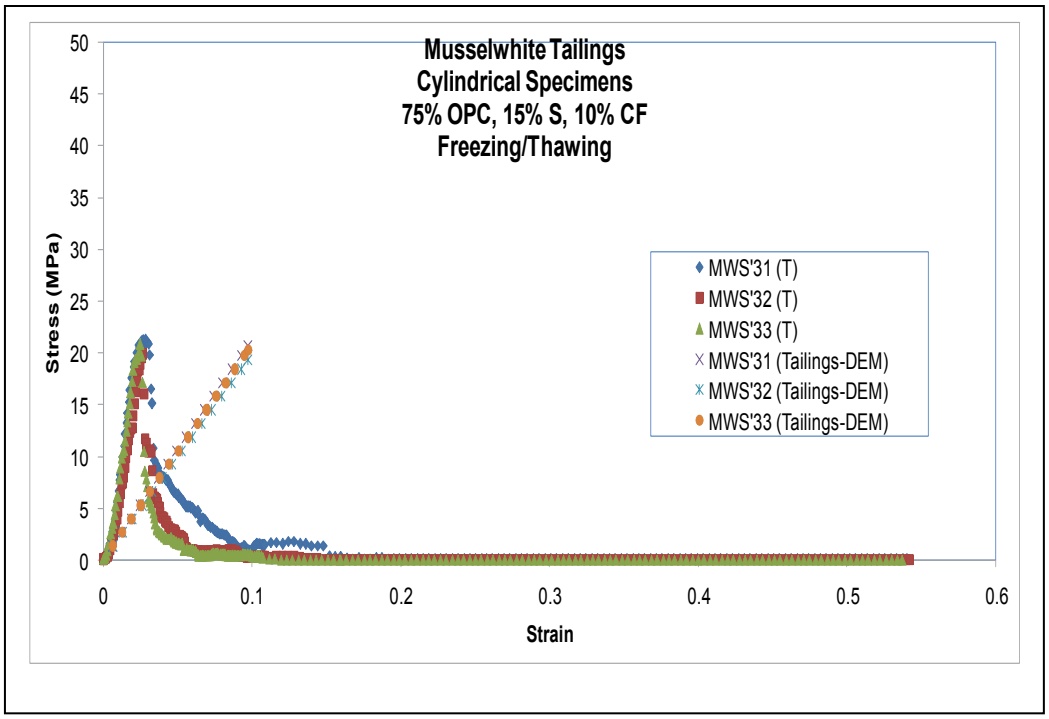


Figure 5.23 Computational and experimental UCS values for Musselwhite samples MWS'11-MWS'16 after freezing/thawing



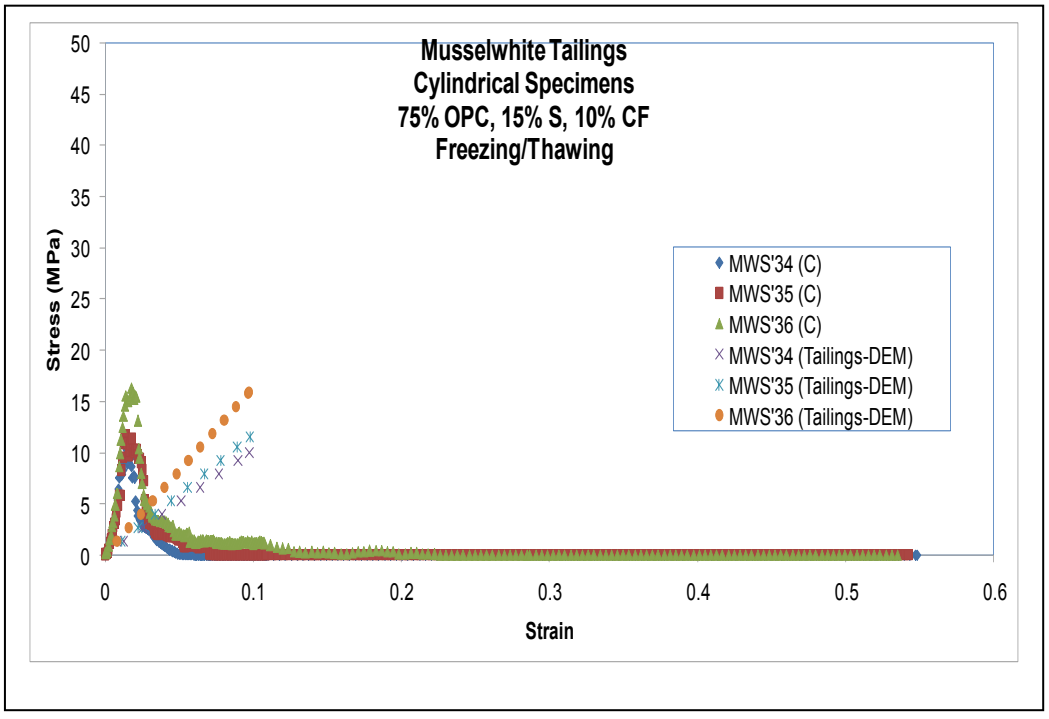
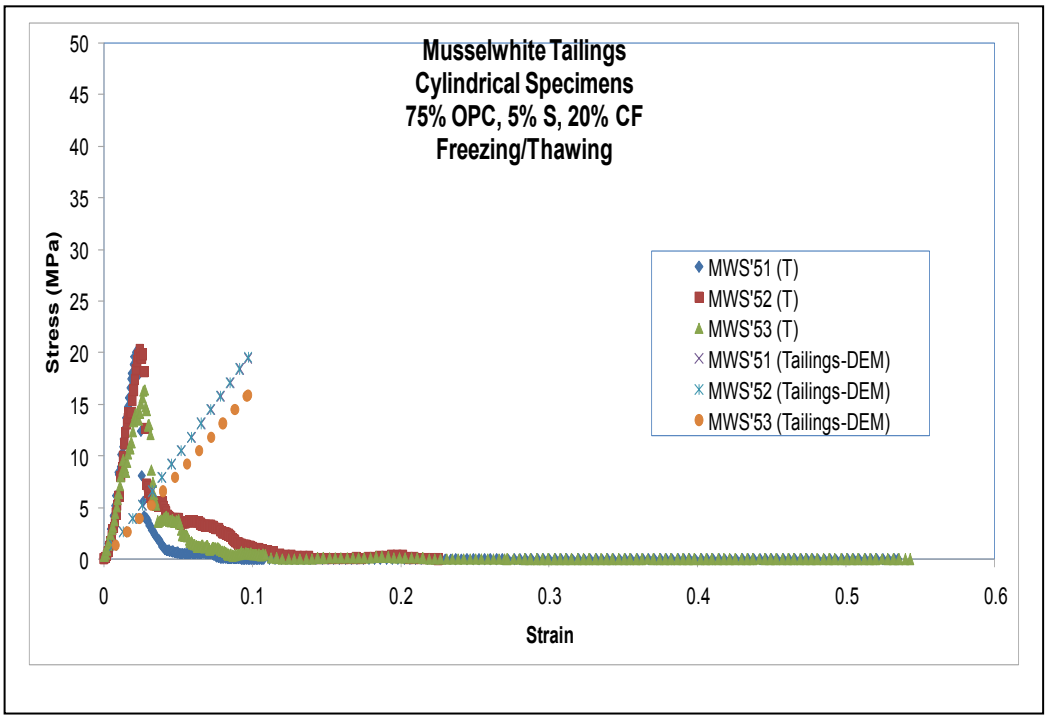


Figure 5.24 Computational and experimental UCS values for Musselwhite samples MWS'31-MWS'36 after freezing/thawing



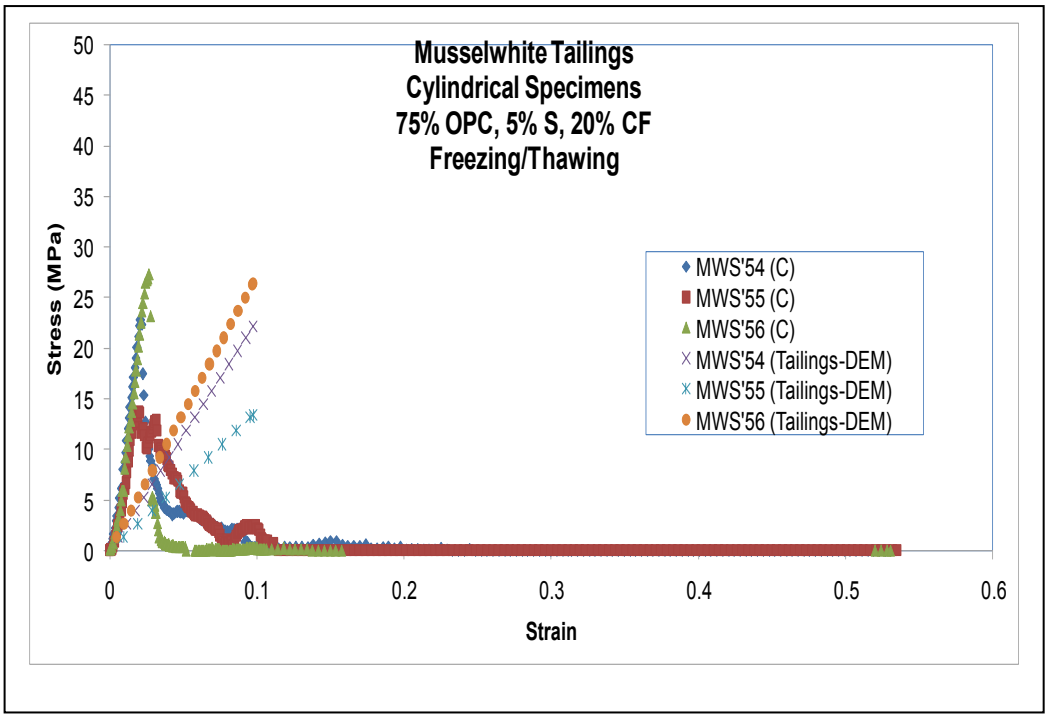


Figure 5.25 Computational and experimental UCS values for Musselwhite samples MWS'51-MWS'56 after freezing/thawing

5.4.2 Discussion of Tailings-DEM™ strength modeling

The computational data generally followed closely experimental results for these tailings matrices for the cases shown. Maximum values and slopes for the experimental and computational results followed closely. It is evident, however, that the freezing /thawing specimens have higher slopes for the same tailings indicating the move from ductile to more brittle behavior.

Generally, computational results followed the experimental values with Mont Wright tailings matrices having higher computational strength values than their Musselwhite counterparts.

The marginal differences between the computational and experimental UCS results could be attributed to the experimental errors in the preparation of the specimens tested. It is known that soil cement matrices suffer from result variability as was explained in Chapter 4. Furthermore, the presence of metals; Fe in Mont Wright tailings and Fe, Zn and Pb in Musselwhite has further complicated the outcome of the UCS experiments as was discussed earlier in Chapter 4. These experimental results were used in the multiple regression analysis that led to the creation of the predictive Equation (4.17).

Another reason could be the assumptions used in defining Tailings-DEM™. The assumptions were based on the particles being right-isosceles triangles. This assumption had affected the

other assumption of contact mechanism as was explained earlier in Section 5.3. In reality, however, soil particles have different shapes and smaller sizes and their contact mechanism is much more complicated and varied. In addition, most soils contain different amounts of moisture, an affect that was not modeled in Tailings-DEM™.

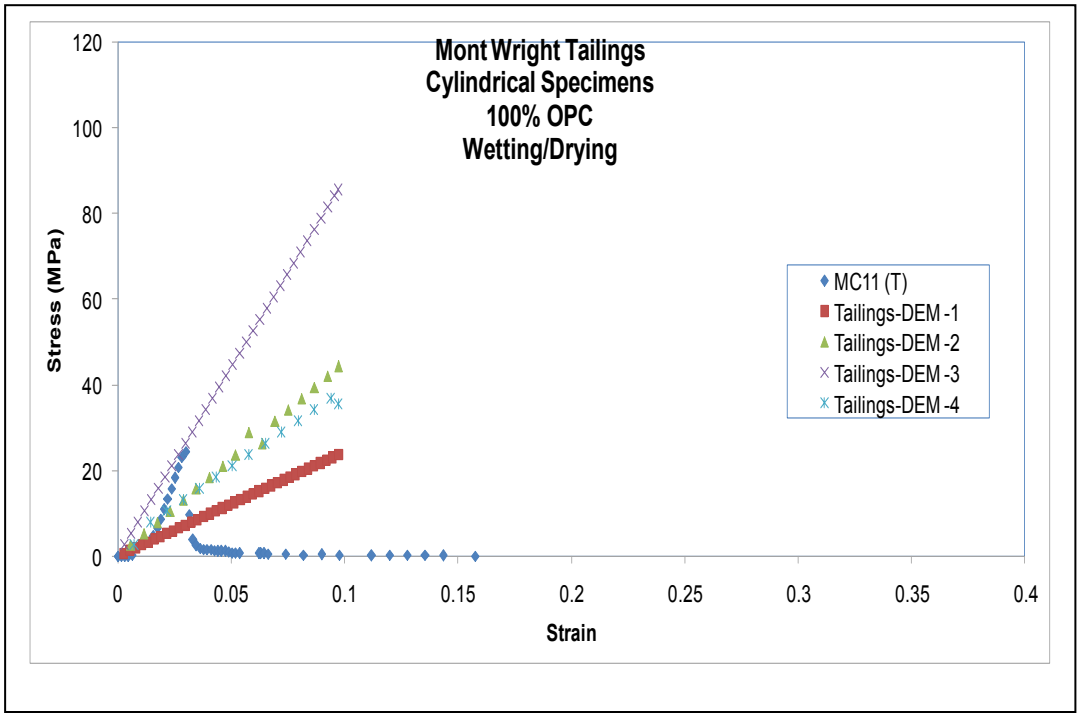
5.4.3 Sensitivity Analysis

In order to verify the sensitivity of the proposed model, the size of the triangular particles has been changed for the subsequent four runs. Dimensions of the particles used for computation in Tailings-DEM, were increased and decreased twice and four times as shown in Table 5.2.

Table 5.2 Particle size variations for the Sensitivity Analysis test

Program Code Number	**Particle Size: Height (mm) x Base (mm)**
Sensitivity series 1 (original particle size)	2.12 x 4.243
Sensitivity series 2	(2.12/2 x 4.243/2) = 1.06 x 2.1215
Sensitivity series 3	(2.12/4 x 4.243/4) = 0.53 x 1.061
Sensitivity series 4	(2.12(2) x 4.243(2)) = 4.24 x 8.486
Sensitivity series 5	(2.12(4) x 4.243(4)) = 8.48 x 16.972

General behavior patterns are similar for all matrices tested in this research program as can be seen when examining the results of Tailings-DEM™ in Figures 5.9 to 5.25. The MC1 and MW1 (Chapter 4) specimens and their derivatives, which comprise a total of 11 specimens, were chosen for the sensitivity analysis. Both matrices represent the two types of tailings matrices used in these experiments; Mont Wright and Musslwhite. Both tailings matrices combinations are shown in Table 5.3. The two matrices were chosen mainly because of their Portland cement only binder content. It is known that cementitious stabilization/solidification is one of the most widely used techniques for the treatment and ultimate disposal of hazardous waste and low level radioactive waste (USEPA 1996). Therefore, comparison with other research results would be easier.



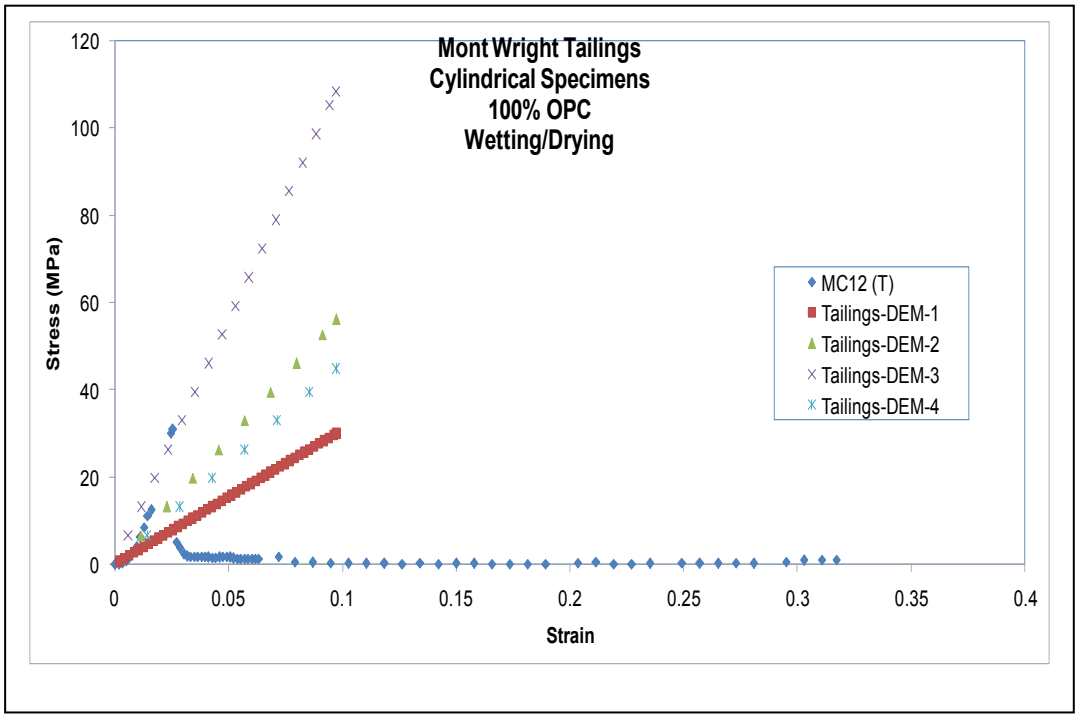
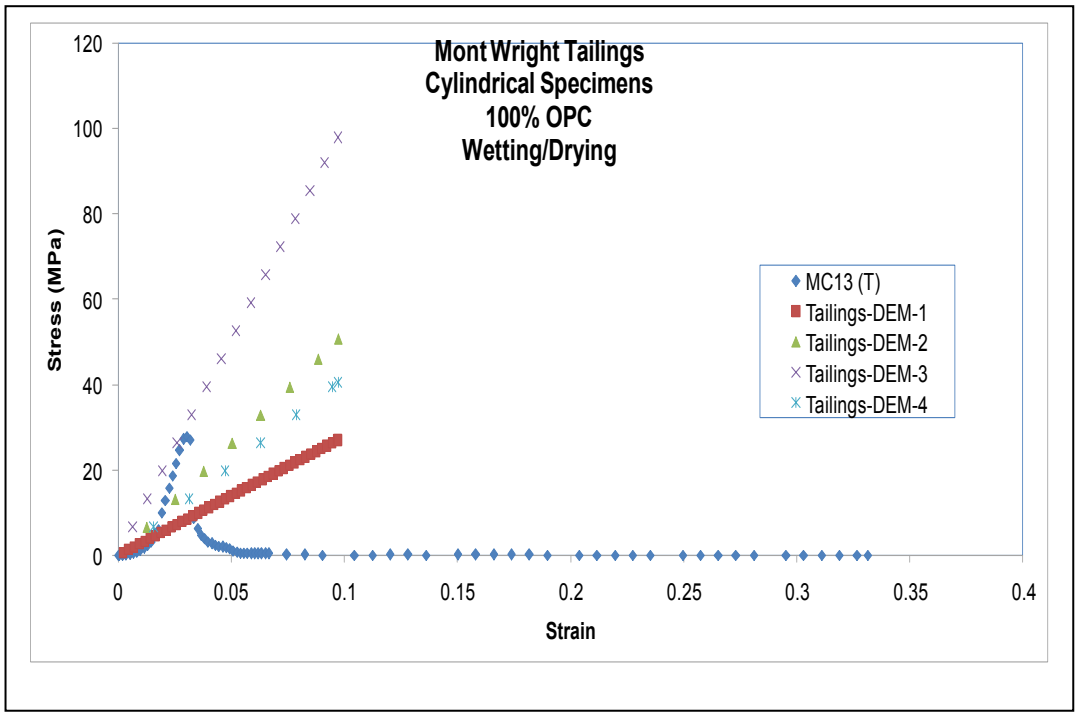


Figure 5.26 Sensitivity Analysis Test for Mont Wright samples MC11 and MC12



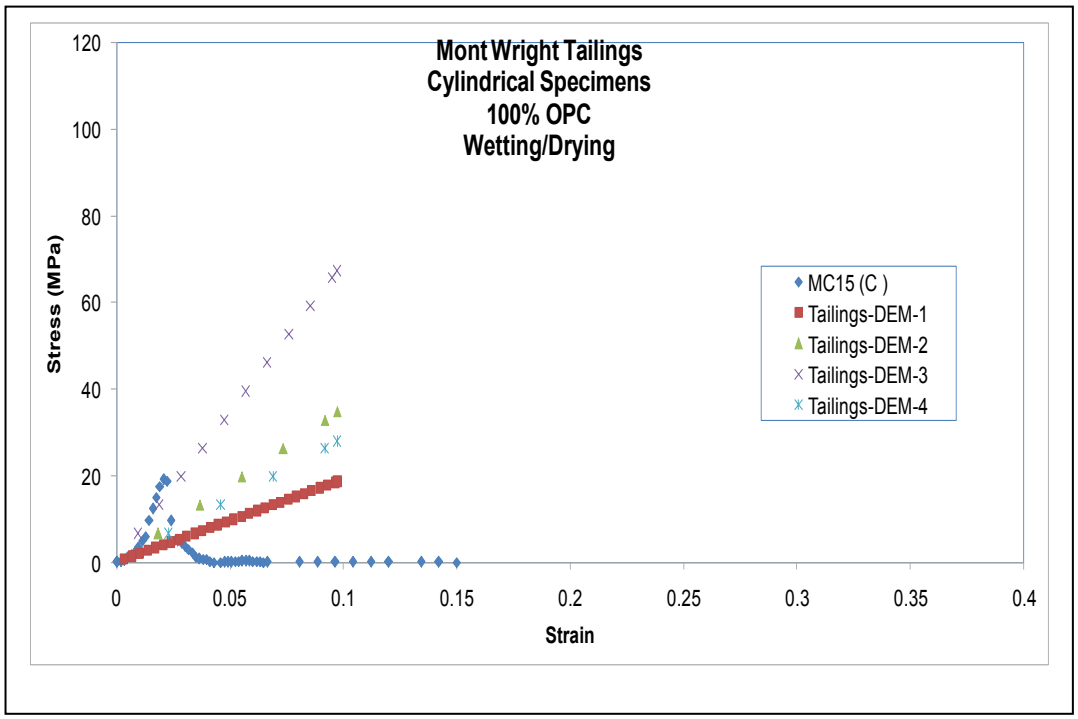
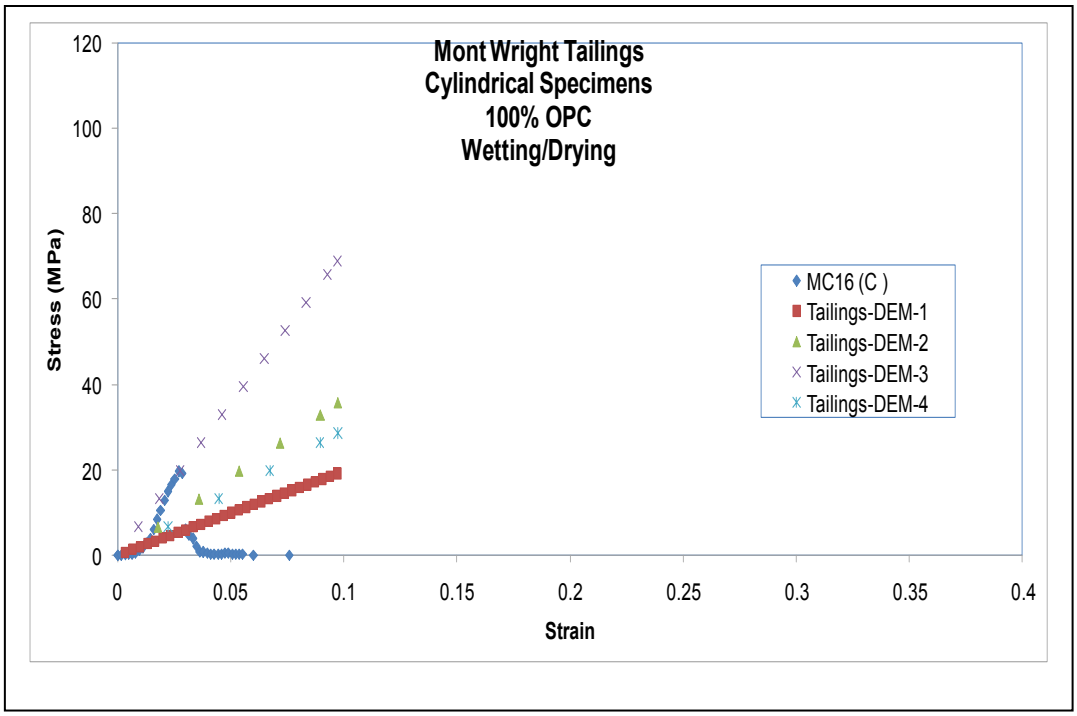


Figure 5.27 Sensitivity Analysis Test for Mont Wright samples MC13 and MC15



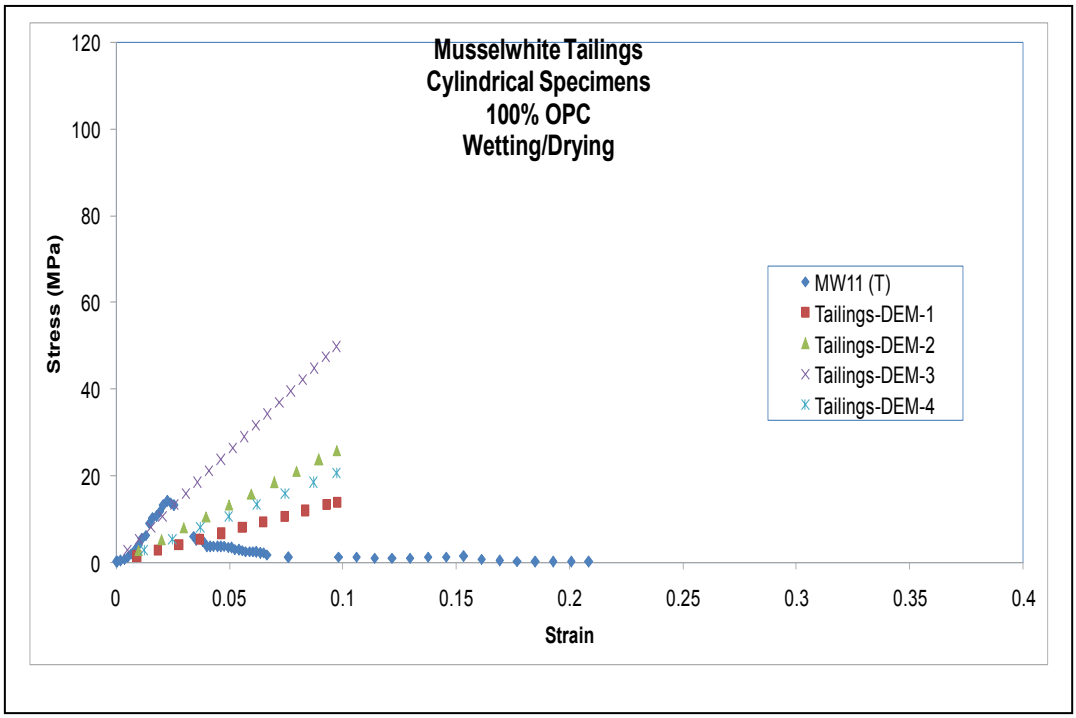


Figure 5.28 Sensitivity Analysis Test for Mont Wright sample MC16 and Musselwhite sample MW11

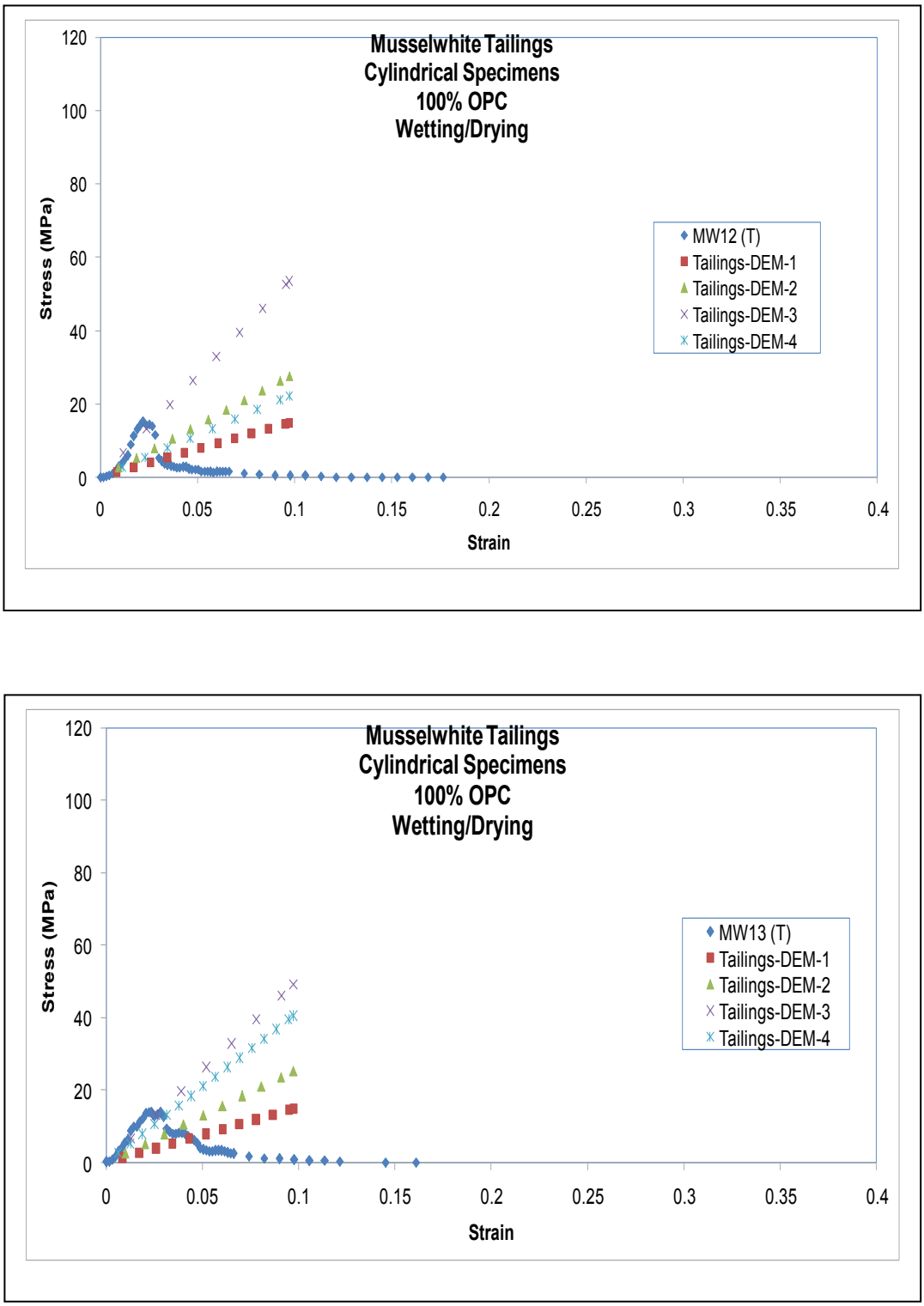
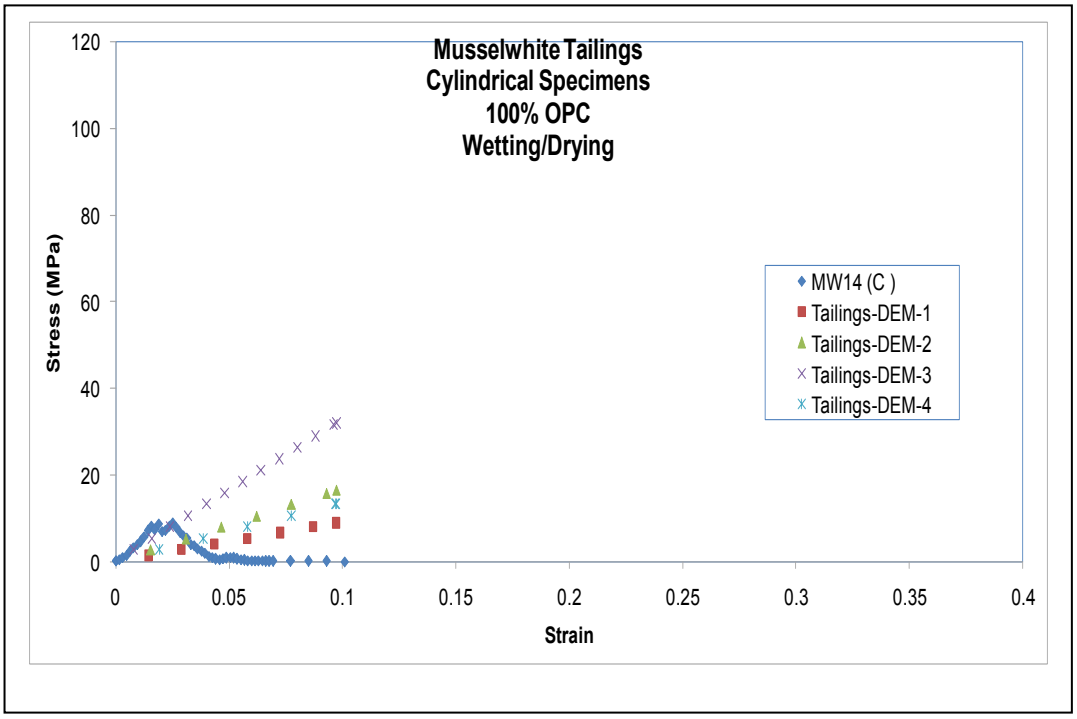


Figure 5.29 Sensitivity Analysis Test for Musselwhite samples MW12 and MW13



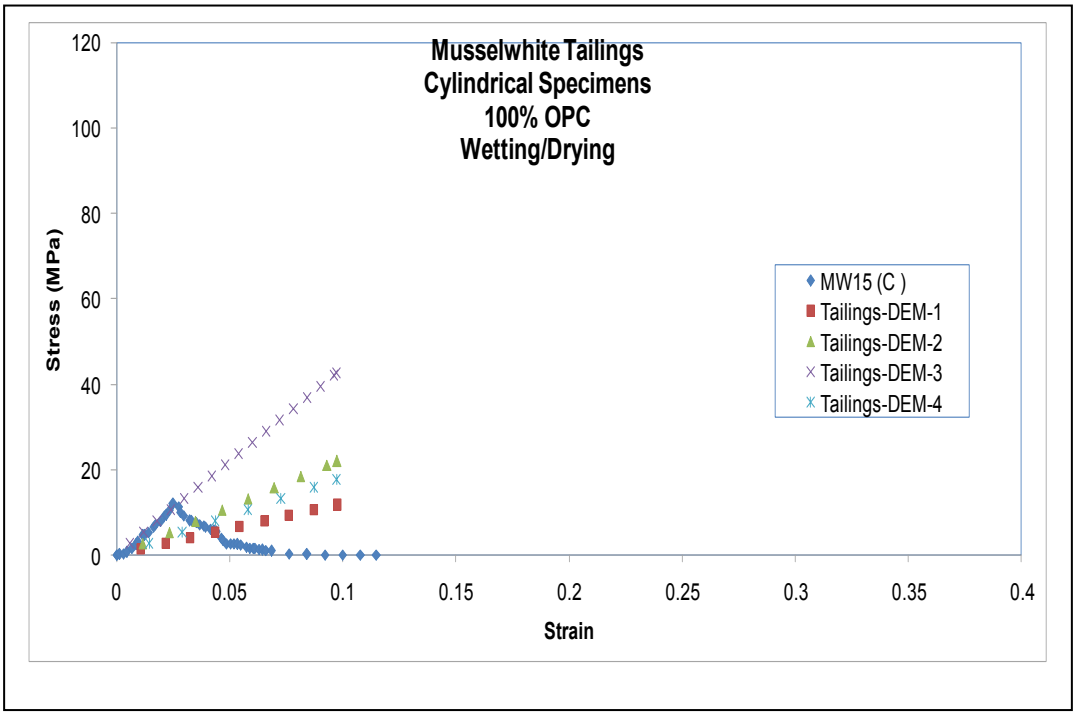


Figure 5.30 Sensitivity Analysis Test for Musselwhite samples MW14 and MW15

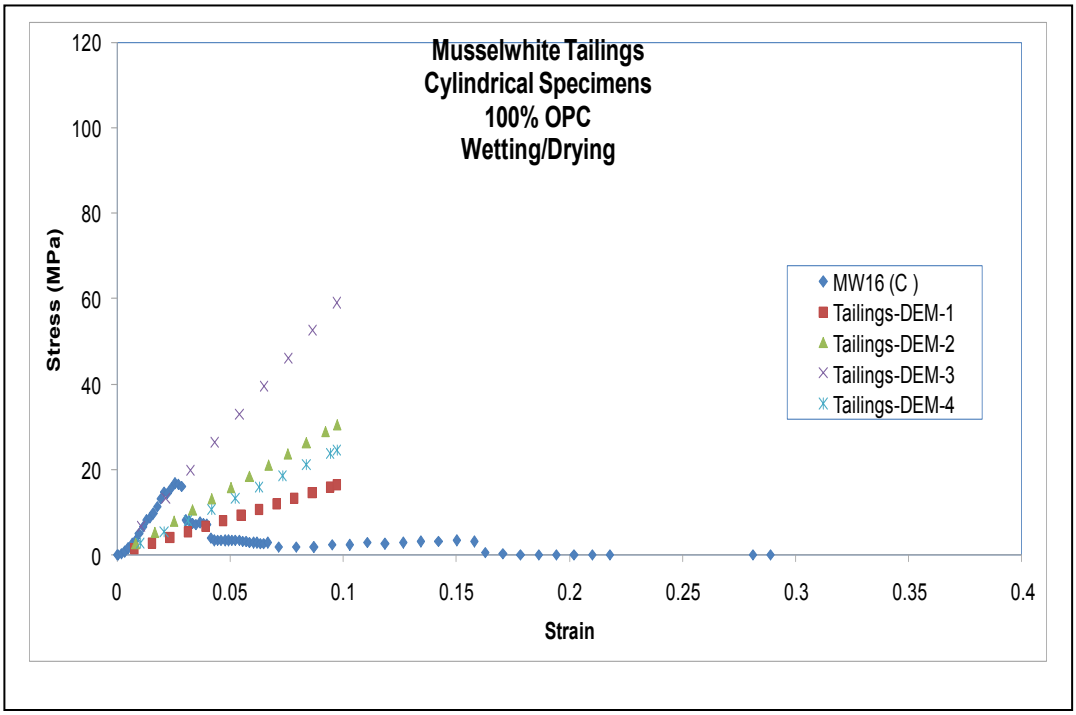


Figure 5.31 Sensitivity Analysis Test for Musselwhite sample MW16

Table 5.3 Tailings matrices used in the Sensitivity Analysis test

Sample Code	**Tailings Type**	**Specimens Type**	**Binder/Tailings (%)**	**Binder Combination (%)**
MC1	Mont Wright	Testing: MC11, MC12, MC13 Control: MC15, MC16	0.5	100% OPC
MW1	Musselwhite	Testing: MW11, MW12, MW13 Control: MW14, MW15, MW16	0.375	100% OPC

UCS values for Tailings-DEM code series: 2, 3, 4 deviated from the experimental UCSand the difference between UCS values these of these three codes and the experimental is much higher than the difference between the original particle size (series 1) and the experimental UCS. This can be seen in Figures 5.26 to 5.31. This clearly shows that the previously chosen particle size (series 1) for the triangles used in the simulation is the optimum and that Tailings-DEM™ is sensitive to the size of the particles used in relation to the size of the cylindrical specimen.

No results were produced byTailings-DEM™ code series 5.It is believed that this is due to the small number of particles produced in relation to the size of the cylindrical specimen that remained the same. Since particle number and size are related,it can be concluded that with this small number of particles the condition put in the program for contact between adjacent particles could not take effect. Contact between neighboring particles, as explained above, is the mechanism by which Equations (5.3) to (5.12) are activated. Appendices E, F, G and H show the codes for the sensitivity analysis.

5.5 Stage 3: Tailings-DEM™modeling of freezing/ thawing

Equation (5.2) was used to compute the thermal stresses in the matrix resulting from the change in temperatures during its exposure to the freezing/thawing cycles. Matrix weight loss was used as an indicator of matrix response to the freezing/thawing environment.

As has been shown in Chapter 3, the matrix was previously exposed to the freezing/thawing cycles test following ASTM D 4842 (1996) standard. Temperature range used in the computation was according to the lab test;i.e. from room temperature (22 ^{o}C) to the freezing temperature of-20 ^{o}C. Results shown in Chapter 4 prove that all tailings matrices tested fared well in this test as the maximum value of the cumulative mass loss did not exceed 0.8%.

The thermal stress resulting from the change in the specimen temperature is found according to Equation (5.2) ($\sigma = E\ \alpha\ \Delta t$) as shown above. The coefficient of thermal expansion used in this equation is a measure of the materials expansion or contraction with temperature. Because of the relatively small length changes associated with thermal expansion, it is usually expressed in terms of micro strains per unit temperature change. The United States Federal Highway Administration employs a value range of (18-20) x 10^{-6}/ ºC for the coefficient of thermal expansion of saturated cement pastes (FHWA 2011). The average of this range, 19 x 10^{-6}/ ºC, is used in calculating the thermal stresses in this study.

The modulus of elasticity (E) for each tailing matrix is assumed equal to the slope of the linear portion of the stress versus strain experimental diagram of that matrix. This value of the modulus is used in Equation (5.2) to find the thermal stress.

Once the thermal stress is found according to Equation (5.2), it is multiplied by the cross sectional area of the cylindrical specimen to get the maximum thermal load. Dimensions of the cylindrical specimenused here are the same as in Chapter 4(44 *mm* diameter and 74 *mm* height). These values represent the actual dimensions used experimentally.

Subsequently, this load is used in finding thechange in length (ΔL) associated with the computational results, which represents the length response of the specimen to the vertical load imposed on it during the UCS test. This change in length divided by the initial specimenlength is

assumed here to represent the weight loss from freezing and thawing thermal stresses expressed in *MPa*.The change in length (ΔL) is found by graphically reading off the displacement corresponding to the thermal load found from Equation (5.2) for the tailings matrix in question. Thecomputational weight loss for the tailings matrix equals this displacement divided by the length of the specimen (74 *mm*).

Table 5.4 shows the computational percentage weight loss found using the procedure explained and its comparison with the experimental weight loss results for the freezing/thawing tests on the tailings matrices. The % difference is found by subtracting the computational weight loss from the experimental and dividing the result by the experimental weight loss.Subsequently, the factor of safety is found by dividing the experimental on the computational weight loss.

Table 5.4a Experimental and computational weight loss in the freezing/thawing test

Specimen (description in Chapter 3)	Computational Weight Loss (%)	Average Computational Weight Loss (%)	Experimental Weight Loss (%)	Difference (%)	Factor of Safety
MC'11	0.27				
MC'12	0.28				
MC'13	0.26	0.27	0.6	54.58	2.2
MC'14	0.27				
MC'15	0.34				
MC'16	0.24	0.28	0.65	56.63	2.31
MC'31	0.26				
MC'33	0.3	0.28	0.49	43.26	1.76
MC'34	0.29				
MC'35	0.37				
MC'36	0.15	0.27	0.62	56.72	2.31
MC'61	0.34				
MC'62	0.37				
MC'63	0.39	0.37	0.49	25.46	1.34
MC'64	0.23				
MC'65	0.26				
MC'66	0.26	0.25	0.62	60.14	2.51
MCF'51	0.28				
MCF'52	0.32				
MCF'53	0.30	0.30	0.46	33.7	1.51
MCF'54	0.27				
MCF'55	0.32				

MCF'56	0.37	0.32	0.54	40.98	1.69
MCS'11	0.38				

Table 5.4b Experimental and computational weight loss in the freezing/thawing test

Specimen (description in Chapter 3)	Computational Weight Loss (%)	Average Computational Weight Loss (%)	Experimental Weight Loss (%)	Difference (%)	Factor of Safety
MCS'12	0.26				
MCS'13	0.32	0.32	0.42	24.61	1.33
MCS'14	0.38				
MCS'15	0.36				
MCS'16	0.24	0.33	0.47	30.11	1.43
MCS'31	0.31				
MCS'32	0.34				
MCS'33	0.38	0.34	0.44	21.41	1.27
MCS'34	0.3				
MCS'35	0.26				
MCS'36	0.26	0.27	0.53	48.6	1.95
MCS'51	0.3				
MCS'52	0.37				
MCS'53	0.28	0.32	0.61	48.37	1.94
MCS'54	0.3				
MCS'55	0.33				
MCS'56	0.31	0.31	0.64	50.89	2.04
MW'11	0.45				
MW'12	0.39				
MW'13	0.41	0.41	0.51	18.85	1.23
MW'14	0.37				
MW'15	0.35				
MW'16	0.43	0.38	0.56	31.05	1.45
MW'31	0.37				

Table 5.4c Experimental and computational weight loss in the freezing/thawing test

Specimen (description in Chapter 3)	Computational Weight Loss (%)	Average Computational Weight Loss (%)	Experimental Weight Loss (%)	Difference (%)	Factor of Safety
MW'32	0.43				
MW'33	0.27	0.35	0.49	28.13	1.39
MW'34	0.45				
MW'35	0.34				
MW'36	0.42	0.40	0.57	29.14	1.41
MW'61	0.33				
MW'62	0.33				
MW'63	0.29	0.31	0.71	55.59	2.25
MW'64	0.24				
MW'65	0.25				
MW'66	0.41	0.3	0.76	60.45	2.53
MWF'11	0.39				
MWF'12	0.43				
MWF'13	0.42	0.41	0.44	4.79	1.05
MWF'14	0.43				
MWF'15	0.37				
MWF'16	0.47	0.43	0.45	5.19	1.05
MWF'31	0.39				
MWF'32	0.35				
MWF'33	0.43	0.39	0.58	32.48	1.48
MWF'34	0.42				

Table 5.4d Experimental and computational weight loss in the freezing/thawing test

Specimen (description in Chapter 3)	Computational Weight Loss (%)	Average Computational Weight Loss (%)	Experimental Weight Loss (%)	Difference (%)	Factor of Safety
MWF'35	0.43				
MWF'36	0.5	0.45	0.55	18.52	1.23
MWF'51	0.47				
MWF'52	0.37				
MWF'53	0.36	0.398	0.60	33.73	1.51

Table 5.4e Experimental and computational weight loss in the freezing/thawing test

Specimen (description in Chapter 3)	Computational Weight Loss (%)	Average Computational Weight Loss (%)	Experimental Weight Loss (%)	Difference (%)	Factor of Safety
MWF'54	0.42				
MWF'55	0.39				
MWF'56	0.41	0.41	0.77	47.36	1.9
MWS'11	0.45				
MWS'12	0.32				
MWS'13	0.38	0.38	0.32	-20.40	1
MWS'14	0.55				
MWS'15	0.46				
MWS'16	0.32	0.45	0.35	-28.08	1
MWS'31	0.31				
MWS'32	0.3				
MWS'33	0.31	0.31	0.36	14.02	1.16
MWS'34	0.57				
MWS'35	0.61				
MWS'36	0.45	0.54	0.44	-24.36	1
MWS'51	0.37				
MWS'52	0.32				
MWS'53	0.28	0.32	0.5	34.52	1.53
MWS'54	0.38				
MWS'55	0.38				
MWS'56	0.29	0.35	0.64	45.58	1.84

5.5.1 Discussion of Tailings-DEM™ modeling of freezing/thawing

Based on the results shown in Table 5.4, it is evident that differences in the freezing/thawing weathering results are within an average factor of safety of 2. The experimental values were higher than the computational for all Mont Wright and most Musselwhite matrices tested. The Musselwhite matrices with the following binder combinations had higher computational weight loss values than experimental: 75% OPC and 25% slag (testing and control) and 75% OPC, 15% slag, 10% CF (control).

The reason behind this discrepancy could be attributed to the higher ratio of particles smaller than 75 *micrometer* that Musselwhite tailings contain as revealed in the particle size distribution results shown Table 4-1. This fact coupled with slag's average specific gravity of 2-3 (Chapter 3), may have resulted in more matrix voids filled by slag particles during the mixing stage rendering the matrix more resistant to freezing and thawing weathering effects. It is also seen that when the binder content contained 25% of slag, weight loss was lower for all Musselwhite and Mont Wright matrices, for both testing and control specimens. This is clearly evident in Figures 4-13, 4-14, 4-15 and 4-16. Musselwhite tailings matrix with 25% slag had 1 *mg/L*Fe content when tested in the TCLP test as shown in Table 4-5. Compared to this, no Fe content was detectable with the equivalent Mont Wright tailings matrix. Research by Al-Otaibi (2008) who used steel mill (iron oxides) as fine aggregate in cement mortars, found that drying shrinkage was lower when using steel mill scale. In addition, he showed that replacing 40% of sand with steel mill gave the highest increase in compressive strength and increased flexural strength. This

may have contributed to Musselwhite matrices having less % weight loss than their Mont Wright counterparts. The US Federal Highway Administration report research by Malhotra (1983) who did freeze thaw tests on concrete incorporating 25-65% slag. Test results indicated that regardless of the water- to- (cement + slag) ratio, air entrained concrete slag specimens performed excellently in freeze thaw tests, with relative durability factors greater than 91% (FHWA 2011).

There is not much deviation between the experimental and computational results for the Musselwhite tailings matrix when the binder content is composed of 75% OPC and 25% fly ash as can be seen in Table 5.4. This is explained by the fact that finding computationally the weight loss was based on the stiffness coefficient obtained experimentally. It can be seen in Figure A-31 that all 6 specimens of this matrix combination had very close strength and stiffness values, hence closing the gap between experimental and computational results.

Other reasons for the differences between computational and experimental results can be attributed to the particle shape used in the Tailings-DEM™ program (triangular). Changing the particle shape to a pentagon or a more circular shape may lead to the results being more closely related. Reducing their size and increasing their number may also contribute to more accurate results.

The above suggests that Tailings-DEM™ can be used to account for and predict freezing and thawing weathering effects, as described by the procedure in Chapter 3, with confidence. However, it should be used with caution when there is a detectableFe content in these matrices coupled with slag binder content between 25% - 10% in the matrix.

5.6 General discussion

Experimental and computational UCS values for the tailings matrices have similar types of curves indicating that computational modeling of this test was to a large extent accurate. Marginal differences between the experimental and computational values could be attributed to several factors.

The main difference between the computational and the experimental programs is the approximation to 2D modeling when designing the program. Also, binder constituents were largely modeled through the UCS in Equation (4-14) that was derived statistically. As explained in Chapter 4, this equation found, out of the four binders used, the ratio of Calsifrit and slag to be most effective when calculating the UCS. The effect of cement could not be associated directly with the UCS.Its effect,however,was evident in conjunction with the other combinations, namely, the binder/tailings ratio. Also, the experimental tests used a certain platen speed that was not modeled through the computational program.

The computer program is capable of modeling other types of waste materials and binders. The matrix mass coefficient shown above can be used with other types of wastes. By first finding the experimental UCS for the waste matrix in question, the matrix mass coefficient can be incorporated into the program and used as shown above.

Since the both Mont Wright and Musselwhite tailings were predominantly sandy, the program can be incorporated to model other types of aggregates, such as sands and gravely sands. Tailings-DEM™ can be utilized to model sands with a particle size distribution close to Mont Wright and Musselwhite tailings.In addition, it can also be used with the other 4 tailings; Louvicourt, Noranda, Copper and Golden Giant tailings matrices.

5.7 Tailings-DEM™ applications

The program developed above can be applied to the following types of waste materials:

1) Demolition waste,
2) Coal mine refuse,
3) Chat.

5.7.1 Demolition waste

The Ministry of Industry Canada defines the construction and demolition waste as “waste materials from the construction and demolition of roads, bridges and buildings such as wood

gypsum and metal", (Saotome 2007). Hence most tailings components are common with demolition waste.

Construction and demolition debris are one of the most abundant types of man-made wastes in Canada. For example in Alberta, construction and demolition waste accounts for about 25% of the total amount of municipal solid waste sent to landfill (Alberta Environment 2011).

Compressive strength for construction and demolition debris cubes at the age of 28 days range from 11.05 to 28.25 *MPa*(Kumutha and Vijay 2010). This makes them structurally close, in terms of strength, to the Mont Wright and Musselwhitetailings being modeled in this study.

The matrix mass coefficient for the construction and demolition debris samples can be obtained from these studies and used in the current computer program along with the stiffness coefficient. The same study produced experimental values of the modulus of elasticity, (Kumutha and Vijay 2010). These values can be substituted for the stiffness coefficient as the first input to the program.

5.7.2 Coal mine refuse

Coal refuse is a by-product of the coal mining industry. Coal refuse is generally defined by a minimum ash content combined with a maximum heating value measured on a dry basis

(EPA2008). The material consists primarily of non-combustible rock with some attached carbon material that cannot be effectively separated (EPA 2008).

Coal continues to be one of the primary sources of energy in several countries including the United States where mining of coal results in the production of large quantities of coal refuse. It is estimated that over 500 million *tons*of coal mining and preparation refuse are produced annually in the United States (Kumar et al. 2001).

Cementation and stabilization of coal mine refuse for proper re-use in construction applications removes most of the harmful environmental consequences resulting from land filling.

Kumar et al. (2001) report that several studies have been conducted with the purpose of investigating the use ofstabilized and un-stabilized mixtures of coal mine refuse and fly ash, the two most important by-products of the coal industry, for construction of highway embankments and base courses (Butler 1974, Moulton et al. 1974, Wilmoth and Scott 1974, Drake 1976, Head et al. 1982 and EPRI 1989).

The matrix mass coefficient can be produced from the compressive strength of coal refuse mixtures. Crandall (1992) reports that Knissel and Helms (1983) provided values of uniaxial compressive strength of coal mine washeryrefuse samples up to 4MPa. These values were the result of 28 days curing with 10% cement (Crandall 1992).

The matrix mass coefficient defined in this study can be successfully used withcement stabilized coal mine refuse, after experimentally finding the proper UCS values and stiffness coefficients as defined above.

5.7.3 Chat

Mine tailing piles (chat) are comprised mainly of angular chert fragments and contain residual amounts of lead sulfide (galena), zinc sulfide (sphalerite) minerals and their weathering products (ODEQ 2003). The difference between chat and tailings material is that chat is a fine to coarse dolomite produced during the early milling process that used density separation (US Department of Health and Human Services 1998). Chat was transported mechanically by conveyor and deposited in large piles. Tailings were produced in the later years using a wet chemical process and are typically smaller fragments than chat (US Department of Health and Human Services 1998). The tailings were hydraulically deposited into impoundments known as tailings ponds (EPA 1994).

Chat has been used for the stabilization of roadway bases with additives such as fly ash and cement kiln dust for stabilization/solidification purposes (Wasiuddin et al 2005).

An experimental study by Wasiuddin et al. (2005) to optimize the use of chat in hot mix asphalt for pavement application showed great potential in theuse of chat for both surface and base course applications.

Teredesai et al. (2005) reported UCS values up to 5.2 *MPa*for cylindrical specimens of chat stabilized with class C fly ash and cement kiln dust and cured for 28 days.

The matrix mass coefficient can be used in conjunction with chat mechanical properties to optimize the use of chat with other binders and determine its suitability for construction applications. Higher binder content may need to be used with the angular chat fragments.

Chapter 6

SUMMARY AND CONCLUSIONS

6.1 Conclusions from Phase 1

6.1.1 Formulation of tailings binder matrices and their properties

- Using Calsifrit as a Portland cement replacement increased the values of the layer coefficients for both tailings; Mont Wright and Musselwhite. Layer coefficients in this case were higher than the cement-alone layer coefficients; mixtures with 10% and 25% Calsifrit/cement ratio had higher layer coefficients in comparison with 100% cement samples for both wetting/drying and freeze/thaw samples,

- Strength decreased with increasing water content for the tailings matrices,

- Compression strength values for the 5 cm cubes went higher as the curing period was increased from 1, 7 to 28 days and as the cement was allowed to gain more of its hydration products,

- Substituting Calsifrit for fly ash and slag resulted in higher values of layer coefficients; 10% and 20% Calcifrit replacement samples had higher layer coefficients when compared to the Portland cement-slag and the Portland cement-fly ash samples,

- The general trend indicates an increase in density with the increase in Calsifrit replacement ratio,

- Musselwhite matrices had higher porosity values than Mont Wright matrices,

- Increasing the percentage of Calsifrit in the matrices has generally led to a decrease in the porosity of the matrices tested for Musselwhite tailings. As for Mont Wright tailings matrices it seems that the optimum Calsifrit percentage with the least porosity is 10% of the total weight of binder used,

- Using Calsifrit in addition to slag and fly ash as partial cement replacement reduced the porosity and improved the strength and durability of the final product,

- Mont Wright matrices had higher strength values than Musselwhite matrices for the 1 day cured cubes, these values however became closer for the 28 days curing period. The same observation can be seen for the fly ash and slag mixed specimens, where Mont Wright matrices had higher values than the same matrices of Musselwhite for the 1 day curing period. This difference in strength values between the two tailings might be due to the Musselwhite tailings content of Zn and Pb,

- The initial water content of the samples might have also been contributing to the decrease in the UCS values. The initial water content of 30.15% for Musselwhite

plus the water/cement ratio of 0.3 might have been excessive for cement hydration especially as the Calsifrit ratio of the matrices increased,

- Visual observation during the experiments revealed that increasing the amount of Calsifrit increased the fluidity of the mixes,

- The maximum dry densities for Musselwhite and Mont Wright tailings were 20.8 kN/m^3 and 16.9 kN/m^3 respectively. The better compaction characteristics of Musselwhite tailings has left less void space for water hence the optimum moisture content of 11.7% for Musselwhite was smaller than that for Mont Wright, 15.5%, which had more voids to fill among the coarser sand particles,

- Both tailings reached 71% saturation at the optimum moisture content,

- When performing the CBR tests on the tailings without binder, Mont Wright achieved a CBR of 48.5 and Musselwhite had a CBR of 80.8, which are qualified as good and excellent base materials respectively,

- Adding 25% binder by weight composed of 90% ordinary Portland Cement and 10% Calsifrit to both tailings has resulted in further increases in the CBR values, with Mont Wright achieving a CBR of 61.2, 6 hours after compaction and Musselwhite having CBR values of 103.4, 116.7 and 279 at 0, 3 and 6 hours after compaction respectively,

6.1.2 Feasibility of using formulated tailings matrices for construction

- The tailings layer coefficients match reasonably well with values from 10 state departments of transportation reported in the literature thus indicating that these mine tailings matrices are suitable structurally for road construction,

- Mont Wright tailings matrices had higher UCS values than their Musselwhite counterparts for the same mixtures. One reason for the increase of Mont Wright matrices strength is the increase in cement to tailings ratio as compared to Musselwhite: Mont Wright had a cement/wet tailings ratio of 0.5 compared to Musselwhite's 0.375. Another reason could be the presence of Zn in Musselwhite tailings. Previous studies have shown that Zn has a retarding effect in hydration of Portland cement,

- A comparison between the values of standard deviation reported in the literature and cited above and the values found in this work indicates that compression values for this study have much less variability and the scatter is typical for cement paste mortars and therefore could be considered accurate and applicable for design and calculation purposes,

- Cylindrical strength tests, 5 cm cube strength tests, bulk density values, specific gravity of hardened tailings, freeze thaw weathering resistance tests, and TCLP test results for all matrices tested fall within Student's t 95% confidence interval.

- Multiple regression analysis used to fit the data suggests that UCS is a function of Calsifrit/binder, slag/binder and binder/tailings ratio. It was found during this analysis that the effect of cement could not be associated directly with the UCS, however its effect was evident in conjunction with the other combinations,

- Predictive equation that governs the behavior of the tailings-binder matrices was generated.

6.1.3 Resistance of the matrices to weathering (freezing/thawing and wetting/drying)

- All matrices tested for both Mont Wright and Musselwhite tailings faired well in the freezing and thawing durability test as the maximum cumulative weight loss did not exceed 0.8% during 12 cycles of freezing and thawing,

- Adding Calsifrit to the cement-only matrices had a positive effect on freezing and thawing durability for the Mont Wright testing and control matrices with weight loss decreasing with the addition of 10% and 20% Calsifrit by total weight of binder to the cement-only matrices,

- On the other hand, slag-OPC matrices faired the best with the least weight loss out of all tailings matrices tested. The 5% Fly ash -20% Calsifrit -75% OPC matrix had less weight loss than the same mixture with slag for Mont Wright tailings,

- The heavy metals with a high percentage of retention in the TCLP test are Cr, Ni, Cu, Zn and Pb.

6.2 Conclusions from Phase 2

6.2.1 Development of a Discrete Element Method program

- A numerical tool was developed to model the strength and weathering characteristics of the tailings matrices. The tool was built using the compiler Code Blocks and the computer language C++,

- The DEM tool developed (Tailings-DEM™) is capable of simulating, with great accuracy, the UCS strength values of the tailings matrices tested,

- Tailings-DEM™ is capable of modeling other types of waste materials and binders. By using the matrix mass coefficient, other types of wastes can be modeled. These wastes might include: demolition waste, coal mine refuse and chat,

- Tailings-DEM™ can be incorporated to model other types of aggregates, such as sands and gravely sands.

6.2.2 Simulating the vulnerability of tailings binder matrices

- Simulating the vulnerability of tailings binder matrices to cold climate, through the freezing/thawing test, using the Discrete Element Method was successful,

- Differences between the experimental and computational freezing/thawing weathering results are within an average factor of safety of 2.

6.3 Contribution to knowledge

1. Formulation of new tailings matrices which use local waste, decrease energy consumption, decrease impact on environment and are resistant to cold climate,

2. Determination of the predictive equation linking UCS with binder and pozzolanic materials and tailings content,

3. Development of a tool permitting the simulation of complex matrices' properties,

4. Development of a tool permitting the prediction of weathering on matrices properties and integrity,

5. Verification of the new tool for new materials applied in road construction.

6.4 Recommendations for future research

1. A detailed investigation into the behavior of mine tailings matrices using dynamic tests such as the plate load tests and cyclic vibrations for monitoring the reaction to dynamic stresses of the tailings matrices in situ. These tests will give a detailed account of the dynamic structural response of these matrices,

2. Further development of Tailings-DEM™ to account for 3D modeling, leaching characteristics, moisture effects including surface tension and ice lensing effects in the freezing/thawing tests,

3. Life cycle cost analysis involving the tailings matrices,

4. Effects of chemical components of binders on UCS in Tailings-DEM™,

5. Verification of Tailings-DEM™ using other types of tailings matrices and binder combinations.

REFERENCES

AASHTO. 1986. Guide for design of pavement structures. American Association of State Highway and Transportation Officials (AASHTO), Washington, D.C.

ACI Committee 116. 1994. 116R-90, Cement and concrete terminology, ACI manual of concrete practice.

ACI Committee 230. 1990. State of the art report on soil cement. ACI Materials Journal, **87**(4): 395-417.

Adaska, W.S., Bruin, W.T., and Day, S.R. 1992. Remediation of oil refinery sludge basin. *In* Proceedings of Cement Industry Solutions to Waste Management, Calgary, Alberta, 7-9 October 1992. Edited by Robert Piggott.

Adu-Wusu, C., Yanful, E.K. and Mian, M.H. 2000. Measurement of wind induced resuspension of tailings at Quirke tailings site, Elliot Lake. *In* Proceedings of the 53rd Canadian Geotechnical Conference, Canadian Geotechnical Society, 15-18 October 2000, Montreal, Quebec, Vol. 1, pp. 641-647.

Alberta Environment. 2011. Construction and demolition waste reduction program. Technical Info, Alberta Environment Pollution Prevention and Conservation, www.environment.alberta.ca/info

Aldridge, C.W. and Naguib, A. 1992. In situ mixing of dry and slurried reagents in soil and sludge using shallow soil mixing. *In* Proceedings of the 85th Annual Meeting and Exhibition Geo-Con, Inc., Kansas City, Missouri, 21-26 June 1992.

Al-Otaibi, S. 2008. Recycling steel mill scale as fine aggregate in cement mortars. European Journal of Scientific Research, **24**(3): 332-338.

Amjad, A.A. 2005. Cost benefit analysis for construction projects. IEP-SAC Journal, **2004-2005**: 85-90.

Amuda, O.S. and Ibrahim, A.O. 2006. Industrial wastewater treatment using natural material as adsorbent. African Journal of Biotechnology, **5**(16): 1483- 1487.

Anandarajah, A. and Yao, M. 2002. Three-dimensional Discrete Element Method of analysis of clays. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen. pp. 237-241.

Andromalos, K.B. and Ameel, M.E. 1994. In-situ stabilization of the Geiger (C & M Oil) superfund site. *In* Proceedings of SUPERFUND XV Conference and Exhibition, HMCRI. 29 November- 1 December 1994. Washington, D.C.

Andromalos, K. B., Hegazy, Y. A., and Jasperse, B. H. 2000. Stabilization of soft soils by soil mixing. *In* Proceedings of Soft Ground Technology Conference, United Engineering Foundation and ASCE Geo-Institute. May 28- June 2 2000, Noorwijkerout, the Netherlands.

Ann, K.Y., Moon, H.Y., Kim, Y.B. and Ryou, J. 2008. Durability of recycled aggregate concrete using pozzolanic materials. Waste Management, **28**(6): 993-999.

Asavapisit, S., Fowler, G., and Cheeseman, C.R. 1997. Solution chemistry during cement hydration in the presence of metal hydroxide wastes. Cement and Concrete Research, **27**(8): 1249–1260.

ASTM C 109. 1992. Standard test method for compressive strength of hydraulic cement mortars (using 2-in. or 50 mm cube specimens). Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.01.

ASTM C 109/C 109 M. 1999. Standard test method for compressive strength of hydraulic cement mortars (using 2-in. or 50 mm cube specimens). Annual Book of ASTM

Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.01.

ASTM C 125. 2007. Standard terminology relating to concrete and concrete aggregates. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.02.

ASTM C 305. 1999. Standard practice for mechanical mixing of hydraulic cement pastes and mortars of plastic consistency. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.01

ASTM C 618-08a. 2008. Standard specification for coal fly ash and raw or calcined natural pozzolan for use in concrete. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.02.

ASTM C 989. 2006. Standard specification for ground granulated blast-furnace slag for use in concrete and mortars. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.02.

ASTM D 421-85. 2002. Practice for dry preparation of soil samples for particle-size analysis and determination of soil constants. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.08.

ASTM D 422-63. 2002. Standard test method for particle size analysis of soils. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.08.

ASTM D 854. 1998. Standard test method for specific gravity of soils. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.08.

ASTM D 1557. 2007. Standard test methods for laboratory compaction characteristics of soil using modified effort (56,000 ft-lbf/ft^3 (2,700 kN-m/m^3)). Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.08.

ASTM D 1883. 2007. Standard test method for CBR (California Bearing Ratio) of laboratory-compacted soils. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.08.

ASTM D 2216. 1998. Standard test method for laboratory determination of water (moisture) content of soil and rock by mass. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 04.08.

ASTM D 4842-90. 1996. Standard test method for determining the resistance of solid wastes to freezing and thawing. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 11.04.

ASTM D 4843-88. 2004. Standard test method for wetting and drying test of solid wastes. Annual Book of ASTM Standards, American Society for Testing and Materials, Philadelphia, Pennsylvania. Vol. 11.04.

Atis, Cengiz Duran and Bilim, C. 2007. Wet and dry cured compressive strength of concrete containing ground granulated blast-furnace slag. Building and Environment, **42** (8): 3060–3065.

Bangash, T. and Munjiza, A. 2002. A computationally efficient beam element for FEM/DEM simulations of structural failure and collapse. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Barich, J.J., Greene, J. and Bond, R. 1987. Soil stabilization treatability study at the western processing superfund site. *In* Proceedings of the 8th National Conference: Superfund' 87. 16-18 November 1987. Washington, D.C. pp. 198-203.

Bathurst, R.J. and Rothenburg, L. 1989. Investigation of micromechanical features of Idealized granular assemblies using DEM. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Baxter, C.W., Smith, D.W., and Stanley, S. J. 2004. A comparison of artificial neural networks and multiple regression methods for the analysis of pilot scale data. Journal of Environmental Engineering and Science, **3**(S1): S45-S58.

Bell, F.G. 1976. The influence of the mineral content of clays on their stabilization by cement. Bulletin. Association of Engineering Geologists, **13**(4): 267–278.

Benzaazoua, M., Belem, T., and Bussiere, B. 2002. Chemical factors that influence the performance of mine sulphidic paste backfill. Cement and Concrete Research, **32**(7): 1133-1144.

Benzaazoua, M., Bussiere, B., Demers, I., Aubertin, M., Fried, E., and Blier, A. 2008. Integrated mine tailings management by combining environmental desulphurization and cemented paste backfill: Application to mine Doyon, Quebec, Canada. Minerals Engineering, **21**(4): 330–340.

Bhandari, A. and Han, j. 2010. Investigation of geotextile–soil interaction under a cyclic vertical load using the Discrete Element Method. Geotextiles and Geomembranes, **28**(1): 33–43.

Bhatty, J.I., Miller, M.F., West, P.B., and Ost, B.W. 1999. Stabilization of heavy metals in Portland cement, silica fume/Portland cement and masonry cement matrices. Portland Cement Association, Skokie, Illinois. PCA R&D Serial No. 2067.

Bijen, J. 1996. Benefits of slag and fly ash. Construction and Building Materials, **10**(5): 309–314.

Boutt, D. and McPherson, B. 2002. The role of particle packing in modeling rock mechanical behavior using discrete elements. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Bowles, Joseph E. 1986. Engineering properties of soils and their measurement, 3rd Edition. McGraw-Hill book company, New York.

Boyer, R. 2002. Concepts in Biochemistry, 2nd Edition. John Wiley and Sons Publishers, Inc. New Jersey.

Butler, P.E. 1974. Utilization of coal mine refuse in the construction of highway embankments. *In* Proceedings of the First Symposium on Mine and Preparation Plant Refuse Disposal. Louisville, Kentucky. pp. 237 –255.

Campbell, K.M., El-Korchi, T., Gress, D. and Bishop, P. 1987. Stabilization of cadmium and lead in Portland cement paste using a synthetic seawater leachant. Environmental Progress, **6**(2): 99-103.

Celik, O., Elbeyli, I.Y., and Piskin, S. 2006. Utilization of gold tailings as an additive in Portland cement. Waste Management and Research, **24**(3): 215-224.

Cheilas, A., Katsioti, M., Georgiades, A., Malliou, O., Teas, C., and Haniotakis, E. 2007. Impact of hardening conditions on to stabilized/solidified products of cement–sewage sludge–jarosite/alunite. Cement and Concrete Composites, **29**(4): 263–269.

Chen, F., Drumm, E.C. and Guiochon, G. 2009. 3D DEM analysis of graded rock fill sinkhole repair: particle size effects on the probability of stability. *In* Proceedings of the 88th Annual Meeting of the Transportation Research Board. 11-15 January 2009. Transportation Research Board. Washington, DC.

Cheng, K.Y. and Bishop, P.L. 1990. Developing a kinetic leaching model for solidified/stabilized hazardous wastes. Journal of Hazardous Materials, **24**(2-3): 213-224.

Cheng, Y.P. Bolton, M.D. and Nakata, Y. 2002. The modelling of soil plasticity. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Collins, R.J. and Ciesielski, S.K. 1992. Highway construction use of wastes and by-products. *In* Proceedings of Utilization of Waste Materials in Civil Engineering Construction, ASCE. 13-17 September 1992. New York, NY. pp 140-152.

Conner, J.R. 1974. Ultimate disposal of liquid wastes by chemical fixation. *In* Proceedings of the 29th Annual Purdue Industrial Waste Conference. Purdue University, West Lafayette, Indiana. pp. 906-922.

Conner, J. 1990. Chemical fixation and solidification of hazardous wastes. Van Nostrand Reinhold, New York.

Cook, Benjamin K., Noble, David R., and Williams, John R. 2002. A coupled DEM-LB model for the simulation of particle-fluid systems. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Cote, P.L. 1986. Contaminant leaching from cement-based waste forms under acidic conditions. Ph.D. thesis, McMaster University, Hamilton, Ontario.

Crandall, Wallace E. 1992. SME Mining Engineering handbook, Volume 2, Chapter 19.3. The Society for Mining Metallurgy and Exploration, Englewood, Colorado.

Cross, S.A. and Fager, G.A. 1995. Fly ash in cold recycled bituminous pavements. Transportation Research Record Issue Number: 1486, Environmental Testing and Evaluation of Stabilized Wastes, Performance of Stabilized Materials and New Aggregate Tests, Transportation Research Board. ISSN: 0361-1981, pp. 49-56.

Crowder, J.J., Grabinsky, M.W.F., and Landriault, D.A. 2000. Consolidation testing and S.E.M. images of tailings pastes for surface disposal. *In* Proceedings of the 53rd Canadian Geotechnical Conference, Canadian Geotechnical Society. 15-18 October 2000. Montreal, Quebec. pp. 625-632.

Cullinane, M.J., Jones, L.W., and Malone, P.G. 1986. Handbook for stabilization/solidification of hazardous waste. U.S. Environmental Protection Agency, Hazardous Waste Engineering Research Laboratory (HWERL), EPA/540/2-86/001.

Cundall, P.A. 1971. A computer model for simulating progressive large scale movements in block rock systems. *In* Proceedings of the Symposium of the International Society of Rock Mechanics, Nancy, France. Article 8.

Cundall, P.A. 1974. A computer model for rock-mass behavior using interactive graphics for the input and output of geomechanical data. Report for Contract number DACW45-74-C-0066, for the Missouri River Division, US Army Corps of Engineers, University of Minnesota, Minneapolis, Minnesota.

Cundall, P.A. 1976. Explicit finite-difference method in geomechanics. *In* Proceedings of the 2nd International Conference on Numerical Methods in Geomechanics, Blacksburg, Virginia. pp. 132-150.

Cundall, P.A., and Strack O.D.L. 1979*a*. A discrete numerical model for granular assemblies. Geotechnique, **29**(1): 47-65.

Cundall, P. A., and Strack O. D. L. 1979*b*. The development of constitutive laws for soil using the Distinct Element Method. *In* Proceedings of Numerical Methods in Geomechanics, Rotterdam, Balkema. Vol. 1, pp. 289-317.

Cundall, P.A., and Strack O.D.L., 1979*c*. The distinct element method as a tool for research in granular media. Part II, Report to the National Science Foundation,

Department of Civil and Mineral Engineering, University of Minnesota, Minneapolis, Minnesota.

Cundall, P.A., and Strack O.D.L. 1983. Modeling of microscopic mechanisms in granular material. *In* Proceedings of Mechanics of Granular Materials: New Method and Constitutive Relations, (Eds. J.T. Jenkins and M. Satake), Elsevier Scientific, Amsterdam, pp. 137- 149.

Cundall, P.A. 1988. Computer simulation of dense sphere assemblies. *In* Proceedings of Micromechanics of Granular Materials (Eds. J.T. Jenkins and M. Satake), Elsevier Science, Amsterdam, The Netherlands. pp. 113- 123.

Cundall, P.A., and Strack, O.D.L. 1989. Numerical experiment on localization in frictional materials. Archive of Applied Mechanics, **59**(2): 148- 159.

Cundall, Peter A. 2002. A discontinuous future for numerical modeling in soil and rock. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Das, Braja M. 2000. Fundamentals of geotechnical engineering. Brooks/Cole Publishing Company, California.

Demers, B., and Haile, G. 2003. Management of tailings stabilized by lime and cement at Canadian electrolytic zinc, Valleyfield, Quebec. *In* Proceedings of Sudbury 2003 Mining and the Environment. 25-28 May, Sudbury, Ontario.

Diez, J.M., Madrid, J. and Macias, A. 1997. Characterization of cement-stabilized Cd waste. Cement and Concrete Research, **27**(3): 337-343.

Dobry R. and Ng, T.-T. 1992. Discrete modeling of stress-strain behavior of granular media at small and large strains. Engineering Computations, **9**(2): 129-143.

Double, D.D. and Hellawell, A. 1977. The solidification of Portland cement. Scientific American, **237**(1): 82-90.

Drake, W.B. 1976. Coal refuse in highway embankment and aggregates. *In* Proceedings of the 2nd Kentucky Coal Refuse Disposal and Utilization Seminar, University of Kentucky, Lexington, Kentucky. pp. 17–19.

EPA 1994. Design and evaluation of tailings dams. Technical Report, EPA 530-R-94-038, US Environmental Protection Agency, Ariel Rios Building, 1200 Pennsylvania Ave. NW Washington, DC 20004.

EPA 2008. Coal refuse. Materials Characterization Paper in Support of the Advanced Notice of Proposed Rulemaking –Identification of Nonhazardous Materials That Are

Solid Waste. US Environmental Protection Agency, Ariel Rios Building, 1200 Pennsylvania Ave. NW Washington, DC 20004.

Ercikdi, B., Cihangir, F., Kesimal, A., Deveci, H., and Alp, I. 2010. Effect of natural pozzolans as mineral admixture on the performance of cemented-paste backfill of sulphide-rich tailings. Waste Management and Research, **28**(5): 430–435.

Ercikdi, B., Kesimal, A., Cihangir, F., Deveci, H., and Alp, I. 2009. Cemented paste backfill of sulphide-rich tailings: importance of binder type and dosage. Cement and Concrete Composites, **31**(4): 268–274.

Erdogan, TY. 2003. Concrete. Metu Press (in Turkish), Ankara, Turkey.

EPRI 1989. Ash utilization in highway: Pennsylvania demonstration project. Electric Power Research Institute. EPRI Report No. GS – 6431. June 1989. Palo Alto, California.

Fahey, M., Newson, T.A., and Fujiyasu, Y.V. 2002. Engineering with tailings. *In* Proceedings of the 4th International Congress on Environmental Geotechnics, 11-15 August 2002. Rio de Janeiro, Brazil. pp. 947-973.

Fall, M., Benzaazoua, M., and Ouellet, S. 2005. Experimental characterization of the influence of tailings fineness and density on the quality of cemented paste backfill. Minerals Engineering, **18**(1): 41–44.

Fall, M. and Pokharel, M. 2010. Coupled effects of sulphate and temperature on the strength development of cemented tailings backfills: Portland cement-paste backfill. Cement and Concrete Composites, **32**(10): 819–828.

Fall, M., Benzaazoua, M., and Saa, E.G. 2008. Mix proportioning of underground cemented tailings backfill. Tunnelling and Underground Space Technology, **23**(1): 80–90.

Favier, J.F. and Kremmer, M. 2002. Modeling a particle metering device using the Finite Wall Method. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Felt, E.J., and Abrams, M.S. 1957. Strength and elastic properties of compacted soil-cement mixtures. Bulletin D16, Portland Cement Association (PCA), Skokie, Illinois.

Feng, Y.T., Han, K., and Owen, D.R.J. 2002. An advancing front packing of polygons, ellipses and spheres. *In* Proceedings of the 3rd International Conference on Discrete

Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Feng, Y.T. and Owen, D.R.J. 2002. An energy based corner to contact algorithm. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Ferguson, G. 1993. Use of self-cementing fly ash as a soil stabilizing agent. Geotechnical Special Publication No. 36, ASCE, New York.

Fernández Olmo, I., Somarriba, P. and Irabien, A. 2003*a*. Mechanical and environmental properties of cement/foundry waste systems. *In* Proceedings of WASCON, International Conference on the Environmental and Technical Implication of Construction with Alternative Materials. 4-6 June 2003, San Sebastian, Spain.

Fernández Olmo, I., Chacon, E. and Irabien, A. 2003*b*. Leaching behaviour of lead, chromium and zinc in cement/metal oxides systems. Journal of Environmental Engineering. ASCE, **129**(6): 532-539.

FHWA 1995, 2007, 2011, US Federal Highway Administration, website info, http://www.fhwa.dot.gov, 1200 New Jersey Ave, SE Washington, DC 20590.

Foth and Van Dyke 2003. All season road construction cost analysis-Clark County, Wisconsin. Technical Report. Foth and Van Dyke Associates Inc., 2737 S. Ridge Road P.O. Box 19012, Green Bay, Wisconsin 54307-9012.

Fourie, A.B., Blight, G.E., and Papageorgiou, G. 2001. Static liquefaction as a possible explanation for the Merriespruit tailings dam failure. Canadian Geotechnical Journal, **38**(4): 707-719.

Fourie, A.B., and Papageorgiou, G. 2001. Defining an appropriate steady state line for Merriespruit gold tailings. Canadian Geotechnical Journal, **38**(4): 695-706.

Friedel, R. and Murray, L. 2010. Cyclic and post-cyclic laboratory test results on undisturbed samples of filter pressed mine tailings. *In* Proceedings of GEO 2010, Calgary, Alberta. pp. 1681-1688.

Garand, P., Vezina, S., and Bocking, K. 2000. Effects of flocculent on deposition of tailings sludge for upstream raise of impoundment dykes. *In* Proceedings of the 53rd Canadian Geotechnical Conference, Canadian Geotechnical Society. 15-18 October 2000. Montreal, Quebec. pp. 633-640.

Gautam, R., Yanful, E.K. and Karamanev, D.G. 2000. Laboratory scale sulphide tailings oxidation. *In* Proceedings of the 53rd Canadian Geotechnical Conference, Canadian Geotechnical Society. 15-18 October 2000. Montreal, Quebec. pp. 649-656.

Gervais, C. and Ouki, SK. 2002. Performance study of cementitious systems containing zeolite and silica fume: effects of 4 metal nitrates on the setting time, strength and leaching characteristics. Journal of Hazardous Materials, **93**(2): 187-200.

Ghodrati, M., Sims, J.T. and Vasilas, B.L. 1995. Evaluation of fly ash as a soil amendment for the Atlantic coastal plain: I. soil hydraulic properties and elemental leaching. Water, Air and Soil Pollution, **81**(3-4): 349-361.

Ghosh, A. and Sobbarao, C. 1998. Hydraulic conductivity and leachate characteristics of stabilized fly ash. Journal of Environmental Engineering. ASCE, **24**(9): 812-820.

Girard, J. 2005. Sealer options for concrete countertops. Textbook for Precast Concrete Countertops 101, the Concrete Countertop Institute, LLC. Raleigh, North Carolina.

Gosset, William S. 1908. The probable error of a mean. Biometrika, **6**(1): 1-25.

Haile, G., O'Callaghan, W.B., Hartney, T.F., and Cameron, W. 2000. Experience with thickened red mud tailings (paste) stacking tropical and temperate climates and applicability of the technology to other tailings. *In* Proceedings of the 53rd Canadian Geotechnical Conference, Canadian Geotechnical Society. 15-18 October 2000. Montreal, Quebec.

Han, J. and Bhandari, A. 2009. Evaluation of geogrid-reinforced pile-supported embankments under cyclic loading using Discrete Element Method. Geotechnical Special Publication No. 188. Editors: J. Han, G. Zheng, V.R. Schaefer, M.S. Huang. *In* Proceedings of US-China Workshop on Ground Improvement Technologies, ASCE, Orlando, Florida. pp. 73-82.

Hashash, Youssef M.A. and Ghaboussi, J. 2002. Discrete element modeling for the development of a real-time soil model in a virtual reality environment. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

HBMS 2007. Hudson Bay Mining and Smelting Co. Limited, FFTIS Dust Control Guidelines, Document No. LAI-612. Flin Flon, Manitoba.

Head, W.J., McQuade, P.V., and Anderson, R.B. 1982. Coal refuse and fly ash compositions: potential highway base course materials. Transportation Research Board. Transportation Research Record, 839: 11 – 19.

Helinski, M., Fahey, M., and Fourie, A. 2010. Coupled two-dimensional finite element modeling of mine backfilling with cemented tailings. Canadian Geotechnical Journal, **47**(11): 1187-1200.

Hills, C.D., Sollars, C.J., and Perry, R. 1993. Ordinary Portland cement based solidification of toxic wastes: the role of OPC reviewed. Cement and Concrete Research, **23**(1): 196-212.

Hilt, G.H., and Davidson, D.T. 1960. Lime fixation in clayey soils. Highway Research Board, HRB Bulletin No. 262.

Hopkins, Mark A. 2002. Discrete element modeling based on mathematical morphology. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Horpibulsuk, S., Miura, N. and Nagaraj, T.S. 2005. Clay-water/cement ratio identity for cement admixed soft clays. Journal of Geotechnical and Geoenvironmental Engineering. ASCE, **131**(2): 187-192.

Huang, A.B. and Lee, J.S. 1989. A DEM particulate calibration chamber. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

ICOLD 2003. International Commission on Large Dams. Report. 151 Boulevard Haussmann, 75008 Paris, France.

Ishibashi, I., Agarwal, T., and Ashraf, S.A. 1989. Anisotropic behavior of glass spheres by a discrete element method and laboratory experiments. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Ishibashi, I. and Hakuno, M. 1989. A DEM simulation for cliff collapse. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Itasca 1999. Itasca Consulting Group Inc., 111 Third Ave. South, Suite 450 Minneapolis, Minnesota 55401. www.itascacg.com.

Iwashita, K., Taruni, V., Casaverde, L., Vermura, D, Meguro,K. and Hakuno, M. 1988. Granular assembly simulation for ground collapse. *In* Proceedings of Mechanics of Granular Materials, (Eds. J.T. Jenkins and M. Satake), Elsevier, Amsterdam, pp. 125-132.

Jasperse, B.H. and Ryan, C.R. 1992. In situ stabilization and fixation of contaminated soils by soil mixing. *In* Proceedings of the ASCE Geotechnical Division Specialty Conference Grouting, Soil Improvement and Geosynthetics. 25-28 February 1992. New Orleans, Louisiana.

Johnson, S. and Williams, J. 2002. Formation of packing structures in discrete element modeling with disks. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Kawaguchi, Toshihiro, Kajiyama, Satoshi, Tanaka, Toshitsugu and Tsuji, Yutaka 2002. DEM simulation of 2-D fluidized bed using similarity model. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Kazemian, S., Prasad A., Huat, B. B.K., Mohammed, T.A., and Abdul Aziz, F.N.A. 2010. Effect of cement, sodium silicate, kaolinite and water on the viscosity of the grout. Scientific Research and Essays, **5**(22): 3434-3442.

Kezdi, A. 1979. Stabilized earth roads. Elsevier, Amsterdam, the Netherlands.

Khwaja, M. 1996. Discrete Element Method: micro-mechanical and large scale modeling. M.Sc. thesis, University of Massachusetts Lowell, Massachusetts.

Kishino, Y. 1988. Disc model analysis of granular media. *In* Proceedings of Micromechanics of Granular Materials, (Eds. J.T. Jenkins and M. Satake), Elsevier, Amsterdam. The Netherlands. pp 143- 152.

Kiyama, H. and Fujimura, H. 1983. Application of Cundall's discrete block method to gravity flow analysis of rock like granular material. *In* Proceedings of the Japanese Society of Civil Engineering (JSCE). Vol. 333, pp. 137- 146.

Knissel, W. and Helms, W. 1983. Strength of cemented rockfill from washery refuse results from laboratory investigations. *In* Proceedings of the International Symposium on Mining with Backfill. 7-9 June 1983. Lulea, Sweden. pp. 31-37.

Komodromos, Petros I. and Williams, John R. 2002. Utilization of Java and database technology in the development of a combined discrete and finite element multibody dynamics simulator. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Kosmatka, S.H., Kerkhoff, B., and Panarese, W.C. 2002. Design and Control of Concrete Mixtures, 14th Edition. Portland Cement Association, Skokie, Illinois.

Kuhn, M. and Mitchell, J. 1989. The modeling of soil creep with the discrete element method. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Kuhn, Matthew R. 2002. A torus primitive for particle shapes with the Discrete Element Method. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Kumar, S., Puri, V.K., Das, Braja M. and Devkota, B.C. 2001. Geotechnical properties of fly ash and lime- fly ash stabilized coal mine refuse. Electronic Journal of Geotechnical Engineering. Oklahoma State University, **6**, http://geotech.civen.okstate.edu/ejge/2001.

Kumutha, R. and Vijai, K. 2010. Strength of concrete incorporating aggregates recycled from demolition. ARPN Journal of Engineering and Applied Sciences, **5**(5): 64-71.

Lafarge North America 2007. 12950 Worldgate Dr., Ste. 500 Herndon, Virginia 20170.

LaGrega, M.D., Buckingham, P.L., and Evans, J.C. 1994. Hazardous waste management. McGraw-Hill, Inc., New York.

Lea, F.M. 1971. The chemistry of cement and concrete. Chemical Publishing Company, Inc., New York.

Lechman, J. Mustoe, G.G.W. Miller, K.T. Lu, Ning and Eccleston, K. 2002. Comparison of DEM simulations and physical experiments for direct measurement of strongly attractive particle-particle interactions. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Li, X.D., Poon, C.S., Sun, H., Lo, I.M.C., and Kirk, D.W. 2001. Heavy metal speciation and leaching behaviors in cement based solidified/stabilized waste materials. Journal of Hazardous Materials, **82**(3): 215–230.

Lin, Jeen-Shang 2002. A unified framework for discrete and continuum analysis. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Little, D.N., Males, E.H., Prusinski, J.R., and Stewart, B. 2000. Cementitious stabilization. *In* Proceedings of the 79th Millennium Rep. Series, Transportation Research Board, Washington, D.C.

MacLaughlin, Mary M. and Clapp, Kathryn K. 2002. Discrete Element Analysis of an underground opening in blocky rock: an investigation of the differences between UDEC and DDA results. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Mahmood, A.A. and Mulligan, C.N. 2001. Insights into research on mine tailings. *In* Proceedings of the International Water Association Conference on Water and Wastewater Management for Developing Countries. 29-31 October 2001. Kuala Lumpur, Malaysia.

Mahmood, A.A. and Mulligan, C.N. 2002. Liquefaction studies: a review. *In* Proceedings of the 30th Annual Conference of the Canadian Society for Civil Engineering. 5-8 June 2002. Montreal, Quebec.

Malhotra, V.M. 1983. Strength and durability characteristics of concrete incorporating a pelletized blast-furnace slag. ACI special publication SP-79. The use of fly ash, silica

fume, slag and other mineral by-products in concrete, Editor V.M. Malhotra, 892 - 921 and 923—31, American Concrete Institute, Detroit, Michigan.

Malviyah, Rachana and Chaudhary, R. 2006. Factors affecting hazardous waste solidification/stabilization: a review. Journal of Hazardous Materials, **137**(1): 267-276.

Mamlouk, M.S. and Zaniewski, J.P. 1999. Materials for civil and construction engineers. Addison-Wesley-Longman, Menlo Park, California.

Masala, S., Chan, D., Lu, H. and Chalaturnyk, R. 2002. A Java-based graphical user interface for a 2-D Discrete Element program. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Massazza, F. 1998. Pozzolana and pozzolanic cements. In Leas's chemistry of cement and concrete, 4th edition edited by P.C. Hewlett. Butterworth-Heinemann, U.K., pp. 471–635.

Matar, M.I. 2005. Modeling of Montmorillonite clay-water interactions with particle subdivisions using three dimensional Discrete Element Method. Ph.D. thesis, North Dakota State University, Fargo, North Dakota.

Matar, M.I., Katti, D.R. and Katti, K.S. 2007. Modeling the evolution of Montmorillonite clay particulate structure: A Discrete Element modeling study. *In* Proceedings of ASCE, GEO-Denver 2007: New Peaks in Geotechnics, GSP 173 Advances in Measurement and Modeling of Soil Behavior. 18-21 February 2007. Denver, Colorado.

McBride, A., Govender, I., Powell, M., and Balden, V. 2002. Three-dimensional validation of DEM using a laboratory ball mill. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen. pp. 207-211.

Mittal, H., and Morgenstern, N.R. 1975. Parameters for the design of tailings dams. Canadian Geotechnical Journal, **12**(2): 235-261.

Mori, Hiroshi, Ogawa, Yoshimi, and Cao, Guoqiang, 2002. Liquefaction analysis of river dike with Discrete Element Method. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Moulton, L.K., Anderson, D.A., Seals, R.K. and Hussian, S.M. 1974. Coal refuse: an engineering material. *In* Proceedings of the First Symposium on Mine and Preparation Plant Refuse Disposal. National Coal Association. Louisville, Kentucky. pp. 1-25.

Murakami, Takashi, Murakami, Akira, Hori, Muneo and Sakaguchi, Hide 2002. Inverse analysis of stress developed in a granular assemblage under trap-door conditions and its validation using the Discrete Element Method. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Murray, L., McLeod, H., and Plewes, H. 2000. Aging of tailings dams. *In* Proceedings of the 53rd Canadian Geotechnical Conference, Canadian Geotechnical Society. 15-18 October 2000. Montreal, Quebec.

Neville, A.M. 1995. Properties of concrete. 4th Edition, Longman Scientific and Technical, UK.

Neville, A.M. and Aitcin, P.C. 1998. High-performance concrete — an overview. Materials and Structures, **31**(206): 111–117.

Newson, T.A. and Duliere, A. 2002. The compression behaviour of structured clayey soils. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Nezami, Erfan G. and Hashash, Youssef M. A. 2002. The use of static Discrete Element Method to simulate biaxial compression test. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Ng, T.T. and Dobry, R. 1988. Computer modeling of granular soils under cyclic loading: a particulate mechanics approach. Report CE-88-03, Department of Civil Engineering, Rensselaer Polytechnic Institute, Troy, New York.

Ng, T.T. 1989. Numerical simulation of granular soil under monotonic and cyclic loading: a particulate mechanics approach. PhD Dissertation, Department of Civil Engineering, Rensselaer Polytechnic Institute, Troy, New York.

Ng, T-T. 2002. Hydrostatic boundaries in Discrete Element Methods. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element

Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Northeim, C.M. 1989. Development of a laboratory method for estimating leachate. *In* Proceedings of the International Conference on Municipal Waste Combustion. 11-14 April 1989. Hollywood, Florida. pp. 39-52.

NovaFrit International. 2006. 1200 Garnier Street, Ville Ste-Catherine, Quebec, J0L 1E0.

ODEQ 2003. Summary report of washed and unwashed mine tailings (chat) from two piles at the Tar Creek superfund site Ottawa County, Oklahoma. Technical Report, Oklahoma Department of Environmental Quality, 707 N. Robinson, Oklahoma City, Oklahoma 73102.

Ola, SA. 1975. Stabilization of Nigerian lateritic soils with cement, bitumen and lime. *In* Proceedings of the 6th Regional Conference for Africa on Soil Mechanics and Foundation Engineering. Durban, South Africa. pp. 145-152.

Olson, Jon Narayanasamy, R. Holder, Jon, Rauch, Alan and Comacho, B. 2002. DEM study of wave propagation in weak sandstone. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical

Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Oner, M. 1984. Analysis of fabric change during cyclic loading of granular soils. *In* Proceedings of the 8th World Conference on Earthquake Engineering. San Francisco, California. pp. 55-62.

Oner, A. and Akyuz, S. 2007. An experimental study on optimum usage of GGBS for the compressive strength of concrete. Cement and Concrete Composites, **29**(6): 505–514.

O'Sullivan, C. Bray, J.D. and. Riemer, M.F. 2002. 3-D DEM validation using steel balls with regular packing arrangements. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Ouki, S.K. and Hills, C.D. 2002. Microstructure of Portland cement pastes containing metal nitrate salts. Waste Management, **22**(2): 147-151.

Palomo, A., and Palacios, M. 2003. Alkali-activated cementititous materials: alternative matrices for the immobilization of hazardous wastes part II. Stabilization of chromium and lead. Cement Concrete Research, **33**(2): 289–295.

Pamukcu, S. and Hijazi, H. 1992. Improvement of fuel oil contaminated soils by additives. *In* Proceedings of the ASCE Geotechnical Division Specialty Conference Grouting, Soil Improvement and Geosynthetics. 25-28 February 1992. New Orleans, Louisiana.

Park, H.I., and Lee, S.R. 2010. Evaluation of bearing capacity for multi-layered clay deposits with geosynthetic reinforcement using Discrete Element Method. Marine Georesources and Geotechnology, **28**(4): 363–374.

Paterson, A.J.C., Salas, U.O., and Williamson, J.R.G. 2004. Hydraulic transport considerations for high density thickened copper tailings at Southern Peru Copper Corporation. *In* Proceedings of the 16th International Conference on Slurry Handling and Pipeline Transport, Hydrotransport 16. Santiago, Chile.

Peng, F.F., and Lu, Z. 1998. Polymer flocculation and coagulation for sedimentation of copper flotation tailings. *In* Proceedings of Minerals and Metallurgical Processing, Society of Mineral Engineering annual meeting. Denver, Colorado. pp. 14-20.

Petrakis, E., Dobry, R., and Ng, T. 1989. Small strain response of random arrays of elastic spheres using a non-linear distinct element procedure. *In* Proceedings of the International Symposium on Wave Propagation in Granular Media. Annual Winter Meeting of ASME. San Francisco, California. December 1989. AMD Vol. 101. pp 17-27.

Petrakis, E., and Dobry, R. 1989. Micromechanical behavior and modeling of granular soil. Report to Air force Office of Scientific research, Department of Civil Engineering, Rensselaer Polytechnic Institute, Troy, New York.

Petry, T.M. and Little, D.N. 2002. Review of stabilization of clays and expansive soils in pavements and lightly loaded structures-history, practice and future. Journal of Materials in Civil Engineering, **14**(6): 447-460.

Pojacek, R.B. 1979. Ed: Toxic and hazardous waste disposal. Ann Arbor Science Publishers, Ann Arbor, Michigan.

Polettini, A., Pomi, R., and Sirini, P. 2002. Fractional factorial design to investigate the influence of heavy metals and anions on acid neutralization behavior of cement-based products. Environmental Science and Technology, **36**(7): 1584-1591.

Polettini, A., Pomi, R., Sirini, P., and Testa, F. 2001. Properties of Portland cement stabilized MSWI fly ashes. Journal of Hazardous Materials, **88**(1): 123-138.

Potyondy, David 2002. A bonded-disk model for rock: relating micro properties and macro properties. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25

September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Potyondy, D.O. and Cundall, P.A. 2004. A bonded-particle model for rock. International Journal of Rock Mechanics and Mining Sciences, **41**(8): 1329–1364.

Pytel, A. and Singer F.L. 1987. Strength of materials. Harper and Row Publishers, Inc., New York.

Qiu, Y., and Sego, D.C. 2001. Laboratory properties of mine tailings. Canadian Geotechnical Journal, **38**(1): 183-190.

Raouf, M.W.A. and Nowier, H.G. 2004. Assessment of fossil fuel fly ash formulations in the immobilization of hazardous wastes. Journal of Environmental Engineering. ASCE, **130**(5): 499-507.

Reinhold, F. 1955. Elastic behaviour of soil-cement mixtures. Bulletin, Issue No. 108, Highway Research Board. Washington, D.C. pp. 128-137.

Richardson, D.N. 1996. AASHTO layer coefficients for cement-stabilized soil bases. Journal of Materials in Civil Engineering, **8**(2): 83-87.

Richardson, J.T.G. and Haynes, D.J. 2001. Slag bound materials in composite roads. Society of Chemical Industry (SCI) Lecture paper Series (LPS), LPS number 111/2001. SCI Publications, London, UK.

Rico, M. Benito, G. Salgueiro, A.R. Díez-Herrero, A. and Pereira, H.G. 2008. Reported tailings dam failures: a review of the European incidents in the worldwide context. Journal of Hazardous Materials, **152**(2): 846-852.

Robinsky, E. 1978. Tailings disposal by the thickened discharge method for improved economy and environmental control. Tailings Disposal Today, Vol. 2, pp. 75-92, 1979 by Miller Freeman Publications Inc., San Francisco, California. *In* Proceedings of the 2nd International Tailings Symposium, May 1978. Denver, Colorado.

Robinsky, E.I. 1999. Tailing dam failures need not be disasters-The Thickened Tailings Disposal (TTD) system. CIM Magazine, **92**(1028): 140-142.

Robinson, H. 2007. TARMAC-hydraulically bound materials technical overview. Seminar, British Lime Association, London, UK.

Robinson, M.B. 1995. Leachability reduction using lime/fly ash: an experimental optimization study. M.S. Thesis, New Mexico State University, Las Cruces, New Mexico.

Rothenburg, L., and Bathurst, R.J. 1992. Micromechanical features of granular assemblies with planner elliptical particles. Geotechnique, **42**(1): 79-95.

Rumsey, D. 2007. Intermediate statistics for dummies. John Wiley and Sons, New Jersey.

Ryan, C.R. and Jasperse, B.H. 1989. Deep soil mixing at the Jackson Lake dam. *In* Proceedings of the ASCE Geotechnical and Construction Divisions Special Conference. 25-29 June 1989.

Salami, M. Reza and Amini, F. 2002. Numerical model for the implementation of discontinuous deformation analysis in Finite Element mesh. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Salihoglu, G., Pinarli, V., Salihoglu, N.K., and Karaca, G. 2007. Properties of steel foundry electric arc furnace dust solidified/stabilized with Portland cement. Journal of Environmental Management, **85**(1): 190-197.

Saotome, T. 2007. Development of construction and demolition waste recycling in Ontario. School of Engineering Practice SEP 704, McMaster University, Ontario.

Sarracino, R.S. Dong, H.-J. and Moys, M.H. 2002. An experimental and theoretical study of ball-wall impact for DEM modeling of tumbling mills. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Scanlon, P.L. Sonnichsen, J.C., and Phillips, S.J. 1995. Coal ash usage in environmental restoration at the Hanford site. *In* Proceedings of the 57th Annual American Power Conference. 18-20 April 1995. Chicago, Illinois. pp. 304-308.

Shen, Hayley H. 2002. Regimes of granular shear flows. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Shin, H., Her, N. and Koo, J. 1988. Design optimization for solidification of hazardous wastes. Hazardous Waste and Hazardous Materials, **5**(3): 239-250.

Shutt, R.C. 1997. Economics for the construction industry. Addison Wesley Long man Limited, Harlow, Essex, UK.

Simieritsch, T. Obad, J., and Dyer S. 2009. Tailings plan review- an assessment of oil sands company submissions for compliance with ERCB Directive 074: tailings performance criteria and requirements for oil sands mining schemes. Pembina Institute, Box 7558, Drayton Valley, Alberta, T7A 1S7.

Singhal, A., Tewari, V.K. and Prakash, S. 2008. Utilization of treated spent liquor sludge with fly ash in cement and concrete. Building and Environment, **43**(6): 991-998.

Skempton, A.W. and Petley, D.J. 1970. Ignition loss and other properties of peats and clays from Avonmouth, King's Lynn and Cranberry Moss. Geotechnique, **20**(4): 343-356.

Stegemann, J.A. 2001. Neural Network Analysis of the effects of contaminants on properties of cement pastes. PhD Thesis, Imperial College of Science, Technology and Medicine, London, UK.

Stegemann, J.A., Butcher, E.J., Irabien, A., Johnston, P., de Miguel, R., Ouki, S.K., Polettini, A., and Sassaroli, G., Eds. 2001. Neural network analysis for prediction of

interactions in cement/waste systems. Final Report, Contract No. BRPR-CT97-0570. Commission of the European Community, Brussels, Belgium.

Stegemann, J.A., Perera, A.S., Cheeseman, C. and Buenfeld, N.R. 2000. 1/8 factorial study of metal effects on acid neutralization by cement. Journal of Environmental Engineering, **126**(10): 925-933.

Stegemann, J.A. and Côté, P. 1991. Investigation of test methods for solidified waste evaluation - a cooperative program. Environment Canada Report EPA 3/HA/8, Ottawa, Ontario.

Strack, O.D.L. and Cundall, P.A. 1978. The distinct element method as a tool for research in granular media. Part I, Report to the National Science Foundation, Department of Civil and Mineral engineering, University of Minnesota. Minneapolis, Minnesota.

Strack, O.D.L. and Cundall, P.A. 1984. Fundamental studies of fabric in granular materials. Interim Report to the National Science Foundation, CEE-8310729, Department of Civil and Mineral engineering, University of Minnesota. Minneapolis, Minnesota.

Roy, S., Adhikari, G.R., and Gupta, R.N. 2007. Use of gold mill tailings in making bricks: a feasibility study. Waste Management and Research, **25**(5): 475–482.

Swami, R.K., Pundhir, N.K.S., and Mathur, S. 2007. Kimberlite tailings a road construction material. Transportation Research Record: Journal of the Transportation Research Board, **2**(1989): 131–134. Transportation Research Board of the National Academies, Washington, D.C.

Tannant, Dwayne D. and Wang, Caigen 2002. Thin rock support liners modeled with Particle Flow Code. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Tarumi, Y. and Hakuno, M.A. 1989. A DEM for sand liquefaction. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Tavarez, Federico A., Plesha, Michael E., and Bank, Lawrence C. 2002. Discrete Element Method (DEM) for modeling solid and particulate media. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Teredesai, R., Musharraf, Z., Miller, G.A. and Nairn, R. 2005. Cementitious stabilization of raw chat for roadway base application. Technical Report, Oklahoma Department of Environmental Quality, Oklahoma City, Oklahoma.

Theriault, J.A., Frostiak, J., and Welch, D. 2003. Surface disposal of paste tailings at the Bulyanhulu gold mine, Tanzania. *In* Proceedings of Sudbury 2003 Mining and the Environment. 25-28 May 2003. Sudbury, Ontario.

Thevenin, G. and Pera, J. 1999. Interactions between lead and different binders. Cement Concrete Research, **29**(10): 1605–1610.

Thornton, C. and Randall, C.W. 1988. Applications of theoretical contact mechanics to solid particle system simulations. *In* Proceedings of Micromechanics of Granular Materials: New Method and Constitutive Relations, (Eds. J.T. Jenkins and M. Satake), Elsevier, Amsterdam. The Netherlands. pp. 133-142.

Ting, J.M., Corkum, B.T., Kauffman, C.R., and Greco, C.A. 1986. Application of the distinct element method in geotechnical engineering. *In* Proceedings of the 2nd International Symposium on Numerical Models in Geomechanics. Ghent, Belgium.

Ting, J.M., Corkum, B.T., Kauffman, C.R., and Greco, C.A. 1987. Discrete numerical modeling of soil: validation and application. Publication 87-03, Department of Civil Engineering, University of Toronto, Toronto, Ontario.

Ting, J.M., and Corkum, B.T. 1988*a*. Strength behavior of granular materials using discrete numerical modeling. *In* Proceedings of the 6th International Symposium on Numerical Models in Geomechanics, Innsbruck, Austria. pp. 305-310.

Ting, J.M., and Corkum, B.T. 1988*b*. Soil-structure interaction by discrete element modeling. *In* Proceedings of the Annual Canadian Society of Civil Engineering (CSCE) Conference. Calgary, Alberta. pp. 196-215.

Ting, J. M., and Corkum, B. T. 1988*c*. Application of DEM in geotechnical engineering. *In* Proceedings of the 3rd International Conference on Computational Civil Engineering. Vancouver, British Columbia. pp. 578-594.

Ting, J.M., Corkum, B.T., Kauffman, C.R., and Greco, C.A. 1989. Discrete numerical model for soil mechanics. Journal of Geotechnical Engineering. ASCE, **115**(3): 379-398.

Thornton, Colin and Kafui, D. 2002. 3D DEM simulations of gas-solid fluidised beds. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Torrey, S. 1978. Coal ash utilization: fly ash, bottom ash and slag. Noyes Data Corporation, Park Ridge, New Jersey.

Toutanji, H., Delatte, N., Aggoun, S., Duval, R., and Danson, A. 2004. Effect of supplementary cementitious materials on the compressive strength and durability of short-term cured concrete. Cement and Concrete Research, **34**(2): 311–319.

Trezza, M. A. 2007. Hydration study of ordinary Portland cement in the presence of zinc ions. Materials Research, **10**(4): 331-334.

Turner, John P. 1997. Evaluation of western cola fly ashes for stabilization of low-volume roads. *In* Proceedings of Testing Soil Mixed with Waste or Recycled Materials, ASTM STP 1275. Mark A. Wasemiller and Keith B. Hoddinott, Eds., American Society for Testing and Materials. pp. 157-171.

Uchida, Y. and Hakuno, M.A. 1989. A DEM simulation for debris flow. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Uemura, D. and Hakuno, M. 1987. Granular assembly simulation with Cundall's model for the dynamic collapse of the structural foundation. *In* Proceedings of the Japanese Society of Civil Engineering (JSCE), Structural Engineering/Earthquake Engineering, Vol. 4, pp. 181-190.

Uemura, D. and Hakuno, M. 1989. A DEM simulation for pile penetrating into ground. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

US Department of Health and Human Services 1998. Health consultation: A note of explanation. Technical Report, US Department of Health and Human Services, 200 Independence Avenue, S.W. - Washington, D.C. 20201.

USEPA Method 1311. 1992. Toxicity characteristic leaching procedure. US Environmental Protection Agency, Washington, D.C.

USEPA 1996. Stabilization/solidification processes for mixed waste. Report, EPA 402-R-96-014, US Environmental Protection Agency, Office of Radiation and Indoor Air, 401 M Street, S.W. Washington, D.C. 20460.

USEPA 2011. US Environmental Protection Agency, website info, http://www.epa.gov, Ariel Rios Building, 1200 Pennsylvania Ave. NW, Washington, DC 20004.

Usmen, M.A. 1986. Properties, disposal and stabilization of combined coal refuse. *In* Proceedings of the International Conference on Environmental Geotechnology, Vol. 1. April 1986. Lehigh, Pennsylvania.

Valles, N. 1996. Transient behavior during the toxicity characteristic leaching procedure. M.S. Thesis, New Mexico State University, Las Cruces, New Mexico.

Vanicek, I. 2002. Tailings dams-practical problems of aging. *In* Proceedings of the 4th International Conference on Environmental Geotechnics. Rio de Janeiro, Brazil. pp. 301-304.

Van Riessen, G.J. and Hansen, K.D. 1992. Cement-stabilized soil for coal retaining berms. *In* Proceedings of the ASCE Geotechnical Division Specialty Conference Grouting, Soil Improvement and Geosynthetics, New Orleans, Louisiana, 25-28 Feb 1992.

Vermeulen, N.J., Rust, E., and Clayton, C.R.I. 2002. Variations in composition on SA gold tailings dams. *In* Proceedings of the 9th International Conference on Tailings and Mine Waste. Fort Collins, Colorado.

Vipulanandan, C. and Shenoy, S. 1992. Properties of cement grouts and grouted sands with additives. *In* Proceedings of the ASCE Geotechnical Division Specialty Conference Grouting, Soil Improvement and Geosynthetics, New Orleans, Louisiana, 25-28 Feb 1992.

Walton, O.R. 1982. Explicit particle dynamics model for granular materials. *In* Proceedings of the 4th International Conference on Numerical methods in Geomechanics. Edmonton, Alberta. pp.1262-1269.

Walton, O.R. 1983. Particle-dynamic calculations of shear flow. *In* Proceedings of Mechanics of Granular Materials: New Method and Constitutive Relations, (Editors J.T. Jenkins and M. Satake), Elsevier. Amsterdam, the Netherlands. pp 327- 338.

Walton, O.R. and Braun, R.L. 1986. Viscosity, granular-temperature and stress calculations for shearing assemblies of inelastic, frictional discs. Journal of Rheology, **30**(5): 949-980.

Wang, S.Y. and Vipulanandan, C. 1996. Leachability of lead from solidified cement-fly ash binders. Cement and Concrete Research, **26**(6): 895-905.

Wang, S. and Vipulanandan, C. 2000. Solidification/stabilization of Cr(VI) with cement: leachability and XRD analyses. Cement and Concrete Research, **30**(3): 385-389.

Wasayo, S. 2003. Suitability of mine tailings for shotcrete as a ground support. M.A.Sc. thesis, Mining Engineering, Dalhousie University, Halifax, Nova Scotia.

Wasiuddin, N.M., Hamid, W.B., Kolothody, N. Gause, C.F., Zaman, M.M. and Nairn, R.R. 2005. A laboratory study to optimize the use of raw chat in hot mix asphalt for pavement application: final report. Technical Report, Oklahoma Department of Environmental Quality 707 N Robinson, Oklahoma City, Oklahoma, 73102.

Weitzman, L., Hamel, L.E., and Barth, E. 1988. Evaluation of solidification/stabilization as best demonstrated available technology. *In* Proceedings of the 14th Annual Research Symposium, U.S. EPA, Cincinnati, Ohio.

Wilmoth, R.C. and Scott, R.B. 1974. Use of coal mine refuse and fly ash as a road base material. *In* Proceedings of the First Symposium on Mine and Preparation Plant Refuse Disposal. October 1974. Louisville, Kentucky. pp. 263 – 275.

Wu, W.Y. and Liu, Z.D. 1989. The simulation of fabrics for granular material with DEM. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Yamamoto, T. and Hakuno, M.A. 1989. A Dem simulation for tunnel excavation. *In* Proceedings of the 1st US Conference on Discrete Element Methods. 19-20 October 1989. Golden, Colorado.

Yanful, E.K., and Verma, A. 1999. Oxidation of flooded mine tailings due to resuspension. Canadian Geotechnical Journal, **36**(5): 826-845.

Yanful, E.K., Verma, A. and Straatman, A.G. 2000. Turbulence-driven metal release from resuspended pyrrhotite tailings. Journal of Geotechnical and Geoenvironmental Engineering. ASCE, **126**(12): 1157-1165.

Yang, H., Chen, C., Pan, L., Lu, H., Sun, H., and Hu, X. 2009. Preparation of double-layer glass-ceramic/ceramic tile from bauxite tailings and red mud. Journal of the European Ceramic Society, **29**(10): 1887–1894.

Yarbaşı, Necmi, Kalkan, Ekrem and Akbulut, S. 2007. Modification of the geotechnical properties, as influenced by freeze–thaw, of granular soils with waste additives. Cold Regions Science and Technology, **48**(1): 44-54.

Zamorani, E., Sheikh, I. and Serrini, G. 1989. A study of the influence of nickel chloride on the physical characteristics and leachability of Portland cement. Cement and Concrete Research, **19**(2): 259-266.

Zhang, Duan Z. 2002. Effects of particle interaction history in dense and slow granular flows. *In* Proceedings of the 3rd International Conference on Discrete Element Methods,

Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Zhang, Y., and Cundall, P.A. 1986. A numerical simulation of slow deformations. *In* Proceedings of the Symposium on Mechanics of Particulate Media, Tenth US National Congress on Applied Mechanics. University of Texas, Austin, Texas.

Zhang, Y., Wang, Z., Xu, X., Chen, Y., and Qi, T. 1999. Recovery of heavy metals from electroplating sludge and stainless steel pickle waste liquid by ammonia leaching method. Journal of Environmental Sciences, English edition, China, **11**(3): 381–384.

Zeghal, Morched Edil, Tuncer B. and Plesha, Michael E. 2002. Discrete Element Method for sand-structure interaction. *In* Proceedings of the 3rd International Conference on Discrete Element Methods, Discrete Element Methods: Numerical Modeling of Discontinua. 23-25 September 2002. Santa Fe, New Mexico. Edited by Benjamin K. Cook and Richard P. Jensen.

Zou, D.H. and Huang, Y. 1997. Industrial application of HiFa Bond. Report To Dzou Geomechanics Int. and National Research Council Canada (Contract # 04470E). Department of Mineral and Metallurgical Engineering, Dalhousie Tech, Dalhousie University, Halifax, Nova Scotia.

Zou, D.H. and Li, L.P. 1999. Strengthening of solidified dilute tailings slurry. Journal of Geotechnical and Geoenvironmental Engineering. ASCE, **125**(1): 11-15.

Zou, D.H. and Sahito, W. 2004. Suitability of mine tailings for shotcrete as a ground support. Canadian Journal of Civil Engineering, **31**(4): 632–636.

Appendix A:
Experimental Figures

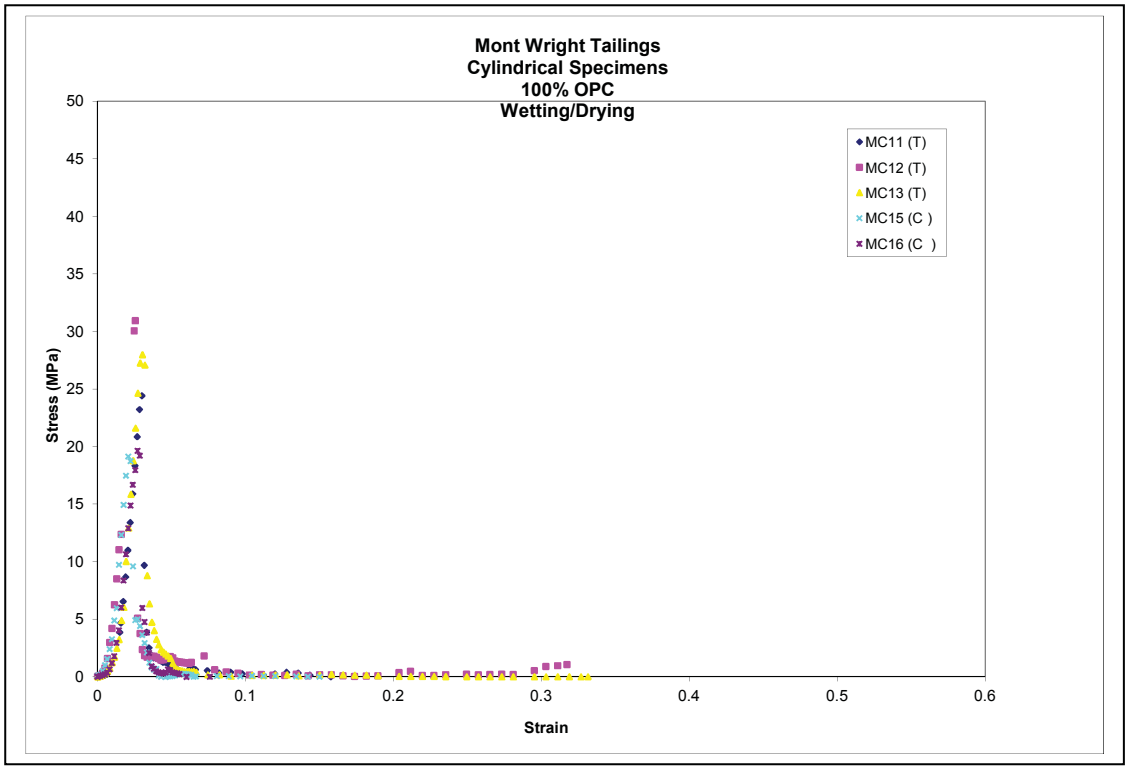


Figure A-1 Compression strength for Mont Wright samples MC11-MC16 after wetting/drying

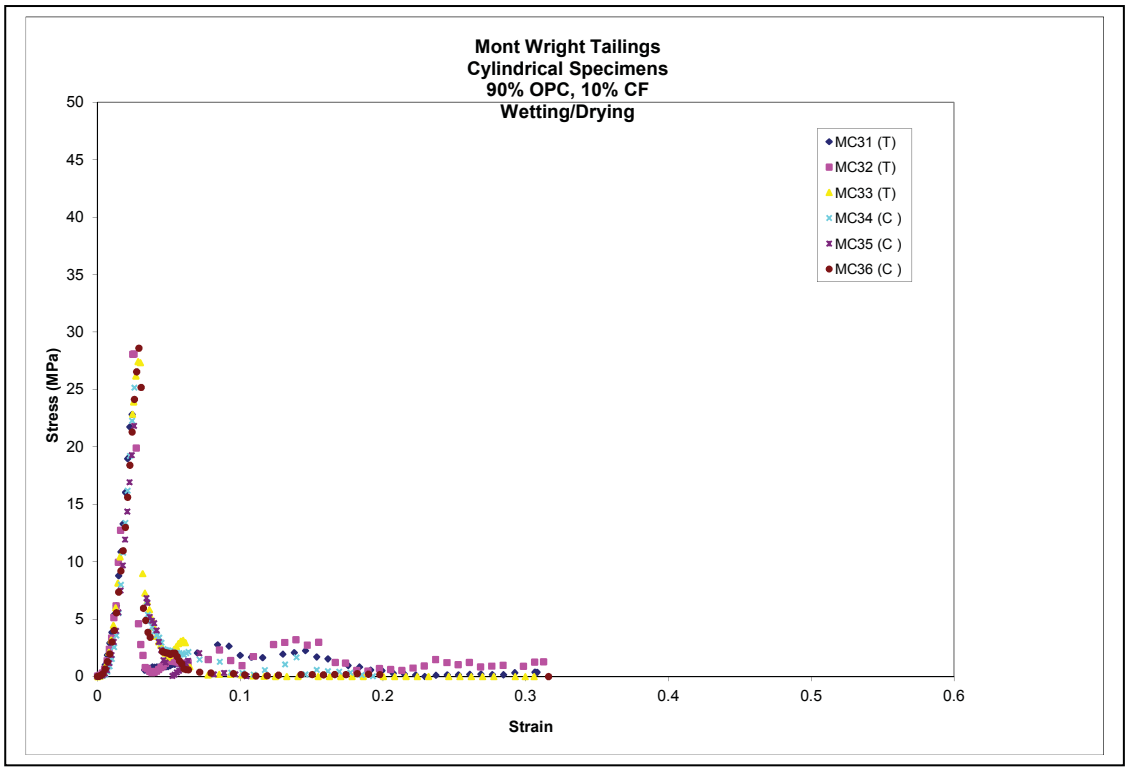


Figure A-2 Compression strength for Mont Wright samples MC31-MC36 after wetting/drying

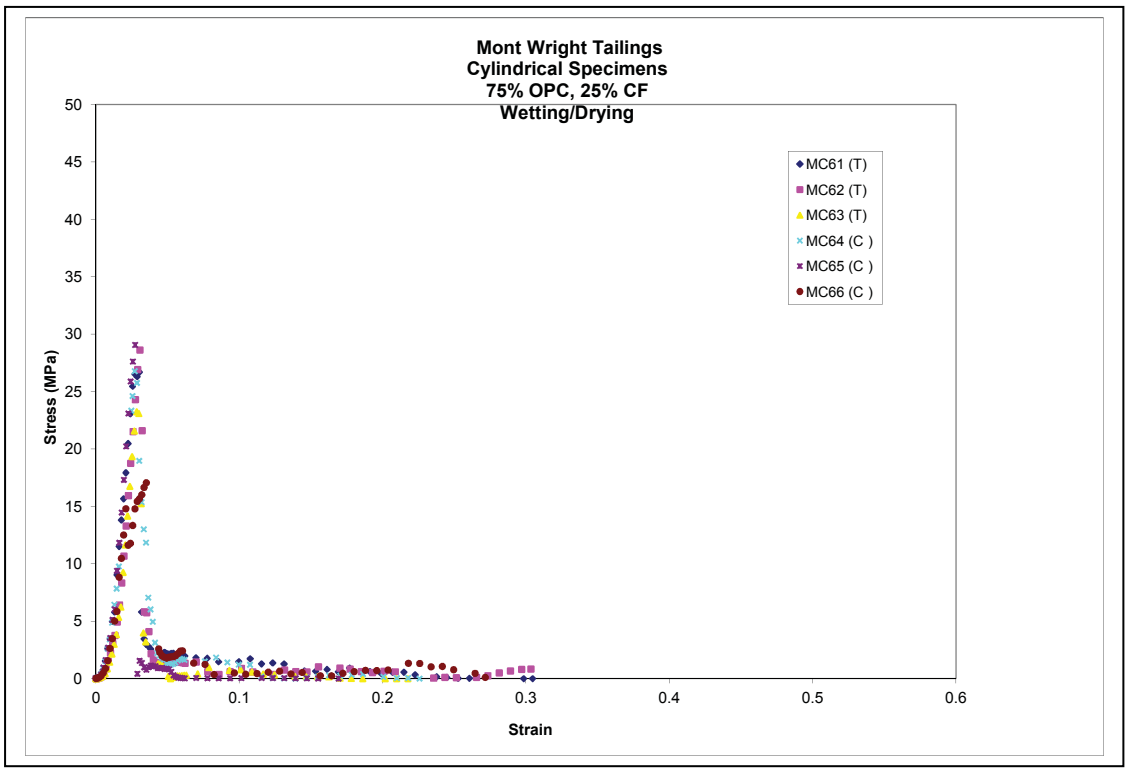


Figure A-3 Compression strength for Mont Wright samples MC61-MC66 after wetting/drying

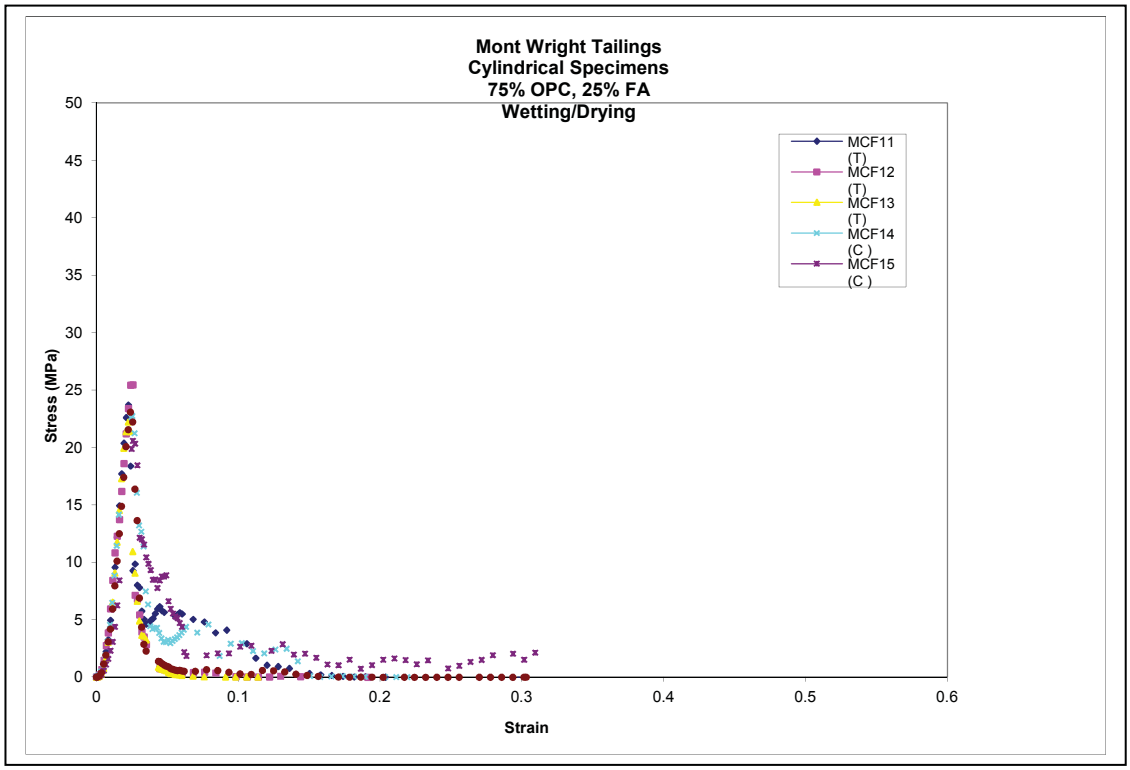


Figure A-4 Compression strength for Mont Wright samples MCF11-MCF16 after wetting/drying

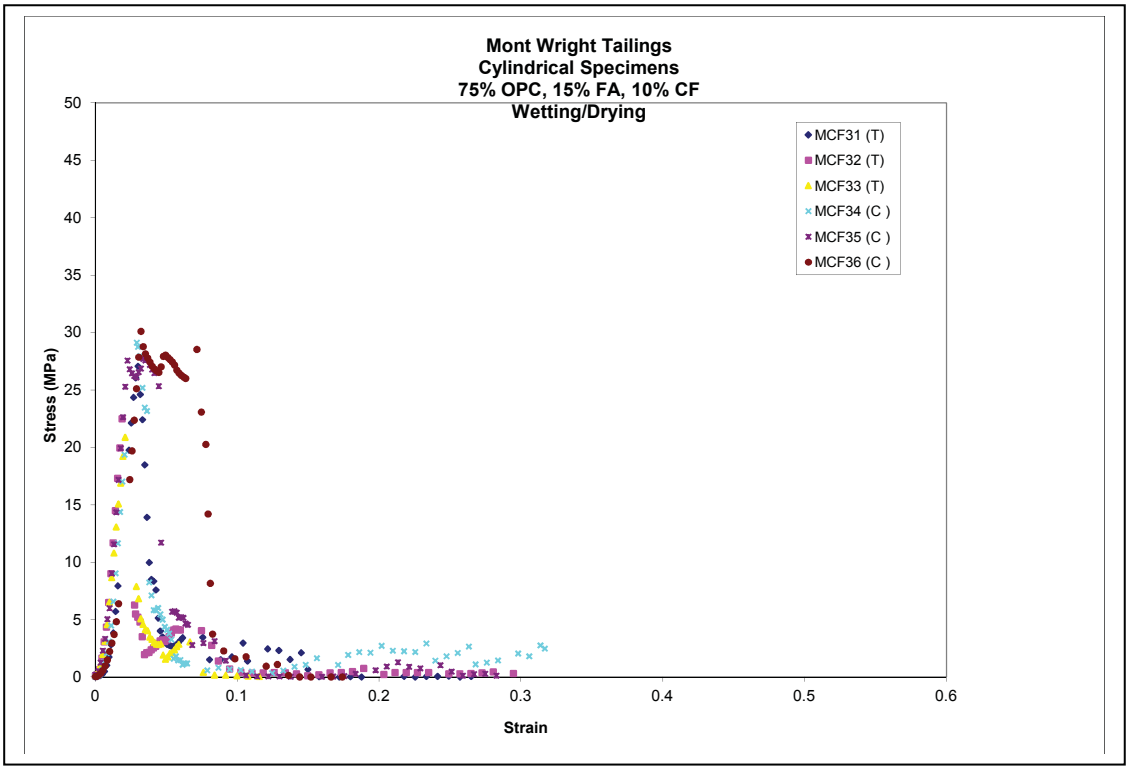


Figure A-5 Compression strength for Mont Wright samples MCF31-MCF36 after wetting/drying

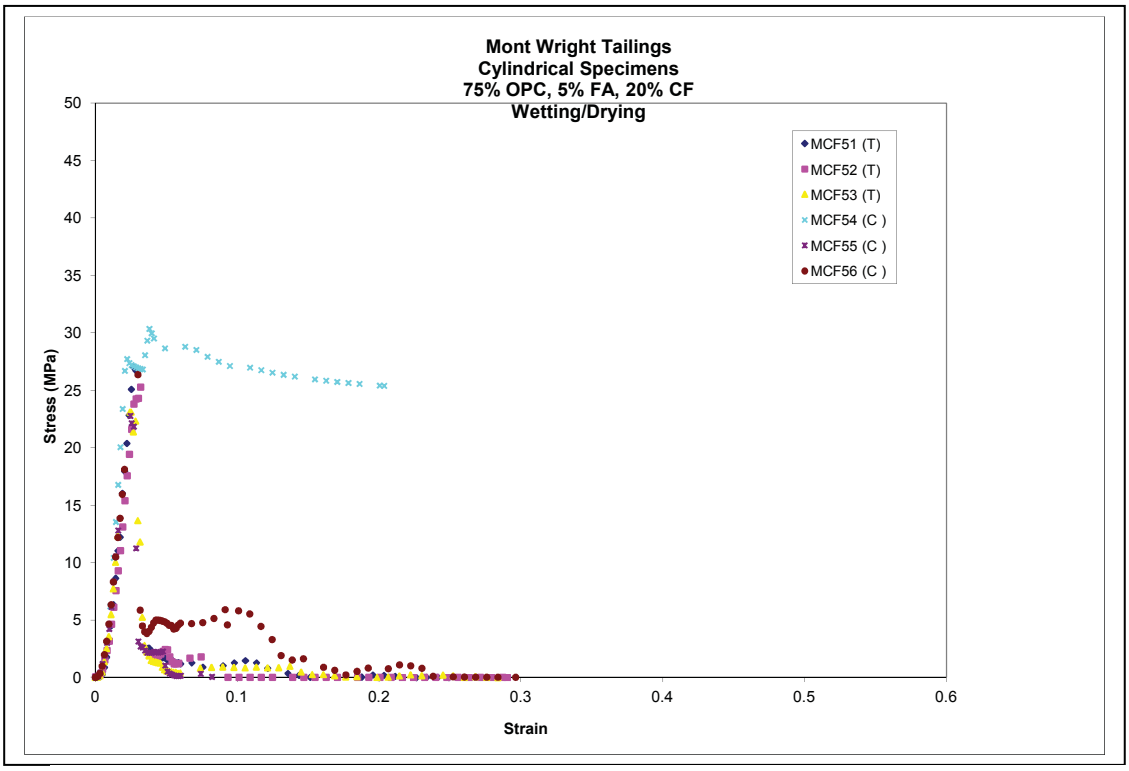


Figure A-6 Compression strength for Mont Wright samples MCF51-MCF56 after wetting/drying

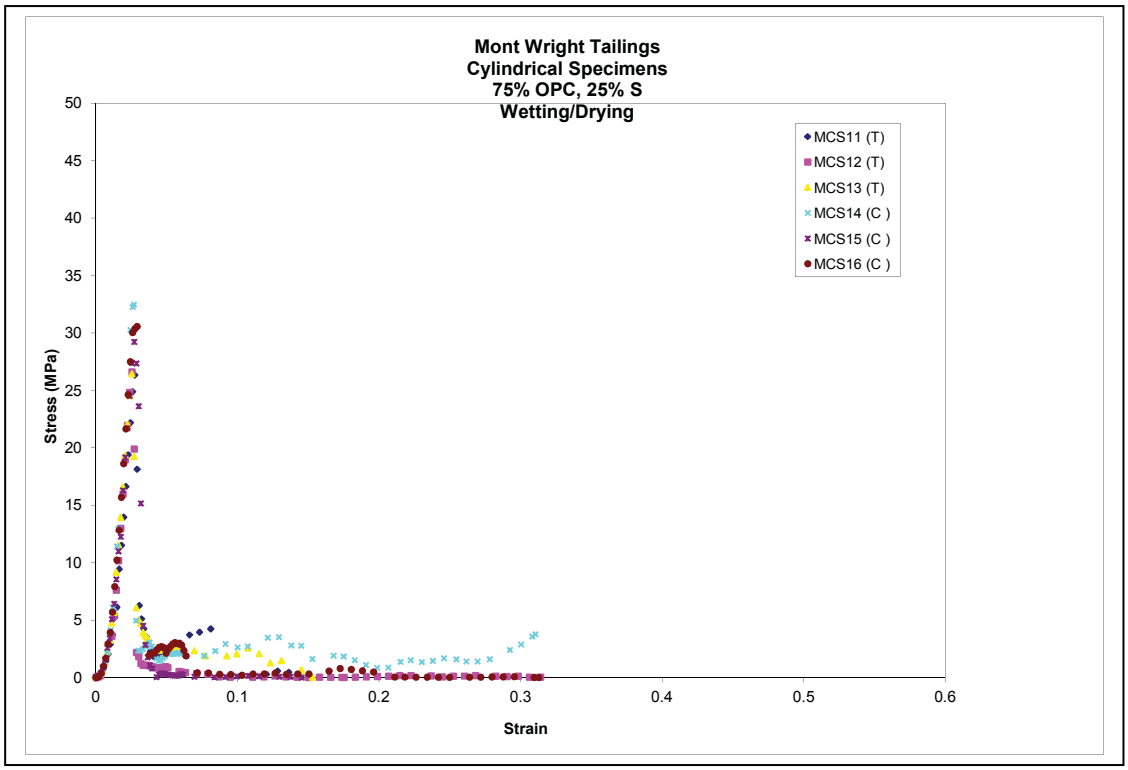


Figure A-7 Compression strength for Mont Wright samples MCS11-MCS16 after wetting/drying

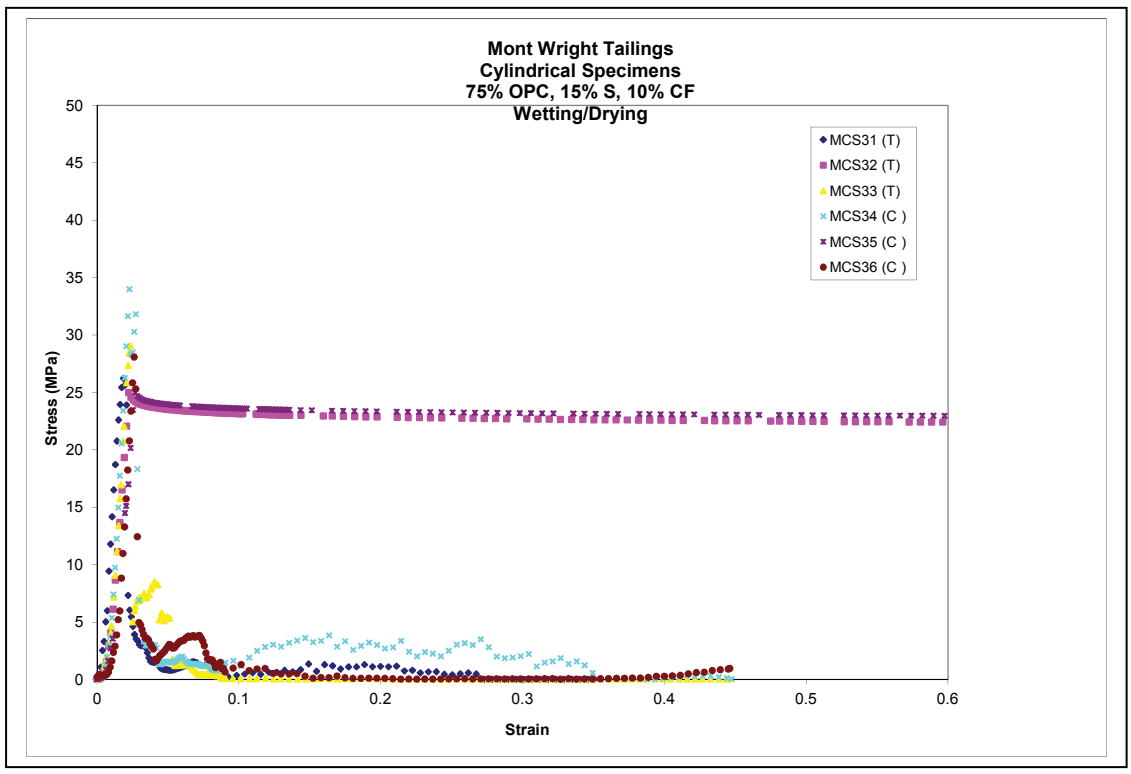


Figure A-8 Compression strength for Mont Wright samples MCS31-MCS36 after wetting/drying

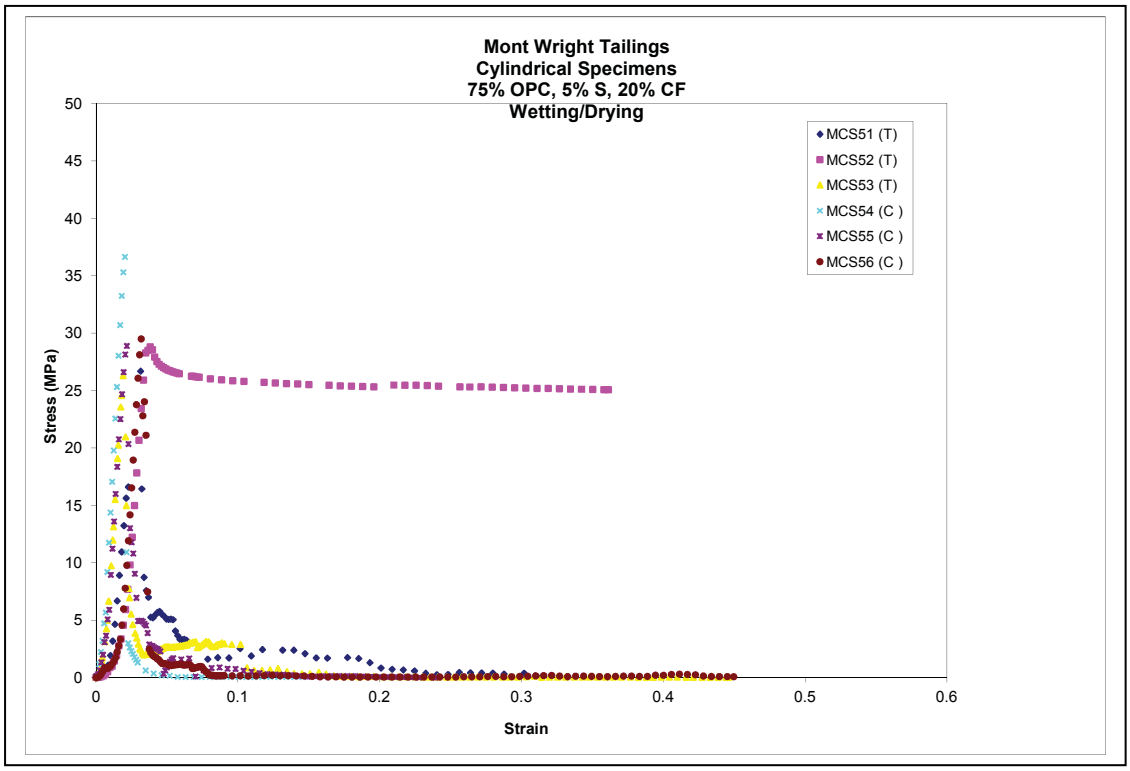


Figure A-9 Compression strength for Mont Wright samples MCS51-MCS56 after wetting/drying

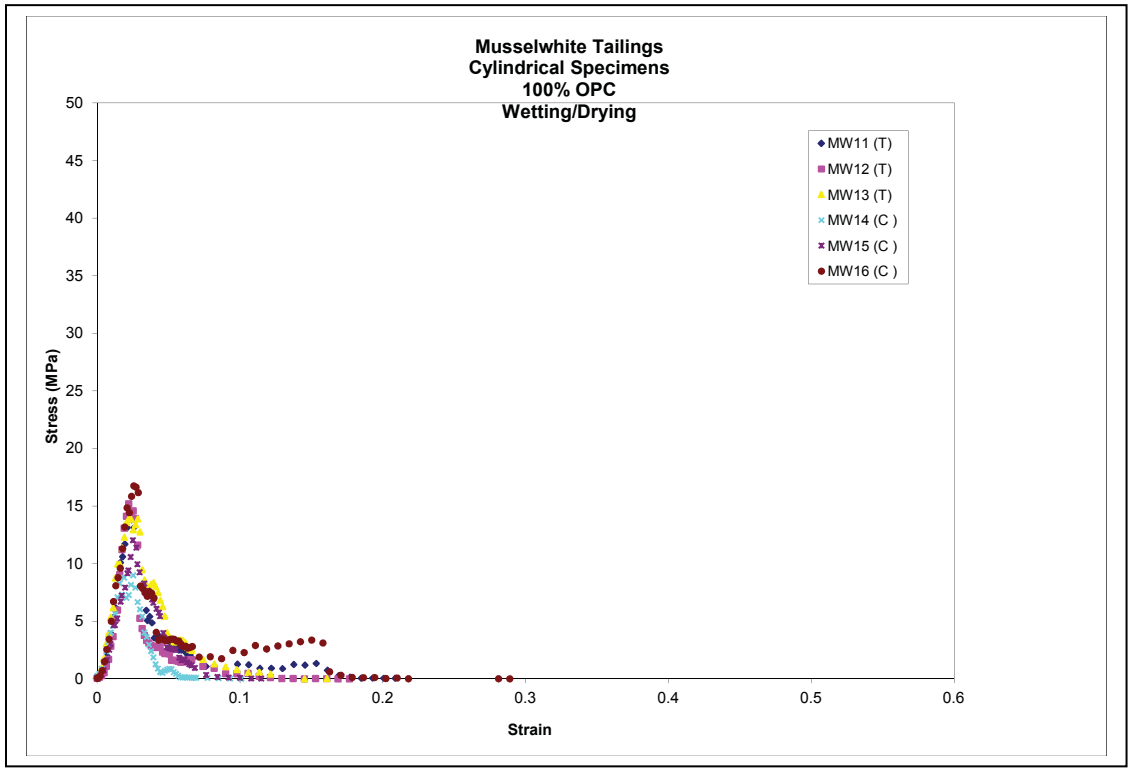


Figure A-10 Compression strength for Musselwhite samples MW11-MW16 after wetting/drying

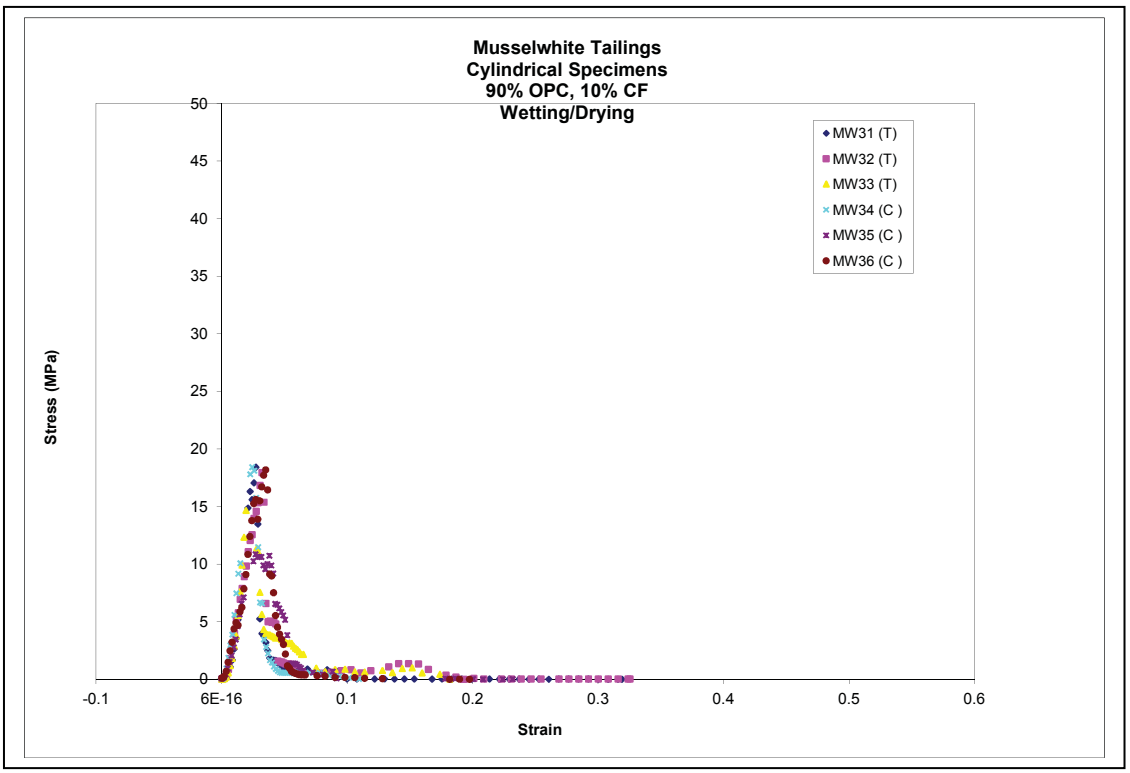


Figure A-11 Compression strength for Musselwhite samples MW31-MC36 after wetting/drying

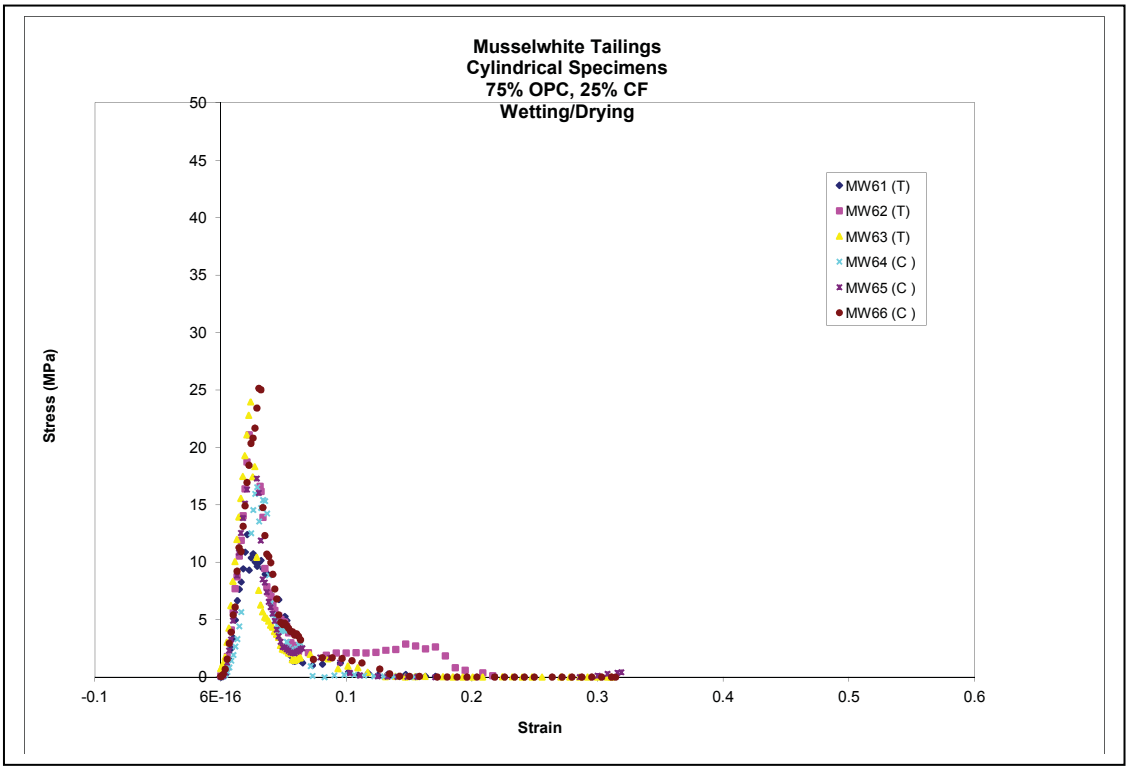


Figure A-12 Compression strength for Musselwhite samples MW61-MC66 after wetting/drying

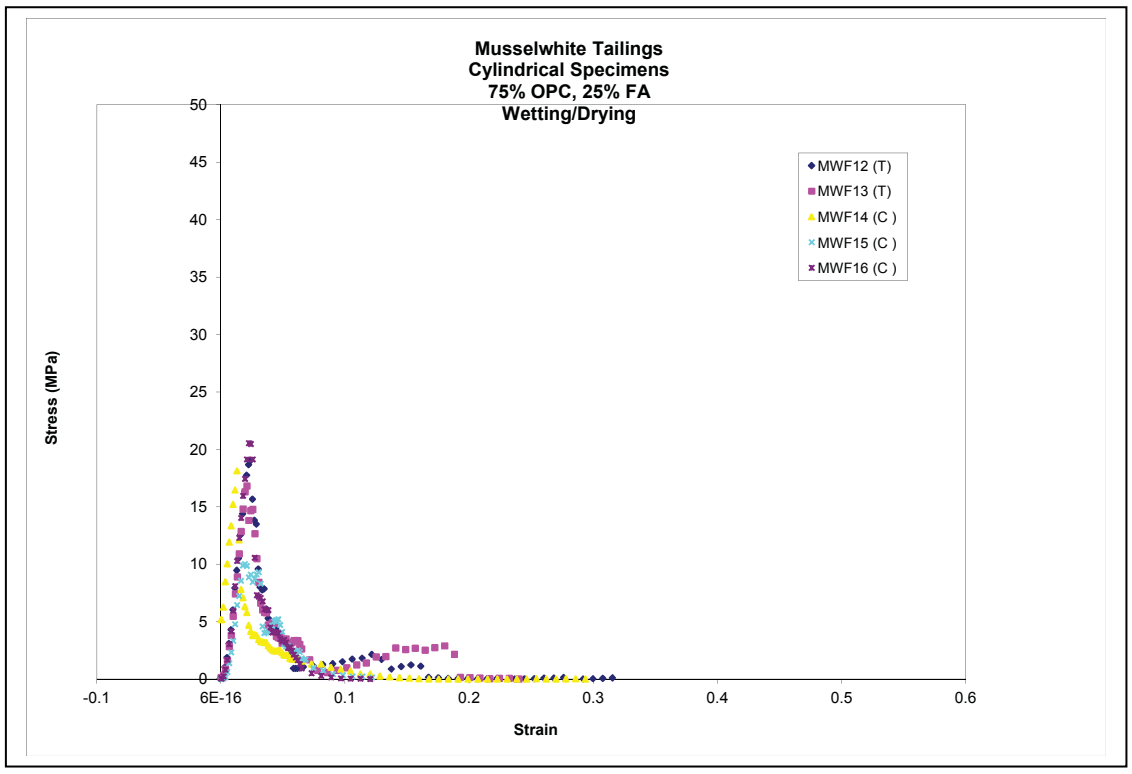


Figure A-13 Compression strength for Musselwhite samples MWF11-MWF16 after wetting/drying

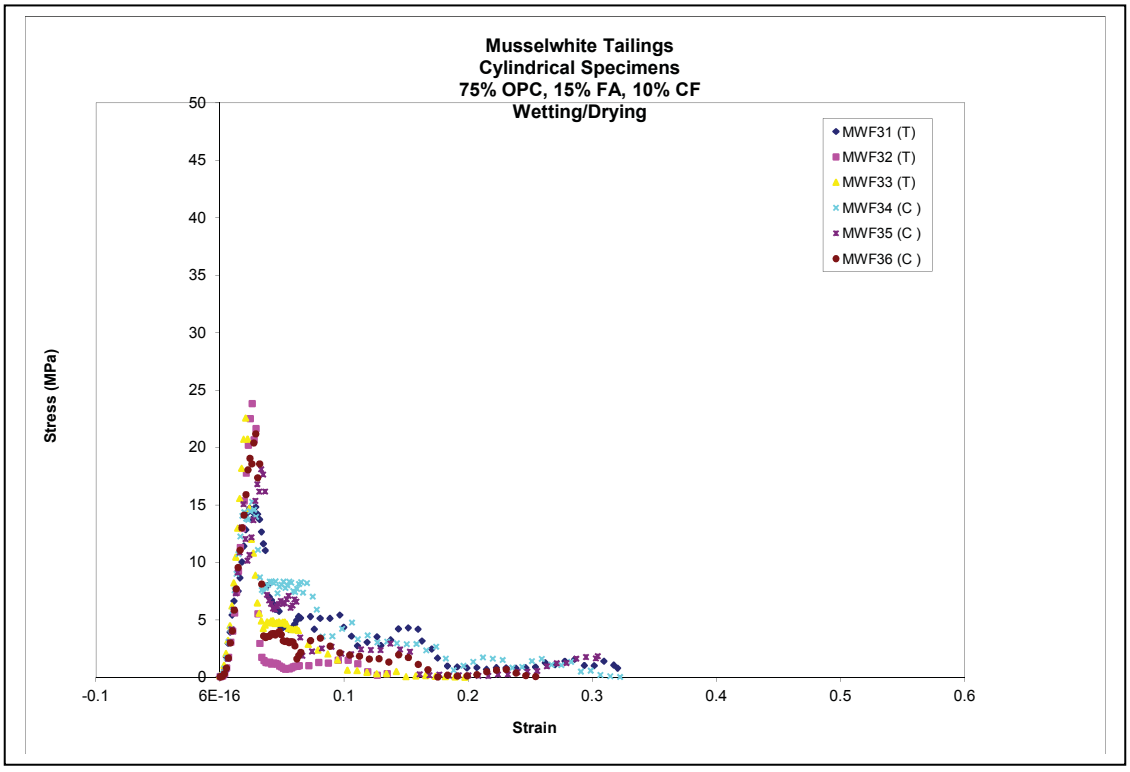


Figure A-14 Compression strength for Musselwhite samples MWF31-MWF36 after wetting/drying

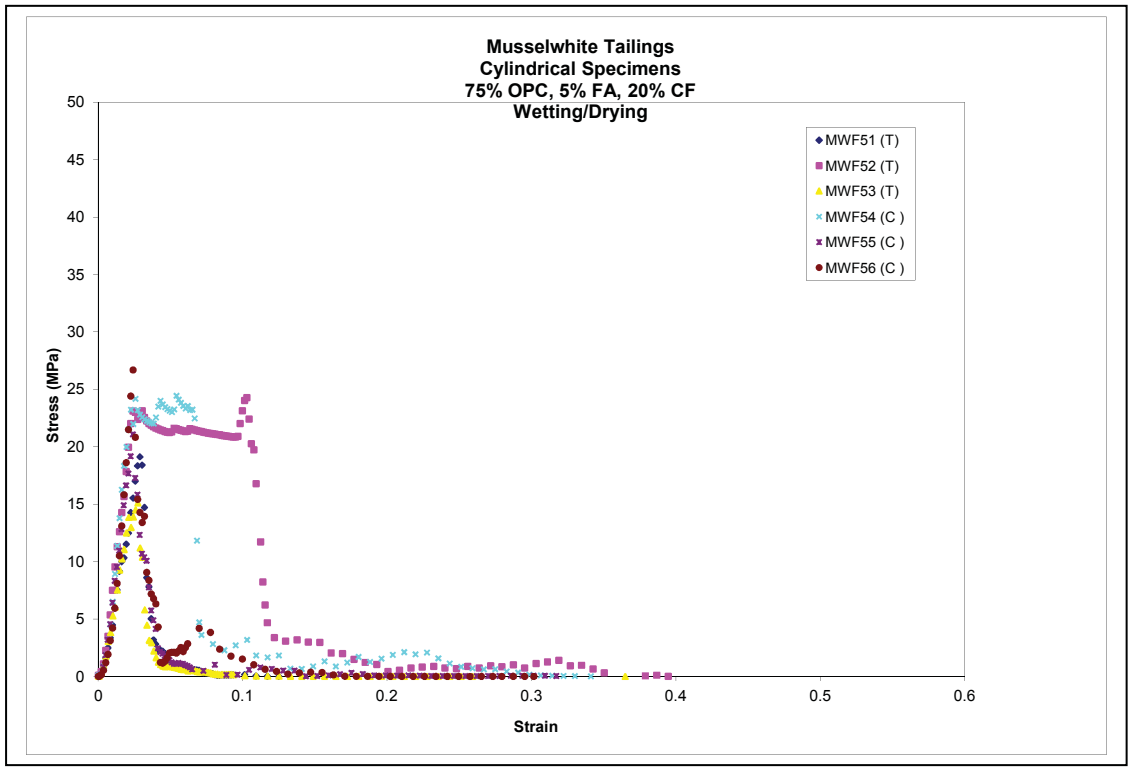


Figure A-15 Compression strength for Musselwhite samples MWF51-MWF56 after wetting/drying

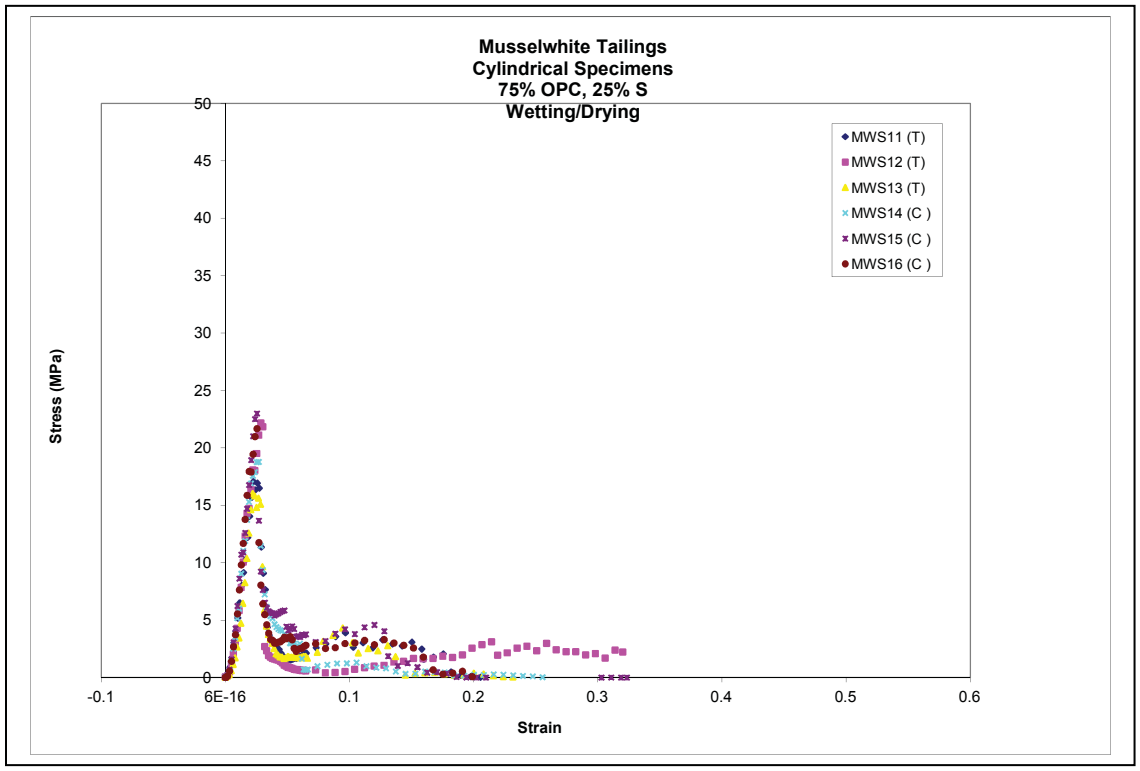


Figure A-16 Compression strength for Musselwhite samples MWS11-MWS16 after wetting/drying

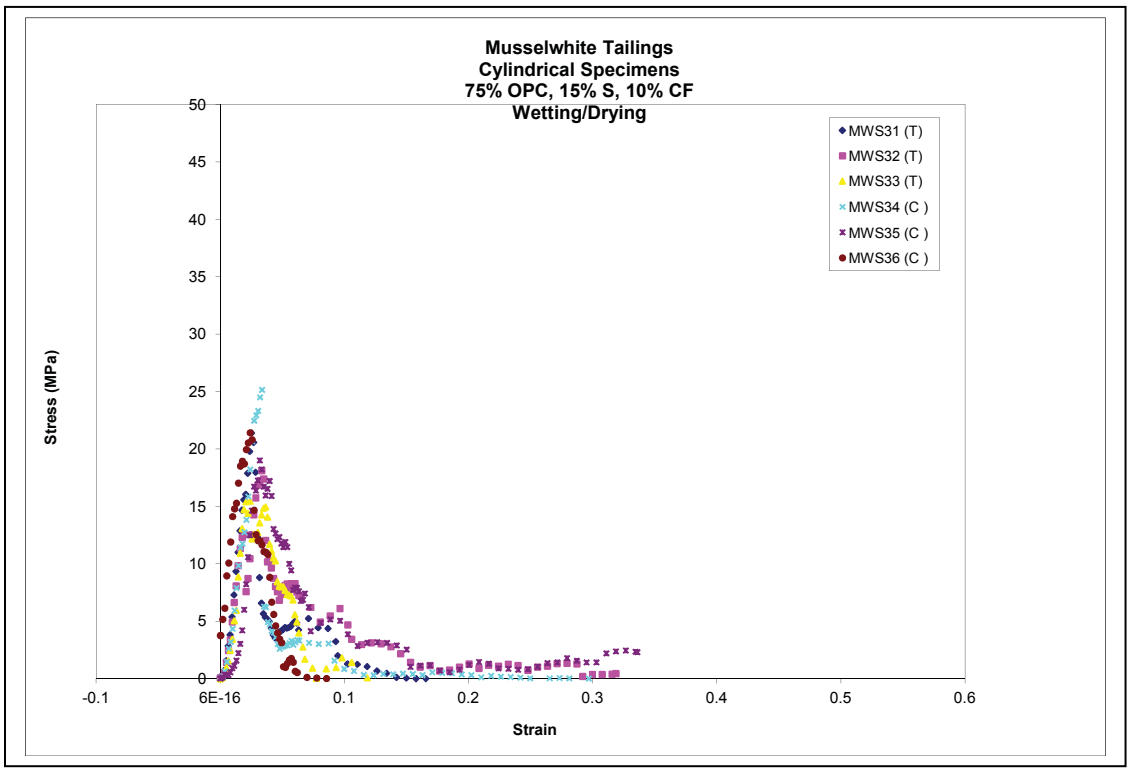


Figure A-17 Compression strength for Musselwhite samples MWS31-MWS36 after wetting/drying

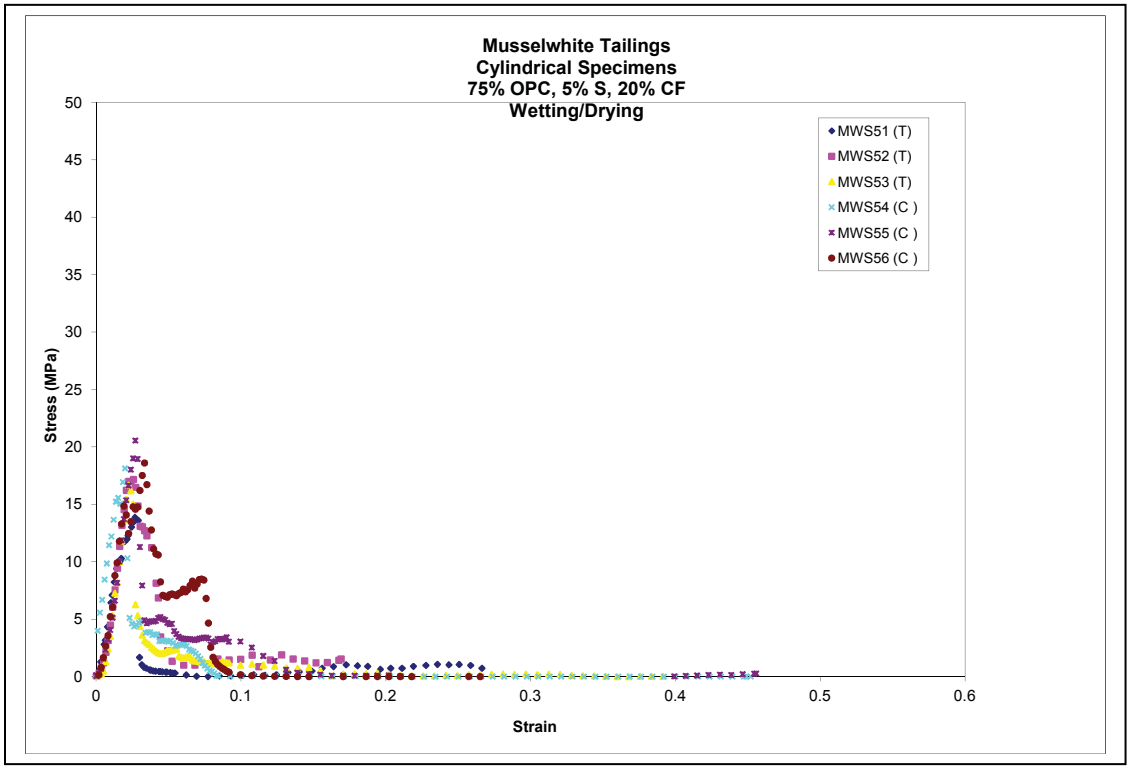


Figure A-18 Compression strength for Musselwhite samples MWS51-MWS56 after wetting/drying

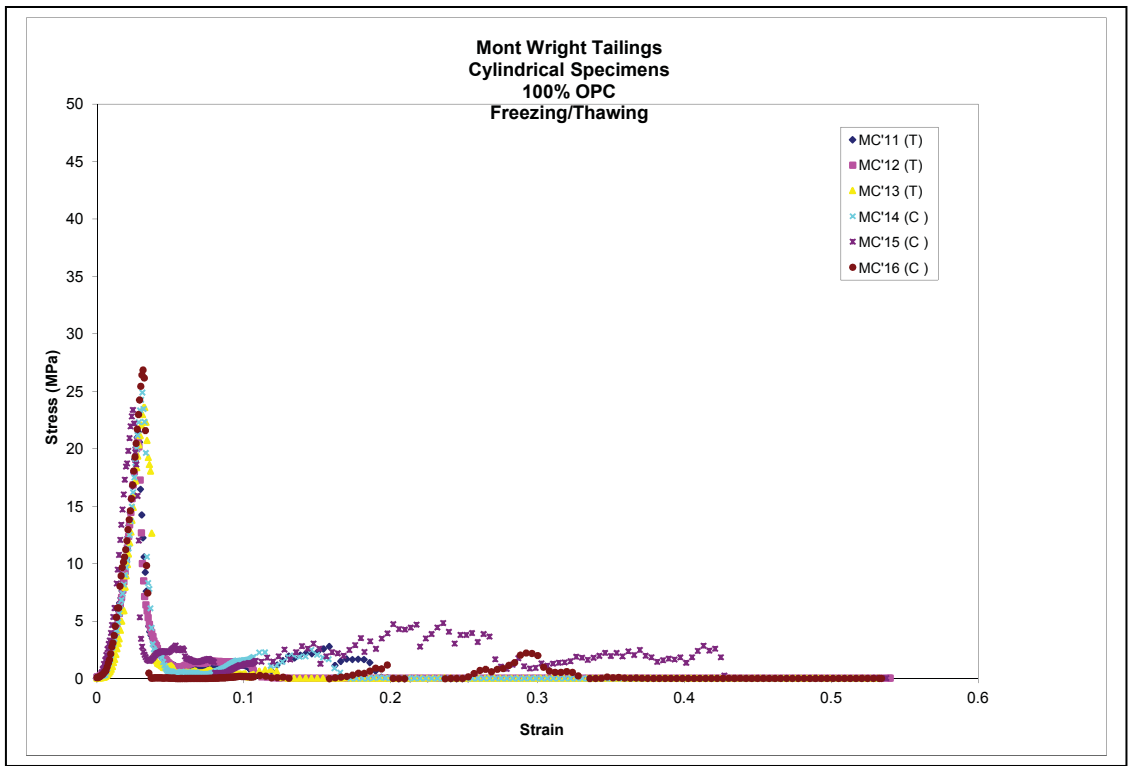


Figure A-19 Compression strength for Mont Wright samples MC'11-MC'16 after freezing/thawing

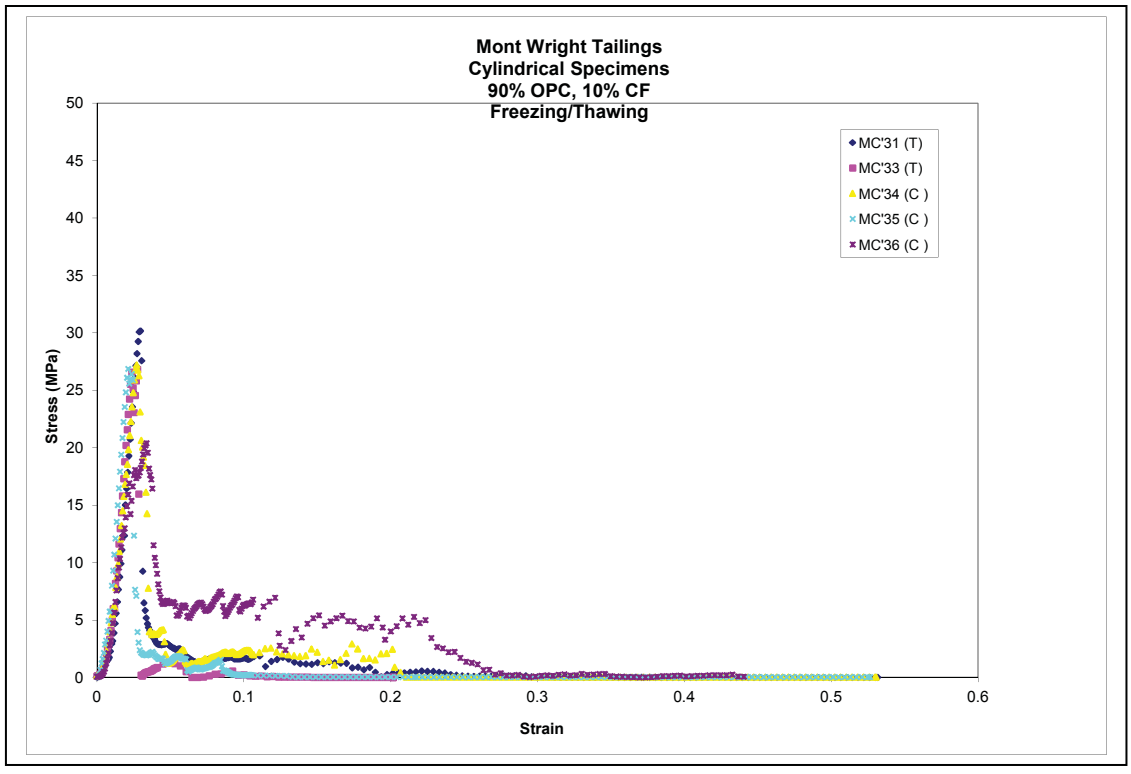


Figure A-20 Compression strength for Mont Wright samples MC'31-MC'36 after freezing/thawing

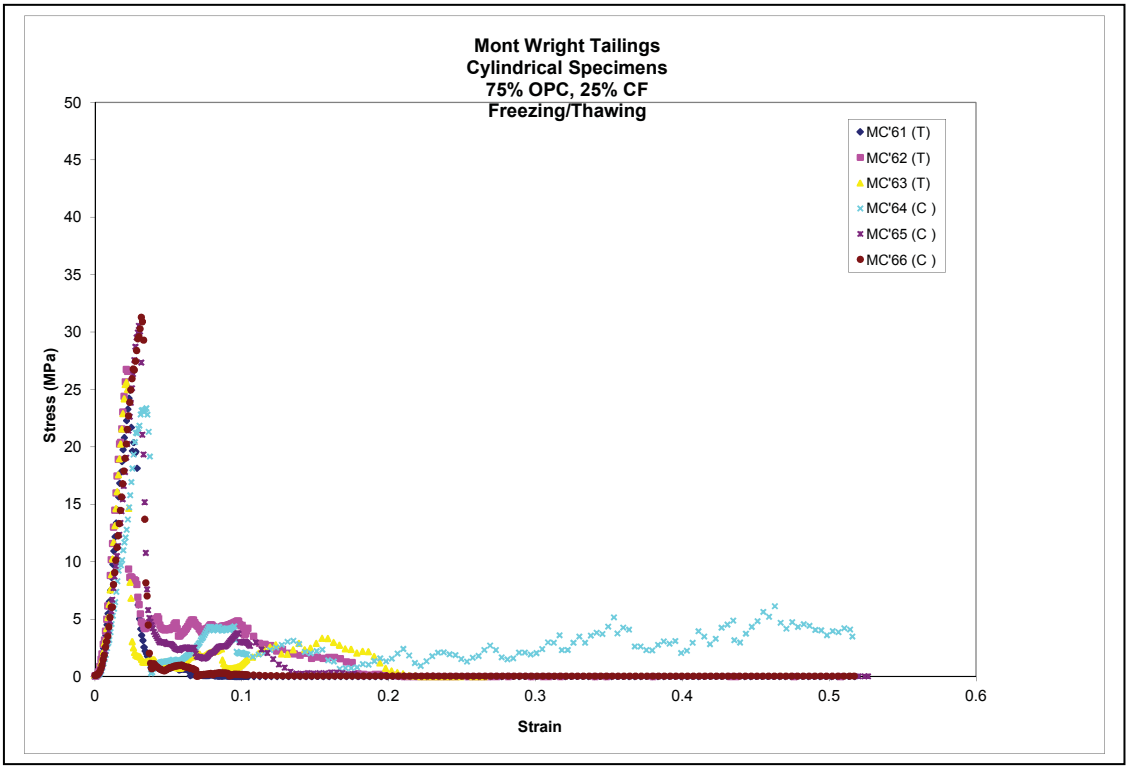


Figure A-21 Compression strength for Mont Wright samples MC'61-MC'66 after freezing/thawing

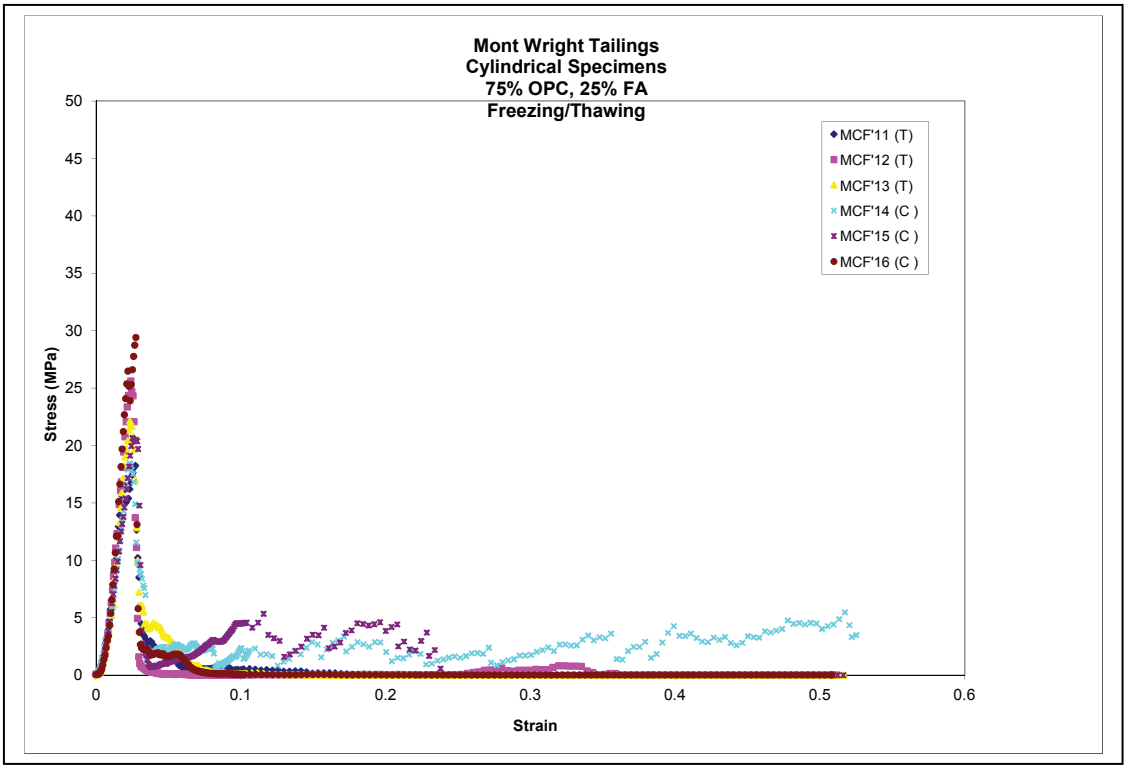


Figure A-22 Compression strength for Mont Wright samples MCF'11-MCF'16 after freezing/thawing

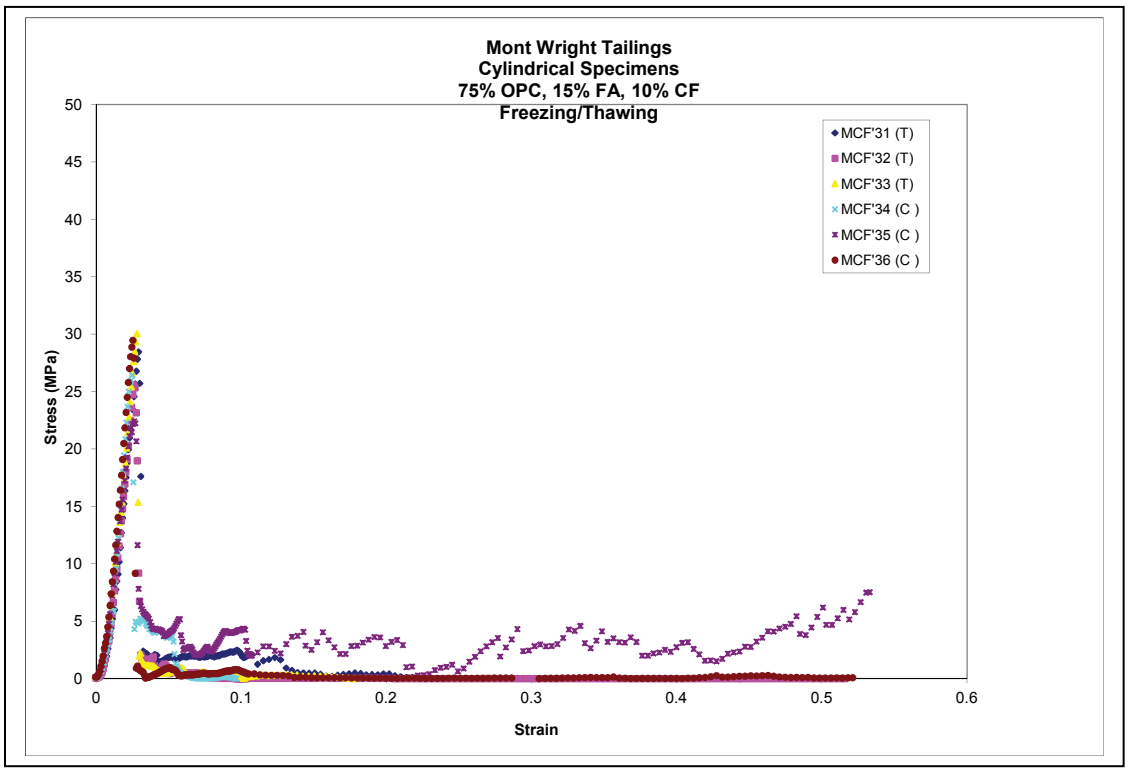


Figure A-23 Compression strength for Mont Wright samples MCF'31-MCF'36 after freezing/thawing

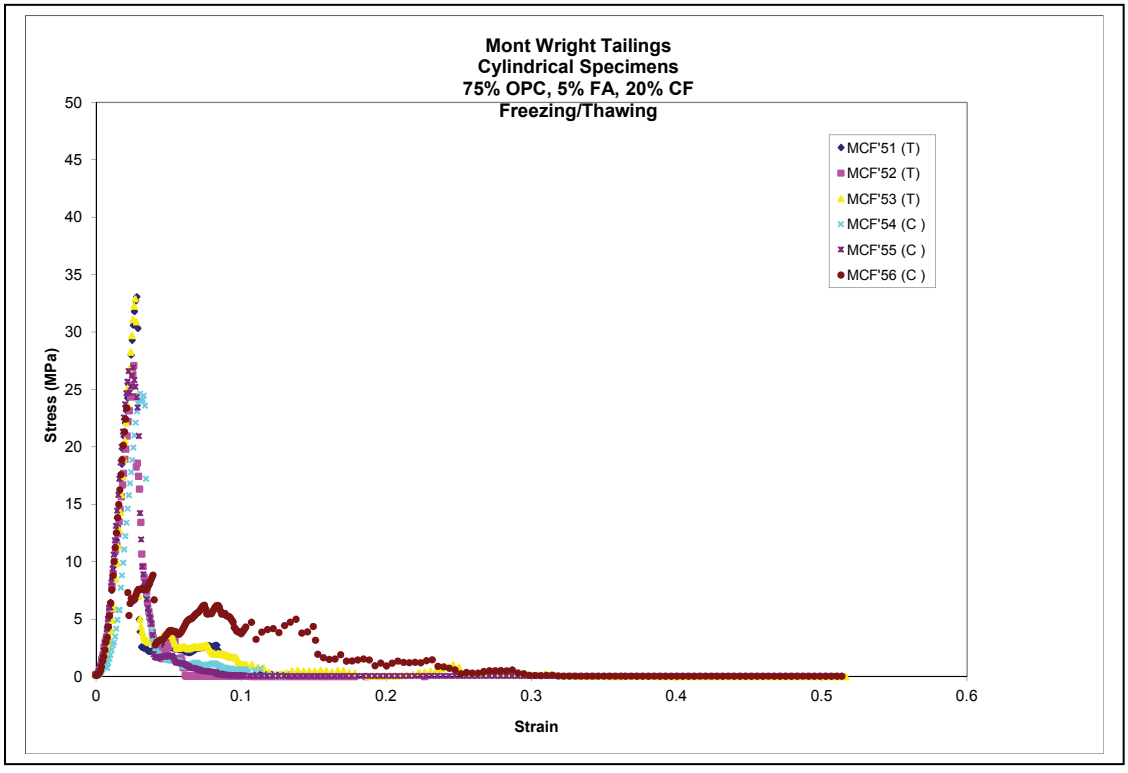


Figure A-24 Compression strength for Mont Wright samples MCF'51-MCF'56 after freezing/thawing

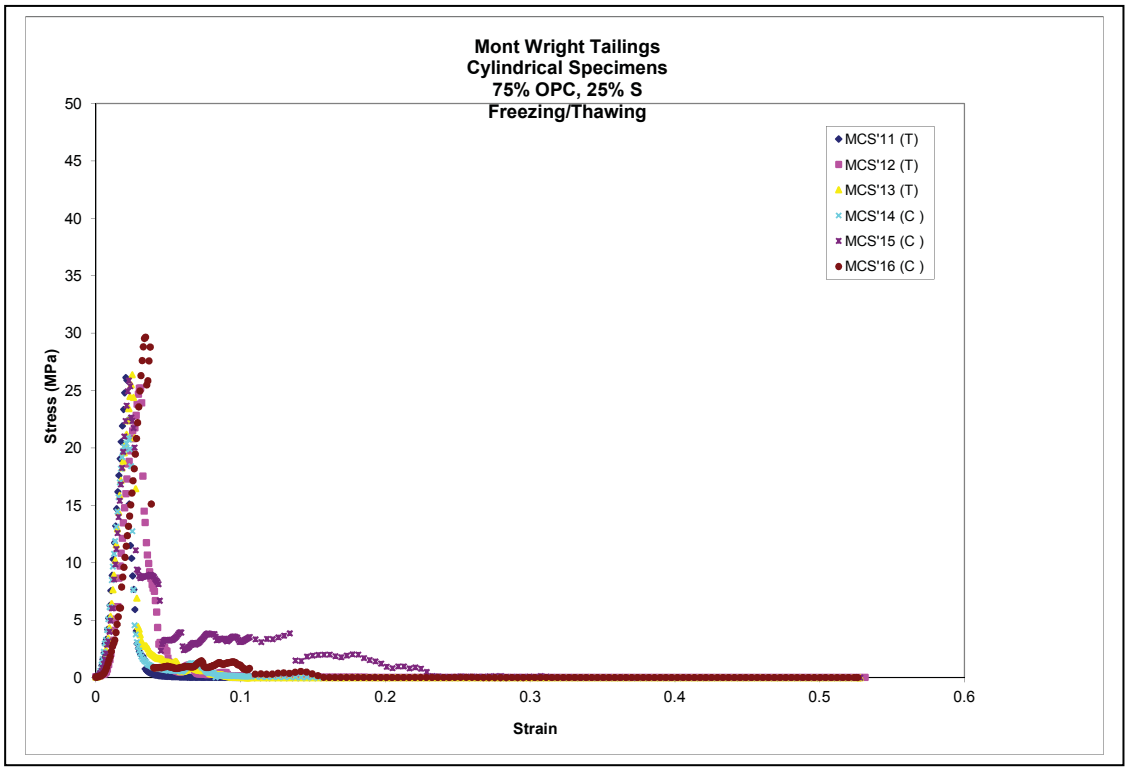


Figure A-25 Compression strength for Mont Wright samples MCS'11-MCS'16 after freezing/thawing

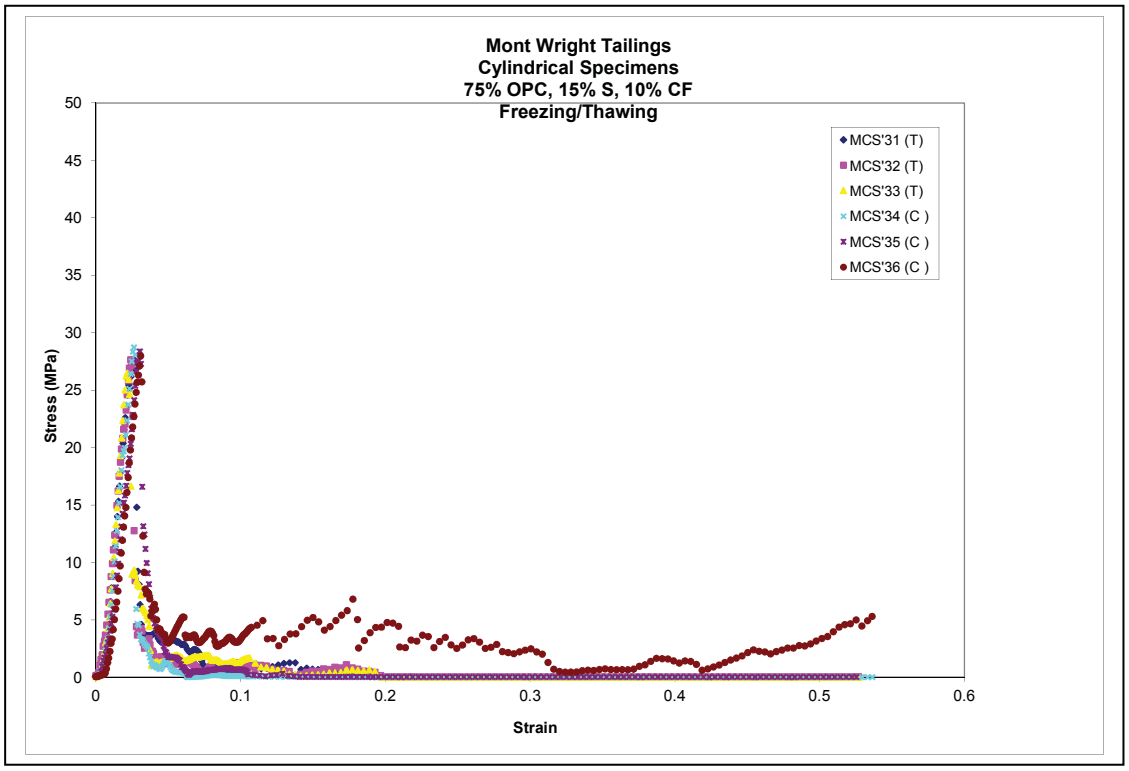


Figure A-26 Compression strength for Mont Wright samples MCS'31-MCS'36 after freezing/thawing

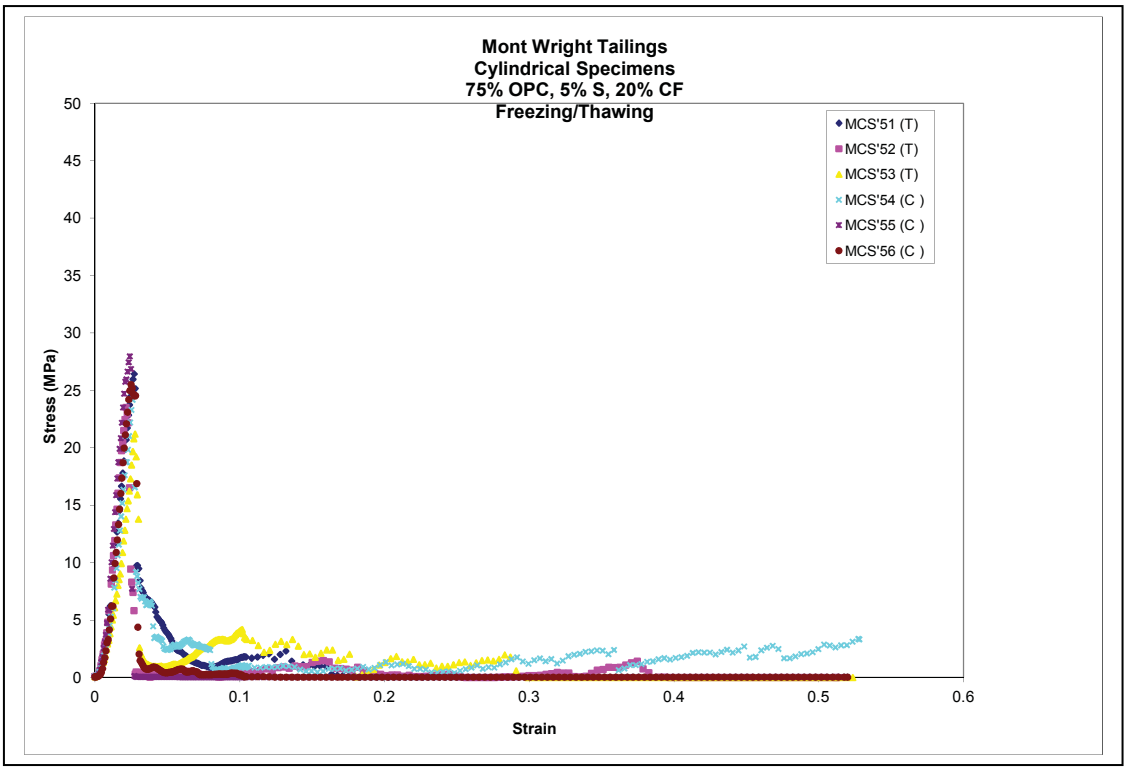


Figure A-27 Compression strength for Mont Wright samples MCS'51-MCS'56 after freezing/thawing

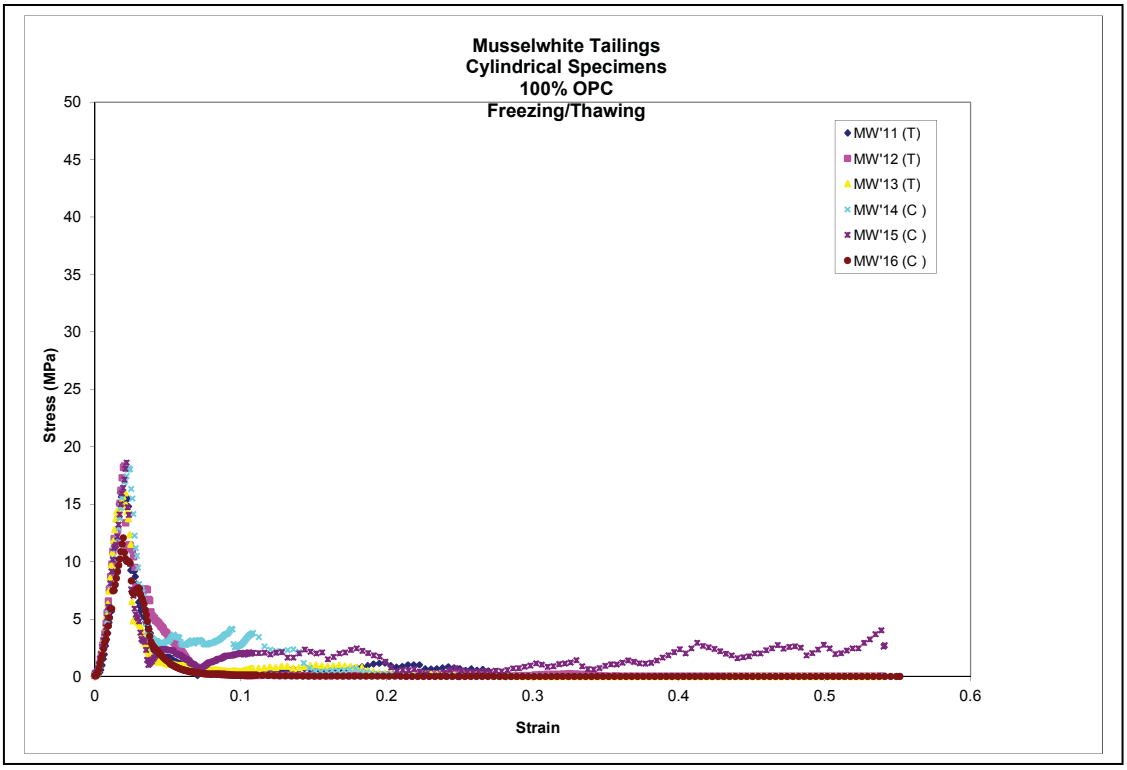


Figure A-28 Compression strength for Musselwhite samples MW'11-MW'16 after freezing/thawing

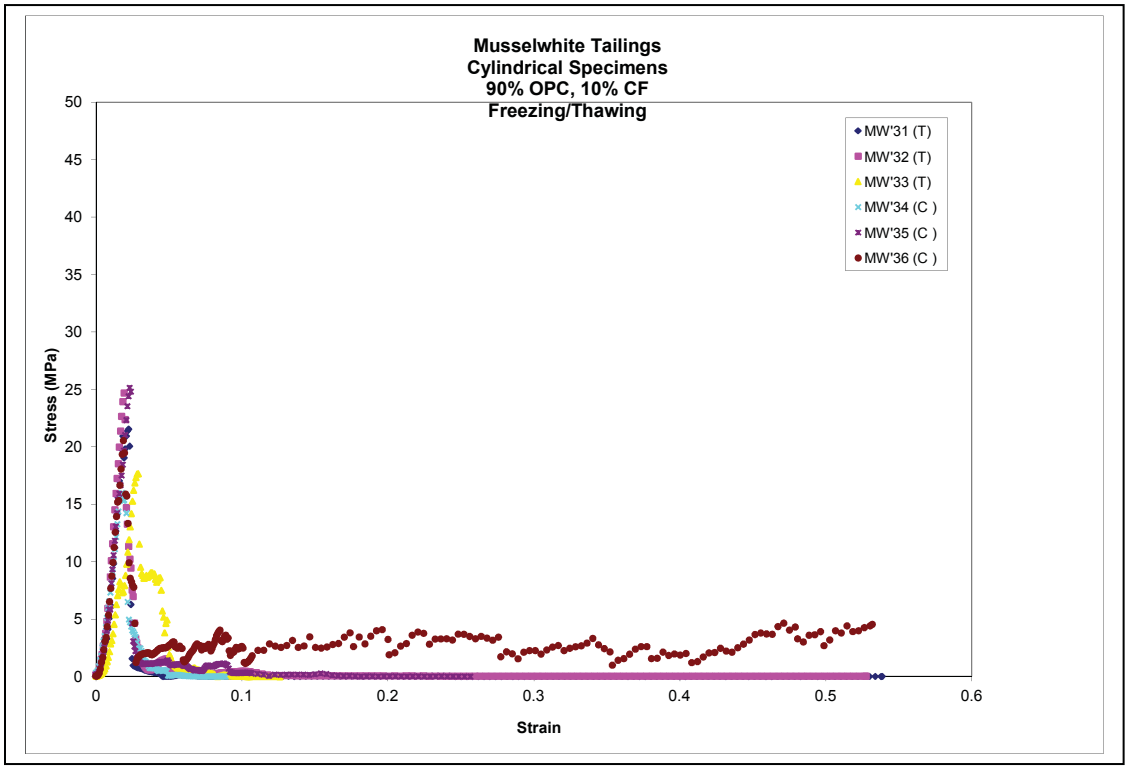


Figure A-29 Compression strength for Musselwhite samples MW'31-MW'36 after freezing/thawing

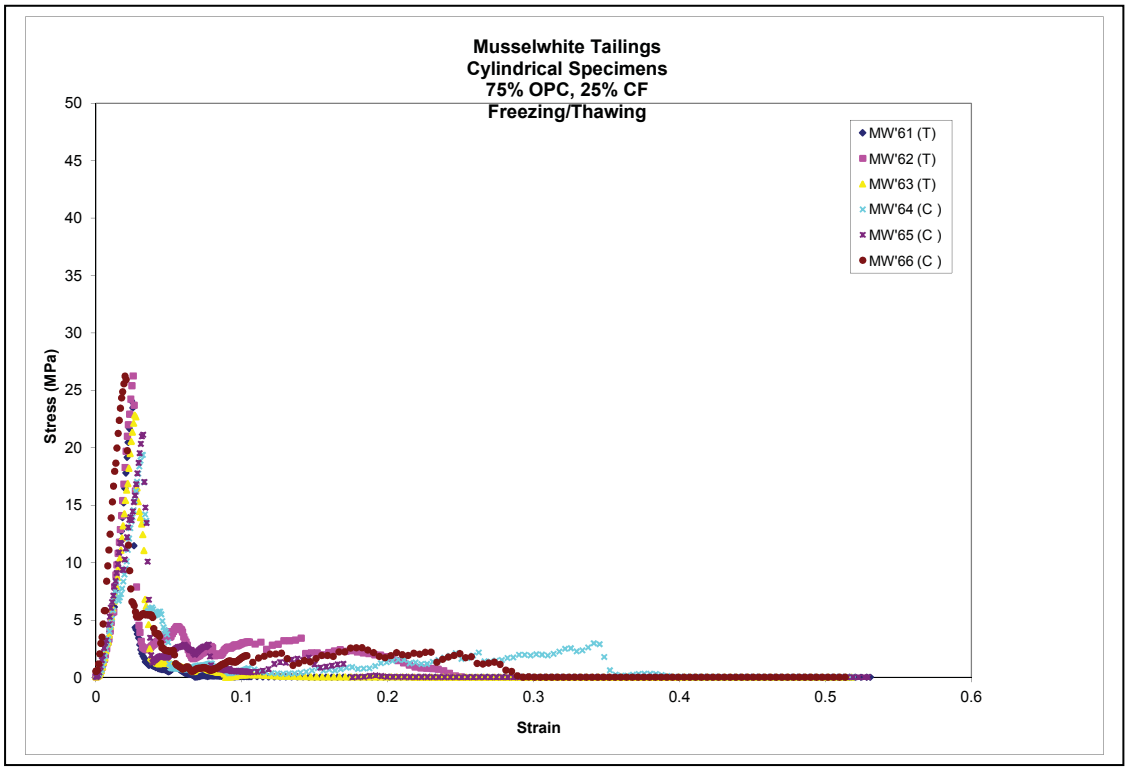


Figure A-30 Compression strength for Musselwhite samples MW'61-MW'66 after freezing/thawing

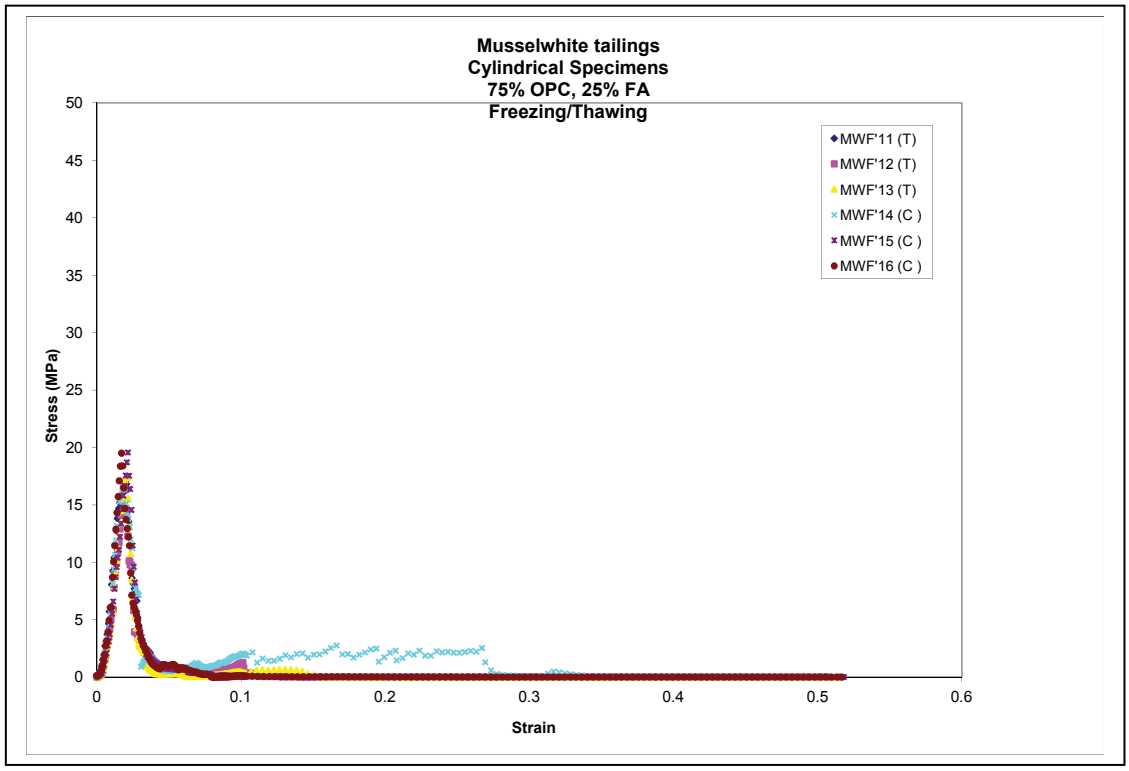


Figure A-31 Compression strength for Musselwhite samples MWF'11-MWF'16 after freezing/thawing

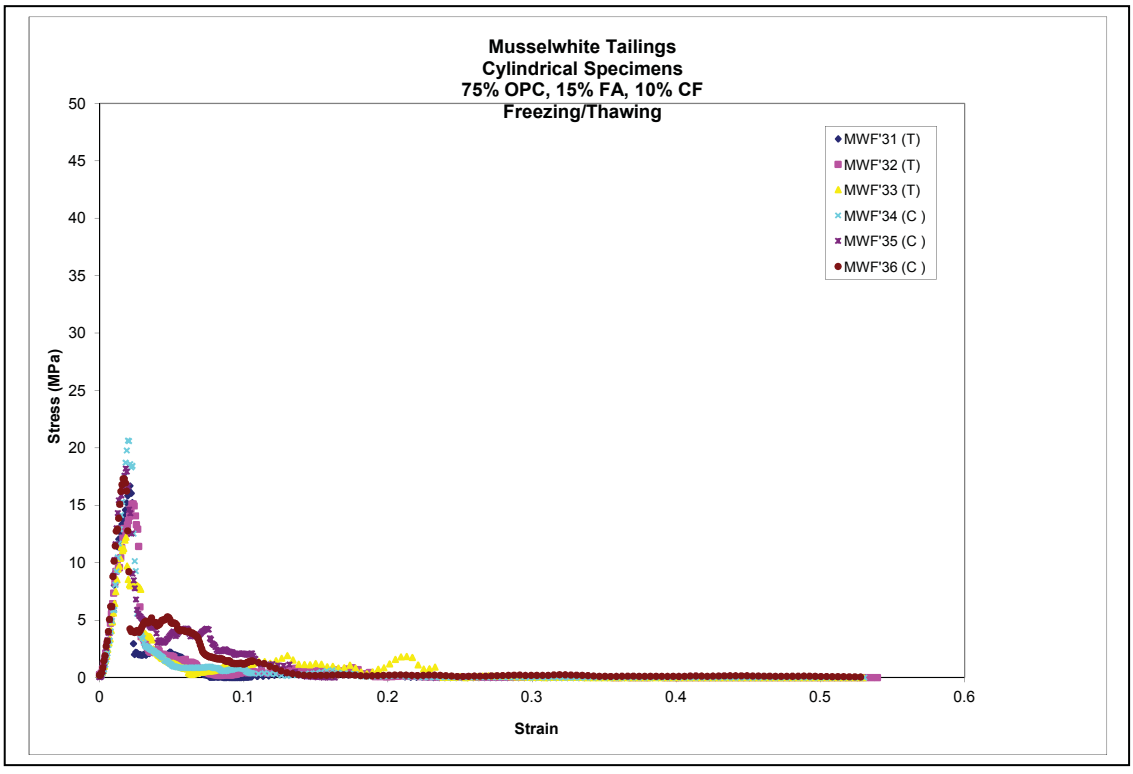


Figure A-32 Compression strength for Musselwhite samples MWF'31-MWF'36 after freezing/thawing

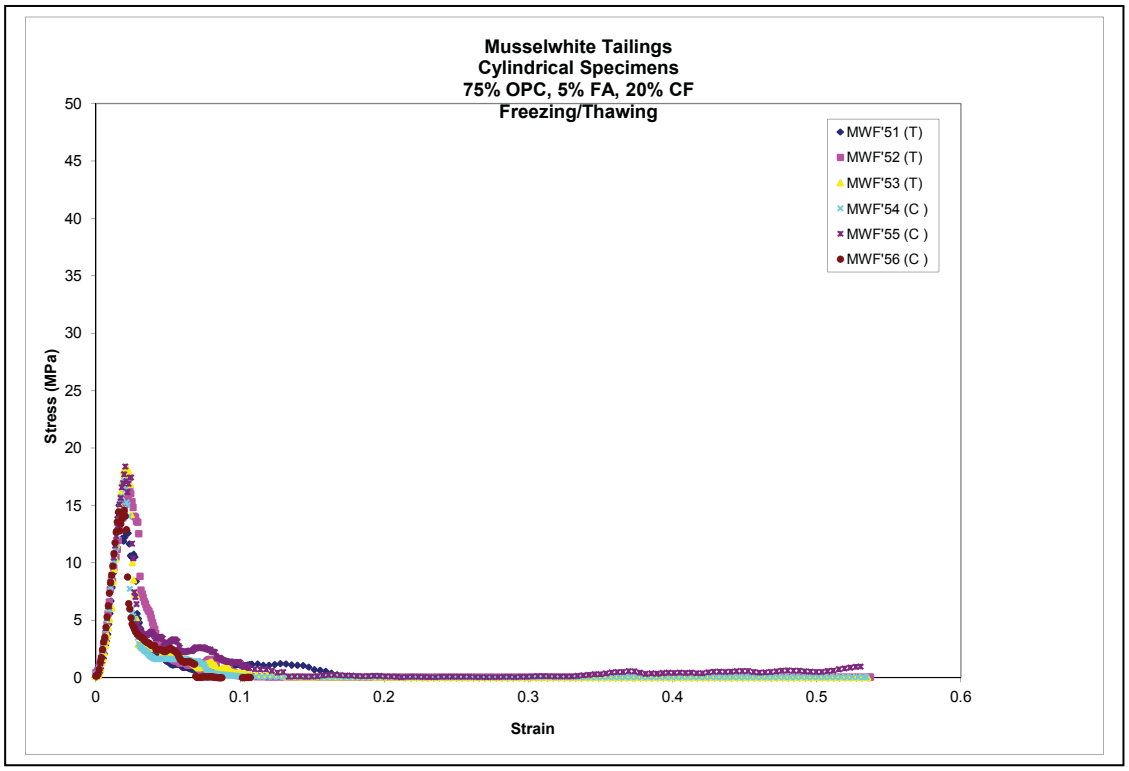


Figure A-33 Compression strength for Musselwhite samples MWF'51-MWF'56 after freezing/thawing

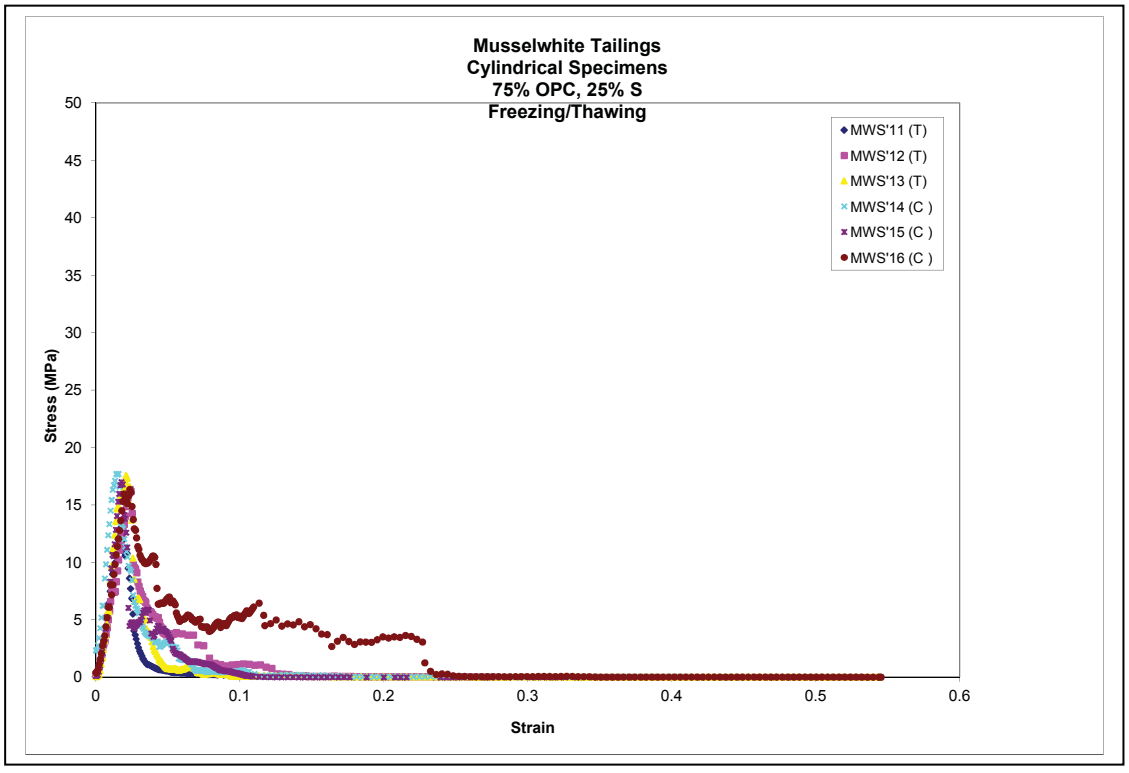


Figure A-34 Compression strength for Musselwhite samples MWS'11-MWS'16 after freezing/thawing

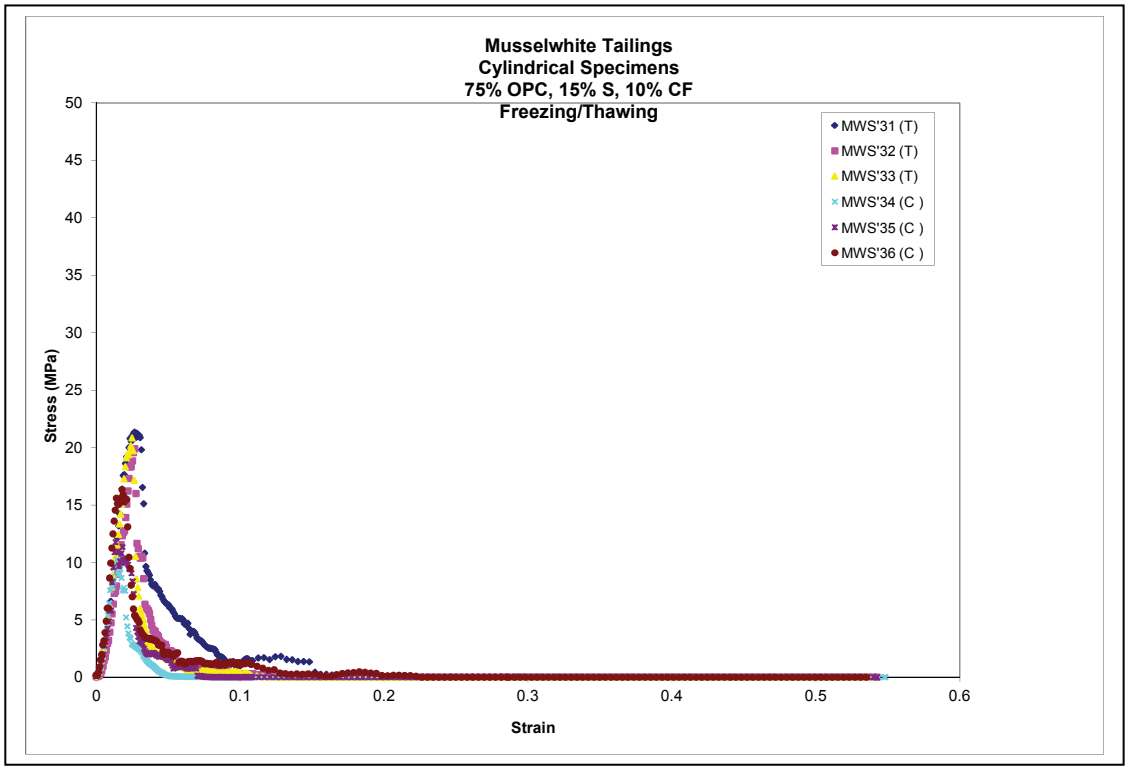


Figure A-35 Compression strength for Musselwhite samples MWS'31-MWS'36 after freezing/thawing

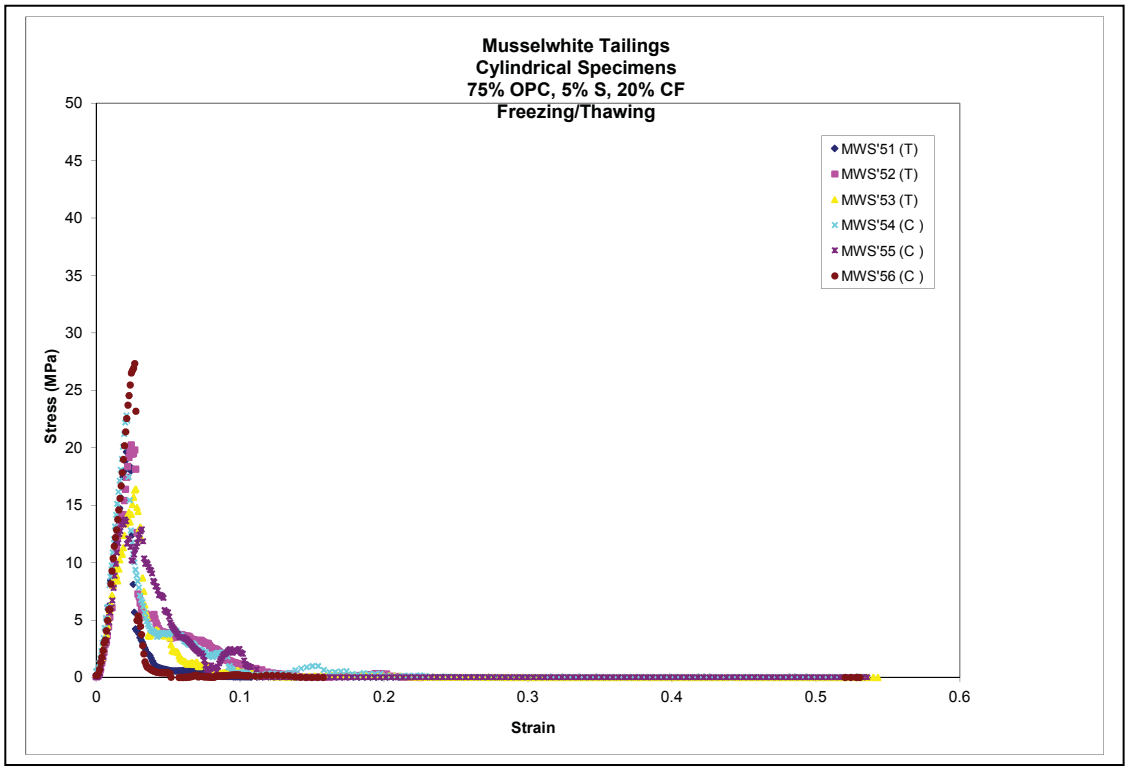


Figure A-36 Compression strength for Musselwhite samples MWS'51-MWS'56 after freezing/thawing

Appendix B:

Tables

Table B-1 Bulk density values for Mont Wright and Musselwhite matrices (1-*day* curing)

Cube #	Matrix	Density (gm/cm^3)	Standard Deviation
MC1	100% OPC	2.13	0.0451
MC2	95% OPC, 5% CF	2.123	0.0124
MC3	90% OPC, 10% CF	2.153	0.0261
MC4	85% OPC, 15% CF	2.142	0.0165
MC5	80% OPC, 20% CF	2.178	0.0143
MC6	75% OPC, 25% CF	2.207	0.0126
MW1	100% OPC	2.237	0.0042
MW2	95% OPC, 5% CF	2.242	0.014
MW3	90% OPC, 10% CF	2.225	0.013
MW4	85% OPC, 15% CF	2.231	0.0133
MW5	80% OPC, 20% CF	2.262	0.0045
MW6	75% OPC, 25% CF	2.231	0.0049

Table B-2 Bulk density values for Mont Wright and Musselwhite matrices (7-*day* curing)

Cube #	Matrix	Density (gm/cm^3)	Standard Deviation
MC(II)1	100% OPC	2.201	0.005
MC(II)2	95% OPC, 5% CF	2.164	0.011
MC(II)3	90% OPC, 10% CF	2.166	0.007
MC(II)4	85% OPC, 15% CF	2.178	0.007
MC(II)5	80% OPC, 20% CF	2.181	0.006
MC(II)6	75% OPC, 25% CF	2.142	0.008
MW(II)1	100% OPC	2.199	0.004
MW(II)2	95% OPC, 5% CF	2.183	0.015
MW(II)3	90% OPC, 10% CF	2.212	0.011
MW(II)4	85% OPC, 15% CF	2.294	0.015
MW(II)5	80% OPC, 20% CF	2.229	0.005
MW(II)6	75% OPC, 25% CF	2.256	0.01

Table B-3 Bulk density values for Mont Wright and Musselwhite matrices (28-*day* curing)

Cube #	Matrix	Density (*gm/cm*^{3})	Standard Deviation
MC(III)1	100% OPC	2.167	0.005
MC(III)2	95% OPC, 5% CF	2.167	0.004
MC(III)3	90% OPC, 10% CF	2.151	0.01
MC(III)4	85% OPC, 15% CF	2.157	0.009
MC(III)5	80% OPC, 20% CF	2.193	0.009
MC(III)6	75% OPC, 25% CF	2.195	0.007
MW(III)1	100% OPC	2.277	0.013
MW(III)2	95% OPC, 5% CF	2.278	0.025
MW(III)3	90% OPC, 10% CF	2.276	0.018
MW(III)4	85% OPC, 15% CF	2.31	0.005
MW(III)5	80% OPC, 20% CF	2.275	0.009
MW(III)6	75% OPC, 25% CF	2.272	0.011

Table B-4 Bulk density values for Mont Wright and Musselwhite matrices mixed with slag (1 *day* curing)

Cube #	Matrix	Density (*gm/cm*^{3})	Standard Deviation
MCS1	75% OPC, 25% S	2.185	0.0203
MCS2	75% OPC, 20% S, 5% CF	2.18	0.0119
MCS3	75% OPC, 15% S, 10% CF	2.161	0.0089
MCS4	75% OPC, 10% S, 15% CF	2.175	0.0016
MCS5	75% OPC, 5% S, 20% CF	2.212	0.0122
MWS1	75% OPC, 25% S	2.201	0.0060
MWS2	75% OPC, 20% S, 5% CF	2.196	0.0131
MWS3	75% OPC, 15% S, 10% CF	2.155	0.0085
MWS4	75% OPC, 10% S, 15% CF	2.167	0.014
MWS5	75% OPC, 5% S, 20% CF	2.2	0.0129

Table B-5 Bulk density values for Mont Wright and Musselwhite matrices mixed with fly ash (1 *day* curing)

Cube #	Matrix	Density (gm/cm^3)	Standard Deviation
MCF1	75% OPC, 25% FA	2.184	0.02391
MCF2	75% OPC, 20% FA, 5% CF	2.163	0.0158
MCF3	75% OPC, 15% FA, 10% CF	2.187	0.0087
MCF4	75% OPC, 10% FA, 15% CF	2.188	0.0153
MCF5	75% OPC, 5% FA, 20% CF	2.169	0.0133
MWF1	75% OPC, 25% FA	2.336	0.0114
MWF2	75% OPC, 20% FA, 5% CF	2.333	0.0060
MWF3	75% OPC, 15% FA, 10% CF	2.393	0.00797
MWF4	75% OPC, 10% FA, 15% CF	2.278	0.0144
MWF5	75% OPC, 5% FA, 20% CF	2.288	0.0143

Table B-6 Moisture content values of the hardened Mont Wright tailings matrices

Sample Code	Moisture Content (%)
MC (III) 11, 12, 13	9.77
MC (III) 21, 22, 23	9.54
MC (III) 51, 52, 53	8.65
MC 51, 52, 53	10.12
MC (III) 41, 42, 43	10.88
MC 61, 62, 63	10.04
MC 21, 22, 23	9.22
MC 31, 32, 33	8.92
MC 11, 12, 13	9.10
MC 41, 42, 43	9.1
MC (III) 61, 62, 63	9
MCS 11, 12, 13	6.44
MCS 21, 22, 23	6.9
MCS 31, 32, 33	6.82
MC (II) 11, 12, 13	8.75
MCS 51, 52, 53	7.92
MC (II) 41, 42, 43	9.71
MCF 31, 32, 33	6.31
MC (II) 61, 62, 63	8.96
MCF 21, 22, 23	6.36
MCF 51, 52, 53	6.38
MC (II) 21, 22, 23	8.98
MC (II) 51, 52, 53	8.41
MC (II) 31, 32, 33	9.33
MC (III) 31, 32, 33	7.42
MCS 41, 42, 43	7.17
MCF 11, 12, 13	5.98
MCF 41, 42, 43	7.1

Table B-7 Moisture content values of the Musselwhite tailings matrices

Sample Code	Moisture Content (%)
MW 41, 42, 43	13.39
MW 31, 32, 33	13.59
MW 51, 52, 53	12.06
MW 61, 62, 63	12.61
MW 11, 12, 13	14.13
MW (II) 11, 12, 13	12.98
MW (II) 21, 22, 23	10.33
MW (III) 21, 22, 23	9.62
MW (III) 11, 12, 13	10.5
MW (III) 31, 32, 33	10.02
MW (II) 51, 52, 53	10.03
MW (II) 31, 32, 33	9.86
MWF 21, 22, 23	9.73
MW 21, 22, 23	10.63
MW (III) 61, 62, 63	11
MW (II) 61, 62, 63	10.29
MWF 51, 52, 53	10.92
MW (III) 41, 42, 43	9.75
MW (II) 41, 42, 43	9.72
MW (III) 51, 52, 53	9.78
MWF 11, 12, 13	9.12
MWF 41, 42, 43	10.03
MWF 31, 32, 33	8.45
MWS 31, 32, 33	12.17
MWS 51, 52, 53	11.30
MWS 11, 12, 13	12.79
MWS 21, 22, 23	12.12
MWS 41, 42, 43	13.28

Table B-8 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite 5-*cm* cube strength tests after 1 day of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC	16.93	11.08	22.53	6.84	9.31	7.51	10.6	5.99
95% OPC, 5% CF	16.15	12.29	19.21	9.55	7.73	6.89	8.64	6.26
90% OPC, 10% CF	17.58	14.23	20.84	11.8	6.52	5.9	7.06	5.46
85% OPC, 15% CF	12.99	12.42	13.51	12	8.22	6.89	9.21	5.89
80% OPC, 20% CF	12.55	11.12	13.63	10.1	7.79	6.3	9.37	5.18
75% OPC, 25% CF	12.75	10.2	15.21	8.35	4.82	4.55	5.03	4.36

Table B-9 Data range and Student t 95% confidence interval for the Musselwhite 5-*cm* cube strength tests after 7 days of curing

Matrix	Musselwhite			
	Data Range		Student t	
	MAX	MIN	MAX	MIN
100% OPC	13.88	9.12	19.07	5.48
95% OPC, 5% CF	9.03	6.89	10.57	4.88
90% OPC, 10% CF	10.01	8.67	11.27	7.71
85% OPC, 15% CF	15.03	14.3	15.87	13.7
80% OPC, 20% CF	11.79	8.05	14.8	5.39
75% OPC, 25% CF	9.964	8.5	11.1	7.44

Table B-10 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite 5-*cm* cube strength tests after 28 days of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC	17.61	14.5	19.87	12.15	12.1	11.74	12.55	11.3
95% OPC, 5% CF	21.35	16.57	24.74	12.58	17.17	13.79	19.58	10.73
90% OPC, 10% CF	21.17	18.82	22.89	16.52	16.64	13.88	19.33	11.87
85% OPC, 15% CF	20.28	15.03	24.01	10.72	15.92	12.9	18.12	10.59
80% OPC, 20% CF	17.35	14.95	19.86	13.15	19.25	16.32	21.47	14.17
75% OPC, 25% CF	19.17	16.46	21.11	14.32	19.93	14.25	26.1	9.921

Table B-11 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite fly ash mixed 5-*cm* cube strength tests after 1 day of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
75% OPC, 25% FA	10.07	8.24	11.4	6.83	7.54	4.45	9.86	1.2
75% OPC, 20% FA, 5% CF	10.01	9.13	10.65	8.28	6.01	5.58	6.32	5.2
75% OPC, 15% FA, 10% CF	12.13	10.3	13.52	8.96	7.62	7.56	7.68	7.52
75% OPC, 10% FA, 15% CF	11.37	9.07	13.05	6.79	7.21	5.75	8.48	4.71
75% OPC, 5% FA, 20% CF	12.92	9.16	15.68	5.4	7.14	6.33	7.72	5.69

Table B-12 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite slag mixed 5-*cm* cube strength tests after 1 day of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
75% OPC, 25% S	11.11	9.59	12.25	7.99	4	3.59	4.3	3.24
75% OPC, 20% S, 5% CF	10.32	9.39	11.04	8.71	5.65	5.02	6.1	4.47
75% OPC, 15% S, 10% CF	9.662	7.7	11.34	6.32	5.57	4.63	6.24	3.8
75% OPC, 10% S, 15% CF	9.395	7.77	10.59	6.14	5.48	4.8	5.99	4.31
75% OPC, 5% S, 20% CF	8.363	7.49	9.007	6.84	4.57	3.51	5.63	2.73

Table B-13 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite 5-*cm* cubes bulk density after 1 day of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC	2.16	2.08	2.24	2.02	2.24	2.23	2.25	2.23
95% OPC, 5% CF	2.13	2.11	2.15	2.09	2.25	2.23	2.28	2.21
90% OPC, 10% CF	2.18	2.13	2.22	2.09	2.24	2.21	2.26	2.19
85% OPC, 15% CF	2.16	2.12	2.18	2.1	2.24	2.22	2.26	2.2
80% OPC, 20% CF	2.19	2.16	2.21	2.14	2.27	2.26	2.27	2.25
75% OPC, 25% CF	2.22	2.19	2.24	2.18	2.23	2.23	2.24	2.22

Table B-14 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite 5-*cm* cubes bulk density after 7 days of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC	2.21	2.2	2.21	2.19	2.2	2.2	2.21	2.19
95% OPC, 5% CF	2.17	2.15	2.19	2.14	2.2	2.17	2.22	2.14
90% OPC, 10% CF	2.17	2.16	2.18	2.15	2.22	2.2	2.24	2.19
85% OPC, 15% CF	2.18	2.17	2.2	2.16	2.31	2.28	2.33	2.26
80% OPC, 20% CF	2.19	2.18	2.2	2.16	2.23	2.22	2.24	2.22
75% OPC, 25% CF	2.15	2.14	2.16	2.12	2.27	2.25	2.28	2.23

Table B-15 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite 5-*cm* cubes bulk density after 28 days of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
100% OPC	2.17	2.16	2.18	2.15	2.29	2.27	2.31	2.24
95% OPC, 5% CF	2.17	2.16	2.18	2.16	2.3	2.25	2.34	2.22
90% OPC, 10% CF	2.16	2.14	2.18	2.13	2.3	2.26	2.32	2.23
85% OPC, 15% CF	2.16	2.15	2.18	2.13	2.32	2.3	2.32	2.3
80% OPC, 20% CF	2.2	2.18	2.22	2.17	2.28	2.26	2.3	2.25
75% OPC, 25% CF	2.2	2.19	2.21	2.18	2.28	2.26	2.3	2.24

Table B-16 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite fly ash mixed 5-*cm* cubes bulk density after 1 day of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
75% OPC, 25% FA	2.21	2.17	2.24	2.12	2.35	2.33	2.36	2.31
75% OPC, 20% FA, 5% CF	2.18	2.15	2.2	2.12	2.34	2.33	2.35	2.32
75% OPC, 15% FA, 10% CF	2.19	2.18	2.21	2.17	2.4	2.39	2.41	2.37
75% OPC, 10% FA, 15% CF	2.2	2.17	2.23	2.15	2.29	2.26	2.31	2.24
75% OPC, 5% FA, 20% CF	2.18	2.16	2.2	2.14	2.3	2.27	2.32	2.25

Table B-17 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite slag mixed 5-*cm* cubes bulk density after 1 day of curing

Matrix	Mont Wright				Musselwhite			
	Data Range		Student t		Data Range		Student t	
	MAX	MIN	MAX	MIN	MAX	MIN	MAX	MIN
75% OPC, 25% S	2.21	2.17	2.24	2.13	2.21	2.19	2.22	2.19
75% OPC, 20% S, 5% CF	2.19	2.17	2.21	2.15	2.21	2.18	2.23	2.16
75% OPC, 15% S, 10% CF	2.17	2.16	2.18	2.14	2.16	2.15	2.18	2.13
75% OPC, 10% S, 15% CF	2.18	2.17	2.18	2.17	2.18	2.16	2.2	2.13
75% OPC, 5% S, 20% CF	2.23	2.2	2.24	2.18	2.21	2.19	2.23	2.17

Table B-18 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 1 cycle of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0013	0.0011	0.0014	0.001
	(C)	0.0013	0.0011	0.0016	0.0009
	90% OPC, 10% CF (T)	0.0012	0.0012	0.0013	0.0012
	(C)	0.0016	0.0012	0.0018	0.0009
	75% OPC, 25% CF (T)	0.0015	0.0014	0.0015	0.0014
	(C)	0.0018	0.0014	0.0021	0.001
	75% OPC, 5% FA, 20% CF (T)	0.0013	0.0012	0.0014	0.0012
	(C)	0.0018	0.0012	0.0023	0.0005
	75% OPC, 25% S (T)	0.001	0.0009	0.0011	0.0008
	(C)	0.0013	0.001	0.0016	0.0007
	75% OPC, 15% S, 10% CF (T)	0.0011	0.001	0.0012	0.0009
	(C)	0.0014	0.001	0.0016	0.0006
	75% OPC, 5% S, 20% CF (T)	0.0012	0.001	0.0014	0.0009
	(C)	0.0017	0.001	0.0022	0.0003
Musselwhite	100% OPC (T)	0.0012	0.0011	0.0013	0.0011
	(C)	0.0012	0.0011	0.0013	0.0009
	90% OPC, 10% CF (T)	0.0013	0.0011	0.0014	0.0009
	(C)	0.0014	0.0011	0.0016	0.0009
	75% OPC, 25% CF (T)	0.002	0.0017	0.0022	0.0014
	(C)	0.0022	0.0015	0.0027	0.0008
	75% OPC, 25% FA (T)	0.001	0.0009	0.0011	0.0008
	(C)	0.0011	0.0009	0.0012	0.0008
	75% OPC, 15% FA, 10% CF (T)	0.0022	0.0011	0.003	0.0002
	(C)	0.0015	0.0011	0.0017	0.0007
	75% OPC, 5% FA, 20% CF (T)	0.0021	0.0018	0.0023	0.0016
	(C)	0.0024	0.0016	0.0029	0.0009
	75% OPC, 25% S (T)	0.001	0.0009	0.001	0.0009
	(C)	0.001	0.001	0.001	0.0009
	75% OPC, 15% S, 10% CF (T)	0.001	0.0009	0.001	0.0009
	(C)	0.0013	0.001	0.0016	0.0006
	75% OPC, 5% S, 20% CF (T)	0.0016	0.0015	0.0017	0.0015
	(C)	0.002	0.0016	0.0023	0.0012

Table B-19 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 2 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.001	0.0008	0.0012	0.0006
	(C)	0.0012	0.001	0.0012	0.001
	90% OPC, 10% CF (T)	0.0009	0.0008	0.0009	0.0008
	(C)	0.0012	0.0009	0.0014	0.0005
	75% OPC, 25% CF (T)	0.0008	0.0007	0.0008	0.0007
	(C)	0.0012	0.0008	0.0014	0.0004
	75% OPC, 5% FA, 20% CF (T)	0.0006	0.0005	0.0008	0.0004
	(C)	0.001	0.0006	0.0013	0.0002
	75% OPC, 25% S (T)	0.0006	0.0005	0.0007	0.0004
	(C)	0.0008	0.0006	0.0009	0.0005
	75% OPC, 15% S, 10% CF (T)	0.0007	0.0006	0.0007	0.0006
	(C)	0.0009	0.0007	0.001	0.0005
	75% OPC, 5% S, 20% CF (T)	0.0009	0.0007	0.001	0.0006
	(C)	0.0014	0.0007	0.0019	0.0001
Musselwhite	100% OPC (T)	0.0009	0.0008	0.001	0.0007
	(C)	0.0011	0.001	0.0011	0.001
	90% OPC, 10% CF (T)	0.0008	0.0007	0.0009	0.0007
	(C)	0.0013	0.0009	0.0015	0.0006
	75% OPC, 25% CF (T)	0.0009	0.0007	0.001	0.0006
	(C)	0.0015	0.0008	0.0021	6E-05
	75% OPC, 25% FA (T)	0.0007	0.0006	0.0008	0.0005
	(C)	0.0009	0.0007	0.0011	0.0005
	75% OPC, 15% FA, 10% CF (T)	0.0008	0.0007	0.0009	0.0006
	(C)	0.0013	0.0009	0.0016	0.0005
	75% OPC, 5% FA, 20% CF (T)	0.0009	0.0008	0.001	0.0007
	(C)	0.002	0.0008	0.0029	0
	75% OPC, 25% S (T)	0.0006	0.0004	0.0007	0.0003
	(C)	0.0008	0.0005	0.001	0.0003
	75% OPC, 15% S, 10% CF (T)	0.0006	0.0005	0.0006	0.0005
	(C)	0.0009	0.0006	0.0011	0.0003
	75% OPC, 5% S, 20% CF (T)	0.0008	0.0006	0.0009	0.0005
	(C)	0.0015	0.0006	0.0022	0

Table B-20 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 3 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0008	0.0007	0.0009	0.0006
	(C)	0.001	0.0009	0.0011	0.0007
	90% OPC, 10% CF (T)	0.0005	0.0005	0.0006	0.0004
	(C)	0.0009	0.0006	0.001	0.0004
	75% OPC, 25% CF (T)	0.0005	0.0004	0.0005	0.0004
	(C)	0.0008	0.0004	0.001	0.0001
	75% OPC, 5% FA, 20% CF (T)	0.0006	0.0005	0.0006	0.0004
	(C)	0.0009	0.0004	0.0012	0
	75% OPC, 25% S (T)	0.0005	0.0004	0.0005	0.0004
	(C)	0.0006	0.0005	0.0007	0.0004
	75% OPC, 15% S, 10% CF (T)	0.0005	0.0004	0.0006	0.0003
	(C)	0.0007	0.0006	0.0008	0.0005
	75% OPC, 5% S, 20% CF (T)	0.0007	0.0007	0.0008	0.0006
	(C)	0.0012	0.0007	0.0015	0.0003
Musselwhite	100% OPC (T)	0.0008	0.0007	0.0009	0.0006
	(C)	0.0009	0.0008	0.0009	0.0008
	90% OPC, 10% CF (T)	0.0007	0.0006	0.0008	0.0005
	(C)	0.001	0.0007	0.0012	0.0004
	75% OPC, 25% CF (T)	0.0007	0.0006	0.0008	0.0005
	(C)	0.0011	0.0007	0.0014	0.0003
	75% OPC, 25% FA (T)	0.0005	0.0005	0.0006	0.0005
	(C)	0.0007	0.0005	0.0009	0.0003
	75% OPC, 15% FA, 10% CF (T)	0.0008	0.0006	0.001	0.0004
	(C)	0.001	0.0006	0.0013	0.0002
	75% OPC, 5% FA, 20% CF (T)	0.0005	0.0004	0.0005	0.0003
	(C)	0.0009	0.0006	0.0012	0.0002
	75% OPC, 25% S (T)	0.0004	0.0003	0.0004	0.0002
	(C)	0.0004	0.0002	0.0005	0.0001
	75% OPC, 15% S, 10% CF (T)	0.0004	0.0003	0.0004	0.0003
	(C)	0.0008	0.0004	0.001	0.0002
	75% OPC, 5% S, 20% CF (T)	0.0004	0.0003	0.0006	0.0001
	(C)	0.0011	0.0004	0.0015	0

Table B-21 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 4 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0007	0.0006	0.0008	0.0005
	(C)	0.0008	0.0007	0.0009	0.0006
	90% OPC, 10% CF (T)	0.0004	0.0004	0.0004	0.0004
	(C)	0.0007	0.0005	0.0008	0.0004
	75% OPC, 25% CF (T)	0.0004	0.0003	0.0005	0.0002
	(C)	0.0007	0.0004	0.0009	0.0001
	75% OPC, 5% FA, 20% CF (T)	0.0006	0.0005	0.0007	0.0003
	(C)	0.0008	0.0005	0.001	0.0003
	75% OPC, 25% S (T)	0.0005	0.0005	0.0006	0.0004
	(C)	0.0006	0.0005	0.0007	0.0004
	75% OPC, 15% S, 10% CF (T)	0.0005	0.0005	0.0005	0.0005
	(C)	0.0007	0.0005	0.0008	0.0004
	75% OPC, 5% S, 20% CF (T)	0.0007	0.0007	0.0008	0.0006
	(C)	0.0008	0.0005	0.0011	0.0003
Musselwhite	100% OPC (T)	0.0006	0.0006	0.0007	0.0005
	(C)	0.0008	0.0006	0.0009	0.0005
	90% OPC, 10% CF (T)	0.0003	0.0002	0.0004	0.0001
	(C)	0.0005	0.0003	0.0007	0.0001
	75% OPC, 25% CF (T)	0.0012	0.0003	0.0019	0
	(C)	0.0008	0.0005	0.0011	0.0001
	75% OPC, 25% FA (T)	0.0005	0.0004	0.0005	0.0003
	(C)	0.0005	0.0004	0.0007	0.0002
	75% OPC, 15% FA, 10% CF (T)	0.0005	0.0004	0.0006	0.0004
	(C)	0.0008	0.0004	0.001	6E-05
	75% OPC, 5% FA, 20% CF (T)	0.0005	0.0004	0.0006	0.0003
	(C)	0.001	0.0006	0.0013	0.0002
	75% OPC, 25% S (T)	0.0004	0.0003	0.0004	0.0002
	(C)	0.0004	0.0003	0.0005	0.0002
	75% OPC, 15% S, 10% CF (T)	0.0004	0.0003	0.0004	0.0002
	(C)	0.0005	0.0004	0.0006	0.0003
	75% OPC, 5% S, 20% CF (T)	0.0004	0.0003	0.0006	0.0001
	(C)	0.0009	0.0003	0.0014	0

Table B-22 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 5 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0006	0.0005	0.0007	0.0003
	(C)	0.0007	0.0005	0.0008	0.0004
	90% OPC, 10% CF (T)	0.0004	0.0003	0.0005	0.0002
	(C)	0.0006	0.0004	0.0007	0.0002
	75% OPC, 25% CF (T)	0.0003	0.0003	0.0003	0.0003
	(C)	0.0007	0.0004	0.0009	0.0002
	75% OPC, 5% FA, 20% CF (T)	0.0004	5E-05	0.0008	0
	(C)	0.0006	0.0004	0.0008	0.0002
	75% OPC, 25% S (T)	0.0004	0.0004	0.0005	0.0003
	(C)	0.0006	0.0003	0.0008	0
	75% OPC, 15% S, 10% CF (T)	0.0004	0.0004	0.0004	0.0003
	(C)	0.0005	0.0004	0.0006	0.0003
	75% OPC, 5% S, 20% CF (T)	0.0005	0.0003	0.0006	0.0002
	(C)	0.0006	0.0003	0.0008	0
Musselwhite	100% OPC (T)	0.0003	0.0002	0.0004	0.0002
	(C)	0.0005	0.0002	0.0008	0
	90% OPC, 10% CF (T)	0.0004	0.0003	0.0005	0.0001
	(C)	0.0006	0.0004	0.0007	0.0002
	75% OPC, 25% CF (T)	0.0005	0.0004	0.0006	0.0003
	(C)	0.0007	0.0004	0.001	9E-05
	75% OPC, 25% FA (T)	0.0004	0.0003	0.0004	0.0002
	(C)	0.0004	0.0004	0.0005	0.0003
	75% OPC, 15% FA, 10% CF (T)	0.0005	0.0003	0.0007	0.0002
	(C)	0.0005	0.0003	0.0006	0.0002
	75% OPC, 5% FA, 20% CF (T)	0.0006	0.0004	0.0007	0.0002
	(C)	0.0008	0.0006	0.001	0.0004
	75% OPC, 25% S (T)	0.0003	0.0003	0.0003	0.0003
	(C)	0.0003	0.0002	0.0004	0.0002
	75% OPC, 15% S, 10% CF (T)	0.0004	0.0003	0.0004	0.0002
	(C)	0.0004	4E-05	0.0007	0
	75% OPC, 5% S, 20% CF (T)	0.0004	0.0004	0.0004	0.0003
	(C)	0.0007	0.0003	0.001	0

Table B-23 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 6 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0004	0.0002	0.0006	9E-05
	(C)	0.0004	0.0004	0.0005	0.0003
	90% OPC, 10% CF (T)	0.0004	0.0004	0.0004	0.0004
	(C)	0.0005	0.0004	0.0006	0.0003
	75% OPC, 25% CF (T)	0.0005	0.0003	0.0006	0.0002
	(C)	0.0006	0.0003	0.0009	4E-05
	75% OPC, 5% FA, 20% CF (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0005	0.0002	0.0007	0
	75% OPC, 25% S (T)	0.0003	0.0002	0.0003	0.0002
	(C)	0.0003	0.0002	0.0003	0.0001
	75% OPC, 15% S, 10% CF (T)	0.0003	0.0003	0.0004	0.0002
	(C)	0.0004	0.0004	0.0005	0.0003
	75% OPC, 5% S, 20% CF (T)	0.0005	0.0004	0.0006	0.0002
	(C)	0.0006	0.0003	0.0008	9E-05
Musselwhite	100% OPC (T)	0.0004	0.0003	0.0005	0.0002
	(C)	0.0004	0.0003	0.0004	0.0003
	90% OPC, 10% CF (T)	0.0003	0.0002	0.0004	0.0001
	(C)	0.0005	0.0003	0.0007	0.0001
	75% OPC, 25% CF (T)	0.0006	0.0003	0.0007	0.0001
	(C)	0.0007	0.0004	0.0008	0.0002
	75% OPC, 25% FA (T)	0.0003	0.0003	0.0004	0.0002
	(C)	0.0004	0.0003	0.0004	0.0003
	75% OPC, 15% FA, 10% CF (T)	0.0005	0.0003	0.0006	0.0002
	(C)	0.0005	0.0003	0.0006	0.0001
	75% OPC, 5% FA, 20% CF (T)	0.0005	0.0003	0.0006	0.0002
	(C)	0.0006	0.0004	0.0008	0.0002
	75% OPC, 25% S (T)	0.0002	0.0002	0.0003	0.0001
	(C)	0.0003	0.0002	0.0004	0.0001
	75% OPC, 15% S, 10% CF (T)	0.0003	0.0003	0.0003	0.0003
	(C)	0.0004	0.0001	0.0006	0
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0003	0.0003	0.0003
	(C)	0.0006	0.0003	0.0009	0

Table B-24 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 7 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0004	0.0003	0.0005	0.0002
	(C)	0.0004	0.0003	0.0005	0.0002
	90% OPC, 10% CF (T)	0.0003	0.0002	0.0004	9E-05
	(C)	0.0004	0.0003	0.0004	0.0003
	75% OPC, 25% CF (T)	0.0003	0.0002	0.0004	0.0002
	(C)	0.0005	0.0002	0.0007	3E-05
	75% OPC, 5% FA, 20% CF (T)	0.0003	0.0002	0.0003	0.0001
	(C)	0.0004	0.0002	0.0005	7E-05
	75% OPC, 25% S (T)	0.0003	0.0002	0.0003	0.0002
	(C)	0.0003	0.0002	0.0004	7E-05
	75% OPC, 15% S, 10% CF (T)	0.0003	0.0002	0.0004	9E-05
	(C)	0.0004	0.0003	0.0004	0.0002
	75% OPC, 5% S, 20% CF (T)	0.0005	0.0004	0.0005	0.0003
	(C)	0.0005	0.0003	0.0007	0.0001
Musselwhite	100% OPC (T)	0.0004	0.0002	0.0005	5E-05
	(C)	0.0003	0.0002	0.0005	0.0001
	90% OPC, 10% CF (T)	0.0003	0.0003	0.0003	0.0003
	(C)	0.0004	0.0002	0.0006	0
	75% OPC, 25% CF (T)	0.0005	0.0003	0.0007	0.0002
	(C)	0.0011	0.0003	0.0016	0
	75% OPC, 25% FA (T)	0.0003	0.0003	0.0003	0.0003
	(C)	0.0003	0.0002	0.0004	0.0002
	75% OPC, 15% FA, 10% CF (T)	0.0004	0.0002	0.0006	7E-05
	(C)	0.0004	0.0002	0.0005	0.0001
	75% OPC, 5% FA, 20% CF (T)	0.0004	0.0003	0.0005	0.0002
	(C)	0.0006	0.0004	0.0008	0.0001
	75% OPC, 25% S (T)	0.0001	0.0001	0.0001	0.0001
	(C)	0.0002	0.0001	0.0002	9E-05
	75% OPC, 15% S, 10% CF (T)	0.0003	0.0002	0.0004	7E-05
	(C)	0.0004	0.0002	0.0005	3E-05
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0003	0.0004	0.0002
	(C)	0.0004	0.0003	0.0006	0.0002

Table B-25 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 8 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0006	0.0001	0.0009	0
	(C)	0.0004	0.0004	0.0005	0.0003
	90% OPC, 10% CF (T)	0.0003	0.0002	0.0004	7E-05
	(C)	0.0004	0.0003	0.0004	0.0002
	75% OPC, 25% CF (T)	0.0002	0.0002	0.0002	0.0002
	(C)	0.0004	0.0002	0.0005	6E-05
	75% OPC, 5% FA, 20% CF (T)	0.0003	0.0001	0.0005	0
	(C)	0.0004	0.0003	0.0004	0.0002
	75% OPC, 25% S (T)	0.0003	0.0001	0.0004	4E-05
	(C)	0.0003	0.0002	0.0003	0.0001
	75% OPC, 15% S, 10% CF (T)	0.0002	0.0002	0.0002	0.0002
	(C)	0.0003	0.0002	0.0003	0.0002
	75% OPC, 5% S, 20% CF (T)	0.0004	0.0003	0.0005	0.0003
	(C)	0.0005	0.0001	0.0008	0
Musselwhite	100% OPC (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0003	0.0001	0.0004	2E-05
	90% OPC, 10% CF (T)	0.0003	0.0003	0.0003	0.0003
	(C)	0.0003	0.0002	0.0004	0.0002
	75% OPC, 25% CF (T)	0.0007	0.0003	0.0009	0
	(C)	0.0005	0.0003	0.0007	0.0001
	75% OPC, 25% FA (T)	0.0003	0.0002	0.0004	0.0001
	(C)	0.0002	0.0002	0.0003	0.0001
	75% OPC, 15% FA, 10% CF (T)	0.0003	0.0002	0.0004	8E-05
	(C)	0.0004	0.0001	0.0005	0
	75% OPC, 5% FA, 20% CF (T)	0.0003	0.0002	0.0004	8E-05
	(C)	0.0005	0.0003	0.0006	0.0002
	75% OPC, 25% S (T)	0.0002	0.0001	0.0002	1E-04
	(C)	0.0002	0.0001	0.0003	3E-05
	75% OPC, 15% S, 10% CF (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0003	0.0001	0.0004	4E-06
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0003	0.0004	0.0002
	(C)	0.0004	0.0003	0.0006	7E-05

Table B-26 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 9 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0003	0.0003	0.0004	0.0002
	(C)	0.0004	0.0002	0.0006	9E-05
	90% OPC, 10% CF (T)	0.0002	0.0002	0.0003	0.0001
	(C)	0.0004	0.0003	0.0004	0.0002
	75% OPC, 25% CF (T)	0.0002	0.0002	0.0002	0.0002
	(C)	0.0004	0.0002	0.0005	6E-05
	75% OPC, 5% FA, 20% CF (T)	0.0003	0.0002	0.0003	0.0001
	(C)	0.0003	0.0001	0.0005	2E-06
	75% OPC, 25% S (T)	0.0003	0.0002	0.0003	0.0001
	(C)	0.0002	0.0002	0.0003	0.0001
	75% OPC, 15% S, 10% CF (T)	0.0002	0.0002	0.0003	0.0001
	(C)	0.0003	0.0001	0.0004	4E-06
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0003	0.0004	0.0002
	(C)	0.0003	0.0002	0.0004	7E-05
Musselwhite	100% OPC (T)	0.0002	9E-05	0.0004	0
	(C)	0.0002	0.0002	0.0002	0.0002
	90% OPC, 10% CF (T)	0.0004	0.0002	0.0005	3E-05
	(C)	0.0003	0.0001	0.0004	0
	75% OPC, 25% CF (T)	0.0003	0.0002	0.0005	0.0001
	(C)	0.0004	0.0003	0.0005	0.0003
	75% OPC, 25% FA (T)	0.0003	0.0002	0.0003	0.0001
	(C)	0.0002	9E-05	0.0003	0
	75% OPC, 15% FA, 10% CF (T)	0.0003	0.0002	0.0004	0.0002
	(C)	0.0003	0.0001	0.0004	0
	75% OPC, 5% FA, 20% CF (T)	0.0003	0.0002	0.0004	0.0002
	(C)	0.0004	0.0002	0.0006	6E-06
	75% OPC, 25% S (T)	0.0002	9E-05	0.0002	0
	(C)	0.0001	9E-05	0.0002	6E-05
	75% OPC, 15% S, 10% CF (T)	0.0002	5E-05	0.0004	0
	(C)	0.0003	0.0001	0.0004	3E-05
	75% OPC, 5% S, 20% CF (T)	0.0002	0.0002	0.0003	0.0001
	(C)	0.0004	0.0002	0.0006	1E-06

Table B-27 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 10 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0002	9E-05	0.0003	0
	(C)	0.0001	9E-05	0.0002	4E-05
	90% OPC, 10% CF (T)	0.0002	0.0002	0.0003	0.0001
	(C)	0.0004	0.0002	0.0005	2E-05
	75% OPC, 25% CF (T)	0.0002	0.0002	0.0003	0.0001
	(C)	0.0003	0.0002	0.0003	0.0001
	75% OPC, 5% FA, 20% CF (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0004	5E-05	0.0006	0
	75% OPC, 25% S (T)	0.0002	0.0001	0.0002	9E-05
	(C)	0.0003	9E-05	0.0004	0
	75% OPC, 15% S, 10% CF (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0003	0.0001	0.0004	0
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0003	0.0004	0.0002
	(C)	0.0004	0.0001	0.0005	0
Musselwhite	100% OPC (T)	0.0002	0.0001	0.0002	9E-05
	(C)	0.0002	0.0001	0.0002	8E-05
	90% OPC, 10% CF (T)	0.0005	9E-05	0.0008	0
	(C)	0.0002	0.0001	0.0003	6E-05
	75% OPC, 25% CF (T)	0.0014	0.0003	0.0022	0
	(C)	0.0004	0.0003	0.0005	0.0002
	75% OPC, 25% FA (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0002	9E-05	0.0003	0
	75% OPC, 15% FA, 10% CF (T)	0.0003	0.0002	0.0003	0.0002
	(C)	0.0003	0.0001	0.0004	0
	75% OPC, 5% FA, 20% CF (T)	0.0003	0.0001	0.0004	4E-05
	(C)	0.0003	0.0003	0.0004	0.0002
	75% OPC, 25% S (T)	0.0001	9E-05	0.0002	5E-05
	(C)	0.0001	0	0.0002	0
	75% OPC, 15% S, 10% CF (T)	0.0002	5E-05	0.0003	0
	(C)	0.0002	0.0001	0.0003	4E-05
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0002	0.0003	0.0002
	(C)	0.0003	0.0002	0.0004	0.0001

Table B-28 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 11 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0003	5E-05	0.0005	0
	(C)	0.0002	9E-05	0.0004	0
	90% OPC, 10% CF (T)	0.0002	0.0001	0.0003	6.9E-05
	(C)	0.0001	0.0001	0.0001	0.00013
	75% OPC, 25% CF (T)	0.0002	0.0001	0.0003	6.8E-05
	(C)	0.0004	0.0002	0.0005	1.7E-05
	75% OPC, 5% FA, 20% CF (T)	0.0002	0.0001	0.0002	0.0001
	(C)	0.0003	5E-05	0.0005	0
	75% OPC, 25% S (T)	0.0002	0.0002	0.0003	0.00013
	(C)	0.0002	5E-05	0.0004	0
	75% OPC, 15% S, 10% CF (T)	0.0002	0	0.0004	0
	(C)	0.0002	9E-05	0.0003	0
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0002	0.0004	0.00016
	(C)	0.0004	0.0002	0.0005	0
Musselwhite	100% OPC (T)	0.0001	9E-05	0.0002	4E-05
	(C)	0.0002	9E-05	0.0002	0
	90% OPC, 10% CF (T)	0.0002	0.0001	0.0002	9.8E-05
	(C)	0.0003	0.0001	0.0004	0
	75% OPC, 25% CF (T)	0.0003	0.0002	0.0003	0.00019
	(C)	0.0004	0.0001	0.0006	0
	75% OPC, 25% FA (T)	0.0001	9E-05	0.0002	4E-05
	(C)	0.0001	5E-05	0.0002	0
	75% OPC, 15% FA, 10% CF (T)	0.0004	0.0001	0.0006	0
	(C)	0.0002	0.0001	0.0003	3.8E-05
	75% OPC, 5% FA, 20% CF (T)	0.0002	0.0002	0.0003	0.00015
	(C)	0.0003	0.0003	0.0004	0.00024
	75% OPC, 25% S (T)	0.0001	0	0.0002	0
	(C)	0.0001	5E-05	0.0002	0
	75% OPC, 15% S, 10% CF (T)	9E-05	0	0.0002	0
	(C)	0.0001	0	0.0002	0
	75% OPC, 5% S, 20% CF (T)	0.0001	0.0001	0.0001	0.00013
	(C)	0.0003	0.0001	0.0004	0

Table B-29 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite cylindrical matrices percentage weight loss after 12 cycles of freezing and thawing

Tailings	Matrix	Data Range		Student t	
		MAX	MIN	MAX	MIN
Mont Wright	100% OPC (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0002	9E-05	0.0003	0
	90% OPC, 10% CF (T)	0.0002	9E-05	0.0003	0
	(C)	0.0002	0.0002	0.0003	0.0001
	75% OPC, 25% CF (T)	0.0001	0.0001	0.0001	0.0001
	(C)	0.0003	0.0001	0.0004	0
	75% OPC, 5% FA, 20% CF (T)	0.0003	0.0001	0.0004	2E-05
	(C)	0.0003	5E-05	0.0005	0
	75% OPC, 25% S (T)	0.0002	5E-05	0.0004	0
	(C)	0.0001	9E-05	0.0002	4E-05
	75% OPC, 15% S, 10% CF (T)	0.0001	9E-05	0.0002	6E-05
	(C)	0.0002	9E-05	0.0002	3E-05
	75% OPC, 5% S, 20% CF (T)	0.0003	0.0002	0.0003	0.0002
	(C)	0.0004	0.0001	0.0005	0
Musselwhite	100% OPC (T)	0.0002	0.0002	0.0003	0.0001
	(C)	0.0002	9E-05	0.0002	2E-05
	90% OPC, 10% CF (T)	0.0002	0.0001	0.0003	7E-05
	(C)	0.0003	0.0002	0.0003	0.0001
	75% OPC, 25% CF (T)	0.0003	0.0002	0.0005	7E-05
	(C)	0.0003	0.0002	0.0004	0.0002
	75% OPC, 25% FA (T)	0.0002	0.0001	0.0002	9E-05
	(C)	9E-05	0	0.0002	0
	75% OPC, 15% FA, 10% CF (T)	0.0002	5E-05	0.0003	0
	(C)	0.0002	9E-05	0.0003	3E-05
	75% OPC, 5% FA, 20% CF (T)	0.0002	9E-05	0.0003	0
	(C)	0.0003	0.0002	0.0003	0.0001
	75% OPC, 25% S (T)	0.0001	5E-05	0.0002	0
	(C)	9E-05	0	0.0002	0
	75% OPC, 15% S, 10% CF (T)	9E-05	0	0.0002	0
	(C)	0.0001	0	0.0002	0
	75% OPC, 5% S, 20% CF (T)	9E-05	0	0.0002	0
	(C)	0.0003	0.0001	0.0004	4E-05

Table B-30 Data range and Student t 95% confidence interval for the Mont Wright and Musselwhite TCLP tests

Heavy Metal		**Cr (*mg/L*)**	**Cu (*mg/L*)**	**Ni (*mg/L*)**	**Zn (*mg/L*)**	**Fe (*mg/L*)**	**Pb (*mg/L*)**
Data Range	**MIN**	0	0	0	0	0	0
	MAX	0	0	0	0	1	0
Student t	**MAX**	0.112	0.022	1.995	0.025	1.644	0.13
	MIN	0	0	0	0	0	0

Table B-31 Specific gravity values for Mont Wright tailings matrices after 1, 7 and 28 days curing

Matrix	Specific Gravity	Standard Deviation
95% OPC, 5% CF (7 *days*), Mont Wright	2.72	0.06
90% OPC, 10% CF (28 *days*), Mont Wright	2.85	0.05
90% OPC, 10% CF (7 *days*), Mont Wright	2.57	0.09
95% OPC, 5% CF (28 *days*) Mont Wright	2.92	0.11
80% OPC, 20% CF (7 *days*), Mont Wright	3.34	0.19
100% OPC (7 *days*), Mont Wright	3.15	0.199
100% OPC (1 *day*), Mont Wright	2.79	0.17
75% OPC, 25% CF (1 *day*) Mont Wright	2.99	0.14
95% OPC, 5% CF (1 *day*), Mont Wright	2.92	0.23
90% OPC, 10% CF (1 *day*), Mont Wright	2.76	0.07
100% OPC (28 *days*), Mont Wright	2.79	0.11
75% OPC, 25% CF (7 *days*), Mont Wright	3.25	0.16
75% OPC, 25% CF (28 *days*), Mont Wright	3.04	0.02

Table B-32 Specific gravity values for Musselwhite tailings matrices after 1, 7 and 28 days curing

Matrix	Specific Gravity	Standard Deviation
90% OPC, 10% CF (1 *day*), Musselwhite	3.19	0.55
100% OPC (1 day), Musselwhite	3.34	0.09
75% OPC, 25% CF (1 *day*), Musselwhite	3.3	0.07
100% OPC (7 *days*), Musselwhite	3.34	0.098
90% OPC, 10% CF (7 *days*), Musselwhite	3.05	0.06
75% OPC, 25% CF (7 *days*), Musselwhite	3.11	0.10
100 OPC (28 *days*), Musselwhite	3.16	0.16
90% OPC, 10% CF (28 *days*), Musselwhite	3.24	0.02
75% OPC, 25% CF (28 days), Musselwhite	*3.32*	0.05

Table B-33 Specific gravity values for fly ash and slag tailings matrices after 1 day curing

Matrix	Specific Gravity	Standard Deviation
75% OPC, 25% FA (1 *day*), Mont Wright	2.98	0.097
75% OPC, 15% FA, 10% CF (1 *day*), Mont Wright	2.98	0.086
75% OPC, 25% S (1 *day*), Mont Wright	2.8	0.117
75% OPC, 15% S, 10% CF (1 *day*), Mont Wright	2.91	0.024
75% OPC, 5% FA, 20% CF (1 *day*), Mont Wright	3.09	0.14
75% OPC, 5% S, 20% CF (1 *day*), Mont Wright	3.01	0.038
75% OPC, 25% FA (1 *day*), Musselwhite	3.14	0.1
75% OPC, 15% FA, 10% CF (1 *day*), Musslewhite	3.22	0.052
75% OPC, 25% S (1 *day*), Musselwhite	3.20	0.15
75% OPC, 15% S, 10% CF (1 *day*), Musselwhite	3.15	0.095
75% OPC, 5% FA, 20% CF (1 *day*), Musselwhite	3.15	0.076
75% OPC, 5% S, 20% CF (1 *day*), Musselwhite	3.07	0.045

Appendix C:
Experimental and Computational Figures

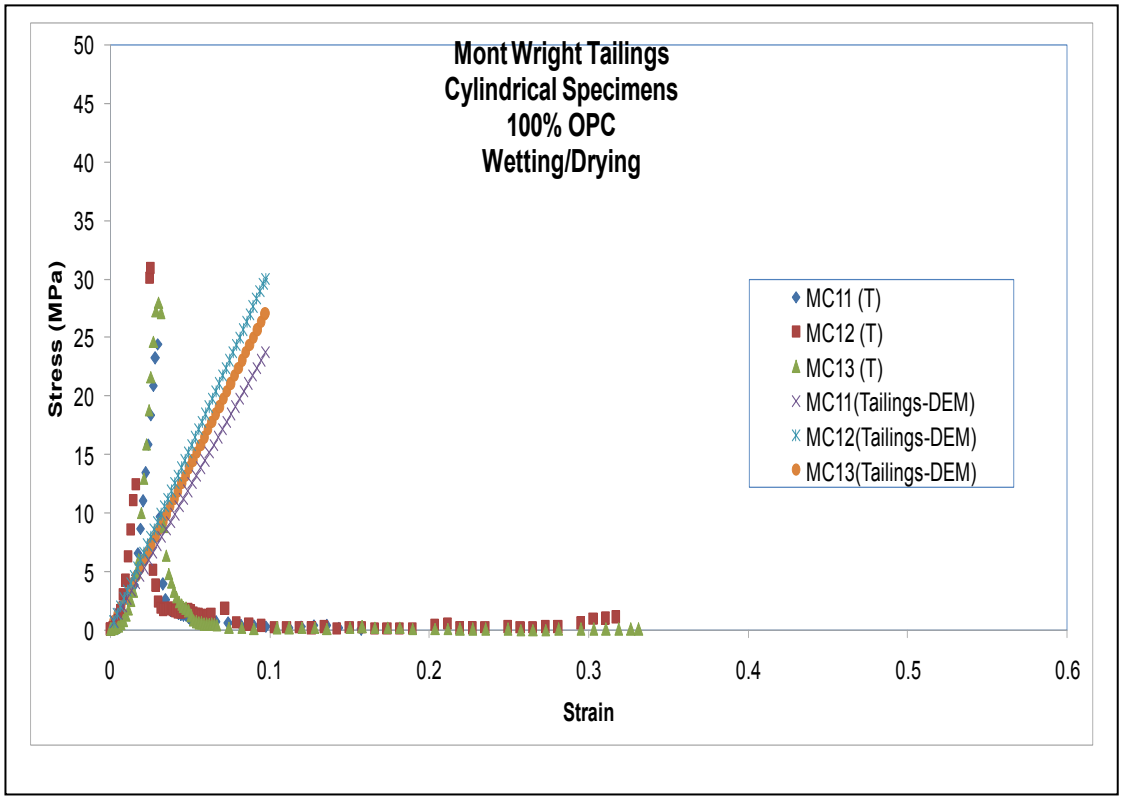


Figure C-1 Computational and experimental UCS values for Mont Wright samples MC11-MC13 after wetting/drying

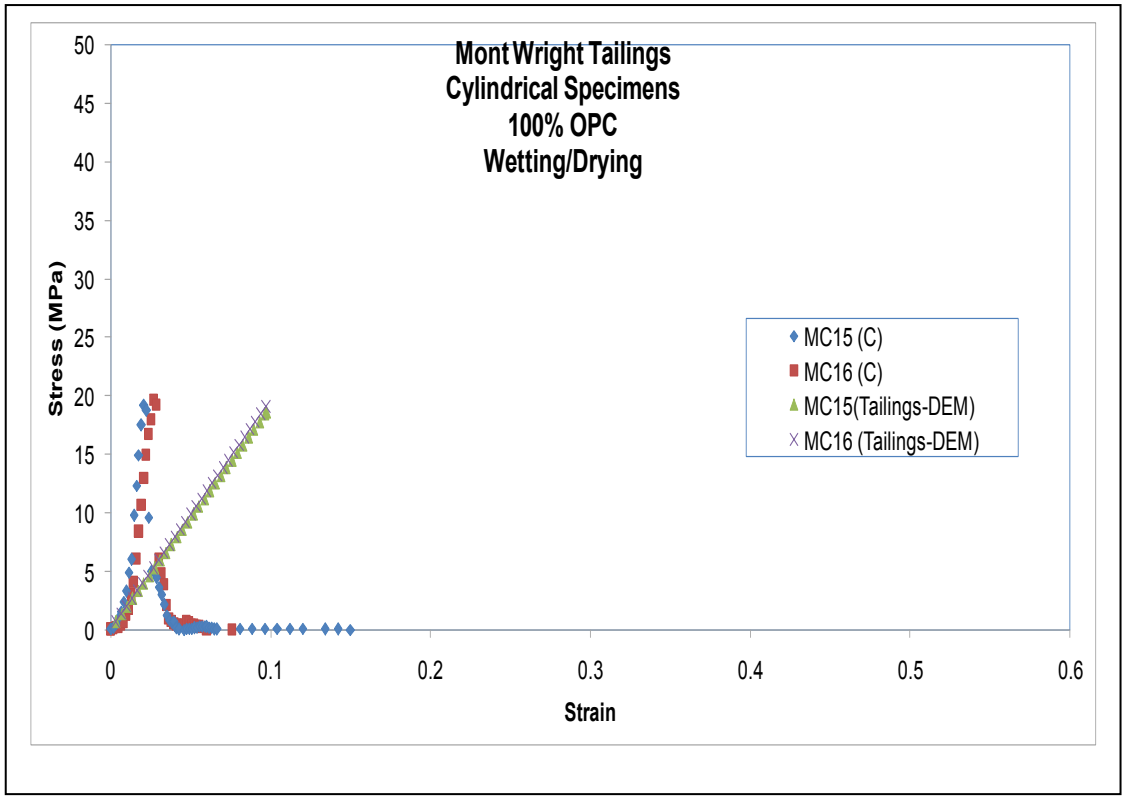


Figure C-2 Computational and experimental UCS values for Mont Wright samples MC15-MC16 after wetting/drying

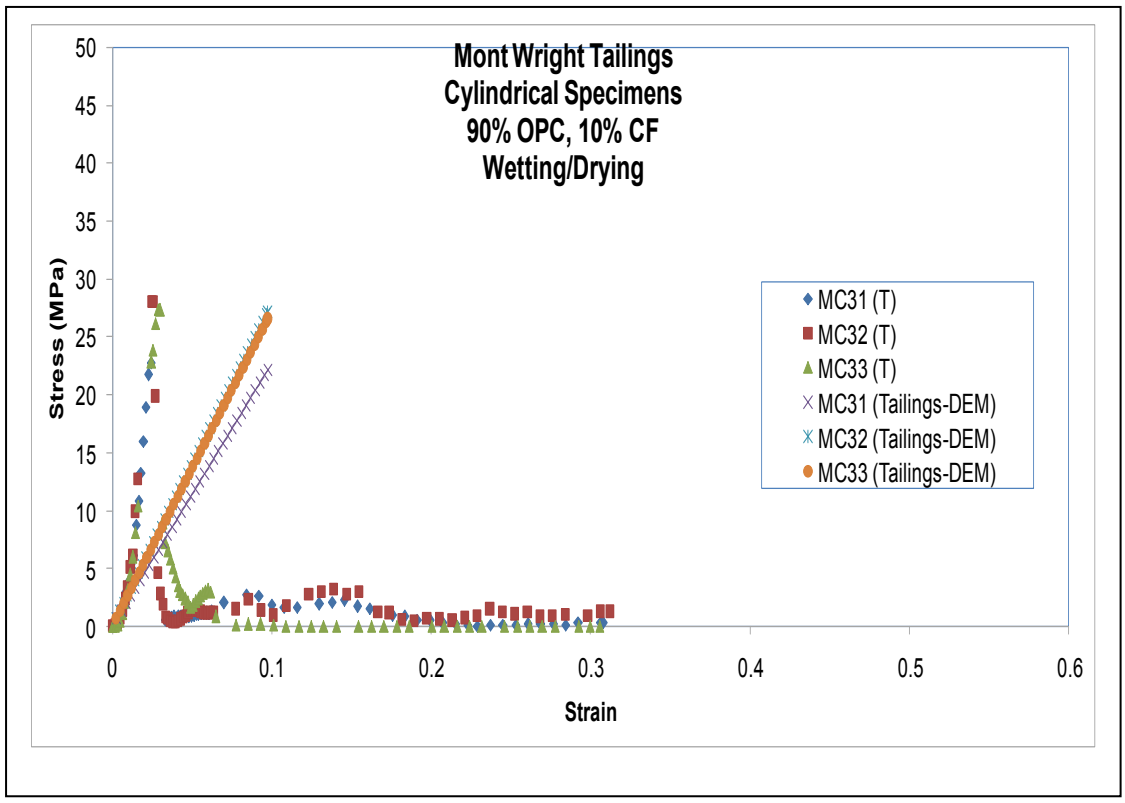


Figure C-3 Computational and experimental UCS values for Mont Wright samples MC31-MC33 after wetting/drying

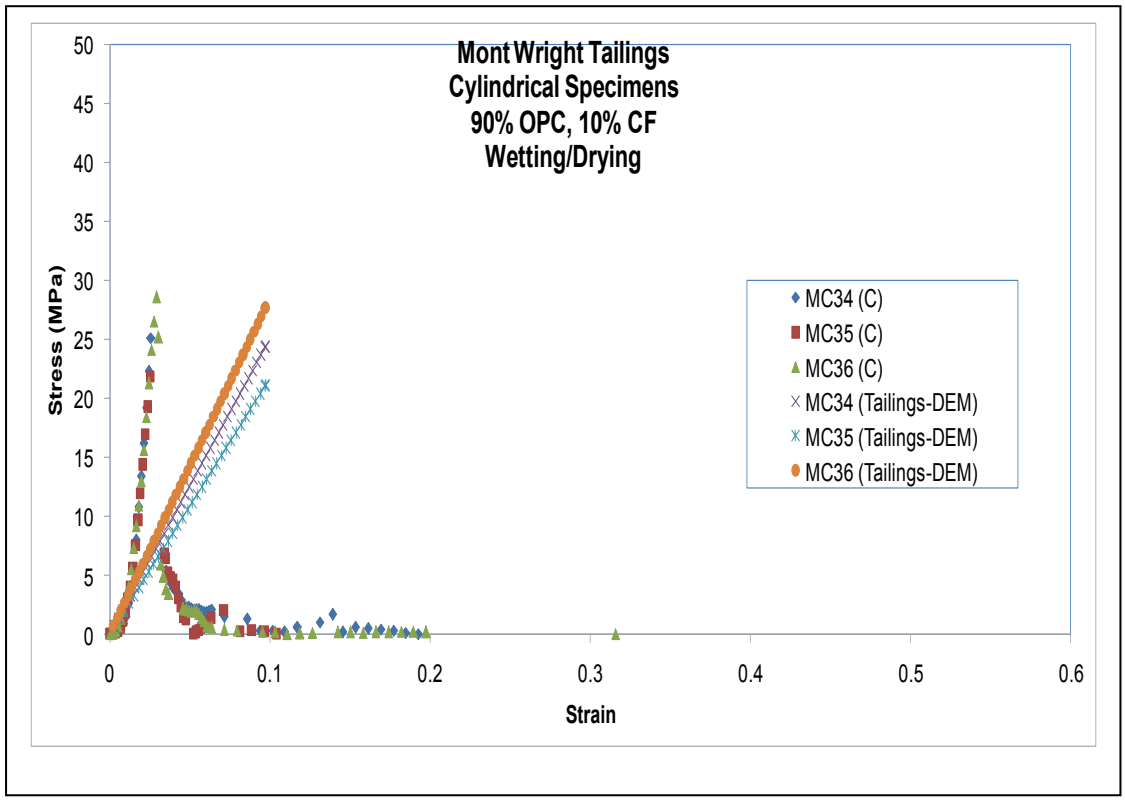


Figure C-4 Computational and experimental UCS values for Mont Wright samples MC34-MC36 after wetting/drying

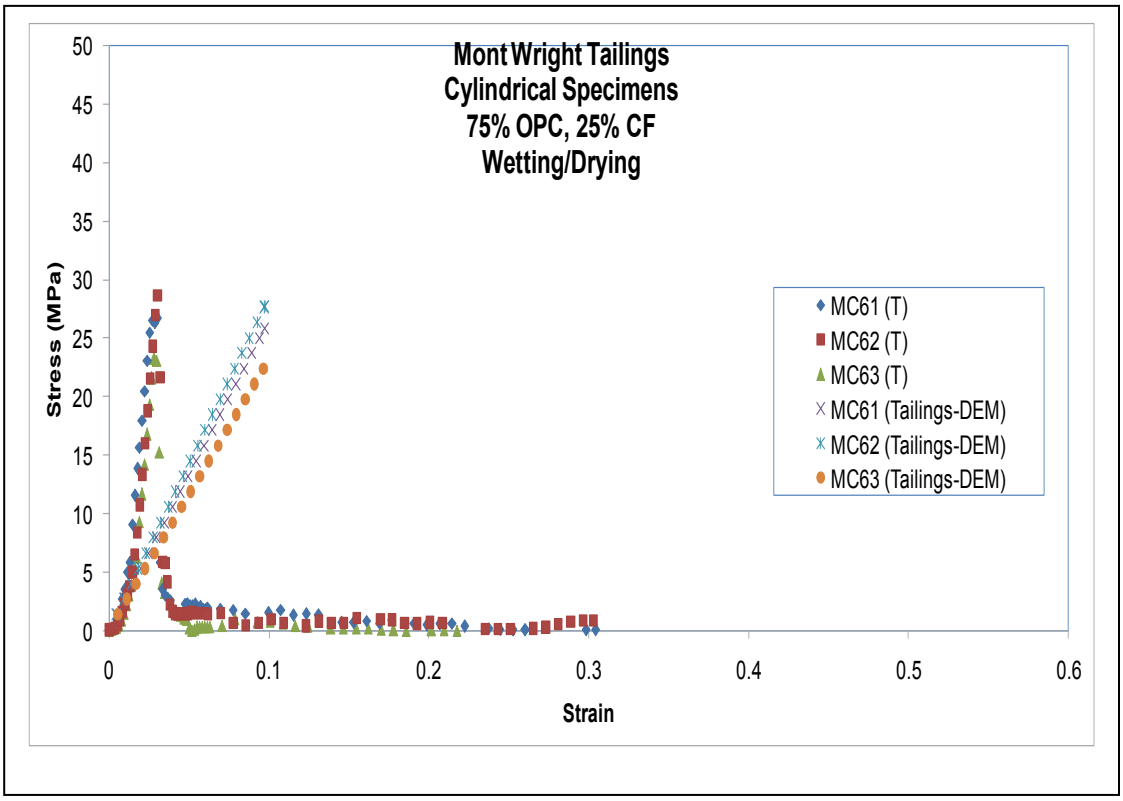


Figure C-5 Computational and experimental UCS values for Mont Wright samples MC61-MC63 after wetting/drying

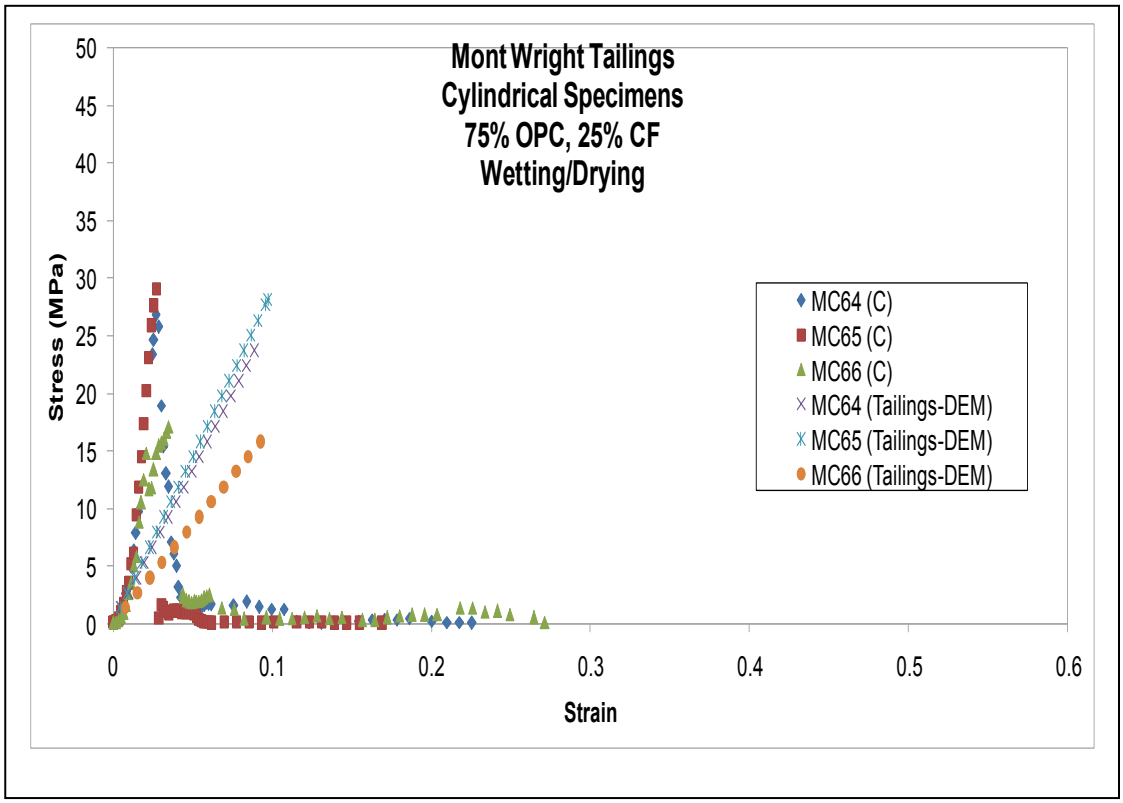


Figure C-6 Computational and experimental UCS values for Mont Wright samples MC64-MC66 after wetting/drying

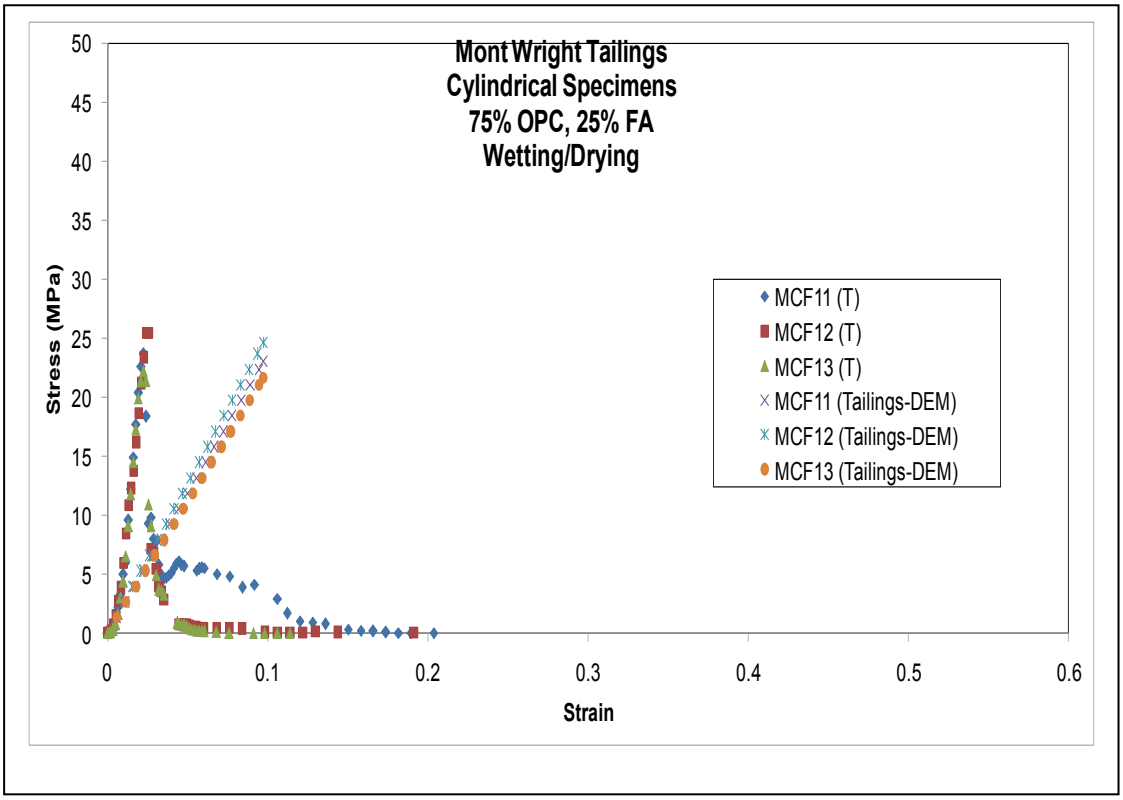


Figure C-7 Computational and experimental UCS values for Mont Wright samples MCF11-MCF13 after wetting/drying

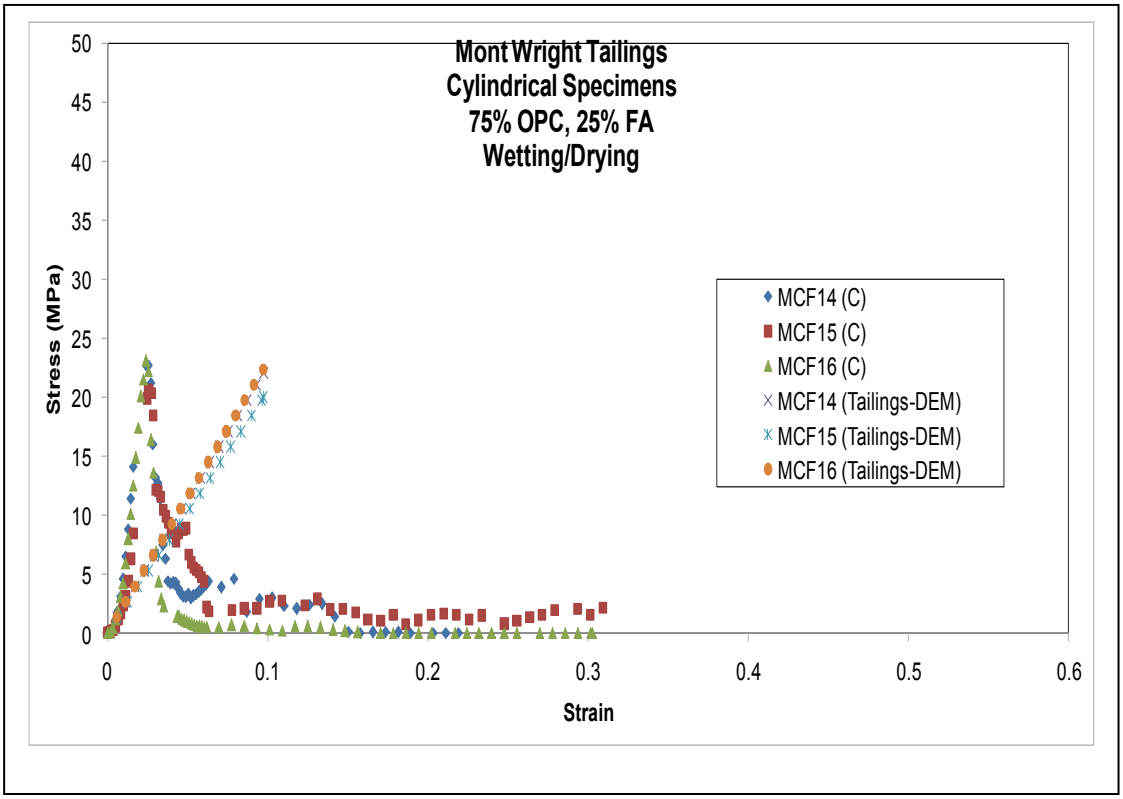


Figure C-8 Computational and experimental UCS values for Mont Wright samples MCF14-MCF16 after wetting/drying

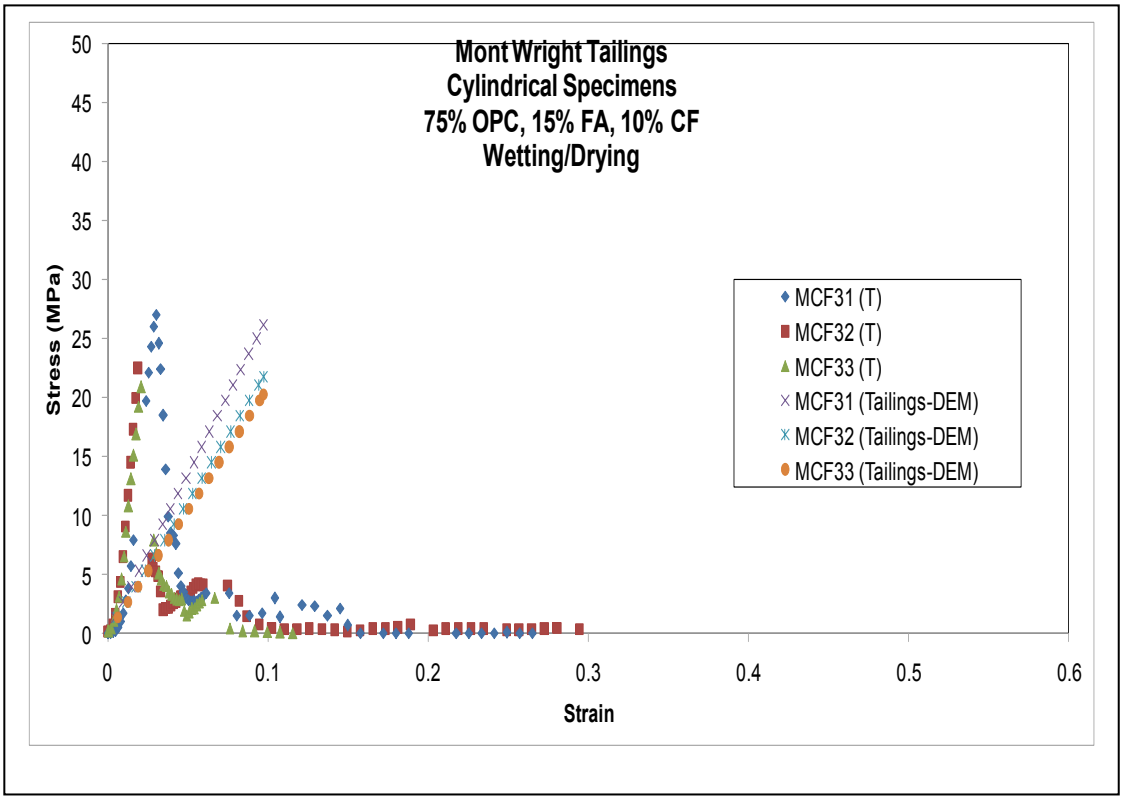


Figure C-9 Computational and experimental UCS values for Mont Wright samples MCF31-MCF33 after wetting/drying

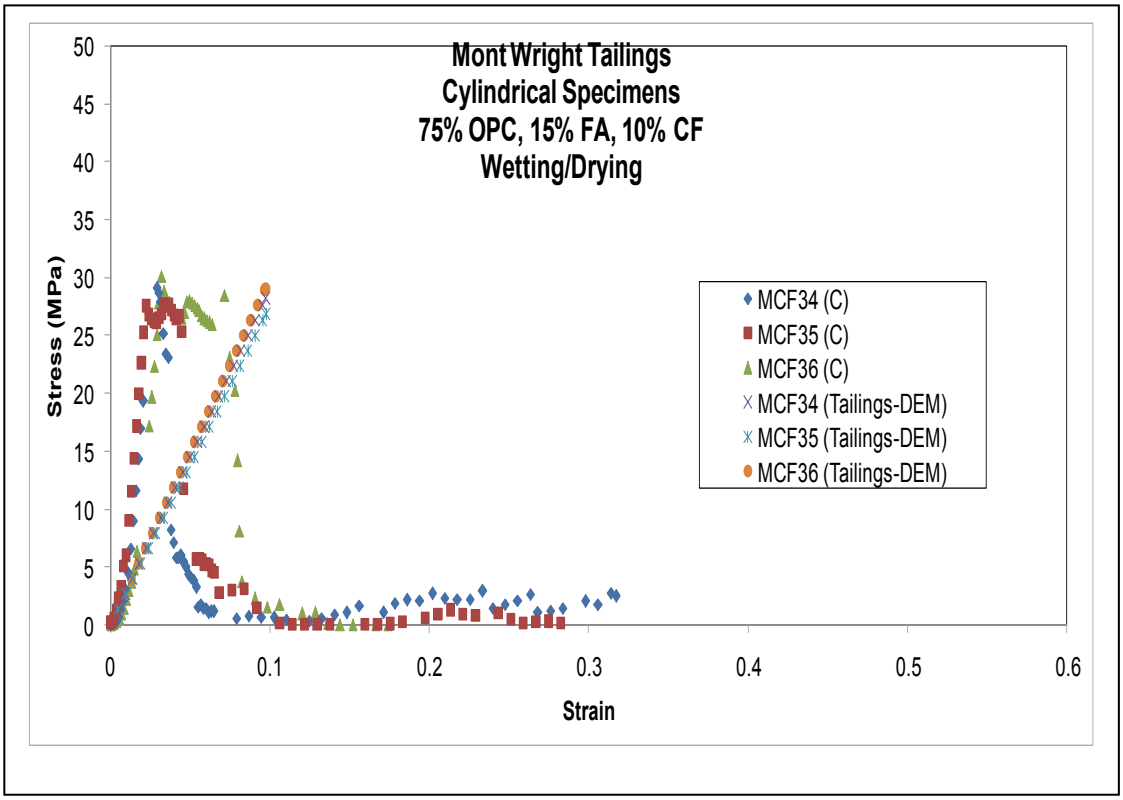


Figure C-10 Computational and experimental UCS values for Mont Wright samples MCF34-MCF36 after wetting/drying

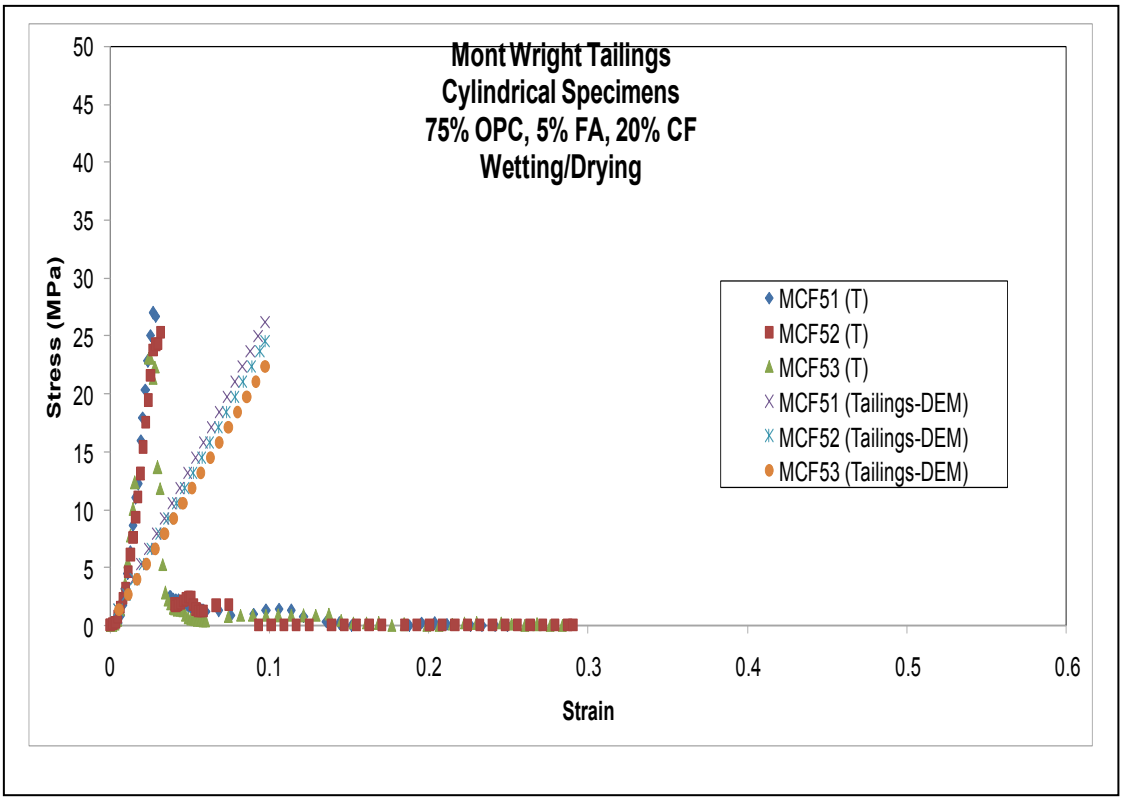


Figure C-11 Computational and experimental UCS values for Mont Wright samples MCF51-MCF53 after wetting/drying

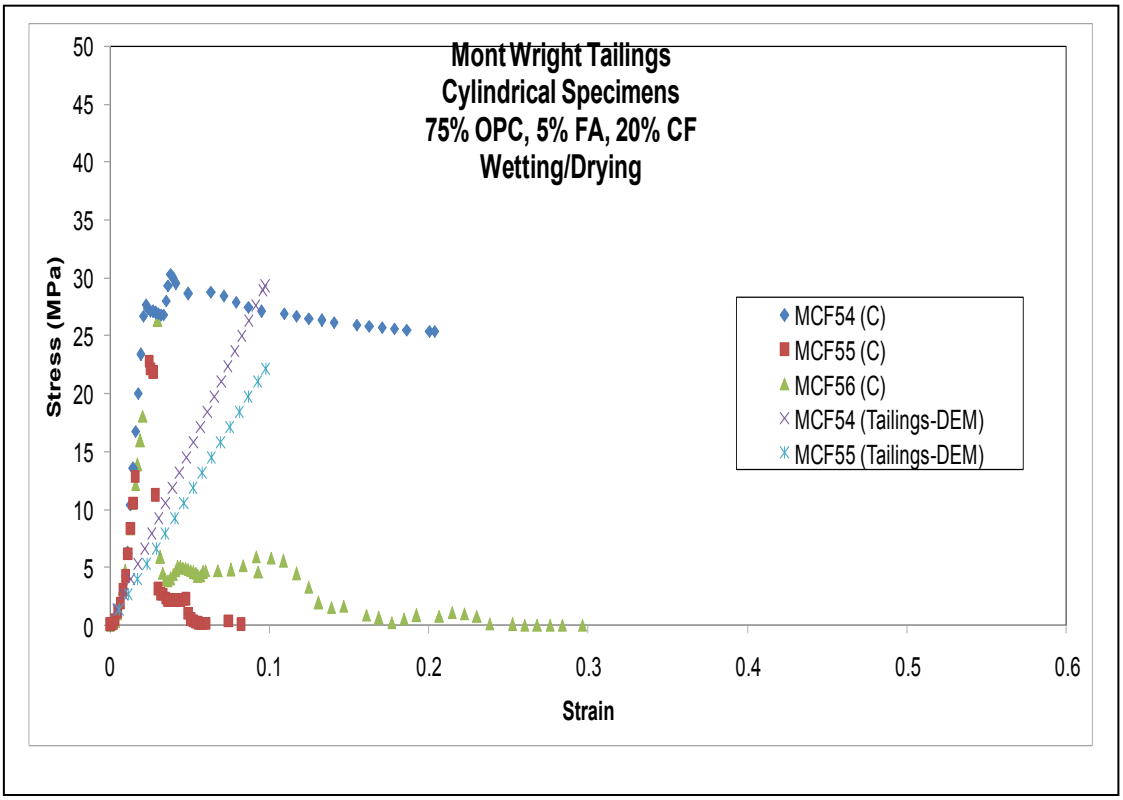


Figure C-12 Computational and experimental UCS values for Mont Wright samples MCF54-MCF56 after wetting/drying

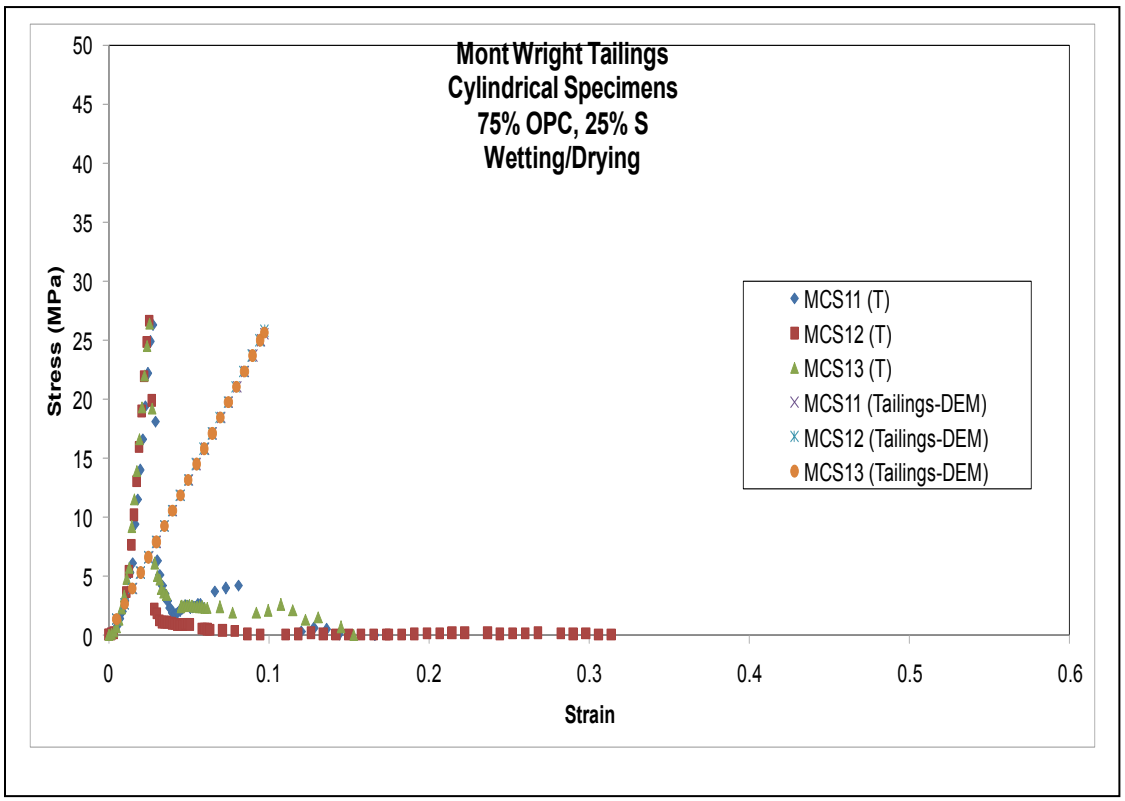


Figure C-13 Computational and experimental UCS values for Mont Wright samples MCS11-MCS13 after wetting/drying

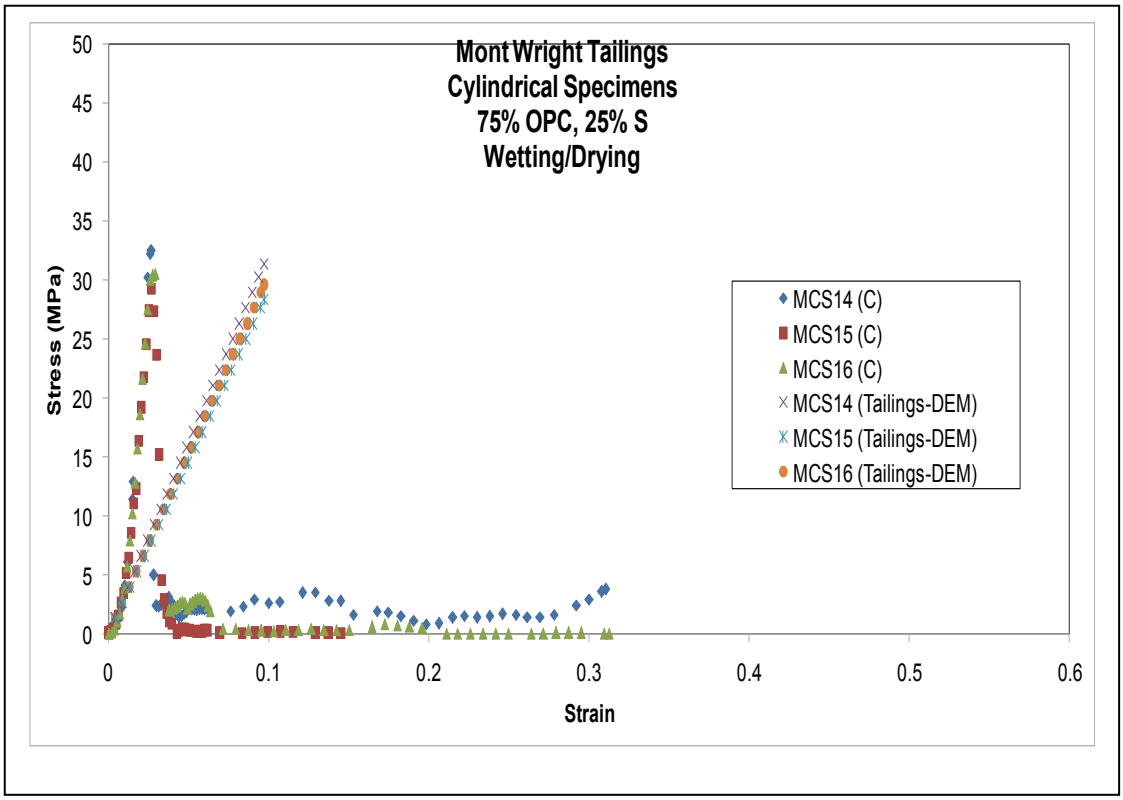


Figure C-14 Computational and experimental UCS values for Mont Wright samples MCS14-MCS16 after wetting/drying

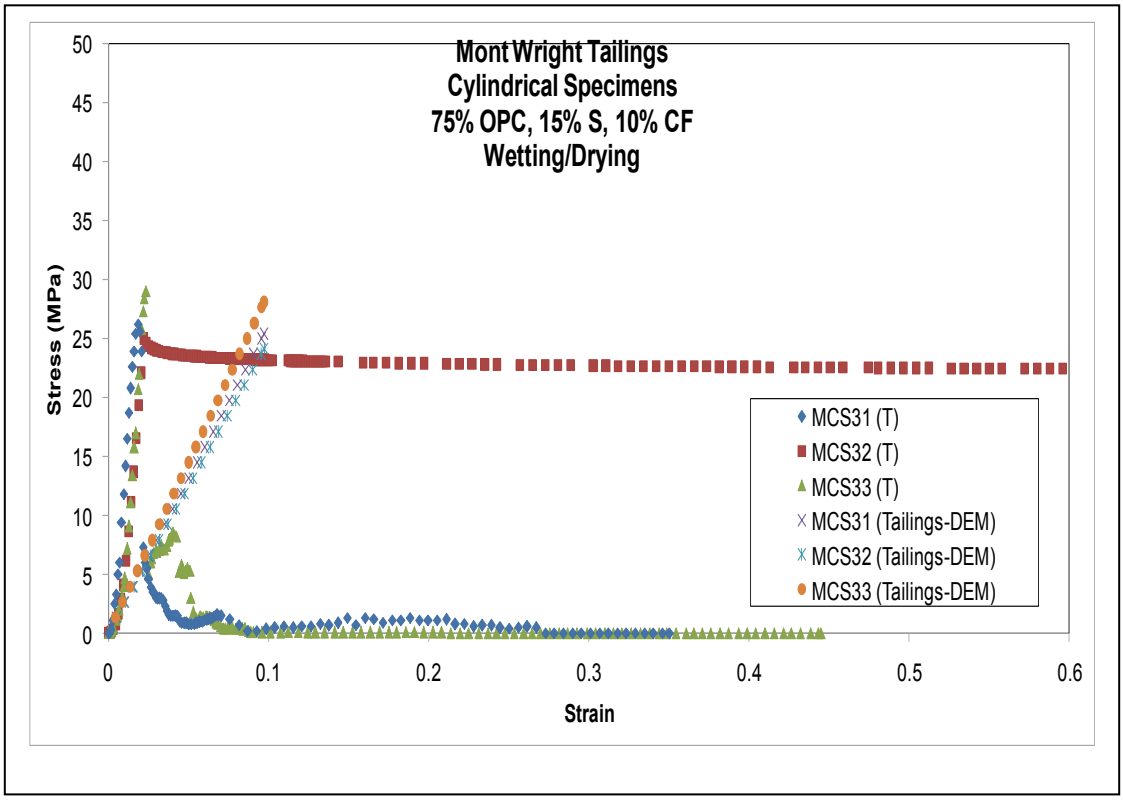


Figure C-15 Computational and experimental UCS values for Mont Wright samples MCS31-MCS33 after wetting/drying

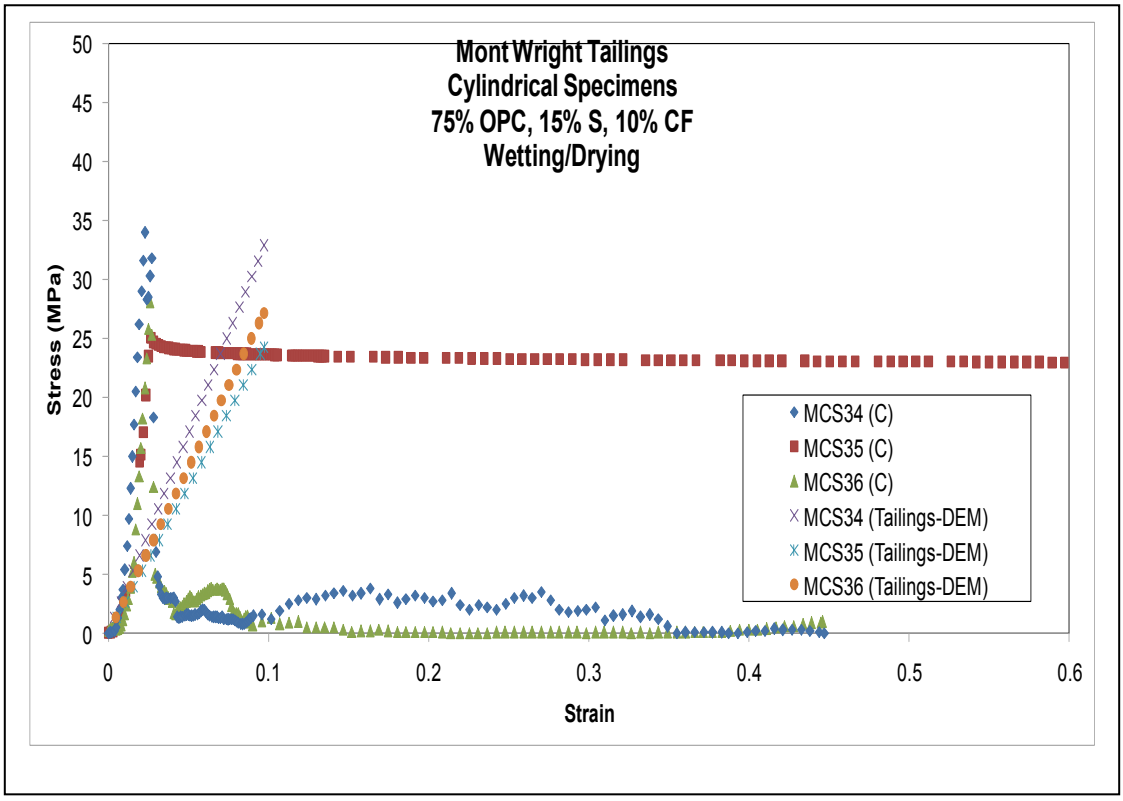


Figure C-16 Computational and experimental UCS values for Mont Wright samples MCS34-MCS36 after wetting/drying

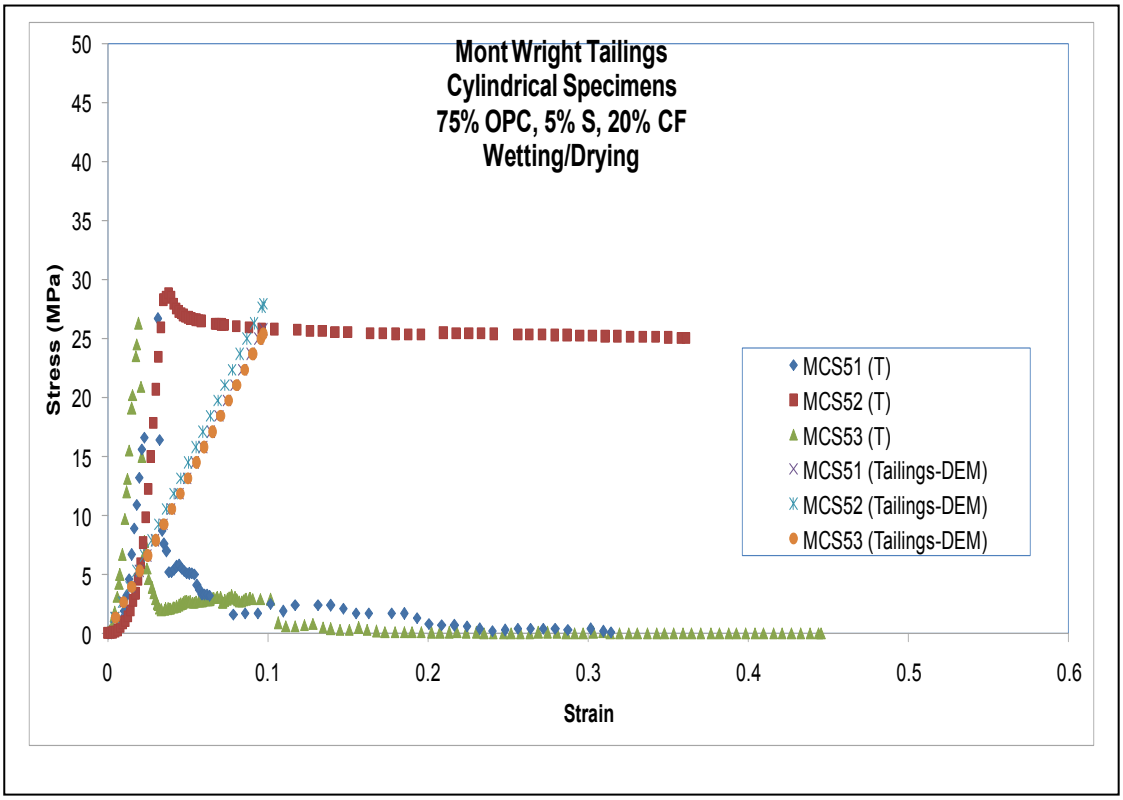


Figure C-17 Computational and experimental UCS values for Mont Wright samples MCS51-MCS53 after wetting/drying

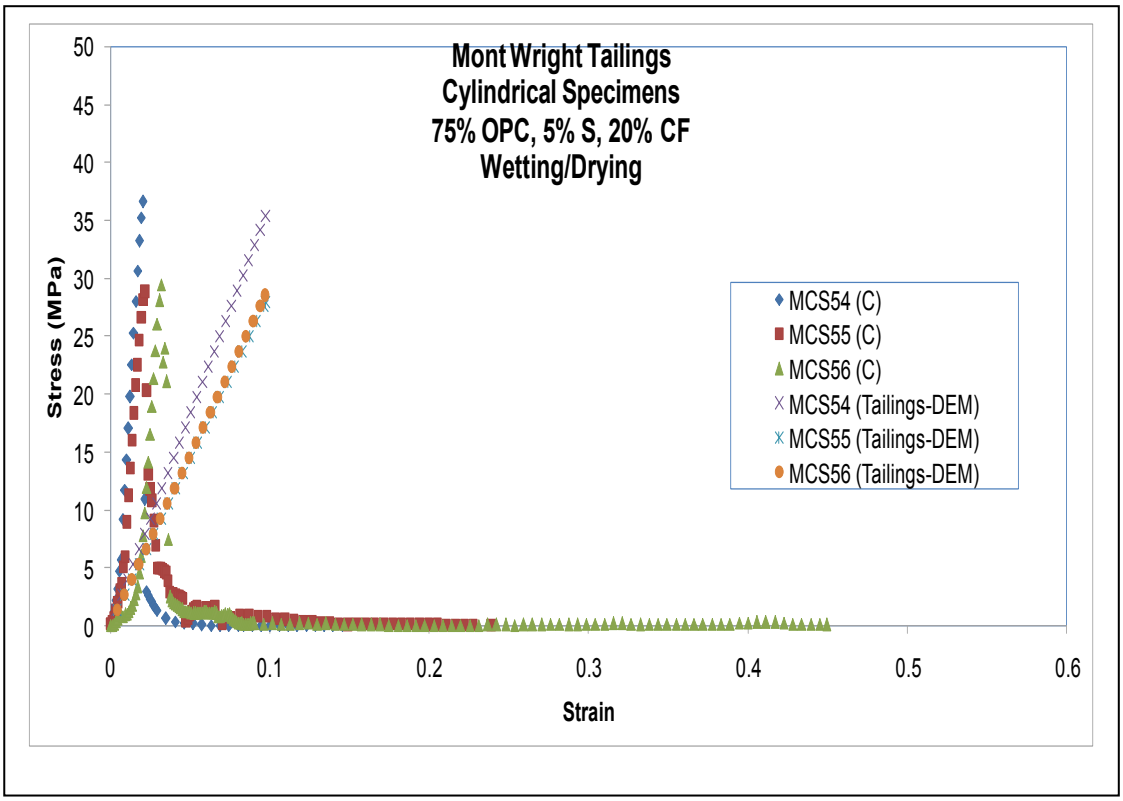


Figure C-18 Computational and experimental UCS values for Mont Wright samples MCS54-MCS56 after wetting/drying

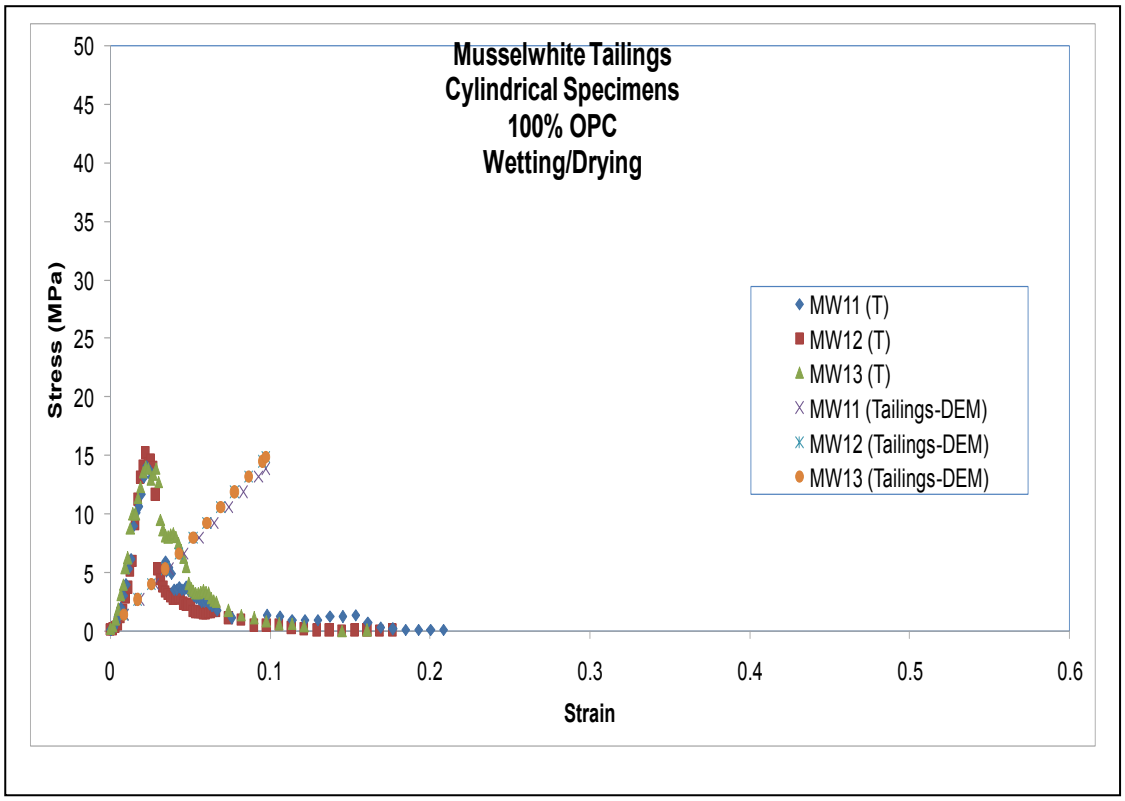


Figure C-19 Computational and experimental UCS values for Musselwhite samples MW11-MW13 after wetting/drying

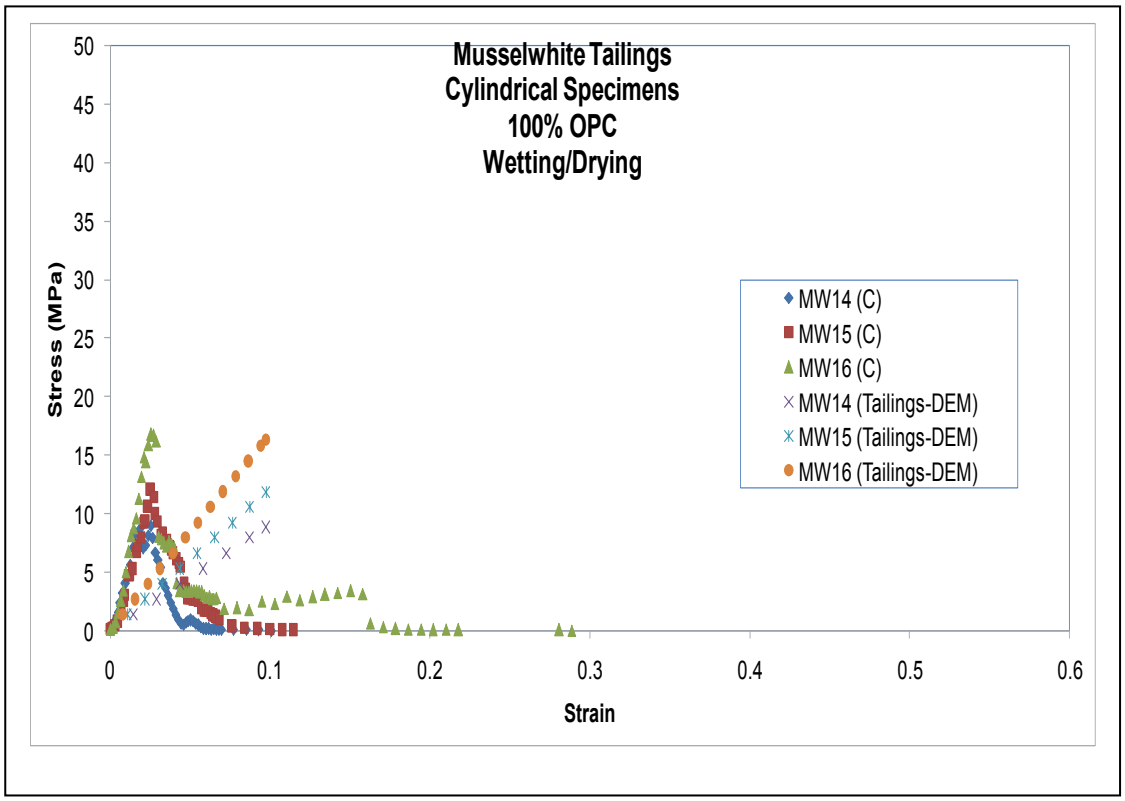


Figure C-20 Computational and experimental UCS values for Musselwhite samples MW14-MW16 after wetting/drying

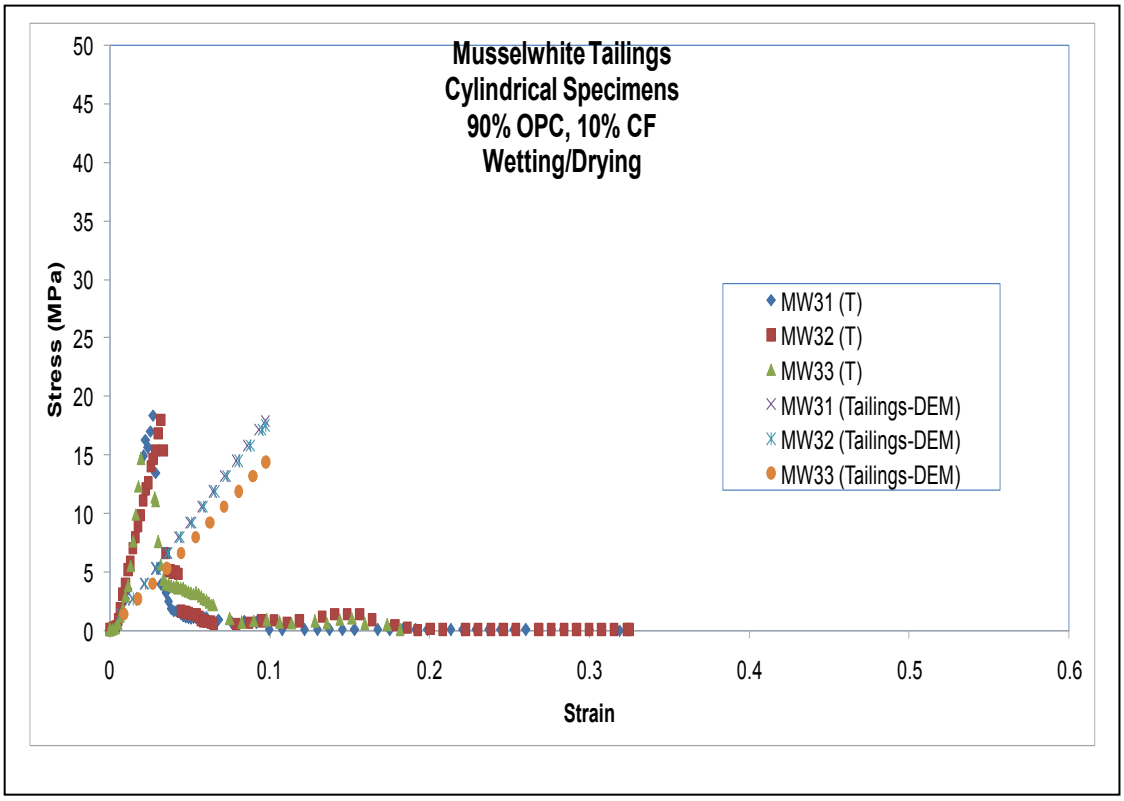


Figure C-21 Computational and experimental UCS values for Musselwhite samples MW31-MW33 after wetting/drying

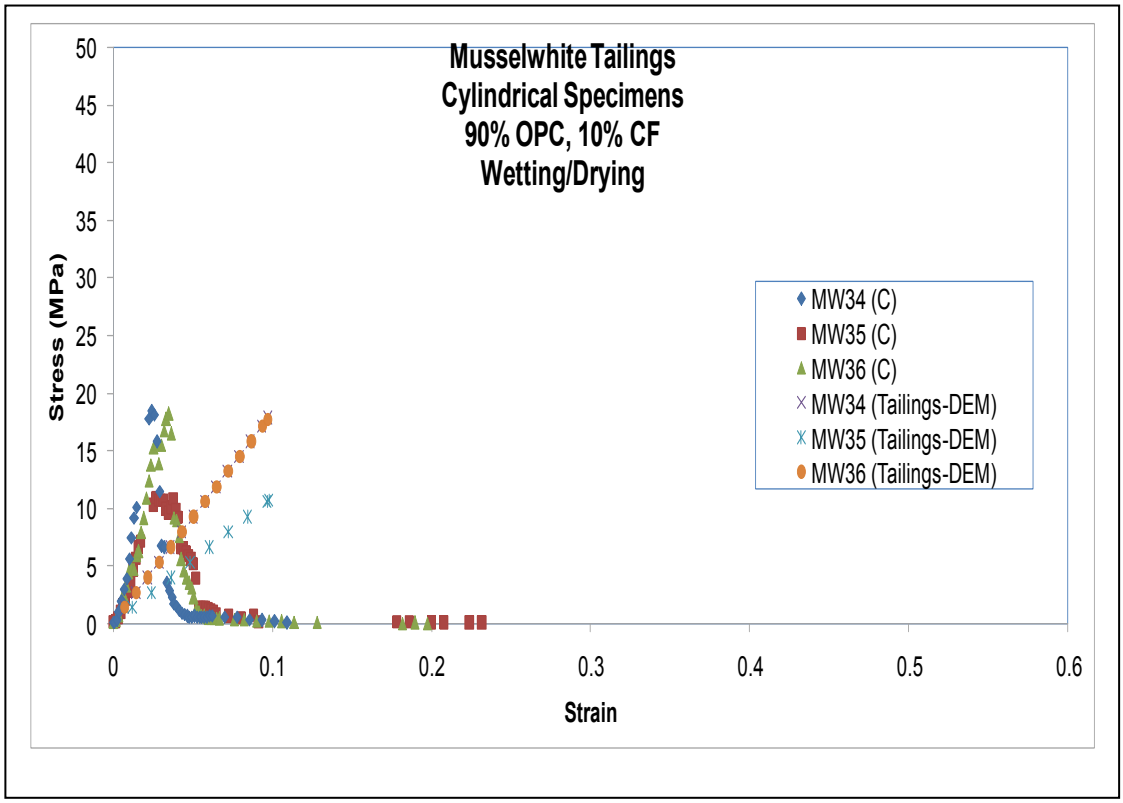


Figure C-22 Computational and experimental UCS values for Musselwhite samples MW34-MW36 after wetting/drying

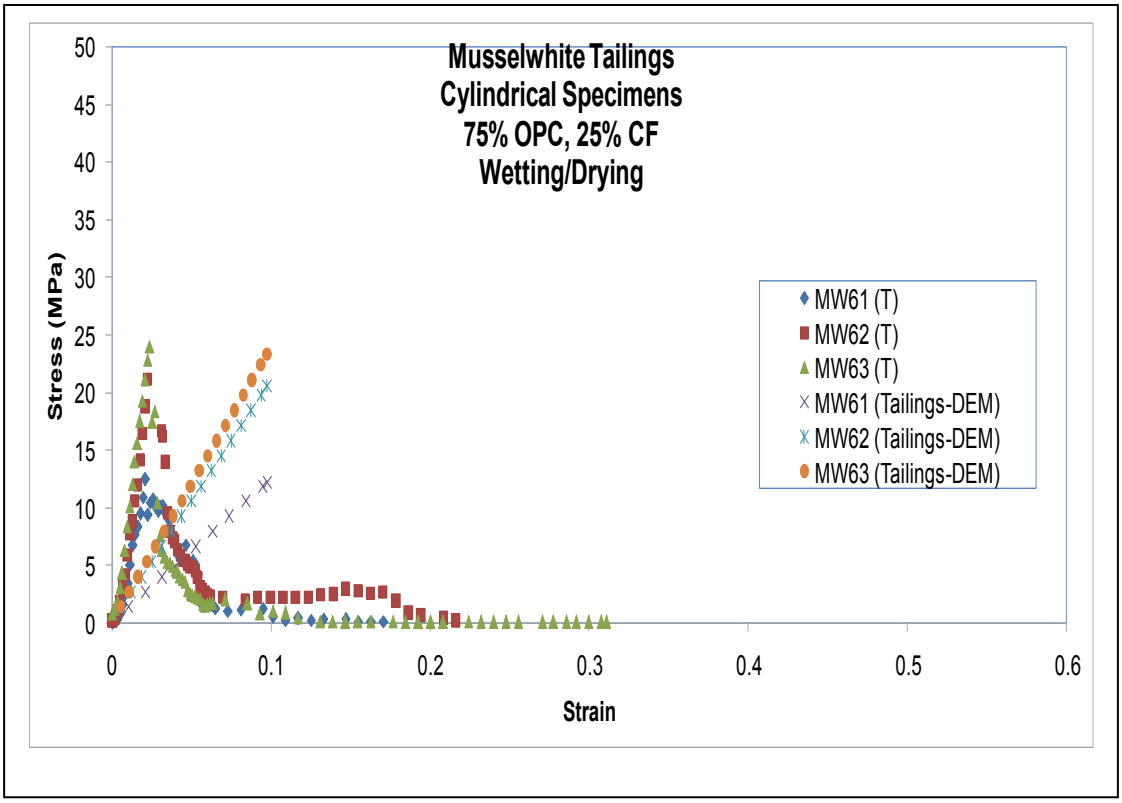


Figure C-23 Computational and experimental UCS values for Musselwhite samples MW61-MW63 after wetting/drying

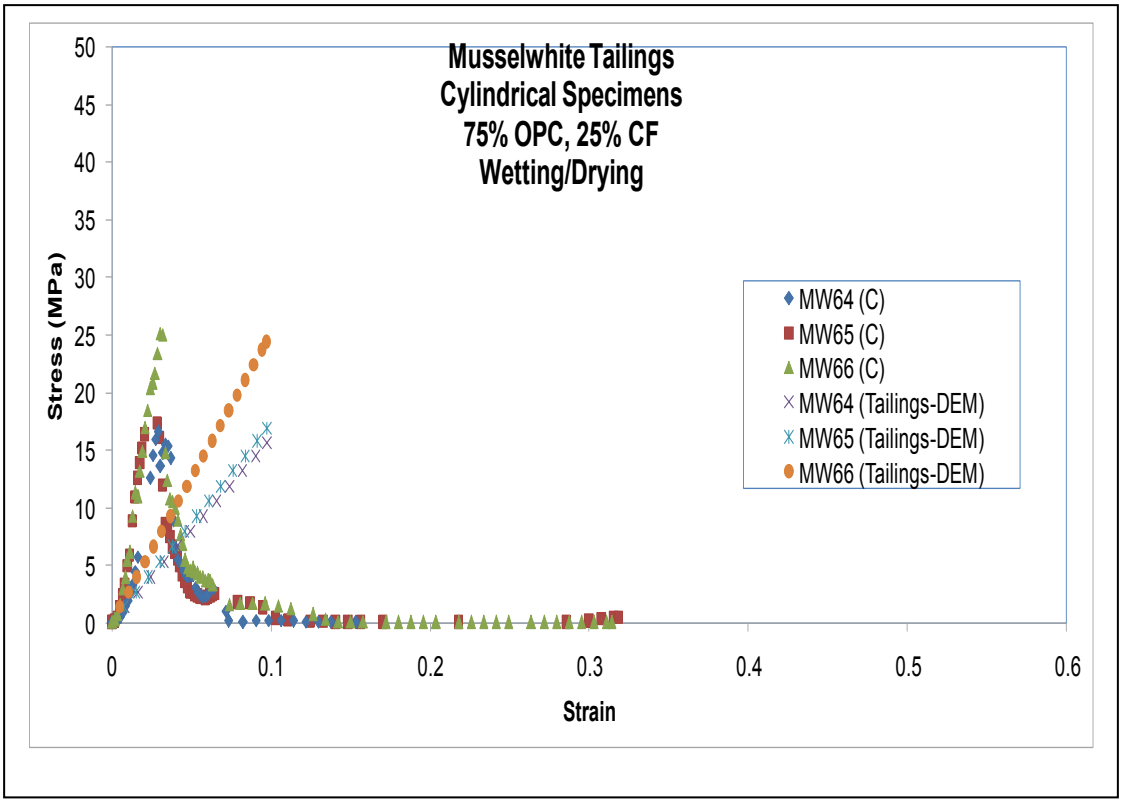


Figure C-24 Computational and experimental UCS values for Musselwhite samples MW64-MW66 after wetting/drying

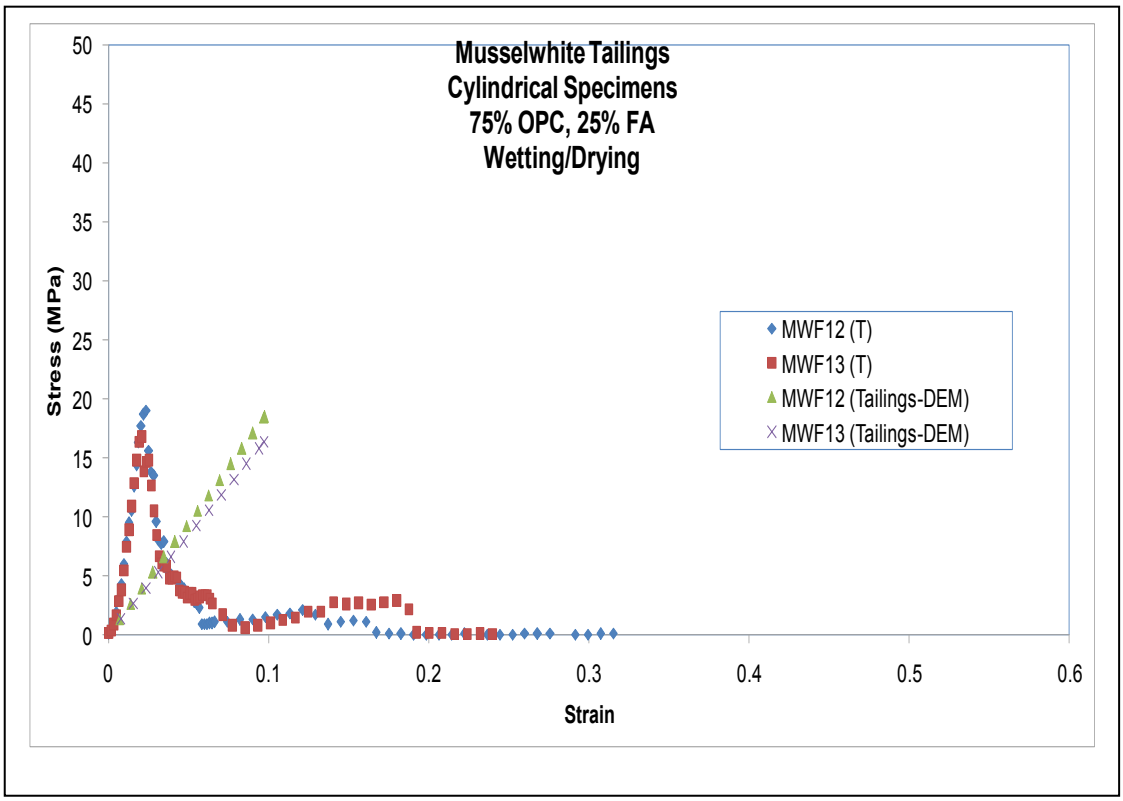


Figure C-25 Computational and experimental UCS values for Musselwhite samples MWF12-MWF13 after wetting/drying

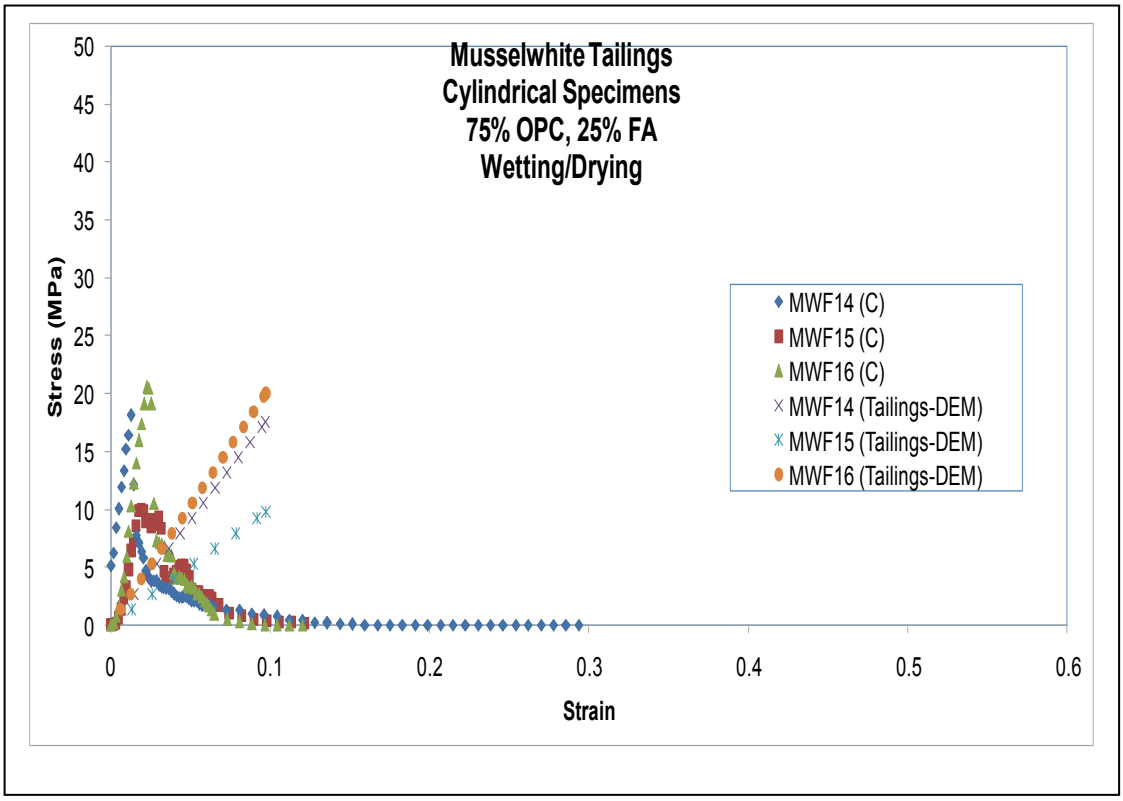


Figure C-26 Computational and experimental UCS values for Musselwhite samples MWF14-MWF16 after wetting/drying

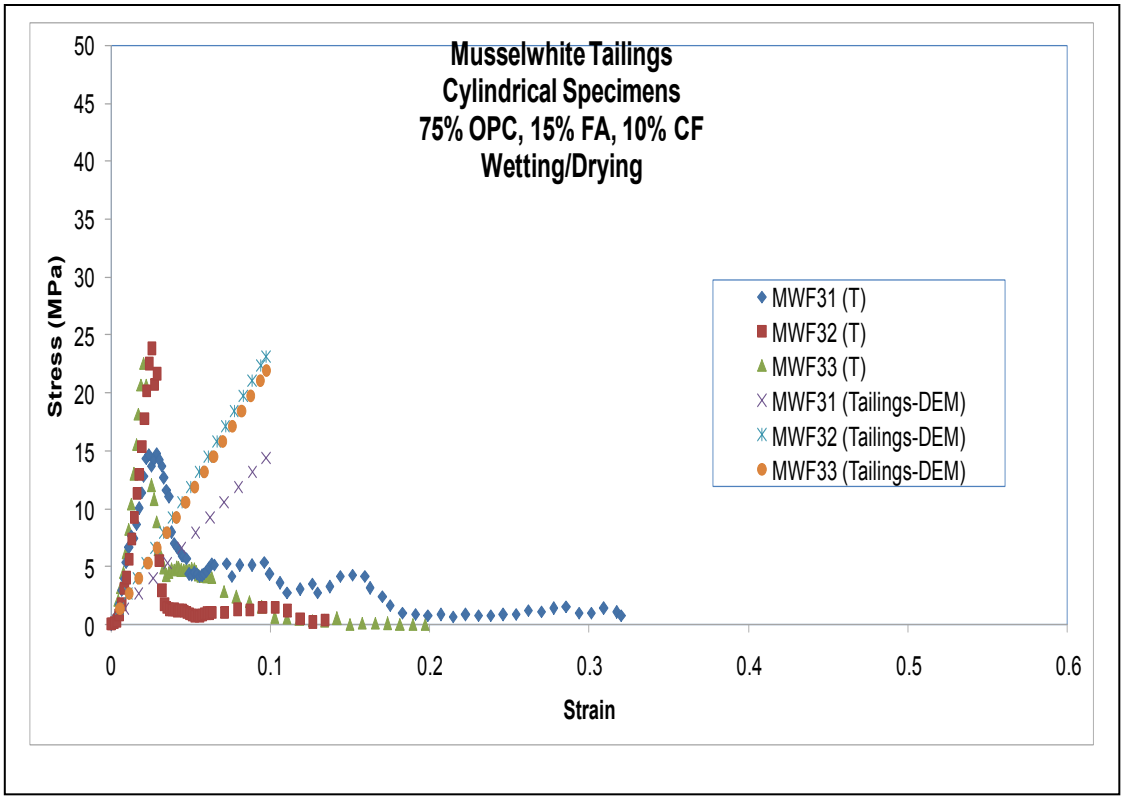


Figure C-27 Computational and experimental UCS values for Musselwhite samples MWF31-MWF33 after wetting/drying

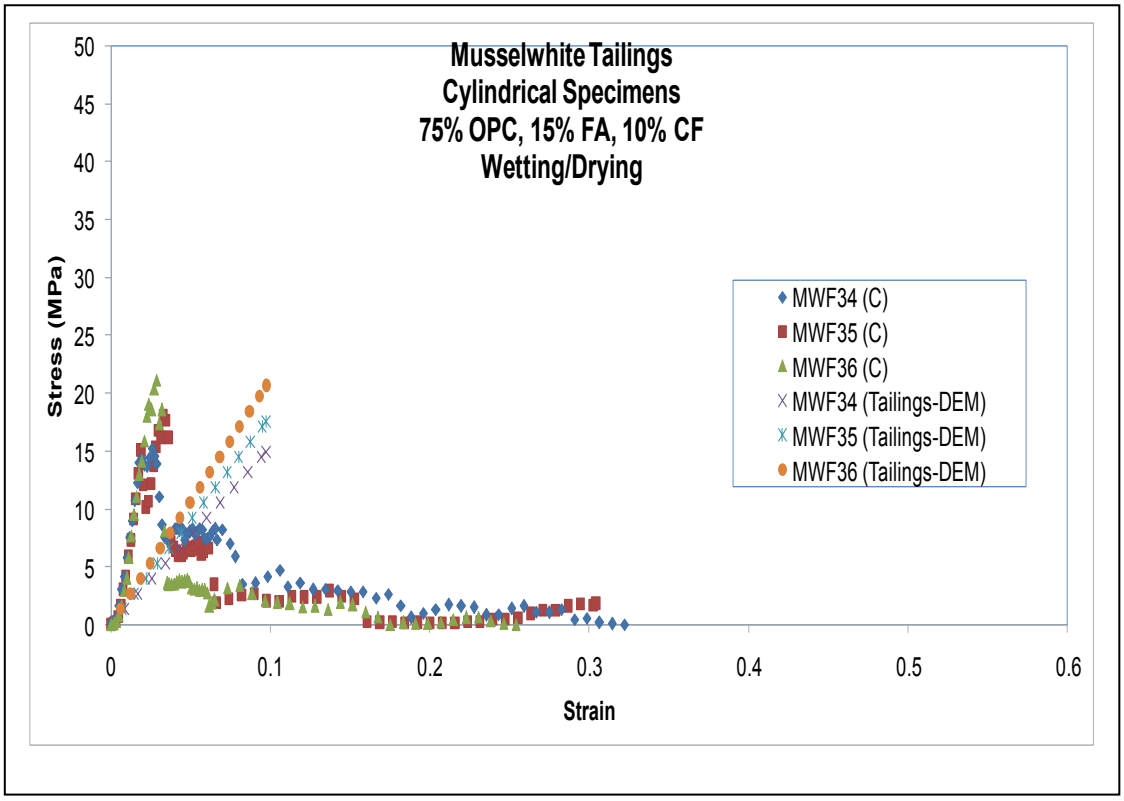


Figure C-28 Computational and experimental UCS values for Musselwhite samples MWF34-MWF36 after wetting/drying

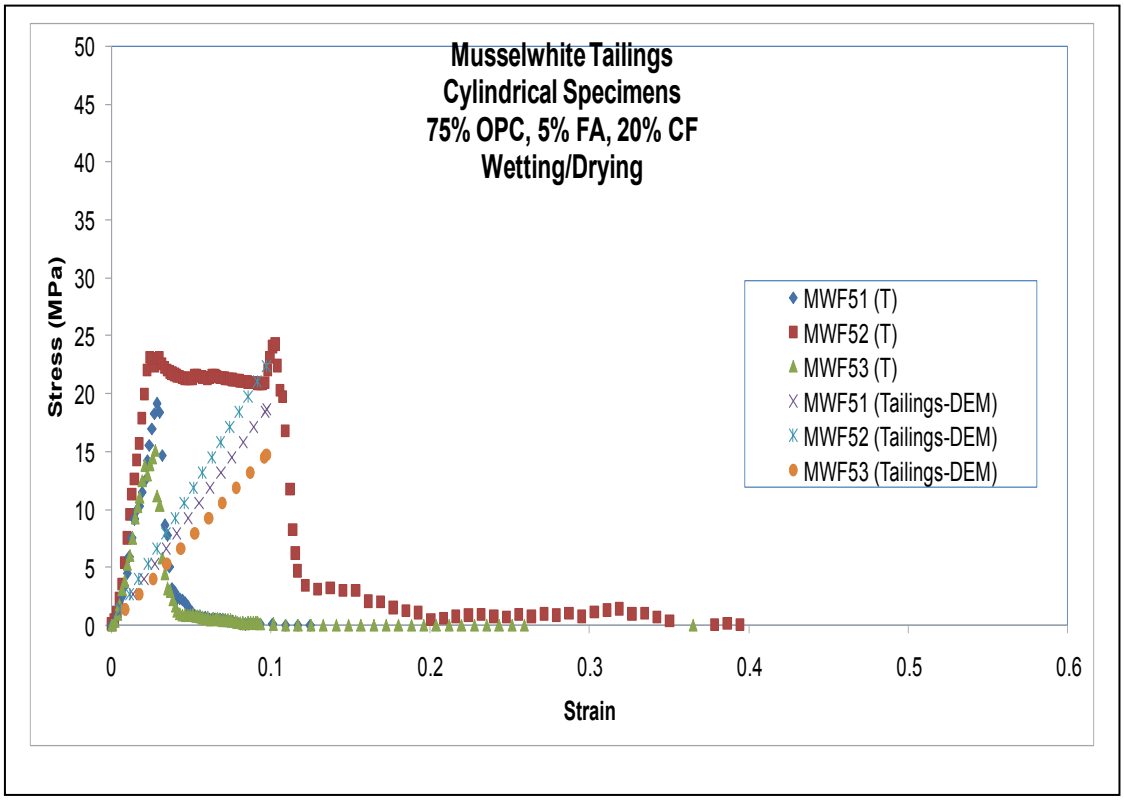


Figure C-29 Computational and experimental UCS values for Musselwhite samples MWF51-MWF53 after wetting/drying

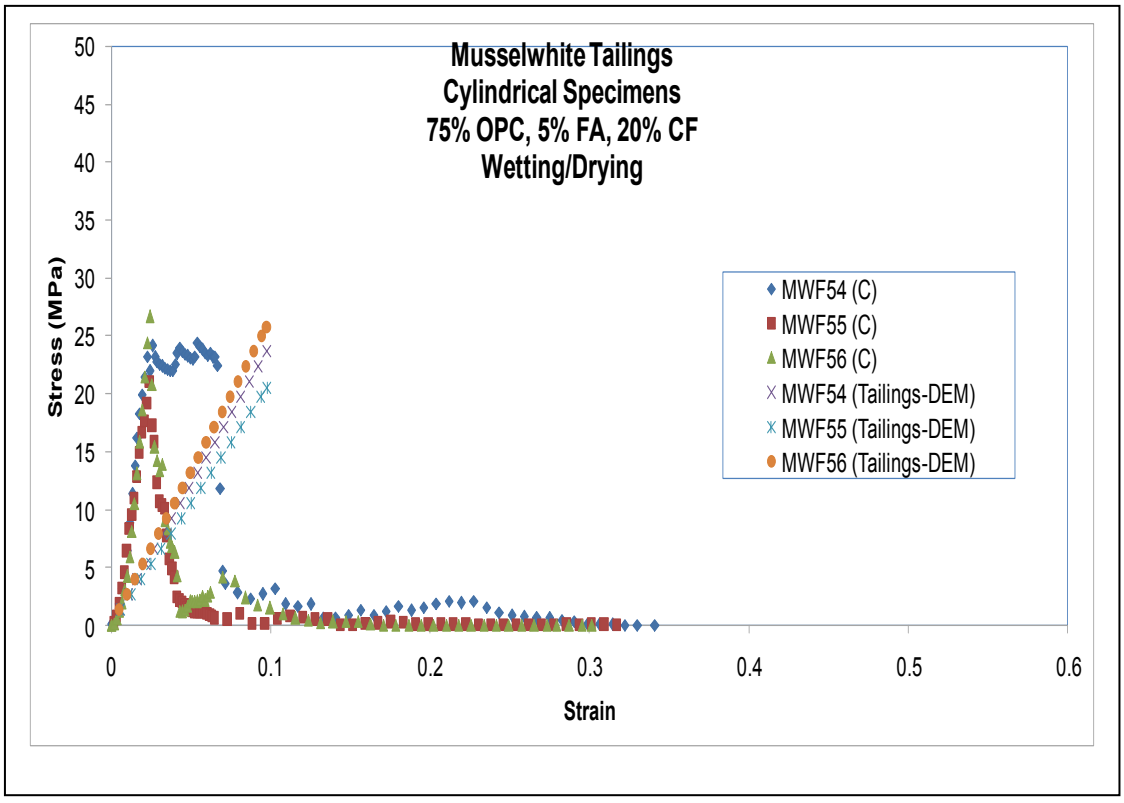


Figure C-30 Computational and experimental UCS values for Musselwhite samples MWF54-MWF56 after wetting/drying

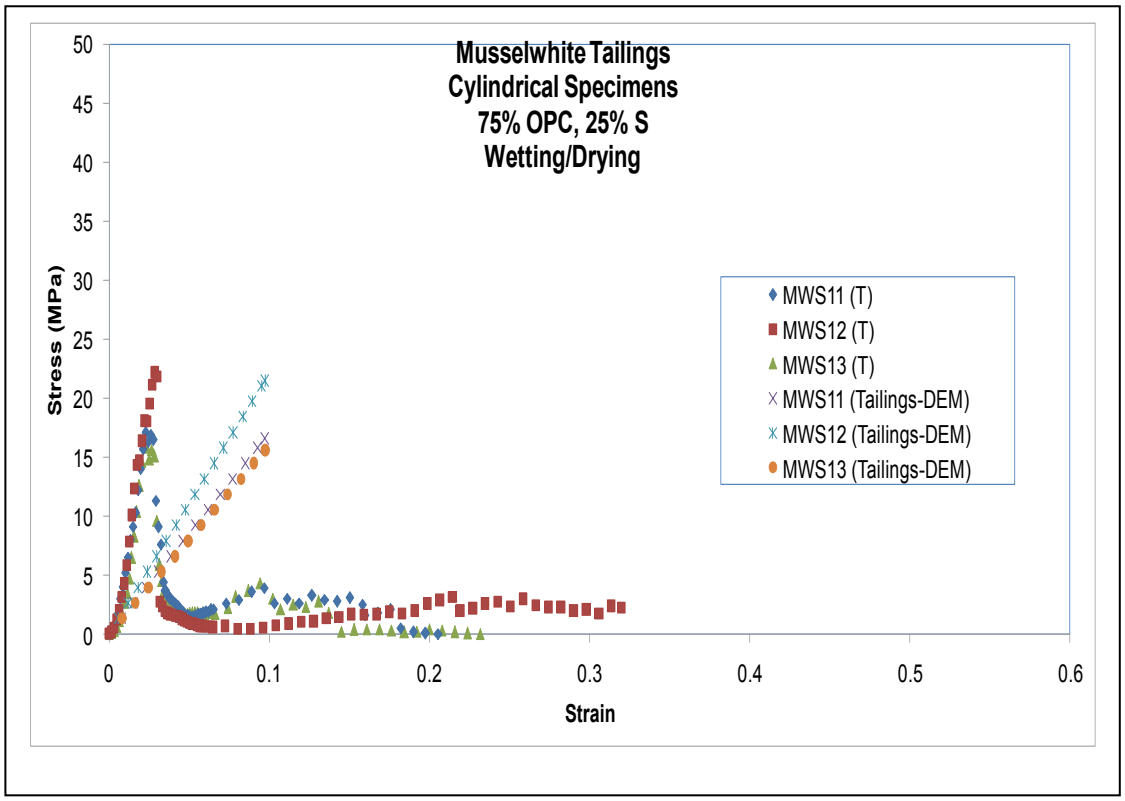


Figure C-31 Computational and experimental UCS values for Musselwhite samples MWS11-MWS13 after wetting/drying

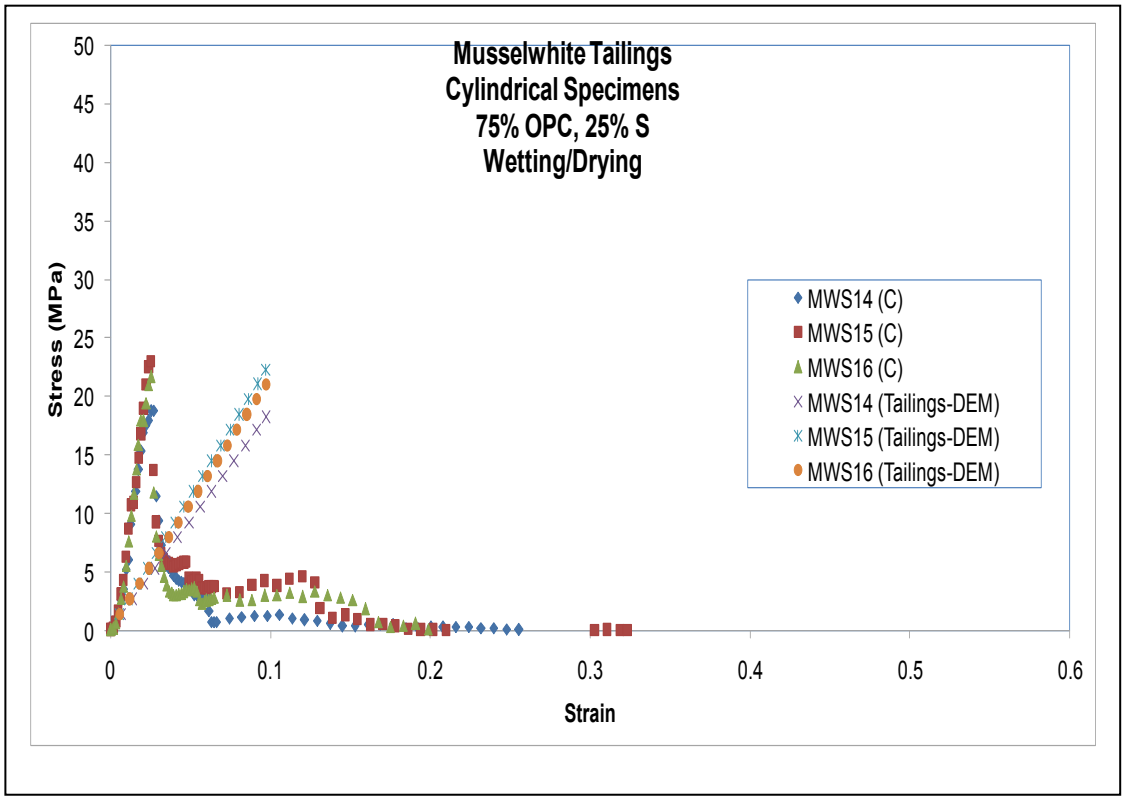


Figure C-32 Computational and experimental UCS values for Musselwhite samples MWS14-MWS16 after wetting/drying

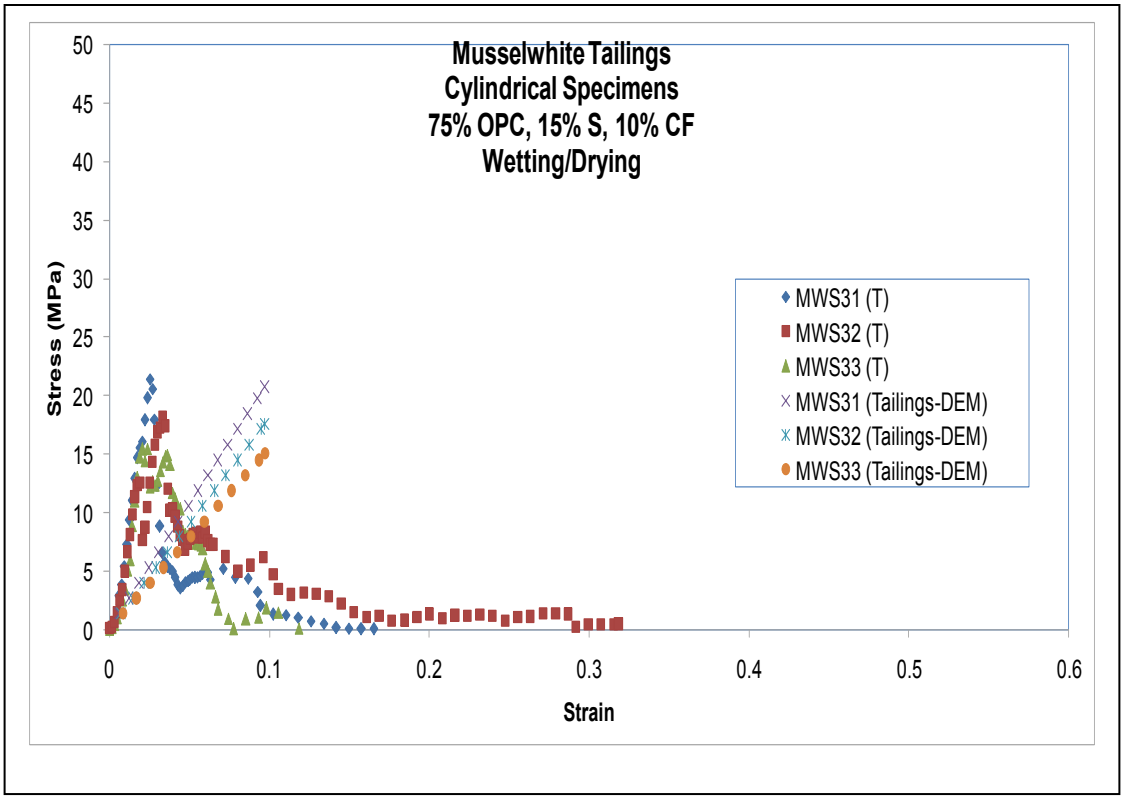


Figure C-33 Computational and experimental UCS values for Musselwhite samples MWS31-MWS33 after wetting/drying

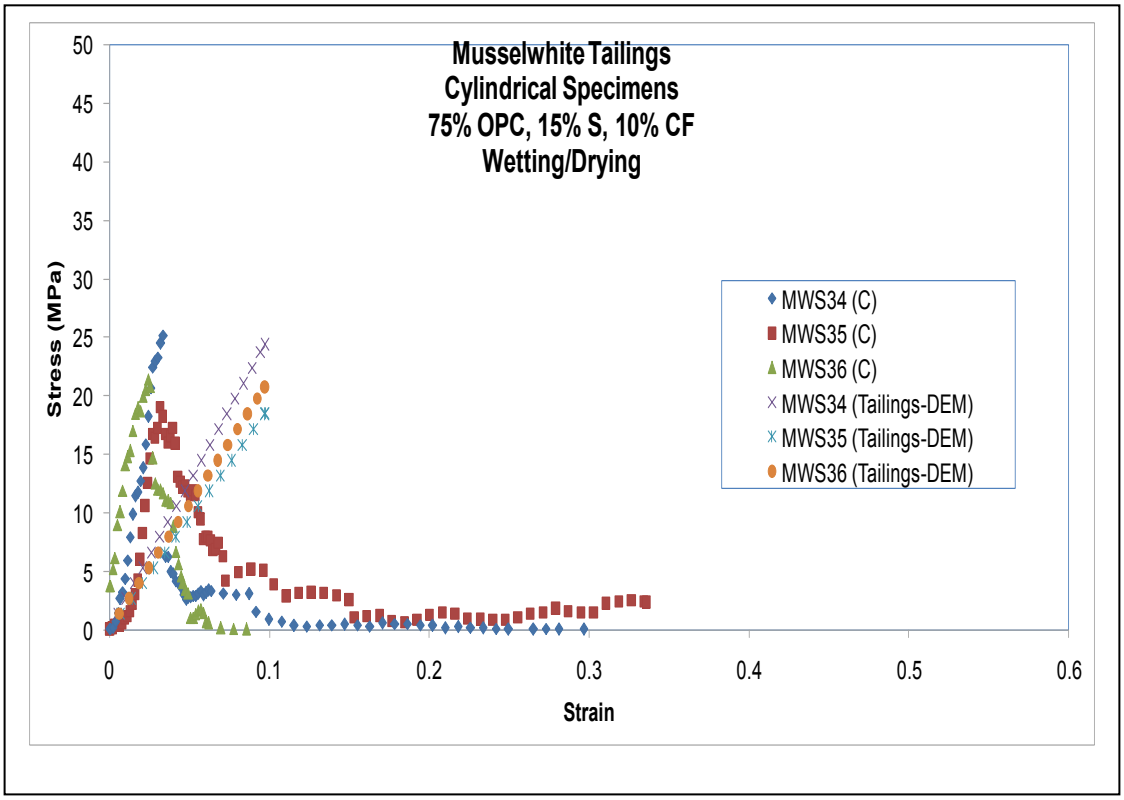


Figure C-34 Computational and experimental UCS values for Musselwhite samples MWS34-MWS36 after wetting/drying

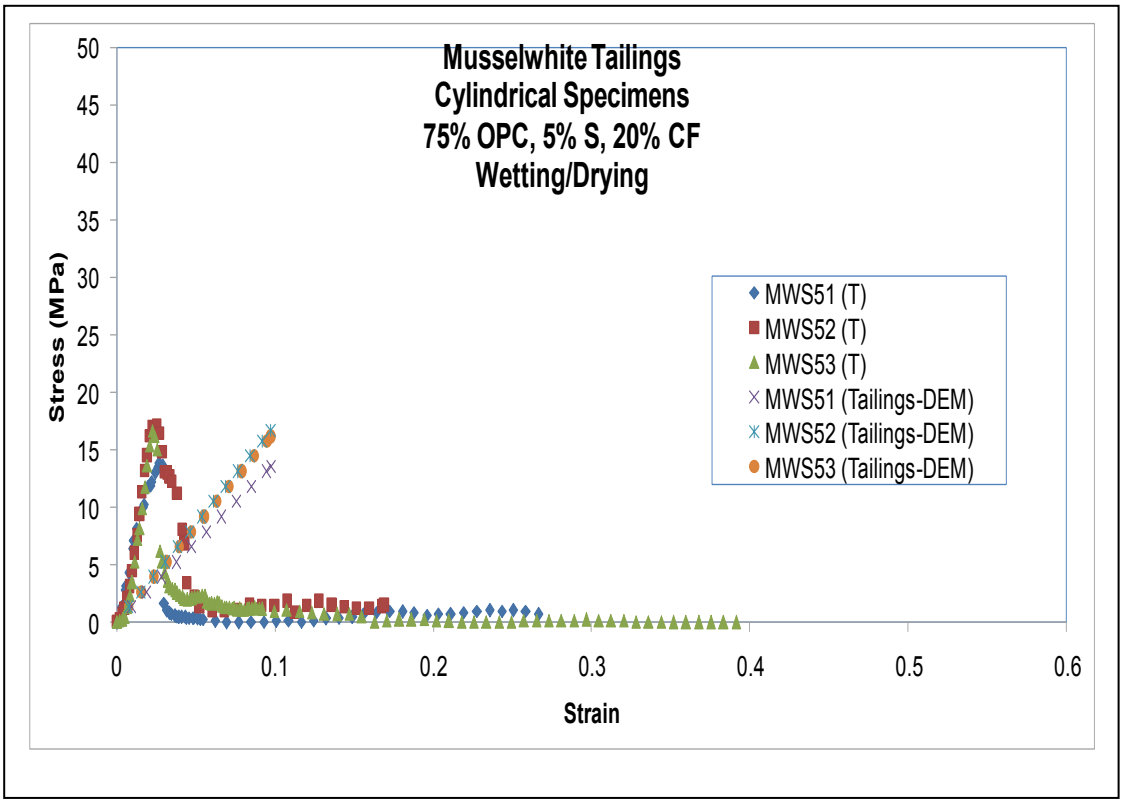


Figure C-35 Computational and experimental UCS values for Musselwhite samples MWS51-MWS53 after wetting/drying

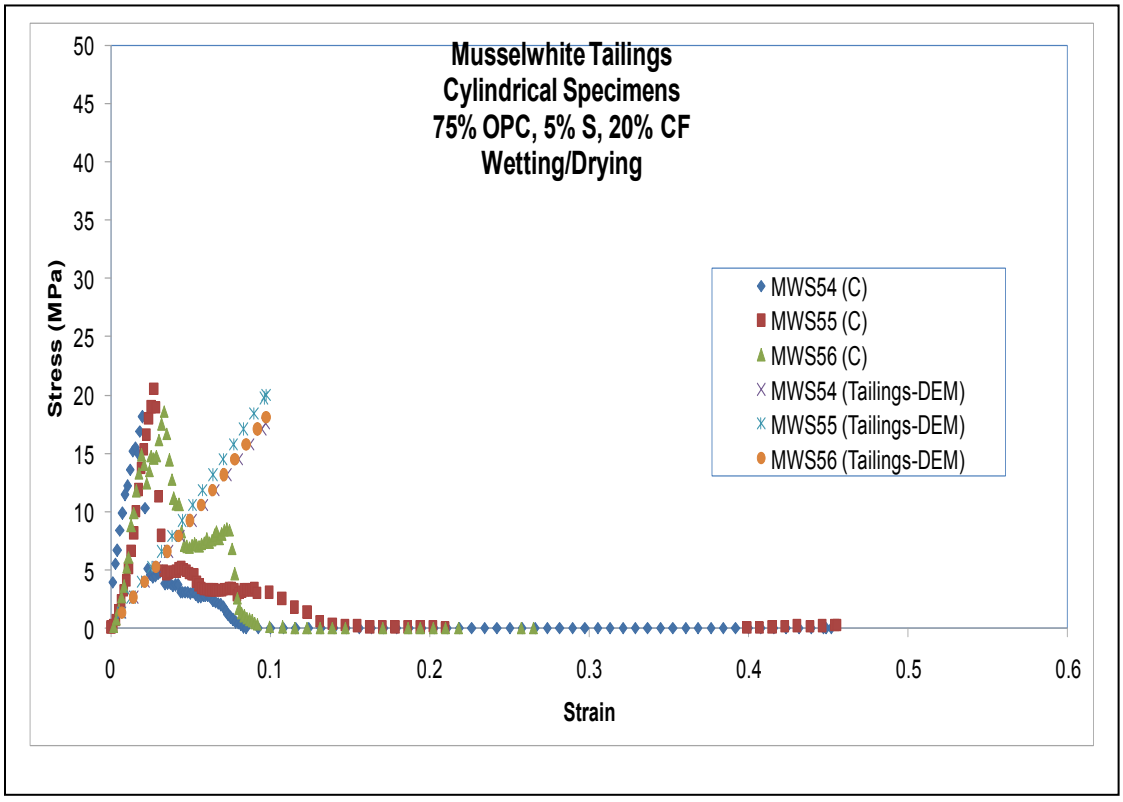


Figure C-36 Computational and experimental UCS values for Musselwhite samples MWS54-MWS56 after wetting/drying

Appendix D:

Code for Tailings-DEM (Series 1)

```
# include <iostream>
# include <cmath>
# include <fstream>
#include <numeric>

using namespace std;


 struct geometricalshapes  {  // subroutine for assigning values to the particle geometrical shapes
 float side1, side2, side3, side4, side5;
 float angle1, angle2, angle3, angle4, angle5;
 float height, base, centroidx, centroidy, momentofinertiax, momentofinertiay, area;

};

float ddamping_coefficient, dk;
float dstiffness_coefficient;
float dtime_increment;
float dmass;
float ii;  // mass moment of inertia
float dresult;


int isocounterbox1[100], isocounterbox2[100], isocounterbox3[100], isocounterbox4[100], isocounterbox5[100], isocounterbox6[100], isocounterbox7[100], isocounterbox8[100], isocounterbox9[100], isocounterbox10[100], isocounterbox11[100], isocounterbox12[100], isocounterbox13[100], isocounterbox14[100], isocounterbox15[100], isocounterbox16[100];
float p, p2;
int bb;


float ydisplat [100][100], xdisplat [100][100], delus [100][100], delun [100][100], fn [100][100], fs [100][000], dn [100][100], ds [100][100];
float yforce [100][100], xforce [100][100], fxsum [100][100], x [100], resultfx [100], fysum [100][100];
float y [100], resultfy [100], msum [100][100], m [100], resultm [100], udoty [100][100], udotysum [100], resultudoty [100];
float udotx [100][100], udotxsum [100], resultudotx [100], thetadot [100][100], thetadotsum [100], resultthetadot [100];
float deluy [100][100], deluysum [100], resultdeluy [100], delux [100][100], deluxsum [100], resultdelux [100], deltheta [100][100];
float delthetasum [100], resultdeltheta [100], uy [100][100], uysum [100], resultuy [100], ux [100][100], uxsum [100];
float resultux [100], theta [100][100], thetasum [100], resulttheta [100];


```

```
38.float rightisosceles [150][4][10];
39.int isoscelescounter = 0;
40.float ydisplacement2 = 0.0, yvelocity2= 0;
41.float xdisplacement2 = 0.0;
42.float alpha [100], alpha1 = 0.785398;
43.int v = 1;
44.float n, n1;
45.int num3 =  1 + (39-4)/5;    // number of isosceles triangles per row
46.int hh, oo, pp;
47.
48.
49.
50.int main ()
51.
52.{
53.
54.geometricalshapes righttriangle; //the particle
55.righttriangle.side1 = 4.243;
56.righttriangle.side2 = 3;
57.righttriangle.side3 = 3;
58.righttriangle.angle1 = 45;
59.righttriangle.angle2 = 45;
60.righttriangle.angle3 = 90;
61.righttriangle.height = 2.12;
62.righttriangle.base = 4.243;
63.righttriangle.centroidx = 0;
64.righttriangle.centroidy = 0.707;
65.righttriangle.momentofinertiax = 1.123;
66.righttriangle.momentofinertiay = 3.374;
67.righttriangle.area = 4.498;
68.
69. ofstream a_file ("data-trial.txt");
70.
71.
72.
73. cout<< "Please enter the stiffness coefficient in N/m" << endl;
74. cin>> dstiffness_coefficient;
75.
76.
77. cout<< "Please enter the mass of the particle in kg" << endl;
78. cin>> dmass;
79.
80.ii =  ((righttriangle.base/2) * (righttriangle.base/2)* (dmass)/3) ;
81.
82.
83. // time increment verification
84.
85. begin: //goto label
86. cout<< "Please enter the time increment" << endl;
87. cin>> dtime_increment;
88.
89. dresult = (2 * sqrt (dmass/dstiffness_coefficient));
```

```
90. if (dtime_increment >=  dresult ) // time increment condition
91.
92.{
93.
94. cout<< "Time increment does not satisfy condition"<< endl;
95. goto begin;
96.
97.}
98.
99. else
100.
101.      {
102.        cout<< "Time increment satisfies condition"<< endl;
103.
104.     }
105.
106.
107.     dk = 2 * (sqrt (dmass * dstiffness_coefficient)); // subroutine for
    calculating the damping coefficient
108.     ddamping_coefficient = dk/dtime_increment;
109.
110.
111.      system ("pause");
112.
113.
114.
115.     for (hh = 0; hh <150; hh = hh+1) {
116.         for (oo = 0; oo <4; oo = oo+1) { for (pp = 0; pp <10; pp = pp +
    1)
117.         {
118.             rightisosceles [hh][oo][pp] = 0.0;
119.
120.
121.         }
122.         }
123.     }
124.
125.
126.
127.      for (n = 3.0 ; n <= 70; n = n + 5) {       // positioning particles

128.         for (n1 = 4.0 ; n1 <= 39; n1 = n1 + 5) {
129.
130.         rightisosceles [v][0][0] = n1 ;            // right isosceles x
    coordinates
131.         rightisosceles [v][1][0] = rightisosceles [v][0][0] + (righttria
    ngle.base /2);
132.         rightisosceles [v][2][0] = rightisosceles [v][0][0] -
     (righttriangle.base /2);
133.         rightisosceles [v][3][0] = rightisosceles [v][0][0];
134.
```

```
135.         rightisosceles [v][0][1] = n ;    // right isosceles y coordinat
   es
136.         rightisosceles [v][1][1] = rightisosceles [v][0][1] + (righttria
   ngle.height /3);
137.         rightisosceles [v][2][1] = rightisosceles [v][1][1];
138.         rightisosceles [v][3][1] = rightisosceles [v][2][1] -
    (righttriangle.height);
139.
140.
141.
142.         v = v + 1;
143.         isoscelescounter = isoscelescounter + 1;
144.
145.
146.
147.     }
148.     }
149.
150.     p2 = 10/num3;
151.
152.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
153.     ydisplacement2 =  ydisplacement2 + (yvelocity2 * dtime_increment);

154.
155.
156.
157.
158.
159.     for (p = 100; p < 60000 ; p = p+100)  // load cycle in Newtons
160.     {
161.     p2 = p/num3;
162.
163.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
164.
165.
166.       a_file <<  p << endl;
167.
168.
169.     int isobox1 = 0, isobox2 = 0, isobox3 = 0, isobox4 = 0, isobox5 = 0,
    isobox6 = 0, isobox7 = 0, isobox8 = 0, isobox9 = 0, isobox10 = 0, isobox1
   1 = 0, isobox12 = 0, isobox13 = 0, isobox14 = 0, isobox15 = 0, isobox16 =
   0;
170.
171.
172.     for (bb = 0; bb <100; bb = bb+1) {
173.         isocounterbox1[bb] = 0, isocounterbox2[bb] = 0, isocounterbox3[b
   b] = 0, isocounterbox4[bb]= 0;
174.         isocounterbox5[bb] = 0, isocounterbox6[bb] = 0, isocounterbox7[b
   b] = 0, isocounterbox8[bb] = 0;
```

```
175.          isocounterbox9[bb] = 0, isocounterbox10[bb] = 0, isocounterbox11
   [bb] = 0, isocounterbox12[bb] = 0;
176.          isocounterbox13[bb] = 0, isocounterbox14[bb] = 0, isocounterbox1
   5[bb] = 0, isocounterbox16[bb] = 0;
177.
178.          }
179.
180.
181.
182.     for (int dd = 1; dd <= isoscelescounter ; dd = dd + 1)  // locating
   the isosceles triangles within the 16 screen boxes
183.     {
184.
185.      if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][0]
   <= 11) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][1]<=
   18.5))
186.     {isocounterbox1[isobox1] = dd; isobox1 = isobox1 + 1;}
187.
188.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
189.     {isocounterbox2[isobox2] = dd; isobox2 = isobox2 + 1;}
190.
191.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
192.     {isocounterbox3[isobox3] = dd; isobox3 = isobox3 + 1;}
193.
194.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
195.     {isocounterbox4[isobox4] = dd; isobox4 = isobox4 + 1;}
196.
197.     else if( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][
   0] <= 11) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 37))
198.     {isocounterbox5[isobox5] = dd; isobox5 = isobox5 + 1;}
199.
200.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
201.     {isocounterbox6[isobox6] = dd; isobox6 = isobox6 + 1;}
202.
203.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
204.     {isocounterbox7[isobox7] = dd; isobox7 = isobox7 + 1;}
205.
206.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
207.     {isocounterbox8[isobox8] = dd; isobox8 = isobox8 + 1;}
```

```
208.
209.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 55.5))
210.     {isocounterbox9[isobox9] = dd; isobox9 = isobox9 + 1;}
211.
212.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
213.     {isocounterbox10[isobox10] = dd; isobox10 = isobox10 + 1;}
214.
215.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
216.     {isocounterbox11[isobox11] = dd; isobox11 = isobox11 + 1;}
217.
218.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
219.     {isocounterbox12[isobox12] = dd; isobox12 = isobox12 + 1;}
220.
221.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0
   ][1]<= 74))
222.     {isocounterbox13[isobox13] = dd; isobox13 = isobox13 + 1;}
223.
224.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
225.     {isocounterbox14[isobox14] = dd; isobox14 = isobox14 + 1;}
226.
227.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
228.     {isocounterbox15[isobox15] = dd; isobox15 = isobox15 + 1;}
229.
230.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
231.     {isocounterbox16[isobox16] = dd; isobox16 = isobox16 + 1;}
232.
233.
234.     }  // end of isosceles triangles location within the 16 screen boxes

235.
236.
237.     int gg, ww;
238.
239.     for (gg = 0; gg < 100; gg = gg+1) {
240.
241.      for (ww = 0; ww < 100; ww = ww +1) {
242.
```

```
243.     ydisplat [gg][ww] = 0.0;
244.     xdisplat [gg][ww] = 0.0;
245.     delus [gg][ww] = 0.0;
246.     delun [gg][ww] = 0.0;
247.     fn [gg][ww] = 0.0;
248.     fs [gg][ww] = 0.0;
249.     dn [gg][ww] = 0.0;
250.     ds [gg][ww] = 0.0;
251.     yforce [gg][ww] = 0.0;
252.     xforce [gg][ww] = 0.0;
253.     fxsum [gg][ww] = 0.0;
254.     x [ww] = 0.0;
255.     resultfx [ww] = 0.0;
256.     fysum [gg][ww] = 0.0;
257.     y [ww] = 0.0;
258.     resultfy [ww] = 0.0;
259.     msum [gg][ww] = 0.0;
260.     m [ww] = 0.0;
261.     resultm [ww] = 0.0;
262.     udoty [gg][ww] = 0.0;
263.     udotysum [ww] = 0.0;
264.     resultudoty [ww] = 0.0;
265.     udotx [gg][ww] = 0.0;
266.     udotxsum [ww] = 0.0;
267.     resultudotx [ww] = 0.0;
268.     thetadot [gg][ww] = 0.0;
269.     thetadotsum [ww] = 0.0;
270.     resultthetadot [ww] = 0.0;
271.     deluy [gg][ww] = 0.0;
272.     deluysum [ww] = 0.0;
273.     resultdeluy [ww] = 0.0;
274.     delux [gg][ww] = 0.0;
275.     deluxsum [ww] = 0.0;
276.     resultdelux [ww] =0.0;
277.     deltheta [gg][ww] = 0.0;
278.     delthetasum [ww] = 0.0;
279.     resultdeltheta [ww] = 0.0;
280.     uy [gg][ww] = 0.0;
281.     uysum [ww] = 0.0;
282.     resultuy [ww] = 0.0;
283.     ux [gg][ww] = 0.0;
284.     uxsum [ww] = 0.0;
285.     resultux [ww] = 0.0;
286.     theta [gg][ww] = 0.0;
287.     thetasum [ww] = 0.0;
288.     resulttheta [ww] = 0.0;
289.     alpha [ww] = 0.0;
290.
291.     }
292.
293.     }
294.
```

```
295.
296.
297.     int xx1 = 0, qq1 = 0, contactpnt1 = 0, l1, i1;
298.
299.
300.     for (i1 = 0; i1 < isobox1; i1= i1+1)
301.
302.     {
303.
304.     xx1 = isocounterbox1[i1];
305.
306.
307.
308.     rightisosceles [xx1][0][0] = rightisosceles [xx1][0][0] + resultux [
   xx1] ;
309.
310.     rightisosceles [xx1][0][1] = rightisosceles [xx1][0][1] + resultuy [
   xx1] ;
311.
312.
313.
314.     for ( l1 = 0; l1 < isobox1; l1 = l1+1)
315.
316.     {
317.
318.
319.     qq1 = isocounterbox1[l1];
320.
321.
322.     if ( ((pow((rightisosceles [xx1][0][0] -
    rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) > 0) && ((pow((rightisosceles [xx1][0][0
   ] - rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) <= 50))
323.
324.     {
325.
326.     contactpnt1 = contactpnt1 + 1;
327.
328.     if (contactpnt1 >= 4) {goto next1;}
329.
330.     if ( rightisosceles [qq1][3][1] >  rightisosceles [xx1][1][1] )
331.
332.     {
333.
334.
335.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][0] -
    rightisosceles [qq1][0][0]));
```

```
336.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][1] -
   rightisosceles [qq1][0][1]));
337.
338.
339.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
340.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
341.
342.
343.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
344.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
345.
346.
347.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
348.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
349.
350.
351.     yforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * sin (alpha1)) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha1)));
352.     xforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * cos (alpha1)) + ((fn [xx1][contactpnt1] + dn [xx1][contactpn
   t1]) * sin (alpha1)));
353.
354.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
355.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
356.
357.
358.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
359.     y [xx1] = fysum [xx1][contactpnt1];
360.     resultfy [xx1] += y[xx1];
361.
362.
363.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
364.     x [xx1] = fxsum [xx1][contactpnt1];
365.     resultfx [xx1] += x[xx1];
366.
367.
368.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][3][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][3][1] -
    rightisosceles [xx1][0][1]));
369.     m [xx1] = msum [xx1][contactpnt1];
```

```
370.     resultm [xx1] += m[xx1];
371.
372.
373.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
374.     udotysum [xx1] = udoty [xx1][contactpnt1];
375.     resultudoty [xx1] += udotysum [xx1];
376.
377.
378.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
379.     udotxsum [xx1] = udotx [xx1][contactpnt1];
380.     resultudotx [xx1] += udotxsum [xx1];
381.
382.
383.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
384.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
385.     resultthetadot [xx1] += thetadotsum [xx1];
386.
387.
388.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
389.     deluysum [xx1] = deluy [xx1][contactpnt1];
390.     resultdeluy [xx1] += deluysum [xx1];
391.
392.
393.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
394.     deluxsum [xx1] = delux [xx1][contactpnt1];
395.     resultdelux [xx1] += deluxsum [xx1];
396.
397.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
398.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
399.     resultdeltheta [xx1] += delthetasum [xx1];
400.
401.
402.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
403.     uysum [xx1] = uy [xx1][contactpnt1];
404.     resultuy [xx1]+= uysum [xx1];
405.
406.
407.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
408.     uxsum [xx1] = ux [xx1][contactpnt1];
409.     resultux [xx1] += uxsum [xx1];
410.
411.
412.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
413.     thetasum [xx1] = theta [xx1][contactpnt1];
414.     resulttheta [xx1] += thetasum [xx1];
415.     alpha [xx1] = alpha1 + resulttheta [xx1];
```

```
416.
417.     }
418.
419.
420.     else
421.     {
422.
423.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][0][0] -
    rightisosceles [qq1][0][0]));
424.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][1][1] -
   rightisosceles [qq1][0][1]));
425.
426.
427.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
428.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
429.
430.
431.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
432.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
433.
434.
435.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
436.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
437.
438.
439.     yforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * sin (alpha [xx1])) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha [xx1])));

440.     xforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * cos (alpha [xx1])) + ((fn [xx1][contactpnt1] + dn [xx1][conta
   ctpnt1]) * sin (alpha [xx1])));
441.
442.
443.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
444.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
445.
446.
```

```
447.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
448.     x [xx1] = fxsum [xx1][contactpnt1];
449.     resultfx [xx1] += x[xx1];
450.
451.
452.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
453.     y [xx1] = fysum [xx1][contactpnt1];
454.     resultfy [xx1] += y[xx1];
455.
456.
457.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][1][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][1][1] -
    rightisosceles [xx1][0][1]));
458.     m [xx1] = msum [xx1][contactpnt1];
459.     resultm [xx1] += m[xx1];
460.
461.
462.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
463.     udotysum [xx1] = udoty [xx1][contactpnt1];
464.     resultudoty [xx1] += udotysum [xx1];
465.
466.
467.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
468.     udotxsum [xx1] = udotx [xx1][contactpnt1];
469.     resultudotx [xx1] += udotxsum [xx1];
470.
471.
472.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
473.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
474.     resultthetadot [xx1] += thetadotsum [xx1];
475.
476.
477.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
478.     deluysum [xx1] = deluy [xx1][contactpnt1];
479.     resultdeluy [xx1] += deluysum [xx1];
480.
481.
482.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
483.     deluxsum [xx1] = delux [xx1][contactpnt1];
484.     resultdelux [xx1] += deluxsum [xx1];
485.
486.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
487.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
488.     resultdeltheta [xx1] += delthetasum [xx1];
489.
```

```
490.
491.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
492.     uysum [xx1] = uy [xx1][contactpnt1];
493.     resultuy [xx1]+= uysum [xx1];
494.
495.
496.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
497.     uxsum [xx1] = ux [xx1][contactpnt1];
498.     resultux [xx1] += uxsum [xx1];
499.
500.
501.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
502.     thetasum [xx1] = theta [xx1][contactpnt1];
503.     resulttheta [xx1] += thetasum [xx1];
504.     alpha [xx1] = alpha1 + resulttheta [xx1];
505.
506.
507.     }
508.
509.
510.
511.      a_file <<  p  << endl;
512.      a_file << xx1 << endl;
513.      a_file << resultuy [xx1]/10 << endl;
514.
515.     if ( resultuy [xx1] > 72 ) { goto display;}
516.
517.
518.
519.     }
520.
521.
522.     }
523.
524.     next1:
525.     ;}
526.
527.
528.
529.     int xx2 = 0, qq2 = 0, contactpnt2 = 0, l2, i2;
530.
531.
532.     for (i2 = 0; i2 < isobox2; i2= i2+1)
533.
534.     {
535.
536.     xx2 = isocounterbox2[i2];
537.
538.
539.     rightisosceles [xx2][0][0] = rightisosceles [xx2][0][0] + resultux [
    xx2] ;
540.
```

```
541.     rightisosceles [xx2][0][1] = rightisosceles [xx2][0][1] + resultuy [
   xx2] ;
542.
543.
544.     for ( l2 = 0; l2 < isobox2; l2 = l2+1)
545.
546.     {
547.
548.
549.     qq2 = isocounterbox2[l2];
550.
551.
552.
553.     if ( ((pow((rightisosceles [xx2][0][0] -
    rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
    rightisosceles [qq2][0][1]), 2)) > 0) && ((pow((rightisosceles [xx2][0][0
   ] - rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
    rightisosceles [qq2][0][1]), 2)) <= 50))
554.
555.     {
556.
557.     contactpnt2 = contactpnt2 + 1;
558.
559.     if (contactpnt2 >= 4) {goto next2;}
560.
561.
562.     if ( rightisosceles [qq2][3][1] >  rightisosceles [xx2][1][1] )
563.
564.     {
565.
566.
567.     ydisplat [xx2][contactpnt2] = ydisplacement2  + (deluy [xx2][contact
   pnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][3][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][0] -
    rightisosceles [qq2][0][0]));
568.     xdisplat [xx2][contactpnt2] = xdisplacement2 + (delux [xx2][contactp
   nt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][3][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][1] -
   rightisosceles [qq2][0][1]));
569.
570.
571.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
572.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
573.
574.
```

```
575.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
576.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
577.
578.
579.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
580.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
581.
582.
583.     yforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * sin (alpha1)) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha1)));
584.     xforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * cos (alpha1)) + ((fn [xx2][contactpnt2] + dn [xx2][contactpn
   t2]) * sin (alpha1)));
585.
586.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
587.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
588.
589.
590.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
591.     y [xx2] = fysum [xx2][contactpnt2];
592.     resultfy [xx2] += y[xx2];
593.
594.
595.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
596.     x [xx2] = fxsum [xx2][contactpnt2];
597.     resultfx [xx2] += x[xx2];
598.
599.
600.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][3][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][3][1] -
    rightisosceles [xx2][0][1]));
601.     m [xx2] = msum [xx2][contactpnt2];
602.     resultm [xx2] += m[xx2];
603.
604.
605.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent) *1000/ dmass); // mm/sec
606.     udotysum [xx2] = udoty [xx2][contactpnt2];
607.     resultudoty [xx2] += udotysum [xx2];
608.
609.
610.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
611.     udotxsum [xx2] = udotx [xx2][contactpnt2];
612.     resultudotx [xx2] += udotxsum [xx2];
613.
```

```
614.
615.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
616.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
617.     resultthetadot [xx2] += thetadotsum [xx2];
618.
619.
620.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
621.     deluysum [xx2] = deluy [xx2][contactpnt2];
622.     resultdeluy [xx2] += deluysum [xx2];
623.
624.
625.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
626.     deluxsum [xx2] = delux [xx2][contactpnt2];
627.     resultdelux [xx2] += deluxsum [xx2];
628.
629.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
630.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
631.     resultdeltheta [xx2] += delthetasum [xx2];
632.
633.
634.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
635.     uysum [xx2] = uy [xx2][contactpnt2];
636.     resultuy [xx2]+= uysum [xx2];
637.
638.
639.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
640.     uxsum [xx2] = ux [xx2][contactpnt2];
641.     resultux [xx2] += uxsum [xx2];
642.
643.
644.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
645.     thetasum [xx2] = theta [xx2][contactpnt2];
646.     resulttheta [xx2] += thetasum [xx2];
647.     alpha [xx2] = alpha1 + resulttheta [xx2];
648.
649.     }
650.
651.     else
652.
653.     {
654.
655.
656.     ydisplat [xx2][contactpnt2] += ydisplacement2  + (deluy [xx2][contac
   tpnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][0][0] -
    rightisosceles [qq2][0][0]));
```

```
657.     xdisplat [xx2][contactpnt2] += xdisplacement2 + (delux [xx2][contact
   pnt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][1][1] -
   rightisosceles [qq2][0][1]));
658.
659.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
660.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
661.
662.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
663.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
664.
665.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
666.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
667.
668.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * sin (alpha [xx2])) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

669.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
   ctpnt2]) * sin (alpha [xx2])));
670.
671.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
672.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
673.
674.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
675.     x [xx2] = fxsum [xx2][contactpnt2];
676.     resultfx [xx2] += x[xx2];
677.
678.
679.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
680.     y [xx2] = fysum [xx2][contactpnt2];
681.     resultfy [xx2] += y[xx2];
682.
683.
684.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][1][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][1][1] -
    rightisosceles [xx2][0][1]));
685.     m [xx2] = msum [xx2][contactpnt2];
686.     resultm [xx2] += m[xx2];
687.
688.
```

```
689.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
690.     udotysum [xx2] = udoty [xx2][contactpnt2];
691.     resultudoty [xx2] += udotysum [xx2];
692.
693.
694.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
695.     udotxsum [xx2] = udotx [xx2][contactpnt2];
696.     resultudotx [xx2] += udotxsum [xx2];
697.
698.
699.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
700.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
701.     resultthetadot [xx2] += thetadotsum [xx2];
702.
703.
704.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
705.     deluysum [xx2] = deluy [xx2][contactpnt2];
706.     resultdeluy [xx2] += deluysum [xx2];
707.
708.
709.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
710.     deluxsum [xx2] = delux [xx2][contactpnt2];
711.     resultdelux [xx2] += deluxsum [xx2];
712.
713.
714.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
715.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
716.     resultdeltheta [xx2] += delthetasum [xx2];
717.
718.
719.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
720.     uysum [xx2] = uy [xx2][contactpnt2];
721.     resultuy [xx2]+= uysum [xx2];
722.
723.
724.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
725.     uxsum [xx2] = ux [xx2][contactpnt2];
726.     resultux [xx2] += uxsum [xx2];
727.
728.
729.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
730.     thetasum [xx2] = theta [xx2][contactpnt2];
731.     resulttheta [xx2] += thetasum [xx2];
732.     alpha [xx2] = alpha1 + resulttheta [xx2];
733.
734.     }
```

```
735.     }
736.
737.     }
738.
739.     next2:
740.     ;}
741.
742.     int xx3 = 0, qq3 = 0, contactpnt3 = 0, l3, i3;
743.
744.
745.     for (i3 = 0; i3 < isobox3; i3= i3+1)
746.
747.     {
748.
749.     xx3 = isocounterbox3[i3];
750.
751.
752.     rightisosceles [xx3][0][0] = rightisosceles [xx3][0][0] + resultux [
    xx3] ;
753.
754.     rightisosceles [xx3][0][1] = rightisosceles [xx3][0][1] + resultuy [
    xx3] ;
755.
756.
757.     for ( l3 = 0; l3 < isobox3; l3 = l3+1)
758.
759.     {
760.
761.
762.     qq3 = isocounterbox3[l3];
763.
764.
765.
766.     if ( ((pow((rightisosceles [xx3][0][0] -
    rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) > 0) && ((pow((rightisosceles [xx3][0][0
    ] - rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) <= 50))
767.
768.     {
769.
770.     contactpnt3 = contactpnt3 + 1;
771.
772.     if (contactpnt3 >= 4) {goto next3;}
773.
774.
775.     if ( rightisosceles [qq3][3][1] >  rightisosceles [xx3][1][1] )
776.
777.     {
778.
779.
```

```
780.     ydisplat [xx3][contactpnt3] = ydisplacement2  + (deluy [xx3][contact
   pnt3] -
    deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][0] - rightisosceles [xx3][0][0])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][0] -
    rightisosceles [qq3][0][0]));
781.     xdisplat [xx3][contactpnt3] = xdisplacement2 + (delux [xx3][contactp
   nt3] -
    delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][1] - rightisosceles [xx3][0][1])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][1] -
   rightisosceles [qq3][0][1]));
782.
783.
784.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
   1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
785.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
   1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
786.
787.
788.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
   1) *  (dstiffness_coefficient/1000);
789.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
   fficient/1000);
790.
791.
792.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
   1) * (ddamping_coefficient/1000);
793.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
   ient/1000);
794.
795.
796.     yforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * sin (alpha1)) -
    ((fn [xx3][contactpnt3] + dn [xx3][contactpnt3]) * cos (alpha1)));
797.     xforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * cos (alpha1)) + ((fn [xx3][contactpnt3] + dn [xx3][contactpn
   t3]) * sin (alpha1)));
798.
799.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
800.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
801.
802.
803.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
804.     y [xx3] = fysum [xx3][contactpnt3];
805.     resultfy [xx3] += y[xx3];
806.
807.
808.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
809.     x [xx3] = fxsum [xx3][contactpnt3];
810.     resultfx [xx3] += x[xx3];
811.
```

```
812.
813.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][3][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][3][1] -
    rightisosceles [xx3][0][1]));
814.     m [xx3] = msum [xx3][contactpnt3];
815.     resultm [xx3] += m[xx3];
816.
817.
818.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent) *1000/ dmass); // mm/sec
819.     udotysum [xx3] = udoty [xx3][contactpnt3];
820.     resultudoty [xx3] += udotysum [xx3];
821.
822.
823.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
824.     udotxsum [xx3] = udotx [xx3][contactpnt3];
825.     resultudotx [xx3] += udotxsum [xx3];
826.
827.
828.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
829.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
830.     resultthetadot [xx3] += thetadotsum [xx3];
831.
832.
833.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
834.     deluysum [xx3] = deluy [xx3][contactpnt3];
835.     resultdeluy [xx3] += deluysum [xx3];
836.
837.
838.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
839.     deluxsum [xx3] = delux [xx3][contactpnt3];
840.     resultdelux [xx3] += deluxsum [xx3];
841.
842.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
843.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
844.     resultdeltheta [xx3] += delthetasum [xx3];
845.
846.
847.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
848.     uysum [xx3] = uy [xx3][contactpnt3];
849.     resultuy [xx3]+= uysum [xx3];
850.
851.
852.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
853.     uxsum [xx3] = ux [xx3][contactpnt3];
854.     resultux [xx3] += uxsum [xx3];
```

```
855.
856.
857.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
858.     thetasum [xx3] = theta [xx3][contactpnt3];
859.     resulttheta [xx3] += thetasum [xx3];
860.     alpha [xx3] = alpha1 + resulttheta [xx3];
861.
862.     }
863.
864.      else
865.     {
866.
867.
868.     ydisplat [xx3][contactpnt2] += ydisplacement2  + (deluy [xx3][contac
    tpnt3] -
     deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
     [qq3][1][0] - rightisosceles [xx3][0][0])  -
     deltheta [qq3][contactpnt3] * (rightisosceles [qq3][0][0] -
     rightisosceles [qq3][0][0]));
869.     xdisplat [xx3][contactpnt2] += xdisplacement2 + (delux [xx3][contact
    pnt3] -
     delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
     [qq3][1][1] - rightisosceles [xx3][0][1])  -
     deltheta [qq3][contactpnt3] * (rightisosceles [qq3][1][1] -
    rightisosceles [qq3][0][1]));
870.
871.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
    1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
872.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
    1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
873.
874.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
    1) *  (dstiffness_coefficient/1000);
875.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
    fficient/1000);
876.
877.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
    1) * (ddamping_coefficient/1000);
878.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
    ient/1000);
879.
880.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
    tactpnt2]) * sin (alpha [xx2])) -
     ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

881.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
    tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
    ctpnt2]) * sin (alpha [xx2])));
882.
883.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
884.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
885.
```

```
886.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
887.     x [xx3] = fxsum [xx3][contactpnt3];
888.     resultfx [xx3] += x[xx3];
889.
890.
891.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
892.     y [xx3] = fysum [xx3][contactpnt3];
893.     resultfy [xx3] += y[xx3];
894.
895.
896.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][1][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][1][1] -
    rightisosceles [xx3][0][1]));
897.     m [xx3] = msum [xx3][contactpnt3];
898.     resultm [xx3] += m[xx3];
899.
900.
901.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
902.     udotysum [xx3] = udoty [xx3][contactpnt3];
903.     resultudoty [xx3] += udotysum [xx3];
904.
905.
906.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
907.     udotxsum [xx3] = udotx [xx3][contactpnt3];
908.     resultudotx [xx3] += udotxsum [xx3];
909.
910.
911.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
912.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
913.     resultthetadot [xx3] += thetadotsum [xx3];
914.
915.
916.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
917.     deluysum [xx3] = deluy [xx3][contactpnt3];
918.     resultdeluy [xx3] += deluysum [xx3];
919.
920.
921.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
922.     deluxsum [xx3] = delux [xx3][contactpnt3];
923.     resultdelux [xx3] += deluxsum [xx3];
924.
925.
926.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
927.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
928.     resultdeltheta [xx3] += delthetasum [xx3];
```

```
929.
930.
931.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
932.     uysum [xx3] = uy [xx3][contactpnt3];
933.     resultuy [xx3]+= uysum [xx3];
934.
935.
936.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
937.     uxsum [xx3] = ux [xx3][contactpnt3];
938.     resultux [xx3] += uxsum [xx3];
939.
940.
941.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
942.     thetasum [xx3] = theta [xx3][contactpnt3];
943.     resulttheta [xx3] += thetasum [xx3];
944.     alpha [xx3] = alpha1 + resulttheta [xx3];
945.     }
946.
947.     }
948.
949.     }
950.
951.     next3:
952.     ;}
953.
954.
955.     int xx4 = 0, qq4 = 0, contactpnt4 = 0, l4, i4;
956.
957.
958.     for (i4 = 0; i4 < isobox4; i4= i4+1)
959.
960.     {
961.
962.     xx4 = isocounterbox4[i4];
963.
964.
965.     rightisosceles [xx4][0][0] = rightisosceles [xx4][0][0] + resultux [
    xx4] ;
966.
967.     rightisosceles [xx4][0][1] = rightisosceles [xx4][0][1] + resultuy [
    xx4] ;
968.
969.
970.     for ( l4 = 0; l4 < isobox4; l4 = l4+1)
971.
972.     {
973.
974.
975.     qq4 = isocounterbox4[l4];
976.
977.
978.
```

```
    if ( ((pow((rightisosceles [xx4][0][0] -
     rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) > 0) && ((pow((rightisosceles [xx4][0][0
    ] - rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) <= 50))

    {

    contactpnt4 = contactpnt4 + 1;

    if (contactpnt4 >= 4) {goto next4;}


    if ( rightisosceles [qq4][3][1] >  rightisosceles [xx4][1][1] )

    {


    ydisplat [xx4][contactpnt4] = ydisplacement2  + (deluy [xx4][contact
   pnt4] -
     deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][0] - rightisosceles [xx4][0][0])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][0] -
     rightisosceles [qq4][0][0]));
    xdisplat [xx4][contactpnt4] = xdisplacement2 + (delux [xx4][contactp
   nt4] -
     delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][1] - rightisosceles [xx4][0][1])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][1] -
   rightisosceles [qq4][0][1]));


    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));


    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-
   1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coe
   fficient/1000);


    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-
   1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coeffic
   ient/1000);


```

```
    yforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * sin (alpha1)) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha1)));
    xforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * cos (alpha1)) + ((fn [xx4][contactpnt4] + dn [xx4][contactpn
t4]) * sin (alpha1)));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisoscel
es [qq4][3][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][3][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
    resultthetadot [xx4] += thetadotsum [xx4];


    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
t;
    deluysum [xx4] = deluy [xx4][contactpnt4];
    resultdeluy [xx4] += deluysum [xx4];

```

```
1050.
1051.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1052.    deluxsum [xx4] = delux [xx4][contactpnt4];
1053.    resultdelux [xx4] += deluxsum [xx4];
1054.
1055.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1056.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1057.    resultdeltheta [xx4] += delthetasum [xx4];
1058.
1059.
1060.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1061.    uysum [xx4] = uy [xx4][contactpnt4];
1062.    resultuy [xx4]+= uysum [xx4];
1063.
1064.
1065.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1066.    uxsum [xx4] = ux [xx4][contactpnt4];
1067.    resultux [xx4] += uxsum [xx4];
1068.
1069.
1070.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1071.    thetasum [xx4] = theta [xx4][contactpnt4];
1072.    resulttheta [xx4] += thetasum [xx4];
1073.    alpha [xx4] = alpha1 + resulttheta [xx4];
1074.
1075.    }
1076.
1077.    else
1078.
1079.    {
1080.
1081.
1082.    ydisplat [xx4][contactpnt4] += ydisplacement2  + (deluy [xx4][contac
   tpnt4] -
     deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][1][0] - rightisosceles [xx4][0][0])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][0][0] -
     rightisosceles [qq4][0][0]));
1083.    xdisplat [xx4][contactpnt4] += xdisplacement2 + (delux [xx4][contact
   pnt4] -
     delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][1][1] - rightisosceles [xx4][0][1])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][1][1] -
   rightisosceles [qq4][0][1]));
1084.
1085.    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
1086.    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));
1087.
```

```
    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coefficient/1000);

    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coefficient/1000);

    yforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][contactpnt4]) * sin (alpha [xx4])) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha [xx4])));

    xforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][contactpnt4]) * cos (alpha [xx4])) + ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * sin (alpha [xx4])));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);

    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisosceles [qq4][1][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][1][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increment)* 1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increment)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


```

```
1125.    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
   ement)*1000/ ii); //1/s
1126.    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
1127.    resultthetadot [xx4] += thetadotsum [xx4];
1128.
1129.
1130.    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
   t;
1131.    deluysum [xx4] = deluy [xx4][contactpnt4];
1132.    resultdeluy [xx4] += deluysum [xx4];
1133.
1134.
1135.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1136.    deluxsum [xx4] = delux [xx4][contactpnt4];
1137.    resultdelux [xx4] += deluxsum [xx4];
1138.
1139.
1140.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1141.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1142.    resultdeltheta [xx4] += delthetasum [xx4];
1143.
1144.
1145.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1146.    uysum [xx4] = uy [xx4][contactpnt4];
1147.    resultuy [xx4]+= uysum [xx4];
1148.
1149.
1150.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1151.    uxsum [xx4] = ux [xx4][contactpnt4];
1152.    resultux [xx4] += uxsum [xx4];
1153.
1154.
1155.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1156.    thetasum [xx4] = theta [xx4][contactpnt4];
1157.    resulttheta [xx4] += thetasum [xx4];
1158.    alpha [xx4] = alpha1 + resulttheta [xx4];
1159.    }
1160.
1161.    }
1162.
1163.    }
1164.
1165.    next4:
1166.    ;}
1167.
1168.
1169.    int xx5 = 0, qq5 = 0, contactpnt5 = 0, l5, i5;
1170.
1171.
1172.    for (i5 = 0; i5 < isobox5; i5= i5+1)
```

```
1173.
1174.    {
1175.
1176.    xx5 = isocounterbox5[i5];
1177.
1178.    rightisosceles [xx5][0][0] = rightisosceles [xx5][0][0] + resultux [
   xx5] ;
1179.
1180.    rightisosceles [xx5][0][1] = rightisosceles [xx5][0][1] + resultuy [
   xx5] ;
1181.
1182.
1183.
1184.    for ( l5 = 0; l5 < isobox5; l5 = l5+1)
1185.
1186.    {
1187.
1188.
1189.    qq5 = isocounterbox5[l5];
1190.
1191.
1192.
1193.    if ( ((pow((rightisosceles [xx5][0][0] -
    rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) > 0) && ((pow((rightisosceles [xx5][0][0
   ] - rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) <= 50))
1194.
1195.    {
1196.
1197.    contactpnt5 = contactpnt5 + 1;
1198.
1199.    if (contactpnt5 >= 4) {goto next5;}
1200.
1201.
1202.    if ( rightisosceles [qq5][3][1] >  rightisosceles [xx5][1][1] )
1203.
1204.    {
1205.
1206.
1207.    ydisplat [xx5][contactpnt5] = ydisplacement2  + (deluy [xx5][contact
   pnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][0] -
    rightisosceles [qq5][0][0]));
1208.    xdisplat [xx5][contactpnt5] = xdisplacement2 + (delux [xx5][contactp
   nt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][1] -
   rightisosceles [qq5][0][1]));
```

```
1209.
1210.
1211.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1212.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1213.
1214.
1215.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1216.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1217.
1218.
1219.    dn [xx5][contactpnt5] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1220.    ds [xx5][contactpnt5] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1221.
1222.
1223.    yforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * sin (alpha1)) -
     ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha1)));
1224.    xforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * cos (alpha1)) + ((fn [xx5][contactpnt5] + dn [xx5][contactpn
   t5]) * sin (alpha1)));
1225.
1226.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1227.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1228.
1229.
1230.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1231.    y [xx5] = fysum [xx5][contactpnt5];
1232.    resultfy [xx5] += y[xx5];
1233.
1234.
1235.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1236.    x [xx5] = fxsum [xx5][contactpnt5];
1237.    resultfx [xx5] += x[xx5];
1238.
1239.
1240.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][3][0] - rightisosceles [xx5][0][0]) -
     xforce [xx5][contactpnt5] * ( rightisosceles [qq5][3][1] -
     rightisosceles [xx5][0][1]));
1241.    m [xx5] = msum [xx5][contactpnt5];
1242.    resultm [xx5] += m[xx5];
1243.
1244.
1245.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1246.    udotysum [xx5] = udoty [xx5][contactpnt5];
```

```
1247.    resultudoty [xx5] += udotysum [xx5];
1248.
1249.
1250.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1251.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1252.    resultudotx [xx5] += udotxsum [xx5];
1253.
1254.
1255.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1256.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1257.    resultthetadot [xx5] += thetadotsum [xx5];
1258.
1259.
1260.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1261.    deluysum [xx5] = deluy [xx5][contactpnt5];
1262.    resultdeluy [xx5] += deluysum [xx5];
1263.
1264.
1265.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1266.    deluxsum [xx5] = delux [xx5][contactpnt5];
1267.    resultdelux [xx5] += deluxsum [xx5];
1268.
1269.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1270.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1271.    resultdeltheta [xx5] += delthetasum [xx5];
1272.
1273.
1274.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1275.    uysum [xx5] = uy [xx5][contactpnt5];
1276.    resultuy [xx5]+= uysum [xx5];
1277.
1278.
1279.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1280.    uxsum [xx5] = ux [xx5][contactpnt5];
1281.    resultux [xx5] += uxsum [xx5];
1282.
1283.
1284.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1285.    thetasum [xx5] = theta [xx5][contactpnt5];
1286.    resulttheta [xx5] += thetasum [xx5];
1287.    alpha [xx5] = alpha1 + resulttheta [xx5];
1288.
1289.    }
1290.
1291.    else
1292.
1293.    {
```

```
1294.
1295.
1296.    ydisplat [xx5][contactpnt5] += ydisplacement2  + (deluy [xx5][contac
   tpnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][0][0] -
    rightisosceles [qq5][0][0]));
1297.    xdisplat [xx5][contactpnt5] += xdisplacement2 + (delux [xx5][contact
   pnt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][1][1] -
   rightisosceles [qq5][0][1]));
1298.
1299.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1300.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1301.
1302.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1303.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1304.
1305.    dn [xx5][contactpnt2] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1306.    ds [xx5][contactpnt2] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1307.
1308.    yforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * sin (alpha [xx5])) -
    ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha [xx5])));

1309.    xforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * cos (alpha [xx5])) + ((fn [xx5][contactpnt5] + dn [xx5][conta
   ctpnt5]) * sin (alpha [xx5])));
1310.
1311.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1312.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1313.
1314.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1315.    x [xx5] = fxsum [xx5][contactpnt5];
1316.    resultfx [xx5] += x[xx5];
1317.
1318.
1319.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1320.    y [xx5] = fysum [xx5][contactpnt5];
1321.    resultfy [xx5] += y[xx5];
1322.
1323.
```

```
1324.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][1][0] - rightisosceles [xx5][0][0]) -
    xforce [xx5][contactpnt5] * ( rightisosceles [qq5][1][1] -
    rightisosceles [xx5][0][1]));
1325.    m [xx5] = msum [xx5][contactpnt5];
1326.    resultm [xx5] += m[xx5];
1327.
1328.
1329.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1330.    udotysum [xx5] = udoty [xx5][contactpnt5];
1331.    resultudoty [xx5] += udotysum [xx5];
1332.
1333.
1334.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1335.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1336.    resultudotx [xx5] += udotxsum [xx5];
1337.
1338.
1339.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1340.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1341.    resultthetadot [xx5] += thetadotsum [xx5];
1342.
1343.
1344.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1345.    deluysum [xx5] = deluy [xx5][contactpnt5];
1346.    resultdeluy [xx5] += deluysum [xx5];
1347.
1348.
1349.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1350.    deluxsum [xx5] = delux [xx5][contactpnt5];
1351.    resultdelux [xx5] += deluxsum [xx5];
1352.
1353.
1354.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1355.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1356.    resultdeltheta [xx5] += delthetasum [xx5];
1357.
1358.
1359.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1360.    uysum [xx5] = uy [xx5][contactpnt5];
1361.    resultuy [xx5]+= uysum [xx5];
1362.
1363.
1364.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1365.    uxsum [xx5] = ux [xx5][contactpnt5];
1366.    resultux [xx5] += uxsum [xx5];
```

```
1367.
1368.
1369.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1370.    thetasum [xx5] = theta [xx5][contactpnt5];
1371.    resulttheta [xx5] += thetasum [xx5];
1372.    alpha [xx5] = alpha1 + resulttheta [xx5];
1373.
1374.    }
1375.    }
1376.
1377.    }
1378.
1379.    next5:
1380.    ;}
1381.
1382.
1383.    int xx6 = 0, qq6 = 0, contactpnt6 = 0, l6, i6;
1384.
1385.
1386.    for (i6 = 0; i6 < isobox6; i6= i6+1)
1387.
1388.    {
1389.
1390.    xx6 = isocounterbox6[i6];
1391.
1392.    rightisosceles [xx6][0][0] = rightisosceles [xx6][0][0] + resultux [
   xx6] ;
1393.
1394.    rightisosceles [xx6][0][1] = rightisosceles [xx6][0][1] + resultuy [
   xx6] ;
1395.
1396.
1397.
1398.    for ( l6 = 0; l6 < isobox6; l6 = l6+1)
1399.
1400.    {
1401.
1402.
1403.    qq6 = isocounterbox6[l6];
1404.
1405.
1406.
1407.    if ( ((pow((rightisosceles [xx6][0][0] -
    rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) > 0) && ((pow((rightisosceles [xx6][0][0
   ] - rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) <= 50))
1408.
1409.    {
1410.
1411.    contactpnt6 = contactpnt6 + 1;
1412.
```

```
1413.    if (contactpnt6 >= 4) {goto next6;}
1414.
1415.
1416.    if ( rightisosceles [qq6][3][1] >  rightisosceles [xx6][1][1] )
1417.
1418.    {
1419.
1420.
1421.    ydisplat [xx6][contactpnt6] = ydisplacement2  + (deluy [xx6][contact
   pnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][0] -
    rightisosceles [qq6][0][0]));
1422.    xdisplat [xx6][contactpnt6] = xdisplacement2 + (delux [xx6][contactp
   nt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][1] -
   rightisosceles [qq6][0][1]));
1423.
1424.
1425.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1426.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1427.
1428.
1429.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1430.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1431.
1432.
1433.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1434.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1435.
1436.
1437.    yforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * sin (alpha1)) -
     ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha1)));
1438.    xforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * cos (alpha1)) + ((fn [xx6][contactpnt6] + dn [xx6][contactpn
   t6]) * sin (alpha1)));
1439.
1440.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1441.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1442.
1443.
1444.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
```

```
1445.    y [xx6] = fysum [xx6][contactpnt6];
1446.    resultfy [xx6] += y[xx6];
1447.
1448.
1449.    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
1450.    x [xx6] = fxsum [xx6][contactpnt6];
1451.    resultfx [xx6] += x[xx6];
1452.
1453.
1454.    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][3][0] - rightisosceles [xx6][0][0]) -
    xforce [xx6][contactpnt6] * ( rightisosceles [qq6][3][1] -
    rightisosceles [xx6][0][1]));
1455.    m [xx6] = msum [xx6][contactpnt6];
1456.    resultm [xx6] += m[xx6];
1457.
1458.
1459.    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1460.    udotysum [xx6] = udoty [xx6][contactpnt6];
1461.    resultudoty [xx6] += udotysum [xx6];
1462.
1463.
1464.    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1465.    udotxsum [xx6] = udotx [xx6][contactpnt6];
1466.    resultudotx [xx6] += udotxsum [xx6];
1467.
1468.
1469.    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
1470.    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
1471.    resultthetadot [xx6] += thetadotsum [xx6];
1472.
1473.
1474.    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
1475.    deluysum [xx6] = deluy [xx6][contactpnt6];
1476.    resultdeluy [xx6] += deluysum [xx6];
1477.
1478.
1479.    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
1480.    deluxsum [xx6] = delux [xx6][contactpnt6];
1481.    resultdelux [xx6] += deluxsum [xx6];
1482.
1483.    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
1484.    delthetasum [xx6] = deltheta [xx6][contactpnt6];
1485.    resultdeltheta [xx6] += delthetasum [xx6];
1486.
1487.
```

```
1488.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1489.    uysum [xx6] = uy [xx6][contactpnt6];
1490.    resultuy [xx6]+= uysum [xx6];
1491.
1492.
1493.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1494.    uxsum [xx6] = ux [xx6][contactpnt6];
1495.    resultux [xx6] += uxsum [xx6];
1496.
1497.
1498.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1499.    thetasum [xx6] = theta [xx6][contactpnt6];
1500.    resulttheta [xx6] += thetasum [xx6];
1501.    alpha [xx6] = alpha1 + resulttheta [xx6];
1502.
1503.    }
1504.
1505.    else
1506.
1507.    {
1508.
1509.
1510.
1511.    ydisplat [xx6][contactpnt6] += ydisplacement2  + (deluy [xx6][contac
   tpnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][0][0] -
    rightisosceles [qq6][0][0]));
1512.    xdisplat [xx6][contactpnt6] += xdisplacement2 + (delux [xx6][contact
   pnt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][1][1] -
   rightisosceles [qq6][0][1]));
1513.
1514.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1515.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1516.
1517.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1518.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1519.
1520.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1521.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1522.
```

```
1523.    yforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * sin (alpha [xx6])) -
    ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha [xx6])));

1524.    xforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * cos (alpha [xx6])) + ((fn [xx6][contactpnt6] + dn [xx6][conta
   ctpnt6]) * sin (alpha [xx6])));
1525.
1526.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1527.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1528.
1529.    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
1530.    x [xx6] = fxsum [xx6][contactpnt6];
1531.    resultfx [xx6] += x[xx6];
1532.
1533.
1534.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
1535.    y [xx6] = fysum [xx6][contactpnt6];
1536.    resultfy [xx6] += y[xx6];
1537.
1538.
1539.    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][1][0] - rightisosceles [xx6][0][0]) -
    xforce [xx6][contactpnt6] * ( rightisosceles [qq6][1][1] -
    rightisosceles [xx6][0][1]));
1540.    m [xx6] = msum [xx6][contactpnt6];
1541.    resultm [xx6] += m[xx6];
1542.
1543.
1544.    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1545.    udotysum [xx6] = udoty [xx6][contactpnt6];
1546.    resultudoty [xx6] += udotysum [xx6];
1547.
1548.
1549.    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1550.    udotxsum [xx6] = udotx [xx6][contactpnt6];
1551.    resultudotx [xx6] += udotxsum [xx6];
1552.
1553.
1554.    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
1555.    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
1556.    resultthetadot [xx6] += thetadotsum [xx6];
1557.
1558.
1559.    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
1560.    deluysum [xx6] = deluy [xx6][contactpnt6];
1561.    resultdeluy [xx6] += deluysum [xx6];
1562.
```

```
1563.
1564.    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
1565.    deluxsum [xx6] = delux [xx6][contactpnt6];
1566.    resultdelux [xx6] += deluxsum [xx6];
1567.
1568.
1569.    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
1570.    delthetasum [xx6] = deltheta [xx6][contactpnt6];
1571.    resultdeltheta [xx6] += delthetasum [xx6];
1572.
1573.
1574.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1575.    uysum [xx6] = uy [xx6][contactpnt6];
1576.    resultuy [xx6]+= uysum [xx6];
1577.
1578.
1579.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1580.    uxsum [xx6] = ux [xx6][contactpnt6];
1581.    resultux [xx6] += uxsum [xx6];
1582.
1583.
1584.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1585.    thetasum [xx6] = theta [xx6][contactpnt6];
1586.    resulttheta [xx6] += thetasum [xx6];
1587.    alpha [xx6] = alpha1 + resulttheta [xx6];
1588.
1589.    }
1590.
1591.    }
1592.
1593.    }
1594.
1595.    next6:
1596.    ;}
1597.
1598.
1599.
1600.    int xx7 = 0, qq7 = 0, contactpnt7 = 0, l7, i7;
1601.
1602.
1603.    for (i7 = 0; i7 < isobox7; i7= i7+1)
1604.
1605.    {
1606.
1607.    xx7 = isocounterbox7[i7];
1608.
1609.    rightisosceles [xx7][0][0] = rightisosceles [xx7][0][0] + resultux [
   xx7] ;
1610.
```

```
    rightisosceles [xx7][0][1] = rightisosceles [xx7][0][1] + resultuy [
   xx7] ;



    for ( l7 = 0; l7 < isobox7; l7 = l7+1)

    {


    qq7 = isocounterbox7[l7];


    if ( ((pow((rightisosceles [xx7][0][0] -
    rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) > 0) && ((pow((rightisosceles [xx7][0][0
   ] - rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) <= 50))

    {

    contactpnt7 = contactpnt7 + 1;

    if (contactpnt7 >= 4) {goto next7;}


    if ( rightisosceles [qq7][3][1] >  rightisosceles [xx7][1][1] )

    {


    ydisplat [xx7][contactpnt7] = ydisplacement2  + (deluy [xx7][contact
   pnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][0] - rightisosceles [xx7][0][0])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][0] -
    rightisosceles [qq7][0][0]));
    xdisplat [xx7][contactpnt7] = xdisplacement2 + (delux [xx7][contactp
   nt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][1] -
   rightisosceles [qq7][0][1]));


    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));


```

```
1646.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1647.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1648.
1649.
1650.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1651.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1652.
1653.
1654.    yforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * sin (alpha1)) -
     ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha1)));
1655.    xforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * cos (alpha1)) + ((fn [xx7][contactpnt7] + dn [xx7][contactpn
   t7]) * sin (alpha1)));
1656.
1657.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1658.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1659.
1660.
1661.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1662.    y [xx7] = fysum [xx7][contactpnt7];
1663.    resultfy [xx7] += y[xx7];
1664.
1665.
1666.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1667.    x [xx7] = fxsum [xx7][contactpnt7];
1668.    resultfx [xx7] += x[xx7];
1669.
1670.
1671.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][3][0] - rightisosceles [xx7][0][0]) -
     xforce [xx7][contactpnt7] * ( rightisosceles [qq7][3][1] -
     rightisosceles [xx7][0][1]));
1672.    m [xx7] = msum [xx7][contactpnt7];
1673.    resultm [xx7] += m[xx7];
1674.
1675.
1676.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1677.    udotysum [xx7] = udoty [xx7][contactpnt7];
1678.    resultudoty [xx7] += udotysum [xx7];
1679.
1680.
1681.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1682.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1683.    resultudotx [xx7] += udotxsum [xx7];
1684.
```

```
1685.
1686.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1687.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1688.    resultthetadot [xx7] += thetadotsum [xx7];
1689.
1690.
1691.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1692.    deluysum [xx7] = deluy [xx7][contactpnt7];
1693.    resultdeluy [xx7] += deluysum [xx7];
1694.
1695.
1696.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1697.    deluxsum [xx7] = delux [xx7][contactpnt7];
1698.    resultdelux [xx7] += deluxsum [xx7];
1699.
1700.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1701.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1702.    resultdeltheta [xx7] += delthetasum [xx7];
1703.
1704.
1705.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1706.    uysum [xx7] = uy [xx7][contactpnt7];
1707.    resultuy [xx7]+= uysum [xx7];
1708.
1709.
1710.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1711.    uxsum [xx7] = ux [xx7][contactpnt7];
1712.    resultux [xx7] += uxsum [xx7];
1713.
1714.
1715.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1716.    thetasum [xx7] = theta [xx7][contactpnt7];
1717.    resultttheta [xx7] += thetasum [xx7];
1718.    alpha [xx7] = alpha1 + resultttheta [xx7];
1719.
1720.    }
1721.
1722.    else
1723.
1724.    {
1725.
1726.
1727.
1728.    ydisplat [xx7][contactpnt7] += ydisplacement2  + (deluy [xx7][contac
   tpnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][0] - rightisosceles [xx7][0][0])  -
```

```
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][0][0] -
    rightisosceles [qq7][0][0]));
1729.    xdisplat [xx7][contactpnt7] += xdisplacement2 + (delux [xx7][contact
   pnt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][1][1] -
   rightisosceles [qq7][0][1]));
1730.
1731.    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
1732.    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));
1733.
1734.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1735.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1736.
1737.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1738.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1739.
1740.    yforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * sin (alpha [xx7])) -
    ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha [xx7])));

1741.    xforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * cos (alpha [xx7])) + ((fn [xx7][contactpnt7] + dn [xx7][conta
   ctpnt7]) * sin (alpha [xx7])));
1742.
1743.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1744.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1745.
1746.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1747.    x [xx7] = fxsum [xx7][contactpnt7];
1748.    resultfx [xx7] += x[xx7];
1749.
1750.
1751.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1752.    y [xx7] = fysum [xx7][contactpnt7];
1753.    resultfy [xx7] += y[xx7];
1754.
1755.
1756.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][1][0] - rightisosceles [xx7][0][0]) -
    xforce [xx7][contactpnt7] * ( rightisosceles [qq7][1][1] -
    rightisosceles [xx7][0][1]));
1757.    m [xx7] = msum [xx7][contactpnt7];
1758.    resultm [xx7] += m[xx7];
1759.
```

```
1760.
1761.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1762.    udotysum [xx7] = udoty [xx7][contactpnt7];
1763.    resultudoty [xx7] += udotysum [xx7];
1764.
1765.
1766.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1767.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1768.    resultudotx [xx7] += udotxsum [xx7];
1769.
1770.
1771.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1772.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1773.    resultthetadot [xx7] += thetadotsum [xx7];
1774.
1775.
1776.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1777.    deluysum [xx7] = deluy [xx7][contactpnt7];
1778.    resultdeluy [xx7] += deluysum [xx7];
1779.
1780.
1781.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1782.    deluxsum [xx7] = delux [xx7][contactpnt7];
1783.    resultdelux [xx7] += deluxsum [xx7];
1784.
1785.
1786.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1787.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1788.    resultdeltheta [xx7] += delthetasum [xx7];
1789.
1790.
1791.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1792.    uysum [xx7] = uy [xx7][contactpnt7];
1793.    resultuy [xx7]+= uysum [xx7];
1794.
1795.
1796.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1797.    uxsum [xx7] = ux [xx7][contactpnt7];
1798.    resultux [xx7] += uxsum [xx7];
1799.
1800.
1801.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1802.    thetasum [xx7] = theta [xx7][contactpnt7];
1803.    resulttheta [xx7] += thetasum [xx7];
1804.    alpha [xx7] = alpha1 + resulttheta [xx7];
1805.    }
```

```
1806.
1807.    }
1808.
1809.    }
1810.
1811.    next7:
1812.    ;}
1813.
1814.
1815.
1816.    int xx8 = 0, qq8 = 0, contactpnt8 = 0, l8, i8;
1817.
1818.
1819.    for (i8 = 0; i8 < isobox8; i8= i8+1)
1820.
1821.    {
1822.
1823.    xx8 = isocounterbox8[i8];
1824.
1825.    rightisosceles [xx8][0][0] = rightisosceles [xx8][0][0] + resultux [
   xx8] ;
1826.
1827.    rightisosceles [xx8][0][1] = rightisosceles [xx8][0][1] + resultuy [
   xx8] ;
1828.
1829.
1830.
1831.    for ( l8 = 0; l8 < isobox8; l8 = l8+1)
1832.
1833.    {
1834.
1835.
1836.    qq8 = isocounterbox8[l8];
1837.
1838.
1839.
1840.    if ( ((pow((rightisosceles [xx8][0][0] -
    rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) > 0) && ((pow((rightisosceles [xx8][0][0
   ] - rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) <= 50))
1841.
1842.    {
1843.
1844.    contactpnt8 = contactpnt8 + 1;
1845.
1846.    if (contactpnt8 >= 4) {goto next8;}
1847.
1848.
1849.    if ( rightisosceles [qq8][3][1] >  rightisosceles [xx8][1][1] )
1850.
1851.    {
```

```
1852.
1853.
1854.    ydisplat [xx8][contactpnt8] = ydisplacement2  + (deluy [xx8][contact
   pnt8] -
    deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][0] - rightisosceles [xx8][0][0])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][0] -
    rightisosceles [qq8][0][0]));
1855.    xdisplat [xx8][contactpnt8] = xdisplacement2 + (delux [xx8][contactp
   nt8] -
    delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][1] - rightisosceles [xx8][0][1])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][1] -
   rightisosceles [qq8][0][1]));
1856.
1857.
1858.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1859.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1860.
1861.
1862.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt8] * (-
   1) *  (dstiffness_coefficient/1000);
1863.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt8] *  (dstiffness_coe
   fficient/1000);
1864.
1865.
1866.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1867.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1868.
1869.
1870.    yforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * sin (alpha1)) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha1)));
1871.    xforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * cos (alpha1)) + ((fn [xx8][contactpnt8] + dn [xx8][contactpn
   t8]) * sin (alpha1)));
1872.
1873.    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
1874.    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);
1875.
1876.
1877.    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
1878.    y [xx8] = fysum [xx8][contactpnt8];
1879.    resultfy [xx8] += y[xx8];
1880.
1881.
1882.    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
1883.    x [xx8] = fxsum [xx8][contactpnt8];
```

```
1884.    resultfx [xx8] += x[xx8];
1885.
1886.
1887.    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
   es [qq8][3][0] - rightisosceles [xx8][0][0]) -
    xforce [xx8][contactpnt8] * ( rightisosceles [qq8][3][1] -
    rightisosceles [xx8][0][1]));
1888.    m [xx8] = msum [xx8][contactpnt8];
1889.    resultm [xx8] += m[xx8];
1890.
1891.
1892.    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1893.    udotysum [xx8] = udoty [xx8][contactpnt8];
1894.    resultudoty [xx8] += udotysum [xx8];
1895.
1896.
1897.    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1898.    udotxsum [xx8] = udotx [xx8][contactpnt8];
1899.    resultudotx [xx8] += udotxsum [xx8];
1900.
1901.
1902.    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
   ement)*1000/ ii); //1/s
1903.    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
1904.    resultthetadot [xx8] += thetadotsum [xx8];
1905.
1906.
1907.    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
   t;
1908.    deluysum [xx8] = deluy [xx8][contactpnt8];
1909.    resultdeluy [xx8] += deluysum [xx8];
1910.
1911.
1912.    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
   t;
1913.    deluxsum [xx8] = delux [xx8][contactpnt8];
1914.    resultdelux [xx8] += deluxsum [xx8];
1915.
1916.    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
   crement;
1917.    delthetasum [xx8] = deltheta [xx8][contactpnt8];
1918.    resultdeltheta [xx8] += delthetasum [xx8];
1919.
1920.
1921.    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
1922.    uysum [xx8] = uy [xx8][contactpnt8];
1923.    resultuy [xx8]+= uysum [xx8];
1924.
1925.
1926.    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
```

```
1927.    uxsum [xx8] = ux [xx8][contactpnt8];
1928.    resultux [xx8] += uxsum [xx8];
1929.
1930.
1931.    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
1932.    thetasum [xx8] = theta [xx8][contactpnt8];
1933.    resulttheta [xx8] += thetasum [xx8];
1934.    alpha [xx8] = alpha1 + resulttheta [xx8];
1935.
1936.    }
1937.
1938.    else
1939.
1940.    {
1941.
1942.
1943.    ydisplat [xx8][contactpnt8] += ydisplacement2  + (deluy [xx8][contac
   tpnt8] -
     deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][0] - rightisosceles [xx8][0][0])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][0][0] -
     rightisosceles [qq8][0][0]));
1944.    xdisplat [xx8][contactpnt8] += xdisplacement2 + (delux [xx8][contact
   pnt8] -
     delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][1] - rightisosceles [xx8][0][1])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][1][1] -
   rightisosceles [qq8][0][1]));
1945.
1946.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1947.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1948.
1949.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1950.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1951.
1952.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1953.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1954.
1955.    yforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * sin (alpha [xx8])) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha [xx8])));

1956.    xforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * cos (alpha [xx8])) + ((fn [xx8][contactpnt8] + dn [xx8][conta
   ctpnt8]) * sin (alpha [xx8])));
1957.
```

```
    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);

    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
    x [xx8] = fxsum [xx8][contactpnt8];
    resultfx [xx8] += x[xx8];


    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
    y [xx8] = fysum [xx8][contactpnt8];
    resultfy [xx8] += y[xx8];


    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
es [qq8][1][0] - rightisosceles [xx8][0][0]) -
     xforce [xx8][contactpnt8] * ( rightisosceles [qq8][1][1] -
     rightisosceles [xx8][0][1]));
    m [xx8] = msum [xx8][contactpnt8];
    resultm [xx8] += m[xx8];


    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
ent)* 1000/ dmass); // mm/sec
    udotysum [xx8] = udoty [xx8][contactpnt8];
    resultudoty [xx8] += udotysum [xx8];


    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx8] = udotx [xx8][contactpnt8];
    resultudotx [xx8] += udotxsum [xx8];


    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
    resultthetadot [xx8] += thetadotsum [xx8];


    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
t;
    deluysum [xx8] = deluy [xx8][contactpnt8];
    resultdeluy [xx8] += deluysum [xx8];


    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
t;
    deluxsum [xx8] = delux [xx8][contactpnt8];
    resultdelux [xx8] += deluxsum [xx8];


```

```
    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
crement;
    delthetasum [xx8] = deltheta [xx8][contactpnt8];
    resultdeltheta [xx8] += delthetasum [xx8];


    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
    uysum [xx8] = uy [xx8][contactpnt8];
    resultuy [xx8]+= uysum [xx8];


    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
    uxsum [xx8] = ux [xx8][contactpnt8];
    resultux [xx8] += uxsum [xx8];


    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
    thetasum [xx8] = theta [xx8][contactpnt8];
    resulttheta [xx8] += thetasum [xx8];
    alpha [xx8] = alpha1 + resulttheta [xx8];


    }
    }

    }

    next8:
    ;}


    int xx9 = 0, qq9 = 0, contactpnt9 = 0, l9, i9;


    for (i9 = 0; i9 < isobox9; i9= i9+1)

    {

    xx9 = isocounterbox9[i9];

    rightisosceles [xx9][0][0] = rightisosceles [xx9][0][0] + resultux [
xx9] ;

    rightisosceles [xx9][0][1] = rightisosceles [xx9][0][1] + resultuy [
xx9] ;


    for ( l9 = 0; l9 < isobox9; l9 = l9+1)

    {
```

```
2050.
2051.
2052.    qq9 = isocounterbox9[19];
2053.
2054.
2055.
2056.    if ( ((pow((rightisosceles [xx9][0][0] -
    rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) > 0) && ((pow((rightisosceles [xx9][0][0
   ] - rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) <= 50))
2057.
2058.    {
2059.
2060.    contactpnt9 = contactpnt9 + 1;
2061.
2062.    if (contactpnt9 >= 4) {goto next9;}
2063.
2064.
2065.    if ( rightisosceles [qq9][3][1] >  rightisosceles [xx9][1][1] )
2066.
2067.    {
2068.
2069.
2070.    ydisplat [xx9][contactpnt9] = ydisplacement2  + (deluy [xx9][contact
   pnt9] -
    deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][0] - rightisosceles [xx9][0][0])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][0] -
    rightisosceles [qq9][0][0]));
2071.    xdisplat [xx9][contactpnt9] = xdisplacement2 + (delux [xx9][contactp
   nt9] -
    delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][1] - rightisosceles [xx9][0][1])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][1] -
   rightisosceles [qq9][0][1]));
2072.
2073.
2074.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2075.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2076.
2077.
2078.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2079.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2080.
2081.
2082.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
```

```
    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
ient/1000);


    yforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * sin (alpha1)) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha1)));
    xforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * cos (alpha1)) + ((fn [xx9][contactpnt9] + dn [xx9][contactpn
t9]) * sin (alpha1)));

    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);


    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
    y [xx9] = fysum [xx9][contactpnt9];
    resultfy [xx9] += y[xx9];


    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
    x [xx9] = fxsum [xx9][contactpnt9];
    resultfx [xx9] += x[xx9];


    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
es [qq9][3][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][3][1] -
     rightisosceles [xx9][0][1]));
    m [xx9] = msum [xx9][contactpnt9];
    resultm [xx9] += m[xx9];


    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx9] = udoty [xx9][contactpnt9];
    resultudoty [xx9] += udotysum [xx9];


    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


```

```
2123.    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
   t;
2124.    deluysum [xx9] = deluy [xx9][contactpnt9];
2125.    resultdeluy [xx9] += deluysum [xx9];
2126.
2127.
2128.    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
   t;
2129.    deluxsum [xx9] = delux [xx9][contactpnt9];
2130.    resultdelux [xx9] += deluxsum [xx9];
2131.
2132.    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
   crement;
2133.    delthetasum [xx9] = deltheta [xx9][contactpnt9];
2134.    resultdeltheta [xx9] += delthetasum [xx9];
2135.
2136.
2137.    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
2138.    uysum [xx9] = uy [xx9][contactpnt9];
2139.    resultuy [xx9]+= uysum [xx9];
2140.
2141.
2142.    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
2143.    uxsum [xx9] = ux [xx9][contactpnt9];
2144.    resultux [xx9] += uxsum [xx9];
2145.
2146.
2147.    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
2148.    thetasum [xx9] = theta [xx9][contactpnt9];
2149.    resulttheta [xx9] += thetasum [xx9];
2150.    alpha [xx9] = alpha1 + resulttheta [xx9];
2151.
2152.    }
2153.
2154.    else
2155.
2156.    {
2157.
2158.
2159.    ydisplat [xx9][contactpnt9] += ydisplacement2  + (deluy [xx9][contac
   tpnt9] -
     deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][0] - rightisosceles [xx9][0][0])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][0][0] -
     rightisosceles [qq9][0][0]));
2160.    xdisplat [xx9][contactpnt9] += xdisplacement2 + (delux [xx9][contact
   pnt9] -
     delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][1] - rightisosceles [xx9][0][1])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][1][1] -
   rightisosceles [qq9][0][1]));
2161.
```

```
2162.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2163.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2164.
2165.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2166.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2167.
2168.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
2169.    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
   ient/1000);
2170.
2171.    yforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * sin (alpha [xx9])) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha [xx9])));

2172.    xforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * cos (alpha [xx9])) + ((fn [xx9][contactpnt9] + dn [xx9][conta
   ctpnt9]) * sin (alpha [xx9])));
2173.
2174.    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
2175.    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);
2176.
2177.    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
2178.    x [xx9] = fxsum [xx9][contactpnt9];
2179.    resultfx [xx9] += x[xx9];
2180.
2181.
2182.    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
2183.    y [xx9] = fysum [xx9][contactpnt9];
2184.    resultfy [xx9] += y[xx9];
2185.
2186.
2187.    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
   es [qq9][1][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][1][1] -
     rightisosceles [xx9][0][1]));
2188.    m [xx9] = msum [xx9][contactpnt9];
2189.    resultm [xx9] += m[xx9];
2190.
2191.
2192.    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
2193.    udotysum [xx9] = udoty [xx9][contactpnt9];
2194.    resultudoty [xx9] += udotysum [xx9];
2195.
2196.
2197.    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
   ent)*1000/ dmass); //mm/sec
```

```
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
t;
    deluysum [xx9] = deluy [xx9][contactpnt9];
    resultdeluy [xx9] += deluysum [xx9];


    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
t;
    deluxsum [xx9] = delux [xx9][contactpnt9];
    resultdelux [xx9] += deluxsum [xx9];


    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
crement;
    delthetasum [xx9] = deltheta [xx9][contactpnt9];
    resultdeltheta [xx9] += delthetasum [xx9];


    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
    uysum [xx9] = uy [xx9][contactpnt9];
    resultuy [xx9]+= uysum [xx9];


    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
    uxsum [xx9] = ux [xx9][contactpnt9];
    resultux [xx9] += uxsum [xx9];


    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
    thetasum [xx9] = theta [xx9][contactpnt9];
    resulttheta [xx9] += thetasum [xx9];
    alpha [xx9] = alpha1 + resulttheta [xx9];

    }
    }

    }

    next9:
    ;}

```

```
2246.
2247.    int xx10 = 0, qq10 = 0, contactpnt10 = 0, l10, i10;
2248.
2249.
2250.    for (i10 = 0; i10 < isobox10; i10= i10+1)
2251.
2252.    {
2253.
2254.    xx10 = isocounterbox10[i10];
2255.
2256.    rightisosceles [xx10][0][0] = rightisosceles [xx10][0][0] + resultux
    [xx10] ;
2257.
2258.    rightisosceles [xx10][0][1] = rightisosceles [xx10][0][1] + resultuy
    [xx10] ;
2259.
2260.
2261.
2262.    for ( l10 = 0; l10 < isobox10; l10 = l10+1)
2263.
2264.    {
2265.
2266.
2267.    qq10 = isocounterbox10[l10];
2268.
2269.
2270.
2271.    if ( ((pow((rightisosceles [xx10][0][0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) > 0) && ((pow((rightisosceles [xx10][0]
   [0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) <= 50))
2272.
2273.    {
2274.
2275.    contactpnt10 = contactpnt10 + 1;
2276.
2277.    if (contactpnt10 >= 4) {goto next10;}
2278.
2279.
2280.    if ( rightisosceles [qq10][3][1] >  rightisosceles [xx10][1][1] )
2281.
2282.    {
2283.
2284.
2285.    ydisplat [xx10][contactpnt10] = ydisplacement2  + (deluy [xx10][cont
   actpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][0] -
    rightisosceles [qq10][0][0]));
```

```
2286.    xdisplat [xx10][contactpnt10] = xdisplacement2 + (delux [xx10][conta
   ctpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][1] -
   rightisosceles [qq10][0][1]));
2287.
2288.
2289.    delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2290.    delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2291.
2292.
2293.    fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2294.    fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2295.
2296.
2297.    dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2298.    ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2299.
2300.
2301.    yforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * sin (alpha1)) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha1)));
2302.    xforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * cos (alpha1)) + ((fn [xx10][contactpnt10] + dn [xx10][
   contactpnt10]) * sin (alpha1)));
2303.
2304.    yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2305.    xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2306.
2307.
2308.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2309.    y [xx10] = fysum [xx10][contactpnt10];
2310.    resultfy [xx10] += y[xx10];
2311.
2312.
2313.    fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2314.    x [xx10] = fxsum [xx10][contactpnt10];
2315.    resultfx [xx10] += x[xx10];
2316.
2317.
2318.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][3][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][3][1] -
    rightisosceles [xx10][0][1]));
2319.    m [xx10] = msum [xx10][contactpnt10];
```

```
2320.    resultm [xx10] += m[xx10];
2321.
2322.
2323.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement) *1000/ dmass); // mm/sec
2324.    udotysum [xx10] = udoty [xx10][contactpnt10];
2325.    resultudoty [xx10] += udotysum [xx10];
2326.
2327.
2328.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2329.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2330.    resultudotx [xx10] += udotxsum [xx10];
2331.
2332.
2333.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2334.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2335.    resultthetadot [xx10] += thetadotsum [xx10];
2336.
2337.
2338.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2339.    deluysum [xx10] = deluy [xx10][contactpnt10];
2340.    resultdeluy [xx10] += deluysum [xx10];
2341.
2342.
2343.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2344.    deluxsum [xx10] = delux [xx10][contactpnt10];
2345.    resultdelux [xx10] += deluxsum [xx10];
2346.
2347.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2348.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2349.    resultdeltheta [xx10] += delthetasum [xx10];
2350.
2351.
2352.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2353.    uysum [xx10] = uy [xx10][contactpnt10];
2354.    resultuy [xx10]+= uysum [xx10];
2355.
2356.
2357.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2358.    uxsum [xx10] = ux [xx10][contactpnt10];
2359.    resultux [xx10] += uxsum [xx10];
2360.
2361.
2362.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2363.    thetasum [xx10] = theta [xx10][contactpnt10];
2364.    resultthetа [xx10] += thetasum [xx10];
2365.    alpha [xx10] = alpha1 + resultthetа [xx10];
```

```
2366.
2367.    }
2368.
2369.    else
2370.
2371.    {
2372.
2373.
2374.    ydisplat [xx10][contactpnt10] += ydisplacement2  + (deluy [xx10][con
   tactpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][0][0] -
    rightisosceles [qq10][0][0]));
2375.    xdisplat [xx10][contactpnt10] += xdisplacement2 + (delux [xx10][cont
   actpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][1][1] -
   rightisosceles [qq10][0][1]));
2376.
2377.    delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2378.    delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2379.
2380.    fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2381.    fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2382.
2383.    dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2384.    ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2385.
2386.    yforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * sin (alpha [xx10])) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha [xx10])
   ));
2387.    xforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * cos (alpha [xx10])) + ((fn [xx10][contactpnt10] + dn [x
   x10][contactpnt10]) * sin (alpha [xx10])));
2388.
2389.    yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2390.    xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2391.
2392.    fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2393.    x [xx10] = fxsum [xx10][contactpnt10];
2394.    resultfx [xx10] += x[xx10];
2395.
2396.
```

```
2397.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2398.    y [xx10] = fysum [xx10][contactpnt10];
2399.    resultfy [xx10] += y[xx10];
2400.
2401.
2402.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][1][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][1][1] -
    rightisosceles [xx10][0][1]));
2403.    m [xx10] = msum [xx10][contactpnt10];
2404.    resultm [xx10] += m[xx10];
2405.
2406.
2407.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2408.    udotysum [xx10] = udoty [xx10][contactpnt10];
2409.    resultudoty [xx10] += udotysum [xx10];
2410.
2411.
2412.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2413.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2414.    resultudotx [xx10] += udotxsum [xx10];
2415.
2416.
2417.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2418.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2419.    resultthetadot [xx10] += thetadotsum [xx10];
2420.
2421.
2422.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2423.    deluysum [xx10] = deluy [xx10][contactpnt10];
2424.    resultdeluy [xx10] += deluysum [xx10];
2425.
2426.
2427.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2428.    deluxsum [xx10] = delux [xx10][contactpnt10];
2429.    resultdelux [xx10] += deluxsum [xx10];
2430.
2431.
2432.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2433.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2434.    resultdeltheta [xx10] += delthetasum [xx10];
2435.
2436.
2437.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2438.    uysum [xx10] = uy [xx10][contactpnt10];
2439.    resultuy [xx10]+= uysum [xx10];
```

```
2440.
2441.
2442.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2443.    uxsum [xx10] = ux [xx10][contactpnt10];
2444.    resultux [xx10] += uxsum [xx10];
2445.
2446.
2447.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2448.    thetasum [xx10] = theta [xx10][contactpnt10];
2449.    resulttheta [xx10] += thetasum [xx10];
2450.    alpha [xx10] = alpha1 + resulttheta [xx10];
2451.
2452.    }
2453.    }
2454.
2455.    }
2456.
2457.    next10:
2458.    ;}
2459.
2460.
2461.    int xx11 = 0, qq11 = 0, contactpnt11 = 0, l11, i11;
2462.
2463.
2464.    for (i11 = 0; i11 < isobox11; i11= i11+1)
2465.
2466.    {
2467.
2468.    xx11 = isocounterbox11[i11];
2469.
2470.    rightisosceles [xx11][0][0] = rightisosceles [xx11][0][0] + resultux
    [xx11] ;
2471.
2472.    rightisosceles [xx11][0][1] = rightisosceles [xx11][0][1] + resultuy
    [xx11] ;
2473.
2474.
2475.
2476.    for ( l11 = 0; l11 < isobox11; l11 = l11+1)
2477.
2478.    {
2479.
2480.
2481.    qq11 = isocounterbox11[l11];
2482.
2483.
2484.
2485.    if ( ((pow((rightisosceles [xx11][0][0] -
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) > 0) && ((pow((rightisosceles [xx11][0]
   [0] -
```

```
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) <= 50))
2486.
2487.    {
2488.
2489.    contactpnt11 = contactpnt11 + 1;
2490.
2491.    if (contactpnt11 >= 4) {goto next11;}
2492.
2493.
2494.    if ( rightisosceles [qq11][3][1] >  rightisosceles [xx11][1][1] )
2495.
2496.    {
2497.
2498.
2499.    ydisplat [xx11][contactpnt11] = ydisplacement2  + (deluy [xx11][cont
   actpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][0] -
     rightisosceles [qq11][0][0]));
2500.    xdisplat [xx11][contactpnt11] = xdisplacement2 + (delux [xx11][conta
   ctpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][1] -
   rightisosceles [qq11][0][1]));
2501.
2502.
2503.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2504.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2505.
2506.
2507.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
2508.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2509.
2510.
2511.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2512.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2513.
2514.
2515.    yforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * sin (alpha1)) -
     ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha1)));
```

```
2516.    xforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * cos (alpha1)) + ((fn [xx11][contactpnt11] + dn [xx11][
   contactpnt11]) * sin (alpha1)));
2517.
2518.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2519.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2520.
2521.
2522.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2523.    y [xx11] = fysum [xx11][contactpnt11];
2524.    resultfy [xx11] += y[xx11];
2525.
2526.
2527.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2528.    x [xx11] = fxsum [xx11][contactpnt11];
2529.    resultfx [xx11] += x[xx11];
2530.
2531.
2532.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][3][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][3][1] -
    rightisosceles [xx11][0][1]));
2533.    m [xx11] = msum [xx11][contactpnt11];
2534.    resultm [xx11] += m[xx11];
2535.
2536.
2537.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement) *1000/ dmass); // mm/sec
2538.    udotysum [xx11] = udoty [xx11][contactpnt11];
2539.    resultudoty [xx11] += udotysum [xx11];
2540.
2541.
2542.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2543.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2544.    resultudotx [xx11] += udotxsum [xx11];
2545.
2546.
2547.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2548.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
2549.    resultthetadot [xx11] += thetadotsum [xx11];
2550.
2551.
2552.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2553.    deluysum [xx11] = deluy [xx11][contactpnt11];
2554.    resultdeluy [xx11] += deluysum [xx11];
2555.
2556.
2557.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
```

```
2558.    deluxsum [xx11] = delux [xx11][contactpnt11];
2559.    resultdelux [xx11] += deluxsum [xx11];
2560.
2561.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2562.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2563.    resultdeltheta [xx11] += delthetasum [xx11];
2564.
2565.
2566.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2567.    uysum [xx11] = uy [xx11][contactpnt11];
2568.    resultuy [xx11]+= uysum [xx11];
2569.
2570.
2571.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2572.    uxsum [xx11] = ux [xx11][contactpnt11];
2573.    resultux [xx11] += uxsum [xx11];
2574.
2575.
2576.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2577.    thetasum [xx11] = theta [xx11][contactpnt11];
2578.    resulttheta [xx11] += thetasum [xx11];
2579.    alpha [xx11] = alpha1 + resulttheta [xx11];
2580.
2581.    }
2582.
2583.    else
2584.
2585.    {
2586.
2587.
2588.
2589.    ydisplat [xx11][contactpnt11] += ydisplacement2  + (deluy [xx11][con
   tactpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][0][0] -
     rightisosceles [qq11][0][0]));
2590.    xdisplat [xx11][contactpnt11] += xdisplacement2 + (delux [xx11][cont
   actpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][1][1] -
   rightisosceles [qq11][0][1]));
2591.
2592.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2593.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2594.
2595.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
2596.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2597.
2598.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2599.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2600.
2601.    yforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * sin (alpha [xx11])) -
    ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha [xx11])
   ));
2602.    xforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * cos (alpha [xx11])) + ((fn [xx11][contactpnt11] + dn [x
   x11][contactpnt11]) * sin (alpha [xx11])));
2603.
2604.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2605.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2606.
2607.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2608.    x [xx11] = fxsum [xx11][contactpnt11];
2609.    resultfx [xx11] += x[xx11];
2610.
2611.
2612.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2613.    y [xx11] = fysum [xx11][contactpnt11];
2614.    resultfy [xx11] += y[xx11];
2615.
2616.
2617.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][1][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][1][1] -
    rightisosceles [xx11][0][1]));
2618.    m [xx11] = msum [xx11][contactpnt11];
2619.    resultm [xx11] += m[xx11];
2620.
2621.
2622.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2623.    udotysum [xx11] = udoty [xx11][contactpnt11];
2624.    resultudoty [xx11] += udotysum [xx11];
2625.
2626.
2627.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2628.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2629.    resultudotx [xx11] += udotxsum [xx11];
2630.
2631.
2632.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2633.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
```

```
2634.    resultthetadot [xx11] += thetadotsum [xx11];
2635.
2636.
2637.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2638.    deluysum [xx11] = deluy [xx11][contactpnt11];
2639.    resultdeluy [xx11] += deluysum [xx11];
2640.
2641.
2642.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
2643.    deluxsum [xx11] = delux [xx11][contactpnt11];
2644.    resultdelux [xx11] += deluxsum [xx11];
2645.
2646.
2647.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2648.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2649.    resultdeltheta [xx11] += delthetasum [xx11];
2650.
2651.
2652.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2653.    uysum [xx11] = uy [xx11][contactpnt11];
2654.    resultuy [xx11]+= uysum [xx11];
2655.
2656.
2657.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2658.    uxsum [xx11] = ux [xx11][contactpnt11];
2659.    resultux [xx11] += uxsum [xx11];
2660.
2661.
2662.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2663.    thetasum [xx11] = theta [xx11][contactpnt11];
2664.    resulttheta [xx11] += thetasum [xx11];
2665.    alpha [xx11] = alpha1 + resulttheta [xx11];
2666.
2667.
2668.
2669.    }
2670.
2671.    }
2672.
2673.    }
2674.
2675.    next11:
2676.    ;}
2677.
2678.
2679.
2680.    int xx12 = 0, qq12 = 0, contactpnt12 = 0, l12, i12;
2681.
2682.
```

```
2683.    for (i12 = 0; i12 < isobox12; i12= i12+1)
2684.
2685.    {
2686.
2687.    xx12 = isocounterbox12[i12];
2688.
2689.    rightisosceles [xx12][0][0] = rightisosceles [xx12][0][0] + resultux
    [xx12] ;
2690.
2691.    rightisosceles [xx12][0][1] = rightisosceles [xx12][0][1] + resultuy
    [xx12] ;
2692.
2693.
2694.
2695.    for ( l12 = 0; l12 < isobox12; l12 = l12+1)
2696.
2697.    {
2698.
2699.
2700.    qq12 = isocounterbox12[l12];
2701.
2702.
2703.
2704.    if ( ((pow((rightisosceles [xx12][0][0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) > 0) && ((pow((rightisosceles [xx12][0]
   [0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) <= 50))
2705.
2706.    {
2707.
2708.    contactpnt12 = contactpnt12 + 1;
2709.
2710.    if (contactpnt12 >= 4) {goto next12;}
2711.
2712.
2713.    if ( rightisosceles [qq12][3][1] >  rightisosceles [xx12][1][1] )
2714.
2715.    {
2716.
2717.
2718.    ydisplat [xx12][contactpnt12] = ydisplacement2  + (deluy [xx12][cont
   actpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][0] -
    rightisosceles [qq12][0][0]));
2719.    xdisplat [xx12][contactpnt12] = xdisplacement2 + (delux [xx12][conta
   ctpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][1] - rightisosceles [xx12][0][1])  -
```

```
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][1] -
   rightisosceles [qq12][0][1]));
2720.
2721.
2722.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2723.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2724.
2725.
2726.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2727.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2728.
2729.
2730.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2731.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2732.
2733.
2734.    yforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * sin (alpha1)) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha1)));
2735.    xforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * cos (alpha1)) + ((fn [xx12][contactpnt12] + dn [xx12][
   contactpnt12]) * sin (alpha1)));
2736.
2737.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2738.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2739.
2740.
2741.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2742.    y [xx12] = fysum [xx12][contactpnt12];
2743.    resultfy [xx12] += y[xx12];
2744.
2745.
2746.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2747.    x [xx12] = fxsum [xx12][contactpnt12];
2748.    resultfx [xx12] += x[xx12];
2749.
2750.
2751.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][3][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][3][1] -
    rightisosceles [xx12][0][1]));
2752.    m [xx12] = msum [xx12][contactpnt12];
2753.    resultm [xx12] += m[xx12];
2754.
2755.
```

```
2756.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement) *1000/ dmass); // mm/sec
2757.    udotysum [xx12] = udoty [xx12][contactpnt12];
2758.    resultudoty [xx12] += udotysum [xx12];
2759.
2760.
2761.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2762.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2763.    resultudotx [xx12] += udotxsum [xx12];
2764.
2765.
2766.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2767.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2768.    resultthetadot [xx12] += thetadotsum [xx12];
2769.
2770.
2771.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2772.    deluysum [xx12] = deluy [xx12][contactpnt12];
2773.    resultdeluy [xx12] += deluysum [xx12];
2774.
2775.
2776.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2777.    deluxsum [xx12] = delux [xx12][contactpnt12];
2778.    resultdelux [xx12] += deluxsum [xx12];
2779.
2780.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2781.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2782.    resultdeltheta [xx12] += delthetasum [xx12];
2783.
2784.
2785.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2786.    uysum [xx12] = uy [xx12][contactpnt12];
2787.    resultuy [xx12]+= uysum [xx12];
2788.
2789.
2790.    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
2791.    uxsum [xx12] = ux [xx12][contactpnt12];
2792.    resultux [xx12] += uxsum [xx12];
2793.
2794.
2795.    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
2796.    thetasum [xx12] = theta [xx12][contactpnt12];
2797.    resultttheta [xx12] += thetasum [xx12];
2798.    alpha [xx12] = alpha1 + resultttheta [xx12];
2799.
2800.    }
2801.
```

```
2802.    else
2803.
2804.    {
2805.
2806.
2807.
2808.    ydisplat [xx12][contactpnt12] += ydisplacement2  + (deluy [xx12][con
   tactpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][0][0] -
    rightisosceles [qq12][0][0]));
2809.    xdisplat [xx12][contactpnt12] += xdisplacement2 + (delux [xx12][cont
   actpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][1] - rightisosceles [xx12][0][1])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][1][1] -
   rightisosceles [qq12][0][1]));
2810.
2811.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2812.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2813.
2814.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2815.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2816.
2817.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2818.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2819.
2820.    yforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * sin (alpha [xx12])) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha [xx12])
   ));
2821.    xforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * cos (alpha [xx12])) + ((fn [xx12][contactpnt12] + dn [x
   x12][contactpnt12]) * sin (alpha [xx12])));
2822.
2823.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2824.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2825.
2826.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2827.    x [xx12] = fxsum [xx12][contactpnt12];
2828.    resultfx [xx12] += x[xx12];
2829.
2830.
2831.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2832.    y [xx12] = fysum [xx12][contactpnt12];
```

```
2833.    resultfy [xx12] += y[xx12];
2834.
2835.
2836.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][1][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][1][1] -
    rightisosceles [xx12][0][1]));
2837.    m [xx12] = msum [xx12][contactpnt12];
2838.    resultm [xx12] += m[xx12];
2839.
2840.
2841.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2842.    udotysum [xx12] = udoty [xx12][contactpnt12];
2843.    resultudoty [xx12] += udotysum [xx12];
2844.
2845.
2846.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2847.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2848.    resultudotx [xx12] += udotxsum [xx12];
2849.
2850.
2851.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2852.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2853.    resultthetadot [xx12] += thetadotsum [xx12];
2854.
2855.
2856.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2857.    deluysum [xx12] = deluy [xx12][contactpnt12];
2858.    resultdeluy [xx12] += deluysum [xx12];
2859.
2860.
2861.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2862.    deluxsum [xx12] = delux [xx12][contactpnt12];
2863.    resultdelux [xx12] += deluxsum [xx12];
2864.
2865.
2866.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2867.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2868.    resultdeltheta [xx12] += delthetasum [xx12];
2869.
2870.
2871.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2872.    uysum [xx12] = uy [xx12][contactpnt12];
2873.    resultuy [xx12]+= uysum [xx12];
2874.
2875.
```

```
2876.    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
2877.    uxsum [xx12] = ux [xx12][contactpnt12];
2878.    resultux [xx12] += uxsum [xx12];
2879.
2880.
2881.    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
2882.    thetasum [xx12] = theta [xx12][contactpnt12];
2883.    resulttheta [xx12] += thetasum [xx12];
2884.    alpha [xx12] = alpha1 + resulttheta [xx12];
2885.
2886.    }
2887.    }
2888.
2889.    }
2890.
2891.    next12:
2892.    ;}
2893.
2894.
2895.
2896.    int xx13 = 0, qq13 = 0, contactpnt13 = 0, l13, i13;
2897.
2898.
2899.    for (i13 = 0; i13 < isobox13; i13= i13+1)
2900.
2901.    {
2902.
2903.    xx13 = isocounterbox13[i13];
2904.
2905.
2906.
2907.    rightisosceles [xx13][0][0] = rightisosceles [xx13][0][0] + resultux
    [xx13] ;
2908.
2909.    rightisosceles [xx13][0][1] = rightisosceles [xx13][0][1] + resultuy
    [xx13] ;
2910.
2911.
2912.
2913.    for ( l13 = 0; l13 < isobox13; l13 = l13+1)
2914.
2915.    {
2916.
2917.
2918.    qq13 = isocounterbox13[l13];
2919.
2920.
2921.
2922.    if ( ((pow((rightisosceles [xx13][0][0] -
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) > 0) && ((pow((rightisosceles [xx13][0]
   [0] -
```

```
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) <= 50))
2923.
2924.    {
2925.
2926.    contactpnt13 = contactpnt13 + 1;
2927.
2928.    if (contactpnt13 >= 4) {goto next13;}
2929.
2930.
2931.    if ( rightisosceles [qq13][3][1] >  rightisosceles [xx13][1][1] )
2932.
2933.    {
2934.
2935.
2936.    ydisplat [xx13][contactpnt13] = ydisplacement2  + (deluy [xx13][cont
   actpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][0] -
    rightisosceles [qq13][0][0]));
2937.    xdisplat [xx13][contactpnt13] = xdisplacement2 + (delux [xx13][conta
   ctpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][1] -
   rightisosceles [qq13][0][1]));
2938.
2939.
2940.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
2941.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
2942.
2943.
2944.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
2945.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
2946.
2947.
2948.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
2949.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
2950.
2951.
2952.    yforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * sin (alpha1)) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha1)));
```

```
2953.    xforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * cos (alpha1)) + ((fn [xx13][contactpnt13] + dn [xx13][
   contactpnt13]) * sin (alpha1)));
2954.
2955.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
2956.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
2957.
2958.
2959.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
2960.    y [xx13] = fysum [xx13][contactpnt13];
2961.    resultfy [xx13] += y[xx13];
2962.
2963.
2964.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
2965.    x [xx13] = fxsum [xx13][contactpnt13];
2966.    resultfx [xx13] += x[xx13];
2967.
2968.
2969.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][3][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][3][1] -
    rightisosceles [xx13][0][1]));
2970.    m [xx13] = msum [xx13][contactpnt13];
2971.    resultm [xx13] += m[xx13];
2972.
2973.
2974.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement) *1000/ dmass); // mm/sec
2975.    udotysum [xx13] = udoty [xx13][contactpnt13];
2976.    resultudoty [xx13] += udotysum [xx13];
2977.
2978.
2979.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
2980.    udotxsum [xx13] = udotx [xx13][contactpnt13];
2981.    resultudotx [xx13] += udotxsum [xx13];
2982.
2983.
2984.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
2985.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
2986.    resultthetadot [xx13] += thetadotsum [xx13];
2987.
2988.
2989.    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
   ement;
2990.    deluysum [xx13] = deluy [xx13][contactpnt13];
2991.    resultdeluy [xx13] += deluysum [xx13];
2992.
2993.
2994.    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
   ement;
```

```
2995.    deluxsum [xx13] = delux [xx13][contactpnt13];
2996.    resultdelux [xx13] += deluxsum [xx13];
2997.
2998.    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
   e_increment;
2999.    delthetasum [xx13] = deltheta [xx13][contactpnt13];
3000.    resultdeltheta [xx13] += delthetasum [xx13];
3001.
3002.
3003.    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
3004.    uysum [xx13] = uy [xx13][contactpnt13];
3005.    resultuy [xx13]+= uysum [xx13];
3006.
3007.
3008.    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
3009.    uxsum [xx13] = ux [xx13][contactpnt13];
3010.    resultux [xx13] += uxsum [xx13];
3011.
3012.
3013.    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
3014.    thetasum [xx13] = theta [xx13][contactpnt13];
3015.    resulttheta [xx13] += thetasum [xx13];
3016.    alpha [xx13] = alpha1 + resulttheta [xx13];
3017.
3018.    }
3019.
3020.    else
3021.
3022.    {
3023.
3024.
3025.
3026.    ydisplat [xx13][contactpnt13] += ydisplacement2  + (deluy [xx13][con
   tactpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][0][0] -
    rightisosceles [qq13][0][0]));
3027.    xdisplat [xx13][contactpnt13] += xdisplacement2 + (delux [xx13][cont
   actpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][1][1] -
   rightisosceles [qq13][0][1]));
3028.
3029.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
3030.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
3031.
3032.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
3033.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
3034.
3035.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
3036.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
3037.
3038.    yforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * sin (alpha [xx13])) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha [xx13])
   ));
3039.    xforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * cos (alpha [xx13])) + ((fn [xx13][contactpnt13] + dn [x
   x13][contactpnt13]) * sin (alpha [xx13])));
3040.
3041.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
3042.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
3043.
3044.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
3045.    x [xx13] = fxsum [xx13][contactpnt13];
3046.    resultfx [xx13] += x[xx13];
3047.
3048.
3049.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
3050.    y [xx13] = fysum [xx13][contactpnt13];
3051.    resultfy [xx13] += y[xx13];
3052.
3053.
3054.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][1][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][1][1] -
    rightisosceles [xx13][0][1]));
3055.    m [xx13] = msum [xx13][contactpnt13];
3056.    resultm [xx13] += m[xx13];
3057.
3058.
3059.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3060.    udotysum [xx13] = udoty [xx13][contactpnt13];
3061.    resultudoty [xx13] += udotysum [xx13];
3062.
3063.
3064.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
3065.    udotxsum [xx13] = udotx [xx13][contactpnt13];
3066.    resultudotx [xx13] += udotxsum [xx13];
3067.
3068.
3069.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
3070.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
```

```
3071.    resultthetadot [xx13] += thetadotsum [xx13];
3072.
3073.
3074.    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
   ement;
3075.    deluysum [xx13] = deluy [xx13][contactpnt13];
3076.    resultdeluy [xx13] += deluysum [xx13];
3077.
3078.
3079.    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
   ement;
3080.    deluxsum [xx13] = delux [xx13][contactpnt13];
3081.    resultdelux [xx13] += deluxsum [xx13];
3082.
3083.
3084.    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
   e_increment;
3085.    delthetasum [xx13] = deltheta [xx13][contactpnt13];
3086.    resultdeltheta [xx13] += delthetasum [xx13];
3087.
3088.
3089.    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
3090.    uysum [xx13] = uy [xx13][contactpnt13];
3091.    resultuy [xx13]+= uysum [xx13];
3092.
3093.
3094.    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
3095.    uxsum [xx13] = ux [xx13][contactpnt13];
3096.    resultux [xx13] += uxsum [xx13];
3097.
3098.
3099.    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
3100.    thetasum [xx13] = theta [xx13][contactpnt13];
3101.    resulttheta [xx13] += thetasum [xx13];
3102.    alpha [xx13] = alpha1 + resulttheta [xx13];
3103.
3104.
3105.
3106.    }
3107.
3108.    }
3109.
3110.    }
3111.
3112.    next13:
3113.    ;}
3114.
3115.
3116.
3117.    int xx14 = 0, qq14 = 0, contactpnt14 = 0, l14, i14;
3118.
3119.
```

```
3120.    for (i14 = 0; i14 < isobox14; i14= i14+1)
3121.
3122.    {
3123.
3124.    xx14 = isocounterbox14[i14];
3125.
3126.    rightisosceles [xx14][0][0] = rightisosceles [xx14][0][0] + resultux
    [xx14] ;
3127.
3128.    rightisosceles [xx14][0][1] = rightisosceles [xx14][0][1] + resultuy
    [xx14] ;
3129.
3130.
3131.
3132.    for ( l14 = 0; l14 < isobox14; l14 = l14+1)
3133.
3134.    {
3135.
3136.
3137.    qq14 = isocounterbox14[l14];
3138.
3139.
3140.
3141.    if ( ((pow((rightisosceles [xx14][0][0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) > 0) && ((pow((rightisosceles [xx14][0]
   [0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) <= 50))
3142.
3143.    {
3144.
3145.    contactpnt14 = contactpnt14 + 1;
3146.
3147.    if (contactpnt14 >= 4) {goto next14;}
3148.
3149.
3150.    if ( rightisosceles [qq14][3][1] >  rightisosceles [xx14][1][1] )
3151.
3152.    {
3153.
3154.
3155.    ydisplat [xx14][contactpnt14] = ydisplacement2  + (deluy [xx14][cont
   actpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][0] -
    rightisosceles [qq14][0][0]));
3156.    xdisplat [xx14][contactpnt14] = xdisplacement2 + (delux [xx14][conta
   ctpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][1] - rightisosceles [xx14][0][1])  -
```

```
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][1] -
   rightisosceles [qq14][0][1]));
3157.
3158.
3159.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3160.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3161.
3162.
3163.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3164.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3165.
3166.
3167.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3168.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3169.
3170.
3171.    yforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * sin (alpha1)) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha1)));
3172.    xforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * cos (alpha1)) + ((fn [xx14][contactpnt14] + dn [xx14][
   contactpnt14]) * sin (alpha1)));
3173.
3174.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3175.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3176.
3177.
3178.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3179.    y [xx14] = fysum [xx14][contactpnt14];
3180.    resultfy [xx14] += y[xx14];
3181.
3182.
3183.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3184.    x [xx14] = fxsum [xx14][contactpnt14];
3185.    resultfx [xx14] += x[xx14];
3186.
3187.
3188.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][3][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][3][1] -
    rightisosceles [xx14][0][1]));
3189.    m [xx14] = msum [xx14][contactpnt14];
3190.    resultm [xx14] += m[xx14];
3191.
3192.
```

```
3193.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement) *1000/ dmass); // mm/sec
3194.    udotysum [xx14] = udoty [xx14][contactpnt14];
3195.    resultudoty [xx14] += udotysum [xx14];
3196.
3197.
3198.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3199.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3200.    resultudotx [xx14] += udotxsum [xx14];
3201.
3202.
3203.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3204.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3205.    resultthetadot [xx14] += thetadotsum [xx14];
3206.
3207.
3208.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3209.    deluysum [xx14] = deluy [xx14][contactpnt14];
3210.    resultdeluy [xx14] += deluysum [xx14];
3211.
3212.
3213.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3214.    deluxsum [xx14] = delux [xx14][contactpnt14];
3215.    resultdelux [xx14] += deluxsum [xx14];
3216.
3217.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3218.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3219.    resultdeltheta [xx14] += delthetasum [xx14];
3220.
3221.
3222.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3223.    uysum [xx14] = uy [xx14][contactpnt14];
3224.    resultuy [xx14]+= uysum [xx14];
3225.
3226.
3227.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3228.    uxsum [xx14] = ux [xx14][contactpnt14];
3229.    resultux [xx14] += uxsum [xx14];
3230.
3231.
3232.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3233.    thetasum [xx14] = theta [xx14][contactpnt14];
3234.    resultttheta [xx14] += thetasum [xx14];
3235.    alpha [xx14] = alpha1 + resultttheta [xx14];
3236.
3237.    }
3238.
```

```
3239.    else
3240.
3241.    {
3242.
3243.
3244.
3245.    ydisplat [xx14][contactpnt14] += ydisplacement2  + (deluy [xx14][con
   tactpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][0][0] -
    rightisosceles [qq14][0][0]));
3246.    xdisplat [xx14][contactpnt14] += xdisplacement2 + (delux [xx14][cont
   actpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][1] - rightisosceles [xx14][0][1])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][1][1] -
   rightisosceles [qq14][0][1]));
3247.
3248.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3249.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3250.
3251.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3252.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3253.
3254.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3255.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3256.
3257.    yforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * sin (alpha [xx14])) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha [xx14])
   ));
3258.    xforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * cos (alpha [xx14])) + ((fn [xx14][contactpnt14] + dn [x
   x14][contactpnt14]) * sin (alpha [xx14])));
3259.
3260.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3261.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3262.
3263.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3264.    x [xx14] = fxsum [xx14][contactpnt14];
3265.    resultfx [xx14] += x[xx14];
3266.
3267.
3268.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3269.    y [xx14] = fysum [xx14][contactpnt14];
```

```
3270.    resultfy [xx14] += y[xx14];
3271.
3272.
3273.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][1][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][1][1] -
    rightisosceles [xx14][0][1]));
3274.    m [xx14] = msum [xx14][contactpnt14];
3275.    resultm [xx14] += m[xx14];
3276.
3277.
3278.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3279.    udotysum [xx14] = udoty [xx14][contactpnt14];
3280.    resultudoty [xx14] += udotysum [xx14];
3281.
3282.
3283.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3284.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3285.    resultudotx [xx14] += udotxsum [xx14];
3286.
3287.
3288.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3289.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3290.    resultthetadot [xx14] += thetadotsum [xx14];
3291.
3292.
3293.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3294.    deluysum [xx14] = deluy [xx14][contactpnt14];
3295.    resultdeluy [xx14] += deluysum [xx14];
3296.
3297.
3298.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3299.    deluxsum [xx14] = delux [xx14][contactpnt14];
3300.    resultdelux [xx14] += deluxsum [xx14];
3301.
3302.
3303.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3304.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3305.    resultdeltheta [xx14] += delthetasum [xx14];
3306.
3307.
3308.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3309.    uysum [xx14] = uy [xx14][contactpnt14];
3310.    resultuy [xx14]+= uysum [xx14];
3311.
3312.
```

```
3313.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3314.    uxsum [xx14] = ux [xx14][contactpnt14];
3315.    resultux [xx14] += uxsum [xx14];
3316.
3317.
3318.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3319.    thetasum [xx14] = theta [xx14][contactpnt14];
3320.    resulttheta [xx14] += thetasum [xx14];
3321.    alpha [xx14] = alpha1 + resulttheta [xx14];
3322.
3323.    }
3324.    }
3325.
3326.    }
3327.
3328.    next14:
3329.    ;}
3330.
3331.
3332.
3333.    int xx15 = 0, qq15 = 0, contactpnt15 = 0, l15, i15;
3334.
3335.
3336.    for (i15 = 0; i15 < isobox15; i15= i15+1)
3337.
3338.    {
3339.
3340.    xx15 = isocounterbox15[i15];
3341.
3342.    rightisosceles [xx15][0][0] = rightisosceles [xx15][0][0] + resultux
    [xx15] ;
3343.
3344.    rightisosceles [xx15][0][1] = rightisosceles [xx15][0][1] + resultuy
    [xx15] ;
3345.
3346.
3347.
3348.    for ( l15 = 0; l15 < isobox15; l15 = l15+1)
3349.
3350.    {
3351.
3352.
3353.    qq15 = isocounterbox15[l15];
3354.
3355.
3356.
3357.    if ( ((pow((rightisosceles [xx15][0][0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) > 0) && ((pow((rightisosceles [xx15][0]
   [0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) <= 50))
```

```
3358.
3359.    {
3360.
3361.    contactpnt15 = contactpnt15 + 1;
3362.
3363.    if (contactpnt15 >= 4) {goto next15;}
3364.
3365.
3366.    if ( rightisosceles [qq15][3][1] >  rightisosceles [xx15][1][1] )
3367.
3368.    {
3369.
3370.
3371.    ydisplat [xx15][contactpnt15] = ydisplacement2  + (deluy [xx15][cont
   actpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][0] -
    rightisosceles [qq15][0][0]));
3372.    xdisplat [xx15][contactpnt15] = xdisplacement2 + (delux [xx15][conta
   ctpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][1] -
   rightisosceles [qq15][0][1]));
3373.
3374.
3375.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3376.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3377.
3378.
3379.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3380.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3381.
3382.
3383.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3384.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3385.
3386.
3387.    yforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * sin (alpha1)) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha1)));
3388.    xforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * cos (alpha1)) + ((fn [xx15][contactpnt15] + dn [xx15][
   contactpnt15]) * sin (alpha1)));
3389.
```

```
3390.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3391.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3392.
3393.
3394.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3395.    y [xx15] = fysum [xx15][contactpnt15];
3396.    resultfy [xx15] += y[xx15];
3397.
3398.
3399.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3400.    x [xx15] = fxsum [xx15][contactpnt15];
3401.    resultfx [xx15] += x[xx15];
3402.
3403.
3404.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][3][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][3][1] -
    rightisosceles [xx15][0][1]));
3405.    m [xx15] = msum [xx15][contactpnt15];
3406.    resultm [xx15] += m[xx15];
3407.
3408.
3409.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement) *1000/ dmass); // mm/sec
3410.    udotysum [xx15] = udoty [xx15][contactpnt15];
3411.    resultudoty [xx15] += udotysum [xx15];
3412.
3413.
3414.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3415.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3416.    resultudotx [xx15] += udotxsum [xx15];
3417.
3418.
3419.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3420.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3421.    resultthetadot [xx15] += thetadotsum [xx15];
3422.
3423.
3424.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3425.    deluysum [xx15] = deluy [xx15][contactpnt15];
3426.    resultdeluy [xx15] += deluysum [xx15];
3427.
3428.
3429.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3430.    deluxsum [xx15] = delux [xx15][contactpnt15];
3431.    resultdelux [xx15] += deluxsum [xx15];
3432.
```

```
3433.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3434.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3435.    resultdeltheta [xx15] += delthetasum [xx15];
3436.
3437.
3438.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3439.    uysum [xx15] = uy [xx15][contactpnt15];
3440.    resultuy [xx15]+= uysum [xx15];
3441.
3442.
3443.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3444.    uxsum [xx15] = ux [xx15][contactpnt15];
3445.    resultux [xx15] += uxsum [xx15];
3446.
3447.
3448.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3449.    thetasum [xx15] = theta [xx15][contactpnt15];
3450.    resulttheta [xx15] += thetasum [xx15];
3451.    alpha [xx15] = alpha1 + resulttheta [xx15];
3452.
3453.    }
3454.
3455.    else
3456.
3457.    {
3458.
3459.
3460.
3461.    ydisplat [xx15][contactpnt15] += ydisplacement2  + (deluy [xx15][con
   tactpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][0][0] -
    rightisosceles [qq15][0][0]));
3462.    xdisplat [xx15][contactpnt15] += xdisplacement2 + (delux [xx15][cont
   actpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][1][1] -
   rightisosceles [qq15][0][1]));
3463.
3464.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3465.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3466.
3467.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3468.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3469.
```

```
3470.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3471.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3472.
3473.    yforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * sin (alpha [xx15])) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha [xx15])
   ));
3474.    xforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * cos (alpha [xx15])) + ((fn [xx15][contactpnt15] + dn [x
   x15][contactpnt15]) * sin (alpha [xx15])));
3475.
3476.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3477.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3478.
3479.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3480.    x [xx15] = fxsum [xx15][contactpnt15];
3481.    resultfx [xx15] += x[xx15];
3482.
3483.
3484.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3485.    y [xx15] = fysum [xx15][contactpnt15];
3486.    resultfy [xx15] += y[xx15];
3487.
3488.
3489.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][1][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][1][1] -
    rightisosceles [xx15][0][1]));
3490.    m [xx15] = msum [xx15][contactpnt15];
3491.    resultm [xx15] += m[xx15];
3492.
3493.
3494.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3495.    udotysum [xx15] = udoty [xx15][contactpnt15];
3496.    resultudoty [xx15] += udotysum [xx15];
3497.
3498.
3499.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3500.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3501.    resultudotx [xx15] += udotxsum [xx15];
3502.
3503.
3504.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3505.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3506.    resultthetadot [xx15] += thetadotsum [xx15];
3507.
3508.
```

```
3509.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3510.    deluysum [xx15] = deluy [xx15][contactpnt15];
3511.    resultdeluy [xx15] += deluysum [xx15];
3512.
3513.
3514.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3515.    deluxsum [xx15] = delux [xx15][contactpnt15];
3516.    resultdelux [xx15] += deluxsum [xx15];
3517.
3518.
3519.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3520.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3521.    resultdeltheta [xx15] += delthetasum [xx15];
3522.
3523.
3524.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3525.    uysum [xx15] = uy [xx15][contactpnt15];
3526.    resultuy [xx15]+= uysum [xx15];
3527.
3528.
3529.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3530.    uxsum [xx15] = ux [xx15][contactpnt15];
3531.    resultux [xx15] += uxsum [xx15];
3532.
3533.
3534.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3535.    thetasum [xx15] = theta [xx15][contactpnt15];
3536.    resulttheta [xx15] += thetasum [xx15];
3537.    alpha [xx15] = alpha1 + resulttheta [xx15];
3538.
3539.    }
3540.    }
3541.
3542.    }
3543.
3544.    next15:
3545.    ;}
3546.
3547.
3548.
3549.    int xx16 = 0, qq16 = 0, contactpnt16 = 0, l16, i16;
3550.
3551.
3552.    for (i16 = 0; i16 < isobox16; i16= i16+1)
3553.
3554.    {
3555.
3556.    xx16 = isocounterbox16[i16];
3557.
```

```
3558.
3559.
3560.    rightisosceles [xx16][0][0] = rightisosceles [xx16][0][0] + resultux
    [xx16] ;
3561.
3562.    rightisosceles [xx16][0][1] = rightisosceles [xx16][0][1] + resultuy
    [xx16] ;
3563.
3564.
3565.
3566.    for ( l16 = 0; l16 < isobox16; l16 = l16+1)
3567.
3568.    {
3569.
3570.
3571.    qq16 = isocounterbox16[l16];
3572.
3573.
3574.
3575.    if ( ((pow((rightisosceles [xx16][0][0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) > 0) && ((pow((rightisosceles [xx16][0]
   [0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) <= 50))
3576.
3577.    {
3578.
3579.    contactpnt16 = contactpnt16 + 1;
3580.
3581.    if (contactpnt16 >= 4) {goto next16;}
3582.
3583.
3584.    if ( rightisosceles [qq16][3][1] >  rightisosceles [xx16][1][1] )
3585.
3586.    {
3587.
3588.
3589.    ydisplat [xx16][contactpnt16] = ydisplacement2  + (deluy [xx16][cont
   actpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][0] -
    rightisosceles [qq16][0][0]));
3590.    xdisplat [xx16][contactpnt16] = xdisplacement2 + (delux [xx16][conta
   ctpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][1] -
   rightisosceles [qq16][0][1]));
3591.
3592.
```

```
3593.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3594.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3595.
3596.
3597.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3598.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3599.
3600.
3601.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3602.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3603.
3604.
3605.    yforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * sin (alpha1)) -
     ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha1)));
3606.    xforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * cos (alpha1)) + ((fn [xx16][contactpnt16] + dn [xx16][
   contactpnt16]) * sin (alpha1)));
3607.
3608.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3609.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3610.
3611.
3612.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3613.    y [xx16] = fysum [xx16][contactpnt16];
3614.    resultfy [xx16] += y[xx16];
3615.
3616.
3617.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3618.    x [xx16] = fxsum [xx16][contactpnt16];
3619.    resultfx [xx16] += x[xx16];
3620.
3621.
3622.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][3][0] - rightisosceles [xx16][0][0]) -
     xforce [xx16][contactpnt16] * ( rightisosceles [qq16][3][1] -
     rightisosceles [xx16][0][1]));
3623.    m [xx16] = msum [xx16][contactpnt16];
3624.    resultm [xx16] += m[xx16];
3625.
3626.
3627.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement) *1000/ dmass); // mm/sec
3628.    udotysum [xx16] = udoty [xx16][contactpnt16];
3629.    resultudoty [xx16] += udotysum [xx16];
3630.
```

```
3631.
3632.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3633.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3634.    resultudotx [xx16] += udotxsum [xx16];
3635.
3636.
3637.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3638.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3639.    resultthetadot [xx16] += thetadotsum [xx16];
3640.
3641.
3642.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3643.    deluysum [xx16] = deluy [xx16][contactpnt16];
3644.    resultdeluy [xx16] += deluysum [xx16];
3645.
3646.
3647.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3648.    deluxsum [xx16] = delux [xx16][contactpnt16];
3649.    resultdelux [xx16] += deluxsum [xx16];
3650.
3651.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3652.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3653.    resultdeltheta [xx16] += delthetasum [xx16];
3654.
3655.
3656.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3657.    uysum [xx16] = uy [xx16][contactpnt16];
3658.    resultuy [xx16]+= uysum [xx16];
3659.
3660.
3661.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3662.    uxsum [xx16] = ux [xx16][contactpnt16];
3663.    resultux [xx16] += uxsum [xx16];
3664.
3665.
3666.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3667.    thetasum [xx16] = theta [xx16][contactpnt16];
3668.    resultttheta [xx16] += thetasum [xx16];
3669.    alpha [xx16] = alpha1 + resultttheta [xx16];
3670.
3671.    }
3672.
3673.    else
3674.
3675.    {
3676.
3677.
```

```
3678.
3679.    ydisplat [xx16][contactpnt16] += ydisplacement2  + (deluy [xx16][con
   tactpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][0][0] -
    rightisosceles [qq16][0][0]));
3680.    xdisplat [xx16][contactpnt16] += xdisplacement2 + (delux [xx16][cont
   actpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][1][1] -
   rightisosceles [qq16][0][1]));
3681.
3682.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3683.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3684.
3685.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3686.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3687.
3688.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3689.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3690.
3691.    yforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * sin (alpha [xx16])) -
    ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha [xx16])
   ));
3692.    xforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * cos (alpha [xx16])) + ((fn [xx16][contactpnt16] + dn [x
   x16][contactpnt16]) * sin (alpha [xx16])));
3693.
3694.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3695.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3696.
3697.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3698.    x [xx16] = fxsum [xx16][contactpnt16];
3699.    resultfx [xx16] += x[xx16];
3700.
3701.
3702.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3703.    y [xx16] = fysum [xx16][contactpnt16];
3704.    resultfy [xx16] += y[xx16];
3705.
3706.
3707.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][1][0] - rightisosceles [xx16][0][0]) -
```

```
    xforce [xx16][contactpnt16] * ( rightisosceles [qq16][1][1] -
    rightisosceles [xx16][0][1]));
3708.    m [xx16] = msum [xx16][contactpnt16];
3709.    resultm [xx16] += m[xx16];
3710.
3711.
3712.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3713.    udotysum [xx16] = udoty [xx16][contactpnt16];
3714.    resultudoty [xx16] += udotysum [xx16];
3715.
3716.
3717.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3718.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3719.    resultudotx [xx16] += udotxsum [xx16];
3720.
3721.
3722.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3723.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3724.    resultthetadot [xx16] += thetadotsum [xx16];
3725.
3726.
3727.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3728.    deluysum [xx16] = deluy [xx16][contactpnt16];
3729.    resultdeluy [xx16] += deluysum [xx16];
3730.
3731.
3732.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3733.    deluxsum [xx16] = delux [xx16][contactpnt16];
3734.    resultdelux [xx16] += deluxsum [xx16];
3735.
3736.
3737.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3738.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3739.    resultdeltheta [xx16] += delthetasum [xx16];
3740.
3741.
3742.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3743.    uysum [xx16] = uy [xx16][contactpnt16];
3744.    resultuy [xx16]+= uysum [xx16];
3745.
3746.
3747.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3748.    uxsum [xx16] = ux [xx16][contactpnt16];
3749.    resultux [xx16] += uxsum [xx16];
3750.
3751.
```

```
    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
    thetasum [xx16] = theta [xx16][contactpnt16];
    resulttheta [xx16] += thetasum [xx16];
    alpha [xx16] = alpha1 + resulttheta [xx16];


    }

    }

    }

    next16:
    ;}

    ;}

    display:


     a_file<< dmass;

    return 0;
    }
```

Appendix E:

Code for Tailings-DEM (Series 2)

```
# include <iostream>
# include <cmath>
# include <fstream>
#include <numeric>

using namespace std;


 struct geometricalshapes  {  // subroutine for assigning values to the particle geometrical shapes
 float side1, side2, side3, side4, side5;
 float angle1, angle2, angle3, angle4, angle5;
 float height, base, centroidx, centroidy, momentofinertiax, momentofinertiay, area;

};

float ddamping_coefficient, dk;
float dstiffness_coefficient;
float dtime_increment;
float dmass;
float ii;  // mass moment of inertia
float dresult;


int isocounterbox1[100], isocounterbox2[100], isocounterbox3[100], isocounterbox4[100], isocounterbox5[100], isocounterbox6[100], isocounterbox7[100], isocounterbox8[100], isocounterbox9[100], isocounterbox10[100], isocounterbox11[100], isocounterbox12[100], isocounterbox13[100], isocounterbox14[100], isocounterbox15[100], isocounterbox16[100];
float p, p2;
int bb;


float ydisplat [500][500], xdisplat [500][500], delus [500][500], delun [500][500], fn [500][500], fs [500][000], dn [500][500], ds [500][500];
float yforce [500][500], xforce [500][500], fxsum [500][500], x [500], resultfx [500], fysum [500][500];
float y [500], resultfy [500], msum [500][500], m [500], resultm [500], udoty [500][500], udotysum [500], resultudoty [500];
float udotx [500][500], udotxsum [500], resultudotx [500], thetadot [500][500], thetadotsum [500], resultthetadot [500];
float deluy [500][500], deluysum [500], resultdeluy [500], delux [500][500], deluxsum [500], resultdelux [500], deltheta [500][500];
float delthetasum [500], resultdeltheta [500], uy [500][500], uysum [500], resultuy [500], ux [500][500], uxsum [500];
float resultux [500], theta [500][500], thetasum [500], resulttheta [500];


float rightisosceles [1000][4][10];
int isoscelescounter = 0;
```

```
float ydisplacement2 = 0.0, yvelocity2= 0;
float xdisplacement2 = 0.0;
float alpha [500], alpha1 = 0.785398;
int v = 1;
float n, n1;
int num3 =  1 + (39-4)/2.5;    // number of isosceles triangles per row
int hh, oo, pp;


int main ()

{

geometricalshapes righttriangle; //the particle
righttriangle.side1 = 4.243;
righttriangle.side2 = 3;
righttriangle.side3 = 3;
righttriangle.angle1 = 45;
righttriangle.angle2 = 45;
righttriangle.angle3 = 90;
righttriangle.height = 1.06;
righttriangle.base = 2.1215;
righttriangle.centroidx = 0;
righttriangle.centroidy = 0.707;
righttriangle.momentofinertiax = 1.123;
righttriangle.momentofinertiay = 3.374;
righttriangle.area = 1.1244;

 ofstream a_file ("data s2-mw16.txt");


 cout<< "Please enter the stiffness coefficient in N/m" << endl;
 cin>> dstiffness_coefficient;


 cout<< "Please enter the mass of the particle in kg" << endl;
 cin>> dmass;

ii =  ((righttriangle.base/2) * (righttriangle.base/2)* (dmass)/3) ;


 // time increment verification

 begin: //goto label
 cout<< "Please enter the time increment" << endl;
 cin>> dtime_increment;

 dresult = (2 * sqrt (dmass/dstiffness_coefficient));
 if (dtime_increment >=  dresult ) // time increment condition

```

```
92.{
93.
94. cout<< "Time increment does not satisfy condition"<< endl;
95. goto begin;
96.
97.}
98.
99. else
100.
101.      {
102.        cout<< "Time increment satisfies condition"<< endl;
103.
104.     }
105.
106.
107.     dk = 2 * (sqrt (dmass * dstiffness_coefficient)); // subroutine for
    calculating the damping coefficient
108.     ddamping_coefficient = dk/dtime_increment;
109.
110.
111.
112.
113.
114.
115.     for (hh = 0; hh <1000; hh = hh+1) {
116.         for (oo = 0; oo <4; oo = oo+1) { for (pp = 0; pp <10; pp = pp +
    1)
117.         {
118.             rightisosceles [hh][oo][pp] = 0.0;
119.
120.
121.         }
122.         }
123.     }
124.
125.
126.
127.      for (n = 3.0 ; n <= 70; n = n + 2.5) {       // positioning particl
    es
128.         for (n1 = 4.0 ; n1 <= 39; n1 = n1 + 2.5) {
129.
130.         rightisosceles [v][0][0] = n1 ;            // right isosceles x
    coordinates
131.         rightisosceles [v][1][0] = rightisosceles [v][0][0] + (righttria
    ngle.base /2);
132.         rightisosceles [v][2][0] = rightisosceles [v][0][0] -
     (righttriangle.base /2);
133.         rightisosceles [v][3][0] = rightisosceles [v][0][0];
134.
135.         rightisosceles [v][0][1] = n ;    // right isosceles y coordinat
    es
```

```
136.         rightisosceles [v][1][1] = rightisosceles [v][0][1] + (righttria
   ngle.height /3);
137.         rightisosceles [v][2][1] = rightisosceles [v][1][1];
138.         rightisosceles [v][3][1] = rightisosceles [v][2][1] -
    (righttriangle.height);
139.
140.
141.
142.         v = v + 1;
143.         isoscelescounter = isoscelescounter + 1;
144.
145.
146.
147.     }
148.     }
149.
150.     p2 = 10/num3;
151.
152.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
153.     ydisplacement2 =  ydisplacement2 + (yvelocity2 * dtime_increment);

154.
155.
156.
157.
158.
159.     for (p = 100; p < 100000 ; p = p+100)  // load cycle in Newtons
160.     {
161.     p2 = p/num3;
162.
163.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
164.
165.
166.       a_file <<  p << endl;
167.
168.
169.     int isobox1 = 0, isobox2 = 0, isobox3 = 0, isobox4 = 0, isobox5 = 0,
    isobox6 = 0, isobox7 = 0, isobox8 = 0, isobox9 = 0, isobox10 = 0, isobox1
   1 = 0, isobox12 = 0, isobox13 = 0, isobox14 = 0, isobox15 = 0, isobox16 =
   0;
170.
171.
172.     for (bb = 0; bb <100; bb = bb+1) {
173.         isocounterbox1[bb] = 0, isocounterbox2[bb] = 0, isocounterbox3[b
   b] = 0, isocounterbox4[bb]= 0;
174.         isocounterbox5[bb] = 0, isocounterbox6[bb] = 0, isocounterbox7[b
   b] = 0, isocounterbox8[bb] = 0;
175.         isocounterbox9[bb] = 0, isocounterbox10[bb] = 0, isocounterbox11
   [bb] = 0, isocounterbox12[bb] = 0;
```

```
176.          isocounterbox13[bb] = 0, isocounterbox14[bb] = 0, isocounterbox1
   5[bb] = 0, isocounterbox16[bb] = 0;
177.
178.          }
179.
180.
181.
182.     for (int dd = 1; dd <= isoscelescounter ; dd = dd + 1)  // locating
   the isosceles triangles within the 16 screen boxes
183.     {
184.
185.      if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][0]
   <= 11) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][1]<=
   18.5))
186.     {isocounterbox1[isobox1] = dd; isobox1 = isobox1 + 1;}
187.
188.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
189.     {isocounterbox2[isobox2] = dd; isobox2 = isobox2 + 1;}
190.
191.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
192.     {isocounterbox3[isobox3] = dd; isobox3 = isobox3 + 1;}
193.
194.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
195.     {isocounterbox4[isobox4] = dd; isobox4 = isobox4 + 1;}
196.
197.     else if( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][
   0] <= 11) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 37))
198.     {isocounterbox5[isobox5] = dd; isobox5 = isobox5 + 1;}
199.
200.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
201.     {isocounterbox6[isobox6] = dd; isobox6 = isobox6 + 1;}
202.
203.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
204.     {isocounterbox7[isobox7] = dd; isobox7 = isobox7 + 1;}
205.
206.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
207.     {isocounterbox8[isobox8] = dd; isobox8 = isobox8 + 1;}
208.
```

```
209.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 55.5))
210.     {isocounterbox9[isobox9] = dd; isobox9 = isobox9 + 1;}
211.
212.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
213.     {isocounterbox10[isobox10] = dd; isobox10 = isobox10 + 1;}
214.
215.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
216.     {isocounterbox11[isobox11] = dd; isobox11 = isobox11 + 1;}
217.
218.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
219.     {isocounterbox12[isobox12] = dd; isobox12 = isobox12 + 1;}
220.
221.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0
   ][1]<= 74))
222.     {isocounterbox13[isobox13] = dd; isobox13 = isobox13 + 1;}
223.
224.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
225.     {isocounterbox14[isobox14] = dd; isobox14 = isobox14 + 1;}
226.
227.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
228.     {isocounterbox15[isobox15] = dd; isobox15 = isobox15 + 1;}
229.
230.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
231.     {isocounterbox16[isobox16] = dd; isobox16 = isobox16 + 1;}
232.
233.
234.     }  // end of isosceles triangles location within the 16 screen boxes

235.
236.
237.     int gg, ww;
238.
239.     for (gg = 0; gg < 500; gg = gg+1) {
240.
241.      for (ww = 0; ww < 500; ww = ww +1) {
242.
243.     ydisplat [gg][ww] = 0.0;
```

```
244.     xdisplat [gg][ww] = 0.0;
245.     delus [gg][ww] = 0.0;
246.     delun [gg][ww] = 0.0;
247.     fn [gg][ww] = 0.0;
248.     fs [gg][ww] = 0.0;
249.     dn [gg][ww] = 0.0;
250.     ds [gg][ww] = 0.0;
251.     yforce [gg][ww] = 0.0;
252.     xforce [gg][ww] = 0.0;
253.     fxsum [gg][ww] = 0.0;
254.     x [ww] = 0.0;
255.     resultfx [ww] = 0.0;
256.     fysum [gg][ww] = 0.0;
257.     y [ww] = 0.0;
258.     resultfy [ww] = 0.0;
259.     msum [gg][ww] = 0.0;
260.     m [ww] = 0.0;
261.     resultm [ww] = 0.0;
262.     udoty [gg][ww] = 0.0;
263.     udotysum [ww] = 0.0;
264.     resultudoty [ww] = 0.0;
265.     udotx [gg][ww] = 0.0;
266.     udotxsum [ww] = 0.0;
267.     resultudotx [ww] = 0.0;
268.     thetadot [gg][ww] = 0.0;
269.     thetadotsum [ww] = 0.0;
270.     resultthetadot [ww] = 0.0;
271.     deluy [gg][ww] = 0.0;
272.     deluysum [ww] = 0.0;
273.     resultdeluy [ww] = 0.0;
274.     delux [gg][ww] = 0.0;
275.     deluxsum [ww] = 0.0;
276.     resultdelux [ww] =0.0;
277.     deltheta [gg][ww] = 0.0;
278.     delthetasum [ww] = 0.0;
279.     resultdeltheta [ww] = 0.0;
280.     uy [gg][ww] = 0.0;
281.     uysum [ww] = 0.0;
282.     resultuy [ww] = 0.0;
283.     ux [gg][ww] = 0.0;
284.     uxsum [ww] = 0.0;
285.     resultux [ww] = 0.0;
286.     theta [gg][ww] = 0.0;
287.     thetasum [ww] = 0.0;
288.     resulttheta [ww] = 0.0;
289.     alpha [ww] = 0.0;
290.
291.     }
292.
293.     }
294.
295.
```

```
296.
297.     int xx1 = 0, qq1 = 0, contactpnt1 = 0, l1, i1;
298.
299.
300.     for (i1 = 0; i1 < isobox1; i1= i1+1)
301.
302.     {
303.
304.     xx1 = isocounterbox1[i1];
305.
306.
307.
308.     rightisosceles [xx1][0][0] = rightisosceles [xx1][0][0] + resultux [
   xx1] ;
309.
310.     rightisosceles [xx1][0][1] = rightisosceles [xx1][0][1] + resultuy [
   xx1] ;
311.
312.
313.
314.     for ( l1 = 0; l1 < isobox1; l1 = l1+1)
315.
316.     {
317.
318.
319.     qq1 = isocounterbox1[l1];
320.
321.
322.     if ( ((pow((rightisosceles [xx1][0][0] -
    rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) > 0) && ((pow((rightisosceles [xx1][0][0
   ] - rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) <= 12.5))
323.
324.     {
325.
326.     contactpnt1 = contactpnt1 + 1;
327.
328.     if (contactpnt1 >= 4) {goto next1;}
329.
330.     if ( rightisosceles [qq1][3][1] >  rightisosceles [xx1][1][1] )
331.
332.     {
333.
334.
335.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][0] -
    rightisosceles [qq1][0][0]));
```

```
336.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][1] -
   rightisosceles [qq1][0][1]));
337.
338.
339.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
340.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
341.
342.
343.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
344.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
345.
346.
347.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
348.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
349.
350.
351.     yforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * sin (alpha1)) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha1)));
352.     xforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * cos (alpha1)) + ((fn [xx1][contactpnt1] + dn [xx1][contactpn
   t1]) * sin (alpha1)));
353.
354.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
355.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
356.
357.
358.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
359.     y [xx1] = fysum [xx1][contactpnt1];
360.     resultfy [xx1] += y[xx1];
361.
362.
363.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
364.     x [xx1] = fxsum [xx1][contactpnt1];
365.     resultfx [xx1] += x[xx1];
366.
367.
368.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][3][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][3][1] -
    rightisosceles [xx1][0][1]));
369.     m [xx1] = msum [xx1][contactpnt1];
```

```
370.     resultm [xx1] += m[xx1];
371.
372.
373.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
374.     udotysum [xx1] = udoty [xx1][contactpnt1];
375.     resultudoty [xx1] += udotysum [xx1];
376.
377.
378.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
379.     udotxsum [xx1] = udotx [xx1][contactpnt1];
380.     resultudotx [xx1] += udotxsum [xx1];
381.
382.
383.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
384.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
385.     resultthetadot [xx1] += thetadotsum [xx1];
386.
387.
388.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
389.     deluysum [xx1] = deluy [xx1][contactpnt1];
390.     resultdeluy [xx1] += deluysum [xx1];
391.
392.
393.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
394.     deluxsum [xx1] = delux [xx1][contactpnt1];
395.     resultdelux [xx1] += deluxsum [xx1];
396.
397.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
398.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
399.     resultdeltheta [xx1] += delthetasum [xx1];
400.
401.
402.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
403.     uysum [xx1] = uy [xx1][contactpnt1];
404.     resultuy [xx1]+= uysum [xx1];
405.
406.
407.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
408.     uxsum [xx1] = ux [xx1][contactpnt1];
409.     resultux [xx1] += uxsum [xx1];
410.
411.
412.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
413.     thetasum [xx1] = theta [xx1][contactpnt1];
414.     resulttheta [xx1] += thetasum [xx1];
415.     alpha [xx1] = alpha1 + resulttheta [xx1];
```

```
416.
417.     }
418.
419.
420.     else
421.     {
422.
423.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][0][0] -
    rightisosceles [qq1][0][0]));
424.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][1][1] -
   rightisosceles [qq1][0][1]));
425.
426.
427.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
428.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
429.
430.
431.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
432.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
433.
434.
435.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
436.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
437.
438.
439.     yforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * sin (alpha [xx1])) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha [xx1])));

440.     xforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * cos (alpha [xx1])) + ((fn [xx1][contactpnt1] + dn [xx1][conta
   ctpnt1]) * sin (alpha [xx1])));
441.
442.
443.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
444.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
445.
446.
```

```
447.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
448.     x [xx1] = fxsum [xx1][contactpnt1];
449.     resultfx [xx1] += x[xx1];
450.
451.
452.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
453.     y [xx1] = fysum [xx1][contactpnt1];
454.     resultfy [xx1] += y[xx1];
455.
456.
457.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][1][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][1][1] -
    rightisosceles [xx1][0][1]));
458.     m [xx1] = msum [xx1][contactpnt1];
459.     resultm [xx1] += m[xx1];
460.
461.
462.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
463.     udotysum [xx1] = udoty [xx1][contactpnt1];
464.     resultudoty [xx1] += udotysum [xx1];
465.
466.
467.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
468.     udotxsum [xx1] = udotx [xx1][contactpnt1];
469.     resultudotx [xx1] += udotxsum [xx1];
470.
471.
472.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
473.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
474.     resultthetadot [xx1] += thetadotsum [xx1];
475.
476.
477.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
478.     deluysum [xx1] = deluy [xx1][contactpnt1];
479.     resultdeluy [xx1] += deluysum [xx1];
480.
481.
482.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
483.     deluxsum [xx1] = delux [xx1][contactpnt1];
484.     resultdelux [xx1] += deluxsum [xx1];
485.
486.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
487.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
488.     resultdeltheta [xx1] += delthetasum [xx1];
489.
```

```
490.
491.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
492.     uysum [xx1] = uy [xx1][contactpnt1];
493.     resultuy [xx1]+= uysum [xx1];
494.
495.
496.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
497.     uxsum [xx1] = ux [xx1][contactpnt1];
498.     resultux [xx1] += uxsum [xx1];
499.
500.
501.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
502.     thetasum [xx1] = theta [xx1][contactpnt1];
503.     resulttheta [xx1] += thetasum [xx1];
504.     alpha [xx1] = alpha1 + resulttheta [xx1];
505.
506.
507.     }
508.
509.
510.
511.      a_file <<  p  << endl;
512.      a_file << xx1 << endl;
513.      a_file << resultuy [xx1]/10 << endl;
514.
515.     if ( resultuy [xx1] > 72 ) { goto display;}
516.
517.
518.
519.     }
520.
521.
522.     }
523.
524.     next1:
525.     ;}
526.
527.
528.
529.     int xx2 = 0, qq2 = 0, contactpnt2 = 0, l2, i2;
530.
531.
532.     for (i2 = 0; i2 < isobox2; i2= i2+1)
533.
534.     {
535.
536.     xx2 = isocounterbox2[i2];
537.
538.
539.     rightisosceles [xx2][0][0] = rightisosceles [xx2][0][0] + resultux [
    xx2] ;
540.
```

```
    rightisosceles [xx2][0][1] = rightisosceles [xx2][0][1] + resultuy [
xx2] ;


    for ( l2 = 0; l2 < isobox2; l2 = l2+1)

    {


    qq2 = isocounterbox2[l2];


    if ( ((pow((rightisosceles [xx2][0][0] -
 rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
 rightisosceles [qq2][0][1]), 2)) > 0) && ((pow((rightisosceles [xx2][0][0
] - rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
 rightisosceles [qq2][0][1]), 2)) <= 12.5))

    {

    contactpnt2 = contactpnt2 + 1;

    if (contactpnt2 >= 4) {goto next2;}


    if ( rightisosceles [qq2][3][1] >  rightisosceles [xx2][1][1] )

    {


    ydisplat [xx2][contactpnt2] = ydisplacement2  + (deluy [xx2][contact
pnt2] -
 deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
 [qq2][3][0] - rightisosceles [xx2][0][0])  -
 deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][0] -
 rightisosceles [qq2][0][0]));
    xdisplat [xx2][contactpnt2] = xdisplacement2 + (delux [xx2][contactp
nt2] -
 delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
 [qq2][3][1] - rightisosceles [xx2][0][1])  -
 deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][1] -
rightisosceles [qq2][0][1]));


    delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
    delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
1) - xdisplat [xx2][contactpnt2] * sin (alpha1));


```

```
575.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
576.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
577.
578.
579.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
580.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
581.
582.
583.     yforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * sin (alpha1)) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha1)));
584.     xforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * cos (alpha1)) + ((fn [xx2][contactpnt2] + dn [xx2][contactpn
   t2]) * sin (alpha1)));
585.
586.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
587.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
588.
589.
590.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
591.     y [xx2] = fysum [xx2][contactpnt2];
592.     resultfy [xx2] += y[xx2];
593.
594.
595.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
596.     x [xx2] = fxsum [xx2][contactpnt2];
597.     resultfx [xx2] += x[xx2];
598.
599.
600.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][3][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][3][1] -
    rightisosceles [xx2][0][1]));
601.     m [xx2] = msum [xx2][contactpnt2];
602.     resultm [xx2] += m[xx2];
603.
604.
605.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent) *1000/ dmass); // mm/sec
606.     udotysum [xx2] = udoty [xx2][contactpnt2];
607.     resultudoty [xx2] += udotysum [xx2];
608.
609.
610.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
611.     udotxsum [xx2] = udotx [xx2][contactpnt2];
612.     resultudotx [xx2] += udotxsum [xx2];
613.
```

```
614.
615.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
616.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
617.     resultthetadot [xx2] += thetadotsum [xx2];
618.
619.
620.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
621.     deluysum [xx2] = deluy [xx2][contactpnt2];
622.     resultdeluy [xx2] += deluysum [xx2];
623.
624.
625.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
626.     deluxsum [xx2] = delux [xx2][contactpnt2];
627.     resultdelux [xx2] += deluxsum [xx2];
628.
629.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
630.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
631.     resultdeltheta [xx2] += delthetasum [xx2];
632.
633.
634.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
635.     uysum [xx2] = uy [xx2][contactpnt2];
636.     resultuy [xx2]+= uysum [xx2];
637.
638.
639.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
640.     uxsum [xx2] = ux [xx2][contactpnt2];
641.     resultux [xx2] += uxsum [xx2];
642.
643.
644.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
645.     thetasum [xx2] = theta [xx2][contactpnt2];
646.     resulttheta [xx2] += thetasum [xx2];
647.     alpha [xx2] = alpha1 + resulttheta [xx2];
648.
649.     }
650.
651.     else
652.
653.     {
654.
655.
656.     ydisplat [xx2][contactpnt2] += ydisplacement2  + (deluy [xx2][contac
   tpnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][0][0] -
    rightisosceles [qq2][0][0]));
```

```
657.     xdisplat [xx2][contactpnt2] += xdisplacement2 + (delux [xx2][contact
   pnt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][1][1] -
   rightisosceles [qq2][0][1]));
658.
659.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
660.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
661.
662.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
663.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
664.
665.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
666.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
667.
668.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * sin (alpha [xx2])) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

669.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
   ctpnt2]) * sin (alpha [xx2])));
670.
671.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
672.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
673.
674.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
675.     x [xx2] = fxsum [xx2][contactpnt2];
676.     resultfx [xx2] += x[xx2];
677.
678.
679.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
680.     y [xx2] = fysum [xx2][contactpnt2];
681.     resultfy [xx2] += y[xx2];
682.
683.
684.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][1][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][1][1] -
    rightisosceles [xx2][0][1]));
685.     m [xx2] = msum [xx2][contactpnt2];
686.     resultm [xx2] += m[xx2];
687.
688.
```

```
689.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
690.     udotysum [xx2] = udoty [xx2][contactpnt2];
691.     resultudoty [xx2] += udotysum [xx2];
692.
693.
694.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
695.     udotxsum [xx2] = udotx [xx2][contactpnt2];
696.     resultudotx [xx2] += udotxsum [xx2];
697.
698.
699.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
700.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
701.     resultthetadot [xx2] += thetadotsum [xx2];
702.
703.
704.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
705.     deluysum [xx2] = deluy [xx2][contactpnt2];
706.     resultdeluy [xx2] += deluysum [xx2];
707.
708.
709.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
710.     deluxsum [xx2] = delux [xx2][contactpnt2];
711.     resultdelux [xx2] += deluxsum [xx2];
712.
713.
714.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
715.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
716.     resultdeltheta [xx2] += delthetasum [xx2];
717.
718.
719.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
720.     uysum [xx2] = uy [xx2][contactpnt2];
721.     resultuy [xx2]+= uysum [xx2];
722.
723.
724.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
725.     uxsum [xx2] = ux [xx2][contactpnt2];
726.     resultux [xx2] += uxsum [xx2];
727.
728.
729.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
730.     thetasum [xx2] = theta [xx2][contactpnt2];
731.     resulttheta [xx2] += thetasum [xx2];
732.     alpha [xx2] = alpha1 + resulttheta [xx2];
733.
734.     }
```

```
735.     }
736.
737.     }
738.
739.     next2:
740.     ;}
741.
742.     int xx3 = 0, qq3 = 0, contactpnt3 = 0, l3, i3;
743.
744.
745.     for (i3 = 0; i3 < isobox3; i3= i3+1)
746.
747.     {
748.
749.     xx3 = isocounterbox3[i3];
750.
751.
752.     rightisosceles [xx3][0][0] = rightisosceles [xx3][0][0] + resultux [
   xx3] ;
753.
754.     rightisosceles [xx3][0][1] = rightisosceles [xx3][0][1] + resultuy [
   xx3] ;
755.
756.
757.     for ( l3 = 0; l3 < isobox3; l3 = l3+1)
758.
759.     {
760.
761.
762.     qq3 = isocounterbox3[l3];
763.
764.
765.
766.     if ( ((pow((rightisosceles [xx3][0][0] -
    rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) > 0) && ((pow((rightisosceles [xx3][0][0
   ] - rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) <= 12.5))
767.
768.     {
769.
770.     contactpnt3 = contactpnt3 + 1;
771.
772.     if (contactpnt3 >= 4) {goto next3;}
773.
774.
775.     if ( rightisosceles [qq3][3][1] >  rightisosceles [xx3][1][1] )
776.
777.     {
778.
779.
```

```
780.     ydisplat [xx3][contactpnt3] = ydisplacement2  + (deluy [xx3][contact
   pnt3] -
    deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][0] - rightisosceles [xx3][0][0])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][0] -
    rightisosceles [qq3][0][0]));
781.     xdisplat [xx3][contactpnt3] = xdisplacement2 + (delux [xx3][contactp
   nt3] -
    delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][1] - rightisosceles [xx3][0][1])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][1] -
   rightisosceles [qq3][0][1]));
782.
783.
784.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
   1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
785.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
   1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
786.
787.
788.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
   1) *  (dstiffness_coefficient/1000);
789.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
   fficient/1000);
790.
791.
792.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
   1) * (ddamping_coefficient/1000);
793.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
   ient/1000);
794.
795.
796.     yforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * sin (alpha1)) -
    ((fn [xx3][contactpnt3] + dn [xx3][contactpnt3]) * cos (alpha1)));
797.     xforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * cos (alpha1)) + ((fn [xx3][contactpnt3] + dn [xx3][contactpn
   t3]) * sin (alpha1)));
798.
799.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
800.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
801.
802.
803.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
804.     y [xx3] = fysum [xx3][contactpnt3];
805.     resultfy [xx3] += y[xx3];
806.
807.
808.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
809.     x [xx3] = fxsum [xx3][contactpnt3];
810.     resultfx [xx3] += x[xx3];
811.
```

```
812.
813.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][3][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][3][1] -
    rightisosceles [xx3][0][1]));
814.     m [xx3] = msum [xx3][contactpnt3];
815.     resultm [xx3] += m[xx3];
816.
817.
818.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent) *1000/ dmass); // mm/sec
819.     udotysum [xx3] = udoty [xx3][contactpnt3];
820.     resultudoty [xx3] += udotysum [xx3];
821.
822.
823.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
824.     udotxsum [xx3] = udotx [xx3][contactpnt3];
825.     resultudotx [xx3] += udotxsum [xx3];
826.
827.
828.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
829.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
830.     resultthetadot [xx3] += thetadotsum [xx3];
831.
832.
833.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
834.     deluysum [xx3] = deluy [xx3][contactpnt3];
835.     resultdeluy [xx3] += deluysum [xx3];
836.
837.
838.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
839.     deluxsum [xx3] = delux [xx3][contactpnt3];
840.     resultdelux [xx3] += deluxsum [xx3];
841.
842.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
843.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
844.     resultdeltheta [xx3] += delthetasum [xx3];
845.
846.
847.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
848.     uysum [xx3] = uy [xx3][contactpnt3];
849.     resultuy [xx3]+= uysum [xx3];
850.
851.
852.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
853.     uxsum [xx3] = ux [xx3][contactpnt3];
854.     resultux [xx3] += uxsum [xx3];
```

```
855.
856.
857.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
858.     thetasum [xx3] = theta [xx3][contactpnt3];
859.     resulttheta [xx3] += thetasum [xx3];
860.     alpha [xx3] = alpha1 + resulttheta [xx3];
861.
862.     }
863.
864.      else
865.     {
866.
867.
868.     ydisplat [xx3][contactpnt2] += ydisplacement2  + (deluy [xx3][contac
    tpnt3] -
     deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
     [qq3][1][0] - rightisosceles [xx3][0][0])  -
     deltheta [qq3][contactpnt3] * (rightisosceles [qq3][0][0] -
     rightisosceles [qq3][0][0]));
869.     xdisplat [xx3][contactpnt2] += xdisplacement2 + (delux [xx3][contact
    pnt3] -
     delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
     [qq3][1][1] - rightisosceles [xx3][0][1])  -
     deltheta [qq3][contactpnt3] * (rightisosceles [qq3][1][1] -
    rightisosceles [qq3][0][1]));
870.
871.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
    1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
872.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
    1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
873.
874.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
    1) *  (dstiffness_coefficient/1000);
875.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
    fficient/1000);
876.
877.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
    1) * (ddamping_coefficient/1000);
878.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
    ient/1000);
879.
880.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
    tactpnt2]) * sin (alpha [xx2])) -
     ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

881.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
    tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
    ctpnt2]) * sin (alpha [xx2])));
882.
883.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
884.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
885.
```

```
886.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
887.     x [xx3] = fxsum [xx3][contactpnt3];
888.     resultfx [xx3] += x[xx3];
889.
890.
891.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
892.     y [xx3] = fysum [xx3][contactpnt3];
893.     resultfy [xx3] += y[xx3];
894.
895.
896.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][1][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][1][1] -
    rightisosceles [xx3][0][1]));
897.     m [xx3] = msum [xx3][contactpnt3];
898.     resultm [xx3] += m[xx3];
899.
900.
901.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
902.     udotysum [xx3] = udoty [xx3][contactpnt3];
903.     resultudoty [xx3] += udotysum [xx3];
904.
905.
906.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
907.     udotxsum [xx3] = udotx [xx3][contactpnt3];
908.     resultudotx [xx3] += udotxsum [xx3];
909.
910.
911.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
912.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
913.     resultthetadot [xx3] += thetadotsum [xx3];
914.
915.
916.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
917.     deluysum [xx3] = deluy [xx3][contactpnt3];
918.     resultdeluy [xx3] += deluysum [xx3];
919.
920.
921.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
922.     deluxsum [xx3] = delux [xx3][contactpnt3];
923.     resultdelux [xx3] += deluxsum [xx3];
924.
925.
926.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
927.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
928.     resultdeltheta [xx3] += delthetasum [xx3];
```

```
929.
930.
931.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
932.     uysum [xx3] = uy [xx3][contactpnt3];
933.     resultuy [xx3]+= uysum [xx3];
934.
935.
936.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
937.     uxsum [xx3] = ux [xx3][contactpnt3];
938.     resultux [xx3] += uxsum [xx3];
939.
940.
941.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
942.     thetasum [xx3] = theta [xx3][contactpnt3];
943.     resulttheta [xx3] += thetasum [xx3];
944.     alpha [xx3] = alpha1 + resulttheta [xx3];
945.     }
946.
947.     }
948.
949.     }
950.
951.     next3:
952.     ;}
953.
954.
955.     int xx4 = 0, qq4 = 0, contactpnt4 = 0, l4, i4;
956.
957.
958.     for (i4 = 0; i4 < isobox4; i4= i4+1)
959.
960.     {
961.
962.     xx4 = isocounterbox4[i4];
963.
964.
965.     rightisosceles [xx4][0][0] = rightisosceles [xx4][0][0] + resultux [
    xx4] ;
966.
967.     rightisosceles [xx4][0][1] = rightisosceles [xx4][0][1] + resultuy [
    xx4] ;
968.
969.
970.     for ( l4 = 0; l4 < isobox4; l4 = l4+1)
971.
972.     {
973.
974.
975.     qq4 = isocounterbox4[l4];
976.
977.
978.
```

```
    if ( ((pow((rightisosceles [xx4][0][0] -
 rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
 rightisosceles [qq4][0][1]), 2)) > 0) && ((pow((rightisosceles [xx4][0][0
] - rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
 rightisosceles [qq4][0][1]), 2)) <= 12.5))

    {

    contactpnt4 = contactpnt4 + 1;

    if (contactpnt4 >= 4) {goto next4;}


    if ( rightisosceles [qq4][3][1] >  rightisosceles [xx4][1][1] )

    {


    ydisplat [xx4][contactpnt4] = ydisplacement2  + (deluy [xx4][contact
pnt4] -
 deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
 [qq4][3][0] - rightisosceles [xx4][0][0])  -
 deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][0] -
 rightisosceles [qq4][0][0]));
    xdisplat [xx4][contactpnt4] = xdisplacement2 + (delux [xx4][contactp
nt4] -
 delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
 [qq4][3][1] - rightisosceles [xx4][0][1])  -
 deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][1] -
rightisosceles [qq4][0][1]));


    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
1) - xdisplat [xx4][contactpnt4] * sin (alpha1));


    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-
1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coe
fficient/1000);


    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-
1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coeffic
ient/1000);


```

```
    yforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * sin (alpha1)) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha1)));
    xforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * cos (alpha1)) + ((fn [xx4][contactpnt4] + dn [xx4][contactpn
t4]) * sin (alpha1)));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisoscel
es [qq4][3][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][3][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
    resultthetadot [xx4] += thetadotsum [xx4];


    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
t;
    deluysum [xx4] = deluy [xx4][contactpnt4];
    resultdeluy [xx4] += deluysum [xx4];

```

```
1050.
1051.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1052.    deluxsum [xx4] = delux [xx4][contactpnt4];
1053.    resultdelux [xx4] += deluxsum [xx4];
1054.
1055.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1056.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1057.    resultdeltheta [xx4] += delthetasum [xx4];
1058.
1059.
1060.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1061.    uysum [xx4] = uy [xx4][contactpnt4];
1062.    resultuy [xx4]+= uysum [xx4];
1063.
1064.
1065.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1066.    uxsum [xx4] = ux [xx4][contactpnt4];
1067.    resultux [xx4] += uxsum [xx4];
1068.
1069.
1070.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1071.    thetasum [xx4] = theta [xx4][contactpnt4];
1072.    resulttheta [xx4] += thetasum [xx4];
1073.    alpha [xx4] = alpha1 + resulttheta [xx4];
1074.
1075.    }
1076.
1077.    else
1078.
1079.    {
1080.
1081.
1082.    ydisplat [xx4][contactpnt4] += ydisplacement2  + (deluy [xx4][contac
   tpnt4] -
     deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][1][0] - rightisosceles [xx4][0][0])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][0][0] -
     rightisosceles [qq4][0][0]));
1083.    xdisplat [xx4][contactpnt4] += xdisplacement2 + (delux [xx4][contact
   pnt4] -
     delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][1][1] - rightisosceles [xx4][0][1])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][1][1] -
   rightisosceles [qq4][0][1]));
1084.
1085.    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
1086.    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));
1087.
```

```
    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coefficient/1000);

    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coefficient/1000);

    yforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][contactpnt4]) * sin (alpha [xx4])) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha [xx4])));

    xforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][contactpnt4]) * cos (alpha [xx4])) + ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * sin (alpha [xx4])));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);

    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisosceles [qq4][1][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][1][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increment)* 1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increment)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


```

```
    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
    resultthetadot [xx4] += thetadotsum [xx4];


    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
t;
    deluysum [xx4] = deluy [xx4][contactpnt4];
    resultdeluy [xx4] += deluysum [xx4];


    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
t;
    deluxsum [xx4] = delux [xx4][contactpnt4];
    resultdelux [xx4] += deluxsum [xx4];


    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
crement;
    delthetasum [xx4] = deltheta [xx4][contactpnt4];
    resultdeltheta [xx4] += delthetasum [xx4];


    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
    uysum [xx4] = uy [xx4][contactpnt4];
    resultuy [xx4]+= uysum [xx4];


    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
    uxsum [xx4] = ux [xx4][contactpnt4];
    resultux [xx4] += uxsum [xx4];


    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
    thetasum [xx4] = theta [xx4][contactpnt4];
    resulttheta [xx4] += thetasum [xx4];
    alpha [xx4] = alpha1 + resulttheta [xx4];
    }

    }

    }

    next4:
    ;}


    int xx5 = 0, qq5 = 0, contactpnt5 = 0, l5, i5;


    for (i5 = 0; i5 < isobox5; i5= i5+1)
```

```
1173.
1174.    {
1175.
1176.    xx5 = isocounterbox5[i5];
1177.
1178.    rightisosceles [xx5][0][0] = rightisosceles [xx5][0][0] + resultux [
   xx5] ;
1179.
1180.    rightisosceles [xx5][0][1] = rightisosceles [xx5][0][1] + resultuy [
   xx5] ;
1181.
1182.
1183.
1184.    for ( l5 = 0; l5 < isobox5; l5 = l5+1)
1185.
1186.    {
1187.
1188.
1189.    qq5 = isocounterbox5[l5];
1190.
1191.
1192.
1193.    if ( ((pow((rightisosceles [xx5][0][0] -
    rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) > 0) && ((pow((rightisosceles [xx5][0][0
   ] - rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) <= 12.5))
1194.
1195.    {
1196.
1197.    contactpnt5 = contactpnt5 + 1;
1198.
1199.    if (contactpnt5 >= 4) {goto next5;}
1200.
1201.
1202.    if ( rightisosceles [qq5][3][1] >  rightisosceles [xx5][1][1] )
1203.
1204.    {
1205.
1206.
1207.    ydisplat [xx5][contactpnt5] = ydisplacement2  + (deluy [xx5][contact
   pnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][0] -
    rightisosceles [qq5][0][0]));
1208.    xdisplat [xx5][contactpnt5] = xdisplacement2 + (delux [xx5][contactp
   nt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][1] -
   rightisosceles [qq5][0][1]));
```

```
1209.
1210.
1211.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1212.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1213.
1214.
1215.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1216.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1217.
1218.
1219.    dn [xx5][contactpnt5] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1220.    ds [xx5][contactpnt5] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1221.
1222.
1223.    yforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * sin (alpha1)) -
     ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha1)));
1224.    xforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * cos (alpha1)) + ((fn [xx5][contactpnt5] + dn [xx5][contactpn
   t5]) * sin (alpha1)));
1225.
1226.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1227.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1228.
1229.
1230.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1231.    y [xx5] = fysum [xx5][contactpnt5];
1232.    resultfy [xx5] += y[xx5];
1233.
1234.
1235.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1236.    x [xx5] = fxsum [xx5][contactpnt5];
1237.    resultfx [xx5] += x[xx5];
1238.
1239.
1240.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][3][0] - rightisosceles [xx5][0][0]) -
     xforce [xx5][contactpnt5] * ( rightisosceles [qq5][3][1] -
     rightisosceles [xx5][0][1]));
1241.    m [xx5] = msum [xx5][contactpnt5];
1242.    resultm [xx5] += m[xx5];
1243.
1244.
1245.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1246.    udotysum [xx5] = udoty [xx5][contactpnt5];
```

```
1247.    resultudoty [xx5] += udotysum [xx5];
1248.
1249.
1250.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1251.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1252.    resultudotx [xx5] += udotxsum [xx5];
1253.
1254.
1255.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1256.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1257.    resultthetadot [xx5] += thetadotsum [xx5];
1258.
1259.
1260.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1261.    deluysum [xx5] = deluy [xx5][contactpnt5];
1262.    resultdeluy [xx5] += deluysum [xx5];
1263.
1264.
1265.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1266.    deluxsum [xx5] = delux [xx5][contactpnt5];
1267.    resultdelux [xx5] += deluxsum [xx5];
1268.
1269.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1270.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1271.    resultdeltheta [xx5] += delthetasum [xx5];
1272.
1273.
1274.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1275.    uysum [xx5] = uy [xx5][contactpnt5];
1276.    resultuy [xx5]+= uysum [xx5];
1277.
1278.
1279.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1280.    uxsum [xx5] = ux [xx5][contactpnt5];
1281.    resultux [xx5] += uxsum [xx5];
1282.
1283.
1284.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1285.    thetasum [xx5] = theta [xx5][contactpnt5];
1286.    resulttheta [xx5] += thetasum [xx5];
1287.    alpha [xx5] = alpha1 + resulttheta [xx5];
1288.
1289.    }
1290.
1291.    else
1292.
1293.    {
```

```
1294.
1295.
1296.    ydisplat [xx5][contactpnt5] += ydisplacement2  + (deluy [xx5][contac
   tpnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][0][0] -
    rightisosceles [qq5][0][0]));
1297.    xdisplat [xx5][contactpnt5] += xdisplacement2 + (delux [xx5][contact
   pnt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][1][1] -
   rightisosceles [qq5][0][1]));
1298.
1299.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1300.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1301.
1302.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1303.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1304.
1305.    dn [xx5][contactpnt2] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1306.    ds [xx5][contactpnt2] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1307.
1308.    yforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * sin (alpha [xx5])) -
    ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha [xx5])));

1309.    xforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * cos (alpha [xx5])) + ((fn [xx5][contactpnt5] + dn [xx5][conta
   ctpnt5]) * sin (alpha [xx5])));
1310.
1311.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1312.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1313.
1314.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1315.    x [xx5] = fxsum [xx5][contactpnt5];
1316.    resultfx [xx5] += x[xx5];
1317.
1318.
1319.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1320.    y [xx5] = fysum [xx5][contactpnt5];
1321.    resultfy [xx5] += y[xx5];
1322.
1323.
```

```
1324.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][1][0] - rightisosceles [xx5][0][0]) -
    xforce [xx5][contactpnt5] * ( rightisosceles [qq5][1][1] -
    rightisosceles [xx5][0][1]));
1325.    m [xx5] = msum [xx5][contactpnt5];
1326.    resultm [xx5] += m[xx5];
1327.
1328.
1329.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1330.    udotysum [xx5] = udoty [xx5][contactpnt5];
1331.    resultudoty [xx5] += udotysum [xx5];
1332.
1333.
1334.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1335.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1336.    resultudotx [xx5] += udotxsum [xx5];
1337.
1338.
1339.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1340.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1341.    resultthetadot [xx5] += thetadotsum [xx5];
1342.
1343.
1344.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1345.    deluysum [xx5] = deluy [xx5][contactpnt5];
1346.    resultdeluy [xx5] += deluysum [xx5];
1347.
1348.
1349.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1350.    deluxsum [xx5] = delux [xx5][contactpnt5];
1351.    resultdelux [xx5] += deluxsum [xx5];
1352.
1353.
1354.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1355.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1356.    resultdeltheta [xx5] += delthetasum [xx5];
1357.
1358.
1359.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1360.    uysum [xx5] = uy [xx5][contactpnt5];
1361.    resultuy [xx5]+= uysum [xx5];
1362.
1363.
1364.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1365.    uxsum [xx5] = ux [xx5][contactpnt5];
1366.    resultux [xx5] += uxsum [xx5];
```

```
1367.
1368.
1369.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1370.    thetasum [xx5] = theta [xx5][contactpnt5];
1371.    resulttheta [xx5] += thetasum [xx5];
1372.    alpha [xx5] = alpha1 + resulttheta [xx5];
1373.
1374.    }
1375.    }
1376.
1377.    }
1378.
1379.    next5:
1380.    ;}
1381.
1382.
1383.    int xx6 = 0, qq6 = 0, contactpnt6 = 0, l6, i6;
1384.
1385.
1386.    for (i6 = 0; i6 < isobox6; i6= i6+1)
1387.
1388.    {
1389.
1390.    xx6 = isocounterbox6[i6];
1391.
1392.    rightisosceles [xx6][0][0] = rightisosceles [xx6][0][0] + resultux [
   xx6] ;
1393.
1394.    rightisosceles [xx6][0][1] = rightisosceles [xx6][0][1] + resultuy [
   xx6] ;
1395.
1396.
1397.
1398.    for ( l6 = 0; l6 < isobox6; l6 = l6+1)
1399.
1400.    {
1401.
1402.
1403.    qq6 = isocounterbox6[l6];
1404.
1405.
1406.
1407.    if ( ((pow((rightisosceles [xx6][0][0] -
    rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) > 0) && ((pow((rightisosceles [xx6][0][0
   ] - rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) <= 12.5))
1408.
1409.    {
1410.
1411.    contactpnt6 = contactpnt6 + 1;
1412.
```

```
1413.    if (contactpnt6 >= 4) {goto next6;}
1414.
1415.
1416.    if ( rightisosceles [qq6][3][1] >  rightisosceles [xx6][1][1] )
1417.
1418.    {
1419.
1420.
1421.    ydisplat [xx6][contactpnt6] = ydisplacement2  + (deluy [xx6][contact
   pnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][0] -
    rightisosceles [qq6][0][0]));
1422.    xdisplat [xx6][contactpnt6] = xdisplacement2 + (delux [xx6][contactp
   nt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][1] -
   rightisosceles [qq6][0][1]));
1423.
1424.
1425.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1426.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1427.
1428.
1429.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1430.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1431.
1432.
1433.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1434.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1435.
1436.
1437.    yforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * sin (alpha1)) -
     ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha1)));
1438.    xforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * cos (alpha1)) + ((fn [xx6][contactpnt6] + dn [xx6][contactpn
   t6]) * sin (alpha1)));
1439.
1440.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1441.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1442.
1443.
1444.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
```

```
    y [xx6] = fysum [xx6][contactpnt6];
    resultfy [xx6] += y[xx6];


    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
    x [xx6] = fxsum [xx6][contactpnt6];
    resultfx [xx6] += x[xx6];


    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][3][0] - rightisosceles [xx6][0][0]) -
     xforce [xx6][contactpnt6] * ( rightisosceles [qq6][3][1] -
     rightisosceles [xx6][0][1]));
    m [xx6] = msum [xx6][contactpnt6];
    resultm [xx6] += m[xx6];


    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent) *1000/ dmass); // mm/sec
    udotysum [xx6] = udoty [xx6][contactpnt6];
    resultudoty [xx6] += udotysum [xx6];


    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
    udotxsum [xx6] = udotx [xx6][contactpnt6];
    resultudotx [xx6] += udotxsum [xx6];


    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
    resultthetadot [xx6] += thetadotsum [xx6];


    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
    deluysum [xx6] = deluy [xx6][contactpnt6];
    resultdeluy [xx6] += deluysum [xx6];


    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
    deluxsum [xx6] = delux [xx6][contactpnt6];
    resultdelux [xx6] += deluxsum [xx6];

    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
    delthetasum [xx6] = deltheta [xx6][contactpnt6];
    resultdeltheta [xx6] += delthetasum [xx6];


```

```
1488.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1489.    uysum [xx6] = uy [xx6][contactpnt6];
1490.    resultuy [xx6]+= uysum [xx6];
1491.
1492.
1493.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1494.    uxsum [xx6] = ux [xx6][contactpnt6];
1495.    resultux [xx6] += uxsum [xx6];
1496.
1497.
1498.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1499.    thetasum [xx6] = theta [xx6][contactpnt6];
1500.    resulttheta [xx6] += thetasum [xx6];
1501.    alpha [xx6] = alpha1 + resulttheta [xx6];
1502.
1503.    }
1504.
1505.    else
1506.
1507.    {
1508.
1509.
1510.
1511.    ydisplat [xx6][contactpnt6] += ydisplacement2  + (deluy [xx6][contac
   tpnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][0][0] -
    rightisosceles [qq6][0][0]));
1512.    xdisplat [xx6][contactpnt6] += xdisplacement2 + (delux [xx6][contact
   pnt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][1][1] -
   rightisosceles [qq6][0][1]));
1513.
1514.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1515.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1516.
1517.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1518.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1519.
1520.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1521.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1522.
```

```
1523.    yforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * sin (alpha [xx6])) -
    ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha [xx6])));

1524.    xforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * cos (alpha [xx6])) + ((fn [xx6][contactpnt6] + dn [xx6][conta
   ctpnt6]) * sin (alpha [xx6])));
1525.
1526.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1527.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1528.
1529.    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
1530.    x [xx6] = fxsum [xx6][contactpnt6];
1531.    resultfx [xx6] += x[xx6];
1532.
1533.
1534.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
1535.    y [xx6] = fysum [xx6][contactpnt6];
1536.    resultfy [xx6] += y[xx6];
1537.
1538.
1539.    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][1][0] - rightisosceles [xx6][0][0]) -
    xforce [xx6][contactpnt6] * ( rightisosceles [qq6][1][1] -
    rightisosceles [xx6][0][1]));
1540.    m [xx6] = msum [xx6][contactpnt6];
1541.    resultm [xx6] += m[xx6];
1542.
1543.
1544.    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1545.    udotysum [xx6] = udoty [xx6][contactpnt6];
1546.    resultudoty [xx6] += udotysum [xx6];
1547.
1548.
1549.    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1550.    udotxsum [xx6] = udotx [xx6][contactpnt6];
1551.    resultudotx [xx6] += udotxsum [xx6];
1552.
1553.
1554.    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
1555.    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
1556.    resultthetadot [xx6] += thetadotsum [xx6];
1557.
1558.
1559.    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
1560.    deluysum [xx6] = deluy [xx6][contactpnt6];
1561.    resultdeluy [xx6] += deluysum [xx6];
1562.
```

```
1563.
1564.    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
1565.    deluxsum [xx6] = delux [xx6][contactpnt6];
1566.    resultdelux [xx6] += deluxsum [xx6];
1567.
1568.
1569.    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
1570.    delthetasum [xx6] = deltheta [xx6][contactpnt6];
1571.    resultdeltheta [xx6] += delthetasum [xx6];
1572.
1573.
1574.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1575.    uysum [xx6] = uy [xx6][contactpnt6];
1576.    resultuy [xx6]+= uysum [xx6];
1577.
1578.
1579.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1580.    uxsum [xx6] = ux [xx6][contactpnt6];
1581.    resultux [xx6] += uxsum [xx6];
1582.
1583.
1584.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1585.    thetasum [xx6] = theta [xx6][contactpnt6];
1586.    resulttheta [xx6] += thetasum [xx6];
1587.    alpha [xx6] = alpha1 + resulttheta [xx6];
1588.
1589.    }
1590.
1591.    }
1592.
1593.    }
1594.
1595.    next6:
1596.    ;}
1597.
1598.
1599.
1600.    int xx7 = 0, qq7 = 0, contactpnt7 = 0, l7, i7;
1601.
1602.
1603.    for (i7 = 0; i7 < isobox7; i7= i7+1)
1604.
1605.    {
1606.
1607.    xx7 = isocounterbox7[i7];
1608.
1609.    rightisosceles [xx7][0][0] = rightisosceles [xx7][0][0] + resultux [
   xx7] ;
1610.
```

```
    rightisosceles [xx7][0][1] = rightisosceles [xx7][0][1] + resultuy [
   xx7] ;


    for ( l7 = 0; l7 < isobox7; l7 = l7+1)

    {


    qq7 = isocounterbox7[l7];


    if ( ((pow((rightisosceles [xx7][0][0] -
    rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) > 0) && ((pow((rightisosceles [xx7][0][0
   ] - rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) <= 12.5))

    {

    contactpnt7 = contactpnt7 + 1;

    if (contactpnt7 >= 4) {goto next7;}


    if ( rightisosceles [qq7][3][1] >  rightisosceles [xx7][1][1] )

    {


    ydisplat [xx7][contactpnt7] = ydisplacement2  + (deluy [xx7][contact
   pnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][0] - rightisosceles [xx7][0][0])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][0] -
    rightisosceles [qq7][0][0]));
    xdisplat [xx7][contactpnt7] = xdisplacement2 + (delux [xx7][contactp
   nt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][1] -
   rightisosceles [qq7][0][1]));


    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));


```

```
1646.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1647.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1648.
1649.
1650.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1651.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1652.
1653.
1654.    yforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * sin (alpha1)) -
     ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha1)));
1655.    xforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * cos (alpha1)) + ((fn [xx7][contactpnt7] + dn [xx7][contactpn
   t7]) * sin (alpha1)));
1656.
1657.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1658.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1659.
1660.
1661.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1662.    y [xx7] = fysum [xx7][contactpnt7];
1663.    resultfy [xx7] += y[xx7];
1664.
1665.
1666.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1667.    x [xx7] = fxsum [xx7][contactpnt7];
1668.    resultfx [xx7] += x[xx7];
1669.
1670.
1671.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][3][0] - rightisosceles [xx7][0][0]) -
     xforce [xx7][contactpnt7] * ( rightisosceles [qq7][3][1] -
     rightisosceles [xx7][0][1]));
1672.    m [xx7] = msum [xx7][contactpnt7];
1673.    resultm [xx7] += m[xx7];
1674.
1675.
1676.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1677.    udotysum [xx7] = udoty [xx7][contactpnt7];
1678.    resultudoty [xx7] += udotysum [xx7];
1679.
1680.
1681.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1682.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1683.    resultudotx [xx7] += udotxsum [xx7];
1684.
```

```
1685.
1686.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1687.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1688.    resultthetadot [xx7] += thetadotsum [xx7];
1689.
1690.
1691.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1692.    deluysum [xx7] = deluy [xx7][contactpnt7];
1693.    resultdeluy [xx7] += deluysum [xx7];
1694.
1695.
1696.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1697.    deluxsum [xx7] = delux [xx7][contactpnt7];
1698.    resultdelux [xx7] += deluxsum [xx7];
1699.
1700.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1701.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1702.    resultdeltheta [xx7] += delthetasum [xx7];
1703.
1704.
1705.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1706.    uysum [xx7] = uy [xx7][contactpnt7];
1707.    resultuy [xx7]+= uysum [xx7];
1708.
1709.
1710.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1711.    uxsum [xx7] = ux [xx7][contactpnt7];
1712.    resultux [xx7] += uxsum [xx7];
1713.
1714.
1715.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1716.    thetasum [xx7] = theta [xx7][contactpnt7];
1717.    resultttheta [xx7] += thetasum [xx7];
1718.    alpha [xx7] = alpha1 + resultttheta [xx7];
1719.
1720.    }
1721.
1722.    else
1723.
1724.    {
1725.
1726.
1727.
1728.    ydisplat [xx7][contactpnt7] += ydisplacement2  + (deluy [xx7][contac
   tpnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][0] - rightisosceles [xx7][0][0])  -
```

```
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][0][0] -
    rightisosceles [qq7][0][0]));
1729.    xdisplat [xx7][contactpnt7] += xdisplacement2 + (delux [xx7][contact
   pnt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][1][1] -
   rightisosceles [qq7][0][1]));
1730.
1731.    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
1732.    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));
1733.
1734.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1735.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1736.
1737.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1738.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1739.
1740.    yforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * sin (alpha [xx7])) -
    ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha [xx7])));

1741.    xforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * cos (alpha [xx7])) + ((fn [xx7][contactpnt7] + dn [xx7][conta
   ctpnt7]) * sin (alpha [xx7])));
1742.
1743.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1744.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1745.
1746.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1747.    x [xx7] = fxsum [xx7][contactpnt7];
1748.    resultfx [xx7] += x[xx7];
1749.
1750.
1751.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1752.    y [xx7] = fysum [xx7][contactpnt7];
1753.    resultfy [xx7] += y[xx7];
1754.
1755.
1756.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][1][0] - rightisosceles [xx7][0][0]) -
    xforce [xx7][contactpnt7] * ( rightisosceles [qq7][1][1] -
    rightisosceles [xx7][0][1]));
1757.    m [xx7] = msum [xx7][contactpnt7];
1758.    resultm [xx7] += m[xx7];
1759.
```

```
1760.
1761.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1762.    udotysum [xx7] = udoty [xx7][contactpnt7];
1763.    resultudoty [xx7] += udotysum [xx7];
1764.
1765.
1766.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1767.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1768.    resultudotx [xx7] += udotxsum [xx7];
1769.
1770.
1771.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1772.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1773.    resultthetadot [xx7] += thetadotsum [xx7];
1774.
1775.
1776.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1777.    deluysum [xx7] = deluy [xx7][contactpnt7];
1778.    resultdeluy [xx7] += deluysum [xx7];
1779.
1780.
1781.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1782.    deluxsum [xx7] = delux [xx7][contactpnt7];
1783.    resultdelux [xx7] += deluxsum [xx7];
1784.
1785.
1786.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1787.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1788.    resultdeltheta [xx7] += delthetasum [xx7];
1789.
1790.
1791.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1792.    uysum [xx7] = uy [xx7][contactpnt7];
1793.    resultuy [xx7]+= uysum [xx7];
1794.
1795.
1796.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1797.    uxsum [xx7] = ux [xx7][contactpnt7];
1798.    resultux [xx7] += uxsum [xx7];
1799.
1800.
1801.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1802.    thetasum [xx7] = theta [xx7][contactpnt7];
1803.    resulttheta [xx7] += thetasum [xx7];
1804.    alpha [xx7] = alpha1 + resulttheta [xx7];
1805.    }
```

```
1806.
1807.    }
1808.
1809.    }
1810.
1811.    next7:
1812.    ;}
1813.
1814.
1815.
1816.    int xx8 = 0, qq8 = 0, contactpnt8 = 0, l8, i8;
1817.
1818.
1819.    for (i8 = 0; i8 < isobox8; i8= i8+1)
1820.
1821.    {
1822.
1823.    xx8 = isocounterbox8[i8];
1824.
1825.    rightisosceles [xx8][0][0] = rightisosceles [xx8][0][0] + resultux [
   xx8] ;
1826.
1827.    rightisosceles [xx8][0][1] = rightisosceles [xx8][0][1] + resultuy [
   xx8] ;
1828.
1829.
1830.
1831.    for ( l8 = 0; l8 < isobox8; l8 = l8+1)
1832.
1833.    {
1834.
1835.
1836.    qq8 = isocounterbox8[l8];
1837.
1838.
1839.
1840.    if ( ((pow((rightisosceles [xx8][0][0] -
    rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) > 0) && ((pow((rightisosceles [xx8][0][0
   ] - rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) <= 12.5))
1841.
1842.    {
1843.
1844.    contactpnt8 = contactpnt8 + 1;
1845.
1846.    if (contactpnt8 >= 4) {goto next8;}
1847.
1848.
1849.    if ( rightisosceles [qq8][3][1] >  rightisosceles [xx8][1][1] )
1850.
1851.    {
```

```
1852.
1853.
1854.    ydisplat [xx8][contactpnt8] = ydisplacement2  + (deluy [xx8][contact
   pnt8] -
    deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][0] - rightisosceles [xx8][0][0])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][0] -
    rightisosceles [qq8][0][0]));
1855.    xdisplat [xx8][contactpnt8] = xdisplacement2 + (delux [xx8][contactp
   nt8] -
    delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][1] - rightisosceles [xx8][0][1])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][1] -
   rightisosceles [qq8][0][1]));
1856.
1857.
1858.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1859.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1860.
1861.
1862.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt8] * (-
   1) *  (dstiffness_coefficient/1000);
1863.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt8] *  (dstiffness_coe
   fficient/1000);
1864.
1865.
1866.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1867.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1868.
1869.
1870.    yforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * sin (alpha1)) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha1)));
1871.    xforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * cos (alpha1)) + ((fn [xx8][contactpnt8] + dn [xx8][contactpn
   t8]) * sin (alpha1)));
1872.
1873.    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
1874.    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);
1875.
1876.
1877.    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
1878.    y [xx8] = fysum [xx8][contactpnt8];
1879.    resultfy [xx8] += y[xx8];
1880.
1881.
1882.    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
1883.    x [xx8] = fxsum [xx8][contactpnt8];
```

```
1884.    resultfx [xx8] += x[xx8];
1885.
1886.
1887.    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
   es [qq8][3][0] - rightisosceles [xx8][0][0]) -
    xforce [xx8][contactpnt8] * ( rightisosceles [qq8][3][1] -
    rightisosceles [xx8][0][1]));
1888.    m [xx8] = msum [xx8][contactpnt8];
1889.    resultm [xx8] += m[xx8];
1890.
1891.
1892.    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1893.    udotysum [xx8] = udoty [xx8][contactpnt8];
1894.    resultudoty [xx8] += udotysum [xx8];
1895.
1896.
1897.    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1898.    udotxsum [xx8] = udotx [xx8][contactpnt8];
1899.    resultudotx [xx8] += udotxsum [xx8];
1900.
1901.
1902.    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
   ement)*1000/ ii); //1/s
1903.    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
1904.    resultthetadot [xx8] += thetadotsum [xx8];
1905.
1906.
1907.    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
   t;
1908.    deluysum [xx8] = deluy [xx8][contactpnt8];
1909.    resultdeluy [xx8] += deluysum [xx8];
1910.
1911.
1912.    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
   t;
1913.    deluxsum [xx8] = delux [xx8][contactpnt8];
1914.    resultdelux [xx8] += deluxsum [xx8];
1915.
1916.    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
   crement;
1917.    delthetasum [xx8] = deltheta [xx8][contactpnt8];
1918.    resultdeltheta [xx8] += delthetasum [xx8];
1919.
1920.
1921.    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
1922.    uysum [xx8] = uy [xx8][contactpnt8];
1923.    resultuy [xx8]+= uysum [xx8];
1924.
1925.
1926.    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
```

```
1927.    uxsum [xx8] = ux [xx8][contactpnt8];
1928.    resultux [xx8] += uxsum [xx8];
1929.
1930.
1931.    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
1932.    thetasum [xx8] = theta [xx8][contactpnt8];
1933.    resulttheta [xx8] += thetasum [xx8];
1934.    alpha [xx8] = alpha1 + resulttheta [xx8];
1935.
1936.    }
1937.
1938.    else
1939.
1940.    {
1941.
1942.
1943.    ydisplat [xx8][contactpnt8] += ydisplacement2  + (deluy [xx8][contac
   tpnt8] -
     deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][0] - rightisosceles [xx8][0][0])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][0][0] -
     rightisosceles [qq8][0][0]));
1944.    xdisplat [xx8][contactpnt8] += xdisplacement2 + (delux [xx8][contact
   pnt8] -
     delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][1] - rightisosceles [xx8][0][1])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][1][1] -
   rightisosceles [qq8][0][1]));
1945.
1946.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1947.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1948.
1949.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1950.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1951.
1952.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1953.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1954.
1955.    yforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * sin (alpha [xx8])) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha [xx8])));

1956.    xforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * cos (alpha [xx8])) + ((fn [xx8][contactpnt8] + dn [xx8][conta
   ctpnt8]) * sin (alpha [xx8])));
1957.
```

```
1958.    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
1959.    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);
1960.
1961.    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
1962.    x [xx8] = fxsum [xx8][contactpnt8];
1963.    resultfx [xx8] += x[xx8];
1964.
1965.
1966.    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
1967.    y [xx8] = fysum [xx8][contactpnt8];
1968.    resultfy [xx8] += y[xx8];
1969.
1970.
1971.    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
   es [qq8][1][0] - rightisosceles [xx8][0][0]) -
    xforce [xx8][contactpnt8] * ( rightisosceles [qq8][1][1] -
    rightisosceles [xx8][0][1]));
1972.    m [xx8] = msum [xx8][contactpnt8];
1973.    resultm [xx8] += m[xx8];
1974.
1975.
1976.    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1977.    udotysum [xx8] = udoty [xx8][contactpnt8];
1978.    resultudoty [xx8] += udotysum [xx8];
1979.
1980.
1981.    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1982.    udotxsum [xx8] = udotx [xx8][contactpnt8];
1983.    resultudotx [xx8] += udotxsum [xx8];
1984.
1985.
1986.    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
   ement)*1000/ ii); //1/s
1987.    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
1988.    resultthetadot [xx8] += thetadotsum [xx8];
1989.
1990.
1991.    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
   t;
1992.    deluysum [xx8] = deluy [xx8][contactpnt8];
1993.    resultdeluy [xx8] += deluysum [xx8];
1994.
1995.
1996.    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
   t;
1997.    deluxsum [xx8] = delux [xx8][contactpnt8];
1998.    resultdelux [xx8] += deluxsum [xx8];
1999.
2000.
```

```
    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
crement;
    delthetasum [xx8] = deltheta [xx8][contactpnt8];
    resultdeltheta [xx8] += delthetasum [xx8];


    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
    uysum [xx8] = uy [xx8][contactpnt8];
    resultuy [xx8]+= uysum [xx8];


    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
    uxsum [xx8] = ux [xx8][contactpnt8];
    resultux [xx8] += uxsum [xx8];


    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
    thetasum [xx8] = theta [xx8][contactpnt8];
    resulttheta [xx8] += thetasum [xx8];
    alpha [xx8] = alpha1 + resulttheta [xx8];


    }
    }

    }

    next8:
    ;}


    int xx9 = 0, qq9 = 0, contactpnt9 = 0, l9, i9;


    for (i9 = 0; i9 < isobox9; i9= i9+1)

    {

    xx9 = isocounterbox9[i9];

    rightisosceles [xx9][0][0] = rightisosceles [xx9][0][0] + resultux [
xx9] ;

    rightisosceles [xx9][0][1] = rightisosceles [xx9][0][1] + resultuy [
xx9] ;


    for ( l9 = 0; l9 < isobox9; l9 = l9+1)

    {
```

```
2050.
2051.
2052.    qq9 = isocounterbox9[19];
2053.
2054.
2055.
2056.    if ( ((pow((rightisosceles [xx9][0][0] -
    rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) > 0) && ((pow((rightisosceles [xx9][0][0
   ] - rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) <= 12.5))
2057.
2058.    {
2059.
2060.    contactpnt9 = contactpnt9 + 1;
2061.
2062.    if (contactpnt9 >= 4) {goto next9;}
2063.
2064.
2065.    if ( rightisosceles [qq9][3][1] >  rightisosceles [xx9][1][1] )
2066.
2067.    {
2068.
2069.
2070.    ydisplat [xx9][contactpnt9] = ydisplacement2  + (deluy [xx9][contact
   pnt9] -
    deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][0] - rightisosceles [xx9][0][0])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][0] -
    rightisosceles [qq9][0][0]));
2071.    xdisplat [xx9][contactpnt9] = xdisplacement2 + (delux [xx9][contactp
   nt9] -
    delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][1] - rightisosceles [xx9][0][1])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][1] -
   rightisosceles [qq9][0][1]));
2072.
2073.
2074.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2075.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2076.
2077.
2078.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2079.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2080.
2081.
2082.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
```

```
    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
ient/1000);


    yforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * sin (alpha1)) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha1)));
    xforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * cos (alpha1)) + ((fn [xx9][contactpnt9] + dn [xx9][contactpn
t9]) * sin (alpha1)));

    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);


    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
    y [xx9] = fysum [xx9][contactpnt9];
    resultfy [xx9] += y[xx9];


    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
    x [xx9] = fxsum [xx9][contactpnt9];
    resultfx [xx9] += x[xx9];


    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
es [qq9][3][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][3][1] -
     rightisosceles [xx9][0][1]));
    m [xx9] = msum [xx9][contactpnt9];
    resultm [xx9] += m[xx9];


    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx9] = udoty [xx9][contactpnt9];
    resultudoty [xx9] += udotysum [xx9];


    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


```

```
2123.    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
   t;
2124.    deluysum [xx9] = deluy [xx9][contactpnt9];
2125.    resultdeluy [xx9] += deluysum [xx9];
2126.
2127.
2128.    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
   t;
2129.    deluxsum [xx9] = delux [xx9][contactpnt9];
2130.    resultdelux [xx9] += deluxsum [xx9];
2131.
2132.    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
   crement;
2133.    delthetasum [xx9] = deltheta [xx9][contactpnt9];
2134.    resultdeltheta [xx9] += delthetasum [xx9];
2135.
2136.
2137.    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
2138.    uysum [xx9] = uy [xx9][contactpnt9];
2139.    resultuy [xx9]+= uysum [xx9];
2140.
2141.
2142.    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
2143.    uxsum [xx9] = ux [xx9][contactpnt9];
2144.    resultux [xx9] += uxsum [xx9];
2145.
2146.
2147.    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
2148.    thetasum [xx9] = theta [xx9][contactpnt9];
2149.    resulttheta [xx9] += thetasum [xx9];
2150.    alpha [xx9] = alpha1 + resulttheta [xx9];
2151.
2152.    }
2153.
2154.    else
2155.
2156.    {
2157.
2158.
2159.    ydisplat [xx9][contactpnt9] += ydisplacement2  + (deluy [xx9][contac
   tpnt9] -
     deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][0] - rightisosceles [xx9][0][0])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][0][0] -
     rightisosceles [qq9][0][0]));
2160.    xdisplat [xx9][contactpnt9] += xdisplacement2 + (delux [xx9][contact
   pnt9] -
     delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][1] - rightisosceles [xx9][0][1])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][1][1] -
   rightisosceles [qq9][0][1]));
2161.
```

```
2162.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2163.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2164.
2165.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2166.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2167.
2168.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
2169.    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
   ient/1000);
2170.
2171.    yforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * sin (alpha [xx9])) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha [xx9])));

2172.    xforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * cos (alpha [xx9])) + ((fn [xx9][contactpnt9] + dn [xx9][conta
   ctpnt9]) * sin (alpha [xx9])));
2173.
2174.    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
2175.    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);
2176.
2177.    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
2178.    x [xx9] = fxsum [xx9][contactpnt9];
2179.    resultfx [xx9] += x[xx9];
2180.
2181.
2182.    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
2183.    y [xx9] = fysum [xx9][contactpnt9];
2184.    resultfy [xx9] += y[xx9];
2185.
2186.
2187.    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
   es [qq9][1][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][1][1] -
     rightisosceles [xx9][0][1]));
2188.    m [xx9] = msum [xx9][contactpnt9];
2189.    resultm [xx9] += m[xx9];
2190.
2191.
2192.    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
2193.    udotysum [xx9] = udoty [xx9][contactpnt9];
2194.    resultudoty [xx9] += udotysum [xx9];
2195.
2196.
2197.    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
   ent)*1000/ dmass); //mm/sec
```

```
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
t;
    deluysum [xx9] = deluy [xx9][contactpnt9];
    resultdeluy [xx9] += deluysum [xx9];


    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
t;
    deluxsum [xx9] = delux [xx9][contactpnt9];
    resultdelux [xx9] += deluxsum [xx9];


    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
crement;
    delthetasum [xx9] = deltheta [xx9][contactpnt9];
    resultdeltheta [xx9] += delthetasum [xx9];


    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
    uysum [xx9] = uy [xx9][contactpnt9];
    resultuy [xx9]+= uysum [xx9];


    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
    uxsum [xx9] = ux [xx9][contactpnt9];
    resultux [xx9] += uxsum [xx9];


    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
    thetasum [xx9] = theta [xx9][contactpnt9];
    resulttheta [xx9] += thetasum [xx9];
    alpha [xx9] = alpha1 + resulttheta [xx9];

    }
    }

    }

    next9:
    ;}

```

```
2246.
2247.    int xx10 = 0, qq10 = 0, contactpnt10 = 0, l10, i10;
2248.
2249.
2250.    for (i10 = 0; i10 < isobox10; i10= i10+1)
2251.
2252.    {
2253.
2254.    xx10 = isocounterbox10[i10];
2255.
2256.    rightisosceles [xx10][0][0] = rightisosceles [xx10][0][0] + resultux
    [xx10] ;
2257.
2258.    rightisosceles [xx10][0][1] = rightisosceles [xx10][0][1] + resultuy
    [xx10] ;
2259.
2260.
2261.
2262.    for ( l10 = 0; l10 < isobox10; l10 = l10+1)
2263.
2264.    {
2265.
2266.
2267.    qq10 = isocounterbox10[l10];
2268.
2269.
2270.
2271.    if ( ((pow((rightisosceles [xx10][0][0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) > 0) && ((pow((rightisosceles [xx10][0]
   [0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) <= 12.5))
2272.
2273.    {
2274.
2275.    contactpnt10 = contactpnt10 + 1;
2276.
2277.    if (contactpnt10 >= 4) {goto next10;}
2278.
2279.
2280.    if ( rightisosceles [qq10][3][1] >  rightisosceles [xx10][1][1] )
2281.
2282.    {
2283.
2284.
2285.    ydisplat [xx10][contactpnt10] = ydisplacement2  + (deluy [xx10][cont
   actpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][0] -
    rightisosceles [qq10][0][0]));
```

```
2286.    xdisplat [xx10][contactpnt10] = xdisplacement2 + (delux [xx10][conta
   ctpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][1] -
   rightisosceles [qq10][0][1]));
2287.
2288.
2289.    delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2290.    delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2291.
2292.
2293.    fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2294.    fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2295.
2296.
2297.    dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2298.    ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2299.
2300.
2301.    yforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * sin (alpha1)) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha1)));
2302.    xforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * cos (alpha1)) + ((fn [xx10][contactpnt10] + dn [xx10][
   contactpnt10]) * sin (alpha1)));
2303.
2304.    yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2305.    xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2306.
2307.
2308.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2309.    y [xx10] = fysum [xx10][contactpnt10];
2310.    resultfy [xx10] += y[xx10];
2311.
2312.
2313.    fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2314.    x [xx10] = fxsum [xx10][contactpnt10];
2315.    resultfx [xx10] += x[xx10];
2316.
2317.
2318.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][3][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][3][1] -
    rightisosceles [xx10][0][1]));
2319.    m [xx10] = msum [xx10][contactpnt10];
```

```
2320.    resultm [xx10] += m[xx10];
2321.
2322.
2323.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement) *1000/ dmass); // mm/sec
2324.    udotysum [xx10] = udoty [xx10][contactpnt10];
2325.    resultudoty [xx10] += udotysum [xx10];
2326.
2327.
2328.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2329.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2330.    resultudotx [xx10] += udotxsum [xx10];
2331.
2332.
2333.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2334.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2335.    resultthetadot [xx10] += thetadotsum [xx10];
2336.
2337.
2338.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2339.    deluysum [xx10] = deluy [xx10][contactpnt10];
2340.    resultdeluy [xx10] += deluysum [xx10];
2341.
2342.
2343.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2344.    deluxsum [xx10] = delux [xx10][contactpnt10];
2345.    resultdelux [xx10] += deluxsum [xx10];
2346.
2347.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2348.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2349.    resultdeltheta [xx10] += delthetasum [xx10];
2350.
2351.
2352.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2353.    uysum [xx10] = uy [xx10][contactpnt10];
2354.    resultuy [xx10]+= uysum [xx10];
2355.
2356.
2357.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2358.    uxsum [xx10] = ux [xx10][contactpnt10];
2359.    resultux [xx10] += uxsum [xx10];
2360.
2361.
2362.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2363.    thetasum [xx10] = theta [xx10][contactpnt10];
2364.    resultthetа [xx10] += thetasum [xx10];
2365.    alpha [xx10] = alpha1 + resultthetа [xx10];
```

```
2366.
2367.     }
2368.
2369.     else
2370.
2371.     {
2372.
2373.
2374.     ydisplat [xx10][contactpnt10] += ydisplacement2  + (deluy [xx10][con
   tactpnt10] -
     deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][0] - rightisosceles [xx10][0][0])  -
     deltheta [qq10][contactpnt10] * (rightisosceles [qq10][0][0] -
     rightisosceles [qq10][0][0]));
2375.     xdisplat [xx10][contactpnt10] += xdisplacement2 + (delux [xx10][cont
   actpnt10] -
     delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][1] - rightisosceles [xx10][0][1])  -
     deltheta [qq10][contactpnt10] * (rightisosceles [qq10][1][1] -
   rightisosceles [qq10][0][1]));
2376.
2377.     delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2378.     delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2379.
2380.     fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2381.     fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2382.
2383.     dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2384.     ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2385.
2386.     yforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * sin (alpha [xx10])) -
     ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha [xx10])
   ));
2387.     xforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * cos (alpha [xx10])) + ((fn [xx10][contactpnt10] + dn [x
   x10][contactpnt10]) * sin (alpha [xx10])));
2388.
2389.     yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2390.     xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2391.
2392.     fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2393.     x [xx10] = fxsum [xx10][contactpnt10];
2394.     resultfx [xx10] += x[xx10];
2395.
2396.
```

```
2397.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2398.    y [xx10] = fysum [xx10][contactpnt10];
2399.    resultfy [xx10] += y[xx10];
2400.
2401.
2402.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][1][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][1][1] -
    rightisosceles [xx10][0][1]));
2403.    m [xx10] = msum [xx10][contactpnt10];
2404.    resultm [xx10] += m[xx10];
2405.
2406.
2407.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2408.    udotysum [xx10] = udoty [xx10][contactpnt10];
2409.    resultudoty [xx10] += udotysum [xx10];
2410.
2411.
2412.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2413.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2414.    resultudotx [xx10] += udotxsum [xx10];
2415.
2416.
2417.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2418.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2419.    resultthetadot [xx10] += thetadotsum [xx10];
2420.
2421.
2422.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2423.    deluysum [xx10] = deluy [xx10][contactpnt10];
2424.    resultdeluy [xx10] += deluysum [xx10];
2425.
2426.
2427.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2428.    deluxsum [xx10] = delux [xx10][contactpnt10];
2429.    resultdelux [xx10] += deluxsum [xx10];
2430.
2431.
2432.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2433.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2434.    resultdeltheta [xx10] += delthetasum [xx10];
2435.
2436.
2437.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2438.    uysum [xx10] = uy [xx10][contactpnt10];
2439.    resultuy [xx10]+= uysum [xx10];
```

```
2440.
2441.
2442.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2443.    uxsum [xx10] = ux [xx10][contactpnt10];
2444.    resultux [xx10] += uxsum [xx10];
2445.
2446.
2447.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2448.    thetasum [xx10] = theta [xx10][contactpnt10];
2449.    resulttheta [xx10] += thetasum [xx10];
2450.    alpha [xx10] = alpha1 + resulttheta [xx10];
2451.
2452.    }
2453.    }
2454.
2455.    }
2456.
2457.    next10:
2458.    ;}
2459.
2460.
2461.    int xx11 = 0, qq11 = 0, contactpnt11 = 0, l11, i11;
2462.
2463.
2464.    for (i11 = 0; i11 < isobox11; i11= i11+1)
2465.
2466.    {
2467.
2468.    xx11 = isocounterbox11[i11];
2469.
2470.    rightisosceles [xx11][0][0] = rightisosceles [xx11][0][0] + resultux
    [xx11] ;
2471.
2472.    rightisosceles [xx11][0][1] = rightisosceles [xx11][0][1] + resultuy
    [xx11] ;
2473.
2474.
2475.
2476.    for ( l11 = 0; l11 < isobox11; l11 = l11+1)
2477.
2478.    {
2479.
2480.
2481.    qq11 = isocounterbox11[l11];
2482.
2483.
2484.
2485.    if ( ((pow((rightisosceles [xx11][0][0] -
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) > 0) && ((pow((rightisosceles [xx11][0]
   [0] -
```

```
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) <= 12.5))
2486.
2487.    {
2488.
2489.    contactpnt11 = contactpnt11 + 1;
2490.
2491.    if (contactpnt11 >= 4) {goto next11;}
2492.
2493.
2494.    if ( rightisosceles [qq11][3][1] >  rightisosceles [xx11][1][1] )
2495.
2496.    {
2497.
2498.
2499.    ydisplat [xx11][contactpnt11] = ydisplacement2  + (deluy [xx11][cont
   actpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][0] -
     rightisosceles [qq11][0][0]));
2500.    xdisplat [xx11][contactpnt11] = xdisplacement2 + (delux [xx11][conta
   ctpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][1] -
   rightisosceles [qq11][0][1]));
2501.
2502.
2503.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2504.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2505.
2506.
2507.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
2508.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2509.
2510.
2511.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2512.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2513.
2514.
2515.    yforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * sin (alpha1)) -
     ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha1)));
```

```
2516.    xforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * cos (alpha1)) + ((fn [xx11][contactpnt11] + dn [xx11][
   contactpnt11]) * sin (alpha1)));
2517.
2518.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2519.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2520.
2521.
2522.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2523.    y [xx11] = fysum [xx11][contactpnt11];
2524.    resultfy [xx11] += y[xx11];
2525.
2526.
2527.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2528.    x [xx11] = fxsum [xx11][contactpnt11];
2529.    resultfx [xx11] += x[xx11];
2530.
2531.
2532.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][3][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][3][1] -
    rightisosceles [xx11][0][1]));
2533.    m [xx11] = msum [xx11][contactpnt11];
2534.    resultm [xx11] += m[xx11];
2535.
2536.
2537.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement) *1000/ dmass); // mm/sec
2538.    udotysum [xx11] = udoty [xx11][contactpnt11];
2539.    resultudoty [xx11] += udotysum [xx11];
2540.
2541.
2542.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2543.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2544.    resultudotx [xx11] += udotxsum [xx11];
2545.
2546.
2547.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2548.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
2549.    resultthetadot [xx11] += thetadotsum [xx11];
2550.
2551.
2552.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2553.    deluysum [xx11] = deluy [xx11][contactpnt11];
2554.    resultdeluy [xx11] += deluysum [xx11];
2555.
2556.
2557.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
```

```
2558.    deluxsum [xx11] = delux [xx11][contactpnt11];
2559.    resultdelux [xx11] += deluxsum [xx11];
2560.
2561.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2562.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2563.    resultdeltheta [xx11] += delthetasum [xx11];
2564.
2565.
2566.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2567.    uysum [xx11] = uy [xx11][contactpnt11];
2568.    resultuy [xx11]+= uysum [xx11];
2569.
2570.
2571.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2572.    uxsum [xx11] = ux [xx11][contactpnt11];
2573.    resultux [xx11] += uxsum [xx11];
2574.
2575.
2576.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2577.    thetasum [xx11] = theta [xx11][contactpnt11];
2578.    resulttheta [xx11] += thetasum [xx11];
2579.    alpha [xx11] = alpha1 + resulttheta [xx11];
2580.
2581.    }
2582.
2583.    else
2584.
2585.    {
2586.
2587.
2588.
2589.    ydisplat [xx11][contactpnt11] += ydisplacement2  + (deluy [xx11][con
   tactpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][0][0] -
     rightisosceles [qq11][0][0]));
2590.    xdisplat [xx11][contactpnt11] += xdisplacement2 + (delux [xx11][cont
   actpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][1][1] -
   rightisosceles [qq11][0][1]));
2591.
2592.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2593.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2594.
2595.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
2596.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2597.
2598.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2599.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2600.
2601.    yforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * sin (alpha [xx11])) -
    ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha [xx11])
   ));
2602.    xforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * cos (alpha [xx11])) + ((fn [xx11][contactpnt11] + dn [x
   x11][contactpnt11]) * sin (alpha [xx11])));
2603.
2604.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2605.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2606.
2607.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2608.    x [xx11] = fxsum [xx11][contactpnt11];
2609.    resultfx [xx11] += x[xx11];
2610.
2611.
2612.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2613.    y [xx11] = fysum [xx11][contactpnt11];
2614.    resultfy [xx11] += y[xx11];
2615.
2616.
2617.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][1][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][1][1] -
    rightisosceles [xx11][0][1]));
2618.    m [xx11] = msum [xx11][contactpnt11];
2619.    resultm [xx11] += m[xx11];
2620.
2621.
2622.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2623.    udotysum [xx11] = udoty [xx11][contactpnt11];
2624.    resultudoty [xx11] += udotysum [xx11];
2625.
2626.
2627.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2628.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2629.    resultudotx [xx11] += udotxsum [xx11];
2630.
2631.
2632.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2633.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
```

```
2634.    resultthetadot [xx11] += thetadotsum [xx11];
2635.
2636.
2637.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2638.    deluysum [xx11] = deluy [xx11][contactpnt11];
2639.    resultdeluy [xx11] += deluysum [xx11];
2640.
2641.
2642.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
2643.    deluxsum [xx11] = delux [xx11][contactpnt11];
2644.    resultdelux [xx11] += deluxsum [xx11];
2645.
2646.
2647.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2648.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2649.    resultdeltheta [xx11] += delthetasum [xx11];
2650.
2651.
2652.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2653.    uysum [xx11] = uy [xx11][contactpnt11];
2654.    resultuy [xx11]+= uysum [xx11];
2655.
2656.
2657.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2658.    uxsum [xx11] = ux [xx11][contactpnt11];
2659.    resultux [xx11] += uxsum [xx11];
2660.
2661.
2662.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2663.    thetasum [xx11] = theta [xx11][contactpnt11];
2664.    resulttheta [xx11] += thetasum [xx11];
2665.    alpha [xx11] = alpha1 + resulttheta [xx11];
2666.
2667.
2668.
2669.    }
2670.
2671.    }
2672.
2673.    }
2674.
2675.    next11:
2676.    ;}
2677.
2678.
2679.
2680.    int xx12 = 0, qq12 = 0, contactpnt12 = 0, l12, i12;
2681.
2682.
```

```
2683.    for (i12 = 0; i12 < isobox12; i12= i12+1)
2684.
2685.    {
2686.
2687.    xx12 = isocounterbox12[i12];
2688.
2689.    rightisosceles [xx12][0][0] = rightisosceles [xx12][0][0] + resultux
    [xx12] ;
2690.
2691.    rightisosceles [xx12][0][1] = rightisosceles [xx12][0][1] + resultuy
    [xx12] ;
2692.
2693.
2694.
2695.    for ( l12 = 0; l12 < isobox12; l12 = l12+1)
2696.
2697.    {
2698.
2699.
2700.    qq12 = isocounterbox12[l12];
2701.
2702.
2703.
2704.    if ( ((pow((rightisosceles [xx12][0][0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) > 0) && ((pow((rightisosceles [xx12][0]
   [0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) <= 12.5))
2705.
2706.    {
2707.
2708.    contactpnt12 = contactpnt12 + 1;
2709.
2710.    if (contactpnt12 >= 4) {goto next12;}
2711.
2712.
2713.    if ( rightisosceles [qq12][3][1] >  rightisosceles [xx12][1][1] )
2714.
2715.    {
2716.
2717.
2718.    ydisplat [xx12][contactpnt12] = ydisplacement2  + (deluy [xx12][cont
   actpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][0] -
    rightisosceles [qq12][0][0]));
2719.    xdisplat [xx12][contactpnt12] = xdisplacement2 + (delux [xx12][conta
   ctpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][1] - rightisosceles [xx12][0][1])  -
```

```
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][1] -
   rightisosceles [qq12][0][1]));
2720.
2721.
2722.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2723.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2724.
2725.
2726.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2727.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2728.
2729.
2730.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2731.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2732.
2733.
2734.    yforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * sin (alpha1)) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha1)));
2735.    xforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * cos (alpha1)) + ((fn [xx12][contactpnt12] + dn [xx12][
   contactpnt12]) * sin (alpha1)));
2736.
2737.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2738.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2739.
2740.
2741.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2742.    y [xx12] = fysum [xx12][contactpnt12];
2743.    resultfy [xx12] += y[xx12];
2744.
2745.
2746.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2747.    x [xx12] = fxsum [xx12][contactpnt12];
2748.    resultfx [xx12] += x[xx12];
2749.
2750.
2751.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][3][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][3][1] -
    rightisosceles [xx12][0][1]));
2752.    m [xx12] = msum [xx12][contactpnt12];
2753.    resultm [xx12] += m[xx12];
2754.
2755.
```

```
2756.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement) *1000/ dmass); // mm/sec
2757.    udotysum [xx12] = udoty [xx12][contactpnt12];
2758.    resultudoty [xx12] += udotysum [xx12];
2759.
2760.
2761.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2762.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2763.    resultudotx [xx12] += udotxsum [xx12];
2764.
2765.
2766.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2767.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2768.    resultthetadot [xx12] += thetadotsum [xx12];
2769.
2770.
2771.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2772.    deluysum [xx12] = deluy [xx12][contactpnt12];
2773.    resultdeluy [xx12] += deluysum [xx12];
2774.
2775.
2776.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2777.    deluxsum [xx12] = delux [xx12][contactpnt12];
2778.    resultdelux [xx12] += deluxsum [xx12];
2779.
2780.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2781.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2782.    resultdeltheta [xx12] += delthetasum [xx12];
2783.
2784.
2785.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2786.    uysum [xx12] = uy [xx12][contactpnt12];
2787.    resultuy [xx12]+= uysum [xx12];
2788.
2789.
2790.    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
2791.    uxsum [xx12] = ux [xx12][contactpnt12];
2792.    resultux [xx12] += uxsum [xx12];
2793.
2794.
2795.    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
2796.    thetasum [xx12] = theta [xx12][contactpnt12];
2797.    resultttheta [xx12] += thetasum [xx12];
2798.    alpha [xx12] = alpha1 + resultttheta [xx12];
2799.
2800.    }
2801.
```

```
2802.    else
2803.
2804.    {
2805.
2806.
2807.
2808.    ydisplat [xx12][contactpnt12] += ydisplacement2  + (deluy [xx12][con
   tactpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][0][0] -
    rightisosceles [qq12][0][0]));
2809.    xdisplat [xx12][contactpnt12] += xdisplacement2 + (delux [xx12][cont
   actpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][1] - rightisosceles [xx12][0][1])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][1][1] -
   rightisosceles [qq12][0][1]));
2810.
2811.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2812.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2813.
2814.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2815.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2816.
2817.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2818.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2819.
2820.    yforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * sin (alpha [xx12])) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha [xx12])
   ));
2821.    xforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * cos (alpha [xx12])) + ((fn [xx12][contactpnt12] + dn [x
   x12][contactpnt12]) * sin (alpha [xx12])));
2822.
2823.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2824.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2825.
2826.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2827.    x [xx12] = fxsum [xx12][contactpnt12];
2828.    resultfx [xx12] += x[xx12];
2829.
2830.
2831.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2832.    y [xx12] = fysum [xx12][contactpnt12];
```

```
2833.    resultfy [xx12] += y[xx12];
2834.
2835.
2836.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][1][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][1][1] -
    rightisosceles [xx12][0][1]));
2837.    m [xx12] = msum [xx12][contactpnt12];
2838.    resultm [xx12] += m[xx12];
2839.
2840.
2841.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2842.    udotysum [xx12] = udoty [xx12][contactpnt12];
2843.    resultudoty [xx12] += udotysum [xx12];
2844.
2845.
2846.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2847.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2848.    resultudotx [xx12] += udotxsum [xx12];
2849.
2850.
2851.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2852.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2853.    resultthetadot [xx12] += thetadotsum [xx12];
2854.
2855.
2856.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2857.    deluysum [xx12] = deluy [xx12][contactpnt12];
2858.    resultdeluy [xx12] += deluysum [xx12];
2859.
2860.
2861.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2862.    deluxsum [xx12] = delux [xx12][contactpnt12];
2863.    resultdelux [xx12] += deluxsum [xx12];
2864.
2865.
2866.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2867.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2868.    resultdeltheta [xx12] += delthetasum [xx12];
2869.
2870.
2871.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2872.    uysum [xx12] = uy [xx12][contactpnt12];
2873.    resultuy [xx12]+= uysum [xx12];
2874.
2875.
```

```
    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
    uxsum [xx12] = ux [xx12][contactpnt12];
    resultux [xx12] += uxsum [xx12];


    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
    thetasum [xx12] = theta [xx12][contactpnt12];
    resulttheta [xx12] += thetasum [xx12];
    alpha [xx12] = alpha1 + resulttheta [xx12];

    }
    }

    }

    next12:
    ;}


    int xx13 = 0, qq13 = 0, contactpnt13 = 0, l13, i13;


    for (i13 = 0; i13 < isobox13; i13= i13+1)

    {

    xx13 = isocounterbox13[i13];


    rightisosceles [xx13][0][0] = rightisosceles [xx13][0][0] + resultux
    [xx13] ;

    rightisosceles [xx13][0][1] = rightisosceles [xx13][0][1] + resultuy
    [xx13] ;


    for ( l13 = 0; l13 < isobox13; l13 = l13+1)

    {


    qq13 = isocounterbox13[l13];


    if ( ((pow((rightisosceles [xx13][0][0] -
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) > 0) && ((pow((rightisosceles [xx13][0]
   [0] -
```

```
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) <= 12.5))
2923.
2924.    {
2925.
2926.    contactpnt13 = contactpnt13 + 1;
2927.
2928.    if (contactpnt13 >= 4) {goto next13;}
2929.
2930.
2931.    if ( rightisosceles [qq13][3][1] >  rightisosceles [xx13][1][1] )
2932.
2933.    {
2934.
2935.
2936.    ydisplat [xx13][contactpnt13] = ydisplacement2  + (deluy [xx13][cont
   actpnt13] -
     deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][0] - rightisosceles [xx13][0][0])  -
     deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][0] -
     rightisosceles [qq13][0][0]));
2937.    xdisplat [xx13][contactpnt13] = xdisplacement2 + (delux [xx13][conta
   ctpnt13] -
     delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][1] - rightisosceles [xx13][0][1])  -
     deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][1] -
   rightisosceles [qq13][0][1]));
2938.
2939.
2940.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
2941.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
2942.
2943.
2944.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
2945.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
2946.
2947.
2948.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
2949.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
2950.
2951.
2952.    yforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * sin (alpha1)) -
     ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha1)));
```

```
2953.    xforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * cos (alpha1)) + ((fn [xx13][contactpnt13] + dn [xx13][
   contactpnt13]) * sin (alpha1)));
2954.
2955.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
2956.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
2957.
2958.
2959.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
2960.    y [xx13] = fysum [xx13][contactpnt13];
2961.    resultfy [xx13] += y[xx13];
2962.
2963.
2964.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
2965.    x [xx13] = fxsum [xx13][contactpnt13];
2966.    resultfx [xx13] += x[xx13];
2967.
2968.
2969.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][3][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][3][1] -
    rightisosceles [xx13][0][1]));
2970.    m [xx13] = msum [xx13][contactpnt13];
2971.    resultm [xx13] += m[xx13];
2972.
2973.
2974.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement) *1000/ dmass); // mm/sec
2975.    udotysum [xx13] = udoty [xx13][contactpnt13];
2976.    resultudoty [xx13] += udotysum [xx13];
2977.
2978.
2979.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
2980.    udotxsum [xx13] = udotx [xx13][contactpnt13];
2981.    resultudotx [xx13] += udotxsum [xx13];
2982.
2983.
2984.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
2985.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
2986.    resultthetadot [xx13] += thetadotsum [xx13];
2987.
2988.
2989.    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
   ement;
2990.    deluysum [xx13] = deluy [xx13][contactpnt13];
2991.    resultdeluy [xx13] += deluysum [xx13];
2992.
2993.
2994.    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
   ement;
```

```
2995.    deluxsum [xx13] = delux [xx13][contactpnt13];
2996.    resultdelux [xx13] += deluxsum [xx13];
2997.
2998.    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
   e_increment;
2999.    delthetasum [xx13] = deltheta [xx13][contactpnt13];
3000.    resultdeltheta [xx13] += delthetasum [xx13];
3001.
3002.
3003.    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
3004.    uysum [xx13] = uy [xx13][contactpnt13];
3005.    resultuy [xx13]+= uysum [xx13];
3006.
3007.
3008.    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
3009.    uxsum [xx13] = ux [xx13][contactpnt13];
3010.    resultux [xx13] += uxsum [xx13];
3011.
3012.
3013.    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
3014.    thetasum [xx13] = theta [xx13][contactpnt13];
3015.    resulttheta [xx13] += thetasum [xx13];
3016.    alpha [xx13] = alpha1 + resulttheta [xx13];
3017.
3018.    }
3019.
3020.    else
3021.
3022.    {
3023.
3024.
3025.
3026.    ydisplat [xx13][contactpnt13] += ydisplacement2  + (deluy [xx13][con
   tactpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][0][0] -
    rightisosceles [qq13][0][0]));
3027.    xdisplat [xx13][contactpnt13] += xdisplacement2 + (delux [xx13][cont
   actpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][1][1] -
   rightisosceles [qq13][0][1]));
3028.
3029.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
3030.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
3031.
3032.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
3033.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
3034.
3035.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
3036.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
3037.
3038.    yforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * sin (alpha [xx13])) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha [xx13])
   ));
3039.    xforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * cos (alpha [xx13])) + ((fn [xx13][contactpnt13] + dn [x
   x13][contactpnt13]) * sin (alpha [xx13])));
3040.
3041.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
3042.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
3043.
3044.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
3045.    x [xx13] = fxsum [xx13][contactpnt13];
3046.    resultfx [xx13] += x[xx13];
3047.
3048.
3049.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
3050.    y [xx13] = fysum [xx13][contactpnt13];
3051.    resultfy [xx13] += y[xx13];
3052.
3053.
3054.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][1][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][1][1] -
    rightisosceles [xx13][0][1]));
3055.    m [xx13] = msum [xx13][contactpnt13];
3056.    resultm [xx13] += m[xx13];
3057.
3058.
3059.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3060.    udotysum [xx13] = udoty [xx13][contactpnt13];
3061.    resultudoty [xx13] += udotysum [xx13];
3062.
3063.
3064.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
3065.    udotxsum [xx13] = udotx [xx13][contactpnt13];
3066.    resultudotx [xx13] += udotxsum [xx13];
3067.
3068.
3069.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
3070.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
```

```
3071.    resultthetadot [xx13] += thetadotsum [xx13];
3072.
3073.
3074.    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
   ement;
3075.    deluysum [xx13] = deluy [xx13][contactpnt13];
3076.    resultdeluy [xx13] += deluysum [xx13];
3077.
3078.
3079.    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
   ement;
3080.    deluxsum [xx13] = delux [xx13][contactpnt13];
3081.    resultdelux [xx13] += deluxsum [xx13];
3082.
3083.
3084.    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
   e_increment;
3085.    delthetasum [xx13] = deltheta [xx13][contactpnt13];
3086.    resultdeltheta [xx13] += delthetasum [xx13];
3087.
3088.
3089.    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
3090.    uysum [xx13] = uy [xx13][contactpnt13];
3091.    resultuy [xx13]+= uysum [xx13];
3092.
3093.
3094.    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
3095.    uxsum [xx13] = ux [xx13][contactpnt13];
3096.    resultux [xx13] += uxsum [xx13];
3097.
3098.
3099.    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
3100.    thetasum [xx13] = theta [xx13][contactpnt13];
3101.    resulttheta [xx13] += thetasum [xx13];
3102.    alpha [xx13] = alpha1 + resulttheta [xx13];
3103.
3104.
3105.
3106.    }
3107.
3108.    }
3109.
3110.    }
3111.
3112.    next13:
3113.    ;}
3114.
3115.
3116.
3117.    int xx14 = 0, qq14 = 0, contactpnt14 = 0, l14, i14;
3118.
3119.
```

```
3120.    for (i14 = 0; i14 < isobox14; i14= i14+1)
3121.
3122.    {
3123.
3124.    xx14 = isocounterbox14[i14];
3125.
3126.    rightisosceles [xx14][0][0] = rightisosceles [xx14][0][0] + resultux
    [xx14] ;
3127.
3128.    rightisosceles [xx14][0][1] = rightisosceles [xx14][0][1] + resultuy
    [xx14] ;
3129.
3130.
3131.
3132.    for ( l14 = 0; l14 < isobox14; l14 = l14+1)
3133.
3134.    {
3135.
3136.
3137.    qq14 = isocounterbox14[l14];
3138.
3139.
3140.
3141.    if ( ((pow((rightisosceles [xx14][0][0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) > 0) && ((pow((rightisosceles [xx14][0]
   [0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) <= 12.5))
3142.
3143.    {
3144.
3145.    contactpnt14 = contactpnt14 + 1;
3146.
3147.    if (contactpnt14 >= 4) {goto next14;}
3148.
3149.
3150.    if ( rightisosceles [qq14][3][1] >  rightisosceles [xx14][1][1] )
3151.
3152.    {
3153.
3154.
3155.    ydisplat [xx14][contactpnt14] = ydisplacement2  + (deluy [xx14][cont
   actpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][0] -
    rightisosceles [qq14][0][0]));
3156.    xdisplat [xx14][contactpnt14] = xdisplacement2 + (delux [xx14][conta
   ctpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][1] - rightisosceles [xx14][0][1])  -
```

```
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][1] -
   rightisosceles [qq14][0][1]));
3157.
3158.
3159.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3160.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3161.
3162.
3163.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3164.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3165.
3166.
3167.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3168.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3169.
3170.
3171.    yforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * sin (alpha1)) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha1)));
3172.    xforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * cos (alpha1)) + ((fn [xx14][contactpnt14] + dn [xx14][
   contactpnt14]) * sin (alpha1)));
3173.
3174.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3175.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3176.
3177.
3178.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3179.    y [xx14] = fysum [xx14][contactpnt14];
3180.    resultfy [xx14] += y[xx14];
3181.
3182.
3183.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3184.    x [xx14] = fxsum [xx14][contactpnt14];
3185.    resultfx [xx14] += x[xx14];
3186.
3187.
3188.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][3][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][3][1] -
    rightisosceles [xx14][0][1]));
3189.    m [xx14] = msum [xx14][contactpnt14];
3190.    resultm [xx14] += m[xx14];
3191.
3192.
```

```
3193.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement) *1000/ dmass); // mm/sec
3194.    udotysum [xx14] = udoty [xx14][contactpnt14];
3195.    resultudoty [xx14] += udotysum [xx14];
3196.
3197.
3198.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3199.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3200.    resultudotx [xx14] += udotxsum [xx14];
3201.
3202.
3203.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3204.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3205.    resultthetadot [xx14] += thetadotsum [xx14];
3206.
3207.
3208.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3209.    deluysum [xx14] = deluy [xx14][contactpnt14];
3210.    resultdeluy [xx14] += deluysum [xx14];
3211.
3212.
3213.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3214.    deluxsum [xx14] = delux [xx14][contactpnt14];
3215.    resultdelux [xx14] += deluxsum [xx14];
3216.
3217.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3218.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3219.    resultdeltheta [xx14] += delthetasum [xx14];
3220.
3221.
3222.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3223.    uysum [xx14] = uy [xx14][contactpnt14];
3224.    resultuy [xx14]+= uysum [xx14];
3225.
3226.
3227.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3228.    uxsum [xx14] = ux [xx14][contactpnt14];
3229.    resultux [xx14] += uxsum [xx14];
3230.
3231.
3232.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3233.    thetasum [xx14] = theta [xx14][contactpnt14];
3234.    resultttheta [xx14] += thetasum [xx14];
3235.    alpha [xx14] = alpha1 + resultttheta [xx14];
3236.
3237.    }
3238.
```

```
3239.    else
3240.
3241.    {
3242.
3243.
3244.
3245.    ydisplat [xx14][contactpnt14] += ydisplacement2  + (deluy [xx14][con
   tactpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][0][0] -
    rightisosceles [qq14][0][0]));
3246.    xdisplat [xx14][contactpnt14] += xdisplacement2 + (delux [xx14][cont
   actpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][1] - rightisosceles [xx14][0][1])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][1][1] -
   rightisosceles [qq14][0][1]));
3247.
3248.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3249.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3250.
3251.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3252.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3253.
3254.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3255.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3256.
3257.    yforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * sin (alpha [xx14])) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha [xx14])
   ));
3258.    xforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * cos (alpha [xx14])) + ((fn [xx14][contactpnt14] + dn [x
   x14][contactpnt14]) * sin (alpha [xx14])));
3259.
3260.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3261.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3262.
3263.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3264.    x [xx14] = fxsum [xx14][contactpnt14];
3265.    resultfx [xx14] += x[xx14];
3266.
3267.
3268.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3269.    y [xx14] = fysum [xx14][contactpnt14];
```

```
3270.    resultfy [xx14] += y[xx14];
3271.
3272.
3273.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][1][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][1][1] -
    rightisosceles [xx14][0][1]));
3274.    m [xx14] = msum [xx14][contactpnt14];
3275.    resultm [xx14] += m[xx14];
3276.
3277.
3278.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3279.    udotysum [xx14] = udoty [xx14][contactpnt14];
3280.    resultudoty [xx14] += udotysum [xx14];
3281.
3282.
3283.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3284.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3285.    resultudotx [xx14] += udotxsum [xx14];
3286.
3287.
3288.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3289.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3290.    resultthetadot [xx14] += thetadotsum [xx14];
3291.
3292.
3293.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3294.    deluysum [xx14] = deluy [xx14][contactpnt14];
3295.    resultdeluy [xx14] += deluysum [xx14];
3296.
3297.
3298.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3299.    deluxsum [xx14] = delux [xx14][contactpnt14];
3300.    resultdelux [xx14] += deluxsum [xx14];
3301.
3302.
3303.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3304.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3305.    resultdeltheta [xx14] += delthetasum [xx14];
3306.
3307.
3308.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3309.    uysum [xx14] = uy [xx14][contactpnt14];
3310.    resultuy [xx14]+= uysum [xx14];
3311.
3312.
```

```
3313.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3314.    uxsum [xx14] = ux [xx14][contactpnt14];
3315.    resultux [xx14] += uxsum [xx14];
3316.
3317.
3318.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3319.    thetasum [xx14] = theta [xx14][contactpnt14];
3320.    resulttheta [xx14] += thetasum [xx14];
3321.    alpha [xx14] = alpha1 + resulttheta [xx14];
3322.
3323.    }
3324.    }
3325.
3326.    }
3327.
3328.    next14:
3329.    ;}
3330.
3331.
3332.
3333.    int xx15 = 0, qq15 = 0, contactpnt15 = 0, l15, i15;
3334.
3335.
3336.    for (i15 = 0; i15 < isobox15; i15= i15+1)
3337.
3338.    {
3339.
3340.    xx15 = isocounterbox15[i15];
3341.
3342.    rightisosceles [xx15][0][0] = rightisosceles [xx15][0][0] + resultux
    [xx15] ;
3343.
3344.    rightisosceles [xx15][0][1] = rightisosceles [xx15][0][1] + resultuy
    [xx15] ;
3345.
3346.
3347.
3348.    for ( l15 = 0; l15 < isobox15; l15 = l15+1)
3349.
3350.    {
3351.
3352.
3353.    qq15 = isocounterbox15[l15];
3354.
3355.
3356.
3357.    if ( ((pow((rightisosceles [xx15][0][0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) > 0) && ((pow((rightisosceles [xx15][0]
   [0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) <= 12.5))
```

```
3358.
3359.    {
3360.
3361.    contactpnt15 = contactpnt15 + 1;
3362.
3363.    if (contactpnt15 >= 4) {goto next15;}
3364.
3365.
3366.    if ( rightisosceles [qq15][3][1] >  rightisosceles [xx15][1][1] )
3367.
3368.    {
3369.
3370.
3371.    ydisplat [xx15][contactpnt15] = ydisplacement2  + (deluy [xx15][cont
   actpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][0] -
    rightisosceles [qq15][0][0]));
3372.    xdisplat [xx15][contactpnt15] = xdisplacement2 + (delux [xx15][conta
   ctpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][1] -
   rightisosceles [qq15][0][1]));
3373.
3374.
3375.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3376.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3377.
3378.
3379.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3380.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3381.
3382.
3383.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3384.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3385.
3386.
3387.    yforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * sin (alpha1)) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha1)));
3388.    xforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * cos (alpha1)) + ((fn [xx15][contactpnt15] + dn [xx15][
   contactpnt15]) * sin (alpha1)));
3389.
```

```
3390.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3391.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3392.
3393.
3394.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3395.    y [xx15] = fysum [xx15][contactpnt15];
3396.    resultfy [xx15] += y[xx15];
3397.
3398.
3399.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3400.    x [xx15] = fxsum [xx15][contactpnt15];
3401.    resultfx [xx15] += x[xx15];
3402.
3403.
3404.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][3][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][3][1] -
    rightisosceles [xx15][0][1]));
3405.    m [xx15] = msum [xx15][contactpnt15];
3406.    resultm [xx15] += m[xx15];
3407.
3408.
3409.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement) *1000/ dmass); // mm/sec
3410.    udotysum [xx15] = udoty [xx15][contactpnt15];
3411.    resultudoty [xx15] += udotysum [xx15];
3412.
3413.
3414.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3415.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3416.    resultudotx [xx15] += udotxsum [xx15];
3417.
3418.
3419.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3420.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3421.    resultthetadot [xx15] += thetadotsum [xx15];
3422.
3423.
3424.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3425.    deluysum [xx15] = deluy [xx15][contactpnt15];
3426.    resultdeluy [xx15] += deluysum [xx15];
3427.
3428.
3429.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3430.    deluxsum [xx15] = delux [xx15][contactpnt15];
3431.    resultdelux [xx15] += deluxsum [xx15];
3432.
```

```
3433.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3434.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3435.    resultdeltheta [xx15] += delthetasum [xx15];
3436.
3437.
3438.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3439.    uysum [xx15] = uy [xx15][contactpnt15];
3440.    resultuy [xx15]+= uysum [xx15];
3441.
3442.
3443.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3444.    uxsum [xx15] = ux [xx15][contactpnt15];
3445.    resultux [xx15] += uxsum [xx15];
3446.
3447.
3448.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3449.    thetasum [xx15] = theta [xx15][contactpnt15];
3450.    resulttheta [xx15] += thetasum [xx15];
3451.    alpha [xx15] = alpha1 + resulttheta [xx15];
3452.
3453.    }
3454.
3455.    else
3456.
3457.    {
3458.
3459.
3460.
3461.    ydisplat [xx15][contactpnt15] += ydisplacement2  + (deluy [xx15][con
   tactpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][0][0] -
    rightisosceles [qq15][0][0]));
3462.    xdisplat [xx15][contactpnt15] += xdisplacement2 + (delux [xx15][cont
   actpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][1][1] -
   rightisosceles [qq15][0][1]));
3463.
3464.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3465.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3466.
3467.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3468.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3469.
```

```
3470.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3471.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3472.
3473.    yforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * sin (alpha [xx15])) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha [xx15])
   ));
3474.    xforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * cos (alpha [xx15])) + ((fn [xx15][contactpnt15] + dn [x
   x15][contactpnt15]) * sin (alpha [xx15])));
3475.
3476.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3477.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3478.
3479.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3480.    x [xx15] = fxsum [xx15][contactpnt15];
3481.    resultfx [xx15] += x[xx15];
3482.
3483.
3484.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3485.    y [xx15] = fysum [xx15][contactpnt15];
3486.    resultfy [xx15] += y[xx15];
3487.
3488.
3489.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][1][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][1][1] -
    rightisosceles [xx15][0][1]));
3490.    m [xx15] = msum [xx15][contactpnt15];
3491.    resultm [xx15] += m[xx15];
3492.
3493.
3494.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3495.    udotysum [xx15] = udoty [xx15][contactpnt15];
3496.    resultudoty [xx15] += udotysum [xx15];
3497.
3498.
3499.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3500.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3501.    resultudotx [xx15] += udotxsum [xx15];
3502.
3503.
3504.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3505.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3506.    resultthetadot [xx15] += thetadotsum [xx15];
3507.
3508.
```

```
3509.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3510.    deluysum [xx15] = deluy [xx15][contactpnt15];
3511.    resultdeluy [xx15] += deluysum [xx15];
3512.
3513.
3514.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3515.    deluxsum [xx15] = delux [xx15][contactpnt15];
3516.    resultdelux [xx15] += deluxsum [xx15];
3517.
3518.
3519.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3520.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3521.    resultdeltheta [xx15] += delthetasum [xx15];
3522.
3523.
3524.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3525.    uysum [xx15] = uy [xx15][contactpnt15];
3526.    resultuy [xx15]+= uysum [xx15];
3527.
3528.
3529.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3530.    uxsum [xx15] = ux [xx15][contactpnt15];
3531.    resultux [xx15] += uxsum [xx15];
3532.
3533.
3534.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3535.    thetasum [xx15] = theta [xx15][contactpnt15];
3536.    resulttheta [xx15] += thetasum [xx15];
3537.    alpha [xx15] = alpha1 + resulttheta [xx15];
3538.
3539.    }
3540.    }
3541.
3542.    }
3543.
3544.    next15:
3545.    ;}
3546.
3547.
3548.
3549.    int xx16 = 0, qq16 = 0, contactpnt16 = 0, l16, i16;
3550.
3551.
3552.    for (i16 = 0; i16 < isobox16; i16= i16+1)
3553.
3554.    {
3555.
3556.    xx16 = isocounterbox16[i16];
3557.
```

```
3558.
3559.
3560.    rightisosceles [xx16][0][0] = rightisosceles [xx16][0][0] + resultux
    [xx16] ;
3561.
3562.    rightisosceles [xx16][0][1] = rightisosceles [xx16][0][1] + resultuy
    [xx16] ;
3563.
3564.
3565.
3566.    for ( l16 = 0; l16 < isobox16; l16 = l16+1)
3567.
3568.    {
3569.
3570.
3571.    qq16 = isocounterbox16[l16];
3572.
3573.
3574.
3575.    if ( ((pow((rightisosceles [xx16][0][0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) > 0) && ((pow((rightisosceles [xx16][0]
   [0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) <= 12.5))
3576.
3577.    {
3578.
3579.    contactpnt16 = contactpnt16 + 1;
3580.
3581.    if (contactpnt16 >= 4) {goto next16;}
3582.
3583.
3584.    if ( rightisosceles [qq16][3][1] >  rightisosceles [xx16][1][1] )
3585.
3586.    {
3587.
3588.
3589.    ydisplat [xx16][contactpnt16] = ydisplacement2  + (deluy [xx16][cont
   actpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][0] -
    rightisosceles [qq16][0][0]));
3590.    xdisplat [xx16][contactpnt16] = xdisplacement2 + (delux [xx16][conta
   ctpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][1] -
   rightisosceles [qq16][0][1]));
3591.
3592.
```

```
3593.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3594.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3595.
3596.
3597.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3598.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3599.
3600.
3601.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3602.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3603.
3604.
3605.    yforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * sin (alpha1)) -
     ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha1)));
3606.    xforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * cos (alpha1)) + ((fn [xx16][contactpnt16] + dn [xx16][
   contactpnt16]) * sin (alpha1)));
3607.
3608.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3609.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3610.
3611.
3612.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3613.    y [xx16] = fysum [xx16][contactpnt16];
3614.    resultfy [xx16] += y[xx16];
3615.
3616.
3617.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3618.    x [xx16] = fxsum [xx16][contactpnt16];
3619.    resultfx [xx16] += x[xx16];
3620.
3621.
3622.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][3][0] - rightisosceles [xx16][0][0]) -
     xforce [xx16][contactpnt16] * ( rightisosceles [qq16][3][1] -
     rightisosceles [xx16][0][1]));
3623.    m [xx16] = msum [xx16][contactpnt16];
3624.    resultm [xx16] += m[xx16];
3625.
3626.
3627.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement) *1000/ dmass); // mm/sec
3628.    udotysum [xx16] = udoty [xx16][contactpnt16];
3629.    resultudoty [xx16] += udotysum [xx16];
3630.
```

```
3631.
3632.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3633.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3634.    resultudotx [xx16] += udotxsum [xx16];
3635.
3636.
3637.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3638.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3639.    resultthetadot [xx16] += thetadotsum [xx16];
3640.
3641.
3642.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3643.    deluysum [xx16] = deluy [xx16][contactpnt16];
3644.    resultdeluy [xx16] += deluysum [xx16];
3645.
3646.
3647.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3648.    deluxsum [xx16] = delux [xx16][contactpnt16];
3649.    resultdelux [xx16] += deluxsum [xx16];
3650.
3651.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3652.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3653.    resultdeltheta [xx16] += delthetasum [xx16];
3654.
3655.
3656.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3657.    uysum [xx16] = uy [xx16][contactpnt16];
3658.    resultuy [xx16]+= uysum [xx16];
3659.
3660.
3661.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3662.    uxsum [xx16] = ux [xx16][contactpnt16];
3663.    resultux [xx16] += uxsum [xx16];
3664.
3665.
3666.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3667.    thetasum [xx16] = theta [xx16][contactpnt16];
3668.    resultttheta [xx16] += thetasum [xx16];
3669.    alpha [xx16] = alpha1 + resultttheta [xx16];
3670.
3671.    }
3672.
3673.    else
3674.
3675.    {
3676.
3677.
```

```
3678.
3679.    ydisplat [xx16][contactpnt16] += ydisplacement2  + (deluy [xx16][con
   tactpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][0][0] -
    rightisosceles [qq16][0][0]));
3680.    xdisplat [xx16][contactpnt16] += xdisplacement2 + (delux [xx16][cont
   actpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][1][1] -
   rightisosceles [qq16][0][1]));
3681.
3682.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3683.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3684.
3685.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3686.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3687.
3688.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3689.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3690.
3691.    yforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * sin (alpha [xx16])) -
    ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha [xx16])
   ));
3692.    xforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * cos (alpha [xx16])) + ((fn [xx16][contactpnt16] + dn [x
   x16][contactpnt16]) * sin (alpha [xx16])));
3693.
3694.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3695.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3696.
3697.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3698.    x [xx16] = fxsum [xx16][contactpnt16];
3699.    resultfx [xx16] += x[xx16];
3700.
3701.
3702.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3703.    y [xx16] = fysum [xx16][contactpnt16];
3704.    resultfy [xx16] += y[xx16];
3705.
3706.
3707.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][1][0] - rightisosceles [xx16][0][0]) -
```

```
    xforce [xx16][contactpnt16] * ( rightisosceles [qq16][1][1] -
    rightisosceles [xx16][0][1]));
3708.    m [xx16] = msum [xx16][contactpnt16];
3709.    resultm [xx16] += m[xx16];
3710.
3711.
3712.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3713.    udotysum [xx16] = udoty [xx16][contactpnt16];
3714.    resultudoty [xx16] += udotysum [xx16];
3715.
3716.
3717.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3718.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3719.    resultudotx [xx16] += udotxsum [xx16];
3720.
3721.
3722.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3723.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3724.    resultthetadot [xx16] += thetadotsum [xx16];
3725.
3726.
3727.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3728.    deluysum [xx16] = deluy [xx16][contactpnt16];
3729.    resultdeluy [xx16] += deluysum [xx16];
3730.
3731.
3732.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3733.    deluxsum [xx16] = delux [xx16][contactpnt16];
3734.    resultdelux [xx16] += deluxsum [xx16];
3735.
3736.
3737.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3738.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3739.    resultdeltheta [xx16] += delthetasum [xx16];
3740.
3741.
3742.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3743.    uysum [xx16] = uy [xx16][contactpnt16];
3744.    resultuy [xx16]+= uysum [xx16];
3745.
3746.
3747.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3748.    uxsum [xx16] = ux [xx16][contactpnt16];
3749.    resultux [xx16] += uxsum [xx16];
3750.
3751.
```

```
3752.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3753.    thetasum [xx16] = theta [xx16][contactpnt16];
3754.    resulttheta [xx16] += thetasum [xx16];
3755.    alpha [xx16] = alpha1 + resulttheta [xx16];
3756.
3757.
3758.
3759.    }
3760.
3761.    }
3762.
3763.    }
3764.
3765.    next16:
3766.    ;}
3767.
3768.
3769.    ;}
3770.
3771.    display:
3772.
3773.
3774.     a_file<< dmass;
3775.
3776.    return 0;
3777.    }
```

Appendix F:

Code for Tailings-DEM (Series 3)

```
# include <iostream>
# include <cmath>
# include <fstream>
#include <numeric>

using namespace std;


 struct geometricalshapes  {  // subroutine for assigning values to the particle geometrical shapes
 float side1, side2, side3, side4, side5;
 float angle1, angle2, angle3, angle4, angle5;
 float height, base, centroidx, centroidy, momentofinertiax, momentofinertiay, area;

};

float ddamping_coefficient, dk;
float dstiffness_coefficient;
float dtime_increment;
float dmass;
float ii;  // mass moment of inertia
float dresult;


int isocounterbox1[100], isocounterbox2[100], isocounterbox3[100], isocounterbox4[100], isocounterbox5[100], isocounterbox6[100], isocounterbox7[100], isocounterbox8[100], isocounterbox9[100], isocounterbox10[100], isocounterbox11[100], isocounterbox12[100], isocounterbox13[100], isocounterbox14[100], isocounterbox15[100], isocounterbox16[100];
float p, p2;
int bb;


float ydisplat [500][500], xdisplat [500][500], delus [500][500], delun [500][500], fn [500][500], fs [500][000], dn [500][500], ds [500][500];
float yforce [500][500], xforce [500][500], fxsum [500][500], x [500], resultfx [500], fysum [500][500];
float y [500], resultfy [500], msum [500][500], m [500], resultm [500], udoty [500][500], udotysum [500], resultudoty [500];
float udotx [500][500], udotxsum [500], resultudotx [500], thetadot [500][500], thetadotsum [500], resultthetadot [500];
float deluy [500][500], deluysum [500], resultdeluy [500], delux [500][500], deluxsum [500], resultdelux [500], deltheta [500][500];
float delthetasum [500], resultdeltheta [500], uy [500][500], uysum [500], resultuy [500], ux [500][500], uxsum [500];
float resultux [500], theta [500][500], thetasum [500], resulttheta [500];


float rightisosceles [10000][4][10];
int isoscelescounter = 0;
```

```
40.float ydisplacement2 = 0.0, yvelocity2= 0;
41.float xdisplacement2 = 0.0;
42.float alpha [500], alpha1 = 0.785398;
43.int v = 1;
44.float n, n1;
45.int num3 =  1 + (39-4)/1.25;    // number of isosceles triangles per row

46.int hh, oo, pp;
47.
48.
49.
50.int main ()
51.
52.{
53.
54.geometricalshapes righttriangle; //the particle
55.righttriangle.side1 = 4.243;
56.righttriangle.side2 = 3;
57.righttriangle.side3 = 3;
58.righttriangle.angle1 = 45;
59.righttriangle.angle2 = 45;
60.righttriangle.angle3 = 90;
61.righttriangle.height = 0.53;
62.righttriangle.base = 1.061;
63.righttriangle.centroidx = 0;
64.righttriangle.centroidy = 0.707;
65.righttriangle.momentofinertiax = 1.123;
66.righttriangle.momentofinertiay = 3.374;
67.righttriangle.area = 0.2812;
68.
69. ofstream a_file ("data-s3-mw16.txt");
70.
71.
72.
73. cout<< "Please enter the stiffness coefficient in N/m" << endl;
74. cin>> dstiffness_coefficient;
75.
76.
77. cout<< "Please enter the mass of the particle in kg" << endl;
78. cin>> dmass;
79.
80.ii =  ((righttriangle.base/2) * (righttriangle.base/2)* (dmass)/3) ;
81.
82.
83. // time increment verification
84.
85. begin: //goto label
86. cout<< "Please enter the time increment" << endl;
87. cin>> dtime_increment;
88.
89. dresult = (2 * sqrt (dmass/dstiffness_coefficient));
90. if (dtime_increment >=  dresult ) // time increment condition
```

```
91.
92.{
93.
94. cout<< "Time increment does not satisfy condition"<< endl;
95. goto begin;
96.
97.}
98.
99. else
100.
101.      {
102.        cout<< "Time increment satisfies condition"<< endl;
103.
104.     }
105.
106.
107.     dk = 2 * (sqrt (dmass * dstiffness_coefficient)); // subroutine for
    calculating the damping coefficient
108.     ddamping_coefficient = dk/dtime_increment;
109.
110.
111.
112.
113.
114.
115.     for (hh = 0; hh <10000; hh = hh+1) {
116.         for (oo = 0; oo <4; oo = oo+1) { for (pp = 0; pp <10; pp = pp +
    1)
117.         {
118.             rightisosceles [hh][oo][pp] = 0.0;
119.
120.
121.         }
122.         }
123.     }
124.
125.
126.
127.      for (n = 3.0 ; n <= 70; n = n + 1.25) {       // positioning partic
    les
128.         for (n1 = 4.0 ; n1 <= 39; n1 = n1 + 1.25) {
129.
130.         rightisosceles [v][0][0] = n1 ;            // right isosceles x
    coordinates
131.         rightisosceles [v][1][0] = rightisosceles [v][0][0] + (righttria
    ngle.base /2);
132.         rightisosceles [v][2][0] = rightisosceles [v][0][0] -
     (righttriangle.base /2);
133.         rightisosceles [v][3][0] = rightisosceles [v][0][0];
134.
135.         rightisosceles [v][0][1] = n ;    // right isosceles y coordinat
    es
```

```
136.         rightisosceles [v][1][1] = rightisosceles [v][0][1] + (righttria
   ngle.height /3);
137.         rightisosceles [v][2][1] = rightisosceles [v][1][1];
138.         rightisosceles [v][3][1] = rightisosceles [v][2][1] -
    (righttriangle.height);
139.
140.
141.
142.         v = v + 1;
143.         isoscelescounter = isoscelescounter + 1;
144.
145.
146.
147.     }
148.     }
149.
150.     p2 = 10/num3;
151.
152.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
153.     ydisplacement2 =  ydisplacement2 + (yvelocity2 * dtime_increment);

154.
155.
156.
157.
158.
159.     for (p = 100; p < 500000 ; p = p+100)  // load cycle in Newtons
160.     {
161.     p2 = p/num3;
162.
163.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
164.
165.
166.       a_file <<  p << endl;
167.
168.
169.     int isobox1 = 0, isobox2 = 0, isobox3 = 0, isobox4 = 0, isobox5 = 0,
    isobox6 = 0, isobox7 = 0, isobox8 = 0, isobox9 = 0, isobox10 = 0, isobox1
   1 = 0, isobox12 = 0, isobox13 = 0, isobox14 = 0, isobox15 = 0, isobox16 =
   0;
170.
171.
172.     for (bb = 0; bb <100; bb = bb+1) {
173.         isocounterbox1[bb] = 0, isocounterbox2[bb] = 0, isocounterbox3[b
   b] = 0, isocounterbox4[bb]= 0;
174.         isocounterbox5[bb] = 0, isocounterbox6[bb] = 0, isocounterbox7[b
   b] = 0, isocounterbox8[bb] = 0;
175.         isocounterbox9[bb] = 0, isocounterbox10[bb] = 0, isocounterbox11
   [bb] = 0, isocounterbox12[bb] = 0;
```

```
176.          isocounterbox13[bb] = 0, isocounterbox14[bb] = 0, isocounterbox1
   5[bb] = 0, isocounterbox16[bb] = 0;
177.
178.          }
179.
180.
181.
182.     for (int dd = 1; dd <= isoscelescounter ; dd = dd + 1)  // locating
   the isosceles triangles within the 16 screen boxes
183.     {
184.
185.      if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][0]
   <= 11) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][1]<=
   18.5))
186.     {isocounterbox1[isobox1] = dd; isobox1 = isobox1 + 1;}
187.
188.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
189.     {isocounterbox2[isobox2] = dd; isobox2 = isobox2 + 1;}
190.
191.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
192.     {isocounterbox3[isobox3] = dd; isobox3 = isobox3 + 1;}
193.
194.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
195.     {isocounterbox4[isobox4] = dd; isobox4 = isobox4 + 1;}
196.
197.     else if( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][
   0] <= 11) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 37))
198.     {isocounterbox5[isobox5] = dd; isobox5 = isobox5 + 1;}
199.
200.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
201.     {isocounterbox6[isobox6] = dd; isobox6 = isobox6 + 1;}
202.
203.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
204.     {isocounterbox7[isobox7] = dd; isobox7 = isobox7 + 1;}
205.
206.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
207.     {isocounterbox8[isobox8] = dd; isobox8 = isobox8 + 1;}
208.
```

```
209.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 55.5))
210.     {isocounterbox9[isobox9] = dd; isobox9 = isobox9 + 1;}
211.
212.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
213.     {isocounterbox10[isobox10] = dd; isobox10 = isobox10 + 1;}
214.
215.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
216.     {isocounterbox11[isobox11] = dd; isobox11 = isobox11 + 1;}
217.
218.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
219.     {isocounterbox12[isobox12] = dd; isobox12 = isobox12 + 1;}
220.
221.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0
   ][1]<= 74))
222.     {isocounterbox13[isobox13] = dd; isobox13 = isobox13 + 1;}
223.
224.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
225.     {isocounterbox14[isobox14] = dd; isobox14 = isobox14 + 1;}
226.
227.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
228.     {isocounterbox15[isobox15] = dd; isobox15 = isobox15 + 1;}
229.
230.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
231.     {isocounterbox16[isobox16] = dd; isobox16 = isobox16 + 1;}
232.
233.
234.     }  // end of isosceles triangles location within the 16 screen boxes

235.
236.
237.     int gg, ww;
238.
239.     for (gg = 0; gg < 500; gg = gg+1) {
240.
241.      for (ww = 0; ww < 500; ww = ww +1) {
242.
243.     ydisplat [gg][ww] = 0.0;
```

```
    xdisplat [gg][ww] = 0.0;
    delus [gg][ww] = 0.0;
    delun [gg][ww] = 0.0;
    fn [gg][ww] = 0.0;
    fs [gg][ww] = 0.0;
    dn [gg][ww] = 0.0;
    ds [gg][ww] = 0.0;
    yforce [gg][ww] = 0.0;
    xforce [gg][ww] = 0.0;
    fxsum [gg][ww] = 0.0;
    x [ww] = 0.0;
    resultfx [ww] = 0.0;
    fysum [gg][ww] = 0.0;
    y [ww] = 0.0;
    resultfy [ww] = 0.0;
    msum [gg][ww] = 0.0;
    m [ww] = 0.0;
    resultm [ww] = 0.0;
    udoty [gg][ww] = 0.0;
    udotysum [ww] = 0.0;
    resultudoty [ww] = 0.0;
    udotx [gg][ww] = 0.0;
    udotxsum [ww] = 0.0;
    resultudotx [ww] = 0.0;
    thetadot [gg][ww] = 0.0;
    thetadotsum [ww] = 0.0;
    resultthetadot [ww] = 0.0;
    deluy [gg][ww] = 0.0;
    deluysum [ww] = 0.0;
    resultdeluy [ww] = 0.0;
    delux [gg][ww] = 0.0;
    deluxsum [ww] = 0.0;
    resultdelux [ww] =0.0;
    deltheta [gg][ww] = 0.0;
    delthetasum [ww] = 0.0;
    resultdeltheta [ww] = 0.0;
    uy [gg][ww] = 0.0;
    uysum [ww] = 0.0;
    resultuy [ww] = 0.0;
    ux [gg][ww] = 0.0;
    uxsum [ww] = 0.0;
    resultux [ww] = 0.0;
    theta [gg][ww] = 0.0;
    thetasum [ww] = 0.0;
    resulttheta [ww] = 0.0;
    alpha [ww] = 0.0;

    }

    }

```

```
296.
297.     int xx1 = 0, qq1 = 0, contactpnt1 = 0, l1, i1;
298.
299.
300.     for (i1 = 0; i1 < isobox1; i1= i1+1)
301.
302.     {
303.
304.     xx1 = isocounterbox1[i1];
305.
306.
307.
308.     rightisosceles [xx1][0][0] = rightisosceles [xx1][0][0] + resultux [
   xx1] ;
309.
310.     rightisosceles [xx1][0][1] = rightisosceles [xx1][0][1] + resultuy [
   xx1] ;
311.
312.
313.
314.     for ( l1 = 0; l1 < isobox1; l1 = l1+1)
315.
316.     {
317.
318.
319.     qq1 = isocounterbox1[l1];
320.
321.
322.     if ( ((pow((rightisosceles [xx1][0][0] -
    rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) > 0) && ((pow((rightisosceles [xx1][0][0
   ] - rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) <= 3.125))
323.
324.     {
325.
326.     contactpnt1 = contactpnt1 + 1;
327.
328.     if (contactpnt1 >= 4) {goto next1;}
329.
330.     if ( rightisosceles [qq1][3][1] >  rightisosceles [xx1][1][1] )
331.
332.     {
333.
334.
335.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][0] -
    rightisosceles [qq1][0][0]));
```

```
336.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][1] -
   rightisosceles [qq1][0][1]));
337.
338.
339.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
340.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
341.
342.
343.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
344.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
345.
346.
347.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
348.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
349.
350.
351.     yforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * sin (alpha1)) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha1)));
352.     xforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * cos (alpha1)) + ((fn [xx1][contactpnt1] + dn [xx1][contactpn
   t1]) * sin (alpha1)));
353.
354.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
355.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
356.
357.
358.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
359.     y [xx1] = fysum [xx1][contactpnt1];
360.     resultfy [xx1] += y[xx1];
361.
362.
363.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
364.     x [xx1] = fxsum [xx1][contactpnt1];
365.     resultfx [xx1] += x[xx1];
366.
367.
368.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][3][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][3][1] -
    rightisosceles [xx1][0][1]));
369.     m [xx1] = msum [xx1][contactpnt1];
```

```
370.     resultm [xx1] += m[xx1];
371.
372.
373.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
374.     udotysum [xx1] = udoty [xx1][contactpnt1];
375.     resultudoty [xx1] += udotysum [xx1];
376.
377.
378.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
379.     udotxsum [xx1] = udotx [xx1][contactpnt1];
380.     resultudotx [xx1] += udotxsum [xx1];
381.
382.
383.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
384.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
385.     resultthetadot [xx1] += thetadotsum [xx1];
386.
387.
388.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
389.     deluysum [xx1] = deluy [xx1][contactpnt1];
390.     resultdeluy [xx1] += deluysum [xx1];
391.
392.
393.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
394.     deluxsum [xx1] = delux [xx1][contactpnt1];
395.     resultdelux [xx1] += deluxsum [xx1];
396.
397.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
398.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
399.     resultdeltheta [xx1] += delthetasum [xx1];
400.
401.
402.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
403.     uysum [xx1] = uy [xx1][contactpnt1];
404.     resultuy [xx1]+= uysum [xx1];
405.
406.
407.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
408.     uxsum [xx1] = ux [xx1][contactpnt1];
409.     resultux [xx1] += uxsum [xx1];
410.
411.
412.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
413.     thetasum [xx1] = theta [xx1][contactpnt1];
414.     resulttheta [xx1] += thetasum [xx1];
415.     alpha [xx1] = alpha1 + resulttheta [xx1];
```

```
416.
417.     }
418.
419.
420.     else
421.     {
422.
423.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][0][0] -
    rightisosceles [qq1][0][0]));
424.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][1][1] -
   rightisosceles [qq1][0][1]));
425.
426.
427.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
428.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
429.
430.
431.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
432.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
433.
434.
435.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
436.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
437.
438.
439.     yforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * sin (alpha [xx1])) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha [xx1])));

440.     xforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * cos (alpha [xx1])) + ((fn [xx1][contactpnt1] + dn [xx1][conta
   ctpnt1]) * sin (alpha [xx1])));
441.
442.
443.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
444.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
445.
446.
```

```
447.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
448.     x [xx1] = fxsum [xx1][contactpnt1];
449.     resultfx [xx1] += x[xx1];
450.
451.
452.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
453.     y [xx1] = fysum [xx1][contactpnt1];
454.     resultfy [xx1] += y[xx1];
455.
456.
457.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][1][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][1][1] -
    rightisosceles [xx1][0][1]));
458.     m [xx1] = msum [xx1][contactpnt1];
459.     resultm [xx1] += m[xx1];
460.
461.
462.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
463.     udotysum [xx1] = udoty [xx1][contactpnt1];
464.     resultudoty [xx1] += udotysum [xx1];
465.
466.
467.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
468.     udotxsum [xx1] = udotx [xx1][contactpnt1];
469.     resultudotx [xx1] += udotxsum [xx1];
470.
471.
472.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
473.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
474.     resultthetadot [xx1] += thetadotsum [xx1];
475.
476.
477.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
478.     deluysum [xx1] = deluy [xx1][contactpnt1];
479.     resultdeluy [xx1] += deluysum [xx1];
480.
481.
482.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
483.     deluxsum [xx1] = delux [xx1][contactpnt1];
484.     resultdelux [xx1] += deluxsum [xx1];
485.
486.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
487.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
488.     resultdeltheta [xx1] += delthetasum [xx1];
489.
```

```
490.
491.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
492.     uysum [xx1] = uy [xx1][contactpnt1];
493.     resultuy [xx1]+= uysum [xx1];
494.
495.
496.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
497.     uxsum [xx1] = ux [xx1][contactpnt1];
498.     resultux [xx1] += uxsum [xx1];
499.
500.
501.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
502.     thetasum [xx1] = theta [xx1][contactpnt1];
503.     resulttheta [xx1] += thetasum [xx1];
504.     alpha [xx1] = alpha1 + resulttheta [xx1];
505.
506.
507.     }
508.
509.
510.
511.      a_file <<  p  << endl;
512.      a_file << xx1 << endl;
513.      a_file << resultuy [xx1]/10 << endl;
514.
515.     if ( resultuy [xx1] > 72 ) { goto display;}
516.
517.
518.
519.     }
520.
521.
522.     }
523.
524.     next1:
525.     ;}
526.
527.
528.
529.     int xx2 = 0, qq2 = 0, contactpnt2 = 0, l2, i2;
530.
531.
532.     for (i2 = 0; i2 < isobox2; i2= i2+1)
533.
534.     {
535.
536.     xx2 = isocounterbox2[i2];
537.
538.
539.     rightisosceles [xx2][0][0] = rightisosceles [xx2][0][0] + resultux [
    xx2] ;
540.
```

```
541.     rightisosceles [xx2][0][1] = rightisosceles [xx2][0][1] + resultuy [
   xx2] ;
542.
543.
544.     for ( l2 = 0; l2 < isobox2; l2 = l2+1)
545.
546.     {
547.
548.
549.     qq2 = isocounterbox2[l2];
550.
551.
552.
553.     if ( ((pow((rightisosceles [xx2][0][0] -
    rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
    rightisosceles [qq2][0][1]), 2)) > 0) && ((pow((rightisosceles [xx2][0][0
   ] - rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
    rightisosceles [qq2][0][1]), 2)) <= 3.125))
554.
555.     {
556.
557.     contactpnt2 = contactpnt2 + 1;
558.
559.     if (contactpnt2 >= 4) {goto next2;}
560.
561.
562.     if ( rightisosceles [qq2][3][1] >  rightisosceles [xx2][1][1] )
563.
564.     {
565.
566.
567.     ydisplat [xx2][contactpnt2] = ydisplacement2  + (deluy [xx2][contact
   pnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][3][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][0] -
    rightisosceles [qq2][0][0]));
568.     xdisplat [xx2][contactpnt2] = xdisplacement2 + (delux [xx2][contactp
   nt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][3][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][1] -
   rightisosceles [qq2][0][1]));
569.
570.
571.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
572.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
573.
574.
```

```
575.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
576.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
577.
578.
579.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
580.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
581.
582.
583.     yforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * sin (alpha1)) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha1)));
584.     xforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * cos (alpha1)) + ((fn [xx2][contactpnt2] + dn [xx2][contactpn
   t2]) * sin (alpha1)));
585.
586.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
587.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
588.
589.
590.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
591.     y [xx2] = fysum [xx2][contactpnt2];
592.     resultfy [xx2] += y[xx2];
593.
594.
595.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
596.     x [xx2] = fxsum [xx2][contactpnt2];
597.     resultfx [xx2] += x[xx2];
598.
599.
600.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][3][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][3][1] -
    rightisosceles [xx2][0][1]));
601.     m [xx2] = msum [xx2][contactpnt2];
602.     resultm [xx2] += m[xx2];
603.
604.
605.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent) *1000/ dmass); // mm/sec
606.     udotysum [xx2] = udoty [xx2][contactpnt2];
607.     resultudoty [xx2] += udotysum [xx2];
608.
609.
610.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
611.     udotxsum [xx2] = udotx [xx2][contactpnt2];
612.     resultudotx [xx2] += udotxsum [xx2];
613.
```

```
614.
615.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
616.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
617.     resultthetadot [xx2] += thetadotsum [xx2];
618.
619.
620.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
621.     deluysum [xx2] = deluy [xx2][contactpnt2];
622.     resultdeluy [xx2] += deluysum [xx2];
623.
624.
625.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
626.     deluxsum [xx2] = delux [xx2][contactpnt2];
627.     resultdelux [xx2] += deluxsum [xx2];
628.
629.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
630.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
631.     resultdeltheta [xx2] += delthetasum [xx2];
632.
633.
634.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
635.     uysum [xx2] = uy [xx2][contactpnt2];
636.     resultuy [xx2]+= uysum [xx2];
637.
638.
639.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
640.     uxsum [xx2] = ux [xx2][contactpnt2];
641.     resultux [xx2] += uxsum [xx2];
642.
643.
644.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
645.     thetasum [xx2] = theta [xx2][contactpnt2];
646.     resulttheta [xx2] += thetasum [xx2];
647.     alpha [xx2] = alpha1 + resulttheta [xx2];
648.
649.     }
650.
651.     else
652.
653.     {
654.
655.
656.     ydisplat [xx2][contactpnt2] += ydisplacement2  + (deluy [xx2][contac
   tpnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][0][0] -
    rightisosceles [qq2][0][0]));
```

```
657.     xdisplat [xx2][contactpnt2] += xdisplacement2 + (delux [xx2][contact
   pnt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][1][1] -
   rightisosceles [qq2][0][1]));
658.
659.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
660.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
661.
662.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
663.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
664.
665.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
666.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
667.
668.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * sin (alpha [xx2])) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

669.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
   ctpnt2]) * sin (alpha [xx2])));
670.
671.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
672.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
673.
674.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
675.     x [xx2] = fxsum [xx2][contactpnt2];
676.     resultfx [xx2] += x[xx2];
677.
678.
679.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
680.     y [xx2] = fysum [xx2][contactpnt2];
681.     resultfy [xx2] += y[xx2];
682.
683.
684.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][1][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][1][1] -
    rightisosceles [xx2][0][1]));
685.     m [xx2] = msum [xx2][contactpnt2];
686.     resultm [xx2] += m[xx2];
687.
688.
```

```
689.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
690.     udotysum [xx2] = udoty [xx2][contactpnt2];
691.     resultudoty [xx2] += udotysum [xx2];
692.
693.
694.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
695.     udotxsum [xx2] = udotx [xx2][contactpnt2];
696.     resultudotx [xx2] += udotxsum [xx2];
697.
698.
699.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
700.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
701.     resultthetadot [xx2] += thetadotsum [xx2];
702.
703.
704.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
705.     deluysum [xx2] = deluy [xx2][contactpnt2];
706.     resultdeluy [xx2] += deluysum [xx2];
707.
708.
709.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
710.     deluxsum [xx2] = delux [xx2][contactpnt2];
711.     resultdelux [xx2] += deluxsum [xx2];
712.
713.
714.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
715.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
716.     resultdeltheta [xx2] += delthetasum [xx2];
717.
718.
719.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
720.     uysum [xx2] = uy [xx2][contactpnt2];
721.     resultuy [xx2]+= uysum [xx2];
722.
723.
724.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
725.     uxsum [xx2] = ux [xx2][contactpnt2];
726.     resultux [xx2] += uxsum [xx2];
727.
728.
729.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
730.     thetasum [xx2] = theta [xx2][contactpnt2];
731.     resultheta [xx2] += thetasum [xx2];
732.     alpha [xx2] = alpha1 + resultheta [xx2];
733.
734.     }
```

```
735.     }
736.
737.     }
738.
739.     next2:
740.     ;}
741.
742.     int xx3 = 0, qq3 = 0, contactpnt3 = 0, l3, i3;
743.
744.
745.     for (i3 = 0; i3 < isobox3; i3= i3+1)
746.
747.     {
748.
749.     xx3 = isocounterbox3[i3];
750.
751.
752.     rightisosceles [xx3][0][0] = rightisosceles [xx3][0][0] + resultux [
    xx3] ;
753.
754.     rightisosceles [xx3][0][1] = rightisosceles [xx3][0][1] + resultuy [
    xx3] ;
755.
756.
757.     for ( l3 = 0; l3 < isobox3; l3 = l3+1)
758.
759.     {
760.
761.
762.     qq3 = isocounterbox3[l3];
763.
764.
765.
766.     if ( ((pow((rightisosceles [xx3][0][0] -
    rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) > 0) && ((pow((rightisosceles [xx3][0][0
    ] - rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) <= 3.125))
767.
768.     {
769.
770.     contactpnt3 = contactpnt3 + 1;
771.
772.     if (contactpnt3 >= 4) {goto next3;}
773.
774.
775.     if ( rightisosceles [qq3][3][1] >  rightisosceles [xx3][1][1] )
776.
777.     {
778.
779.
```

```
780.     ydisplat [xx3][contactpnt3] = ydisplacement2  + (deluy [xx3][contact
   pnt3] -
    deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][0] - rightisosceles [xx3][0][0])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][0] -
    rightisosceles [qq3][0][0]));
781.     xdisplat [xx3][contactpnt3] = xdisplacement2 + (delux [xx3][contactp
   nt3] -
    delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][1] - rightisosceles [xx3][0][1])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][1] -
   rightisosceles [qq3][0][1]));
782.
783.
784.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
   1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
785.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
   1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
786.
787.
788.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
   1) *  (dstiffness_coefficient/1000);
789.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
   fficient/1000);
790.
791.
792.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
   1) * (ddamping_coefficient/1000);
793.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
   ient/1000);
794.
795.
796.     yforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * sin (alpha1)) -
    ((fn [xx3][contactpnt3] + dn [xx3][contactpnt3]) * cos (alpha1)));
797.     xforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * cos (alpha1)) + ((fn [xx3][contactpnt3] + dn [xx3][contactpn
   t3]) * sin (alpha1)));
798.
799.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
800.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
801.
802.
803.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
804.     y [xx3] = fysum [xx3][contactpnt3];
805.     resultfy [xx3] += y[xx3];
806.
807.
808.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
809.     x [xx3] = fxsum [xx3][contactpnt3];
810.     resultfx [xx3] += x[xx3];
811.
```

```
812.
813.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][3][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][3][1] -
    rightisosceles [xx3][0][1]));
814.     m [xx3] = msum [xx3][contactpnt3];
815.     resultm [xx3] += m[xx3];
816.
817.
818.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent) *1000/ dmass); // mm/sec
819.     udotysum [xx3] = udoty [xx3][contactpnt3];
820.     resultudoty [xx3] += udotysum [xx3];
821.
822.
823.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
824.     udotxsum [xx3] = udotx [xx3][contactpnt3];
825.     resultudotx [xx3] += udotxsum [xx3];
826.
827.
828.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
829.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
830.     resultthetadot [xx3] += thetadotsum [xx3];
831.
832.
833.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
834.     deluysum [xx3] = deluy [xx3][contactpnt3];
835.     resultdeluy [xx3] += deluysum [xx3];
836.
837.
838.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
839.     deluxsum [xx3] = delux [xx3][contactpnt3];
840.     resultdelux [xx3] += deluxsum [xx3];
841.
842.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
843.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
844.     resultdeltheta [xx3] += delthetasum [xx3];
845.
846.
847.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
848.     uysum [xx3] = uy [xx3][contactpnt3];
849.     resultuy [xx3]+= uysum [xx3];
850.
851.
852.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
853.     uxsum [xx3] = ux [xx3][contactpnt3];
854.     resultux [xx3] += uxsum [xx3];
```

```
855.
856.
857.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
858.     thetasum [xx3] = theta [xx3][contactpnt3];
859.     resulttheta [xx3] += thetasum [xx3];
860.     alpha [xx3] = alpha1 + resulttheta [xx3];
861.
862.     }
863.
864.      else
865.     {
866.
867.
868.     ydisplat [xx3][contactpnt2] += ydisplacement2  + (deluy [xx3][contac
    tpnt3] -
     deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
     [qq3][1][0] - rightisosceles [xx3][0][0])  -
     deltheta [qq3][contactpnt3] * (rightisosceles [qq3][0][0] -
     rightisosceles [qq3][0][0]));
869.     xdisplat [xx3][contactpnt2] += xdisplacement2 + (delux [xx3][contact
    pnt3] -
     delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
     [qq3][1][1] - rightisosceles [xx3][0][1])  -
     deltheta [qq3][contactpnt3] * (rightisosceles [qq3][1][1] -
    rightisosceles [qq3][0][1]));
870.
871.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
    1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
872.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
    1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
873.
874.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
    1) *  (dstiffness_coefficient/1000);
875.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
    fficient/1000);
876.
877.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
    1) * (ddamping_coefficient/1000);
878.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
    ient/1000);
879.
880.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
    tactpnt2]) * sin (alpha [xx2])) -
     ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

881.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
    tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
    ctpnt2]) * sin (alpha [xx2])));
882.
883.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
884.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
885.
```

```
886.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
887.     x [xx3] = fxsum [xx3][contactpnt3];
888.     resultfx [xx3] += x[xx3];
889.
890.
891.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
892.     y [xx3] = fysum [xx3][contactpnt3];
893.     resultfy [xx3] += y[xx3];
894.
895.
896.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][1][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][1][1] -
    rightisosceles [xx3][0][1]));
897.     m [xx3] = msum [xx3][contactpnt3];
898.     resultm [xx3] += m[xx3];
899.
900.
901.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
902.     udotysum [xx3] = udoty [xx3][contactpnt3];
903.     resultudoty [xx3] += udotysum [xx3];
904.
905.
906.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
907.     udotxsum [xx3] = udotx [xx3][contactpnt3];
908.     resultudotx [xx3] += udotxsum [xx3];
909.
910.
911.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
912.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
913.     resultthetadot [xx3] += thetadotsum [xx3];
914.
915.
916.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
917.     deluysum [xx3] = deluy [xx3][contactpnt3];
918.     resultdeluy [xx3] += deluysum [xx3];
919.
920.
921.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
922.     deluxsum [xx3] = delux [xx3][contactpnt3];
923.     resultdelux [xx3] += deluxsum [xx3];
924.
925.
926.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
927.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
928.     resultdeltheta [xx3] += delthetasum [xx3];
```

```
929.
930.
931.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
932.     uysum [xx3] = uy [xx3][contactpnt3];
933.     resultuy [xx3]+= uysum [xx3];
934.
935.
936.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
937.     uxsum [xx3] = ux [xx3][contactpnt3];
938.     resultux [xx3] += uxsum [xx3];
939.
940.
941.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
942.     thetasum [xx3] = theta [xx3][contactpnt3];
943.     resulttheta [xx3] += thetasum [xx3];
944.     alpha [xx3] = alpha1 + resulttheta [xx3];
945.     }
946.
947.     }
948.
949.     }
950.
951.     next3:
952.     ;}
953.
954.
955.     int xx4 = 0, qq4 = 0, contactpnt4 = 0, l4, i4;
956.
957.
958.     for (i4 = 0; i4 < isobox4; i4= i4+1)
959.
960.     {
961.
962.     xx4 = isocounterbox4[i4];
963.
964.
965.     rightisosceles [xx4][0][0] = rightisosceles [xx4][0][0] + resultux [
    xx4] ;
966.
967.     rightisosceles [xx4][0][1] = rightisosceles [xx4][0][1] + resultuy [
    xx4] ;
968.
969.
970.     for ( l4 = 0; l4 < isobox4; l4 = l4+1)
971.
972.     {
973.
974.
975.     qq4 = isocounterbox4[l4];
976.
977.
978.
```

```
    if ( ((pow((rightisosceles [xx4][0][0] -
     rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) > 0) && ((pow((rightisosceles [xx4][0][0
    ] - rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) <= 3.125))

    {

    contactpnt4 = contactpnt4 + 1;

    if (contactpnt4 >= 4) {goto next4;}


    if ( rightisosceles [qq4][3][1] >  rightisosceles [xx4][1][1] )

    {


    ydisplat [xx4][contactpnt4] = ydisplacement2  + (deluy [xx4][contact
   pnt4] -
     deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][0] - rightisosceles [xx4][0][0])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][0] -
     rightisosceles [qq4][0][0]));
    xdisplat [xx4][contactpnt4] = xdisplacement2 + (delux [xx4][contactp
   nt4] -
     delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][1] - rightisosceles [xx4][0][1])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][1] -
   rightisosceles [qq4][0][1]));


    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));


    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-
   1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coe
   fficient/1000);


    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-
   1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coeffic
   ient/1000);


```

```
    yforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * sin (alpha1)) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha1)));
    xforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * cos (alpha1)) + ((fn [xx4][contactpnt4] + dn [xx4][contactpn
t4]) * sin (alpha1)));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisoscel
es [qq4][3][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][3][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
    resultthetadot [xx4] += thetadotsum [xx4];


    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
t;
    deluysum [xx4] = deluy [xx4][contactpnt4];
    resultdeluy [xx4] += deluysum [xx4];

```

```
1050.
1051.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1052.    deluxsum [xx4] = delux [xx4][contactpnt4];
1053.    resultdelux [xx4] += deluxsum [xx4];
1054.
1055.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1056.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1057.    resultdeltheta [xx4] += delthetasum [xx4];
1058.
1059.
1060.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1061.    uysum [xx4] = uy [xx4][contactpnt4];
1062.    resultuy [xx4]+= uysum [xx4];
1063.
1064.
1065.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1066.    uxsum [xx4] = ux [xx4][contactpnt4];
1067.    resultux [xx4] += uxsum [xx4];
1068.
1069.
1070.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1071.    thetasum [xx4] = theta [xx4][contactpnt4];
1072.    resulttheta [xx4] += thetasum [xx4];
1073.    alpha [xx4] = alpha1 + resulttheta [xx4];
1074.
1075.    }
1076.
1077.    else
1078.
1079.    {
1080.
1081.
1082.    ydisplat [xx4][contactpnt4] += ydisplacement2  + (deluy [xx4][contac
   tpnt4] -
    deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
    [qq4][1][0] - rightisosceles [xx4][0][0])  -
    deltheta [qq4][contactpnt4] * (rightisosceles [qq4][0][0] -
    rightisosceles [qq4][0][0]));
1083.    xdisplat [xx4][contactpnt4] += xdisplacement2 + (delux [xx4][contact
   pnt4] -
    delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
    [qq4][1][1] - rightisosceles [xx4][0][1])  -
    deltheta [qq4][contactpnt4] * (rightisosceles [qq4][1][1] -
   rightisosceles [qq4][0][1]));
1084.
1085.    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
1086.    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));
1087.
```

```
1088.    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-
   1) *  (dstiffness_coefficient/1000);
1089.    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coe
   fficient/1000);
1090.
1091.    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-
   1) * (ddamping_coefficient/1000);
1092.    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coeffic
   ient/1000);
1093.
1094.    yforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][con
   tactpnt4]) * sin (alpha [xx4])) -
    ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha [xx4])));

1095.    xforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][con
   tactpnt4]) * cos (alpha [xx4])) + ((fn [xx4][contactpnt4] + dn [xx4][conta
   ctpnt4]) * sin (alpha [xx4])));
1096.
1097.    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
1098.    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);
1099.
1100.    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
1101.    x [xx4] = fxsum [xx4][contactpnt4];
1102.    resultfx [xx4] += x[xx4];
1103.
1104.
1105.    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
1106.    y [xx4] = fysum [xx4][contactpnt4];
1107.    resultfy [xx4] += y[xx4];
1108.
1109.
1110.    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisoscel
   es [qq4][1][0] - rightisosceles [xx4][0][0]) -
    xforce [xx4][contactpnt4] * ( rightisosceles [qq4][1][1] -
    rightisosceles [xx4][0][1]));
1111.    m [xx4] = msum [xx4][contactpnt4];
1112.    resultm [xx4] += m[xx4];
1113.
1114.
1115.    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1116.    udotysum [xx4] = udoty [xx4][contactpnt4];
1117.    resultudoty [xx4] += udotysum [xx4];
1118.
1119.
1120.    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1121.    udotxsum [xx4] = udotx [xx4][contactpnt4];
1122.    resultudotx [xx4] += udotxsum [xx4];
1123.
1124.
```

```
1125.    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
   ement)*1000/ ii); //1/s
1126.    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
1127.    resultthetadot [xx4] += thetadotsum [xx4];
1128.
1129.
1130.    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
   t;
1131.    deluysum [xx4] = deluy [xx4][contactpnt4];
1132.    resultdeluy [xx4] += deluysum [xx4];
1133.
1134.
1135.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1136.    deluxsum [xx4] = delux [xx4][contactpnt4];
1137.    resultdelux [xx4] += deluxsum [xx4];
1138.
1139.
1140.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1141.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1142.    resultdeltheta [xx4] += delthetasum [xx4];
1143.
1144.
1145.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1146.    uysum [xx4] = uy [xx4][contactpnt4];
1147.    resultuy [xx4]+= uysum [xx4];
1148.
1149.
1150.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1151.    uxsum [xx4] = ux [xx4][contactpnt4];
1152.    resultux [xx4] += uxsum [xx4];
1153.
1154.
1155.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1156.    thetasum [xx4] = theta [xx4][contactpnt4];
1157.    resulttheta [xx4] += thetasum [xx4];
1158.    alpha [xx4] = alpha1 + resulttheta [xx4];
1159.    }
1160.
1161.    }
1162.
1163.    }
1164.
1165.    next4:
1166.    ;}
1167.
1168.
1169.    int xx5 = 0, qq5 = 0, contactpnt5 = 0, l5, i5;
1170.
1171.
1172.    for (i5 = 0; i5 < isobox5; i5= i5+1)
```

```
1173.
1174.    {
1175.
1176.    xx5 = isocounterbox5[i5];
1177.
1178.    rightisosceles [xx5][0][0] = rightisosceles [xx5][0][0] + resultux [
   xx5] ;
1179.
1180.    rightisosceles [xx5][0][1] = rightisosceles [xx5][0][1] + resultuy [
   xx5] ;
1181.
1182.
1183.
1184.    for ( l5 = 0; l5 < isobox5; l5 = l5+1)
1185.
1186.    {
1187.
1188.
1189.    qq5 = isocounterbox5[l5];
1190.
1191.
1192.
1193.    if ( ((pow((rightisosceles [xx5][0][0] -
    rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) > 0) && ((pow((rightisosceles [xx5][0][0
   ] - rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) <= 3.125))
1194.
1195.    {
1196.
1197.    contactpnt5 = contactpnt5 + 1;
1198.
1199.    if (contactpnt5 >= 4) {goto next5;}
1200.
1201.
1202.    if ( rightisosceles [qq5][3][1] >  rightisosceles [xx5][1][1] )
1203.
1204.    {
1205.
1206.
1207.    ydisplat [xx5][contactpnt5] = ydisplacement2  + (deluy [xx5][contact
   pnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][0] -
    rightisosceles [qq5][0][0]));
1208.    xdisplat [xx5][contactpnt5] = xdisplacement2 + (delux [xx5][contactp
   nt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][1] -
   rightisosceles [qq5][0][1]));
```

```
1209.
1210.
1211.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1212.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1213.
1214.
1215.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1216.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1217.
1218.
1219.    dn [xx5][contactpnt5] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1220.    ds [xx5][contactpnt5] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1221.
1222.
1223.    yforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * sin (alpha1)) -
     ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha1)));
1224.    xforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * cos (alpha1)) + ((fn [xx5][contactpnt5] + dn [xx5][contactpn
   t5]) * sin (alpha1)));
1225.
1226.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1227.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1228.
1229.
1230.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1231.    y [xx5] = fysum [xx5][contactpnt5];
1232.    resultfy [xx5] += y[xx5];
1233.
1234.
1235.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1236.    x [xx5] = fxsum [xx5][contactpnt5];
1237.    resultfx [xx5] += x[xx5];
1238.
1239.
1240.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][3][0] - rightisosceles [xx5][0][0]) -
     xforce [xx5][contactpnt5] * ( rightisosceles [qq5][3][1] -
     rightisosceles [xx5][0][1]));
1241.    m [xx5] = msum [xx5][contactpnt5];
1242.    resultm [xx5] += m[xx5];
1243.
1244.
1245.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1246.    udotysum [xx5] = udoty [xx5][contactpnt5];
```

```
1247.    resultudoty [xx5] += udotysum [xx5];
1248.
1249.
1250.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1251.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1252.    resultudotx [xx5] += udotxsum [xx5];
1253.
1254.
1255.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1256.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1257.    resultthetadot [xx5] += thetadotsum [xx5];
1258.
1259.
1260.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1261.    deluysum [xx5] = deluy [xx5][contactpnt5];
1262.    resultdeluy [xx5] += deluysum [xx5];
1263.
1264.
1265.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1266.    deluxsum [xx5] = delux [xx5][contactpnt5];
1267.    resultdelux [xx5] += deluxsum [xx5];
1268.
1269.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1270.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1271.    resultdeltheta [xx5] += delthetasum [xx5];
1272.
1273.
1274.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1275.    uysum [xx5] = uy [xx5][contactpnt5];
1276.    resultuy [xx5]+= uysum [xx5];
1277.
1278.
1279.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1280.    uxsum [xx5] = ux [xx5][contactpnt5];
1281.    resultux [xx5] += uxsum [xx5];
1282.
1283.
1284.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1285.    thetasum [xx5] = theta [xx5][contactpnt5];
1286.    resulttheta [xx5] += thetasum [xx5];
1287.    alpha [xx5] = alpha1 + resulttheta [xx5];
1288.
1289.    }
1290.
1291.    else
1292.
1293.    {
```

```
1294.
1295.
1296.    ydisplat [xx5][contactpnt5] += ydisplacement2  + (deluy [xx5][contac
   tpnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][0][0] -
    rightisosceles [qq5][0][0]));
1297.    xdisplat [xx5][contactpnt5] += xdisplacement2 + (delux [xx5][contact
   pnt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][1][1] -
   rightisosceles [qq5][0][1]));
1298.
1299.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1300.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1301.
1302.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1303.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1304.
1305.    dn [xx5][contactpnt2] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1306.    ds [xx5][contactpnt2] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1307.
1308.    yforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * sin (alpha [xx5])) -
    ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha [xx5])));

1309.    xforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * cos (alpha [xx5])) + ((fn [xx5][contactpnt5] + dn [xx5][conta
   ctpnt5]) * sin (alpha [xx5])));
1310.
1311.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1312.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1313.
1314.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1315.    x [xx5] = fxsum [xx5][contactpnt5];
1316.    resultfx [xx5] += x[xx5];
1317.
1318.
1319.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1320.    y [xx5] = fysum [xx5][contactpnt5];
1321.    resultfy [xx5] += y[xx5];
1322.
1323.
```

```
1324.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][1][0] - rightisosceles [xx5][0][0]) -
    xforce [xx5][contactpnt5] * ( rightisosceles [qq5][1][1] -
    rightisosceles [xx5][0][1]));
1325.    m [xx5] = msum [xx5][contactpnt5];
1326.    resultm [xx5] += m[xx5];
1327.
1328.
1329.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1330.    udotysum [xx5] = udoty [xx5][contactpnt5];
1331.    resultudoty [xx5] += udotysum [xx5];
1332.
1333.
1334.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1335.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1336.    resultudotx [xx5] += udotxsum [xx5];
1337.
1338.
1339.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1340.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1341.    resultthetadot [xx5] += thetadotsum [xx5];
1342.
1343.
1344.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1345.    deluysum [xx5] = deluy [xx5][contactpnt5];
1346.    resultdeluy [xx5] += deluysum [xx5];
1347.
1348.
1349.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1350.    deluxsum [xx5] = delux [xx5][contactpnt5];
1351.    resultdelux [xx5] += deluxsum [xx5];
1352.
1353.
1354.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1355.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1356.    resultdeltheta [xx5] += delthetasum [xx5];
1357.
1358.
1359.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1360.    uysum [xx5] = uy [xx5][contactpnt5];
1361.    resultuy [xx5]+= uysum [xx5];
1362.
1363.
1364.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1365.    uxsum [xx5] = ux [xx5][contactpnt5];
1366.    resultux [xx5] += uxsum [xx5];
```

```
1367.
1368.
1369.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1370.    thetasum [xx5] = theta [xx5][contactpnt5];
1371.    resulttheta [xx5] += thetasum [xx5];
1372.    alpha [xx5] = alpha1 + resulttheta [xx5];
1373.
1374.    }
1375.    }
1376.
1377.    }
1378.
1379.    next5:
1380.    ;}
1381.
1382.
1383.    int xx6 = 0, qq6 = 0, contactpnt6 = 0, l6, i6;
1384.
1385.
1386.    for (i6 = 0; i6 < isobox6; i6= i6+1)
1387.
1388.    {
1389.
1390.    xx6 = isocounterbox6[i6];
1391.
1392.    rightisosceles [xx6][0][0] = rightisosceles [xx6][0][0] + resultux [
   xx6] ;
1393.
1394.    rightisosceles [xx6][0][1] = rightisosceles [xx6][0][1] + resultuy [
   xx6] ;
1395.
1396.
1397.
1398.    for ( l6 = 0; l6 < isobox6; l6 = l6+1)
1399.
1400.    {
1401.
1402.
1403.    qq6 = isocounterbox6[l6];
1404.
1405.
1406.
1407.    if ( ((pow((rightisosceles [xx6][0][0] -
    rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) > 0) && ((pow((rightisosceles [xx6][0][0
   ] - rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) <= 3.125))
1408.
1409.    {
1410.
1411.    contactpnt6 = contactpnt6 + 1;
1412.
```

```
1413.    if (contactpnt6 >= 4) {goto next6;}
1414.
1415.
1416.    if ( rightisosceles [qq6][3][1] >  rightisosceles [xx6][1][1] )
1417.
1418.    {
1419.
1420.
1421.    ydisplat [xx6][contactpnt6] = ydisplacement2  + (deluy [xx6][contact
   pnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][0] -
    rightisosceles [qq6][0][0]));
1422.    xdisplat [xx6][contactpnt6] = xdisplacement2 + (delux [xx6][contactp
   nt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][1] -
   rightisosceles [qq6][0][1]));
1423.
1424.
1425.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1426.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1427.
1428.
1429.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1430.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1431.
1432.
1433.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1434.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1435.
1436.
1437.    yforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * sin (alpha1)) -
     ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha1)));
1438.    xforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * cos (alpha1)) + ((fn [xx6][contactpnt6] + dn [xx6][contactpn
   t6]) * sin (alpha1)));
1439.
1440.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1441.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1442.
1443.
1444.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
```

```
    y [xx6] = fysum [xx6][contactpnt6];
    resultfy [xx6] += y[xx6];


    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
    x [xx6] = fxsum [xx6][contactpnt6];
    resultfx [xx6] += x[xx6];


    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][3][0] - rightisosceles [xx6][0][0]) -
     xforce [xx6][contactpnt6] * ( rightisosceles [qq6][3][1] -
     rightisosceles [xx6][0][1]));
    m [xx6] = msum [xx6][contactpnt6];
    resultm [xx6] += m[xx6];


    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent) *1000/ dmass); // mm/sec
    udotysum [xx6] = udoty [xx6][contactpnt6];
    resultudoty [xx6] += udotysum [xx6];


    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
    udotxsum [xx6] = udotx [xx6][contactpnt6];
    resultudotx [xx6] += udotxsum [xx6];


    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
    resultthetadot [xx6] += thetadotsum [xx6];


    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
    deluysum [xx6] = deluy [xx6][contactpnt6];
    resultdeluy [xx6] += deluysum [xx6];


    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
    deluxsum [xx6] = delux [xx6][contactpnt6];
    resultdelux [xx6] += deluxsum [xx6];

    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
    delthetasum [xx6] = deltheta [xx6][contactpnt6];
    resultdeltheta [xx6] += delthetasum [xx6];


```

```
1488.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1489.    uysum [xx6] = uy [xx6][contactpnt6];
1490.    resultuy [xx6]+= uysum [xx6];
1491.
1492.
1493.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1494.    uxsum [xx6] = ux [xx6][contactpnt6];
1495.    resultux [xx6] += uxsum [xx6];
1496.
1497.
1498.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1499.    thetasum [xx6] = theta [xx6][contactpnt6];
1500.    resulttheta [xx6] += thetasum [xx6];
1501.    alpha [xx6] = alpha1 + resulttheta [xx6];
1502.
1503.    }
1504.
1505.    else
1506.
1507.    {
1508.
1509.
1510.
1511.    ydisplat [xx6][contactpnt6] += ydisplacement2  + (deluy [xx6][contac
   tpnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][0][0] -
    rightisosceles [qq6][0][0]));
1512.    xdisplat [xx6][contactpnt6] += xdisplacement2 + (delux [xx6][contact
   pnt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][1][1] -
   rightisosceles [qq6][0][1]));
1513.
1514.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1515.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1516.
1517.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1518.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1519.
1520.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1521.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1522.
```

```
1523.    yforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * sin (alpha [xx6])) -
    ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha [xx6])));

1524.    xforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * cos (alpha [xx6])) + ((fn [xx6][contactpnt6] + dn [xx6][conta
   ctpnt6]) * sin (alpha [xx6])));
1525.
1526.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1527.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1528.
1529.    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
1530.    x [xx6] = fxsum [xx6][contactpnt6];
1531.    resultfx [xx6] += x[xx6];
1532.
1533.
1534.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
1535.    y [xx6] = fysum [xx6][contactpnt6];
1536.    resultfy [xx6] += y[xx6];
1537.
1538.
1539.    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][1][0] - rightisosceles [xx6][0][0]) -
    xforce [xx6][contactpnt6] * ( rightisosceles [qq6][1][1] -
    rightisosceles [xx6][0][1]));
1540.    m [xx6] = msum [xx6][contactpnt6];
1541.    resultm [xx6] += m[xx6];
1542.
1543.
1544.    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1545.    udotysum [xx6] = udoty [xx6][contactpnt6];
1546.    resultudoty [xx6] += udotysum [xx6];
1547.
1548.
1549.    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1550.    udotxsum [xx6] = udotx [xx6][contactpnt6];
1551.    resultudotx [xx6] += udotxsum [xx6];
1552.
1553.
1554.    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
1555.    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
1556.    resultthetadot [xx6] += thetadotsum [xx6];
1557.
1558.
1559.    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
1560.    deluysum [xx6] = deluy [xx6][contactpnt6];
1561.    resultdeluy [xx6] += deluysum [xx6];
1562.
```

```
1563.
1564.    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
1565.    deluxsum [xx6] = delux [xx6][contactpnt6];
1566.    resultdelux [xx6] += deluxsum [xx6];
1567.
1568.
1569.    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
1570.    delthetasum [xx6] = deltheta [xx6][contactpnt6];
1571.    resultdeltheta [xx6] += delthetasum [xx6];
1572.
1573.
1574.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1575.    uysum [xx6] = uy [xx6][contactpnt6];
1576.    resultuy [xx6]+= uysum [xx6];
1577.
1578.
1579.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1580.    uxsum [xx6] = ux [xx6][contactpnt6];
1581.    resultux [xx6] += uxsum [xx6];
1582.
1583.
1584.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1585.    thetasum [xx6] = theta [xx6][contactpnt6];
1586.    resulttheta [xx6] += thetasum [xx6];
1587.    alpha [xx6] = alpha1 + resulttheta [xx6];
1588.
1589.    }
1590.
1591.    }
1592.
1593.    }
1594.
1595.    next6:
1596.    ;}
1597.
1598.
1599.
1600.    int xx7 = 0, qq7 = 0, contactpnt7 = 0, l7, i7;
1601.
1602.
1603.    for (i7 = 0; i7 < isobox7; i7= i7+1)
1604.
1605.    {
1606.
1607.    xx7 = isocounterbox7[i7];
1608.
1609.    rightisosceles [xx7][0][0] = rightisosceles [xx7][0][0] + resultux [
   xx7] ;
1610.
```

```
    rightisosceles [xx7][0][1] = rightisosceles [xx7][0][1] + resultuy [
   xx7] ;


    for ( l7 = 0; l7 < isobox7; l7 = l7+1)

    {


    qq7 = isocounterbox7[l7];


    if ( ((pow((rightisosceles [xx7][0][0] -
    rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) > 0) && ((pow((rightisosceles [xx7][0][0
   ] - rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) <= 3.125))

    {

    contactpnt7 = contactpnt7 + 1;

    if (contactpnt7 >= 4) {goto next7;}


    if ( rightisosceles [qq7][3][1] >  rightisosceles [xx7][1][1] )

    {


    ydisplat [xx7][contactpnt7] = ydisplacement2  + (deluy [xx7][contact
   pnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][0] - rightisosceles [xx7][0][0])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][0] -
    rightisosceles [qq7][0][0]));
    xdisplat [xx7][contactpnt7] = xdisplacement2 + (delux [xx7][contactp
   nt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][1] -
   rightisosceles [qq7][0][1]));


    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));


```

```
1646.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1647.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1648.
1649.
1650.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1651.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1652.
1653.
1654.    yforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * sin (alpha1)) -
     ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha1)));
1655.    xforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * cos (alpha1)) + ((fn [xx7][contactpnt7] + dn [xx7][contactpn
   t7]) * sin (alpha1)));
1656.
1657.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1658.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1659.
1660.
1661.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1662.    y [xx7] = fysum [xx7][contactpnt7];
1663.    resultfy [xx7] += y[xx7];
1664.
1665.
1666.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1667.    x [xx7] = fxsum [xx7][contactpnt7];
1668.    resultfx [xx7] += x[xx7];
1669.
1670.
1671.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][3][0] - rightisosceles [xx7][0][0]) -
     xforce [xx7][contactpnt7] * ( rightisosceles [qq7][3][1] -
     rightisosceles [xx7][0][1]));
1672.    m [xx7] = msum [xx7][contactpnt7];
1673.    resultm [xx7] += m[xx7];
1674.
1675.
1676.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1677.    udotysum [xx7] = udoty [xx7][contactpnt7];
1678.    resultudoty [xx7] += udotysum [xx7];
1679.
1680.
1681.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1682.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1683.    resultudotx [xx7] += udotxsum [xx7];
1684.
```

```
1685.
1686.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1687.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1688.    resultthetadot [xx7] += thetadotsum [xx7];
1689.
1690.
1691.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1692.    deluysum [xx7] = deluy [xx7][contactpnt7];
1693.    resultdeluy [xx7] += deluysum [xx7];
1694.
1695.
1696.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1697.    deluxsum [xx7] = delux [xx7][contactpnt7];
1698.    resultdelux [xx7] += deluxsum [xx7];
1699.
1700.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1701.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1702.    resultdeltheta [xx7] += delthetasum [xx7];
1703.
1704.
1705.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1706.    uysum [xx7] = uy [xx7][contactpnt7];
1707.    resultuy [xx7]+= uysum [xx7];
1708.
1709.
1710.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1711.    uxsum [xx7] = ux [xx7][contactpnt7];
1712.    resultux [xx7] += uxsum [xx7];
1713.
1714.
1715.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1716.    thetasum [xx7] = theta [xx7][contactpnt7];
1717.    resultttheta [xx7] += thetasum [xx7];
1718.    alpha [xx7] = alpha1 + resultttheta [xx7];
1719.
1720.    }
1721.
1722.    else
1723.
1724.    {
1725.
1726.
1727.
1728.    ydisplat [xx7][contactpnt7] += ydisplacement2  + (deluy [xx7][contac
   tpnt7] -
     deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
     [qq7][1][0] - rightisosceles [xx7][0][0])  -
```

```
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][0][0] -
    rightisosceles [qq7][0][0]));
1729.    xdisplat [xx7][contactpnt7] += xdisplacement2 + (delux [xx7][contact
   pnt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][1][1] -
   rightisosceles [qq7][0][1]));
1730.
1731.    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
1732.    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));
1733.
1734.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1735.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1736.
1737.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1738.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1739.
1740.    yforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * sin (alpha [xx7])) -
    ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha [xx7])));

1741.    xforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * cos (alpha [xx7])) + ((fn [xx7][contactpnt7] + dn [xx7][conta
   ctpnt7]) * sin (alpha [xx7])));
1742.
1743.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1744.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1745.
1746.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1747.    x [xx7] = fxsum [xx7][contactpnt7];
1748.    resultfx [xx7] += x[xx7];
1749.
1750.
1751.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1752.    y [xx7] = fysum [xx7][contactpnt7];
1753.    resultfy [xx7] += y[xx7];
1754.
1755.
1756.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][1][0] - rightisosceles [xx7][0][0]) -
    xforce [xx7][contactpnt7] * ( rightisosceles [qq7][1][1] -
    rightisosceles [xx7][0][1]));
1757.    m [xx7] = msum [xx7][contactpnt7];
1758.    resultm [xx7] += m[xx7];
1759.
```

```
1760.
1761.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1762.    udotysum [xx7] = udoty [xx7][contactpnt7];
1763.    resultudoty [xx7] += udotysum [xx7];
1764.
1765.
1766.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1767.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1768.    resultudotx [xx7] += udotxsum [xx7];
1769.
1770.
1771.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1772.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1773.    resultthetadot [xx7] += thetadotsum [xx7];
1774.
1775.
1776.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1777.    deluysum [xx7] = deluy [xx7][contactpnt7];
1778.    resultdeluy [xx7] += deluysum [xx7];
1779.
1780.
1781.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1782.    deluxsum [xx7] = delux [xx7][contactpnt7];
1783.    resultdelux [xx7] += deluxsum [xx7];
1784.
1785.
1786.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1787.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1788.    resultdeltheta [xx7] += delthetasum [xx7];
1789.
1790.
1791.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1792.    uysum [xx7] = uy [xx7][contactpnt7];
1793.    resultuy [xx7]+= uysum [xx7];
1794.
1795.
1796.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1797.    uxsum [xx7] = ux [xx7][contactpnt7];
1798.    resultux [xx7] += uxsum [xx7];
1799.
1800.
1801.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1802.    thetasum [xx7] = theta [xx7][contactpnt7];
1803.    resulttheta [xx7] += thetasum [xx7];
1804.    alpha [xx7] = alpha1 + resulttheta [xx7];
1805.    }
```

```
1806.
1807.    }
1808.
1809.    }
1810.
1811.    next7:
1812.    ;}
1813.
1814.
1815.
1816.    int xx8 = 0, qq8 = 0, contactpnt8 = 0, l8, i8;
1817.
1818.
1819.    for (i8 = 0; i8 < isobox8; i8= i8+1)
1820.
1821.    {
1822.
1823.    xx8 = isocounterbox8[i8];
1824.
1825.    rightisosceles [xx8][0][0] = rightisosceles [xx8][0][0] + resultux [
   xx8] ;
1826.
1827.    rightisosceles [xx8][0][1] = rightisosceles [xx8][0][1] + resultuy [
   xx8] ;
1828.
1829.
1830.
1831.    for ( l8 = 0; l8 < isobox8; l8 = l8+1)
1832.
1833.    {
1834.
1835.
1836.    qq8 = isocounterbox8[l8];
1837.
1838.
1839.
1840.    if ( ((pow((rightisosceles [xx8][0][0] -
    rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) > 0) && ((pow((rightisosceles [xx8][0][0
   ] - rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) <= 3.125))
1841.
1842.    {
1843.
1844.    contactpnt8 = contactpnt8 + 1;
1845.
1846.    if (contactpnt8 >= 4) {goto next8;}
1847.
1848.
1849.    if ( rightisosceles [qq8][3][1] >  rightisosceles [xx8][1][1] )
1850.
1851.    {
```

```
1852.
1853.
1854.    ydisplat [xx8][contactpnt8] = ydisplacement2  + (deluy [xx8][contact
   pnt8] -
    deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][0] - rightisosceles [xx8][0][0])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][0] -
    rightisosceles [qq8][0][0]));
1855.    xdisplat [xx8][contactpnt8] = xdisplacement2 + (delux [xx8][contactp
   nt8] -
    delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][1] - rightisosceles [xx8][0][1])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][1] -
   rightisosceles [qq8][0][1]));
1856.
1857.
1858.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1859.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1860.
1861.
1862.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt8] * (-
   1) *  (dstiffness_coefficient/1000);
1863.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt8] *  (dstiffness_coe
   fficient/1000);
1864.
1865.
1866.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1867.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1868.
1869.
1870.    yforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * sin (alpha1)) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha1)));
1871.    xforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * cos (alpha1)) + ((fn [xx8][contactpnt8] + dn [xx8][contactpn
   t8]) * sin (alpha1)));
1872.
1873.    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
1874.    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);
1875.
1876.
1877.    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
1878.    y [xx8] = fysum [xx8][contactpnt8];
1879.    resultfy [xx8] += y[xx8];
1880.
1881.
1882.    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
1883.    x [xx8] = fxsum [xx8][contactpnt8];
```

```
1884.    resultfx [xx8] += x[xx8];
1885.
1886.
1887.    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
   es [qq8][3][0] - rightisosceles [xx8][0][0]) -
    xforce [xx8][contactpnt8] * ( rightisosceles [qq8][3][1] -
    rightisosceles [xx8][0][1]));
1888.    m [xx8] = msum [xx8][contactpnt8];
1889.    resultm [xx8] += m[xx8];
1890.
1891.
1892.    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1893.    udotysum [xx8] = udoty [xx8][contactpnt8];
1894.    resultudoty [xx8] += udotysum [xx8];
1895.
1896.
1897.    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1898.    udotxsum [xx8] = udotx [xx8][contactpnt8];
1899.    resultudotx [xx8] += udotxsum [xx8];
1900.
1901.
1902.    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
   ement)*1000/ ii); //1/s
1903.    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
1904.    resultthetadot [xx8] += thetadotsum [xx8];
1905.
1906.
1907.    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
   t;
1908.    deluysum [xx8] = deluy [xx8][contactpnt8];
1909.    resultdeluy [xx8] += deluysum [xx8];
1910.
1911.
1912.    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
   t;
1913.    deluxsum [xx8] = delux [xx8][contactpnt8];
1914.    resultdelux [xx8] += deluxsum [xx8];
1915.
1916.    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
   crement;
1917.    delthetasum [xx8] = deltheta [xx8][contactpnt8];
1918.    resultdeltheta [xx8] += delthetasum [xx8];
1919.
1920.
1921.    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
1922.    uysum [xx8] = uy [xx8][contactpnt8];
1923.    resultuy [xx8]+= uysum [xx8];
1924.
1925.
1926.    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
```

```
1927.    uxsum [xx8] = ux [xx8][contactpnt8];
1928.    resultux [xx8] += uxsum [xx8];
1929.
1930.
1931.    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
1932.    thetasum [xx8] = theta [xx8][contactpnt8];
1933.    resulttheta [xx8] += thetasum [xx8];
1934.    alpha [xx8] = alpha1 + resulttheta [xx8];
1935.
1936.    }
1937.
1938.    else
1939.
1940.    {
1941.
1942.
1943.    ydisplat [xx8][contactpnt8] += ydisplacement2  + (deluy [xx8][contac
   tpnt8] -
     deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][0] - rightisosceles [xx8][0][0])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][0][0] -
     rightisosceles [qq8][0][0]));
1944.    xdisplat [xx8][contactpnt8] += xdisplacement2 + (delux [xx8][contact
   pnt8] -
     delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][1] - rightisosceles [xx8][0][1])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][1][1] -
   rightisosceles [qq8][0][1]));
1945.
1946.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1947.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1948.
1949.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1950.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1951.
1952.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1953.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1954.
1955.    yforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * sin (alpha [xx8])) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha [xx8])));

1956.    xforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * cos (alpha [xx8])) + ((fn [xx8][contactpnt8] + dn [xx8][conta
   ctpnt8]) * sin (alpha [xx8])));
1957.
```

```
    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);

    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
    x [xx8] = fxsum [xx8][contactpnt8];
    resultfx [xx8] += x[xx8];


    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
    y [xx8] = fysum [xx8][contactpnt8];
    resultfy [xx8] += y[xx8];


    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
es [qq8][1][0] - rightisosceles [xx8][0][0]) -
     xforce [xx8][contactpnt8] * ( rightisosceles [qq8][1][1] -
     rightisosceles [xx8][0][1]));
    m [xx8] = msum [xx8][contactpnt8];
    resultm [xx8] += m[xx8];


    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
ent)* 1000/ dmass); // mm/sec
    udotysum [xx8] = udoty [xx8][contactpnt8];
    resultudoty [xx8] += udotysum [xx8];


    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx8] = udotx [xx8][contactpnt8];
    resultudotx [xx8] += udotxsum [xx8];


    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
    resultthetadot [xx8] += thetadotsum [xx8];


    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
t;
    deluysum [xx8] = deluy [xx8][contactpnt8];
    resultdeluy [xx8] += deluysum [xx8];


    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
t;
    deluxsum [xx8] = delux [xx8][contactpnt8];
    resultdelux [xx8] += deluxsum [xx8];


```

```
    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
crement;
    delthetasum [xx8] = deltheta [xx8][contactpnt8];
    resultdeltheta [xx8] += delthetasum [xx8];


    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
    uysum [xx8] = uy [xx8][contactpnt8];
    resultuy [xx8]+= uysum [xx8];


    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
    uxsum [xx8] = ux [xx8][contactpnt8];
    resultux [xx8] += uxsum [xx8];


    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
    thetasum [xx8] = theta [xx8][contactpnt8];
    resulttheta [xx8] += thetasum [xx8];
    alpha [xx8] = alpha1 + resulttheta [xx8];


    }
    }

    }

    next8:
    ;}


    int xx9 = 0, qq9 = 0, contactpnt9 = 0, l9, i9;


    for (i9 = 0; i9 < isobox9; i9= i9+1)

    {

    xx9 = isocounterbox9[i9];

    rightisosceles [xx9][0][0] = rightisosceles [xx9][0][0] + resultux [
xx9] ;

    rightisosceles [xx9][0][1] = rightisosceles [xx9][0][1] + resultuy [
xx9] ;


    for ( l9 = 0; l9 < isobox9; l9 = l9+1)

    {
```

```
2050.
2051.
2052.    qq9 = isocounterbox9[19];
2053.
2054.
2055.
2056.    if ( ((pow((rightisosceles [xx9][0][0] -
    rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) > 0) && ((pow((rightisosceles [xx9][0][0
   ] - rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) <= 3.125))
2057.
2058.    {
2059.
2060.    contactpnt9 = contactpnt9 + 1;
2061.
2062.    if (contactpnt9 >= 4) {goto next9;}
2063.
2064.
2065.    if ( rightisosceles [qq9][3][1] >  rightisosceles [xx9][1][1] )
2066.
2067.    {
2068.
2069.
2070.    ydisplat [xx9][contactpnt9] = ydisplacement2  + (deluy [xx9][contact
   pnt9] -
    deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][0] - rightisosceles [xx9][0][0])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][0] -
    rightisosceles [qq9][0][0]));
2071.    xdisplat [xx9][contactpnt9] = xdisplacement2 + (delux [xx9][contactp
   nt9] -
    delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][1] - rightisosceles [xx9][0][1])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][1] -
   rightisosceles [qq9][0][1]));
2072.
2073.
2074.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2075.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2076.
2077.
2078.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2079.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2080.
2081.
2082.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
```

```
    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
ient/1000);


    yforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * sin (alpha1)) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha1)));
    xforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * cos (alpha1)) + ((fn [xx9][contactpnt9] + dn [xx9][contactpn
t9]) * sin (alpha1)));

    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);


    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
    y [xx9] = fysum [xx9][contactpnt9];
    resultfy [xx9] += y[xx9];


    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
    x [xx9] = fxsum [xx9][contactpnt9];
    resultfx [xx9] += x[xx9];


    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
es [qq9][3][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][3][1] -
     rightisosceles [xx9][0][1]));
    m [xx9] = msum [xx9][contactpnt9];
    resultm [xx9] += m[xx9];


    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx9] = udoty [xx9][contactpnt9];
    resultudoty [xx9] += udotysum [xx9];


    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


```

```
2123.    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
   t;
2124.    deluysum [xx9] = deluy [xx9][contactpnt9];
2125.    resultdeluy [xx9] += deluysum [xx9];
2126.
2127.
2128.    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
   t;
2129.    deluxsum [xx9] = delux [xx9][contactpnt9];
2130.    resultdelux [xx9] += deluxsum [xx9];
2131.
2132.    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
   crement;
2133.    delthetasum [xx9] = deltheta [xx9][contactpnt9];
2134.    resultdeltheta [xx9] += delthetasum [xx9];
2135.
2136.
2137.    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
2138.    uysum [xx9] = uy [xx9][contactpnt9];
2139.    resultuy [xx9]+= uysum [xx9];
2140.
2141.
2142.    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
2143.    uxsum [xx9] = ux [xx9][contactpnt9];
2144.    resultux [xx9] += uxsum [xx9];
2145.
2146.
2147.    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
2148.    thetasum [xx9] = theta [xx9][contactpnt9];
2149.    resulttheta [xx9] += thetasum [xx9];
2150.    alpha [xx9] = alpha1 + resulttheta [xx9];
2151.
2152.    }
2153.
2154.    else
2155.
2156.    {
2157.
2158.
2159.    ydisplat [xx9][contactpnt9] += ydisplacement2  + (deluy [xx9][contac
   tpnt9] -
     deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][0] - rightisosceles [xx9][0][0])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][0][0] -
     rightisosceles [qq9][0][0]));
2160.    xdisplat [xx9][contactpnt9] += xdisplacement2 + (delux [xx9][contact
   pnt9] -
     delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][1] - rightisosceles [xx9][0][1])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][1][1] -
   rightisosceles [qq9][0][1]));
2161.
```

```
2162.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2163.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2164.
2165.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2166.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2167.
2168.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
2169.    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
   ient/1000);
2170.
2171.    yforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * sin (alpha [xx9])) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha [xx9])));

2172.    xforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * cos (alpha [xx9])) + ((fn [xx9][contactpnt9] + dn [xx9][conta
   ctpnt9]) * sin (alpha [xx9])));
2173.
2174.    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
2175.    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);
2176.
2177.    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
2178.    x [xx9] = fxsum [xx9][contactpnt9];
2179.    resultfx [xx9] += x[xx9];
2180.
2181.
2182.    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
2183.    y [xx9] = fysum [xx9][contactpnt9];
2184.    resultfy [xx9] += y[xx9];
2185.
2186.
2187.    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
   es [qq9][1][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][1][1] -
     rightisosceles [xx9][0][1]));
2188.    m [xx9] = msum [xx9][contactpnt9];
2189.    resultm [xx9] += m[xx9];
2190.
2191.
2192.    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
2193.    udotysum [xx9] = udoty [xx9][contactpnt9];
2194.    resultudoty [xx9] += udotysum [xx9];
2195.
2196.
2197.    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
   ent)*1000/ dmass); //mm/sec
```

```
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
t;
    deluysum [xx9] = deluy [xx9][contactpnt9];
    resultdeluy [xx9] += deluysum [xx9];


    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
t;
    deluxsum [xx9] = delux [xx9][contactpnt9];
    resultdelux [xx9] += deluxsum [xx9];


    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
crement;
    delthetasum [xx9] = deltheta [xx9][contactpnt9];
    resultdeltheta [xx9] += delthetasum [xx9];


    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
    uysum [xx9] = uy [xx9][contactpnt9];
    resultuy [xx9]+= uysum [xx9];


    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
    uxsum [xx9] = ux [xx9][contactpnt9];
    resultux [xx9] += uxsum [xx9];


    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
    thetasum [xx9] = theta [xx9][contactpnt9];
    resulttheta [xx9] += thetasum [xx9];
    alpha [xx9] = alpha1 + resulttheta [xx9];

    }
    }

    }

    next9:
    ;}

```

```
2246.
2247.    int xx10 = 0, qq10 = 0, contactpnt10 = 0, l10, i10;
2248.
2249.
2250.    for (i10 = 0; i10 < isobox10; i10= i10+1)
2251.
2252.    {
2253.
2254.    xx10 = isocounterbox10[i10];
2255.
2256.    rightisosceles [xx10][0][0] = rightisosceles [xx10][0][0] + resultux
    [xx10] ;
2257.
2258.    rightisosceles [xx10][0][1] = rightisosceles [xx10][0][1] + resultuy
    [xx10] ;
2259.
2260.
2261.
2262.    for ( l10 = 0; l10 < isobox10; l10 = l10+1)
2263.
2264.    {
2265.
2266.
2267.    qq10 = isocounterbox10[l10];
2268.
2269.
2270.
2271.    if ( ((pow((rightisosceles [xx10][0][0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) > 0) && ((pow((rightisosceles [xx10][0]
   [0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) <= 3.125))
2272.
2273.    {
2274.
2275.    contactpnt10 = contactpnt10 + 1;
2276.
2277.    if (contactpnt10 >= 4) {goto next10;}
2278.
2279.
2280.    if ( rightisosceles [qq10][3][1] >  rightisosceles [xx10][1][1] )
2281.
2282.    {
2283.
2284.
2285.    ydisplat [xx10][contactpnt10] = ydisplacement2  + (deluy [xx10][cont
   actpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][0] -
    rightisosceles [qq10][0][0]));
```

```
2286.    xdisplat [xx10][contactpnt10] = xdisplacement2 + (delux [xx10][conta
   ctpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][1] -
   rightisosceles [qq10][0][1]));
2287.
2288.
2289.    delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2290.    delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2291.
2292.
2293.    fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2294.    fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2295.
2296.
2297.    dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2298.    ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2299.
2300.
2301.    yforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * sin (alpha1)) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha1)));
2302.    xforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * cos (alpha1)) + ((fn [xx10][contactpnt10] + dn [xx10][
   contactpnt10]) * sin (alpha1)));
2303.
2304.    yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2305.    xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2306.
2307.
2308.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2309.    y [xx10] = fysum [xx10][contactpnt10];
2310.    resultfy [xx10] += y[xx10];
2311.
2312.
2313.    fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2314.    x [xx10] = fxsum [xx10][contactpnt10];
2315.    resultfx [xx10] += x[xx10];
2316.
2317.
2318.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][3][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][3][1] -
    rightisosceles [xx10][0][1]));
2319.    m [xx10] = msum [xx10][contactpnt10];
```

```
2320.    resultm [xx10] += m[xx10];
2321.
2322.
2323.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement) *1000/ dmass); // mm/sec
2324.    udotysum [xx10] = udoty [xx10][contactpnt10];
2325.    resultudoty [xx10] += udotysum [xx10];
2326.
2327.
2328.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2329.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2330.    resultudotx [xx10] += udotxsum [xx10];
2331.
2332.
2333.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2334.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2335.    resultthetadot [xx10] += thetadotsum [xx10];
2336.
2337.
2338.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2339.    deluysum [xx10] = deluy [xx10][contactpnt10];
2340.    resultdeluy [xx10] += deluysum [xx10];
2341.
2342.
2343.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2344.    deluxsum [xx10] = delux [xx10][contactpnt10];
2345.    resultdelux [xx10] += deluxsum [xx10];
2346.
2347.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2348.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2349.    resultdeltheta [xx10] += delthetasum [xx10];
2350.
2351.
2352.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2353.    uysum [xx10] = uy [xx10][contactpnt10];
2354.    resultuy [xx10]+= uysum [xx10];
2355.
2356.
2357.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2358.    uxsum [xx10] = ux [xx10][contactpnt10];
2359.    resultux [xx10] += uxsum [xx10];
2360.
2361.
2362.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2363.    thetasum [xx10] = theta [xx10][contactpnt10];
2364.    resultthetа [xx10] += thetasum [xx10];
2365.    alpha [xx10] = alpha1 + resultthetа [xx10];
```

```
2366.
2367.    }
2368.
2369.    else
2370.
2371.    {
2372.
2373.
2374.    ydisplat [xx10][contactpnt10] += ydisplacement2  + (deluy [xx10][con
   tactpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][0][0] -
    rightisosceles [qq10][0][0]));
2375.    xdisplat [xx10][contactpnt10] += xdisplacement2 + (delux [xx10][cont
   actpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][1][1] -
   rightisosceles [qq10][0][1]));
2376.
2377.    delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2378.    delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2379.
2380.    fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2381.    fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2382.
2383.    dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2384.    ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2385.
2386.    yforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * sin (alpha [xx10])) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha [xx10])
   ));
2387.    xforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * cos (alpha [xx10])) + ((fn [xx10][contactpnt10] + dn [x
   x10][contactpnt10]) * sin (alpha [xx10])));
2388.
2389.    yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2390.    xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2391.
2392.    fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2393.    x [xx10] = fxsum [xx10][contactpnt10];
2394.    resultfx [xx10] += x[xx10];
2395.
2396.
```

```
2397.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2398.    y [xx10] = fysum [xx10][contactpnt10];
2399.    resultfy [xx10] += y[xx10];
2400.
2401.
2402.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][1][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][1][1] -
    rightisosceles [xx10][0][1]));
2403.    m [xx10] = msum [xx10][contactpnt10];
2404.    resultm [xx10] += m[xx10];
2405.
2406.
2407.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2408.    udotysum [xx10] = udoty [xx10][contactpnt10];
2409.    resultudoty [xx10] += udotysum [xx10];
2410.
2411.
2412.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2413.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2414.    resultudotx [xx10] += udotxsum [xx10];
2415.
2416.
2417.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2418.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2419.    resultthetadot [xx10] += thetadotsum [xx10];
2420.
2421.
2422.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2423.    deluysum [xx10] = deluy [xx10][contactpnt10];
2424.    resultdeluy [xx10] += deluysum [xx10];
2425.
2426.
2427.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2428.    deluxsum [xx10] = delux [xx10][contactpnt10];
2429.    resultdelux [xx10] += deluxsum [xx10];
2430.
2431.
2432.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2433.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2434.    resultdeltheta [xx10] += delthetasum [xx10];
2435.
2436.
2437.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2438.    uysum [xx10] = uy [xx10][contactpnt10];
2439.    resultuy [xx10]+= uysum [xx10];
```

```
2440.
2441.
2442.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2443.    uxsum [xx10] = ux [xx10][contactpnt10];
2444.    resultux [xx10] += uxsum [xx10];
2445.
2446.
2447.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2448.    thetasum [xx10] = theta [xx10][contactpnt10];
2449.    resulttheta [xx10] += thetasum [xx10];
2450.    alpha [xx10] = alpha1 + resulttheta [xx10];
2451.
2452.    }
2453.    }
2454.
2455.    }
2456.
2457.    next10:
2458.    ;}
2459.
2460.
2461.    int xx11 = 0, qq11 = 0, contactpnt11 = 0, l11, i11;
2462.
2463.
2464.    for (i11 = 0; i11 < isobox11; i11= i11+1)
2465.
2466.    {
2467.
2468.    xx11 = isocounterbox11[i11];
2469.
2470.    rightisosceles [xx11][0][0] = rightisosceles [xx11][0][0] + resultux
    [xx11] ;
2471.
2472.    rightisosceles [xx11][0][1] = rightisosceles [xx11][0][1] + resultuy
    [xx11] ;
2473.
2474.
2475.
2476.    for ( l11 = 0; l11 < isobox11; l11 = l11+1)
2477.
2478.    {
2479.
2480.
2481.    qq11 = isocounterbox11[l11];
2482.
2483.
2484.
2485.    if ( ((pow((rightisosceles [xx11][0][0] -
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) > 0) && ((pow((rightisosceles [xx11][0]
   [0] -
```

```
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) <= 3.125))
2486.
2487.    {
2488.
2489.    contactpnt11 = contactpnt11 + 1;
2490.
2491.    if (contactpnt11 >= 4) {goto next11;}
2492.
2493.
2494.    if ( rightisosceles [qq11][3][1] >  rightisosceles [xx11][1][1] )
2495.
2496.    {
2497.
2498.
2499.    ydisplat [xx11][contactpnt11] = ydisplacement2  + (deluy [xx11][cont
   actpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][0] -
     rightisosceles [qq11][0][0]));
2500.    xdisplat [xx11][contactpnt11] = xdisplacement2 + (delux [xx11][conta
   ctpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][1] -
   rightisosceles [qq11][0][1]));
2501.
2502.
2503.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2504.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2505.
2506.
2507.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
2508.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2509.
2510.
2511.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2512.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2513.
2514.
2515.    yforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * sin (alpha1)) -
     ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha1)));
```

```
2516.    xforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * cos (alpha1)) + ((fn [xx11][contactpnt11] + dn [xx11][
   contactpnt11]) * sin (alpha1)));
2517.
2518.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2519.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2520.
2521.
2522.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2523.    y [xx11] = fysum [xx11][contactpnt11];
2524.    resultfy [xx11] += y[xx11];
2525.
2526.
2527.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2528.    x [xx11] = fxsum [xx11][contactpnt11];
2529.    resultfx [xx11] += x[xx11];
2530.
2531.
2532.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][3][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][3][1] -
    rightisosceles [xx11][0][1]));
2533.    m [xx11] = msum [xx11][contactpnt11];
2534.    resultm [xx11] += m[xx11];
2535.
2536.
2537.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement) *1000/ dmass); // mm/sec
2538.    udotysum [xx11] = udoty [xx11][contactpnt11];
2539.    resultudoty [xx11] += udotysum [xx11];
2540.
2541.
2542.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2543.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2544.    resultudotx [xx11] += udotxsum [xx11];
2545.
2546.
2547.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2548.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
2549.    resultthetadot [xx11] += thetadotsum [xx11];
2550.
2551.
2552.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2553.    deluysum [xx11] = deluy [xx11][contactpnt11];
2554.    resultdeluy [xx11] += deluysum [xx11];
2555.
2556.
2557.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
```

```
2558.    deluxsum [xx11] = delux [xx11][contactpnt11];
2559.    resultdelux [xx11] += deluxsum [xx11];
2560.
2561.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2562.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2563.    resultdeltheta [xx11] += delthetasum [xx11];
2564.
2565.
2566.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2567.    uysum [xx11] = uy [xx11][contactpnt11];
2568.    resultuy [xx11]+= uysum [xx11];
2569.
2570.
2571.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2572.    uxsum [xx11] = ux [xx11][contactpnt11];
2573.    resultux [xx11] += uxsum [xx11];
2574.
2575.
2576.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2577.    thetasum [xx11] = theta [xx11][contactpnt11];
2578.    resulttheta [xx11] += thetasum [xx11];
2579.    alpha [xx11] = alpha1 + resulttheta [xx11];
2580.
2581.    }
2582.
2583.    else
2584.
2585.    {
2586.
2587.
2588.
2589.    ydisplat [xx11][contactpnt11] += ydisplacement2  + (deluy [xx11][con
   tactpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][0][0] -
     rightisosceles [qq11][0][0]));
2590.    xdisplat [xx11][contactpnt11] += xdisplacement2 + (delux [xx11][cont
   actpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][1][1] -
   rightisosceles [qq11][0][1]));
2591.
2592.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2593.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2594.
2595.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
2596.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2597.
2598.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2599.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2600.
2601.    yforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * sin (alpha [xx11])) -
    ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha [xx11])
   ));
2602.    xforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * cos (alpha [xx11])) + ((fn [xx11][contactpnt11] + dn [x
   x11][contactpnt11]) * sin (alpha [xx11])));
2603.
2604.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2605.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2606.
2607.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2608.    x [xx11] = fxsum [xx11][contactpnt11];
2609.    resultfx [xx11] += x[xx11];
2610.
2611.
2612.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2613.    y [xx11] = fysum [xx11][contactpnt11];
2614.    resultfy [xx11] += y[xx11];
2615.
2616.
2617.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][1][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][1][1] -
    rightisosceles [xx11][0][1]));
2618.    m [xx11] = msum [xx11][contactpnt11];
2619.    resultm [xx11] += m[xx11];
2620.
2621.
2622.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2623.    udotysum [xx11] = udoty [xx11][contactpnt11];
2624.    resultudoty [xx11] += udotysum [xx11];
2625.
2626.
2627.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2628.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2629.    resultudotx [xx11] += udotxsum [xx11];
2630.
2631.
2632.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2633.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
```

```
    resultthetadot [xx11] += thetadotsum [xx11];


    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
ement;
    deluysum [xx11] = deluy [xx11][contactpnt11];
    resultdeluy [xx11] += deluysum [xx11];


    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
ement;
    deluxsum [xx11] = delux [xx11][contactpnt11];
    resultdelux [xx11] += deluxsum [xx11];


    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
e_increment;
    delthetasum [xx11] = deltheta [xx11][contactpnt11];
    resultdeltheta [xx11] += delthetasum [xx11];


    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
    uysum [xx11] = uy [xx11][contactpnt11];
    resultuy [xx11]+= uysum [xx11];


    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
    uxsum [xx11] = ux [xx11][contactpnt11];
    resultux [xx11] += uxsum [xx11];


    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
    thetasum [xx11] = theta [xx11][contactpnt11];
    resulttheta [xx11] += thetasum [xx11];
    alpha [xx11] = alpha1 + resulttheta [xx11];



    }

    }

    }

    next11:
    ;}


    int xx12 = 0, qq12 = 0, contactpnt12 = 0, l12, i12;

```

```
    for (i12 = 0; i12 < isobox12; i12= i12+1)

    {

    xx12 = isocounterbox12[i12];

    rightisosceles [xx12][0][0] = rightisosceles [xx12][0][0] + resultux
   [xx12] ;

    rightisosceles [xx12][0][1] = rightisosceles [xx12][0][1] + resultuy
   [xx12] ;


    for ( l12 = 0; l12 < isobox12; l12 = l12+1)

    {


    qq12 = isocounterbox12[l12];


    if ( ((pow((rightisosceles [xx12][0][0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) > 0) && ((pow((rightisosceles [xx12][0]
   [0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) <= 3.125))

    {

    contactpnt12 = contactpnt12 + 1;

    if (contactpnt12 >= 4) {goto next12;}


    if ( rightisosceles [qq12][3][1] >  rightisosceles [xx12][1][1] )

    {


    ydisplat [xx12][contactpnt12] = ydisplacement2  + (deluy [xx12][cont
   actpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][0] -
    rightisosceles [qq12][0][0]));
    xdisplat [xx12][contactpnt12] = xdisplacement2 + (delux [xx12][conta
   ctpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][1] - rightisosceles [xx12][0][1])  -
```

```
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][1] -
   rightisosceles [qq12][0][1]));
2720.
2721.
2722.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2723.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2724.
2725.
2726.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2727.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2728.
2729.
2730.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2731.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2732.
2733.
2734.    yforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * sin (alpha1)) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha1)));
2735.    xforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * cos (alpha1)) + ((fn [xx12][contactpnt12] + dn [xx12][
   contactpnt12]) * sin (alpha1)));
2736.
2737.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2738.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2739.
2740.
2741.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2742.    y [xx12] = fysum [xx12][contactpnt12];
2743.    resultfy [xx12] += y[xx12];
2744.
2745.
2746.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2747.    x [xx12] = fxsum [xx12][contactpnt12];
2748.    resultfx [xx12] += x[xx12];
2749.
2750.
2751.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][3][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][3][1] -
    rightisosceles [xx12][0][1]));
2752.    m [xx12] = msum [xx12][contactpnt12];
2753.    resultm [xx12] += m[xx12];
2754.
2755.
```

```
2756.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement) *1000/ dmass); // mm/sec
2757.    udotysum [xx12] = udoty [xx12][contactpnt12];
2758.    resultudoty [xx12] += udotysum [xx12];
2759.
2760.
2761.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2762.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2763.    resultudotx [xx12] += udotxsum [xx12];
2764.
2765.
2766.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2767.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2768.    resultthetadot [xx12] += thetadotsum [xx12];
2769.
2770.
2771.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2772.    deluysum [xx12] = deluy [xx12][contactpnt12];
2773.    resultdeluy [xx12] += deluysum [xx12];
2774.
2775.
2776.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2777.    deluxsum [xx12] = delux [xx12][contactpnt12];
2778.    resultdelux [xx12] += deluxsum [xx12];
2779.
2780.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2781.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2782.    resultdeltheta [xx12] += delthetasum [xx12];
2783.
2784.
2785.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2786.    uysum [xx12] = uy [xx12][contactpnt12];
2787.    resultuy [xx12]+= uysum [xx12];
2788.
2789.
2790.    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
2791.    uxsum [xx12] = ux [xx12][contactpnt12];
2792.    resultux [xx12] += uxsum [xx12];
2793.
2794.
2795.    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
2796.    thetasum [xx12] = theta [xx12][contactpnt12];
2797.    resulttheta [xx12] += thetasum [xx12];
2798.    alpha [xx12] = alpha1 + resulttheta [xx12];
2799.
2800.    }
2801.
```

```
    else

    {


    ydisplat [xx12][contactpnt12] += ydisplacement2  + (deluy [xx12][con
tactpnt12] -
     deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
eles [qq12][1][0] - rightisosceles [xx12][0][0])  -
     deltheta [qq12][contactpnt12] * (rightisosceles [qq12][0][0] -
     rightisosceles [qq12][0][0]));
    xdisplat [xx12][contactpnt12] += xdisplacement2 + (delux [xx12][cont
actpnt12] -
     delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
eles [qq12][1][1] - rightisosceles [xx12][0][1])  -
     deltheta [qq12][contactpnt12] * (rightisosceles [qq12][1][1] -
rightisosceles [qq12][0][1]));

    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));

    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
1) *  (dstiffness_coefficient/1000);
    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
_coefficient/1000);

    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
1) * (ddamping_coefficient/1000);
    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
fficient/1000);

    yforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
][contactpnt12]) * sin (alpha [xx12])) -
     ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha [xx12])
));
    xforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
][contactpnt12]) * cos (alpha [xx12])) + ((fn [xx12][contactpnt12] + dn [x
x12][contactpnt12]) * sin (alpha [xx12])));

    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);

    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
    x [xx12] = fxsum [xx12][contactpnt12];
    resultfx [xx12] += x[xx12];


    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
    y [xx12] = fysum [xx12][contactpnt12];
```

```
2833.    resultfy [xx12] += y[xx12];
2834.
2835.
2836.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][1][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][1][1] -
    rightisosceles [xx12][0][1]));
2837.    m [xx12] = msum [xx12][contactpnt12];
2838.    resultm [xx12] += m[xx12];
2839.
2840.
2841.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2842.    udotysum [xx12] = udoty [xx12][contactpnt12];
2843.    resultudoty [xx12] += udotysum [xx12];
2844.
2845.
2846.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2847.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2848.    resultudotx [xx12] += udotxsum [xx12];
2849.
2850.
2851.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2852.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2853.    resultthetadot [xx12] += thetadotsum [xx12];
2854.
2855.
2856.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2857.    deluysum [xx12] = deluy [xx12][contactpnt12];
2858.    resultdeluy [xx12] += deluysum [xx12];
2859.
2860.
2861.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2862.    deluxsum [xx12] = delux [xx12][contactpnt12];
2863.    resultdelux [xx12] += deluxsum [xx12];
2864.
2865.
2866.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2867.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2868.    resultdeltheta [xx12] += delthetasum [xx12];
2869.
2870.
2871.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2872.    uysum [xx12] = uy [xx12][contactpnt12];
2873.    resultuy [xx12]+= uysum [xx12];
2874.
2875.
```

```
    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
    uxsum [xx12] = ux [xx12][contactpnt12];
    resultux [xx12] += uxsum [xx12];


    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
    thetasum [xx12] = theta [xx12][contactpnt12];
    resulttheta [xx12] += thetasum [xx12];
    alpha [xx12] = alpha1 + resulttheta [xx12];

    }
    }

    }

    next12:
    ;}


    int xx13 = 0, qq13 = 0, contactpnt13 = 0, l13, i13;


    for (i13 = 0; i13 < isobox13; i13= i13+1)

    {

    xx13 = isocounterbox13[i13];


    rightisosceles [xx13][0][0] = rightisosceles [xx13][0][0] + resultux
    [xx13] ;

    rightisosceles [xx13][0][1] = rightisosceles [xx13][0][1] + resultuy
    [xx13] ;


    for ( l13 = 0; l13 < isobox13; l13 = l13+1)

    {


    qq13 = isocounterbox13[l13];


    if ( ((pow((rightisosceles [xx13][0][0] -
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) > 0) && ((pow((rightisosceles [xx13][0]
   [0] -
```

```
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) <= 3.125))
2923.
2924.    {
2925.
2926.    contactpnt13 = contactpnt13 + 1;
2927.
2928.    if (contactpnt13 >= 4) {goto next13;}
2929.
2930.
2931.    if ( rightisosceles [qq13][3][1] >  rightisosceles [xx13][1][1] )
2932.
2933.    {
2934.
2935.
2936.    ydisplat [xx13][contactpnt13] = ydisplacement2  + (deluy [xx13][cont
   actpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][0] -
    rightisosceles [qq13][0][0]));
2937.    xdisplat [xx13][contactpnt13] = xdisplacement2 + (delux [xx13][conta
   ctpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][1] -
   rightisosceles [qq13][0][1]));
2938.
2939.
2940.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
2941.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
2942.
2943.
2944.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
2945.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
2946.
2947.
2948.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
2949.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
2950.
2951.
2952.    yforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * sin (alpha1)) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha1)));
```

```
2953.    xforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * cos (alpha1)) + ((fn [xx13][contactpnt13] + dn [xx13][
   contactpnt13]) * sin (alpha1)));
2954.
2955.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
2956.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
2957.
2958.
2959.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
2960.    y [xx13] = fysum [xx13][contactpnt13];
2961.    resultfy [xx13] += y[xx13];
2962.
2963.
2964.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
2965.    x [xx13] = fxsum [xx13][contactpnt13];
2966.    resultfx [xx13] += x[xx13];
2967.
2968.
2969.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][3][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][3][1] -
    rightisosceles [xx13][0][1]));
2970.    m [xx13] = msum [xx13][contactpnt13];
2971.    resultm [xx13] += m[xx13];
2972.
2973.
2974.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement) *1000/ dmass); // mm/sec
2975.    udotysum [xx13] = udoty [xx13][contactpnt13];
2976.    resultudoty [xx13] += udotysum [xx13];
2977.
2978.
2979.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
2980.    udotxsum [xx13] = udotx [xx13][contactpnt13];
2981.    resultudotx [xx13] += udotxsum [xx13];
2982.
2983.
2984.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
2985.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
2986.    resultthetadot [xx13] += thetadotsum [xx13];
2987.
2988.
2989.    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
   ement;
2990.    deluysum [xx13] = deluy [xx13][contactpnt13];
2991.    resultdeluy [xx13] += deluysum [xx13];
2992.
2993.
2994.    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
   ement;
```

```
2995.    deluxsum [xx13] = delux [xx13][contactpnt13];
2996.    resultdelux [xx13] += deluxsum [xx13];
2997.
2998.    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
   e_increment;
2999.    delthetasum [xx13] = deltheta [xx13][contactpnt13];
3000.    resultdeltheta [xx13] += delthetasum [xx13];
3001.
3002.
3003.    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
3004.    uysum [xx13] = uy [xx13][contactpnt13];
3005.    resultuy [xx13]+= uysum [xx13];
3006.
3007.
3008.    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
3009.    uxsum [xx13] = ux [xx13][contactpnt13];
3010.    resultux [xx13] += uxsum [xx13];
3011.
3012.
3013.    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
3014.    thetasum [xx13] = theta [xx13][contactpnt13];
3015.    resulttheta [xx13] += thetasum [xx13];
3016.    alpha [xx13] = alpha1 + resulttheta [xx13];
3017.
3018.    }
3019.
3020.    else
3021.
3022.    {
3023.
3024.
3025.
3026.    ydisplat [xx13][contactpnt13] += ydisplacement2  + (deluy [xx13][con
   tactpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][0][0] -
    rightisosceles [qq13][0][0]));
3027.    xdisplat [xx13][contactpnt13] += xdisplacement2 + (delux [xx13][cont
   actpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][1][1] -
   rightisosceles [qq13][0][1]));
3028.
3029.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
3030.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
3031.
3032.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
3033.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
3034.
3035.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
3036.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
3037.
3038.    yforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * sin (alpha [xx13])) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha [xx13])
   ));
3039.    xforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * cos (alpha [xx13])) + ((fn [xx13][contactpnt13] + dn [x
   x13][contactpnt13]) * sin (alpha [xx13])));
3040.
3041.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
3042.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
3043.
3044.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
3045.    x [xx13] = fxsum [xx13][contactpnt13];
3046.    resultfx [xx13] += x[xx13];
3047.
3048.
3049.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
3050.    y [xx13] = fysum [xx13][contactpnt13];
3051.    resultfy [xx13] += y[xx13];
3052.
3053.
3054.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][1][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][1][1] -
    rightisosceles [xx13][0][1]));
3055.    m [xx13] = msum [xx13][contactpnt13];
3056.    resultm [xx13] += m[xx13];
3057.
3058.
3059.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3060.    udotysum [xx13] = udoty [xx13][contactpnt13];
3061.    resultudoty [xx13] += udotysum [xx13];
3062.
3063.
3064.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
3065.    udotxsum [xx13] = udotx [xx13][contactpnt13];
3066.    resultudotx [xx13] += udotxsum [xx13];
3067.
3068.
3069.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
3070.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
```

```
    resultthetadot [xx13] += thetadotsum [xx13];


    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
ement;
    deluysum [xx13] = deluy [xx13][contactpnt13];
    resultdeluy [xx13] += deluysum [xx13];


    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
ement;
    deluxsum [xx13] = delux [xx13][contactpnt13];
    resultdelux [xx13] += deluxsum [xx13];


    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
e_increment;
    delthetasum [xx13] = deltheta [xx13][contactpnt13];
    resultdeltheta [xx13] += delthetasum [xx13];


    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
    uysum [xx13] = uy [xx13][contactpnt13];
    resultuy [xx13]+= uysum [xx13];


    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
    uxsum [xx13] = ux [xx13][contactpnt13];
    resultux [xx13] += uxsum [xx13];


    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
    thetasum [xx13] = theta [xx13][contactpnt13];
    resulttheta [xx13] += thetasum [xx13];
    alpha [xx13] = alpha1 + resulttheta [xx13];



    }

    }

    }

    next13:
    ;}


    int xx14 = 0, qq14 = 0, contactpnt14 = 0, l14, i14;

```

```
3120.    for (i14 = 0; i14 < isobox14; i14= i14+1)
3121.
3122.    {
3123.
3124.    xx14 = isocounterbox14[i14];
3125.
3126.    rightisosceles [xx14][0][0] = rightisosceles [xx14][0][0] + resultux
    [xx14] ;
3127.
3128.    rightisosceles [xx14][0][1] = rightisosceles [xx14][0][1] + resultuy
    [xx14] ;
3129.
3130.
3131.
3132.    for ( l14 = 0; l14 < isobox14; l14 = l14+1)
3133.
3134.    {
3135.
3136.
3137.    qq14 = isocounterbox14[l14];
3138.
3139.
3140.
3141.    if ( ((pow((rightisosceles [xx14][0][0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) > 0) && ((pow((rightisosceles [xx14][0]
   [0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) <= 3.125))
3142.
3143.    {
3144.
3145.    contactpnt14 = contactpnt14 + 1;
3146.
3147.    if (contactpnt14 >= 4) {goto next14;}
3148.
3149.
3150.    if ( rightisosceles [qq14][3][1] >  rightisosceles [xx14][1][1] )
3151.
3152.    {
3153.
3154.
3155.    ydisplat [xx14][contactpnt14] = ydisplacement2  + (deluy [xx14][cont
   actpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][0] -
    rightisosceles [qq14][0][0]));
3156.    xdisplat [xx14][contactpnt14] = xdisplacement2 + (delux [xx14][conta
   ctpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][1] - rightisosceles [xx14][0][1])  -
```

```
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][1] -
   rightisosceles [qq14][0][1]));
3157.
3158.
3159.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3160.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3161.
3162.
3163.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3164.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3165.
3166.
3167.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3168.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3169.
3170.
3171.    yforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * sin (alpha1)) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha1)));
3172.    xforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * cos (alpha1)) + ((fn [xx14][contactpnt14] + dn [xx14][
   contactpnt14]) * sin (alpha1)));
3173.
3174.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3175.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3176.
3177.
3178.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3179.    y [xx14] = fysum [xx14][contactpnt14];
3180.    resultfy [xx14] += y[xx14];
3181.
3182.
3183.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3184.    x [xx14] = fxsum [xx14][contactpnt14];
3185.    resultfx [xx14] += x[xx14];
3186.
3187.
3188.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][3][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][3][1] -
    rightisosceles [xx14][0][1]));
3189.    m [xx14] = msum [xx14][contactpnt14];
3190.    resultm [xx14] += m[xx14];
3191.
3192.
```

```
3193.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement) *1000/ dmass); // mm/sec
3194.    udotysum [xx14] = udoty [xx14][contactpnt14];
3195.    resultudoty [xx14] += udotysum [xx14];
3196.
3197.
3198.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3199.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3200.    resultudotx [xx14] += udotxsum [xx14];
3201.
3202.
3203.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3204.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3205.    resultthetadot [xx14] += thetadotsum [xx14];
3206.
3207.
3208.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3209.    deluysum [xx14] = deluy [xx14][contactpnt14];
3210.    resultdeluy [xx14] += deluysum [xx14];
3211.
3212.
3213.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3214.    deluxsum [xx14] = delux [xx14][contactpnt14];
3215.    resultdelux [xx14] += deluxsum [xx14];
3216.
3217.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3218.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3219.    resultdeltheta [xx14] += delthetasum [xx14];
3220.
3221.
3222.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3223.    uysum [xx14] = uy [xx14][contactpnt14];
3224.    resultuy [xx14]+= uysum [xx14];
3225.
3226.
3227.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3228.    uxsum [xx14] = ux [xx14][contactpnt14];
3229.    resultux [xx14] += uxsum [xx14];
3230.
3231.
3232.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3233.    thetasum [xx14] = theta [xx14][contactpnt14];
3234.    resultttheta [xx14] += thetasum [xx14];
3235.    alpha [xx14] = alpha1 + resultttheta [xx14];
3236.
3237.    }
3238.
```

```
3239.    else
3240.
3241.    {
3242.
3243.
3244.
3245.    ydisplat [xx14][contactpnt14] += ydisplacement2  + (deluy [xx14][con
   tactpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][0][0] -
    rightisosceles [qq14][0][0]));
3246.    xdisplat [xx14][contactpnt14] += xdisplacement2 + (delux [xx14][cont
   actpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][1] - rightisosceles [xx14][0][1])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][1][1] -
   rightisosceles [qq14][0][1]));
3247.
3248.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3249.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3250.
3251.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3252.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3253.
3254.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3255.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3256.
3257.    yforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * sin (alpha [xx14])) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha [xx14])
   ));
3258.    xforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * cos (alpha [xx14])) + ((fn [xx14][contactpnt14] + dn [x
   x14][contactpnt14]) * sin (alpha [xx14])));
3259.
3260.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3261.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3262.
3263.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3264.    x [xx14] = fxsum [xx14][contactpnt14];
3265.    resultfx [xx14] += x[xx14];
3266.
3267.
3268.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3269.    y [xx14] = fysum [xx14][contactpnt14];
```

```
3270.    resultfy [xx14] += y[xx14];
3271.
3272.
3273.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][1][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][1][1] -
    rightisosceles [xx14][0][1]));
3274.    m [xx14] = msum [xx14][contactpnt14];
3275.    resultm [xx14] += m[xx14];
3276.
3277.
3278.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3279.    udotysum [xx14] = udoty [xx14][contactpnt14];
3280.    resultudoty [xx14] += udotysum [xx14];
3281.
3282.
3283.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3284.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3285.    resultudotx [xx14] += udotxsum [xx14];
3286.
3287.
3288.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3289.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3290.    resultthetadot [xx14] += thetadotsum [xx14];
3291.
3292.
3293.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3294.    deluysum [xx14] = deluy [xx14][contactpnt14];
3295.    resultdeluy [xx14] += deluysum [xx14];
3296.
3297.
3298.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3299.    deluxsum [xx14] = delux [xx14][contactpnt14];
3300.    resultdelux [xx14] += deluxsum [xx14];
3301.
3302.
3303.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3304.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3305.    resultdeltheta [xx14] += delthetasum [xx14];
3306.
3307.
3308.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3309.    uysum [xx14] = uy [xx14][contactpnt14];
3310.    resultuy [xx14]+= uysum [xx14];
3311.
3312.
```

```
3313.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3314.    uxsum [xx14] = ux [xx14][contactpnt14];
3315.    resultux [xx14] += uxsum [xx14];
3316.
3317.
3318.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3319.    thetasum [xx14] = theta [xx14][contactpnt14];
3320.    resulttheta [xx14] += thetasum [xx14];
3321.    alpha [xx14] = alpha1 + resulttheta [xx14];
3322.
3323.    }
3324.    }
3325.
3326.    }
3327.
3328.    next14:
3329.    ;}
3330.
3331.
3332.
3333.    int xx15 = 0, qq15 = 0, contactpnt15 = 0, l15, i15;
3334.
3335.
3336.    for (i15 = 0; i15 < isobox15; i15= i15+1)
3337.
3338.    {
3339.
3340.    xx15 = isocounterbox15[i15];
3341.
3342.    rightisosceles [xx15][0][0] = rightisosceles [xx15][0][0] + resultux
    [xx15] ;
3343.
3344.    rightisosceles [xx15][0][1] = rightisosceles [xx15][0][1] + resultuy
    [xx15] ;
3345.
3346.
3347.
3348.    for ( l15 = 0; l15 < isobox15; l15 = l15+1)
3349.
3350.    {
3351.
3352.
3353.    qq15 = isocounterbox15[l15];
3354.
3355.
3356.
3357.    if ( ((pow((rightisosceles [xx15][0][0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) > 0) && ((pow((rightisosceles [xx15][0]
   [0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) <= 3.125))
```

```
3358.
3359.    {
3360.
3361.    contactpnt15 = contactpnt15 + 1;
3362.
3363.    if (contactpnt15 >= 4) {goto next15;}
3364.
3365.
3366.    if ( rightisosceles [qq15][3][1] >  rightisosceles [xx15][1][1] )
3367.
3368.    {
3369.
3370.
3371.    ydisplat [xx15][contactpnt15] = ydisplacement2  + (deluy [xx15][cont
   actpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][0] -
    rightisosceles [qq15][0][0]));
3372.    xdisplat [xx15][contactpnt15] = xdisplacement2 + (delux [xx15][conta
   ctpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][1] -
   rightisosceles [qq15][0][1]));
3373.
3374.
3375.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3376.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3377.
3378.
3379.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3380.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3381.
3382.
3383.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3384.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3385.
3386.
3387.    yforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * sin (alpha1)) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha1)));
3388.    xforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * cos (alpha1)) + ((fn [xx15][contactpnt15] + dn [xx15][
   contactpnt15]) * sin (alpha1)));
3389.
```

```
3390.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3391.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3392.
3393.
3394.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3395.    y [xx15] = fysum [xx15][contactpnt15];
3396.    resultfy [xx15] += y[xx15];
3397.
3398.
3399.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3400.    x [xx15] = fxsum [xx15][contactpnt15];
3401.    resultfx [xx15] += x[xx15];
3402.
3403.
3404.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][3][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][3][1] -
    rightisosceles [xx15][0][1]));
3405.    m [xx15] = msum [xx15][contactpnt15];
3406.    resultm [xx15] += m[xx15];
3407.
3408.
3409.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement) *1000/ dmass); // mm/sec
3410.    udotysum [xx15] = udoty [xx15][contactpnt15];
3411.    resultudoty [xx15] += udotysum [xx15];
3412.
3413.
3414.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3415.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3416.    resultudotx [xx15] += udotxsum [xx15];
3417.
3418.
3419.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3420.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3421.    resultthetadot [xx15] += thetadotsum [xx15];
3422.
3423.
3424.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3425.    deluysum [xx15] = deluy [xx15][contactpnt15];
3426.    resultdeluy [xx15] += deluysum [xx15];
3427.
3428.
3429.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3430.    deluxsum [xx15] = delux [xx15][contactpnt15];
3431.    resultdelux [xx15] += deluxsum [xx15];
3432.
```

```
3433.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3434.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3435.    resultdeltheta [xx15] += delthetasum [xx15];
3436.
3437.
3438.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3439.    uysum [xx15] = uy [xx15][contactpnt15];
3440.    resultuy [xx15]+= uysum [xx15];
3441.
3442.
3443.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3444.    uxsum [xx15] = ux [xx15][contactpnt15];
3445.    resultux [xx15] += uxsum [xx15];
3446.
3447.
3448.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3449.    thetasum [xx15] = theta [xx15][contactpnt15];
3450.    resulttheta [xx15] += thetasum [xx15];
3451.    alpha [xx15] = alpha1 + resulttheta [xx15];
3452.
3453.    }
3454.
3455.    else
3456.
3457.    {
3458.
3459.
3460.
3461.    ydisplat [xx15][contactpnt15] += ydisplacement2  + (deluy [xx15][con
   tactpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][0][0] -
    rightisosceles [qq15][0][0]));
3462.    xdisplat [xx15][contactpnt15] += xdisplacement2 + (delux [xx15][cont
   actpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][1][1] -
   rightisosceles [qq15][0][1]));
3463.
3464.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3465.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3466.
3467.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3468.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3469.
```

```
3470.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3471.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3472.
3473.    yforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * sin (alpha [xx15])) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha [xx15])
   ));
3474.    xforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * cos (alpha [xx15])) + ((fn [xx15][contactpnt15] + dn [x
   x15][contactpnt15]) * sin (alpha [xx15])));
3475.
3476.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3477.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3478.
3479.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3480.    x [xx15] = fxsum [xx15][contactpnt15];
3481.    resultfx [xx15] += x[xx15];
3482.
3483.
3484.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3485.    y [xx15] = fysum [xx15][contactpnt15];
3486.    resultfy [xx15] += y[xx15];
3487.
3488.
3489.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][1][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][1][1] -
    rightisosceles [xx15][0][1]));
3490.    m [xx15] = msum [xx15][contactpnt15];
3491.    resultm [xx15] += m[xx15];
3492.
3493.
3494.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3495.    udotysum [xx15] = udoty [xx15][contactpnt15];
3496.    resultudoty [xx15] += udotysum [xx15];
3497.
3498.
3499.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3500.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3501.    resultudotx [xx15] += udotxsum [xx15];
3502.
3503.
3504.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3505.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3506.    resultthetadot [xx15] += thetadotsum [xx15];
3507.
3508.
```

```
3509.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3510.    deluysum [xx15] = deluy [xx15][contactpnt15];
3511.    resultdeluy [xx15] += deluysum [xx15];
3512.
3513.
3514.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3515.    deluxsum [xx15] = delux [xx15][contactpnt15];
3516.    resultdelux [xx15] += deluxsum [xx15];
3517.
3518.
3519.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3520.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3521.    resultdeltheta [xx15] += delthetasum [xx15];
3522.
3523.
3524.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3525.    uysum [xx15] = uy [xx15][contactpnt15];
3526.    resultuy [xx15]+= uysum [xx15];
3527.
3528.
3529.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3530.    uxsum [xx15] = ux [xx15][contactpnt15];
3531.    resultux [xx15] += uxsum [xx15];
3532.
3533.
3534.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3535.    thetasum [xx15] = theta [xx15][contactpnt15];
3536.    resulttheta [xx15] += thetasum [xx15];
3537.    alpha [xx15] = alpha1 + resulttheta [xx15];
3538.
3539.    }
3540.    }
3541.
3542.    }
3543.
3544.    next15:
3545.    ;}
3546.
3547.
3548.
3549.    int xx16 = 0, qq16 = 0, contactpnt16 = 0, l16, i16;
3550.
3551.
3552.    for (i16 = 0; i16 < isobox16; i16= i16+1)
3553.
3554.    {
3555.
3556.    xx16 = isocounterbox16[i16];
3557.
```

```
3558.
3559.
3560.    rightisosceles [xx16][0][0] = rightisosceles [xx16][0][0] + resultux
    [xx16] ;
3561.
3562.    rightisosceles [xx16][0][1] = rightisosceles [xx16][0][1] + resultuy
    [xx16] ;
3563.
3564.
3565.
3566.    for ( l16 = 0; l16 < isobox16; l16 = l16+1)
3567.
3568.    {
3569.
3570.
3571.    qq16 = isocounterbox16[l16];
3572.
3573.
3574.
3575.    if ( ((pow((rightisosceles [xx16][0][0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) > 0) && ((pow((rightisosceles [xx16][0]
   [0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) <= 3.125))
3576.
3577.    {
3578.
3579.    contactpnt16 = contactpnt16 + 1;
3580.
3581.    if (contactpnt16 >= 4) {goto next16;}
3582.
3583.
3584.    if ( rightisosceles [qq16][3][1] >  rightisosceles [xx16][1][1] )
3585.
3586.    {
3587.
3588.
3589.    ydisplat [xx16][contactpnt16] = ydisplacement2  + (deluy [xx16][cont
   actpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][0] -
    rightisosceles [qq16][0][0]));
3590.    xdisplat [xx16][contactpnt16] = xdisplacement2 + (delux [xx16][conta
   ctpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][1] -
   rightisosceles [qq16][0][1]));
3591.
3592.
```

```
3593.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3594.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3595.
3596.
3597.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3598.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3599.
3600.
3601.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3602.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3603.
3604.
3605.    yforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * sin (alpha1)) -
     ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha1)));
3606.    xforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * cos (alpha1)) + ((fn [xx16][contactpnt16] + dn [xx16][
   contactpnt16]) * sin (alpha1)));
3607.
3608.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3609.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3610.
3611.
3612.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3613.    y [xx16] = fysum [xx16][contactpnt16];
3614.    resultfy [xx16] += y[xx16];
3615.
3616.
3617.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3618.    x [xx16] = fxsum [xx16][contactpnt16];
3619.    resultfx [xx16] += x[xx16];
3620.
3621.
3622.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][3][0] - rightisosceles [xx16][0][0]) -
     xforce [xx16][contactpnt16] * ( rightisosceles [qq16][3][1] -
     rightisosceles [xx16][0][1]));
3623.    m [xx16] = msum [xx16][contactpnt16];
3624.    resultm [xx16] += m[xx16];
3625.
3626.
3627.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement) *1000/ dmass); // mm/sec
3628.    udotysum [xx16] = udoty [xx16][contactpnt16];
3629.    resultudoty [xx16] += udotysum [xx16];
3630.
```

```
3631.
3632.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3633.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3634.    resultudotx [xx16] += udotxsum [xx16];
3635.
3636.
3637.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3638.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3639.    resultthetadot [xx16] += thetadotsum [xx16];
3640.
3641.
3642.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3643.    deluysum [xx16] = deluy [xx16][contactpnt16];
3644.    resultdeluy [xx16] += deluysum [xx16];
3645.
3646.
3647.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3648.    deluxsum [xx16] = delux [xx16][contactpnt16];
3649.    resultdelux [xx16] += deluxsum [xx16];
3650.
3651.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3652.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3653.    resultdeltheta [xx16] += delthetasum [xx16];
3654.
3655.
3656.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3657.    uysum [xx16] = uy [xx16][contactpnt16];
3658.    resultuy [xx16]+= uysum [xx16];
3659.
3660.
3661.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3662.    uxsum [xx16] = ux [xx16][contactpnt16];
3663.    resultux [xx16] += uxsum [xx16];
3664.
3665.
3666.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3667.    thetasum [xx16] = theta [xx16][contactpnt16];
3668.    resultttheta [xx16] += thetasum [xx16];
3669.    alpha [xx16] = alpha1 + resultttheta [xx16];
3670.
3671.    }
3672.
3673.    else
3674.
3675.    {
3676.
3677.
```

```
3678.
3679.    ydisplat [xx16][contactpnt16] += ydisplacement2  + (deluy [xx16][con
   tactpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][0][0] -
    rightisosceles [qq16][0][0]));
3680.    xdisplat [xx16][contactpnt16] += xdisplacement2 + (delux [xx16][cont
   actpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][1][1] -
   rightisosceles [qq16][0][1]));
3681.
3682.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3683.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3684.
3685.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3686.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3687.
3688.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3689.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3690.
3691.    yforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * sin (alpha [xx16])) -
    ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha [xx16])
   ));
3692.    xforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * cos (alpha [xx16])) + ((fn [xx16][contactpnt16] + dn [x
   x16][contactpnt16]) * sin (alpha [xx16])));
3693.
3694.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3695.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3696.
3697.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3698.    x [xx16] = fxsum [xx16][contactpnt16];
3699.    resultfx [xx16] += x[xx16];
3700.
3701.
3702.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3703.    y [xx16] = fysum [xx16][contactpnt16];
3704.    resultfy [xx16] += y[xx16];
3705.
3706.
3707.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][1][0] - rightisosceles [xx16][0][0]) -
```

```
    xforce [xx16][contactpnt16] * ( rightisosceles [qq16][1][1] -
    rightisosceles [xx16][0][1]));
3708.    m [xx16] = msum [xx16][contactpnt16];
3709.    resultm [xx16] += m[xx16];
3710.
3711.
3712.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3713.    udotysum [xx16] = udoty [xx16][contactpnt16];
3714.    resultudoty [xx16] += udotysum [xx16];
3715.
3716.
3717.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3718.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3719.    resultudotx [xx16] += udotxsum [xx16];
3720.
3721.
3722.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3723.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3724.    resultthetadot [xx16] += thetadotsum [xx16];
3725.
3726.
3727.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3728.    deluysum [xx16] = deluy [xx16][contactpnt16];
3729.    resultdeluy [xx16] += deluysum [xx16];
3730.
3731.
3732.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3733.    deluxsum [xx16] = delux [xx16][contactpnt16];
3734.    resultdelux [xx16] += deluxsum [xx16];
3735.
3736.
3737.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3738.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3739.    resultdeltheta [xx16] += delthetasum [xx16];
3740.
3741.
3742.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3743.    uysum [xx16] = uy [xx16][contactpnt16];
3744.    resultuy [xx16]+= uysum [xx16];
3745.
3746.
3747.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3748.    uxsum [xx16] = ux [xx16][contactpnt16];
3749.    resultux [xx16] += uxsum [xx16];
3750.
3751.
```

```
3752.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3753.    thetasum [xx16] = theta [xx16][contactpnt16];
3754.    resulttheta [xx16] += thetasum [xx16];
3755.    alpha [xx16] = alpha1 + resulttheta [xx16];
3756.
3757.
3758.
3759.    }
3760.
3761.    }
3762.
3763.    }
3764.
3765.    next16:
3766.    ;}
3767.
3768.
3769.    ;}
3770.
3771.    display:
3772.
3773.
3774.     a_file<< dmass;
3775.
3776.    return 0;
3777.    }
```

Appendix G:

Code for Tailings-DEM (Series 4)

```
# include <iostream>
# include <cmath>
# include <fstream>
#include <numeric>

using namespace std;


 struct geometricalshapes  {  // subroutine for assigning values to the particle geometrical shapes
 float side1, side2, side3, side4, side5;
 float angle1, angle2, angle3, angle4, angle5;
 float height, base, centroidx, centroidy, momentofinertiax, momentofinertiay, area;

};

float ddamping_coefficient, dk;
float dstiffness_coefficient;
float dtime_increment;
float dmass;
float ii;  // mass moment of inertia
float dresult;


int isocounterbox1[100], isocounterbox2[100], isocounterbox3[100], isocounterbox4[100], isocounterbox5[100], isocounterbox6[100], isocounterbox7[100], isocounterbox8[100], isocounterbox9[100], isocounterbox10[100], isocounterbox11[100], isocounterbox12[100], isocounterbox13[100], isocounterbox14[100], isocounterbox15[100], isocounterbox16[100];
float p, p2;
int bb;


float ydisplat [100][100], xdisplat [100][100], delus [100][100], delun [100][100], fn [100][100], fs [100][000], dn [100][100], ds [100][100];
float yforce [100][100], xforce [100][100], fxsum [100][100], x [100], resultfx [100], fysum [100][100];
float y [100], resultfy [100], msum [100][100], m [100], resultm [100], udoty [100][100], udotysum [100], resultudoty [100];
float udotx [100][100], udotxsum [100], resultudotx [100], thetadot [100][100], thetadotsum [100], resultthetadot [100];
float deluy [100][100], deluysum [100], resultdeluy [100], delux [100][100], deluxsum [100], resultdelux [100], deltheta [100][100];
float delthetasum [100], resultdeltheta [100], uy [100][100], uysum [100], resultuy [100], ux [100][100], uxsum [100];
float resultux [100], theta [100][100], thetasum [100], resulttheta [100];


float rightisosceles [150][4][10];
int isoscelescounter = 0;
```

```
float ydisplacement2 = 0.0, yvelocity2= 0;
float xdisplacement2 = 0.0;
float alpha [100], alpha1 = 0.785398;
int v = 1;
float n, n1;
int num3 =  1 + (39-4)/10;    // number of isosceles triangles per row
int hh, oo, pp;


int main ()

{

geometricalshapes righttriangle; //the particle
righttriangle.side1 = 4.243;
righttriangle.side2 = 3;
righttriangle.side3 = 3;
righttriangle.angle1 = 45;
righttriangle.angle2 = 45;
righttriangle.angle3 = 90;
righttriangle.height = 4.24;
righttriangle.base = 8.486;
righttriangle.centroidx = 0;
righttriangle.centroidy = 0.707;
righttriangle.momentofinertiax = 1.123;
righttriangle.momentofinertiay = 3.374;
righttriangle.area = 17.99;

 ofstream a_file ("data-s4-mw16.txt");


 cout<< "Please enter the stiffness coefficient in N/m" << endl;
 cin>> dstiffness_coefficient;


 cout<< "Please enter the mass of the particle in kg" << endl;
 cin>> dmass;

ii =  ((righttriangle.base/2) * (righttriangle.base/2)* (dmass)/3) ;


 // time increment verification

 begin: //goto label
 cout<< "Please enter the time increment" << endl;
 cin>> dtime_increment;

 dresult = (2 * sqrt (dmass/dstiffness_coefficient));
 if (dtime_increment >=  dresult ) // time increment condition

```

```
92.{
93.
94. cout<< "Time increment does not satisfy condition"<< endl;
95. goto begin;
96.
97.}
98.
99. else
100.
101.      {
102.        cout<< "Time increment satisfies condition"<< endl;
103.
104.     }
105.
106.
107.     dk = 2 * (sqrt (dmass * dstiffness_coefficient)); // subroutine for
    calculating the damping coefficient
108.     ddamping_coefficient = dk/dtime_increment;
109.
110.
111.
112.
113.
114.
115.     for (hh = 0; hh <150; hh = hh+1) {
116.         for (oo = 0; oo <4; oo = oo+1) { for (pp = 0; pp <10; pp = pp +
    1)
117.         {
118.             rightisosceles [hh][oo][pp] = 0.0;
119.
120.
121.         }
122.         }
123.     }
124.
125.
126.
127.      for (n = 3.0 ; n <= 70; n = n + 10) {       // positioning particle
    s
128.         for (n1 = 4.0 ; n1 <= 39; n1 = n1 + 10) {
129.
130.         rightisosceles [v][0][0] = n1 ;            // right isosceles x
    coordinates
131.         rightisosceles [v][1][0] = rightisosceles [v][0][0] + (righttria
    ngle.base /2);
132.         rightisosceles [v][2][0] = rightisosceles [v][0][0] -
     (righttriangle.base /2);
133.         rightisosceles [v][3][0] = rightisosceles [v][0][0];
134.
135.         rightisosceles [v][0][1] = n ;    // right isosceles y coordinat
    es
```

```
136.         rightisosceles [v][1][1] = rightisosceles [v][0][1] + (righttria
   ngle.height /3);
137.         rightisosceles [v][2][1] = rightisosceles [v][1][1];
138.         rightisosceles [v][3][1] = rightisosceles [v][2][1] -
    (righttriangle.height);
139.
140.
141.
142.         v = v + 1;
143.         isoscelescounter = isoscelescounter + 1;
144.
145.
146.
147.     }
148.     }
149.
150.     p2 = 10/num3;
151.
152.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
153.     ydisplacement2 =  ydisplacement2 + (yvelocity2 * dtime_increment);

154.
155.
156.
157.
158.
159.     for (p = 100; p < 100000 ; p = p+100)  // load cycle in Newtons
160.     {
161.     p2 = p/num3;
162.
163.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
164.
165.
166.       a_file <<  p << endl;
167.
168.
169.     int isobox1 = 0, isobox2 = 0, isobox3 = 0, isobox4 = 0, isobox5 = 0,
    isobox6 = 0, isobox7 = 0, isobox8 = 0, isobox9 = 0, isobox10 = 0, isobox1
   1 = 0, isobox12 = 0, isobox13 = 0, isobox14 = 0, isobox15 = 0, isobox16 =
   0;
170.
171.
172.     for (bb = 0; bb <100; bb = bb+1) {
173.         isocounterbox1[bb] = 0, isocounterbox2[bb] = 0, isocounterbox3[b
   b] = 0, isocounterbox4[bb]= 0;
174.         isocounterbox5[bb] = 0, isocounterbox6[bb] = 0, isocounterbox7[b
   b] = 0, isocounterbox8[bb] = 0;
175.         isocounterbox9[bb] = 0, isocounterbox10[bb] = 0, isocounterbox11
   [bb] = 0, isocounterbox12[bb] = 0;
```

```
176.          isocounterbox13[bb] = 0, isocounterbox14[bb] = 0, isocounterbox1
   5[bb] = 0, isocounterbox16[bb] = 0;
177.
178.          }
179.
180.
181.
182.     for (int dd = 1; dd <= isoscelescounter ; dd = dd + 1)  // locating
   the isosceles triangles within the 16 screen boxes
183.     {
184.
185.      if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][0]
   <= 11) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][1]<=
   18.5))
186.     {isocounterbox1[isobox1] = dd; isobox1 = isobox1 + 1;}
187.
188.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
189.     {isocounterbox2[isobox2] = dd; isobox2 = isobox2 + 1;}
190.
191.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
192.     {isocounterbox3[isobox3] = dd; isobox3 = isobox3 + 1;}
193.
194.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
195.     {isocounterbox4[isobox4] = dd; isobox4 = isobox4 + 1;}
196.
197.     else if( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][
   0] <= 11) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 37))
198.     {isocounterbox5[isobox5] = dd; isobox5 = isobox5 + 1;}
199.
200.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
201.     {isocounterbox6[isobox6] = dd; isobox6 = isobox6 + 1;}
202.
203.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
204.     {isocounterbox7[isobox7] = dd; isobox7 = isobox7 + 1;}
205.
206.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
207.     {isocounterbox8[isobox8] = dd; isobox8 = isobox8 + 1;}
208.
```

```
209.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 55.5))
210.     {isocounterbox9[isobox9] = dd; isobox9 = isobox9 + 1;}
211.
212.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
213.     {isocounterbox10[isobox10] = dd; isobox10 = isobox10 + 1;}
214.
215.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
216.     {isocounterbox11[isobox11] = dd; isobox11 = isobox11 + 1;}
217.
218.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
219.     {isocounterbox12[isobox12] = dd; isobox12 = isobox12 + 1;}
220.
221.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0
   ][1]<= 74))
222.     {isocounterbox13[isobox13] = dd; isobox13 = isobox13 + 1;}
223.
224.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
225.     {isocounterbox14[isobox14] = dd; isobox14 = isobox14 + 1;}
226.
227.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
228.     {isocounterbox15[isobox15] = dd; isobox15 = isobox15 + 1;}
229.
230.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
231.     {isocounterbox16[isobox16] = dd; isobox16 = isobox16 + 1;}
232.
233.
234.     }  // end of isosceles triangles location within the 16 screen boxes

235.
236.
237.     int gg, ww;
238.
239.     for (gg = 0; gg < 100; gg = gg+1) {
240.
241.      for (ww = 0; ww < 100; ww = ww +1) {
242.
243.     ydisplat [gg][ww] = 0.0;
```

```
    xdisplat [gg][ww] = 0.0;
    delus [gg][ww] = 0.0;
    delun [gg][ww] = 0.0;
    fn [gg][ww] = 0.0;
    fs [gg][ww] = 0.0;
    dn [gg][ww] = 0.0;
    ds [gg][ww] = 0.0;
    yforce [gg][ww] = 0.0;
    xforce [gg][ww] = 0.0;
    fxsum [gg][ww] = 0.0;
    x [ww] = 0.0;
    resultfx [ww] = 0.0;
    fysum [gg][ww] = 0.0;
    y [ww] = 0.0;
    resultfy [ww] = 0.0;
    msum [gg][ww] = 0.0;
    m [ww] = 0.0;
    resultm [ww] = 0.0;
    udoty [gg][ww] = 0.0;
    udotysum [ww] = 0.0;
    resultudoty [ww] = 0.0;
    udotx [gg][ww] = 0.0;
    udotxsum [ww] = 0.0;
    resultudotx [ww] = 0.0;
    thetadot [gg][ww] = 0.0;
    thetadotsum [ww] = 0.0;
    resultthetadot [ww] = 0.0;
    deluy [gg][ww] = 0.0;
    deluysum [ww] = 0.0;
    resultdeluy [ww] = 0.0;
    delux [gg][ww] = 0.0;
    deluxsum [ww] = 0.0;
    resultdelux [ww] =0.0;
    deltheta [gg][ww] = 0.0;
    delthetasum [ww] = 0.0;
    resultdeltheta [ww] = 0.0;
    uy [gg][ww] = 0.0;
    uysum [ww] = 0.0;
    resultuy [ww] = 0.0;
    ux [gg][ww] = 0.0;
    uxsum [ww] = 0.0;
    resultux [ww] = 0.0;
    theta [gg][ww] = 0.0;
    thetasum [ww] = 0.0;
    resulttheta [ww] = 0.0;
    alpha [ww] = 0.0;

    }

    }

```

```
296.
297.     int xx1 = 0, qq1 = 0, contactpnt1 = 0, l1, i1;
298.
299.
300.     for (i1 = 0; i1 < isobox1; i1= i1+1)
301.
302.     {
303.
304.     xx1 = isocounterbox1[i1];
305.
306.
307.
308.     rightisosceles [xx1][0][0] = rightisosceles [xx1][0][0] + resultux [
   xx1] ;
309.
310.     rightisosceles [xx1][0][1] = rightisosceles [xx1][0][1] + resultuy [
   xx1] ;
311.
312.
313.
314.     for ( l1 = 0; l1 < isobox1; l1 = l1+1)
315.
316.     {
317.
318.
319.     qq1 = isocounterbox1[l1];
320.
321.
322.     if ( ((pow((rightisosceles [xx1][0][0] -
    rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) > 0) && ((pow((rightisosceles [xx1][0][0
   ] - rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) <= 200))
323.
324.     {
325.
326.     contactpnt1 = contactpnt1 + 1;
327.
328.     if (contactpnt1 >= 4) {goto next1;}
329.
330.     if ( rightisosceles [qq1][3][1] >  rightisosceles [xx1][1][1] )
331.
332.     {
333.
334.
335.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][0] -
    rightisosceles [qq1][0][0]));
```

```
336.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][1] -
   rightisosceles [qq1][0][1]));
337.
338.
339.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
340.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
341.
342.
343.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
344.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
345.
346.
347.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
348.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
349.
350.
351.     yforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * sin (alpha1)) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha1)));
352.     xforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * cos (alpha1)) + ((fn [xx1][contactpnt1] + dn [xx1][contactpn
   t1]) * sin (alpha1)));
353.
354.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
355.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
356.
357.
358.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
359.     y [xx1] = fysum [xx1][contactpnt1];
360.     resultfy [xx1] += y[xx1];
361.
362.
363.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
364.     x [xx1] = fxsum [xx1][contactpnt1];
365.     resultfx [xx1] += x[xx1];
366.
367.
368.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][3][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][3][1] -
    rightisosceles [xx1][0][1]));
369.     m [xx1] = msum [xx1][contactpnt1];
```

```
370.     resultm [xx1] += m[xx1];
371.
372.
373.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
374.     udotysum [xx1] = udoty [xx1][contactpnt1];
375.     resultudoty [xx1] += udotysum [xx1];
376.
377.
378.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
379.     udotxsum [xx1] = udotx [xx1][contactpnt1];
380.     resultudotx [xx1] += udotxsum [xx1];
381.
382.
383.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
384.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
385.     resultthetadot [xx1] += thetadotsum [xx1];
386.
387.
388.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
389.     deluysum [xx1] = deluy [xx1][contactpnt1];
390.     resultdeluy [xx1] += deluysum [xx1];
391.
392.
393.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
394.     deluxsum [xx1] = delux [xx1][contactpnt1];
395.     resultdelux [xx1] += deluxsum [xx1];
396.
397.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
398.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
399.     resultdeltheta [xx1] += delthetasum [xx1];
400.
401.
402.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
403.     uysum [xx1] = uy [xx1][contactpnt1];
404.     resultuy [xx1]+= uysum [xx1];
405.
406.
407.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
408.     uxsum [xx1] = ux [xx1][contactpnt1];
409.     resultux [xx1] += uxsum [xx1];
410.
411.
412.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
413.     thetasum [xx1] = theta [xx1][contactpnt1];
414.     resulttheta [xx1] += thetasum [xx1];
415.     alpha [xx1] = alpha1 + resulttheta [xx1];
```

```
416.
417.     }
418.
419.
420.     else
421.     {
422.
423.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][0][0] -
    rightisosceles [qq1][0][0]));
424.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][1][1] -
   rightisosceles [qq1][0][1]));
425.
426.
427.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
428.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
429.
430.
431.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
432.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
433.
434.
435.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
436.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
437.
438.
439.     yforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * sin (alpha [xx1])) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha [xx1])));

440.     xforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * cos (alpha [xx1])) + ((fn [xx1][contactpnt1] + dn [xx1][conta
   ctpnt1]) * sin (alpha [xx1])));
441.
442.
443.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
444.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
445.
446.
```

```
447.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
448.     x [xx1] = fxsum [xx1][contactpnt1];
449.     resultfx [xx1] += x[xx1];
450.
451.
452.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
453.     y [xx1] = fysum [xx1][contactpnt1];
454.     resultfy [xx1] += y[xx1];
455.
456.
457.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][1][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][1][1] -
    rightisosceles [xx1][0][1]));
458.     m [xx1] = msum [xx1][contactpnt1];
459.     resultm [xx1] += m[xx1];
460.
461.
462.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
463.     udotysum [xx1] = udoty [xx1][contactpnt1];
464.     resultudoty [xx1] += udotysum [xx1];
465.
466.
467.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
468.     udotxsum [xx1] = udotx [xx1][contactpnt1];
469.     resultudotx [xx1] += udotxsum [xx1];
470.
471.
472.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
473.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
474.     resultthetadot [xx1] += thetadotsum [xx1];
475.
476.
477.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
478.     deluysum [xx1] = deluy [xx1][contactpnt1];
479.     resultdeluy [xx1] += deluysum [xx1];
480.
481.
482.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
483.     deluxsum [xx1] = delux [xx1][contactpnt1];
484.     resultdelux [xx1] += deluxsum [xx1];
485.
486.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
487.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
488.     resultdeltheta [xx1] += delthetasum [xx1];
489.
```

```
490.
491.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
492.     uysum [xx1] = uy [xx1][contactpnt1];
493.     resultuy [xx1]+= uysum [xx1];
494.
495.
496.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
497.     uxsum [xx1] = ux [xx1][contactpnt1];
498.     resultux [xx1] += uxsum [xx1];
499.
500.
501.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
502.     thetasum [xx1] = theta [xx1][contactpnt1];
503.     resulttheta [xx1] += thetasum [xx1];
504.     alpha [xx1] = alpha1 + resulttheta [xx1];
505.
506.
507.     }
508.
509.
510.
511.      a_file <<  p  << endl;
512.      a_file << xx1 << endl;
513.      a_file << resultuy [xx1]/10 << endl;
514.
515.     if ( resultuy [xx1] > 72 ) { goto display;}
516.
517.
518.
519.     }
520.
521.
522.     }
523.
524.     next1:
525.     ;}
526.
527.
528.
529.     int xx2 = 0, qq2 = 0, contactpnt2 = 0, l2, i2;
530.
531.
532.     for (i2 = 0; i2 < isobox2; i2= i2+1)
533.
534.     {
535.
536.     xx2 = isocounterbox2[i2];
537.
538.
539.     rightisosceles [xx2][0][0] = rightisosceles [xx2][0][0] + resultux [
    xx2] ;
540.
```

```
541.     rightisosceles [xx2][0][1] = rightisosceles [xx2][0][1] + resultuy [
   xx2] ;
542.
543.
544.     for ( l2 = 0; l2 < isobox2; l2 = l2+1)
545.
546.     {
547.
548.
549.     qq2 = isocounterbox2[l2];
550.
551.
552.
553.     if ( ((pow((rightisosceles [xx2][0][0] -
    rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
    rightisosceles [qq2][0][1]), 2)) > 0) && ((pow((rightisosceles [xx2][0][0
   ] - rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
    rightisosceles [qq2][0][1]), 2)) <= 200))
554.
555.     {
556.
557.     contactpnt2 = contactpnt2 + 1;
558.
559.     if (contactpnt2 >= 4) {goto next2;}
560.
561.
562.     if ( rightisosceles [qq2][3][1] >  rightisosceles [xx2][1][1] )
563.
564.     {
565.
566.
567.     ydisplat [xx2][contactpnt2] = ydisplacement2  + (deluy [xx2][contact
   pnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][3][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][0] -
    rightisosceles [qq2][0][0]));
568.     xdisplat [xx2][contactpnt2] = xdisplacement2 + (delux [xx2][contactp
   nt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][3][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][1] -
   rightisosceles [qq2][0][1]));
569.
570.
571.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
572.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
573.
574.
```

```
575.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
576.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
577.
578.
579.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
580.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
581.
582.
583.     yforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * sin (alpha1)) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha1)));
584.     xforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * cos (alpha1)) + ((fn [xx2][contactpnt2] + dn [xx2][contactpn
   t2]) * sin (alpha1)));
585.
586.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
587.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
588.
589.
590.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
591.     y [xx2] = fysum [xx2][contactpnt2];
592.     resultfy [xx2] += y[xx2];
593.
594.
595.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
596.     x [xx2] = fxsum [xx2][contactpnt2];
597.     resultfx [xx2] += x[xx2];
598.
599.
600.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][3][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][3][1] -
    rightisosceles [xx2][0][1]));
601.     m [xx2] = msum [xx2][contactpnt2];
602.     resultm [xx2] += m[xx2];
603.
604.
605.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent) *1000/ dmass); // mm/sec
606.     udotysum [xx2] = udoty [xx2][contactpnt2];
607.     resultudoty [xx2] += udotysum [xx2];
608.
609.
610.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
611.     udotxsum [xx2] = udotx [xx2][contactpnt2];
612.     resultudotx [xx2] += udotxsum [xx2];
613.
```

```
614.
615.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
616.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
617.     resultthetadot [xx2] += thetadotsum [xx2];
618.
619.
620.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
621.     deluysum [xx2] = deluy [xx2][contactpnt2];
622.     resultdeluy [xx2] += deluysum [xx2];
623.
624.
625.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
626.     deluxsum [xx2] = delux [xx2][contactpnt2];
627.     resultdelux [xx2] += deluxsum [xx2];
628.
629.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
630.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
631.     resultdeltheta [xx2] += delthetasum [xx2];
632.
633.
634.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
635.     uysum [xx2] = uy [xx2][contactpnt2];
636.     resultuy [xx2]+= uysum [xx2];
637.
638.
639.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
640.     uxsum [xx2] = ux [xx2][contactpnt2];
641.     resultux [xx2] += uxsum [xx2];
642.
643.
644.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
645.     thetasum [xx2] = theta [xx2][contactpnt2];
646.     resulttheta [xx2] += thetasum [xx2];
647.     alpha [xx2] = alpha1 + resulttheta [xx2];
648.
649.     }
650.
651.     else
652.
653.     {
654.
655.
656.     ydisplat [xx2][contactpnt2] += ydisplacement2  + (deluy [xx2][contac
   tpnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][0][0] -
    rightisosceles [qq2][0][0]));
```

```
657.     xdisplat [xx2][contactpnt2] += xdisplacement2 + (delux [xx2][contact
   pnt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][1][1] -
   rightisosceles [qq2][0][1]));
658.
659.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
660.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
661.
662.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
663.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
664.
665.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
666.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
667.
668.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * sin (alpha [xx2])) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

669.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
   ctpnt2]) * sin (alpha [xx2])));
670.
671.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
672.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
673.
674.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
675.     x [xx2] = fxsum [xx2][contactpnt2];
676.     resultfx [xx2] += x[xx2];
677.
678.
679.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
680.     y [xx2] = fysum [xx2][contactpnt2];
681.     resultfy [xx2] += y[xx2];
682.
683.
684.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][1][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][1][1] -
    rightisosceles [xx2][0][1]));
685.     m [xx2] = msum [xx2][contactpnt2];
686.     resultm [xx2] += m[xx2];
687.
688.
```

```
689.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
690.     udotysum [xx2] = udoty [xx2][contactpnt2];
691.     resultudoty [xx2] += udotysum [xx2];
692.
693.
694.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
695.     udotxsum [xx2] = udotx [xx2][contactpnt2];
696.     resultudotx [xx2] += udotxsum [xx2];
697.
698.
699.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
700.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
701.     resultthetadot [xx2] += thetadotsum [xx2];
702.
703.
704.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
705.     deluysum [xx2] = deluy [xx2][contactpnt2];
706.     resultdeluy [xx2] += deluysum [xx2];
707.
708.
709.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
710.     deluxsum [xx2] = delux [xx2][contactpnt2];
711.     resultdelux [xx2] += deluxsum [xx2];
712.
713.
714.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
715.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
716.     resultdeltheta [xx2] += delthetasum [xx2];
717.
718.
719.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
720.     uysum [xx2] = uy [xx2][contactpnt2];
721.     resultuy [xx2]+= uysum [xx2];
722.
723.
724.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
725.     uxsum [xx2] = ux [xx2][contactpnt2];
726.     resultux [xx2] += uxsum [xx2];
727.
728.
729.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
730.     thetasum [xx2] = theta [xx2][contactpnt2];
731.     resulttheta [xx2] += thetasum [xx2];
732.     alpha [xx2] = alpha1 + resulttheta [xx2];
733.
734.     }
```

```
735.     }
736.
737.     }
738.
739.     next2:
740.     ;}
741.
742.     int xx3 = 0, qq3 = 0, contactpnt3 = 0, l3, i3;
743.
744.
745.     for (i3 = 0; i3 < isobox3; i3= i3+1)
746.
747.     {
748.
749.     xx3 = isocounterbox3[i3];
750.
751.
752.     rightisosceles [xx3][0][0] = rightisosceles [xx3][0][0] + resultux [
    xx3] ;
753.
754.     rightisosceles [xx3][0][1] = rightisosceles [xx3][0][1] + resultuy [
    xx3] ;
755.
756.
757.     for ( l3 = 0; l3 < isobox3; l3 = l3+1)
758.
759.     {
760.
761.
762.     qq3 = isocounterbox3[l3];
763.
764.
765.
766.     if ( ((pow((rightisosceles [xx3][0][0] -
    rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) > 0) && ((pow((rightisosceles [xx3][0][0
    ] - rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
    rightisosceles [qq3][0][1]), 2)) <= 200))
767.
768.     {
769.
770.     contactpnt3 = contactpnt3 + 1;
771.
772.     if (contactpnt3 >= 4) {goto next3;}
773.
774.
775.     if ( rightisosceles [qq3][3][1] >  rightisosceles [xx3][1][1] )
776.
777.     {
778.
779.
```

```
780.     ydisplat [xx3][contactpnt3] = ydisplacement2  + (deluy [xx3][contact
   pnt3] -
    deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][0] - rightisosceles [xx3][0][0])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][0] -
    rightisosceles [qq3][0][0]));
781.     xdisplat [xx3][contactpnt3] = xdisplacement2 + (delux [xx3][contactp
   nt3] -
    delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][1] - rightisosceles [xx3][0][1])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][1] -
   rightisosceles [qq3][0][1]));
782.
783.
784.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
   1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
785.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
   1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
786.
787.
788.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
   1) *  (dstiffness_coefficient/1000);
789.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
   fficient/1000);
790.
791.
792.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
   1) * (ddamping_coefficient/1000);
793.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
   ient/1000);
794.
795.
796.     yforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * sin (alpha1)) -
    ((fn [xx3][contactpnt3] + dn [xx3][contactpnt3]) * cos (alpha1)));
797.     xforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * cos (alpha1)) + ((fn [xx3][contactpnt3] + dn [xx3][contactpn
   t3]) * sin (alpha1)));
798.
799.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
800.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
801.
802.
803.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
804.     y [xx3] = fysum [xx3][contactpnt3];
805.     resultfy [xx3] += y[xx3];
806.
807.
808.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
809.     x [xx3] = fxsum [xx3][contactpnt3];
810.     resultfx [xx3] += x[xx3];
811.
```

```
812.
813.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][3][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][3][1] -
    rightisosceles [xx3][0][1]));
814.     m [xx3] = msum [xx3][contactpnt3];
815.     resultm [xx3] += m[xx3];
816.
817.
818.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent) *1000/ dmass); // mm/sec
819.     udotysum [xx3] = udoty [xx3][contactpnt3];
820.     resultudoty [xx3] += udotysum [xx3];
821.
822.
823.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
824.     udotxsum [xx3] = udotx [xx3][contactpnt3];
825.     resultudotx [xx3] += udotxsum [xx3];
826.
827.
828.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
829.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
830.     resultthetadot [xx3] += thetadotsum [xx3];
831.
832.
833.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
834.     deluysum [xx3] = deluy [xx3][contactpnt3];
835.     resultdeluy [xx3] += deluysum [xx3];
836.
837.
838.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
839.     deluxsum [xx3] = delux [xx3][contactpnt3];
840.     resultdelux [xx3] += deluxsum [xx3];
841.
842.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
843.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
844.     resultdeltheta [xx3] += delthetasum [xx3];
845.
846.
847.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
848.     uysum [xx3] = uy [xx3][contactpnt3];
849.     resultuy [xx3]+= uysum [xx3];
850.
851.
852.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
853.     uxsum [xx3] = ux [xx3][contactpnt3];
854.     resultux [xx3] += uxsum [xx3];
```

```
855.
856.
857.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
858.     thetasum [xx3] = theta [xx3][contactpnt3];
859.     resulttheta [xx3] += thetasum [xx3];
860.     alpha [xx3] = alpha1 + resulttheta [xx3];
861.
862.     }
863.
864.      else
865.     {
866.
867.
868.     ydisplat [xx3][contactpnt2] += ydisplacement2  + (deluy [xx3][contac
   tpnt3] -
    deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][1][0] - rightisosceles [xx3][0][0])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][0][0] -
    rightisosceles [qq3][0][0]));
869.     xdisplat [xx3][contactpnt2] += xdisplacement2 + (delux [xx3][contact
   pnt3] -
    delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][1][1] - rightisosceles [xx3][0][1])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][1][1] -
   rightisosceles [qq3][0][1]));
870.
871.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
   1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
872.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
   1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
873.
874.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
   1) *  (dstiffness_coefficient/1000);
875.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
   fficient/1000);
876.
877.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
   1) * (ddamping_coefficient/1000);
878.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
   ient/1000);
879.
880.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * sin (alpha [xx2])) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

881.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
   ctpnt2]) * sin (alpha [xx2])));
882.
883.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
884.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
885.
```

```
886.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
887.     x [xx3] = fxsum [xx3][contactpnt3];
888.     resultfx [xx3] += x[xx3];
889.
890.
891.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
892.     y [xx3] = fysum [xx3][contactpnt3];
893.     resultfy [xx3] += y[xx3];
894.
895.
896.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][1][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][1][1] -
    rightisosceles [xx3][0][1]));
897.     m [xx3] = msum [xx3][contactpnt3];
898.     resultm [xx3] += m[xx3];
899.
900.
901.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
902.     udotysum [xx3] = udoty [xx3][contactpnt3];
903.     resultudoty [xx3] += udotysum [xx3];
904.
905.
906.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
907.     udotxsum [xx3] = udotx [xx3][contactpnt3];
908.     resultudotx [xx3] += udotxsum [xx3];
909.
910.
911.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
912.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
913.     resultthetadot [xx3] += thetadotsum [xx3];
914.
915.
916.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
917.     deluysum [xx3] = deluy [xx3][contactpnt3];
918.     resultdeluy [xx3] += deluysum [xx3];
919.
920.
921.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
922.     deluxsum [xx3] = delux [xx3][contactpnt3];
923.     resultdelux [xx3] += deluxsum [xx3];
924.
925.
926.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
927.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
928.     resultdeltheta [xx3] += delthetasum [xx3];
```

```
929.
930.
931.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
932.     uysum [xx3] = uy [xx3][contactpnt3];
933.     resultuy [xx3]+= uysum [xx3];
934.
935.
936.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
937.     uxsum [xx3] = ux [xx3][contactpnt3];
938.     resultux [xx3] += uxsum [xx3];
939.
940.
941.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
942.     thetasum [xx3] = theta [xx3][contactpnt3];
943.     resulttheta [xx3] += thetasum [xx3];
944.     alpha [xx3] = alpha1 + resulttheta [xx3];
945.     }
946.
947.     }
948.
949.     }
950.
951.     next3:
952.     ;}
953.
954.
955.     int xx4 = 0, qq4 = 0, contactpnt4 = 0, l4, i4;
956.
957.
958.     for (i4 = 0; i4 < isobox4; i4= i4+1)
959.
960.     {
961.
962.     xx4 = isocounterbox4[i4];
963.
964.
965.     rightisosceles [xx4][0][0] = rightisosceles [xx4][0][0] + resultux [
    xx4] ;
966.
967.     rightisosceles [xx4][0][1] = rightisosceles [xx4][0][1] + resultuy [
    xx4] ;
968.
969.
970.     for ( l4 = 0; l4 < isobox4; l4 = l4+1)
971.
972.     {
973.
974.
975.     qq4 = isocounterbox4[l4];
976.
977.
978.
```

```
    if ( ((pow((rightisosceles [xx4][0][0] -
     rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) > 0) && ((pow((rightisosceles [xx4][0][0
    ] - rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) <= 200))

    {

    contactpnt4 = contactpnt4 + 1;

    if (contactpnt4 >= 4) {goto next4;}


    if ( rightisosceles [qq4][3][1] >  rightisosceles [xx4][1][1] )

    {


    ydisplat [xx4][contactpnt4] = ydisplacement2  + (deluy [xx4][contact
   pnt4] -
     deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][0] - rightisosceles [xx4][0][0])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][0] -
     rightisosceles [qq4][0][0]));
    xdisplat [xx4][contactpnt4] = xdisplacement2 + (delux [xx4][contactp
   nt4] -
     delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][1] - rightisosceles [xx4][0][1])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][1] -
   rightisosceles [qq4][0][1]));


    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));


    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-
   1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coe
   fficient/1000);


    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-
   1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coeffic
   ient/1000);


```

```
    yforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * sin (alpha1)) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha1)));
    xforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * cos (alpha1)) + ((fn [xx4][contactpnt4] + dn [xx4][contactpn
t4]) * sin (alpha1)));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisoscel
es [qq4][3][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][3][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
    resultthetadot [xx4] += thetadotsum [xx4];


    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
t;
    deluysum [xx4] = deluy [xx4][contactpnt4];
    resultdeluy [xx4] += deluysum [xx4];

```

```
1050.
1051.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1052.    deluxsum [xx4] = delux [xx4][contactpnt4];
1053.    resultdelux [xx4] += deluxsum [xx4];
1054.
1055.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1056.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1057.    resultdeltheta [xx4] += delthetasum [xx4];
1058.
1059.
1060.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1061.    uysum [xx4] = uy [xx4][contactpnt4];
1062.    resultuy [xx4]+= uysum [xx4];
1063.
1064.
1065.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1066.    uxsum [xx4] = ux [xx4][contactpnt4];
1067.    resultux [xx4] += uxsum [xx4];
1068.
1069.
1070.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1071.    thetasum [xx4] = theta [xx4][contactpnt4];
1072.    resulttheta [xx4] += thetasum [xx4];
1073.    alpha [xx4] = alpha1 + resulttheta [xx4];
1074.
1075.    }
1076.
1077.    else
1078.
1079.    {
1080.
1081.
1082.    ydisplat [xx4][contactpnt4] += ydisplacement2  + (deluy [xx4][contac
   tpnt4] -
    deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
    [qq4][1][0] - rightisosceles [xx4][0][0])  -
    deltheta [qq4][contactpnt4] * (rightisosceles [qq4][0][0] -
    rightisosceles [qq4][0][0]));
1083.    xdisplat [xx4][contactpnt4] += xdisplacement2 + (delux [xx4][contact
   pnt4] -
    delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
    [qq4][1][1] - rightisosceles [xx4][0][1])  -
    deltheta [qq4][contactpnt4] * (rightisosceles [qq4][1][1] -
   rightisosceles [qq4][0][1]));
1084.
1085.    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
1086.    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));
1087.
```

```
1088.    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-
   1) *  (dstiffness_coefficient/1000);
1089.    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coe
   fficient/1000);
1090.
1091.    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-
   1) * (ddamping_coefficient/1000);
1092.    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coeffic
   ient/1000);
1093.
1094.    yforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][con
   tactpnt4]) * sin (alpha [xx4])) -
    ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha [xx4])));

1095.    xforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][con
   tactpnt4]) * cos (alpha [xx4])) + ((fn [xx4][contactpnt4] + dn [xx4][conta
   ctpnt4]) * sin (alpha [xx4])));
1096.
1097.    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
1098.    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);
1099.
1100.    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
1101.    x [xx4] = fxsum [xx4][contactpnt4];
1102.    resultfx [xx4] += x[xx4];
1103.
1104.
1105.    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
1106.    y [xx4] = fysum [xx4][contactpnt4];
1107.    resultfy [xx4] += y[xx4];
1108.
1109.
1110.    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisoscel
   es [qq4][1][0] - rightisosceles [xx4][0][0]) -
    xforce [xx4][contactpnt4] * ( rightisosceles [qq4][1][1] -
    rightisosceles [xx4][0][1]));
1111.    m [xx4] = msum [xx4][contactpnt4];
1112.    resultm [xx4] += m[xx4];
1113.
1114.
1115.    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1116.    udotysum [xx4] = udoty [xx4][contactpnt4];
1117.    resultudoty [xx4] += udotysum [xx4];
1118.
1119.
1120.    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1121.    udotxsum [xx4] = udotx [xx4][contactpnt4];
1122.    resultudotx [xx4] += udotxsum [xx4];
1123.
1124.
```

```
1125.    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
   ement)*1000/ ii); //1/s
1126.    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
1127.    resultthetadot [xx4] += thetadotsum [xx4];
1128.
1129.
1130.    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
   t;
1131.    deluysum [xx4] = deluy [xx4][contactpnt4];
1132.    resultdeluy [xx4] += deluysum [xx4];
1133.
1134.
1135.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1136.    deluxsum [xx4] = delux [xx4][contactpnt4];
1137.    resultdelux [xx4] += deluxsum [xx4];
1138.
1139.
1140.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1141.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1142.    resultdeltheta [xx4] += delthetasum [xx4];
1143.
1144.
1145.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1146.    uysum [xx4] = uy [xx4][contactpnt4];
1147.    resultuy [xx4]+= uysum [xx4];
1148.
1149.
1150.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1151.    uxsum [xx4] = ux [xx4][contactpnt4];
1152.    resultux [xx4] += uxsum [xx4];
1153.
1154.
1155.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1156.    thetasum [xx4] = theta [xx4][contactpnt4];
1157.    resulttheta [xx4] += thetasum [xx4];
1158.    alpha [xx4] = alpha1 + resulttheta [xx4];
1159.    }
1160.
1161.    }
1162.
1163.    }
1164.
1165.    next4:
1166.    ;}
1167.
1168.
1169.    int xx5 = 0, qq5 = 0, contactpnt5 = 0, l5, i5;
1170.
1171.
1172.    for (i5 = 0; i5 < isobox5; i5= i5+1)
```

```
1173.
1174.    {
1175.
1176.    xx5 = isocounterbox5[i5];
1177.
1178.    rightisosceles [xx5][0][0] = rightisosceles [xx5][0][0] + resultux [
   xx5] ;
1179.
1180.    rightisosceles [xx5][0][1] = rightisosceles [xx5][0][1] + resultuy [
   xx5] ;
1181.
1182.
1183.
1184.    for ( l5 = 0; l5 < isobox5; l5 = l5+1)
1185.
1186.    {
1187.
1188.
1189.    qq5 = isocounterbox5[l5];
1190.
1191.
1192.
1193.    if ( ((pow((rightisosceles [xx5][0][0] -
    rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) > 0) && ((pow((rightisosceles [xx5][0][0
   ] - rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) <= 200))
1194.
1195.    {
1196.
1197.    contactpnt5 = contactpnt5 + 1;
1198.
1199.    if (contactpnt5 >= 4) {goto next5;}
1200.
1201.
1202.    if ( rightisosceles [qq5][3][1] >  rightisosceles [xx5][1][1] )
1203.
1204.    {
1205.
1206.
1207.    ydisplat [xx5][contactpnt5] = ydisplacement2  + (deluy [xx5][contact
   pnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][0] -
    rightisosceles [qq5][0][0]));
1208.    xdisplat [xx5][contactpnt5] = xdisplacement2 + (delux [xx5][contactp
   nt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][1] -
   rightisosceles [qq5][0][1]));
```

```
1209.
1210.
1211.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1212.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1213.
1214.
1215.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1216.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1217.
1218.
1219.    dn [xx5][contactpnt5] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1220.    ds [xx5][contactpnt5] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1221.
1222.
1223.    yforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * sin (alpha1)) -
     ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha1)));
1224.    xforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * cos (alpha1)) + ((fn [xx5][contactpnt5] + dn [xx5][contactpn
   t5]) * sin (alpha1)));
1225.
1226.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1227.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1228.
1229.
1230.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1231.    y [xx5] = fysum [xx5][contactpnt5];
1232.    resultfy [xx5] += y[xx5];
1233.
1234.
1235.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1236.    x [xx5] = fxsum [xx5][contactpnt5];
1237.    resultfx [xx5] += x[xx5];
1238.
1239.
1240.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][3][0] - rightisosceles [xx5][0][0]) -
     xforce [xx5][contactpnt5] * ( rightisosceles [qq5][3][1] -
     rightisosceles [xx5][0][1]));
1241.    m [xx5] = msum [xx5][contactpnt5];
1242.    resultm [xx5] += m[xx5];
1243.
1244.
1245.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1246.    udotysum [xx5] = udoty [xx5][contactpnt5];
```

```
1247.    resultudoty [xx5] += udotysum [xx5];
1248.
1249.
1250.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1251.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1252.    resultudotx [xx5] += udotxsum [xx5];
1253.
1254.
1255.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1256.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1257.    resultthetadot [xx5] += thetadotsum [xx5];
1258.
1259.
1260.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1261.    deluysum [xx5] = deluy [xx5][contactpnt5];
1262.    resultdeluy [xx5] += deluysum [xx5];
1263.
1264.
1265.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1266.    deluxsum [xx5] = delux [xx5][contactpnt5];
1267.    resultdelux [xx5] += deluxsum [xx5];
1268.
1269.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1270.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1271.    resultdeltheta [xx5] += delthetasum [xx5];
1272.
1273.
1274.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1275.    uysum [xx5] = uy [xx5][contactpnt5];
1276.    resultuy [xx5]+= uysum [xx5];
1277.
1278.
1279.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1280.    uxsum [xx5] = ux [xx5][contactpnt5];
1281.    resultux [xx5] += uxsum [xx5];
1282.
1283.
1284.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1285.    thetasum [xx5] = theta [xx5][contactpnt5];
1286.    resulttheta [xx5] += thetasum [xx5];
1287.    alpha [xx5] = alpha1 + resulttheta [xx5];
1288.
1289.    }
1290.
1291.    else
1292.
1293.    {
```

```
1294.
1295.
1296.    ydisplat [xx5][contactpnt5] += ydisplacement2  + (deluy [xx5][contac
   tpnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][0][0] -
    rightisosceles [qq5][0][0]));
1297.    xdisplat [xx5][contactpnt5] += xdisplacement2 + (delux [xx5][contact
   pnt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][1][1] -
   rightisosceles [qq5][0][1]));
1298.
1299.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1300.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1301.
1302.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1303.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1304.
1305.    dn [xx5][contactpnt2] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1306.    ds [xx5][contactpnt2] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1307.
1308.    yforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * sin (alpha [xx5])) -
    ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha [xx5])));

1309.    xforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * cos (alpha [xx5])) + ((fn [xx5][contactpnt5] + dn [xx5][conta
   ctpnt5]) * sin (alpha [xx5])));
1310.
1311.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1312.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1313.
1314.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1315.    x [xx5] = fxsum [xx5][contactpnt5];
1316.    resultfx [xx5] += x[xx5];
1317.
1318.
1319.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1320.    y [xx5] = fysum [xx5][contactpnt5];
1321.    resultfy [xx5] += y[xx5];
1322.
1323.
```

```
1324.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][1][0] - rightisosceles [xx5][0][0]) -
    xforce [xx5][contactpnt5] * ( rightisosceles [qq5][1][1] -
    rightisosceles [xx5][0][1]));
1325.    m [xx5] = msum [xx5][contactpnt5];
1326.    resultm [xx5] += m[xx5];
1327.
1328.
1329.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1330.    udotysum [xx5] = udoty [xx5][contactpnt5];
1331.    resultudoty [xx5] += udotysum [xx5];
1332.
1333.
1334.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1335.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1336.    resultudotx [xx5] += udotxsum [xx5];
1337.
1338.
1339.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1340.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1341.    resultthetadot [xx5] += thetadotsum [xx5];
1342.
1343.
1344.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1345.    deluysum [xx5] = deluy [xx5][contactpnt5];
1346.    resultdeluy [xx5] += deluysum [xx5];
1347.
1348.
1349.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1350.    deluxsum [xx5] = delux [xx5][contactpnt5];
1351.    resultdelux [xx5] += deluxsum [xx5];
1352.
1353.
1354.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1355.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1356.    resultdeltheta [xx5] += delthetasum [xx5];
1357.
1358.
1359.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1360.    uysum [xx5] = uy [xx5][contactpnt5];
1361.    resultuy [xx5]+= uysum [xx5];
1362.
1363.
1364.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1365.    uxsum [xx5] = ux [xx5][contactpnt5];
1366.    resultux [xx5] += uxsum [xx5];
```

```
1367.
1368.
1369.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1370.    thetasum [xx5] = theta [xx5][contactpnt5];
1371.    resulttheta [xx5] += thetasum [xx5];
1372.    alpha [xx5] = alpha1 + resulttheta [xx5];
1373.
1374.    }
1375.    }
1376.
1377.    }
1378.
1379.    next5:
1380.    ;}
1381.
1382.
1383.    int xx6 = 0, qq6 = 0, contactpnt6 = 0, l6, i6;
1384.
1385.
1386.    for (i6 = 0; i6 < isobox6; i6= i6+1)
1387.
1388.    {
1389.
1390.    xx6 = isocounterbox6[i6];
1391.
1392.    rightisosceles [xx6][0][0] = rightisosceles [xx6][0][0] + resultux [
   xx6] ;
1393.
1394.    rightisosceles [xx6][0][1] = rightisosceles [xx6][0][1] + resultuy [
   xx6] ;
1395.
1396.
1397.
1398.    for ( l6 = 0; l6 < isobox6; l6 = l6+1)
1399.
1400.    {
1401.
1402.
1403.    qq6 = isocounterbox6[l6];
1404.
1405.
1406.
1407.    if ( ((pow((rightisosceles [xx6][0][0] -
    rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) > 0) && ((pow((rightisosceles [xx6][0][0
   ] - rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) <= 200))
1408.
1409.    {
1410.
1411.    contactpnt6 = contactpnt6 + 1;
1412.
```

```
    if (contactpnt6 >= 4) {goto next6;}


    if ( rightisosceles [qq6][3][1] >  rightisosceles [xx6][1][1] )

    {


    ydisplat [xx6][contactpnt6] = ydisplacement2  + (deluy [xx6][contact
pnt6] -
     deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
     [qq6][3][0] - rightisosceles [xx6][0][0])  -
     deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][0] -
     rightisosceles [qq6][0][0]));
    xdisplat [xx6][contactpnt6] = xdisplacement2 + (delux [xx6][contactp
nt6] -
     delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
     [qq6][3][1] - rightisosceles [xx6][0][1])  -
     deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][1] -
    rightisosceles [qq6][0][1]));


    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
1) - xdisplat [xx6][contactpnt6] * sin (alpha1));


    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
1) *  (dstiffness_coefficient/1000);
    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
fficient/1000);


    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
1) * (ddamping_coefficient/1000);
    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
ient/1000);


    yforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
ntactpnt6]) * sin (alpha1)) -
     ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha1)));
    xforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
ntactpnt6]) * cos (alpha1)) + ((fn [xx6][contactpnt6] + dn [xx6][contactpn
t6]) * sin (alpha1)));

    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);


    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
```

```
    y [xx6] = fysum [xx6][contactpnt6];
    resultfy [xx6] += y[xx6];


    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
    x [xx6] = fxsum [xx6][contactpnt6];
    resultfx [xx6] += x[xx6];


    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][3][0] - rightisosceles [xx6][0][0]) -
     xforce [xx6][contactpnt6] * ( rightisosceles [qq6][3][1] -
     rightisosceles [xx6][0][1]));
    m [xx6] = msum [xx6][contactpnt6];
    resultm [xx6] += m[xx6];


    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent) *1000/ dmass); // mm/sec
    udotysum [xx6] = udoty [xx6][contactpnt6];
    resultudoty [xx6] += udotysum [xx6];


    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
    udotxsum [xx6] = udotx [xx6][contactpnt6];
    resultudotx [xx6] += udotxsum [xx6];


    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
    resultthetadot [xx6] += thetadotsum [xx6];


    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
    deluysum [xx6] = deluy [xx6][contactpnt6];
    resultdeluy [xx6] += deluysum [xx6];


    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
    deluxsum [xx6] = delux [xx6][contactpnt6];
    resultdelux [xx6] += deluxsum [xx6];

    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
    delthetasum [xx6] = deltheta [xx6][contactpnt6];
    resultdeltheta [xx6] += delthetasum [xx6];


```

```
1488.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1489.    uysum [xx6] = uy [xx6][contactpnt6];
1490.    resultuy [xx6]+= uysum [xx6];
1491.
1492.
1493.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1494.    uxsum [xx6] = ux [xx6][contactpnt6];
1495.    resultux [xx6] += uxsum [xx6];
1496.
1497.
1498.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1499.    thetasum [xx6] = theta [xx6][contactpnt6];
1500.    resulttheta [xx6] += thetasum [xx6];
1501.    alpha [xx6] = alpha1 + resulttheta [xx6];
1502.
1503.    }
1504.
1505.    else
1506.
1507.    {
1508.
1509.
1510.
1511.    ydisplat [xx6][contactpnt6] += ydisplacement2  + (deluy [xx6][contac
   tpnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][0][0] -
    rightisosceles [qq6][0][0]));
1512.    xdisplat [xx6][contactpnt6] += xdisplacement2 + (delux [xx6][contact
   pnt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][1][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][1][1] -
   rightisosceles [qq6][0][1]));
1513.
1514.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1515.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1516.
1517.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1518.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1519.
1520.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1521.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1522.
```

```
1523.    yforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * sin (alpha [xx6])) -
    ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha [xx6])));

1524.    xforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * cos (alpha [xx6])) + ((fn [xx6][contactpnt6] + dn [xx6][conta
   ctpnt6]) * sin (alpha [xx6])));
1525.
1526.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1527.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1528.
1529.    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
1530.    x [xx6] = fxsum [xx6][contactpnt6];
1531.    resultfx [xx6] += x[xx6];
1532.
1533.
1534.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
1535.    y [xx6] = fysum [xx6][contactpnt6];
1536.    resultfy [xx6] += y[xx6];
1537.
1538.
1539.    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][1][0] - rightisosceles [xx6][0][0]) -
    xforce [xx6][contactpnt6] * ( rightisosceles [qq6][1][1] -
    rightisosceles [xx6][0][1]));
1540.    m [xx6] = msum [xx6][contactpnt6];
1541.    resultm [xx6] += m[xx6];
1542.
1543.
1544.    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1545.    udotysum [xx6] = udoty [xx6][contactpnt6];
1546.    resultudoty [xx6] += udotysum [xx6];
1547.
1548.
1549.    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1550.    udotxsum [xx6] = udotx [xx6][contactpnt6];
1551.    resultudotx [xx6] += udotxsum [xx6];
1552.
1553.
1554.    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
1555.    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
1556.    resultthetadot [xx6] += thetadotsum [xx6];
1557.
1558.
1559.    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
1560.    deluysum [xx6] = deluy [xx6][contactpnt6];
1561.    resultdeluy [xx6] += deluysum [xx6];
1562.
```

```
1563.
1564.    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
1565.    deluxsum [xx6] = delux [xx6][contactpnt6];
1566.    resultdelux [xx6] += deluxsum [xx6];
1567.
1568.
1569.    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
1570.    delthetasum [xx6] = deltheta [xx6][contactpnt6];
1571.    resultdeltheta [xx6] += delthetasum [xx6];
1572.
1573.
1574.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1575.    uysum [xx6] = uy [xx6][contactpnt6];
1576.    resultuy [xx6]+= uysum [xx6];
1577.
1578.
1579.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1580.    uxsum [xx6] = ux [xx6][contactpnt6];
1581.    resultux [xx6] += uxsum [xx6];
1582.
1583.
1584.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1585.    thetasum [xx6] = theta [xx6][contactpnt6];
1586.    resulttheta [xx6] += thetasum [xx6];
1587.    alpha [xx6] = alpha1 + resulttheta [xx6];
1588.
1589.    }
1590.
1591.    }
1592.
1593.    }
1594.
1595.    next6:
1596.    ;}
1597.
1598.
1599.
1600.    int xx7 = 0, qq7 = 0, contactpnt7 = 0, l7, i7;
1601.
1602.
1603.    for (i7 = 0; i7 < isobox7; i7= i7+1)
1604.
1605.    {
1606.
1607.    xx7 = isocounterbox7[i7];
1608.
1609.    rightisosceles [xx7][0][0] = rightisosceles [xx7][0][0] + resultux [
   xx7] ;
1610.
```

```
    rightisosceles [xx7][0][1] = rightisosceles [xx7][0][1] + resultuy [
   xx7] ;
    for ( l7 = 0; l7 < isobox7; l7 = l7+1)
    {
    qq7 = isocounterbox7[l7];
    if ( ((pow((rightisosceles [xx7][0][0] -
    rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) > 0) && ((pow((rightisosceles [xx7][0][0
   ] - rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) <= 200))
    {
    contactpnt7 = contactpnt7 + 1;
    if (contactpnt7 >= 4) {goto next7;}
    if ( rightisosceles [qq7][3][1] >  rightisosceles [xx7][1][1] )
    {
    ydisplat [xx7][contactpnt7] = ydisplacement2  + (deluy [xx7][contact
   pnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][0] - rightisosceles [xx7][0][0])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][0] -
    rightisosceles [qq7][0][0]));
    xdisplat [xx7][contactpnt7] = xdisplacement2 + (delux [xx7][contactp
   nt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][1] -
   rightisosceles [qq7][0][1]));
    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));
```

```
1646.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1647.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1648.
1649.
1650.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1651.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1652.
1653.
1654.    yforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * sin (alpha1)) -
     ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha1)));
1655.    xforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * cos (alpha1)) + ((fn [xx7][contactpnt7] + dn [xx7][contactpn
   t7]) * sin (alpha1)));
1656.
1657.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1658.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1659.
1660.
1661.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1662.    y [xx7] = fysum [xx7][contactpnt7];
1663.    resultfy [xx7] += y[xx7];
1664.
1665.
1666.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1667.    x [xx7] = fxsum [xx7][contactpnt7];
1668.    resultfx [xx7] += x[xx7];
1669.
1670.
1671.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][3][0] - rightisosceles [xx7][0][0]) -
     xforce [xx7][contactpnt7] * ( rightisosceles [qq7][3][1] -
     rightisosceles [xx7][0][1]));
1672.    m [xx7] = msum [xx7][contactpnt7];
1673.    resultm [xx7] += m[xx7];
1674.
1675.
1676.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1677.    udotysum [xx7] = udoty [xx7][contactpnt7];
1678.    resultudoty [xx7] += udotysum [xx7];
1679.
1680.
1681.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1682.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1683.    resultudotx [xx7] += udotxsum [xx7];
1684.
```

```
1685.
1686.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1687.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1688.    resultthetadot [xx7] += thetadotsum [xx7];
1689.
1690.
1691.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1692.    deluysum [xx7] = deluy [xx7][contactpnt7];
1693.    resultdeluy [xx7] += deluysum [xx7];
1694.
1695.
1696.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1697.    deluxsum [xx7] = delux [xx7][contactpnt7];
1698.    resultdelux [xx7] += deluxsum [xx7];
1699.
1700.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1701.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1702.    resultdeltheta [xx7] += delthetasum [xx7];
1703.
1704.
1705.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1706.    uysum [xx7] = uy [xx7][contactpnt7];
1707.    resultuy [xx7]+= uysum [xx7];
1708.
1709.
1710.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1711.    uxsum [xx7] = ux [xx7][contactpnt7];
1712.    resultux [xx7] += uxsum [xx7];
1713.
1714.
1715.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1716.    thetasum [xx7] = theta [xx7][contactpnt7];
1717.    resultttheta [xx7] += thetasum [xx7];
1718.    alpha [xx7] = alpha1 + resultttheta [xx7];
1719.
1720.    }
1721.
1722.    else
1723.
1724.    {
1725.
1726.
1727.
1728.    ydisplat [xx7][contactpnt7] += ydisplacement2  + (deluy [xx7][contac
   tpnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][0] - rightisosceles [xx7][0][0])  -
```

```
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][0][0] -
    rightisosceles [qq7][0][0]));
1729.    xdisplat [xx7][contactpnt7] += xdisplacement2 + (delux [xx7][contact
   pnt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][1][1] -
   rightisosceles [qq7][0][1]));
1730.
1731.    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
1732.    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));
1733.
1734.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1735.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1736.
1737.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1738.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1739.
1740.    yforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * sin (alpha [xx7])) -
    ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha [xx7])));

1741.    xforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * cos (alpha [xx7])) + ((fn [xx7][contactpnt7] + dn [xx7][conta
   ctpnt7]) * sin (alpha [xx7])));
1742.
1743.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1744.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1745.
1746.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1747.    x [xx7] = fxsum [xx7][contactpnt7];
1748.    resultfx [xx7] += x[xx7];
1749.
1750.
1751.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1752.    y [xx7] = fysum [xx7][contactpnt7];
1753.    resultfy [xx7] += y[xx7];
1754.
1755.
1756.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][1][0] - rightisosceles [xx7][0][0]) -
    xforce [xx7][contactpnt7] * ( rightisosceles [qq7][1][1] -
    rightisosceles [xx7][0][1]));
1757.    m [xx7] = msum [xx7][contactpnt7];
1758.    resultm [xx7] += m[xx7];
1759.
```

```
1760.
1761.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1762.    udotysum [xx7] = udoty [xx7][contactpnt7];
1763.    resultudoty [xx7] += udotysum [xx7];
1764.
1765.
1766.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1767.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1768.    resultudotx [xx7] += udotxsum [xx7];
1769.
1770.
1771.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1772.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1773.    resultthetadot [xx7] += thetadotsum [xx7];
1774.
1775.
1776.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1777.    deluysum [xx7] = deluy [xx7][contactpnt7];
1778.    resultdeluy [xx7] += deluysum [xx7];
1779.
1780.
1781.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1782.    deluxsum [xx7] = delux [xx7][contactpnt7];
1783.    resultdelux [xx7] += deluxsum [xx7];
1784.
1785.
1786.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1787.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1788.    resultdeltheta [xx7] += delthetasum [xx7];
1789.
1790.
1791.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1792.    uysum [xx7] = uy [xx7][contactpnt7];
1793.    resultuy [xx7]+= uysum [xx7];
1794.
1795.
1796.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1797.    uxsum [xx7] = ux [xx7][contactpnt7];
1798.    resultux [xx7] += uxsum [xx7];
1799.
1800.
1801.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1802.    thetasum [xx7] = theta [xx7][contactpnt7];
1803.    resulttheta [xx7] += thetasum [xx7];
1804.    alpha [xx7] = alpha1 + resulttheta [xx7];
1805.    }
```

```
1806.
1807.    }
1808.
1809.    }
1810.
1811.    next7:
1812.    ;}
1813.
1814.
1815.
1816.    int xx8 = 0, qq8 = 0, contactpnt8 = 0, l8, i8;
1817.
1818.
1819.    for (i8 = 0; i8 < isobox8; i8= i8+1)
1820.
1821.    {
1822.
1823.    xx8 = isocounterbox8[i8];
1824.
1825.    rightisosceles [xx8][0][0] = rightisosceles [xx8][0][0] + resultux [
   xx8] ;
1826.
1827.    rightisosceles [xx8][0][1] = rightisosceles [xx8][0][1] + resultuy [
   xx8] ;
1828.
1829.
1830.
1831.    for ( l8 = 0; l8 < isobox8; l8 = l8+1)
1832.
1833.    {
1834.
1835.
1836.    qq8 = isocounterbox8[l8];
1837.
1838.
1839.
1840.    if ( ((pow((rightisosceles [xx8][0][0] -
    rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) > 0) && ((pow((rightisosceles [xx8][0][0
   ] - rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) <= 200))
1841.
1842.    {
1843.
1844.    contactpnt8 = contactpnt8 + 1;
1845.
1846.    if (contactpnt8 >= 4) {goto next8;}
1847.
1848.
1849.    if ( rightisosceles [qq8][3][1] >  rightisosceles [xx8][1][1] )
1850.
1851.    {
```

```
1852.
1853.
1854.    ydisplat [xx8][contactpnt8] = ydisplacement2  + (deluy [xx8][contact
   pnt8] -
    deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][0] - rightisosceles [xx8][0][0])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][0] -
    rightisosceles [qq8][0][0]));
1855.    xdisplat [xx8][contactpnt8] = xdisplacement2 + (delux [xx8][contactp
   nt8] -
    delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][1] - rightisosceles [xx8][0][1])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][1] -
   rightisosceles [qq8][0][1]));
1856.
1857.
1858.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1859.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1860.
1861.
1862.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt8] * (-
   1) *  (dstiffness_coefficient/1000);
1863.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt8] *  (dstiffness_coe
   fficient/1000);
1864.
1865.
1866.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1867.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1868.
1869.
1870.    yforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * sin (alpha1)) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha1)));
1871.    xforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * cos (alpha1)) + ((fn [xx8][contactpnt8] + dn [xx8][contactpn
   t8]) * sin (alpha1)));
1872.
1873.    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
1874.    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);
1875.
1876.
1877.    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
1878.    y [xx8] = fysum [xx8][contactpnt8];
1879.    resultfy [xx8] += y[xx8];
1880.
1881.
1882.    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
1883.    x [xx8] = fxsum [xx8][contactpnt8];
```

```
1884.    resultfx [xx8] += x[xx8];
1885.
1886.
1887.    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
   es [qq8][3][0] - rightisosceles [xx8][0][0]) -
    xforce [xx8][contactpnt8] * ( rightisosceles [qq8][3][1] -
    rightisosceles [xx8][0][1]));
1888.    m [xx8] = msum [xx8][contactpnt8];
1889.    resultm [xx8] += m[xx8];
1890.
1891.
1892.    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1893.    udotysum [xx8] = udoty [xx8][contactpnt8];
1894.    resultudoty [xx8] += udotysum [xx8];
1895.
1896.
1897.    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1898.    udotxsum [xx8] = udotx [xx8][contactpnt8];
1899.    resultudotx [xx8] += udotxsum [xx8];
1900.
1901.
1902.    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
   ement)*1000/ ii); //1/s
1903.    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
1904.    resultthetadot [xx8] += thetadotsum [xx8];
1905.
1906.
1907.    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
   t;
1908.    deluysum [xx8] = deluy [xx8][contactpnt8];
1909.    resultdeluy [xx8] += deluysum [xx8];
1910.
1911.
1912.    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
   t;
1913.    deluxsum [xx8] = delux [xx8][contactpnt8];
1914.    resultdelux [xx8] += deluxsum [xx8];
1915.
1916.    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
   crement;
1917.    delthetasum [xx8] = deltheta [xx8][contactpnt8];
1918.    resultdeltheta [xx8] += delthetasum [xx8];
1919.
1920.
1921.    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
1922.    uysum [xx8] = uy [xx8][contactpnt8];
1923.    resultuy [xx8]+= uysum [xx8];
1924.
1925.
1926.    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
```

```
1927.    uxsum [xx8] = ux [xx8][contactpnt8];
1928.    resultux [xx8] += uxsum [xx8];
1929.
1930.
1931.    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
1932.    thetasum [xx8] = theta [xx8][contactpnt8];
1933.    resulttheta [xx8] += thetasum [xx8];
1934.    alpha [xx8] = alpha1 + resulttheta [xx8];
1935.
1936.    }
1937.
1938.    else
1939.
1940.    {
1941.
1942.
1943.    ydisplat [xx8][contactpnt8] += ydisplacement2  + (deluy [xx8][contac
   tpnt8] -
     deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][0] - rightisosceles [xx8][0][0])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][0][0] -
     rightisosceles [qq8][0][0]));
1944.    xdisplat [xx8][contactpnt8] += xdisplacement2 + (delux [xx8][contact
   pnt8] -
     delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][1] - rightisosceles [xx8][0][1])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][1][1] -
   rightisosceles [qq8][0][1]));
1945.
1946.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1947.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1948.
1949.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1950.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1951.
1952.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1953.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1954.
1955.    yforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * sin (alpha [xx8])) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha [xx8])));

1956.    xforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * cos (alpha [xx8])) + ((fn [xx8][contactpnt8] + dn [xx8][conta
   ctpnt8]) * sin (alpha [xx8])));
1957.
```

```
    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);

    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
    x [xx8] = fxsum [xx8][contactpnt8];
    resultfx [xx8] += x[xx8];


    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
    y [xx8] = fysum [xx8][contactpnt8];
    resultfy [xx8] += y[xx8];


    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
es [qq8][1][0] - rightisosceles [xx8][0][0]) -
     xforce [xx8][contactpnt8] * ( rightisosceles [qq8][1][1] -
     rightisosceles [xx8][0][1]));
    m [xx8] = msum [xx8][contactpnt8];
    resultm [xx8] += m[xx8];


    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
ent)* 1000/ dmass); // mm/sec
    udotysum [xx8] = udoty [xx8][contactpnt8];
    resultudoty [xx8] += udotysum [xx8];


    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx8] = udotx [xx8][contactpnt8];
    resultudotx [xx8] += udotxsum [xx8];


    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
    resultthetadot [xx8] += thetadotsum [xx8];


    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
t;
    deluysum [xx8] = deluy [xx8][contactpnt8];
    resultdeluy [xx8] += deluysum [xx8];


    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
t;
    deluxsum [xx8] = delux [xx8][contactpnt8];
    resultdelux [xx8] += deluxsum [xx8];


```

```
    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
crement;
    delthetasum [xx8] = deltheta [xx8][contactpnt8];
    resultdeltheta [xx8] += delthetasum [xx8];


    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
    uysum [xx8] = uy [xx8][contactpnt8];
    resultuy [xx8]+= uysum [xx8];


    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
    uxsum [xx8] = ux [xx8][contactpnt8];
    resultux [xx8] += uxsum [xx8];


    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
    thetasum [xx8] = theta [xx8][contactpnt8];
    resulttheta [xx8] += thetasum [xx8];
    alpha [xx8] = alpha1 + resulttheta [xx8];



    }
    }

    }

    next8:
    ;}


    int xx9 = 0, qq9 = 0, contactpnt9 = 0, l9, i9;


    for (i9 = 0; i9 < isobox9; i9= i9+1)

    {

    xx9 = isocounterbox9[i9];

    rightisosceles [xx9][0][0] = rightisosceles [xx9][0][0] + resultux [
xx9] ;

    rightisosceles [xx9][0][1] = rightisosceles [xx9][0][1] + resultuy [
xx9] ;


    for ( l9 = 0; l9 < isobox9; l9 = l9+1)

    {
```

```
2050.
2051.
2052.    qq9 = isocounterbox9[19];
2053.
2054.
2055.
2056.    if ( ((pow((rightisosceles [xx9][0][0] -
    rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) > 0) && ((pow((rightisosceles [xx9][0][0
   ] - rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) <= 200))
2057.
2058.    {
2059.
2060.    contactpnt9 = contactpnt9 + 1;
2061.
2062.    if (contactpnt9 >= 4) {goto next9;}
2063.
2064.
2065.    if ( rightisosceles [qq9][3][1] >  rightisosceles [xx9][1][1] )
2066.
2067.    {
2068.
2069.
2070.    ydisplat [xx9][contactpnt9] = ydisplacement2  + (deluy [xx9][contact
   pnt9] -
    deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][0] - rightisosceles [xx9][0][0])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][0] -
    rightisosceles [qq9][0][0]));
2071.    xdisplat [xx9][contactpnt9] = xdisplacement2 + (delux [xx9][contactp
   nt9] -
    delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][1] - rightisosceles [xx9][0][1])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][1] -
   rightisosceles [qq9][0][1]));
2072.
2073.
2074.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2075.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2076.
2077.
2078.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2079.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2080.
2081.
2082.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
```

```
    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
ient/1000);


    yforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * sin (alpha1)) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha1)));
    xforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * cos (alpha1)) + ((fn [xx9][contactpnt9] + dn [xx9][contactpn
t9]) * sin (alpha1)));

    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);


    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
    y [xx9] = fysum [xx9][contactpnt9];
    resultfy [xx9] += y[xx9];


    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
    x [xx9] = fxsum [xx9][contactpnt9];
    resultfx [xx9] += x[xx9];


    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
es [qq9][3][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][3][1] -
     rightisosceles [xx9][0][1]));
    m [xx9] = msum [xx9][contactpnt9];
    resultm [xx9] += m[xx9];


    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx9] = udoty [xx9][contactpnt9];
    resultudoty [xx9] += udotysum [xx9];


    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


```

```
2123.    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
   t;
2124.    deluysum [xx9] = deluy [xx9][contactpnt9];
2125.    resultdeluy [xx9] += deluysum [xx9];
2126.
2127.
2128.    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
   t;
2129.    deluxsum [xx9] = delux [xx9][contactpnt9];
2130.    resultdelux [xx9] += deluxsum [xx9];
2131.
2132.    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
   crement;
2133.    delthetasum [xx9] = deltheta [xx9][contactpnt9];
2134.    resultdeltheta [xx9] += delthetasum [xx9];
2135.
2136.
2137.    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
2138.    uysum [xx9] = uy [xx9][contactpnt9];
2139.    resultuy [xx9]+= uysum [xx9];
2140.
2141.
2142.    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
2143.    uxsum [xx9] = ux [xx9][contactpnt9];
2144.    resultux [xx9] += uxsum [xx9];
2145.
2146.
2147.    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
2148.    thetasum [xx9] = theta [xx9][contactpnt9];
2149.    resulttheta [xx9] += thetasum [xx9];
2150.    alpha [xx9] = alpha1 + resulttheta [xx9];
2151.
2152.    }
2153.
2154.    else
2155.
2156.    {
2157.
2158.
2159.    ydisplat [xx9][contactpnt9] += ydisplacement2  + (deluy [xx9][contac
   tpnt9] -
     deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][0] - rightisosceles [xx9][0][0])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][0][0] -
     rightisosceles [qq9][0][0]));
2160.    xdisplat [xx9][contactpnt9] += xdisplacement2 + (delux [xx9][contact
   pnt9] -
     delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][1] - rightisosceles [xx9][0][1])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][1][1] -
   rightisosceles [qq9][0][1]));
2161.
```

```
2162.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2163.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2164.
2165.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2166.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2167.
2168.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
2169.    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
   ient/1000);
2170.
2171.    yforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * sin (alpha [xx9])) -
    ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha [xx9])));

2172.    xforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * cos (alpha [xx9])) + ((fn [xx9][contactpnt9] + dn [xx9][conta
   ctpnt9]) * sin (alpha [xx9])));
2173.
2174.    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
2175.    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);
2176.
2177.    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
2178.    x [xx9] = fxsum [xx9][contactpnt9];
2179.    resultfx [xx9] += x[xx9];
2180.
2181.
2182.    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
2183.    y [xx9] = fysum [xx9][contactpnt9];
2184.    resultfy [xx9] += y[xx9];
2185.
2186.
2187.    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
   es [qq9][1][0] - rightisosceles [xx9][0][0]) -
    xforce [xx9][contactpnt9] * ( rightisosceles [qq9][1][1] -
    rightisosceles [xx9][0][1]));
2188.    m [xx9] = msum [xx9][contactpnt9];
2189.    resultm [xx9] += m[xx9];
2190.
2191.
2192.    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
2193.    udotysum [xx9] = udoty [xx9][contactpnt9];
2194.    resultudoty [xx9] += udotysum [xx9];
2195.
2196.
2197.    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
   ent)*1000/ dmass); //mm/sec
```

```
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
t;
    deluysum [xx9] = deluy [xx9][contactpnt9];
    resultdeluy [xx9] += deluysum [xx9];


    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
t;
    deluxsum [xx9] = delux [xx9][contactpnt9];
    resultdelux [xx9] += deluxsum [xx9];


    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
crement;
    delthetasum [xx9] = deltheta [xx9][contactpnt9];
    resultdeltheta [xx9] += delthetasum [xx9];


    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
    uysum [xx9] = uy [xx9][contactpnt9];
    resultuy [xx9]+= uysum [xx9];


    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
    uxsum [xx9] = ux [xx9][contactpnt9];
    resultux [xx9] += uxsum [xx9];


    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
    thetasum [xx9] = theta [xx9][contactpnt9];
    resulttheta [xx9] += thetasum [xx9];
    alpha [xx9] = alpha1 + resulttheta [xx9];

    }
    }

    }

    next9:
    ;}

```

```
2246.
2247.    int xx10 = 0, qq10 = 0, contactpnt10 = 0, l10, i10;
2248.
2249.
2250.    for (i10 = 0; i10 < isobox10; i10= i10+1)
2251.
2252.    {
2253.
2254.    xx10 = isocounterbox10[i10];
2255.
2256.    rightisosceles [xx10][0][0] = rightisosceles [xx10][0][0] + resultux
    [xx10] ;
2257.
2258.    rightisosceles [xx10][0][1] = rightisosceles [xx10][0][1] + resultuy
    [xx10] ;
2259.
2260.
2261.
2262.    for ( l10 = 0; l10 < isobox10; l10 = l10+1)
2263.
2264.    {
2265.
2266.
2267.    qq10 = isocounterbox10[l10];
2268.
2269.
2270.
2271.    if ( ((pow((rightisosceles [xx10][0][0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) > 0) && ((pow((rightisosceles [xx10][0]
   [0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) <= 200))
2272.
2273.    {
2274.
2275.    contactpnt10 = contactpnt10 + 1;
2276.
2277.    if (contactpnt10 >= 4) {goto next10;}
2278.
2279.
2280.    if ( rightisosceles [qq10][3][1] >  rightisosceles [xx10][1][1] )
2281.
2282.    {
2283.
2284.
2285.    ydisplat [xx10][contactpnt10] = ydisplacement2  + (deluy [xx10][cont
   actpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][0] -
    rightisosceles [qq10][0][0]));
```

```
2286.    xdisplat [xx10][contactpnt10] = xdisplacement2 + (delux [xx10][conta
   ctpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][1] -
   rightisosceles [qq10][0][1]));
2287.
2288.
2289.    delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2290.    delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2291.
2292.
2293.    fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2294.    fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2295.
2296.
2297.    dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2298.    ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2299.
2300.
2301.    yforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * sin (alpha1)) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha1)));
2302.    xforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * cos (alpha1)) + ((fn [xx10][contactpnt10] + dn [xx10][
   contactpnt10]) * sin (alpha1)));
2303.
2304.    yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2305.    xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2306.
2307.
2308.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2309.    y [xx10] = fysum [xx10][contactpnt10];
2310.    resultfy [xx10] += y[xx10];
2311.
2312.
2313.    fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2314.    x [xx10] = fxsum [xx10][contactpnt10];
2315.    resultfx [xx10] += x[xx10];
2316.
2317.
2318.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][3][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][3][1] -
    rightisosceles [xx10][0][1]));
2319.    m [xx10] = msum [xx10][contactpnt10];
```

```
2320.    resultm [xx10] += m[xx10];
2321.
2322.
2323.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement) *1000/ dmass); // mm/sec
2324.    udotysum [xx10] = udoty [xx10][contactpnt10];
2325.    resultudoty [xx10] += udotysum [xx10];
2326.
2327.
2328.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2329.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2330.    resultudotx [xx10] += udotxsum [xx10];
2331.
2332.
2333.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2334.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2335.    resultthetadot [xx10] += thetadotsum [xx10];
2336.
2337.
2338.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2339.    deluysum [xx10] = deluy [xx10][contactpnt10];
2340.    resultdeluy [xx10] += deluysum [xx10];
2341.
2342.
2343.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2344.    deluxsum [xx10] = delux [xx10][contactpnt10];
2345.    resultdelux [xx10] += deluxsum [xx10];
2346.
2347.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2348.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2349.    resultdeltheta [xx10] += delthetasum [xx10];
2350.
2351.
2352.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2353.    uysum [xx10] = uy [xx10][contactpnt10];
2354.    resultuy [xx10]+= uysum [xx10];
2355.
2356.
2357.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2358.    uxsum [xx10] = ux [xx10][contactpnt10];
2359.    resultux [xx10] += uxsum [xx10];
2360.
2361.
2362.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2363.    thetasum [xx10] = theta [xx10][contactpnt10];
2364.    resulttheta [xx10] += thetasum [xx10];
2365.    alpha [xx10] = alpha1 + resulttheta [xx10];
```

```
2366.
2367.     }
2368.
2369.     else
2370.
2371.     {
2372.
2373.
2374.     ydisplat [xx10][contactpnt10] += ydisplacement2  + (deluy [xx10][con
   tactpnt10] -
     deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][0] - rightisosceles [xx10][0][0])  -
     deltheta [qq10][contactpnt10] * (rightisosceles [qq10][0][0] -
     rightisosceles [qq10][0][0]));
2375.     xdisplat [xx10][contactpnt10] += xdisplacement2 + (delux [xx10][cont
   actpnt10] -
     delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][1] - rightisosceles [xx10][0][1])  -
     deltheta [qq10][contactpnt10] * (rightisosceles [qq10][1][1] -
   rightisosceles [qq10][0][1]));
2376.
2377.     delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2378.     delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2379.
2380.     fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2381.     fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2382.
2383.     dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2384.     ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2385.
2386.     yforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * sin (alpha [xx10])) -
     ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha [xx10])
   ));
2387.     xforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * cos (alpha [xx10])) + ((fn [xx10][contactpnt10] + dn [x
   x10][contactpnt10]) * sin (alpha [xx10])));
2388.
2389.     yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2390.     xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2391.
2392.     fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2393.     x [xx10] = fxsum [xx10][contactpnt10];
2394.     resultfx [xx10] += x[xx10];
2395.
2396.
```

```
2397.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2398.    y [xx10] = fysum [xx10][contactpnt10];
2399.    resultfy [xx10] += y[xx10];
2400.
2401.
2402.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][1][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][1][1] -
    rightisosceles [xx10][0][1]));
2403.    m [xx10] = msum [xx10][contactpnt10];
2404.    resultm [xx10] += m[xx10];
2405.
2406.
2407.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2408.    udotysum [xx10] = udoty [xx10][contactpnt10];
2409.    resultudoty [xx10] += udotysum [xx10];
2410.
2411.
2412.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2413.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2414.    resultudotx [xx10] += udotxsum [xx10];
2415.
2416.
2417.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2418.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2419.    resultthetadot [xx10] += thetadotsum [xx10];
2420.
2421.
2422.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2423.    deluysum [xx10] = deluy [xx10][contactpnt10];
2424.    resultdeluy [xx10] += deluysum [xx10];
2425.
2426.
2427.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2428.    deluxsum [xx10] = delux [xx10][contactpnt10];
2429.    resultdelux [xx10] += deluxsum [xx10];
2430.
2431.
2432.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2433.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2434.    resultdeltheta [xx10] += delthetasum [xx10];
2435.
2436.
2437.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2438.    uysum [xx10] = uy [xx10][contactpnt10];
2439.    resultuy [xx10]+= uysum [xx10];
```

```
2440.
2441.
2442.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2443.    uxsum [xx10] = ux [xx10][contactpnt10];
2444.    resultux [xx10] += uxsum [xx10];
2445.
2446.
2447.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2448.    thetasum [xx10] = theta [xx10][contactpnt10];
2449.    resulttheta [xx10] += thetasum [xx10];
2450.    alpha [xx10] = alpha1 + resulttheta [xx10];
2451.
2452.    }
2453.    }
2454.
2455.    }
2456.
2457.    next10:
2458.    ;}
2459.
2460.
2461.    int xx11 = 0, qq11 = 0, contactpnt11 = 0, l11, i11;
2462.
2463.
2464.    for (i11 = 0; i11 < isobox11; i11= i11+1)
2465.
2466.    {
2467.
2468.    xx11 = isocounterbox11[i11];
2469.
2470.    rightisosceles [xx11][0][0] = rightisosceles [xx11][0][0] + resultux
    [xx11] ;
2471.
2472.    rightisosceles [xx11][0][1] = rightisosceles [xx11][0][1] + resultuy
    [xx11] ;
2473.
2474.
2475.
2476.    for ( l11 = 0; l11 < isobox11; l11 = l11+1)
2477.
2478.    {
2479.
2480.
2481.    qq11 = isocounterbox11[l11];
2482.
2483.
2484.
2485.    if ( ((pow((rightisosceles [xx11][0][0] -
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) > 0) && ((pow((rightisosceles [xx11][0]
   [0] -
```

```
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) <= 200))
2486.
2487.    {
2488.
2489.    contactpnt11 = contactpnt11 + 1;
2490.
2491.    if (contactpnt11 >= 4) {goto next11;}
2492.
2493.
2494.    if ( rightisosceles [qq11][3][1] >  rightisosceles [xx11][1][1] )
2495.
2496.    {
2497.
2498.
2499.    ydisplat [xx11][contactpnt11] = ydisplacement2  + (deluy [xx11][cont
   actpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][0] -
     rightisosceles [qq11][0][0]));
2500.    xdisplat [xx11][contactpnt11] = xdisplacement2 + (delux [xx11][conta
   ctpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][1] -
   rightisosceles [qq11][0][1]));
2501.
2502.
2503.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2504.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2505.
2506.
2507.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
2508.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2509.
2510.
2511.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2512.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2513.
2514.
2515.    yforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * sin (alpha1)) -
     ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha1)));
```

```
2516.    xforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * cos (alpha1)) + ((fn [xx11][contactpnt11] + dn [xx11][
   contactpnt11]) * sin (alpha1)));
2517.
2518.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2519.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2520.
2521.
2522.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2523.    y [xx11] = fysum [xx11][contactpnt11];
2524.    resultfy [xx11] += y[xx11];
2525.
2526.
2527.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2528.    x [xx11] = fxsum [xx11][contactpnt11];
2529.    resultfx [xx11] += x[xx11];
2530.
2531.
2532.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][3][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][3][1] -
    rightisosceles [xx11][0][1]));
2533.    m [xx11] = msum [xx11][contactpnt11];
2534.    resultm [xx11] += m[xx11];
2535.
2536.
2537.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement) *1000/ dmass); // mm/sec
2538.    udotysum [xx11] = udoty [xx11][contactpnt11];
2539.    resultudoty [xx11] += udotysum [xx11];
2540.
2541.
2542.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2543.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2544.    resultudotx [xx11] += udotxsum [xx11];
2545.
2546.
2547.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2548.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
2549.    resultthetadot [xx11] += thetadotsum [xx11];
2550.
2551.
2552.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2553.    deluysum [xx11] = deluy [xx11][contactpnt11];
2554.    resultdeluy [xx11] += deluysum [xx11];
2555.
2556.
2557.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
```

```
2558.    deluxsum [xx11] = delux [xx11][contactpnt11];
2559.    resultdelux [xx11] += deluxsum [xx11];
2560.
2561.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2562.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2563.    resultdeltheta [xx11] += delthetasum [xx11];
2564.
2565.
2566.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2567.    uysum [xx11] = uy [xx11][contactpnt11];
2568.    resultuy [xx11]+= uysum [xx11];
2569.
2570.
2571.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2572.    uxsum [xx11] = ux [xx11][contactpnt11];
2573.    resultux [xx11] += uxsum [xx11];
2574.
2575.
2576.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2577.    thetasum [xx11] = theta [xx11][contactpnt11];
2578.    resulttheta [xx11] += thetasum [xx11];
2579.    alpha [xx11] = alpha1 + resulttheta [xx11];
2580.
2581.    }
2582.
2583.    else
2584.
2585.    {
2586.
2587.
2588.
2589.    ydisplat [xx11][contactpnt11] += ydisplacement2  + (deluy [xx11][con
   tactpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][0][0] -
     rightisosceles [qq11][0][0]));
2590.    xdisplat [xx11][contactpnt11] += xdisplacement2 + (delux [xx11][cont
   actpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][1][1] -
   rightisosceles [qq11][0][1]));
2591.
2592.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2593.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2594.
2595.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
2596.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2597.
2598.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2599.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2600.
2601.    yforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * sin (alpha [xx11])) -
    ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha [xx11])
   ));
2602.    xforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * cos (alpha [xx11])) + ((fn [xx11][contactpnt11] + dn [x
   x11][contactpnt11]) * sin (alpha [xx11])));
2603.
2604.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2605.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2606.
2607.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2608.    x [xx11] = fxsum [xx11][contactpnt11];
2609.    resultfx [xx11] += x[xx11];
2610.
2611.
2612.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2613.    y [xx11] = fysum [xx11][contactpnt11];
2614.    resultfy [xx11] += y[xx11];
2615.
2616.
2617.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][1][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][1][1] -
    rightisosceles [xx11][0][1]));
2618.    m [xx11] = msum [xx11][contactpnt11];
2619.    resultm [xx11] += m[xx11];
2620.
2621.
2622.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2623.    udotysum [xx11] = udoty [xx11][contactpnt11];
2624.    resultudoty [xx11] += udotysum [xx11];
2625.
2626.
2627.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2628.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2629.    resultudotx [xx11] += udotxsum [xx11];
2630.
2631.
2632.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2633.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
```

```
2634.    resultthetadot [xx11] += thetadotsum [xx11];
2635.
2636.
2637.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2638.    deluysum [xx11] = deluy [xx11][contactpnt11];
2639.    resultdeluy [xx11] += deluysum [xx11];
2640.
2641.
2642.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
2643.    deluxsum [xx11] = delux [xx11][contactpnt11];
2644.    resultdelux [xx11] += deluxsum [xx11];
2645.
2646.
2647.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2648.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2649.    resultdeltheta [xx11] += delthetasum [xx11];
2650.
2651.
2652.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2653.    uysum [xx11] = uy [xx11][contactpnt11];
2654.    resultuy [xx11]+= uysum [xx11];
2655.
2656.
2657.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2658.    uxsum [xx11] = ux [xx11][contactpnt11];
2659.    resultux [xx11] += uxsum [xx11];
2660.
2661.
2662.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2663.    thetasum [xx11] = theta [xx11][contactpnt11];
2664.    resulttheta [xx11] += thetasum [xx11];
2665.    alpha [xx11] = alpha1 + resulttheta [xx11];
2666.
2667.
2668.
2669.    }
2670.
2671.    }
2672.
2673.    }
2674.
2675.    next11:
2676.    ;}
2677.
2678.
2679.
2680.    int xx12 = 0, qq12 = 0, contactpnt12 = 0, l12, i12;
2681.
2682.
```

```
2683.    for (i12 = 0; i12 < isobox12; i12= i12+1)
2684.
2685.    {
2686.
2687.    xx12 = isocounterbox12[i12];
2688.
2689.    rightisosceles [xx12][0][0] = rightisosceles [xx12][0][0] + resultux
    [xx12] ;
2690.
2691.    rightisosceles [xx12][0][1] = rightisosceles [xx12][0][1] + resultuy
    [xx12] ;
2692.
2693.
2694.
2695.    for ( l12 = 0; l12 < isobox12; l12 = l12+1)
2696.
2697.    {
2698.
2699.
2700.    qq12 = isocounterbox12[l12];
2701.
2702.
2703.
2704.    if ( ((pow((rightisosceles [xx12][0][0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) > 0) && ((pow((rightisosceles [xx12][0]
   [0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) <= 200))
2705.
2706.    {
2707.
2708.    contactpnt12 = contactpnt12 + 1;
2709.
2710.    if (contactpnt12 >= 4) {goto next12;}
2711.
2712.
2713.    if ( rightisosceles [qq12][3][1] >  rightisosceles [xx12][1][1] )
2714.
2715.    {
2716.
2717.
2718.    ydisplat [xx12][contactpnt12] = ydisplacement2  + (deluy [xx12][cont
   actpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][0] -
    rightisosceles [qq12][0][0]));
2719.    xdisplat [xx12][contactpnt12] = xdisplacement2 + (delux [xx12][conta
   ctpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][1] - rightisosceles [xx12][0][1])  -
```

```
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][1] -
   rightisosceles [qq12][0][1]));
2720.
2721.
2722.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2723.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2724.
2725.
2726.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2727.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2728.
2729.
2730.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2731.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2732.
2733.
2734.    yforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * sin (alpha1)) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha1)));
2735.    xforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * cos (alpha1)) + ((fn [xx12][contactpnt12] + dn [xx12][
   contactpnt12]) * sin (alpha1)));
2736.
2737.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2738.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2739.
2740.
2741.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2742.    y [xx12] = fysum [xx12][contactpnt12];
2743.    resultfy [xx12] += y[xx12];
2744.
2745.
2746.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2747.    x [xx12] = fxsum [xx12][contactpnt12];
2748.    resultfx [xx12] += x[xx12];
2749.
2750.
2751.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][3][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][3][1] -
    rightisosceles [xx12][0][1]));
2752.    m [xx12] = msum [xx12][contactpnt12];
2753.    resultm [xx12] += m[xx12];
2754.
2755.
```

```
2756.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement) *1000/ dmass); // mm/sec
2757.    udotysum [xx12] = udoty [xx12][contactpnt12];
2758.    resultudoty [xx12] += udotysum [xx12];
2759.
2760.
2761.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2762.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2763.    resultudotx [xx12] += udotxsum [xx12];
2764.
2765.
2766.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2767.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2768.    resultthetadot [xx12] += thetadotsum [xx12];
2769.
2770.
2771.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2772.    deluysum [xx12] = deluy [xx12][contactpnt12];
2773.    resultdeluy [xx12] += deluysum [xx12];
2774.
2775.
2776.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2777.    deluxsum [xx12] = delux [xx12][contactpnt12];
2778.    resultdelux [xx12] += deluxsum [xx12];
2779.
2780.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2781.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2782.    resultdeltheta [xx12] += delthetasum [xx12];
2783.
2784.
2785.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2786.    uysum [xx12] = uy [xx12][contactpnt12];
2787.    resultuy [xx12]+= uysum [xx12];
2788.
2789.
2790.    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
2791.    uxsum [xx12] = ux [xx12][contactpnt12];
2792.    resultux [xx12] += uxsum [xx12];
2793.
2794.
2795.    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
2796.    thetasum [xx12] = theta [xx12][contactpnt12];
2797.    resultttheta [xx12] += thetasum [xx12];
2798.    alpha [xx12] = alpha1 + resultttheta [xx12];
2799.
2800.    }
2801.
```

```
2802.    else
2803.
2804.    {
2805.
2806.
2807.
2808.    ydisplat [xx12][contactpnt12] += ydisplacement2  + (deluy [xx12][con
   tactpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][0][0] -
    rightisosceles [qq12][0][0]));
2809.    xdisplat [xx12][contactpnt12] += xdisplacement2 + (delux [xx12][cont
   actpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][1] - rightisosceles [xx12][0][1])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][1][1] -
   rightisosceles [qq12][0][1]));
2810.
2811.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2812.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2813.
2814.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2815.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2816.
2817.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2818.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2819.
2820.    yforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * sin (alpha [xx12])) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha [xx12])
   ));
2821.    xforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * cos (alpha [xx12])) + ((fn [xx12][contactpnt12] + dn [x
   x12][contactpnt12]) * sin (alpha [xx12])));
2822.
2823.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2824.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2825.
2826.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2827.    x [xx12] = fxsum [xx12][contactpnt12];
2828.    resultfx [xx12] += x[xx12];
2829.
2830.
2831.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2832.    y [xx12] = fysum [xx12][contactpnt12];
```

```
2833.    resultfy [xx12] += y[xx12];
2834.
2835.
2836.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][1][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][1][1] -
    rightisosceles [xx12][0][1]));
2837.    m [xx12] = msum [xx12][contactpnt12];
2838.    resultm [xx12] += m[xx12];
2839.
2840.
2841.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2842.    udotysum [xx12] = udoty [xx12][contactpnt12];
2843.    resultudoty [xx12] += udotysum [xx12];
2844.
2845.
2846.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2847.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2848.    resultudotx [xx12] += udotxsum [xx12];
2849.
2850.
2851.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2852.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2853.    resultthetadot [xx12] += thetadotsum [xx12];
2854.
2855.
2856.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2857.    deluysum [xx12] = deluy [xx12][contactpnt12];
2858.    resultdeluy [xx12] += deluysum [xx12];
2859.
2860.
2861.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2862.    deluxsum [xx12] = delux [xx12][contactpnt12];
2863.    resultdelux [xx12] += deluxsum [xx12];
2864.
2865.
2866.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2867.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2868.    resultdeltheta [xx12] += delthetasum [xx12];
2869.
2870.
2871.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2872.    uysum [xx12] = uy [xx12][contactpnt12];
2873.    resultuy [xx12]+= uysum [xx12];
2874.
2875.
```

```
    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
    uxsum [xx12] = ux [xx12][contactpnt12];
    resultux [xx12] += uxsum [xx12];


    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
    thetasum [xx12] = theta [xx12][contactpnt12];
    resulttheta [xx12] += thetasum [xx12];
    alpha [xx12] = alpha1 + resulttheta [xx12];

    }
    }

    }

    next12:
    ;}


    int xx13 = 0, qq13 = 0, contactpnt13 = 0, l13, i13;


    for (i13 = 0; i13 < isobox13; i13= i13+1)

    {

    xx13 = isocounterbox13[i13];


    rightisosceles [xx13][0][0] = rightisosceles [xx13][0][0] + resultux
    [xx13] ;

    rightisosceles [xx13][0][1] = rightisosceles [xx13][0][1] + resultuy
    [xx13] ;


    for ( l13 = 0; l13 < isobox13; l13 = l13+1)

    {


    qq13 = isocounterbox13[l13];


    if ( ((pow((rightisosceles [xx13][0][0] -
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) > 0) && ((pow((rightisosceles [xx13][0]
   [0] -
```

```
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) <= 200))
2923.
2924.    {
2925.
2926.    contactpnt13 = contactpnt13 + 1;
2927.
2928.    if (contactpnt13 >= 4) {goto next13;}
2929.
2930.
2931.    if ( rightisosceles [qq13][3][1] >  rightisosceles [xx13][1][1] )
2932.
2933.    {
2934.
2935.
2936.    ydisplat [xx13][contactpnt13] = ydisplacement2  + (deluy [xx13][cont
   actpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][0] -
    rightisosceles [qq13][0][0]));
2937.    xdisplat [xx13][contactpnt13] = xdisplacement2 + (delux [xx13][conta
   ctpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][1] -
   rightisosceles [qq13][0][1]));
2938.
2939.
2940.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
2941.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
2942.
2943.
2944.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
2945.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
2946.
2947.
2948.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
2949.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
2950.
2951.
2952.    yforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * sin (alpha1)) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha1)));
```

```
2953.    xforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * cos (alpha1)) + ((fn [xx13][contactpnt13] + dn [xx13][
   contactpnt13]) * sin (alpha1)));
2954.
2955.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
2956.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
2957.
2958.
2959.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
2960.    y [xx13] = fysum [xx13][contactpnt13];
2961.    resultfy [xx13] += y[xx13];
2962.
2963.
2964.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
2965.    x [xx13] = fxsum [xx13][contactpnt13];
2966.    resultfx [xx13] += x[xx13];
2967.
2968.
2969.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][3][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][3][1] -
    rightisosceles [xx13][0][1]));
2970.    m [xx13] = msum [xx13][contactpnt13];
2971.    resultm [xx13] += m[xx13];
2972.
2973.
2974.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement) *1000/ dmass); // mm/sec
2975.    udotysum [xx13] = udoty [xx13][contactpnt13];
2976.    resultudoty [xx13] += udotysum [xx13];
2977.
2978.
2979.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
2980.    udotxsum [xx13] = udotx [xx13][contactpnt13];
2981.    resultudotx [xx13] += udotxsum [xx13];
2982.
2983.
2984.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
2985.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
2986.    resultthetadot [xx13] += thetadotsum [xx13];
2987.
2988.
2989.    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
   ement;
2990.    deluysum [xx13] = deluy [xx13][contactpnt13];
2991.    resultdeluy [xx13] += deluysum [xx13];
2992.
2993.
2994.    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
   ement;
```

```
2995.    deluxsum [xx13] = delux [xx13][contactpnt13];
2996.    resultdelux [xx13] += deluxsum [xx13];
2997.
2998.    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
   e_increment;
2999.    delthetasum [xx13] = deltheta [xx13][contactpnt13];
3000.    resultdeltheta [xx13] += delthetasum [xx13];
3001.
3002.
3003.    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
3004.    uysum [xx13] = uy [xx13][contactpnt13];
3005.    resultuy [xx13]+= uysum [xx13];
3006.
3007.
3008.    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
3009.    uxsum [xx13] = ux [xx13][contactpnt13];
3010.    resultux [xx13] += uxsum [xx13];
3011.
3012.
3013.    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
3014.    thetasum [xx13] = theta [xx13][contactpnt13];
3015.    resulttheta [xx13] += thetasum [xx13];
3016.    alpha [xx13] = alpha1 + resulttheta [xx13];
3017.
3018.    }
3019.
3020.    else
3021.
3022.    {
3023.
3024.
3025.
3026.    ydisplat [xx13][contactpnt13] += ydisplacement2  + (deluy [xx13][con
   tactpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][0][0] -
    rightisosceles [qq13][0][0]));
3027.    xdisplat [xx13][contactpnt13] += xdisplacement2 + (delux [xx13][cont
   actpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][1][1] -
   rightisosceles [qq13][0][1]));
3028.
3029.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
3030.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
3031.
3032.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
3033.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
3034.
3035.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
3036.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
3037.
3038.    yforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * sin (alpha [xx13])) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha [xx13])
   ));
3039.    xforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * cos (alpha [xx13])) + ((fn [xx13][contactpnt13] + dn [x
   x13][contactpnt13]) * sin (alpha [xx13])));
3040.
3041.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
3042.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
3043.
3044.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
3045.    x [xx13] = fxsum [xx13][contactpnt13];
3046.    resultfx [xx13] += x[xx13];
3047.
3048.
3049.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
3050.    y [xx13] = fysum [xx13][contactpnt13];
3051.    resultfy [xx13] += y[xx13];
3052.
3053.
3054.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][1][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][1][1] -
    rightisosceles [xx13][0][1]));
3055.    m [xx13] = msum [xx13][contactpnt13];
3056.    resultm [xx13] += m[xx13];
3057.
3058.
3059.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3060.    udotysum [xx13] = udoty [xx13][contactpnt13];
3061.    resultudoty [xx13] += udotysum [xx13];
3062.
3063.
3064.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
3065.    udotxsum [xx13] = udotx [xx13][contactpnt13];
3066.    resultudotx [xx13] += udotxsum [xx13];
3067.
3068.
3069.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
3070.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
```

```
    resultthetadot [xx13] += thetadotsum [xx13];


    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
ement;
    deluysum [xx13] = deluy [xx13][contactpnt13];
    resultdeluy [xx13] += deluysum [xx13];


    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
ement;
    deluxsum [xx13] = delux [xx13][contactpnt13];
    resultdelux [xx13] += deluxsum [xx13];


    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
e_increment;
    delthetasum [xx13] = deltheta [xx13][contactpnt13];
    resultdeltheta [xx13] += delthetasum [xx13];


    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
    uysum [xx13] = uy [xx13][contactpnt13];
    resultuy [xx13]+= uysum [xx13];


    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
    uxsum [xx13] = ux [xx13][contactpnt13];
    resultux [xx13] += uxsum [xx13];


    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
    thetasum [xx13] = theta [xx13][contactpnt13];
    resulttheta [xx13] += thetasum [xx13];
    alpha [xx13] = alpha1 + resulttheta [xx13];



    }

    }

    }

    next13:
    ;}


    int xx14 = 0, qq14 = 0, contactpnt14 = 0, l14, i14;

```

```
3120.    for (i14 = 0; i14 < isobox14; i14= i14+1)
3121.
3122.    {
3123.
3124.    xx14 = isocounterbox14[i14];
3125.
3126.    rightisosceles [xx14][0][0] = rightisosceles [xx14][0][0] + resultux
    [xx14] ;
3127.
3128.    rightisosceles [xx14][0][1] = rightisosceles [xx14][0][1] + resultuy
    [xx14] ;
3129.
3130.
3131.
3132.    for ( l14 = 0; l14 < isobox14; l14 = l14+1)
3133.
3134.    {
3135.
3136.
3137.    qq14 = isocounterbox14[l14];
3138.
3139.
3140.
3141.    if ( ((pow((rightisosceles [xx14][0][0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) > 0) && ((pow((rightisosceles [xx14][0]
   [0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) <= 200))
3142.
3143.    {
3144.
3145.    contactpnt14 = contactpnt14 + 1;
3146.
3147.    if (contactpnt14 >= 4) {goto next14;}
3148.
3149.
3150.    if ( rightisosceles [qq14][3][1] >  rightisosceles [xx14][1][1] )
3151.
3152.    {
3153.
3154.
3155.    ydisplat [xx14][contactpnt14] = ydisplacement2  + (deluy [xx14][cont
   actpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][0] -
    rightisosceles [qq14][0][0]));
3156.    xdisplat [xx14][contactpnt14] = xdisplacement2 + (delux [xx14][conta
   ctpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][1] - rightisosceles [xx14][0][1])  -
```

```
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][1] -
   rightisosceles [qq14][0][1]));
3157.
3158.
3159.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3160.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3161.
3162.
3163.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3164.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3165.
3166.
3167.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3168.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3169.
3170.
3171.    yforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * sin (alpha1)) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha1)));
3172.    xforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * cos (alpha1)) + ((fn [xx14][contactpnt14] + dn [xx14][
   contactpnt14]) * sin (alpha1)));
3173.
3174.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3175.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3176.
3177.
3178.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3179.    y [xx14] = fysum [xx14][contactpnt14];
3180.    resultfy [xx14] += y[xx14];
3181.
3182.
3183.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3184.    x [xx14] = fxsum [xx14][contactpnt14];
3185.    resultfx [xx14] += x[xx14];
3186.
3187.
3188.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][3][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][3][1] -
    rightisosceles [xx14][0][1]));
3189.    m [xx14] = msum [xx14][contactpnt14];
3190.    resultm [xx14] += m[xx14];
3191.
3192.
```

```
3193.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement) *1000/ dmass); // mm/sec
3194.    udotysum [xx14] = udoty [xx14][contactpnt14];
3195.    resultudoty [xx14] += udotysum [xx14];
3196.
3197.
3198.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3199.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3200.    resultudotx [xx14] += udotxsum [xx14];
3201.
3202.
3203.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3204.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3205.    resultthetadot [xx14] += thetadotsum [xx14];
3206.
3207.
3208.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3209.    deluysum [xx14] = deluy [xx14][contactpnt14];
3210.    resultdeluy [xx14] += deluysum [xx14];
3211.
3212.
3213.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3214.    deluxsum [xx14] = delux [xx14][contactpnt14];
3215.    resultdelux [xx14] += deluxsum [xx14];
3216.
3217.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3218.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3219.    resultdeltheta [xx14] += delthetasum [xx14];
3220.
3221.
3222.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3223.    uysum [xx14] = uy [xx14][contactpnt14];
3224.    resultuy [xx14]+= uysum [xx14];
3225.
3226.
3227.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3228.    uxsum [xx14] = ux [xx14][contactpnt14];
3229.    resultux [xx14] += uxsum [xx14];
3230.
3231.
3232.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3233.    thetasum [xx14] = theta [xx14][contactpnt14];
3234.    resultttheta [xx14] += thetasum [xx14];
3235.    alpha [xx14] = alpha1 + resultttheta [xx14];
3236.
3237.    }
3238.
```

```
3239.    else
3240.
3241.    {
3242.
3243.
3244.
3245.    ydisplat [xx14][contactpnt14] += ydisplacement2  + (deluy [xx14][con
   tactpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][0][0] -
    rightisosceles [qq14][0][0]));
3246.    xdisplat [xx14][contactpnt14] += xdisplacement2 + (delux [xx14][cont
   actpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][1] - rightisosceles [xx14][0][1])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][1][1] -
   rightisosceles [qq14][0][1]));
3247.
3248.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3249.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3250.
3251.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3252.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3253.
3254.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3255.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3256.
3257.    yforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * sin (alpha [xx14])) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha [xx14])
   ));
3258.    xforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * cos (alpha [xx14])) + ((fn [xx14][contactpnt14] + dn [x
   x14][contactpnt14]) * sin (alpha [xx14])));
3259.
3260.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3261.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3262.
3263.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3264.    x [xx14] = fxsum [xx14][contactpnt14];
3265.    resultfx [xx14] += x[xx14];
3266.
3267.
3268.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3269.    y [xx14] = fysum [xx14][contactpnt14];
```

```
3270.    resultfy [xx14] += y[xx14];
3271.
3272.
3273.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][1][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][1][1] -
    rightisosceles [xx14][0][1]));
3274.    m [xx14] = msum [xx14][contactpnt14];
3275.    resultm [xx14] += m[xx14];
3276.
3277.
3278.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3279.    udotysum [xx14] = udoty [xx14][contactpnt14];
3280.    resultudoty [xx14] += udotysum [xx14];
3281.
3282.
3283.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3284.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3285.    resultudotx [xx14] += udotxsum [xx14];
3286.
3287.
3288.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3289.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3290.    resultthetadot [xx14] += thetadotsum [xx14];
3291.
3292.
3293.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3294.    deluysum [xx14] = deluy [xx14][contactpnt14];
3295.    resultdeluy [xx14] += deluysum [xx14];
3296.
3297.
3298.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3299.    deluxsum [xx14] = delux [xx14][contactpnt14];
3300.    resultdelux [xx14] += deluxsum [xx14];
3301.
3302.
3303.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3304.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3305.    resultdeltheta [xx14] += delthetasum [xx14];
3306.
3307.
3308.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3309.    uysum [xx14] = uy [xx14][contactpnt14];
3310.    resultuy [xx14]+= uysum [xx14];
3311.
3312.
```

```
3313.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3314.    uxsum [xx14] = ux [xx14][contactpnt14];
3315.    resultux [xx14] += uxsum [xx14];
3316.
3317.
3318.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3319.    thetasum [xx14] = theta [xx14][contactpnt14];
3320.    resulttheta [xx14] += thetasum [xx14];
3321.    alpha [xx14] = alpha1 + resulttheta [xx14];
3322.
3323.    }
3324.    }
3325.
3326.    }
3327.
3328.    next14:
3329.    ;}
3330.
3331.
3332.
3333.    int xx15 = 0, qq15 = 0, contactpnt15 = 0, l15, i15;
3334.
3335.
3336.    for (i15 = 0; i15 < isobox15; i15= i15+1)
3337.
3338.    {
3339.
3340.    xx15 = isocounterbox15[i15];
3341.
3342.    rightisosceles [xx15][0][0] = rightisosceles [xx15][0][0] + resultux
    [xx15] ;
3343.
3344.    rightisosceles [xx15][0][1] = rightisosceles [xx15][0][1] + resultuy
    [xx15] ;
3345.
3346.
3347.
3348.    for ( l15 = 0; l15 < isobox15; l15 = l15+1)
3349.
3350.    {
3351.
3352.
3353.    qq15 = isocounterbox15[l15];
3354.
3355.
3356.
3357.    if ( ((pow((rightisosceles [xx15][0][0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) > 0) && ((pow((rightisosceles [xx15][0]
   [0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) <= 200))
```

```
3358.
3359.    {
3360.
3361.    contactpnt15 = contactpnt15 + 1;
3362.
3363.    if (contactpnt15 >= 4) {goto next15;}
3364.
3365.
3366.    if ( rightisosceles [qq15][3][1] >  rightisosceles [xx15][1][1] )
3367.
3368.    {
3369.
3370.
3371.    ydisplat [xx15][contactpnt15] = ydisplacement2  + (deluy [xx15][cont
   actpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][0] -
    rightisosceles [qq15][0][0]));
3372.    xdisplat [xx15][contactpnt15] = xdisplacement2 + (delux [xx15][conta
   ctpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][1] -
   rightisosceles [qq15][0][1]));
3373.
3374.
3375.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3376.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3377.
3378.
3379.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3380.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3381.
3382.
3383.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3384.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3385.
3386.
3387.    yforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * sin (alpha1)) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha1)));
3388.    xforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * cos (alpha1)) + ((fn [xx15][contactpnt15] + dn [xx15][
   contactpnt15]) * sin (alpha1)));
3389.
```

```
3390.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3391.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3392.
3393.
3394.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3395.    y [xx15] = fysum [xx15][contactpnt15];
3396.    resultfy [xx15] += y[xx15];
3397.
3398.
3399.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3400.    x [xx15] = fxsum [xx15][contactpnt15];
3401.    resultfx [xx15] += x[xx15];
3402.
3403.
3404.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][3][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][3][1] -
    rightisosceles [xx15][0][1]));
3405.    m [xx15] = msum [xx15][contactpnt15];
3406.    resultm [xx15] += m[xx15];
3407.
3408.
3409.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement) *1000/ dmass); // mm/sec
3410.    udotysum [xx15] = udoty [xx15][contactpnt15];
3411.    resultudoty [xx15] += udotysum [xx15];
3412.
3413.
3414.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3415.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3416.    resultudotx [xx15] += udotxsum [xx15];
3417.
3418.
3419.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3420.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3421.    resultthetadot [xx15] += thetadotsum [xx15];
3422.
3423.
3424.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3425.    deluysum [xx15] = deluy [xx15][contactpnt15];
3426.    resultdeluy [xx15] += deluysum [xx15];
3427.
3428.
3429.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3430.    deluxsum [xx15] = delux [xx15][contactpnt15];
3431.    resultdelux [xx15] += deluxsum [xx15];
3432.
```

```
3433.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3434.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3435.    resultdeltheta [xx15] += delthetasum [xx15];
3436.
3437.
3438.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3439.    uysum [xx15] = uy [xx15][contactpnt15];
3440.    resultuy [xx15]+= uysum [xx15];
3441.
3442.
3443.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3444.    uxsum [xx15] = ux [xx15][contactpnt15];
3445.    resultux [xx15] += uxsum [xx15];
3446.
3447.
3448.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3449.    thetasum [xx15] = theta [xx15][contactpnt15];
3450.    resulttheta [xx15] += thetasum [xx15];
3451.    alpha [xx15] = alpha1 + resulttheta [xx15];
3452.
3453.    }
3454.
3455.    else
3456.
3457.    {
3458.
3459.
3460.
3461.    ydisplat [xx15][contactpnt15] += ydisplacement2  + (deluy [xx15][con
   tactpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][0][0] -
    rightisosceles [qq15][0][0]));
3462.    xdisplat [xx15][contactpnt15] += xdisplacement2 + (delux [xx15][cont
   actpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][1][1] -
   rightisosceles [qq15][0][1]));
3463.
3464.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3465.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3466.
3467.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3468.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3469.
```

```
3470.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3471.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3472.
3473.    yforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * sin (alpha [xx15])) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha [xx15])
   ));
3474.    xforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * cos (alpha [xx15])) + ((fn [xx15][contactpnt15] + dn [x
   x15][contactpnt15]) * sin (alpha [xx15])));
3475.
3476.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3477.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3478.
3479.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3480.    x [xx15] = fxsum [xx15][contactpnt15];
3481.    resultfx [xx15] += x[xx15];
3482.
3483.
3484.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3485.    y [xx15] = fysum [xx15][contactpnt15];
3486.    resultfy [xx15] += y[xx15];
3487.
3488.
3489.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][1][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][1][1] -
    rightisosceles [xx15][0][1]));
3490.    m [xx15] = msum [xx15][contactpnt15];
3491.    resultm [xx15] += m[xx15];
3492.
3493.
3494.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3495.    udotysum [xx15] = udoty [xx15][contactpnt15];
3496.    resultudoty [xx15] += udotysum [xx15];
3497.
3498.
3499.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3500.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3501.    resultudotx [xx15] += udotxsum [xx15];
3502.
3503.
3504.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3505.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3506.    resultthetadot [xx15] += thetadotsum [xx15];
3507.
3508.
```

```
3509.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3510.    deluysum [xx15] = deluy [xx15][contactpnt15];
3511.    resultdeluy [xx15] += deluysum [xx15];
3512.
3513.
3514.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3515.    deluxsum [xx15] = delux [xx15][contactpnt15];
3516.    resultdelux [xx15] += deluxsum [xx15];
3517.
3518.
3519.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3520.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3521.    resultdeltheta [xx15] += delthetasum [xx15];
3522.
3523.
3524.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3525.    uysum [xx15] = uy [xx15][contactpnt15];
3526.    resultuy [xx15]+= uysum [xx15];
3527.
3528.
3529.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3530.    uxsum [xx15] = ux [xx15][contactpnt15];
3531.    resultux [xx15] += uxsum [xx15];
3532.
3533.
3534.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3535.    thetasum [xx15] = theta [xx15][contactpnt15];
3536.    resulttheta [xx15] += thetasum [xx15];
3537.    alpha [xx15] = alpha1 + resulttheta [xx15];
3538.
3539.    }
3540.    }
3541.
3542.    }
3543.
3544.    next15:
3545.    ;}
3546.
3547.
3548.
3549.    int xx16 = 0, qq16 = 0, contactpnt16 = 0, l16, i16;
3550.
3551.
3552.    for (i16 = 0; i16 < isobox16; i16= i16+1)
3553.
3554.    {
3555.
3556.    xx16 = isocounterbox16[i16];
3557.
```

```
3558.
3559.
3560.    rightisosceles [xx16][0][0] = rightisosceles [xx16][0][0] + resultux
    [xx16] ;
3561.
3562.    rightisosceles [xx16][0][1] = rightisosceles [xx16][0][1] + resultuy
    [xx16] ;
3563.
3564.
3565.
3566.    for ( l16 = 0; l16 < isobox16; l16 = l16+1)
3567.
3568.    {
3569.
3570.
3571.    qq16 = isocounterbox16[l16];
3572.
3573.
3574.
3575.    if ( ((pow((rightisosceles [xx16][0][0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) > 0) && ((pow((rightisosceles [xx16][0]
   [0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) <= 200))
3576.
3577.    {
3578.
3579.    contactpnt16 = contactpnt16 + 1;
3580.
3581.    if (contactpnt16 >= 4) {goto next16;}
3582.
3583.
3584.    if ( rightisosceles [qq16][3][1] >  rightisosceles [xx16][1][1] )
3585.
3586.    {
3587.
3588.
3589.    ydisplat [xx16][contactpnt16] = ydisplacement2  + (deluy [xx16][cont
   actpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][0] -
    rightisosceles [qq16][0][0]));
3590.    xdisplat [xx16][contactpnt16] = xdisplacement2 + (delux [xx16][conta
   ctpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][1] -
   rightisosceles [qq16][0][1]));
3591.
3592.
```

```
3593.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3594.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3595.
3596.
3597.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3598.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3599.
3600.
3601.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3602.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3603.
3604.
3605.    yforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * sin (alpha1)) -
     ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha1)));
3606.    xforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * cos (alpha1)) + ((fn [xx16][contactpnt16] + dn [xx16][
   contactpnt16]) * sin (alpha1)));
3607.
3608.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3609.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3610.
3611.
3612.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3613.    y [xx16] = fysum [xx16][contactpnt16];
3614.    resultfy [xx16] += y[xx16];
3615.
3616.
3617.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3618.    x [xx16] = fxsum [xx16][contactpnt16];
3619.    resultfx [xx16] += x[xx16];
3620.
3621.
3622.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][3][0] - rightisosceles [xx16][0][0]) -
     xforce [xx16][contactpnt16] * ( rightisosceles [qq16][3][1] -
     rightisosceles [xx16][0][1]));
3623.    m [xx16] = msum [xx16][contactpnt16];
3624.    resultm [xx16] += m[xx16];
3625.
3626.
3627.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement) *1000/ dmass); // mm/sec
3628.    udotysum [xx16] = udoty [xx16][contactpnt16];
3629.    resultudoty [xx16] += udotysum [xx16];
3630.
```

```
3631.
3632.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3633.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3634.    resultudotx [xx16] += udotxsum [xx16];
3635.
3636.
3637.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3638.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3639.    resultthetadot [xx16] += thetadotsum [xx16];
3640.
3641.
3642.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3643.    deluysum [xx16] = deluy [xx16][contactpnt16];
3644.    resultdeluy [xx16] += deluysum [xx16];
3645.
3646.
3647.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3648.    deluxsum [xx16] = delux [xx16][contactpnt16];
3649.    resultdelux [xx16] += deluxsum [xx16];
3650.
3651.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3652.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3653.    resultdeltheta [xx16] += delthetasum [xx16];
3654.
3655.
3656.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3657.    uysum [xx16] = uy [xx16][contactpnt16];
3658.    resultuy [xx16]+= uysum [xx16];
3659.
3660.
3661.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3662.    uxsum [xx16] = ux [xx16][contactpnt16];
3663.    resultux [xx16] += uxsum [xx16];
3664.
3665.
3666.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3667.    thetasum [xx16] = theta [xx16][contactpnt16];
3668.    resultttheta [xx16] += thetasum [xx16];
3669.    alpha [xx16] = alpha1 + resultttheta [xx16];
3670.
3671.    }
3672.
3673.    else
3674.
3675.    {
3676.
3677.
```

```
3678.
3679.    ydisplat [xx16][contactpnt16] += ydisplacement2  + (deluy [xx16][con
   tactpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][0][0] -
    rightisosceles [qq16][0][0]));
3680.    xdisplat [xx16][contactpnt16] += xdisplacement2 + (delux [xx16][cont
   actpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][1][1] -
   rightisosceles [qq16][0][1]));
3681.
3682.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3683.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3684.
3685.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3686.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3687.
3688.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3689.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3690.
3691.    yforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * sin (alpha [xx16])) -
    ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha [xx16])
   ));
3692.    xforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * cos (alpha [xx16])) + ((fn [xx16][contactpnt16] + dn [x
   x16][contactpnt16]) * sin (alpha [xx16])));
3693.
3694.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3695.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3696.
3697.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3698.    x [xx16] = fxsum [xx16][contactpnt16];
3699.    resultfx [xx16] += x[xx16];
3700.
3701.
3702.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3703.    y [xx16] = fysum [xx16][contactpnt16];
3704.    resultfy [xx16] += y[xx16];
3705.
3706.
3707.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][1][0] - rightisosceles [xx16][0][0]) -
```

```
    xforce [xx16][contactpnt16] * ( rightisosceles [qq16][1][1] -
    rightisosceles [xx16][0][1]));
3708.    m [xx16] = msum [xx16][contactpnt16];
3709.    resultm [xx16] += m[xx16];
3710.
3711.
3712.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3713.    udotysum [xx16] = udoty [xx16][contactpnt16];
3714.    resultudoty [xx16] += udotysum [xx16];
3715.
3716.
3717.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3718.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3719.    resultudotx [xx16] += udotxsum [xx16];
3720.
3721.
3722.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3723.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3724.    resultthetadot [xx16] += thetadotsum [xx16];
3725.
3726.
3727.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3728.    deluysum [xx16] = deluy [xx16][contactpnt16];
3729.    resultdeluy [xx16] += deluysum [xx16];
3730.
3731.
3732.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3733.    deluxsum [xx16] = delux [xx16][contactpnt16];
3734.    resultdelux [xx16] += deluxsum [xx16];
3735.
3736.
3737.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3738.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3739.    resultdeltheta [xx16] += delthetasum [xx16];
3740.
3741.
3742.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3743.    uysum [xx16] = uy [xx16][contactpnt16];
3744.    resultuy [xx16]+= uysum [xx16];
3745.
3746.
3747.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3748.    uxsum [xx16] = ux [xx16][contactpnt16];
3749.    resultux [xx16] += uxsum [xx16];
3750.
3751.
```

```
3752.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3753.    thetasum [xx16] = theta [xx16][contactpnt16];
3754.    resulttheta [xx16] += thetasum [xx16];
3755.    alpha [xx16] = alpha1 + resulttheta [xx16];
3756.
3757.
3758.
3759.    }
3760.
3761.    }
3762.
3763.    }
3764.
3765.    next16:
3766.    ;}
3767.
3768.
3769.    ;}
3770.
3771.    display:
3772.
3773.
3774.     a_file<< dmass;
3775.
3776.    return 0;
3777.    }
```

Appendix H:

Code for Tailings-DEM (Series 5)

```
# include <iostream>
# include <cmath>
# include <fstream>
#include <numeric>

using namespace std;


 struct geometricalshapes  {  // subroutine for assigning values to the particle geometrical shapes
 float side1, side2, side3, side4, side5;
 float angle1, angle2, angle3, angle4, angle5;
 float height, base, centroidx, centroidy, momentofinertiax, momentofinertiay, area;

};

float ddamping_coefficient, dk;
float dstiffness_coefficient;
float dtime_increment;
float dmass;
float ii;  // mass moment of inertia
float dresult;


int isocounterbox1[100], isocounterbox2[100], isocounterbox3[100], isocounterbox4[100], isocounterbox5[100], isocounterbox6[100], isocounterbox7[100], isocounterbox8[100], isocounterbox9[100], isocounterbox10[100], isocounterbox11[100], isocounterbox12[100], isocounterbox13[100], isocounterbox14[100], isocounterbox15[100], isocounterbox16[100];
float p, p2;
int bb;


float ydisplat [100][100], xdisplat [100][100], delus [100][100], delun [100][100], fn [100][100], fs [100][000], dn [100][100], ds [100][100];
float yforce [100][100], xforce [100][100], fxsum [100][100], x [100], resultfx [100], fysum [100][100];
float y [100], resultfy [100], msum [100][100], m [100], resultm [100], udoty [100][100], udotysum [100], resultudoty [100];
float udotx [100][100], udotxsum [100], resultudotx [100], thetadot [100][100], thetadotsum [100], resultthetadot [100];
float deluy [100][100], deluysum [100], resultdeluy [100], delux [100][100], deluxsum [100], resultdelux [100], deltheta [100][100];
float delthetasum [100], resultdeltheta [100], uy [100][100], uysum [100], resultuy [100], ux [100][100], uxsum [100];
float resultux [100], theta [100][100], thetasum [100], resulttheta [100];


```

```
float rightisosceles [150][4][10];
int isoscelescounter = 0;
float ydisplacement2 = 0.0, yvelocity2= 0;
float xdisplacement2 = 0.0;
float alpha [100], alpha1 = 0.785398;
int v = 1;
float n, n1;
int num3 =  1 + (39-4)/20;    // number of isosceles triangles per row
int hh, oo, pp;


int main ()

{

geometricalshapes righttriangle; //the particle
righttriangle.side1 = 4.243;
righttriangle.side2 = 3;
righttriangle.side3 = 3;
righttriangle.angle1 = 45;
righttriangle.angle2 = 45;
righttriangle.angle3 = 90;
righttriangle.height = 8.48;
righttriangle.base = 16.972;
righttriangle.centroidx = 0;
righttriangle.centroidy = 0.707;
righttriangle.momentofinertiax = 1.123;
righttriangle.momentofinertiay = 3.374;
righttriangle.area = 71.9613;

 ofstream a_file ("data-s5-mw16.txt");


 cout<< "Please enter the stiffness coefficient in N/m" << endl;
 cin>> dstiffness_coefficient;


 cout<< "Please enter the mass of the particle in kg" << endl;
 cin>> dmass;

ii =  ((righttriangle.base/2) * (righttriangle.base/2)* (dmass)/3) ;


 // time increment verification

 begin: //goto label
 cout<< "Please enter the time increment" << endl;
 cin>> dtime_increment;

 dresult = (2 * sqrt (dmass/dstiffness_coefficient));
```

```
90. if (dtime_increment >=  dresult ) // time increment condition
91.
92.{
93.
94. cout<< "Time increment does not satisfy condition"<< endl;
95. goto begin;
96.
97.}
98.
99. else
100.
101.      {
102.        cout<< "Time increment satisfies condition"<< endl;
103.
104.     }
105.
106.
107.     dk = 2 * (sqrt (dmass * dstiffness_coefficient)); // subroutine for
    calculating the damping coefficient
108.     ddamping_coefficient = dk/dtime_increment;
109.
110.
111.
112.
113.
114.
115.     for (hh = 0; hh <150; hh = hh+1) {
116.         for (oo = 0; oo <4; oo = oo+1) { for (pp = 0; pp <10; pp = pp +
    1)
117.         {
118.             rightisosceles [hh][oo][pp] = 0.0;
119.
120.
121.         }
122.         }
123.     }
124.
125.
126.
127.      for (n = 3.0 ; n <= 70; n = n + 20) {       // positioning particle
    s
128.         for (n1 = 4.0 ; n1 <= 39; n1 = n1 + 20) {
129.
130.         rightisosceles [v][0][0] = n1 ;            // right isosceles x
    coordinates
131.         rightisosceles [v][1][0] = rightisosceles [v][0][0] + (righttria
    ngle.base /2);
132.         rightisosceles [v][2][0] = rightisosceles [v][0][0] -
     (righttriangle.base /2);
133.         rightisosceles [v][3][0] = rightisosceles [v][0][0];
134.
```

```
135.         rightisosceles [v][0][1] = n ;    // right isosceles y coordinat
   es
136.         rightisosceles [v][1][1] = rightisosceles [v][0][1] + (righttria
   ngle.height /3);
137.         rightisosceles [v][2][1] = rightisosceles [v][1][1];
138.         rightisosceles [v][3][1] = rightisosceles [v][2][1] -
    (righttriangle.height);
139.
140.
141.
142.         v = v + 1;
143.         isoscelescounter = isoscelescounter + 1;
144.
145.
146.
147.     }
148.     }
149.
150.     p2 = 10/num3;
151.
152.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
153.     ydisplacement2 =  ydisplacement2 + (yvelocity2 * dtime_increment);

154.
155.
156.
157.
158.
159.     for (p = 100; p < 100000 ; p = p+100)  // load cycle in Newtons
160.     {
161.     p2 = p/num3;
162.
163.     yvelocity2 = (p2 *  dtime_increment * 1000 )/ dmass;   // y velocity
    in mm/sec and y displacement in mm for the right isosceles
164.
165.
166.       a_file <<  p << endl;
167.
168.
169.     int isobox1 = 0, isobox2 = 0, isobox3 = 0, isobox4 = 0, isobox5 = 0,
    isobox6 = 0, isobox7 = 0, isobox8 = 0, isobox9 = 0, isobox10 = 0, isobox1
   1 = 0, isobox12 = 0, isobox13 = 0, isobox14 = 0, isobox15 = 0, isobox16 =
   0;
170.
171.
172.     for (bb = 0; bb <100; bb = bb+1) {
173.         isocounterbox1[bb] = 0, isocounterbox2[bb] = 0, isocounterbox3[b
   b] = 0, isocounterbox4[bb]= 0;
174.         isocounterbox5[bb] = 0, isocounterbox6[bb] = 0, isocounterbox7[b
   b] = 0, isocounterbox8[bb] = 0;
```

```
175.          isocounterbox9[bb] = 0, isocounterbox10[bb] = 0, isocounterbox11
   [bb] = 0, isocounterbox12[bb] = 0;
176.          isocounterbox13[bb] = 0, isocounterbox14[bb] = 0, isocounterbox1
   5[bb] = 0, isocounterbox16[bb] = 0;
177.
178.          }
179.
180.
181.
182.     for (int dd = 1; dd <= isoscelescounter ; dd = dd + 1)  // locating
   the isosceles triangles within the 16 screen boxes
183.     {
184.
185.      if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][0]
   <= 11) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][1]<=
   18.5))
186.     {isocounterbox1[isobox1] = dd; isobox1 = isobox1 + 1;}
187.
188.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
189.     {isocounterbox2[isobox2] = dd; isobox2 = isobox2 + 1;}
190.
191.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
192.     {isocounterbox3[isobox3] = dd; isobox3 = isobox3 + 1;}
193.
194.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (0 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 18.5))
195.     {isocounterbox4[isobox4] = dd; isobox4 = isobox4 + 1;}
196.
197.     else if( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0][
   0] <= 11) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 37))
198.     {isocounterbox5[isobox5] = dd; isobox5 = isobox5 + 1;}
199.
200.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
201.     {isocounterbox6[isobox6] = dd; isobox6 = isobox6 + 1;}
202.
203.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
204.     {isocounterbox7[isobox7] = dd; isobox7 = isobox7 + 1;}
205.
206.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (18.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 37))
207.     {isocounterbox8[isobox8] = dd; isobox8 = isobox8 + 1;}
```

```
208.
209.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0][
   1]<= 55.5))
210.     {isocounterbox9[isobox9] = dd; isobox9 = isobox9 + 1;}
211.
212.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
213.     {isocounterbox10[isobox10] = dd; isobox10 = isobox10 + 1;}
214.
215.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
216.     {isocounterbox11[isobox11] = dd; isobox11 = isobox11 + 1;}
217.
218.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (37 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0]
   [1]<= 55.5))
219.     {isocounterbox12[isobox12] = dd; isobox12 = isobox12 + 1;}
220.
221.     else if ( (0 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0]
   [0] <= 11) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][0
   ][1]<= 74))
222.     {isocounterbox13[isobox13] = dd; isobox13 = isobox13 + 1;}
223.
224.     else if ( (11 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 22) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
225.     {isocounterbox14[isobox14] = dd; isobox14 = isobox14 + 1;}
226.
227.     else if ( (22 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 33) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
228.     {isocounterbox15[isobox15] = dd; isobox15 = isobox15 + 1;}
229.
230.     else if ( (33 < rightisosceles [dd][0][0]) && (rightisosceles [dd][0
   ][0] <= 44) && (55.5 < rightisosceles [dd][0][1]) && (rightisosceles [dd][
   0][1]<= 74))
231.     {isocounterbox16[isobox16] = dd; isobox16 = isobox16 + 1;}
232.
233.
234.     }  // end of isosceles triangles location within the 16 screen boxes

235.
236.
237.     int gg, ww;
238.
239.     for (gg = 0; gg < 100; gg = gg+1) {
240.
241.      for (ww = 0; ww < 100; ww = ww +1) {
242.
```

```
243.     ydisplat [gg][ww] = 0.0;
244.     xdisplat [gg][ww] = 0.0;
245.     delus [gg][ww] = 0.0;
246.     delun [gg][ww] = 0.0;
247.     fn [gg][ww] = 0.0;
248.     fs [gg][ww] = 0.0;
249.     dn [gg][ww] = 0.0;
250.     ds [gg][ww] = 0.0;
251.     yforce [gg][ww] = 0.0;
252.     xforce [gg][ww] = 0.0;
253.     fxsum [gg][ww] = 0.0;
254.     x [ww] = 0.0;
255.     resultfx [ww] = 0.0;
256.     fysum [gg][ww] = 0.0;
257.     y [ww] = 0.0;
258.     resultfy [ww] = 0.0;
259.     msum [gg][ww] = 0.0;
260.     m [ww] = 0.0;
261.     resultm [ww] = 0.0;
262.     udoty [gg][ww] = 0.0;
263.     udotysum [ww] = 0.0;
264.     resultudoty [ww] = 0.0;
265.     udotx [gg][ww] = 0.0;
266.     udotxsum [ww] = 0.0;
267.     resultudotx [ww] = 0.0;
268.     thetadot [gg][ww] = 0.0;
269.     thetadotsum [ww] = 0.0;
270.     resultthetadot [ww] = 0.0;
271.     deluy [gg][ww] = 0.0;
272.     deluysum [ww] = 0.0;
273.     resultdeluy [ww] = 0.0;
274.     delux [gg][ww] = 0.0;
275.     deluxsum [ww] = 0.0;
276.     resultdelux [ww] =0.0;
277.     deltheta [gg][ww] = 0.0;
278.     delthetasum [ww] = 0.0;
279.     resultdeltheta [ww] = 0.0;
280.     uy [gg][ww] = 0.0;
281.     uysum [ww] = 0.0;
282.     resultuy [ww] = 0.0;
283.     ux [gg][ww] = 0.0;
284.     uxsum [ww] = 0.0;
285.     resultux [ww] = 0.0;
286.     theta [gg][ww] = 0.0;
287.     thetasum [ww] = 0.0;
288.     resulttheta [ww] = 0.0;
289.     alpha [ww] = 0.0;
290.
291.     }
292.
293.     }
294.
```

```
295.
296.
297.     int xx1 = 0, qq1 = 0, contactpnt1 = 0, l1, i1;
298.
299.
300.     for (i1 = 0; i1 < isobox1; i1= i1+1)
301.
302.     {
303.
304.     xx1 = isocounterbox1[i1];
305.
306.
307.
308.     rightisosceles [xx1][0][0] = rightisosceles [xx1][0][0] + resultux [
   xx1] ;
309.
310.     rightisosceles [xx1][0][1] = rightisosceles [xx1][0][1] + resultuy [
   xx1] ;
311.
312.
313.
314.     for ( l1 = 0; l1 < isobox1; l1 = l1+1)
315.
316.     {
317.
318.
319.     qq1 = isocounterbox1[l1];
320.
321.
322.     if ( ((pow((rightisosceles [xx1][0][0] -
    rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) > 0) && ((pow((rightisosceles [xx1][0][0
   ] - rightisosceles [qq1][0][0]), 2)) + (pow ((rightisosceles [xx1][0][1] -
    rightisosceles [qq1][0][1]), 2)) <= 800))
323.
324.     {
325.
326.     contactpnt1 = contactpnt1 + 1;
327.
328.     if (contactpnt1 >= 4) {goto next1;}
329.
330.     if ( rightisosceles [qq1][3][1] >  rightisosceles [xx1][1][1] )
331.
332.     {
333.
334.
335.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][0] -
    rightisosceles [qq1][0][0]));
```

```
336.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][3][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][3][1] -
   rightisosceles [qq1][0][1]));
337.
338.
339.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
340.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
341.
342.
343.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
344.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
345.
346.
347.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
348.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
349.
350.
351.     yforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * sin (alpha1)) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha1)));
352.     xforce [qq1][contactpnt1] =   (((fs [xx1][contactpnt1] + ds [xx1][co
   ntactpnt1]) * cos (alpha1)) + ((fn [xx1][contactpnt1] + dn [xx1][contactpn
   t1]) * sin (alpha1)));
353.
354.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
355.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
356.
357.
358.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
359.     y [xx1] = fysum [xx1][contactpnt1];
360.     resultfy [xx1] += y[xx1];
361.
362.
363.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
364.     x [xx1] = fxsum [xx1][contactpnt1];
365.     resultfx [xx1] += x[xx1];
366.
367.
368.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][3][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][3][1] -
    rightisosceles [xx1][0][1]));
369.     m [xx1] = msum [xx1][contactpnt1];
```

```
370.     resultm [xx1] += m[xx1];
371.
372.
373.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
374.     udotysum [xx1] = udoty [xx1][contactpnt1];
375.     resultudoty [xx1] += udotysum [xx1];
376.
377.
378.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
379.     udotxsum [xx1] = udotx [xx1][contactpnt1];
380.     resultudotx [xx1] += udotxsum [xx1];
381.
382.
383.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
384.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
385.     resultthetadot [xx1] += thetadotsum [xx1];
386.
387.
388.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
389.     deluysum [xx1] = deluy [xx1][contactpnt1];
390.     resultdeluy [xx1] += deluysum [xx1];
391.
392.
393.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
394.     deluxsum [xx1] = delux [xx1][contactpnt1];
395.     resultdelux [xx1] += deluxsum [xx1];
396.
397.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
398.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
399.     resultdeltheta [xx1] += delthetasum [xx1];
400.
401.
402.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
403.     uysum [xx1] = uy [xx1][contactpnt1];
404.     resultuy [xx1]+= uysum [xx1];
405.
406.
407.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
408.     uxsum [xx1] = ux [xx1][contactpnt1];
409.     resultux [xx1] += uxsum [xx1];
410.
411.
412.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
413.     thetasum [xx1] = theta [xx1][contactpnt1];
414.     resulttheta [xx1] += thetasum [xx1];
415.     alpha [xx1] = alpha1 + resulttheta [xx1];
```

```
416.
417.     }
418.
419.
420.     else
421.     {
422.
423.     ydisplat [xx1][contactpnt1] = ydisplacement2  + (deluy [xx1][contact
   pnt1] -
    deluy [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][0] - rightisosceles [xx1][0][0])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][0][0] -
    rightisosceles [qq1][0][0]));
424.     xdisplat [xx1][contactpnt1] = xdisplacement2 + (delux [xx1][contactp
   nt1] -
    delux [qq1][contactpnt1] + deltheta [xx1][contactpnt1] * ( rightisosceles
    [qq1][1][1] - rightisosceles [xx1][0][1])  -
    deltheta [qq1][contactpnt1] * (rightisosceles [qq1][1][1] -
   rightisosceles [qq1][0][1]));
425.
426.
427.     delus [xx1][contactpnt1] = (xdisplat [xx1][contactpnt1] * cos (alpha
   1) + ydisplat [xx1][contactpnt1] * sin (alpha1));
428.     delun [xx1][contactpnt1] = (ydisplat [xx1][contactpnt1] * cos (alpha
   1) - xdisplat [xx1][contactpnt1] * sin (alpha1));
429.
430.
431.     fn [xx1][contactpnt1] =  delun [xx1][contactpnt1] * (-
   1) *  (dstiffness_coefficient/1000);
432.     fs [xx1][contactpnt1] =  delus [xx1][contactpnt1] *  (dstiffness_coe
   fficient/1000);
433.
434.
435.     dn [xx1][contactpnt1] = delun [xx1][contactpnt1] * (-
   1) * (ddamping_coefficient/1000);
436.     ds [xx1][contactpnt1] = delus [xx1][contactpnt1] * (ddamping_coeffic
   ient/1000);
437.
438.
439.     yforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * sin (alpha [xx1])) -
    ((fn [xx1][contactpnt1] + dn [xx1][contactpnt1]) * cos (alpha [xx1])));

440.     xforce [qq1][contactpnt1] =  (((fs [xx1][contactpnt1] + ds [xx1][con
   tactpnt1]) * cos (alpha [xx1])) + ((fn [xx1][contactpnt1] + dn [xx1][conta
   ctpnt1]) * sin (alpha [xx1])));
441.
442.
443.     yforce [xx1][contactpnt1] = (yforce [qq1][contactpnt1] * -1);
444.     xforce [xx1][contactpnt1] = (xforce [qq1][contactpnt1] * -1);
445.
446.
```

```
447.     fxsum [xx1][contactpnt1] = xforce [xx1][contactpnt1];
448.     x [xx1] = fxsum [xx1][contactpnt1];
449.     resultfx [xx1] += x[xx1];
450.
451.
452.     fysum [xx1][contactpnt1] = yforce [xx1][contactpnt1] + p2 ;
453.     y [xx1] = fysum [xx1][contactpnt1];
454.     resultfy [xx1] += y[xx1];
455.
456.
457.     msum [xx1][contactpnt1] = (yforce [xx1][contactpnt1]* ( rightisoscel
   es [qq1][1][0] - rightisosceles [xx1][0][0]) -
    xforce [xx1][contactpnt1] * ( rightisosceles [qq1][1][1] -
    rightisosceles [xx1][0][1]));
458.     m [xx1] = msum [xx1][contactpnt1];
459.     resultm [xx1] += m[xx1];
460.
461.
462.     udoty [xx1][contactpnt1]=  ((fysum [xx1][contactpnt1] * dtime_increm
   ent) *1000/ dmass); // mm/sec
463.     udotysum [xx1] = udoty [xx1][contactpnt1];
464.     resultudoty [xx1] += udotysum [xx1];
465.
466.
467.     udotx [xx1][contactpnt1] =  ((fxsum [xx1][contactpnt1]* dtime_increm
   ent)*1000/ dmass); //mm/sec
468.     udotxsum [xx1] = udotx [xx1][contactpnt1];
469.     resultudotx [xx1] += udotxsum [xx1];
470.
471.
472.     thetadot [xx1][contactpnt1] = ((msum [xx1][contactpnt1] * dtime_incr
   ement)*1000/ ii); //1/s
473.     thetadotsum [xx1] = thetadot [xx1][contactpnt1];
474.     resultthetadot [xx1] += thetadotsum [xx1];
475.
476.
477.     deluy [xx1][contactpnt1] = udoty [xx1][contactpnt1] * dtime_incremen
   t;
478.     deluysum [xx1] = deluy [xx1][contactpnt1];
479.     resultdeluy [xx1] += deluysum [xx1];
480.
481.
482.     delux [xx1][contactpnt1] = udotx [xx1][contactpnt1] * dtime_incremen
   t;
483.     deluxsum [xx1] = delux [xx1][contactpnt1];
484.     resultdelux [xx1] += deluxsum [xx1];
485.
486.     deltheta [xx1][contactpnt1] = thetadot [xx1][contactpnt1] * dtime_in
   crement;
487.     delthetasum [xx1] = deltheta [xx1][contactpnt1];
488.     resultdeltheta [xx1] += delthetasum [xx1];
489.
```

```
490.
491.     uy [xx1][contactpnt1] = deluy [xx1][contactpnt1];
492.     uysum [xx1] = uy [xx1][contactpnt1];
493.     resultuy [xx1]+= uysum [xx1];
494.
495.
496.     ux [xx1][contactpnt1] = delux [xx1][contactpnt1];
497.     uxsum [xx1] = ux [xx1][contactpnt1];
498.     resultux [xx1] += uxsum [xx1];
499.
500.
501.     theta [xx1][contactpnt1] = deltheta [xx1][contactpnt1];
502.     thetasum [xx1] = theta [xx1][contactpnt1];
503.     resulttheta [xx1] += thetasum [xx1];
504.     alpha [xx1] = alpha1 + resulttheta [xx1];
505.
506.
507.     }
508.
509.
510.
511.      a_file <<  p  << endl;
512.      a_file << xx1 << endl;
513.      a_file << resultuy [xx1]/10 << endl;
514.
515.     if ( resultuy [xx1] > 72 ) { goto display;}
516.
517.
518.
519.     }
520.
521.
522.     }
523.
524.     next1:
525.     ;}
526.
527.
528.
529.     int xx2 = 0, qq2 = 0, contactpnt2 = 0, l2, i2;
530.
531.
532.     for (i2 = 0; i2 < isobox2; i2= i2+1)
533.
534.     {
535.
536.     xx2 = isocounterbox2[i2];
537.
538.
539.     rightisosceles [xx2][0][0] = rightisosceles [xx2][0][0] + resultux [
    xx2] ;
540.
```

```
    rightisosceles [xx2][0][1] = rightisosceles [xx2][0][1] + resultuy [
xx2] ;


    for ( l2 = 0; l2 < isobox2; l2 = l2+1)

    {


    qq2 = isocounterbox2[l2];


    if ( ((pow((rightisosceles [xx2][0][0] -
 rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
 rightisosceles [qq2][0][1]), 2)) > 0) && ((pow((rightisosceles [xx2][0][0
] - rightisosceles [qq2][0][0]), 2)) + (pow ((rightisosceles [xx2][0][1] -
 rightisosceles [qq2][0][1]), 2)) <= 800))

    {

    contactpnt2 = contactpnt2 + 1;

    if (contactpnt2 >= 4) {goto next2;}


    if ( rightisosceles [qq2][3][1] >  rightisosceles [xx2][1][1] )

    {


    ydisplat [xx2][contactpnt2] = ydisplacement2  + (deluy [xx2][contact
pnt2] -
 deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
 [qq2][3][0] - rightisosceles [xx2][0][0])  -
 deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][0] -
 rightisosceles [qq2][0][0]));
    xdisplat [xx2][contactpnt2] = xdisplacement2 + (delux [xx2][contactp
nt2] -
 delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
 [qq2][3][1] - rightisosceles [xx2][0][1])  -
 deltheta [qq2][contactpnt2] * (rightisosceles [qq2][3][1] -
rightisosceles [qq2][0][1]));


    delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
    delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
1) - xdisplat [xx2][contactpnt2] * sin (alpha1));


```

```
575.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
576.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
577.
578.
579.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
580.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
581.
582.
583.     yforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * sin (alpha1)) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha1)));
584.     xforce [qq2][contactpnt2] =   (((fs [xx2][contactpnt2] + ds [xx2][co
   ntactpnt2]) * cos (alpha1)) + ((fn [xx2][contactpnt2] + dn [xx2][contactpn
   t2]) * sin (alpha1)));
585.
586.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
587.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
588.
589.
590.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
591.     y [xx2] = fysum [xx2][contactpnt2];
592.     resultfy [xx2] += y[xx2];
593.
594.
595.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
596.     x [xx2] = fxsum [xx2][contactpnt2];
597.     resultfx [xx2] += x[xx2];
598.
599.
600.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][3][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][3][1] -
    rightisosceles [xx2][0][1]));
601.     m [xx2] = msum [xx2][contactpnt2];
602.     resultm [xx2] += m[xx2];
603.
604.
605.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent) *1000/ dmass); // mm/sec
606.     udotysum [xx2] = udoty [xx2][contactpnt2];
607.     resultudoty [xx2] += udotysum [xx2];
608.
609.
610.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
611.     udotxsum [xx2] = udotx [xx2][contactpnt2];
612.     resultudotx [xx2] += udotxsum [xx2];
613.
```

```
614.
615.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
616.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
617.     resultthetadot [xx2] += thetadotsum [xx2];
618.
619.
620.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
621.     deluysum [xx2] = deluy [xx2][contactpnt2];
622.     resultdeluy [xx2] += deluysum [xx2];
623.
624.
625.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
626.     deluxsum [xx2] = delux [xx2][contactpnt2];
627.     resultdelux [xx2] += deluxsum [xx2];
628.
629.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
630.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
631.     resultdeltheta [xx2] += delthetasum [xx2];
632.
633.
634.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
635.     uysum [xx2] = uy [xx2][contactpnt2];
636.     resultuy [xx2]+= uysum [xx2];
637.
638.
639.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
640.     uxsum [xx2] = ux [xx2][contactpnt2];
641.     resultux [xx2] += uxsum [xx2];
642.
643.
644.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
645.     thetasum [xx2] = theta [xx2][contactpnt2];
646.     resulttheta [xx2] += thetasum [xx2];
647.     alpha [xx2] = alpha1 + resulttheta [xx2];
648.
649.     }
650.
651.     else
652.
653.     {
654.
655.
656.     ydisplat [xx2][contactpnt2] += ydisplacement2  + (deluy [xx2][contac
   tpnt2] -
    deluy [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][0] - rightisosceles [xx2][0][0])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][0][0] -
    rightisosceles [qq2][0][0]));
```

```
657.     xdisplat [xx2][contactpnt2] += xdisplacement2 + (delux [xx2][contact
   pnt2] -
    delux [qq2][contactpnt2] + deltheta [xx2][contactpnt2] * ( rightisosceles
    [qq2][1][1] - rightisosceles [xx2][0][1])  -
    deltheta [qq2][contactpnt2] * (rightisosceles [qq2][1][1] -
   rightisosceles [qq2][0][1]));
658.
659.     delus [xx2][contactpnt2] = (xdisplat [xx2][contactpnt2] * cos (alpha
   1) + ydisplat [xx2][contactpnt2] * sin (alpha1));
660.     delun [xx2][contactpnt2] = (ydisplat [xx2][contactpnt2] * cos (alpha
   1) - xdisplat [xx2][contactpnt2] * sin (alpha1));
661.
662.     fn [xx2][contactpnt2] =  delun [xx2][contactpnt2] * (-
   1) *  (dstiffness_coefficient/1000);
663.     fs [xx2][contactpnt2] =  delus [xx2][contactpnt2] *  (dstiffness_coe
   fficient/1000);
664.
665.     dn [xx2][contactpnt2] = delun [xx2][contactpnt2] * (-
   1) * (ddamping_coefficient/1000);
666.     ds [xx2][contactpnt2] = delus [xx2][contactpnt2] * (ddamping_coeffic
   ient/1000);
667.
668.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * sin (alpha [xx2])) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

669.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
   ctpnt2]) * sin (alpha [xx2])));
670.
671.     yforce [xx2][contactpnt2] = (yforce [qq2][contactpnt2] * -1);
672.     xforce [xx2][contactpnt2] = (xforce [qq2][contactpnt2] * -1);
673.
674.     fxsum [xx2][contactpnt2] = xforce [xx2][contactpnt2];
675.     x [xx2] = fxsum [xx2][contactpnt2];
676.     resultfx [xx2] += x[xx2];
677.
678.
679.     fysum [xx2][contactpnt2] = yforce [xx2][contactpnt2] + p2 ;
680.     y [xx2] = fysum [xx2][contactpnt2];
681.     resultfy [xx2] += y[xx2];
682.
683.
684.     msum [xx2][contactpnt2] = (yforce [xx2][contactpnt2]* ( rightisoscel
   es [qq2][1][0] - rightisosceles [xx2][0][0]) -
    xforce [xx2][contactpnt2] * ( rightisosceles [qq2][1][1] -
    rightisosceles [xx2][0][1]));
685.     m [xx2] = msum [xx2][contactpnt2];
686.     resultm [xx2] += m[xx2];
687.
688.
```

```
689.     udoty [xx2][contactpnt2]=  ((fysum [xx2][contactpnt2] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
690.     udotysum [xx2] = udoty [xx2][contactpnt2];
691.     resultudoty [xx2] += udotysum [xx2];
692.
693.
694.     udotx [xx2][contactpnt2] =  ((fxsum [xx2][contactpnt2]* dtime_increm
   ent)*1000/ dmass); //mm/sec
695.     udotxsum [xx2] = udotx [xx2][contactpnt2];
696.     resultudotx [xx2] += udotxsum [xx2];
697.
698.
699.     thetadot [xx2][contactpnt2] = ((msum [xx2][contactpnt2] * dtime_incr
   ement)*1000/ ii); //1/s
700.     thetadotsum [xx2] = thetadot [xx2][contactpnt2];
701.     resultthetadot [xx2] += thetadotsum [xx2];
702.
703.
704.     deluy [xx2][contactpnt2] = udoty [xx2][contactpnt2] * dtime_incremen
   t;
705.     deluysum [xx2] = deluy [xx2][contactpnt2];
706.     resultdeluy [xx2] += deluysum [xx2];
707.
708.
709.     delux [xx2][contactpnt2] = udotx [xx2][contactpnt2] * dtime_incremen
   t;
710.     deluxsum [xx2] = delux [xx2][contactpnt2];
711.     resultdelux [xx2] += deluxsum [xx2];
712.
713.
714.     deltheta [xx2][contactpnt2] = thetadot [xx2][contactpnt2] * dtime_in
   crement;
715.     delthetasum [xx2] = deltheta [xx2][contactpnt2];
716.     resultdeltheta [xx2] += delthetasum [xx2];
717.
718.
719.     uy [xx2][contactpnt2] = deluy [xx2][contactpnt2];
720.     uysum [xx2] = uy [xx2][contactpnt2];
721.     resultuy [xx2]+= uysum [xx2];
722.
723.
724.     ux [xx2][contactpnt2] = delux [xx2][contactpnt2];
725.     uxsum [xx2] = ux [xx2][contactpnt2];
726.     resultux [xx2] += uxsum [xx2];
727.
728.
729.     theta [xx2][contactpnt2] = deltheta [xx2][contactpnt2];
730.     thetasum [xx2] = theta [xx2][contactpnt2];
731.     resulttheta [xx2] += thetasum [xx2];
732.     alpha [xx2] = alpha1 + resulttheta [xx2];
733.
734.     }
```

```
735.     }
736.
737.     }
738.
739.     next2:
740.     ;}
741.
742.     int xx3 = 0, qq3 = 0, contactpnt3 = 0, l3, i3;
743.
744.
745.     for (i3 = 0; i3 < isobox3; i3= i3+1)
746.
747.     {
748.
749.     xx3 = isocounterbox3[i3];
750.
751.
752.     rightisosceles [xx3][0][0] = rightisosceles [xx3][0][0] + resultux [
    xx3] ;
753.
754.     rightisosceles [xx3][0][1] = rightisosceles [xx3][0][1] + resultuy [
    xx3] ;
755.
756.
757.     for ( l3 = 0; l3 < isobox3; l3 = l3+1)
758.
759.     {
760.
761.
762.     qq3 = isocounterbox3[l3];
763.
764.
765.
766.     if ( ((pow((rightisosceles [xx3][0][0] -
     rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
     rightisosceles [qq3][0][1]), 2)) > 0) && ((pow((rightisosceles [xx3][0][0
    ] - rightisosceles [qq3][0][0]), 2)) + (pow ((rightisosceles [xx3][0][1] -
     rightisosceles [qq3][0][1]), 2)) <= 800))
767.
768.     {
769.
770.     contactpnt3 = contactpnt3 + 1;
771.
772.     if (contactpnt3 >= 4) {goto next3;}
773.
774.
775.     if ( rightisosceles [qq3][3][1] >  rightisosceles [xx3][1][1] )
776.
777.     {
778.
779.
```

```
780.     ydisplat [xx3][contactpnt3] = ydisplacement2  + (deluy [xx3][contact
   pnt3] -
    deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][0] - rightisosceles [xx3][0][0])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][0] -
    rightisosceles [qq3][0][0]));
781.     xdisplat [xx3][contactpnt3] = xdisplacement2 + (delux [xx3][contactp
   nt3] -
    delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][3][1] - rightisosceles [xx3][0][1])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][3][1] -
   rightisosceles [qq3][0][1]));
782.
783.
784.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
   1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
785.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
   1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
786.
787.
788.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
   1) *  (dstiffness_coefficient/1000);
789.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
   fficient/1000);
790.
791.
792.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
   1) * (ddamping_coefficient/1000);
793.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
   ient/1000);
794.
795.
796.     yforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * sin (alpha1)) -
    ((fn [xx3][contactpnt3] + dn [xx3][contactpnt3]) * cos (alpha1)));
797.     xforce [qq3][contactpnt3] =   (((fs [xx3][contactpnt3] + ds [xx3][co
   ntactpnt3]) * cos (alpha1)) + ((fn [xx3][contactpnt3] + dn [xx3][contactpn
   t3]) * sin (alpha1)));
798.
799.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
800.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
801.
802.
803.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
804.     y [xx3] = fysum [xx3][contactpnt3];
805.     resultfy [xx3] += y[xx3];
806.
807.
808.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
809.     x [xx3] = fxsum [xx3][contactpnt3];
810.     resultfx [xx3] += x[xx3];
811.
```

```
812.
813.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][3][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][3][1] -
    rightisosceles [xx3][0][1]));
814.     m [xx3] = msum [xx3][contactpnt3];
815.     resultm [xx3] += m[xx3];
816.
817.
818.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent) *1000/ dmass); // mm/sec
819.     udotysum [xx3] = udoty [xx3][contactpnt3];
820.     resultudoty [xx3] += udotysum [xx3];
821.
822.
823.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
824.     udotxsum [xx3] = udotx [xx3][contactpnt3];
825.     resultudotx [xx3] += udotxsum [xx3];
826.
827.
828.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
829.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
830.     resultthetadot [xx3] += thetadotsum [xx3];
831.
832.
833.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
834.     deluysum [xx3] = deluy [xx3][contactpnt3];
835.     resultdeluy [xx3] += deluysum [xx3];
836.
837.
838.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
839.     deluxsum [xx3] = delux [xx3][contactpnt3];
840.     resultdelux [xx3] += deluxsum [xx3];
841.
842.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
843.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
844.     resultdeltheta [xx3] += delthetasum [xx3];
845.
846.
847.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
848.     uysum [xx3] = uy [xx3][contactpnt3];
849.     resultuy [xx3]+= uysum [xx3];
850.
851.
852.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
853.     uxsum [xx3] = ux [xx3][contactpnt3];
854.     resultux [xx3] += uxsum [xx3];
```

```
855.
856.
857.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
858.     thetasum [xx3] = theta [xx3][contactpnt3];
859.     resulttheta [xx3] += thetasum [xx3];
860.     alpha [xx3] = alpha1 + resulttheta [xx3];
861.
862.     }
863.
864.      else
865.     {
866.
867.
868.     ydisplat [xx3][contactpnt2] += ydisplacement2  + (deluy [xx3][contac
   tpnt3] -
    deluy [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][1][0] - rightisosceles [xx3][0][0])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][0][0] -
    rightisosceles [qq3][0][0]));
869.     xdisplat [xx3][contactpnt2] += xdisplacement2 + (delux [xx3][contact
   pnt3] -
    delux [qq3][contactpnt3] + deltheta [xx3][contactpnt3] * ( rightisosceles
    [qq3][1][1] - rightisosceles [xx3][0][1])  -
    deltheta [qq3][contactpnt3] * (rightisosceles [qq3][1][1] -
   rightisosceles [qq3][0][1]));
870.
871.     delus [xx3][contactpnt3] = (xdisplat [xx3][contactpnt3] * cos (alpha
   1) + ydisplat [xx3][contactpnt3] * sin (alpha1));
872.     delun [xx3][contactpnt3] = (ydisplat [xx3][contactpnt3] * cos (alpha
   1) - xdisplat [xx3][contactpnt3] * sin (alpha1));
873.
874.     fn [xx3][contactpnt3] =  delun [xx3][contactpnt3] * (-
   1) *  (dstiffness_coefficient/1000);
875.     fs [xx3][contactpnt3] =  delus [xx3][contactpnt3] *  (dstiffness_coe
   fficient/1000);
876.
877.     dn [xx3][contactpnt3] = delun [xx3][contactpnt3] * (-
   1) * (ddamping_coefficient/1000);
878.     ds [xx3][contactpnt3] = delus [xx3][contactpnt3] * (ddamping_coeffic
   ient/1000);
879.
880.     yforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * sin (alpha [xx2])) -
    ((fn [xx2][contactpnt2] + dn [xx2][contactpnt2]) * cos (alpha [xx2])));

881.     xforce [qq2][contactpnt2] =  (((fs [xx2][contactpnt2] + ds [xx2][con
   tactpnt2]) * cos (alpha [xx2])) + ((fn [xx2][contactpnt2] + dn [xx2][conta
   ctpnt2]) * sin (alpha [xx2])));
882.
883.     yforce [xx3][contactpnt3] = (yforce [qq3][contactpnt3] * -1);
884.     xforce [xx3][contactpnt3] = (xforce [qq3][contactpnt3] * -1);
885.
```

```
886.     fxsum [xx3][contactpnt3] = xforce [xx3][contactpnt3];
887.     x [xx3] = fxsum [xx3][contactpnt3];
888.     resultfx [xx3] += x[xx3];
889.
890.
891.     fysum [xx3][contactpnt3] = yforce [xx3][contactpnt3] + p2 ;
892.     y [xx3] = fysum [xx3][contactpnt3];
893.     resultfy [xx3] += y[xx3];
894.
895.
896.     msum [xx3][contactpnt3] = (yforce [xx3][contactpnt3]* ( rightisoscel
   es [qq3][1][0] - rightisosceles [xx3][0][0]) -
    xforce [xx3][contactpnt3] * ( rightisosceles [qq3][1][1] -
    rightisosceles [xx3][0][1]));
897.     m [xx3] = msum [xx3][contactpnt3];
898.     resultm [xx3] += m[xx3];
899.
900.
901.     udoty [xx3][contactpnt3]=  ((fysum [xx3][contactpnt3] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
902.     udotysum [xx3] = udoty [xx3][contactpnt3];
903.     resultudoty [xx3] += udotysum [xx3];
904.
905.
906.     udotx [xx3][contactpnt3] =  ((fxsum [xx3][contactpnt3]* dtime_increm
   ent)*1000/ dmass); //mm/sec
907.     udotxsum [xx3] = udotx [xx3][contactpnt3];
908.     resultudotx [xx3] += udotxsum [xx3];
909.
910.
911.     thetadot [xx3][contactpnt3] = ((msum [xx3][contactpnt3] * dtime_incr
   ement)*1000/ ii); //1/s
912.     thetadotsum [xx3] = thetadot [xx3][contactpnt3];
913.     resultthetadot [xx3] += thetadotsum [xx3];
914.
915.
916.     deluy [xx3][contactpnt3] = udoty [xx3][contactpnt3] * dtime_incremen
   t;
917.     deluysum [xx3] = deluy [xx3][contactpnt3];
918.     resultdeluy [xx3] += deluysum [xx3];
919.
920.
921.     delux [xx3][contactpnt3] = udotx [xx3][contactpnt3] * dtime_incremen
   t;
922.     deluxsum [xx3] = delux [xx3][contactpnt3];
923.     resultdelux [xx3] += deluxsum [xx3];
924.
925.
926.     deltheta [xx3][contactpnt3] = thetadot [xx3][contactpnt3] * dtime_in
   crement;
927.     delthetasum [xx3] = deltheta [xx3][contactpnt3];
928.     resultdeltheta [xx3] += delthetasum [xx3];
```

```
929.
930.
931.     uy [xx3][contactpnt3] = deluy [xx3][contactpnt3];
932.     uysum [xx3] = uy [xx3][contactpnt3];
933.     resultuy [xx3]+= uysum [xx3];
934.
935.
936.     ux [xx3][contactpnt3] = delux [xx3][contactpnt3];
937.     uxsum [xx3] = ux [xx3][contactpnt3];
938.     resultux [xx3] += uxsum [xx3];
939.
940.
941.     theta [xx3][contactpnt3] = deltheta [xx3][contactpnt3];
942.     thetasum [xx3] = theta [xx3][contactpnt3];
943.     resulttheta [xx3] += thetasum [xx3];
944.     alpha [xx3] = alpha1 + resulttheta [xx3];
945.     }
946.
947.     }
948.
949.     }
950.
951.     next3:
952.     ;}
953.
954.
955.     int xx4 = 0, qq4 = 0, contactpnt4 = 0, l4, i4;
956.
957.
958.     for (i4 = 0; i4 < isobox4; i4= i4+1)
959.
960.     {
961.
962.     xx4 = isocounterbox4[i4];
963.
964.
965.     rightisosceles [xx4][0][0] = rightisosceles [xx4][0][0] + resultux [
    xx4] ;
966.
967.     rightisosceles [xx4][0][1] = rightisosceles [xx4][0][1] + resultuy [
    xx4] ;
968.
969.
970.     for ( l4 = 0; l4 < isobox4; l4 = l4+1)
971.
972.     {
973.
974.
975.     qq4 = isocounterbox4[l4];
976.
977.
978.
```

```
    if ( ((pow((rightisosceles [xx4][0][0] -
     rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) > 0) && ((pow((rightisosceles [xx4][0][0
    ] - rightisosceles [qq4][0][0]), 2)) + (pow ((rightisosceles [xx4][0][1] -
     rightisosceles [qq4][0][1]), 2)) <= 800))

    {

    contactpnt4 = contactpnt4 + 1;

    if (contactpnt4 >= 4) {goto next4;}


    if ( rightisosceles [qq4][3][1] >  rightisosceles [xx4][1][1] )

    {


    ydisplat [xx4][contactpnt4] = ydisplacement2  + (deluy [xx4][contact
   pnt4] -
     deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][0] - rightisosceles [xx4][0][0])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][0] -
     rightisosceles [qq4][0][0]));
    xdisplat [xx4][contactpnt4] = xdisplacement2 + (delux [xx4][contactp
   nt4] -
     delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
     [qq4][3][1] - rightisosceles [xx4][0][1])  -
     deltheta [qq4][contactpnt4] * (rightisosceles [qq4][3][1] -
   rightisosceles [qq4][0][1]));


    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));


    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-
   1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coe
   fficient/1000);


    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-
   1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coeffic
   ient/1000);


```

```
    yforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * sin (alpha1)) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha1)));
    xforce [qq4][contactpnt4] =   (((fs [xx4][contactpnt4] + ds [xx4][co
ntactpnt4]) * cos (alpha1)) + ((fn [xx4][contactpnt4] + dn [xx4][contactpn
t4]) * sin (alpha1)));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisoscel
es [qq4][3][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][3][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
    resultthetadot [xx4] += thetadotsum [xx4];


    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
t;
    deluysum [xx4] = deluy [xx4][contactpnt4];
    resultdeluy [xx4] += deluysum [xx4];

```

```
1050.
1051.    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
   t;
1052.    deluxsum [xx4] = delux [xx4][contactpnt4];
1053.    resultdelux [xx4] += deluxsum [xx4];
1054.
1055.    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
   crement;
1056.    delthetasum [xx4] = deltheta [xx4][contactpnt4];
1057.    resultdeltheta [xx4] += delthetasum [xx4];
1058.
1059.
1060.    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
1061.    uysum [xx4] = uy [xx4][contactpnt4];
1062.    resultuy [xx4]+= uysum [xx4];
1063.
1064.
1065.    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
1066.    uxsum [xx4] = ux [xx4][contactpnt4];
1067.    resultux [xx4] += uxsum [xx4];
1068.
1069.
1070.    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
1071.    thetasum [xx4] = theta [xx4][contactpnt4];
1072.    resulttheta [xx4] += thetasum [xx4];
1073.    alpha [xx4] = alpha1 + resulttheta [xx4];
1074.
1075.    }
1076.
1077.    else
1078.
1079.    {
1080.
1081.
1082.    ydisplat [xx4][contactpnt4] += ydisplacement2  + (deluy [xx4][contac
   tpnt4] -
    deluy [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
    [qq4][1][0] - rightisosceles [xx4][0][0])  -
    deltheta [qq4][contactpnt4] * (rightisosceles [qq4][0][0] -
    rightisosceles [qq4][0][0]));
1083.    xdisplat [xx4][contactpnt4] += xdisplacement2 + (delux [xx4][contact
   pnt4] -
    delux [qq4][contactpnt4] + deltheta [xx4][contactpnt4] * ( rightisosceles
    [qq4][1][1] - rightisosceles [xx4][0][1])  -
    deltheta [qq4][contactpnt4] * (rightisosceles [qq4][1][1] -
   rightisosceles [qq4][0][1]));
1084.
1085.    delus [xx4][contactpnt4] = (xdisplat [xx4][contactpnt4] * cos (alpha
   1) + ydisplat [xx4][contactpnt4] * sin (alpha1));
1086.    delun [xx4][contactpnt4] = (ydisplat [xx4][contactpnt4] * cos (alpha
   1) - xdisplat [xx4][contactpnt4] * sin (alpha1));
1087.
```

```
    fn [xx4][contactpnt4] =  delun [xx4][contactpnt4] * (-1) *  (dstiffness_coefficient/1000);
    fs [xx4][contactpnt4] =  delus [xx4][contactpnt4] *  (dstiffness_coefficient/1000);

    dn [xx4][contactpnt4] = delun [xx4][contactpnt4] * (-1) * (ddamping_coefficient/1000);
    ds [xx4][contactpnt4] = delus [xx4][contactpnt4] * (ddamping_coefficient/1000);

    yforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][contactpnt4]) * sin (alpha [xx4])) -
     ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * cos (alpha [xx4])));

    xforce [qq4][contactpnt4] =  (((fs [xx4][contactpnt4] + ds [xx4][contactpnt4]) * cos (alpha [xx4])) + ((fn [xx4][contactpnt4] + dn [xx4][contactpnt4]) * sin (alpha [xx4])));

    yforce [xx4][contactpnt4] = (yforce [qq4][contactpnt4] * -1);
    xforce [xx4][contactpnt4] = (xforce [qq4][contactpnt4] * -1);

    fxsum [xx4][contactpnt4] = xforce [xx4][contactpnt4];
    x [xx4] = fxsum [xx4][contactpnt4];
    resultfx [xx4] += x[xx4];


    fysum [xx4][contactpnt4] = yforce [xx4][contactpnt4] + p2 ;
    y [xx4] = fysum [xx4][contactpnt4];
    resultfy [xx4] += y[xx4];


    msum [xx4][contactpnt4] = (yforce [xx4][contactpnt4]* ( rightisosceles [qq4][1][0] - rightisosceles [xx4][0][0]) -
     xforce [xx4][contactpnt4] * ( rightisosceles [qq4][1][1] -
     rightisosceles [xx4][0][1]));
    m [xx4] = msum [xx4][contactpnt4];
    resultm [xx4] += m[xx4];


    udoty [xx4][contactpnt4]=  ((fysum [xx4][contactpnt4] * dtime_increment)* 1000/ dmass); // mm/sec
    udotysum [xx4] = udoty [xx4][contactpnt4];
    resultudoty [xx4] += udotysum [xx4];


    udotx [xx4][contactpnt4] =  ((fxsum [xx4][contactpnt4]* dtime_increment)*1000/ dmass); //mm/sec
    udotxsum [xx4] = udotx [xx4][contactpnt4];
    resultudotx [xx4] += udotxsum [xx4];


```

```
    thetadot [xx4][contactpnt4] = ((msum [xx4][contactpnt4] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx4] = thetadot [xx4][contactpnt4];
    resultthetadot [xx4] += thetadotsum [xx4];


    deluy [xx4][contactpnt4] = udoty [xx4][contactpnt4] * dtime_incremen
t;
    deluysum [xx4] = deluy [xx4][contactpnt4];
    resultdeluy [xx4] += deluysum [xx4];


    delux [xx4][contactpnt4] = udotx [xx4][contactpnt4] * dtime_incremen
t;
    deluxsum [xx4] = delux [xx4][contactpnt4];
    resultdelux [xx4] += deluxsum [xx4];


    deltheta [xx4][contactpnt4] = thetadot [xx4][contactpnt4] * dtime_in
crement;
    delthetasum [xx4] = deltheta [xx4][contactpnt4];
    resultdeltheta [xx4] += delthetasum [xx4];


    uy [xx4][contactpnt4] = deluy [xx4][contactpnt4];
    uysum [xx4] = uy [xx4][contactpnt4];
    resultuy [xx4]+= uysum [xx4];


    ux [xx4][contactpnt4] = delux [xx4][contactpnt4];
    uxsum [xx4] = ux [xx4][contactpnt4];
    resultux [xx4] += uxsum [xx4];


    theta [xx4][contactpnt4] = deltheta [xx4][contactpnt4];
    thetasum [xx4] = theta [xx4][contactpnt4];
    resulttheta [xx4] += thetasum [xx4];
    alpha [xx4] = alpha1 + resulttheta [xx4];
    }

    }

    }

    next4:
    ;}


    int xx5 = 0, qq5 = 0, contactpnt5 = 0, l5, i5;


    for (i5 = 0; i5 < isobox5; i5= i5+1)
```

```
1173.
1174.    {
1175.
1176.    xx5 = isocounterbox5[i5];
1177.
1178.    rightisosceles [xx5][0][0] = rightisosceles [xx5][0][0] + resultux [
   xx5] ;
1179.
1180.    rightisosceles [xx5][0][1] = rightisosceles [xx5][0][1] + resultuy [
   xx5] ;
1181.
1182.
1183.
1184.    for ( l5 = 0; l5 < isobox5; l5 = l5+1)
1185.
1186.    {
1187.
1188.
1189.    qq5 = isocounterbox5[l5];
1190.
1191.
1192.
1193.    if ( ((pow((rightisosceles [xx5][0][0] -
    rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) > 0) && ((pow((rightisosceles [xx5][0][0
   ] - rightisosceles [qq5][0][0]), 2)) + (pow ((rightisosceles [xx5][0][1] -
    rightisosceles [qq5][0][1]), 2)) <= 800))
1194.
1195.    {
1196.
1197.    contactpnt5 = contactpnt5 + 1;
1198.
1199.    if (contactpnt5 >= 4) {goto next5;}
1200.
1201.
1202.    if ( rightisosceles [qq5][3][1] >  rightisosceles [xx5][1][1] )
1203.
1204.    {
1205.
1206.
1207.    ydisplat [xx5][contactpnt5] = ydisplacement2  + (deluy [xx5][contact
   pnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][0] -
    rightisosceles [qq5][0][0]));
1208.    xdisplat [xx5][contactpnt5] = xdisplacement2 + (delux [xx5][contactp
   nt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][3][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][3][1] -
   rightisosceles [qq5][0][1]));
```

```
1209.
1210.
1211.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1212.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1213.
1214.
1215.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1216.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1217.
1218.
1219.    dn [xx5][contactpnt5] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1220.    ds [xx5][contactpnt5] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1221.
1222.
1223.    yforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * sin (alpha1)) -
     ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha1)));
1224.    xforce [qq5][contactpnt5] =   (((fs [xx5][contactpnt5] + ds [xx5][co
   ntactpnt5]) * cos (alpha1)) + ((fn [xx5][contactpnt5] + dn [xx5][contactpn
   t5]) * sin (alpha1)));
1225.
1226.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1227.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1228.
1229.
1230.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1231.    y [xx5] = fysum [xx5][contactpnt5];
1232.    resultfy [xx5] += y[xx5];
1233.
1234.
1235.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1236.    x [xx5] = fxsum [xx5][contactpnt5];
1237.    resultfx [xx5] += x[xx5];
1238.
1239.
1240.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][3][0] - rightisosceles [xx5][0][0]) -
     xforce [xx5][contactpnt5] * ( rightisosceles [qq5][3][1] -
     rightisosceles [xx5][0][1]));
1241.    m [xx5] = msum [xx5][contactpnt5];
1242.    resultm [xx5] += m[xx5];
1243.
1244.
1245.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1246.    udotysum [xx5] = udoty [xx5][contactpnt5];
```

```
1247.    resultudoty [xx5] += udotysum [xx5];
1248.
1249.
1250.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1251.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1252.    resultudotx [xx5] += udotxsum [xx5];
1253.
1254.
1255.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1256.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1257.    resultthetadot [xx5] += thetadotsum [xx5];
1258.
1259.
1260.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1261.    deluysum [xx5] = deluy [xx5][contactpnt5];
1262.    resultdeluy [xx5] += deluysum [xx5];
1263.
1264.
1265.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1266.    deluxsum [xx5] = delux [xx5][contactpnt5];
1267.    resultdelux [xx5] += deluxsum [xx5];
1268.
1269.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1270.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1271.    resultdeltheta [xx5] += delthetasum [xx5];
1272.
1273.
1274.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1275.    uysum [xx5] = uy [xx5][contactpnt5];
1276.    resultuy [xx5]+= uysum [xx5];
1277.
1278.
1279.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1280.    uxsum [xx5] = ux [xx5][contactpnt5];
1281.    resultux [xx5] += uxsum [xx5];
1282.
1283.
1284.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1285.    thetasum [xx5] = theta [xx5][contactpnt5];
1286.    resulttheta [xx5] += thetasum [xx5];
1287.    alpha [xx5] = alpha1 + resulttheta [xx5];
1288.
1289.    }
1290.
1291.    else
1292.
1293.    {
```

```
1294.
1295.
1296.    ydisplat [xx5][contactpnt5] += ydisplacement2  + (deluy [xx5][contac
   tpnt5] -
    deluy [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][0] - rightisosceles [xx5][0][0])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][0][0] -
    rightisosceles [qq5][0][0]));
1297.    xdisplat [xx5][contactpnt5] += xdisplacement2 + (delux [xx5][contact
   pnt5] -
    delux [qq5][contactpnt5] + deltheta [xx5][contactpnt5] * ( rightisosceles
    [qq5][1][1] - rightisosceles [xx5][0][1])  -
    deltheta [qq5][contactpnt5] * (rightisosceles [qq5][1][1] -
   rightisosceles [qq5][0][1]));
1298.
1299.    delus [xx5][contactpnt5] = (xdisplat [xx5][contactpnt5] * cos (alpha
   1) + ydisplat [xx5][contactpnt5] * sin (alpha1));
1300.    delun [xx5][contactpnt5] = (ydisplat [xx5][contactpnt5] * cos (alpha
   1) - xdisplat [xx5][contactpnt5] * sin (alpha1));
1301.
1302.    fn [xx5][contactpnt5] =  delun [xx5][contactpnt5] * (-
   1) *  (dstiffness_coefficient/1000);
1303.    fs [xx5][contactpnt5] =  delus [xx5][contactpnt5] *  (dstiffness_coe
   fficient/1000);
1304.
1305.    dn [xx5][contactpnt2] = delun [xx5][contactpnt5] * (-
   1) * (ddamping_coefficient/1000);
1306.    ds [xx5][contactpnt2] = delus [xx5][contactpnt5] * (ddamping_coeffic
   ient/1000);
1307.
1308.    yforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * sin (alpha [xx5])) -
    ((fn [xx5][contactpnt5] + dn [xx5][contactpnt5]) * cos (alpha [xx5])));

1309.    xforce [qq5][contactpnt5] =  (((fs [xx5][contactpnt5] + ds [xx5][con
   tactpnt5]) * cos (alpha [xx5])) + ((fn [xx5][contactpnt5] + dn [xx5][conta
   ctpnt5]) * sin (alpha [xx5])));
1310.
1311.    yforce [xx5][contactpnt5] = (yforce [qq5][contactpnt5] * -1);
1312.    xforce [xx5][contactpnt5] = (xforce [qq5][contactpnt5] * -1);
1313.
1314.    fxsum [xx5][contactpnt5] = xforce [xx5][contactpnt5];
1315.    x [xx5] = fxsum [xx5][contactpnt5];
1316.    resultfx [xx5] += x[xx5];
1317.
1318.
1319.    fysum [xx5][contactpnt5] = yforce [xx5][contactpnt5] + p2 ;
1320.    y [xx5] = fysum [xx5][contactpnt5];
1321.    resultfy [xx5] += y[xx5];
1322.
1323.
```

```
1324.    msum [xx5][contactpnt5] = (yforce [xx5][contactpnt5]* ( rightisoscel
   es [qq5][1][0] - rightisosceles [xx5][0][0]) -
    xforce [xx5][contactpnt5] * ( rightisosceles [qq5][1][1] -
    rightisosceles [xx5][0][1]));
1325.    m [xx5] = msum [xx5][contactpnt5];
1326.    resultm [xx5] += m[xx5];
1327.
1328.
1329.    udoty [xx5][contactpnt5]=  ((fysum [xx5][contactpnt5] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1330.    udotysum [xx5] = udoty [xx5][contactpnt5];
1331.    resultudoty [xx5] += udotysum [xx5];
1332.
1333.
1334.    udotx [xx5][contactpnt5] =  ((fxsum [xx5][contactpnt5]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1335.    udotxsum [xx5] = udotx [xx5][contactpnt5];
1336.    resultudotx [xx5] += udotxsum [xx5];
1337.
1338.
1339.    thetadot [xx5][contactpnt5] = ((msum [xx5][contactpnt5] * dtime_incr
   ement)*1000/ ii); //1/s
1340.    thetadotsum [xx5] = thetadot [xx5][contactpnt5];
1341.    resultthetadot [xx5] += thetadotsum [xx5];
1342.
1343.
1344.    deluy [xx5][contactpnt5] = udoty [xx5][contactpnt5] * dtime_incremen
   t;
1345.    deluysum [xx5] = deluy [xx5][contactpnt5];
1346.    resultdeluy [xx5] += deluysum [xx5];
1347.
1348.
1349.    delux [xx5][contactpnt5] = udotx [xx5][contactpnt5] * dtime_incremen
   t;
1350.    deluxsum [xx5] = delux [xx5][contactpnt5];
1351.    resultdelux [xx5] += deluxsum [xx5];
1352.
1353.
1354.    deltheta [xx5][contactpnt5] = thetadot [xx5][contactpnt5] * dtime_in
   crement;
1355.    delthetasum [xx5] = deltheta [xx5][contactpnt5];
1356.    resultdeltheta [xx5] += delthetasum [xx5];
1357.
1358.
1359.    uy [xx5][contactpnt5] = deluy [xx5][contactpnt5];
1360.    uysum [xx5] = uy [xx5][contactpnt5];
1361.    resultuy [xx5]+= uysum [xx5];
1362.
1363.
1364.    ux [xx5][contactpnt5] = delux [xx5][contactpnt5];
1365.    uxsum [xx5] = ux [xx5][contactpnt5];
1366.    resultux [xx5] += uxsum [xx5];
```

```
1367.
1368.
1369.    theta [xx5][contactpnt5] = deltheta [xx5][contactpnt5];
1370.    thetasum [xx5] = theta [xx5][contactpnt5];
1371.    resulttheta [xx5] += thetasum [xx5];
1372.    alpha [xx5] = alpha1 + resulttheta [xx5];
1373.
1374.    }
1375.    }
1376.
1377.    }
1378.
1379.    next5:
1380.    ;}
1381.
1382.
1383.    int xx6 = 0, qq6 = 0, contactpnt6 = 0, l6, i6;
1384.
1385.
1386.    for (i6 = 0; i6 < isobox6; i6= i6+1)
1387.
1388.    {
1389.
1390.    xx6 = isocounterbox6[i6];
1391.
1392.    rightisosceles [xx6][0][0] = rightisosceles [xx6][0][0] + resultux [
   xx6] ;
1393.
1394.    rightisosceles [xx6][0][1] = rightisosceles [xx6][0][1] + resultuy [
   xx6] ;
1395.
1396.
1397.
1398.    for ( l6 = 0; l6 < isobox6; l6 = l6+1)
1399.
1400.    {
1401.
1402.
1403.    qq6 = isocounterbox6[l6];
1404.
1405.
1406.
1407.    if ( ((pow((rightisosceles [xx6][0][0] -
    rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) > 0) && ((pow((rightisosceles [xx6][0][0
   ] - rightisosceles [qq6][0][0]), 2)) + (pow ((rightisosceles [xx6][0][1] -
    rightisosceles [qq6][0][1]), 2)) <= 800))
1408.
1409.    {
1410.
1411.    contactpnt6 = contactpnt6 + 1;
1412.
```

```
1413.    if (contactpnt6 >= 4) {goto next6;}
1414.
1415.
1416.    if ( rightisosceles [qq6][3][1] >  rightisosceles [xx6][1][1] )
1417.
1418.    {
1419.
1420.
1421.    ydisplat [xx6][contactpnt6] = ydisplacement2  + (deluy [xx6][contact
   pnt6] -
    deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][0] - rightisosceles [xx6][0][0])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][0] -
    rightisosceles [qq6][0][0]));
1422.    xdisplat [xx6][contactpnt6] = xdisplacement2 + (delux [xx6][contactp
   nt6] -
    delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
    [qq6][3][1] - rightisosceles [xx6][0][1])  -
    deltheta [qq6][contactpnt6] * (rightisosceles [qq6][3][1] -
   rightisosceles [qq6][0][1]));
1423.
1424.
1425.    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
   1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
1426.    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
   1) - xdisplat [xx6][contactpnt6] * sin (alpha1));
1427.
1428.
1429.    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1430.    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1431.
1432.
1433.    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
   1) * (ddamping_coefficient/1000);
1434.    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
   ient/1000);
1435.
1436.
1437.    yforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * sin (alpha1)) -
     ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha1)));
1438.    xforce [qq6][contactpnt6] =   (((fs [xx6][contactpnt6] + ds [xx6][co
   ntactpnt6]) * cos (alpha1)) + ((fn [xx6][contactpnt6] + dn [xx6][contactpn
   t6]) * sin (alpha1)));
1439.
1440.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1441.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1442.
1443.
1444.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
```

```
    y [xx6] = fysum [xx6][contactpnt6];
    resultfy [xx6] += y[xx6];


    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
    x [xx6] = fxsum [xx6][contactpnt6];
    resultfx [xx6] += x[xx6];


    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][3][0] - rightisosceles [xx6][0][0]) -
     xforce [xx6][contactpnt6] * ( rightisosceles [qq6][3][1] -
     rightisosceles [xx6][0][1]));
    m [xx6] = msum [xx6][contactpnt6];
    resultm [xx6] += m[xx6];


    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent) *1000/ dmass); // mm/sec
    udotysum [xx6] = udoty [xx6][contactpnt6];
    resultudoty [xx6] += udotysum [xx6];


    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
    udotxsum [xx6] = udotx [xx6][contactpnt6];
    resultudotx [xx6] += udotxsum [xx6];


    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
    resultthetadot [xx6] += thetadotsum [xx6];


    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
    deluysum [xx6] = deluy [xx6][contactpnt6];
    resultdeluy [xx6] += deluysum [xx6];


    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
    deluxsum [xx6] = delux [xx6][contactpnt6];
    resultdelux [xx6] += deluxsum [xx6];

    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
    delthetasum [xx6] = deltheta [xx6][contactpnt6];
    resultdeltheta [xx6] += delthetasum [xx6];


```

```
    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
    uysum [xx6] = uy [xx6][contactpnt6];
    resultuy [xx6]+= uysum [xx6];


    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
    uxsum [xx6] = ux [xx6][contactpnt6];
    resultux [xx6] += uxsum [xx6];


    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
    thetasum [xx6] = theta [xx6][contactpnt6];
    resulttheta [xx6] += thetasum [xx6];
    alpha [xx6] = alpha1 + resulttheta [xx6];

    }

    else

    {


    ydisplat [xx6][contactpnt6] += ydisplacement2  + (deluy [xx6][contac
tpnt6] -
     deluy [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
     [qq6][1][0] - rightisosceles [xx6][0][0])  -
     deltheta [qq6][contactpnt6] * (rightisosceles [qq6][0][0] -
     rightisosceles [qq6][0][0]));
    xdisplat [xx6][contactpnt6] += xdisplacement2 + (delux [xx6][contact
pnt6] -
     delux [qq6][contactpnt6] + deltheta [xx6][contactpnt6] * ( rightisosceles
     [qq6][1][1] - rightisosceles [xx6][0][1])  -
     deltheta [qq6][contactpnt6] * (rightisosceles [qq6][1][1] -
    rightisosceles [qq6][0][1]));

    delus [xx6][contactpnt6] = (xdisplat [xx6][contactpnt6] * cos (alpha
1) + ydisplat [xx6][contactpnt6] * sin (alpha1));
    delun [xx6][contactpnt6] = (ydisplat [xx6][contactpnt6] * cos (alpha
1) - xdisplat [xx6][contactpnt6] * sin (alpha1));

    fn [xx6][contactpnt6] =  delun [xx6][contactpnt6] * (-
1) *  (dstiffness_coefficient/1000);
    fs [xx6][contactpnt6] =  delus [xx6][contactpnt6] *  (dstiffness_coe
fficient/1000);

    dn [xx6][contactpnt6] = delun [xx6][contactpnt6] * (-
1) * (ddamping_coefficient/1000);
    ds [xx6][contactpnt6] = delus [xx6][contactpnt6] * (ddamping_coeffic
ient/1000);

```

```
1523.    yforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * sin (alpha [xx6])) -
    ((fn [xx6][contactpnt6] + dn [xx6][contactpnt6]) * cos (alpha [xx6])));

1524.    xforce [qq6][contactpnt6] =  (((fs [xx6][contactpnt6] + ds [xx6][con
   tactpnt6]) * cos (alpha [xx6])) + ((fn [xx6][contactpnt6] + dn [xx6][conta
   ctpnt6]) * sin (alpha [xx6])));
1525.
1526.    yforce [xx6][contactpnt6] = (yforce [qq6][contactpnt6] * -1);
1527.    xforce [xx6][contactpnt6] = (xforce [qq6][contactpnt6] * -1);
1528.
1529.    fxsum [xx6][contactpnt6] = xforce [xx6][contactpnt6];
1530.    x [xx6] = fxsum [xx6][contactpnt6];
1531.    resultfx [xx6] += x[xx6];
1532.
1533.
1534.    fysum [xx6][contactpnt6] = yforce [xx6][contactpnt6] + p2 ;
1535.    y [xx6] = fysum [xx6][contactpnt6];
1536.    resultfy [xx6] += y[xx6];
1537.
1538.
1539.    msum [xx6][contactpnt6] = (yforce [xx6][contactpnt6]* ( rightisoscel
   es [qq6][1][0] - rightisosceles [xx6][0][0]) -
    xforce [xx6][contactpnt6] * ( rightisosceles [qq6][1][1] -
    rightisosceles [xx6][0][1]));
1540.    m [xx6] = msum [xx6][contactpnt6];
1541.    resultm [xx6] += m[xx6];
1542.
1543.
1544.    udoty [xx6][contactpnt6]=  ((fysum [xx6][contactpnt6] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1545.    udotysum [xx6] = udoty [xx6][contactpnt6];
1546.    resultudoty [xx6] += udotysum [xx6];
1547.
1548.
1549.    udotx [xx6][contactpnt6] =  ((fxsum [xx6][contactpnt6]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1550.    udotxsum [xx6] = udotx [xx6][contactpnt6];
1551.    resultudotx [xx6] += udotxsum [xx6];
1552.
1553.
1554.    thetadot [xx6][contactpnt6] = ((msum [xx6][contactpnt6] * dtime_incr
   ement)*1000/ ii); //1/s
1555.    thetadotsum [xx6] = thetadot [xx6][contactpnt6];
1556.    resultthetadot [xx6] += thetadotsum [xx6];
1557.
1558.
1559.    deluy [xx6][contactpnt6] = udoty [xx6][contactpnt6] * dtime_incremen
   t;
1560.    deluysum [xx6] = deluy [xx6][contactpnt6];
1561.    resultdeluy [xx6] += deluysum [xx6];
1562.
```

```
1563.
1564.    delux [xx6][contactpnt6] = udotx [xx6][contactpnt6] * dtime_incremen
   t;
1565.    deluxsum [xx6] = delux [xx6][contactpnt6];
1566.    resultdelux [xx6] += deluxsum [xx6];
1567.
1568.
1569.    deltheta [xx6][contactpnt6] = thetadot [xx6][contactpnt6] * dtime_in
   crement;
1570.    delthetasum [xx6] = deltheta [xx6][contactpnt6];
1571.    resultdeltheta [xx6] += delthetasum [xx6];
1572.
1573.
1574.    uy [xx6][contactpnt6] = deluy [xx6][contactpnt6];
1575.    uysum [xx6] = uy [xx6][contactpnt6];
1576.    resultuy [xx6]+= uysum [xx6];
1577.
1578.
1579.    ux [xx6][contactpnt6] = delux [xx6][contactpnt6];
1580.    uxsum [xx6] = ux [xx6][contactpnt6];
1581.    resultux [xx6] += uxsum [xx6];
1582.
1583.
1584.    theta [xx6][contactpnt6] = deltheta [xx6][contactpnt6];
1585.    thetasum [xx6] = theta [xx6][contactpnt6];
1586.    resulttheta [xx6] += thetasum [xx6];
1587.    alpha [xx6] = alpha1 + resulttheta [xx6];
1588.
1589.    }
1590.
1591.    }
1592.
1593.    }
1594.
1595.    next6:
1596.    ;}
1597.
1598.
1599.
1600.    int xx7 = 0, qq7 = 0, contactpnt7 = 0, l7, i7;
1601.
1602.
1603.    for (i7 = 0; i7 < isobox7; i7= i7+1)
1604.
1605.    {
1606.
1607.    xx7 = isocounterbox7[i7];
1608.
1609.    rightisosceles [xx7][0][0] = rightisosceles [xx7][0][0] + resultux [
   xx7] ;
1610.
```

```
    rightisosceles [xx7][0][1] = rightisosceles [xx7][0][1] + resultuy [
   xx7] ;


    for ( l7 = 0; l7 < isobox7; l7 = l7+1)

    {


    qq7 = isocounterbox7[l7];


    if ( ((pow((rightisosceles [xx7][0][0] -
    rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) > 0) && ((pow((rightisosceles [xx7][0][0
   ] - rightisosceles [qq7][0][0]), 2)) + (pow ((rightisosceles [xx7][0][1] -
    rightisosceles [qq7][0][1]), 2)) <= 800))

    {

    contactpnt7 = contactpnt7 + 1;

    if (contactpnt7 >= 4) {goto next7;}


    if ( rightisosceles [qq7][3][1] >  rightisosceles [xx7][1][1] )

    {


    ydisplat [xx7][contactpnt7] = ydisplacement2  + (deluy [xx7][contact
   pnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][0] - rightisosceles [xx7][0][0])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][0] -
    rightisosceles [qq7][0][0]));
    xdisplat [xx7][contactpnt7] = xdisplacement2 + (delux [xx7][contactp
   nt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][3][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][3][1] -
   rightisosceles [qq7][0][1]));


    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));


```

```
1646.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1647.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1648.
1649.
1650.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1651.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1652.
1653.
1654.    yforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * sin (alpha1)) -
     ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha1)));
1655.    xforce [qq7][contactpnt7] =   (((fs [xx7][contactpnt7] + ds [xx7][co
   ntactpnt7]) * cos (alpha1)) + ((fn [xx7][contactpnt7] + dn [xx7][contactpn
   t7]) * sin (alpha1)));
1656.
1657.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1658.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1659.
1660.
1661.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1662.    y [xx7] = fysum [xx7][contactpnt7];
1663.    resultfy [xx7] += y[xx7];
1664.
1665.
1666.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1667.    x [xx7] = fxsum [xx7][contactpnt7];
1668.    resultfx [xx7] += x[xx7];
1669.
1670.
1671.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][3][0] - rightisosceles [xx7][0][0]) -
     xforce [xx7][contactpnt7] * ( rightisosceles [qq7][3][1] -
     rightisosceles [xx7][0][1]));
1672.    m [xx7] = msum [xx7][contactpnt7];
1673.    resultm [xx7] += m[xx7];
1674.
1675.
1676.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1677.    udotysum [xx7] = udoty [xx7][contactpnt7];
1678.    resultudoty [xx7] += udotysum [xx7];
1679.
1680.
1681.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1682.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1683.    resultudotx [xx7] += udotxsum [xx7];
1684.
```

```
1685.
1686.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1687.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1688.    resultthetadot [xx7] += thetadotsum [xx7];
1689.
1690.
1691.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1692.    deluysum [xx7] = deluy [xx7][contactpnt7];
1693.    resultdeluy [xx7] += deluysum [xx7];
1694.
1695.
1696.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1697.    deluxsum [xx7] = delux [xx7][contactpnt7];
1698.    resultdelux [xx7] += deluxsum [xx7];
1699.
1700.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1701.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1702.    resultdeltheta [xx7] += delthetasum [xx7];
1703.
1704.
1705.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1706.    uysum [xx7] = uy [xx7][contactpnt7];
1707.    resultuy [xx7]+= uysum [xx7];
1708.
1709.
1710.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1711.    uxsum [xx7] = ux [xx7][contactpnt7];
1712.    resultux [xx7] += uxsum [xx7];
1713.
1714.
1715.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1716.    thetasum [xx7] = theta [xx7][contactpnt7];
1717.    resultttheta [xx7] += thetasum [xx7];
1718.    alpha [xx7] = alpha1 + resultttheta [xx7];
1719.
1720.    }
1721.
1722.    else
1723.
1724.    {
1725.
1726.
1727.
1728.    ydisplat [xx7][contactpnt7] += ydisplacement2  + (deluy [xx7][contac
   tpnt7] -
    deluy [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][0] - rightisosceles [xx7][0][0])  -
```

```
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][0][0] -
    rightisosceles [qq7][0][0]));
1729.    xdisplat [xx7][contactpnt7] += xdisplacement2 + (delux [xx7][contact
   pnt7] -
    delux [qq7][contactpnt7] + deltheta [xx7][contactpnt7] * ( rightisosceles
    [qq7][1][1] - rightisosceles [xx7][0][1])  -
    deltheta [qq7][contactpnt7] * (rightisosceles [qq7][1][1] -
   rightisosceles [qq7][0][1]));
1730.
1731.    delus [xx7][contactpnt7] = (xdisplat [xx7][contactpnt7] * cos (alpha
   1) + ydisplat [xx7][contactpnt7] * sin (alpha1));
1732.    delun [xx7][contactpnt7] = (ydisplat [xx7][contactpnt7] * cos (alpha
   1) - xdisplat [xx7][contactpnt7] * sin (alpha1));
1733.
1734.    fn [xx7][contactpnt7] =  delun [xx7][contactpnt7] * (-
   1) *  (dstiffness_coefficient/1000);
1735.    fs [xx7][contactpnt7] =  delus [xx7][contactpnt7] *  (dstiffness_coe
   fficient/1000);
1736.
1737.    dn [xx7][contactpnt7] = delun [xx7][contactpnt7] * (-
   1) * (ddamping_coefficient/1000);
1738.    ds [xx7][contactpnt7] = delus [xx7][contactpnt7] * (ddamping_coeffic
   ient/1000);
1739.
1740.    yforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * sin (alpha [xx7])) -
    ((fn [xx7][contactpnt7] + dn [xx7][contactpnt7]) * cos (alpha [xx7])));

1741.    xforce [qq7][contactpnt7] =  (((fs [xx7][contactpnt7] + ds [xx7][con
   tactpnt7]) * cos (alpha [xx7])) + ((fn [xx7][contactpnt7] + dn [xx7][conta
   ctpnt7]) * sin (alpha [xx7])));
1742.
1743.    yforce [xx7][contactpnt7] = (yforce [qq7][contactpnt7] * -1);
1744.    xforce [xx7][contactpnt7] = (xforce [qq7][contactpnt7] * -1);
1745.
1746.    fxsum [xx7][contactpnt7] = xforce [xx7][contactpnt7];
1747.    x [xx7] = fxsum [xx7][contactpnt7];
1748.    resultfx [xx7] += x[xx7];
1749.
1750.
1751.    fysum [xx7][contactpnt7] = yforce [xx7][contactpnt7] + p2 ;
1752.    y [xx7] = fysum [xx7][contactpnt7];
1753.    resultfy [xx7] += y[xx7];
1754.
1755.
1756.    msum [xx7][contactpnt7] = (yforce [xx7][contactpnt7]* ( rightisoscel
   es [qq7][1][0] - rightisosceles [xx7][0][0]) -
    xforce [xx7][contactpnt7] * ( rightisosceles [qq7][1][1] -
    rightisosceles [xx7][0][1]));
1757.    m [xx7] = msum [xx7][contactpnt7];
1758.    resultm [xx7] += m[xx7];
1759.
```

```
1760.
1761.    udoty [xx7][contactpnt7]=  ((fysum [xx7][contactpnt7] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
1762.    udotysum [xx7] = udoty [xx7][contactpnt7];
1763.    resultudoty [xx7] += udotysum [xx7];
1764.
1765.
1766.    udotx [xx7][contactpnt7] =  ((fxsum [xx7][contactpnt7]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1767.    udotxsum [xx7] = udotx [xx7][contactpnt7];
1768.    resultudotx [xx7] += udotxsum [xx7];
1769.
1770.
1771.    thetadot [xx7][contactpnt7] = ((msum [xx7][contactpnt7] * dtime_incr
   ement)*1000/ ii); //1/s
1772.    thetadotsum [xx7] = thetadot [xx7][contactpnt7];
1773.    resultthetadot [xx7] += thetadotsum [xx7];
1774.
1775.
1776.    deluy [xx7][contactpnt7] = udoty [xx7][contactpnt7] * dtime_incremen
   t;
1777.    deluysum [xx7] = deluy [xx7][contactpnt7];
1778.    resultdeluy [xx7] += deluysum [xx7];
1779.
1780.
1781.    delux [xx7][contactpnt7] = udotx [xx7][contactpnt7] * dtime_incremen
   t;
1782.    deluxsum [xx7] = delux [xx7][contactpnt7];
1783.    resultdelux [xx7] += deluxsum [xx7];
1784.
1785.
1786.    deltheta [xx7][contactpnt7] = thetadot [xx7][contactpnt7] * dtime_in
   crement;
1787.    delthetasum [xx7] = deltheta [xx7][contactpnt7];
1788.    resultdeltheta [xx7] += delthetasum [xx7];
1789.
1790.
1791.    uy [xx7][contactpnt7] = deluy [xx7][contactpnt7];
1792.    uysum [xx7] = uy [xx7][contactpnt7];
1793.    resultuy [xx7]+= uysum [xx7];
1794.
1795.
1796.    ux [xx7][contactpnt7] = delux [xx7][contactpnt7];
1797.    uxsum [xx7] = ux [xx7][contactpnt7];
1798.    resultux [xx7] += uxsum [xx7];
1799.
1800.
1801.    theta [xx7][contactpnt7] = deltheta [xx7][contactpnt7];
1802.    thetasum [xx7] = theta [xx7][contactpnt7];
1803.    resulttheta [xx7] += thetasum [xx7];
1804.    alpha [xx7] = alpha1 + resulttheta [xx7];
1805.    }
```

```
1806.
1807.    }
1808.
1809.    }
1810.
1811.    next7:
1812.    ;}
1813.
1814.
1815.
1816.    int xx8 = 0, qq8 = 0, contactpnt8 = 0, l8, i8;
1817.
1818.
1819.    for (i8 = 0; i8 < isobox8; i8= i8+1)
1820.
1821.    {
1822.
1823.    xx8 = isocounterbox8[i8];
1824.
1825.    rightisosceles [xx8][0][0] = rightisosceles [xx8][0][0] + resultux [
   xx8] ;
1826.
1827.    rightisosceles [xx8][0][1] = rightisosceles [xx8][0][1] + resultuy [
   xx8] ;
1828.
1829.
1830.
1831.    for ( l8 = 0; l8 < isobox8; l8 = l8+1)
1832.
1833.    {
1834.
1835.
1836.    qq8 = isocounterbox8[l8];
1837.
1838.
1839.
1840.    if ( ((pow((rightisosceles [xx8][0][0] -
    rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) > 0) && ((pow((rightisosceles [xx8][0][0
   ] - rightisosceles [qq8][0][0]), 2)) + (pow ((rightisosceles [xx8][0][1] -
    rightisosceles [qq8][0][1]), 2)) <= 800))
1841.
1842.    {
1843.
1844.    contactpnt8 = contactpnt8 + 1;
1845.
1846.    if (contactpnt8 >= 4) {goto next8;}
1847.
1848.
1849.    if ( rightisosceles [qq8][3][1] >  rightisosceles [xx8][1][1] )
1850.
1851.    {
```

```
1852.
1853.
1854.    ydisplat [xx8][contactpnt8] = ydisplacement2  + (deluy [xx8][contact
   pnt8] -
    deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][0] - rightisosceles [xx8][0][0])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][0] -
    rightisosceles [qq8][0][0]));
1855.    xdisplat [xx8][contactpnt8] = xdisplacement2 + (delux [xx8][contactp
   nt8] -
    delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
    [qq8][3][1] - rightisosceles [xx8][0][1])  -
    deltheta [qq8][contactpnt8] * (rightisosceles [qq8][3][1] -
   rightisosceles [qq8][0][1]));
1856.
1857.
1858.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1859.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1860.
1861.
1862.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt8] * (-
   1) *  (dstiffness_coefficient/1000);
1863.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt8] *  (dstiffness_coe
   fficient/1000);
1864.
1865.
1866.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1867.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1868.
1869.
1870.    yforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * sin (alpha1)) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha1)));
1871.    xforce [qq8][contactpnt8] =   (((fs [xx8][contactpnt8] + ds [xx8][co
   ntactpnt8]) * cos (alpha1)) + ((fn [xx8][contactpnt8] + dn [xx8][contactpn
   t8]) * sin (alpha1)));
1872.
1873.    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
1874.    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);
1875.
1876.
1877.    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
1878.    y [xx8] = fysum [xx8][contactpnt8];
1879.    resultfy [xx8] += y[xx8];
1880.
1881.
1882.    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
1883.    x [xx8] = fxsum [xx8][contactpnt8];
```

```
1884.    resultfx [xx8] += x[xx8];
1885.
1886.
1887.    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
   es [qq8][3][0] - rightisosceles [xx8][0][0]) -
    xforce [xx8][contactpnt8] * ( rightisosceles [qq8][3][1] -
    rightisosceles [xx8][0][1]));
1888.    m [xx8] = msum [xx8][contactpnt8];
1889.    resultm [xx8] += m[xx8];
1890.
1891.
1892.    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
   ent) *1000/ dmass); // mm/sec
1893.    udotysum [xx8] = udoty [xx8][contactpnt8];
1894.    resultudoty [xx8] += udotysum [xx8];
1895.
1896.
1897.    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
   ent)*1000/ dmass); //mm/sec
1898.    udotxsum [xx8] = udotx [xx8][contactpnt8];
1899.    resultudotx [xx8] += udotxsum [xx8];
1900.
1901.
1902.    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
   ement)*1000/ ii); //1/s
1903.    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
1904.    resultthetadot [xx8] += thetadotsum [xx8];
1905.
1906.
1907.    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
   t;
1908.    deluysum [xx8] = deluy [xx8][contactpnt8];
1909.    resultdeluy [xx8] += deluysum [xx8];
1910.
1911.
1912.    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
   t;
1913.    deluxsum [xx8] = delux [xx8][contactpnt8];
1914.    resultdelux [xx8] += deluxsum [xx8];
1915.
1916.    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
   crement;
1917.    delthetasum [xx8] = deltheta [xx8][contactpnt8];
1918.    resultdeltheta [xx8] += delthetasum [xx8];
1919.
1920.
1921.    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
1922.    uysum [xx8] = uy [xx8][contactpnt8];
1923.    resultuy [xx8]+= uysum [xx8];
1924.
1925.
1926.    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
```

```
1927.    uxsum [xx8] = ux [xx8][contactpnt8];
1928.    resultux [xx8] += uxsum [xx8];
1929.
1930.
1931.    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
1932.    thetasum [xx8] = theta [xx8][contactpnt8];
1933.    resulttheta [xx8] += thetasum [xx8];
1934.    alpha [xx8] = alpha1 + resulttheta [xx8];
1935.
1936.    }
1937.
1938.    else
1939.
1940.    {
1941.
1942.
1943.    ydisplat [xx8][contactpnt8] += ydisplacement2  + (deluy [xx8][contac
   tpnt8] -
     deluy [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][0] - rightisosceles [xx8][0][0])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][0][0] -
     rightisosceles [qq8][0][0]));
1944.    xdisplat [xx8][contactpnt8] += xdisplacement2 + (delux [xx8][contact
   pnt8] -
     delux [qq8][contactpnt8] + deltheta [xx8][contactpnt8] * ( rightisosceles
     [qq8][1][1] - rightisosceles [xx8][0][1])  -
     deltheta [qq8][contactpnt8] * (rightisosceles [qq8][1][1] -
   rightisosceles [qq8][0][1]));
1945.
1946.    delus [xx8][contactpnt8] = (xdisplat [xx8][contactpnt8] * cos (alpha
   1) + ydisplat [xx8][contactpnt8] * sin (alpha1));
1947.    delun [xx8][contactpnt8] = (ydisplat [xx8][contactpnt8] * cos (alpha
   1) - xdisplat [xx8][contactpnt8] * sin (alpha1));
1948.
1949.    fn [xx8][contactpnt8] =  delun [xx8][contactpnt6] * (-
   1) *  (dstiffness_coefficient/1000);
1950.    fs [xx8][contactpnt8] =  delus [xx8][contactpnt6] *  (dstiffness_coe
   fficient/1000);
1951.
1952.    dn [xx8][contactpnt8] = delun [xx8][contactpnt8] * (-
   1) * (ddamping_coefficient/1000);
1953.    ds [xx8][contactpnt8] = delus [xx8][contactpnt8] * (ddamping_coeffic
   ient/1000);
1954.
1955.    yforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * sin (alpha [xx8])) -
     ((fn [xx8][contactpnt8] + dn [xx8][contactpnt8]) * cos (alpha [xx8])));

1956.    xforce [qq8][contactpnt8] =  (((fs [xx8][contactpnt8] + ds [xx8][con
   tactpnt8]) * cos (alpha [xx8])) + ((fn [xx8][contactpnt8] + dn [xx8][conta
   ctpnt8]) * sin (alpha [xx8])));
1957.
```

```
    yforce [xx8][contactpnt8] = (yforce [qq8][contactpnt8] * -1);
    xforce [xx8][contactpnt8] = (xforce [qq8][contactpnt8] * -1);

    fxsum [xx8][contactpnt8] = xforce [xx8][contactpnt8];
    x [xx8] = fxsum [xx8][contactpnt8];
    resultfx [xx8] += x[xx8];


    fysum [xx8][contactpnt8] = yforce [xx8][contactpnt8] + p2 ;
    y [xx8] = fysum [xx8][contactpnt8];
    resultfy [xx8] += y[xx8];


    msum [xx8][contactpnt8] = (yforce [xx8][contactpnt8]* ( rightisoscel
es [qq8][1][0] - rightisosceles [xx8][0][0]) -
     xforce [xx8][contactpnt8] * ( rightisosceles [qq8][1][1] -
     rightisosceles [xx8][0][1]));
    m [xx8] = msum [xx8][contactpnt8];
    resultm [xx8] += m[xx8];


    udoty [xx8][contactpnt8]=  ((fysum [xx8][contactpnt8] * dtime_increm
ent)* 1000/ dmass); // mm/sec
    udotysum [xx8] = udoty [xx8][contactpnt8];
    resultudoty [xx8] += udotysum [xx8];


    udotx [xx8][contactpnt8] =  ((fxsum [xx8][contactpnt8]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx8] = udotx [xx8][contactpnt8];
    resultudotx [xx8] += udotxsum [xx8];


    thetadot [xx8][contactpnt8] = ((msum [xx8][contactpnt8] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx8] = thetadot [xx8][contactpnt8];
    resultthetadot [xx8] += thetadotsum [xx8];


    deluy [xx8][contactpnt8] = udoty [xx8][contactpnt8] * dtime_incremen
t;
    deluysum [xx8] = deluy [xx8][contactpnt8];
    resultdeluy [xx8] += deluysum [xx8];


    delux [xx8][contactpnt8] = udotx [xx8][contactpnt8] * dtime_incremen
t;
    deluxsum [xx8] = delux [xx8][contactpnt8];
    resultdelux [xx8] += deluxsum [xx8];


```

```
    deltheta [xx8][contactpnt8] = thetadot [xx8][contactpnt8] * dtime_in
crement;
    delthetasum [xx8] = deltheta [xx8][contactpnt8];
    resultdeltheta [xx8] += delthetasum [xx8];


    uy [xx8][contactpnt8] = deluy [xx8][contactpnt8];
    uysum [xx8] = uy [xx8][contactpnt8];
    resultuy [xx8]+= uysum [xx8];


    ux [xx8][contactpnt8] = delux [xx8][contactpnt8];
    uxsum [xx8] = ux [xx8][contactpnt8];
    resultux [xx8] += uxsum [xx8];


    theta [xx8][contactpnt8] = deltheta [xx8][contactpnt8];
    thetasum [xx8] = theta [xx8][contactpnt8];
    resulttheta [xx8] += thetasum [xx8];
    alpha [xx8] = alpha1 + resulttheta [xx8];


    }
    }

    }

    next8:
    ;}


    int xx9 = 0, qq9 = 0, contactpnt9 = 0, l9, i9;


    for (i9 = 0; i9 < isobox9; i9= i9+1)

    {

    xx9 = isocounterbox9[i9];

    rightisosceles [xx9][0][0] = rightisosceles [xx9][0][0] + resultux [
xx9] ;

    rightisosceles [xx9][0][1] = rightisosceles [xx9][0][1] + resultuy [
xx9] ;


    for ( l9 = 0; l9 < isobox9; l9 = l9+1)

    {
```

```
2050.
2051.
2052.    qq9 = isocounterbox9[19];
2053.
2054.
2055.
2056.    if ( ((pow((rightisosceles [xx9][0][0] -
    rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) > 0) && ((pow((rightisosceles [xx9][0][0
   ] - rightisosceles [qq9][0][0]), 2)) + (pow ((rightisosceles [xx9][0][1] -
    rightisosceles [qq9][0][1]), 2)) <= 800))
2057.
2058.    {
2059.
2060.    contactpnt9 = contactpnt9 + 1;
2061.
2062.    if (contactpnt9 >= 4) {goto next9;}
2063.
2064.
2065.    if ( rightisosceles [qq9][3][1] >  rightisosceles [xx9][1][1] )
2066.
2067.    {
2068.
2069.
2070.    ydisplat [xx9][contactpnt9] = ydisplacement2  + (deluy [xx9][contact
   pnt9] -
    deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][0] - rightisosceles [xx9][0][0])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][0] -
    rightisosceles [qq9][0][0]));
2071.    xdisplat [xx9][contactpnt9] = xdisplacement2 + (delux [xx9][contactp
   nt9] -
    delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
    [qq9][3][1] - rightisosceles [xx9][0][1])  -
    deltheta [qq9][contactpnt9] * (rightisosceles [qq9][3][1] -
   rightisosceles [qq9][0][1]));
2072.
2073.
2074.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2075.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2076.
2077.
2078.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2079.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2080.
2081.
2082.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
```

```
    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
ient/1000);


    yforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * sin (alpha1)) -
     ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha1)));
    xforce [qq9][contactpnt9] =   (((fs [xx9][contactpnt9] + ds [xx9][co
ntactpnt9]) * cos (alpha1)) + ((fn [xx9][contactpnt9] + dn [xx9][contactpn
t9]) * sin (alpha1)));

    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);


    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
    y [xx9] = fysum [xx9][contactpnt9];
    resultfy [xx9] += y[xx9];


    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
    x [xx9] = fxsum [xx9][contactpnt9];
    resultfx [xx9] += x[xx9];


    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
es [qq9][3][0] - rightisosceles [xx9][0][0]) -
     xforce [xx9][contactpnt9] * ( rightisosceles [qq9][3][1] -
     rightisosceles [xx9][0][1]));
    m [xx9] = msum [xx9][contactpnt9];
    resultm [xx9] += m[xx9];


    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
ent) *1000/ dmass); // mm/sec
    udotysum [xx9] = udoty [xx9][contactpnt9];
    resultudoty [xx9] += udotysum [xx9];


    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
ent)*1000/ dmass); //mm/sec
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


```

```
2123.    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
   t;
2124.    deluysum [xx9] = deluy [xx9][contactpnt9];
2125.    resultdeluy [xx9] += deluysum [xx9];
2126.
2127.
2128.    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
   t;
2129.    deluxsum [xx9] = delux [xx9][contactpnt9];
2130.    resultdelux [xx9] += deluxsum [xx9];
2131.
2132.    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
   crement;
2133.    delthetasum [xx9] = deltheta [xx9][contactpnt9];
2134.    resultdeltheta [xx9] += delthetasum [xx9];
2135.
2136.
2137.    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
2138.    uysum [xx9] = uy [xx9][contactpnt9];
2139.    resultuy [xx9]+= uysum [xx9];
2140.
2141.
2142.    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
2143.    uxsum [xx9] = ux [xx9][contactpnt9];
2144.    resultux [xx9] += uxsum [xx9];
2145.
2146.
2147.    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
2148.    thetasum [xx9] = theta [xx9][contactpnt9];
2149.    resulttheta [xx9] += thetasum [xx9];
2150.    alpha [xx9] = alpha1 + resulttheta [xx9];
2151.
2152.    }
2153.
2154.    else
2155.
2156.    {
2157.
2158.
2159.    ydisplat [xx9][contactpnt9] += ydisplacement2  + (deluy [xx9][contac
   tpnt9] -
     deluy [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][0] - rightisosceles [xx9][0][0])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][0][0] -
     rightisosceles [qq9][0][0]));
2160.    xdisplat [xx9][contactpnt9] += xdisplacement2 + (delux [xx9][contact
   pnt9] -
     delux [qq9][contactpnt9] + deltheta [xx9][contactpnt9] * ( rightisosceles
     [qq9][1][1] - rightisosceles [xx9][0][1])  -
     deltheta [qq9][contactpnt9] * (rightisosceles [qq9][1][1] -
   rightisosceles [qq9][0][1]));
2161.
```

```
2162.    delus [xx9][contactpnt9] = (xdisplat [xx9][contactpnt9] * cos (alpha
   1) + ydisplat [xx9][contactpnt9] * sin (alpha1));
2163.    delun [xx9][contactpnt9] = (ydisplat [xx9][contactpnt9] * cos (alpha
   1) - xdisplat [xx9][contactpnt9] * sin (alpha1));
2164.
2165.    fn [xx9][contactpnt9] =  delun [xx9][contactpnt9] * (-
   1) *  (dstiffness_coefficient/1000);
2166.    fs [xx9][contactpnt9] =  delus [xx9][contactpnt9] *  (dstiffness_coe
   fficient/1000);
2167.
2168.    dn [xx9][contactpnt9] = delun [xx9][contactpnt9] * (-
   1) * (ddamping_coefficient/1000);
2169.    ds [xx9][contactpnt9] = delus [xx9][contactpnt9] * (ddamping_coeffic
   ient/1000);
2170.
2171.    yforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * sin (alpha [xx9])) -
    ((fn [xx9][contactpnt9] + dn [xx9][contactpnt9]) * cos (alpha [xx9])));

2172.    xforce [qq9][contactpnt9] =  (((fs [xx9][contactpnt9] + ds [xx9][con
   tactpnt9]) * cos (alpha [xx9])) + ((fn [xx9][contactpnt9] + dn [xx9][conta
   ctpnt9]) * sin (alpha [xx9])));
2173.
2174.    yforce [xx9][contactpnt9] = (yforce [qq9][contactpnt9] * -1);
2175.    xforce [xx9][contactpnt9] = (xforce [qq9][contactpnt9] * -1);
2176.
2177.    fxsum [xx9][contactpnt9] = xforce [xx9][contactpnt9];
2178.    x [xx9] = fxsum [xx9][contactpnt9];
2179.    resultfx [xx9] += x[xx9];
2180.
2181.
2182.    fysum [xx9][contactpnt9] = yforce [xx9][contactpnt9] + p2 ;
2183.    y [xx9] = fysum [xx9][contactpnt9];
2184.    resultfy [xx9] += y[xx9];
2185.
2186.
2187.    msum [xx9][contactpnt9] = (yforce [xx9][contactpnt9]* ( rightisoscel
   es [qq9][1][0] - rightisosceles [xx9][0][0]) -
    xforce [xx9][contactpnt9] * ( rightisosceles [qq9][1][1] -
    rightisosceles [xx9][0][1]));
2188.    m [xx9] = msum [xx9][contactpnt9];
2189.    resultm [xx9] += m[xx9];
2190.
2191.
2192.    udoty [xx9][contactpnt9]=  ((fysum [xx9][contactpnt9] * dtime_increm
   ent)* 1000/ dmass); // mm/sec
2193.    udotysum [xx9] = udoty [xx9][contactpnt9];
2194.    resultudoty [xx9] += udotysum [xx9];
2195.
2196.
2197.    udotx [xx9][contactpnt9] =  ((fxsum [xx9][contactpnt9]* dtime_increm
   ent)*1000/ dmass); //mm/sec
```

```
    udotxsum [xx9] = udotx [xx9][contactpnt9];
    resultudotx [xx9] += udotxsum [xx9];


    thetadot [xx9][contactpnt9] = ((msum [xx9][contactpnt9] * dtime_incr
ement)*1000/ ii); //1/s
    thetadotsum [xx9] = thetadot [xx9][contactpnt9];
    resultthetadot [xx9] += thetadotsum [xx9];


    deluy [xx9][contactpnt9] = udoty [xx9][contactpnt9] * dtime_incremen
t;
    deluysum [xx9] = deluy [xx9][contactpnt9];
    resultdeluy [xx9] += deluysum [xx9];


    delux [xx9][contactpnt9] = udotx [xx9][contactpnt9] * dtime_incremen
t;
    deluxsum [xx9] = delux [xx9][contactpnt9];
    resultdelux [xx9] += deluxsum [xx9];


    deltheta [xx9][contactpnt9] = thetadot [xx9][contactpnt9] * dtime_in
crement;
    delthetasum [xx9] = deltheta [xx9][contactpnt9];
    resultdeltheta [xx9] += delthetasum [xx9];


    uy [xx9][contactpnt9] = deluy [xx9][contactpnt9];
    uysum [xx9] = uy [xx9][contactpnt9];
    resultuy [xx9]+= uysum [xx9];


    ux [xx9][contactpnt9] = delux [xx9][contactpnt9];
    uxsum [xx9] = ux [xx9][contactpnt9];
    resultux [xx9] += uxsum [xx9];


    theta [xx9][contactpnt9] = deltheta [xx9][contactpnt9];
    thetasum [xx9] = theta [xx9][contactpnt9];
    resulttheta [xx9] += thetasum [xx9];
    alpha [xx9] = alpha1 + resulttheta [xx9];

    }
    }

    }

    next9:
    ;}

```

```
2246.
2247.    int xx10 = 0, qq10 = 0, contactpnt10 = 0, l10, i10;
2248.
2249.
2250.    for (i10 = 0; i10 < isobox10; i10= i10+1)
2251.
2252.    {
2253.
2254.    xx10 = isocounterbox10[i10];
2255.
2256.    rightisosceles [xx10][0][0] = rightisosceles [xx10][0][0] + resultux
    [xx10] ;
2257.
2258.    rightisosceles [xx10][0][1] = rightisosceles [xx10][0][1] + resultuy
    [xx10] ;
2259.
2260.
2261.
2262.    for ( l10 = 0; l10 < isobox10; l10 = l10+1)
2263.
2264.    {
2265.
2266.
2267.    qq10 = isocounterbox10[l10];
2268.
2269.
2270.
2271.    if ( ((pow((rightisosceles [xx10][0][0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) > 0) && ((pow((rightisosceles [xx10][0]
   [0] -
    rightisosceles [qq10][0][0]), 2)) + (pow ((rightisosceles [xx10][0][1] -
    rightisosceles [qq10][0][1]), 2)) <= 800))
2272.
2273.    {
2274.
2275.    contactpnt10 = contactpnt10 + 1;
2276.
2277.    if (contactpnt10 >= 4) {goto next10;}
2278.
2279.
2280.    if ( rightisosceles [qq10][3][1] >  rightisosceles [xx10][1][1] )
2281.
2282.    {
2283.
2284.
2285.    ydisplat [xx10][contactpnt10] = ydisplacement2  + (deluy [xx10][cont
   actpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][0] -
    rightisosceles [qq10][0][0]));
```

```
2286.    xdisplat [xx10][contactpnt10] = xdisplacement2 + (delux [xx10][conta
   ctpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][3][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][3][1] -
   rightisosceles [qq10][0][1]));
2287.
2288.
2289.    delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2290.    delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2291.
2292.
2293.    fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2294.    fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2295.
2296.
2297.    dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2298.    ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2299.
2300.
2301.    yforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * sin (alpha1)) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha1)));
2302.    xforce [qq10][contactpnt10] =   (((fs [xx10][contactpnt10] + ds [xx1
   0][contactpnt10]) * cos (alpha1)) + ((fn [xx10][contactpnt10] + dn [xx10][
   contactpnt10]) * sin (alpha1)));
2303.
2304.    yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2305.    xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2306.
2307.
2308.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2309.    y [xx10] = fysum [xx10][contactpnt10];
2310.    resultfy [xx10] += y[xx10];
2311.
2312.
2313.    fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2314.    x [xx10] = fxsum [xx10][contactpnt10];
2315.    resultfx [xx10] += x[xx10];
2316.
2317.
2318.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][3][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][3][1] -
    rightisosceles [xx10][0][1]));
2319.    m [xx10] = msum [xx10][contactpnt10];
```

```
2320.    resultm [xx10] += m[xx10];
2321.
2322.
2323.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement) *1000/ dmass); // mm/sec
2324.    udotysum [xx10] = udoty [xx10][contactpnt10];
2325.    resultudoty [xx10] += udotysum [xx10];
2326.
2327.
2328.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2329.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2330.    resultudotx [xx10] += udotxsum [xx10];
2331.
2332.
2333.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2334.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2335.    resultthetadot [xx10] += thetadotsum [xx10];
2336.
2337.
2338.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2339.    deluysum [xx10] = deluy [xx10][contactpnt10];
2340.    resultdeluy [xx10] += deluysum [xx10];
2341.
2342.
2343.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2344.    deluxsum [xx10] = delux [xx10][contactpnt10];
2345.    resultdelux [xx10] += deluxsum [xx10];
2346.
2347.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2348.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2349.    resultdeltheta [xx10] += delthetasum [xx10];
2350.
2351.
2352.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2353.    uysum [xx10] = uy [xx10][contactpnt10];
2354.    resultuy [xx10]+= uysum [xx10];
2355.
2356.
2357.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2358.    uxsum [xx10] = ux [xx10][contactpnt10];
2359.    resultux [xx10] += uxsum [xx10];
2360.
2361.
2362.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2363.    thetasum [xx10] = theta [xx10][contactpnt10];
2364.    resultthetа [xx10] += thetasum [xx10];
2365.    alpha [xx10] = alpha1 + resultthetа [xx10];
```

```
2366.
2367.     }
2368.
2369.     else
2370.
2371.     {
2372.
2373.
2374.     ydisplat [xx10][contactpnt10] += ydisplacement2  + (deluy [xx10][con
   tactpnt10] -
    deluy [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][0] - rightisosceles [xx10][0][0])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][0][0] -
    rightisosceles [qq10][0][0]));
2375.     xdisplat [xx10][contactpnt10] += xdisplacement2 + (delux [xx10][cont
   actpnt10] -
    delux [qq10][contactpnt10] + deltheta [xx10][contactpnt10] * ( rightisosc
   eles [qq10][1][1] - rightisosceles [xx10][0][1])  -
    deltheta [qq10][contactpnt10] * (rightisosceles [qq10][1][1] -
   rightisosceles [qq10][0][1]));
2376.
2377.     delus [xx10][contactpnt10] = (xdisplat [xx10][contactpnt10] * cos (a
   lpha1) + ydisplat [xx10][contactpnt10] * sin (alpha1));
2378.     delun [xx10][contactpnt10] = (ydisplat [xx10][contactpnt10] * cos (a
   lpha1) - xdisplat [xx10][contactpnt10] * sin (alpha1));
2379.
2380.     fn [xx10][contactpnt10] =  delun [xx10][contactpnt10] * (-
   1) *  (dstiffness_coefficient/1000);
2381.     fs [xx10][contactpnt10] =  delus [xx10][contactpnt10] *  (dstiffness
   _coefficient/1000);
2382.
2383.     dn [xx10][contactpnt10] = delun [xx10][contactpnt10] * (-
   1) * (ddamping_coefficient/1000);
2384.     ds [xx10][contactpnt10] = delus [xx10][contactpnt10] * (ddamping_coe
   fficient/1000);
2385.
2386.     yforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * sin (alpha [xx10])) -
    ((fn [xx10][contactpnt10] + dn [xx10][contactpnt10]) * cos (alpha [xx10])
   ));
2387.     xforce [qq10][contactpnt10] =  (((fs [xx10][contactpnt10] + ds [xx10
   ][contactpnt10]) * cos (alpha [xx10])) + ((fn [xx10][contactpnt10] + dn [x
   x10][contactpnt10]) * sin (alpha [xx10])));
2388.
2389.     yforce [xx10][contactpnt10] = (yforce [qq10][contactpnt10] * -1);
2390.     xforce [xx10][contactpnt10] = (xforce [qq10][contactpnt10] * -1);
2391.
2392.     fxsum [xx10][contactpnt10] = xforce [xx10][contactpnt10];
2393.     x [xx10] = fxsum [xx10][contactpnt10];
2394.     resultfx [xx10] += x[xx10];
2395.
2396.
```

```
2397.    fysum [xx10][contactpnt10] = yforce [xx10][contactpnt10] + p2 ;
2398.    y [xx10] = fysum [xx10][contactpnt10];
2399.    resultfy [xx10] += y[xx10];
2400.
2401.
2402.    msum [xx10][contactpnt10] = (yforce [xx10][contactpnt10]* ( rightiso
   sceles [qq10][1][0] - rightisosceles [xx10][0][0]) -
    xforce [xx10][contactpnt10] * ( rightisosceles [qq10][1][1] -
    rightisosceles [xx10][0][1]));
2403.    m [xx10] = msum [xx10][contactpnt10];
2404.    resultm [xx10] += m[xx10];
2405.
2406.
2407.    udoty [xx10][contactpnt10]=  ((fysum [xx10][contactpnt10] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2408.    udotysum [xx10] = udoty [xx10][contactpnt10];
2409.    resultudoty [xx10] += udotysum [xx10];
2410.
2411.
2412.    udotx [xx10][contactpnt10] =  ((fxsum [xx10][contactpnt10]* dtime_in
   crement)*1000/ dmass); //mm/sec
2413.    udotxsum [xx10] = udotx [xx10][contactpnt10];
2414.    resultudotx [xx10] += udotxsum [xx10];
2415.
2416.
2417.    thetadot [xx10][contactpnt10] = ((msum [xx10][contactpnt10] * dtime_
   increment)*1000/ ii); //1/s
2418.    thetadotsum [xx10] = thetadot [xx10][contactpnt10];
2419.    resultthetadot [xx10] += thetadotsum [xx10];
2420.
2421.
2422.    deluy [xx10][contactpnt10] = udoty [xx10][contactpnt10] * dtime_incr
   ement;
2423.    deluysum [xx10] = deluy [xx10][contactpnt10];
2424.    resultdeluy [xx10] += deluysum [xx10];
2425.
2426.
2427.    delux [xx10][contactpnt10] = udotx [xx10][contactpnt10] * dtime_incr
   ement;
2428.    deluxsum [xx10] = delux [xx10][contactpnt10];
2429.    resultdelux [xx10] += deluxsum [xx10];
2430.
2431.
2432.    deltheta [xx10][contactpnt10] = thetadot [xx10][contactpnt10] * dtim
   e_increment;
2433.    delthetasum [xx10] = deltheta [xx10][contactpnt10];
2434.    resultdeltheta [xx10] += delthetasum [xx10];
2435.
2436.
2437.    uy [xx10][contactpnt10] = deluy [xx10][contactpnt10];
2438.    uysum [xx10] = uy [xx10][contactpnt10];
2439.    resultuy [xx10]+= uysum [xx10];
```

```
2440.
2441.
2442.    ux [xx10][contactpnt10] = delux [xx10][contactpnt10];
2443.    uxsum [xx10] = ux [xx10][contactpnt10];
2444.    resultux [xx10] += uxsum [xx10];
2445.
2446.
2447.    theta [xx10][contactpnt10] = deltheta [xx10][contactpnt10];
2448.    thetasum [xx10] = theta [xx10][contactpnt10];
2449.    resulttheta [xx10] += thetasum [xx10];
2450.    alpha [xx10] = alpha1 + resulttheta [xx10];
2451.
2452.    }
2453.    }
2454.
2455.    }
2456.
2457.    next10:
2458.    ;}
2459.
2460.
2461.    int xx11 = 0, qq11 = 0, contactpnt11 = 0, l11, i11;
2462.
2463.
2464.    for (i11 = 0; i11 < isobox11; i11= i11+1)
2465.
2466.    {
2467.
2468.    xx11 = isocounterbox11[i11];
2469.
2470.    rightisosceles [xx11][0][0] = rightisosceles [xx11][0][0] + resultux
    [xx11] ;
2471.
2472.    rightisosceles [xx11][0][1] = rightisosceles [xx11][0][1] + resultuy
    [xx11] ;
2473.
2474.
2475.
2476.    for ( l11 = 0; l11 < isobox11; l11 = l11+1)
2477.
2478.    {
2479.
2480.
2481.    qq11 = isocounterbox11[l11];
2482.
2483.
2484.
2485.    if ( ((pow((rightisosceles [xx11][0][0] -
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) > 0) && ((pow((rightisosceles [xx11][0]
   [0] -
```

```
    rightisosceles [qq11][0][0]), 2)) + (pow ((rightisosceles [xx11][0][1] -
    rightisosceles [qq11][0][1]), 2)) <= 800))
2486.
2487.    {
2488.
2489.    contactpnt11 = contactpnt11 + 1;
2490.
2491.    if (contactpnt11 >= 4) {goto next11;}
2492.
2493.
2494.    if ( rightisosceles [qq11][3][1] >  rightisosceles [xx11][1][1] )
2495.
2496.    {
2497.
2498.
2499.    ydisplat [xx11][contactpnt11] = ydisplacement2  + (deluy [xx11][cont
   actpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][0] -
     rightisosceles [qq11][0][0]));
2500.    xdisplat [xx11][contactpnt11] = xdisplacement2 + (delux [xx11][conta
   ctpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][3][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][3][1] -
   rightisosceles [qq11][0][1]));
2501.
2502.
2503.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2504.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2505.
2506.
2507.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
2508.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2509.
2510.
2511.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2512.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2513.
2514.
2515.    yforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * sin (alpha1)) -
     ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha1)));
```

```
2516.    xforce [qq11][contactpnt11] =   (((fs [xx11][contactpnt11] + ds [xx1
   1][contactpnt11]) * cos (alpha1)) + ((fn [xx11][contactpnt11] + dn [xx11][
   contactpnt11]) * sin (alpha1)));
2517.
2518.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2519.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2520.
2521.
2522.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2523.    y [xx11] = fysum [xx11][contactpnt11];
2524.    resultfy [xx11] += y[xx11];
2525.
2526.
2527.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2528.    x [xx11] = fxsum [xx11][contactpnt11];
2529.    resultfx [xx11] += x[xx11];
2530.
2531.
2532.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][3][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][3][1] -
    rightisosceles [xx11][0][1]));
2533.    m [xx11] = msum [xx11][contactpnt11];
2534.    resultm [xx11] += m[xx11];
2535.
2536.
2537.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement) *1000/ dmass); // mm/sec
2538.    udotysum [xx11] = udoty [xx11][contactpnt11];
2539.    resultudoty [xx11] += udotysum [xx11];
2540.
2541.
2542.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2543.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2544.    resultudotx [xx11] += udotxsum [xx11];
2545.
2546.
2547.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2548.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
2549.    resultthetadot [xx11] += thetadotsum [xx11];
2550.
2551.
2552.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2553.    deluysum [xx11] = deluy [xx11][contactpnt11];
2554.    resultdeluy [xx11] += deluysum [xx11];
2555.
2556.
2557.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
```

```
2558.    deluxsum [xx11] = delux [xx11][contactpnt11];
2559.    resultdelux [xx11] += deluxsum [xx11];
2560.
2561.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2562.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2563.    resultdeltheta [xx11] += delthetasum [xx11];
2564.
2565.
2566.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2567.    uysum [xx11] = uy [xx11][contactpnt11];
2568.    resultuy [xx11]+= uysum [xx11];
2569.
2570.
2571.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2572.    uxsum [xx11] = ux [xx11][contactpnt11];
2573.    resultux [xx11] += uxsum [xx11];
2574.
2575.
2576.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2577.    thetasum [xx11] = theta [xx11][contactpnt11];
2578.    resulttheta [xx11] += thetasum [xx11];
2579.    alpha [xx11] = alpha1 + resulttheta [xx11];
2580.
2581.    }
2582.
2583.    else
2584.
2585.    {
2586.
2587.
2588.
2589.    ydisplat [xx11][contactpnt11] += ydisplacement2  + (deluy [xx11][con
   tactpnt11] -
     deluy [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][0] - rightisosceles [xx11][0][0])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][0][0] -
     rightisosceles [qq11][0][0]));
2590.    xdisplat [xx11][contactpnt11] += xdisplacement2 + (delux [xx11][cont
   actpnt11] -
     delux [qq11][contactpnt11] + deltheta [xx11][contactpnt11] * ( rightisosc
   eles [qq11][1][1] - rightisosceles [xx11][0][1])  -
     deltheta [qq11][contactpnt11] * (rightisosceles [qq11][1][1] -
   rightisosceles [qq11][0][1]));
2591.
2592.    delus [xx11][contactpnt11] = (xdisplat [xx11][contactpnt11] * cos (a
   lpha1) + ydisplat [xx11][contactpnt11] * sin (alpha1));
2593.    delun [xx11][contactpnt11] = (ydisplat [xx11][contactpnt11] * cos (a
   lpha1) - xdisplat [xx11][contactpnt11] * sin (alpha1));
2594.
2595.    fn [xx11][contactpnt11] =  delun [xx11][contactpnt11] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
2596.    fs [xx11][contactpnt11] =  delus [xx11][contactpnt11] *  (dstiffness
   _coefficient/1000);
2597.
2598.    dn [xx11][contactpnt11] = delun [xx11][contactpnt11] * (-
   1) * (ddamping_coefficient/1000);
2599.    ds [xx11][contactpnt11] = delus [xx11][contactpnt11] * (ddamping_coe
   fficient/1000);
2600.
2601.    yforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * sin (alpha [xx11])) -
    ((fn [xx11][contactpnt11] + dn [xx11][contactpnt11]) * cos (alpha [xx11])
   ));
2602.    xforce [qq11][contactpnt11] =  (((fs [xx11][contactpnt11] + ds [xx11
   ][contactpnt11]) * cos (alpha [xx11])) + ((fn [xx11][contactpnt11] + dn [x
   x11][contactpnt11]) * sin (alpha [xx11])));
2603.
2604.    yforce [xx11][contactpnt11] = (yforce [qq11][contactpnt11] * -1);
2605.    xforce [xx11][contactpnt11] = (xforce [qq11][contactpnt11] * -1);
2606.
2607.    fxsum [xx11][contactpnt11] = xforce [xx11][contactpnt11];
2608.    x [xx11] = fxsum [xx11][contactpnt11];
2609.    resultfx [xx11] += x[xx11];
2610.
2611.
2612.    fysum [xx11][contactpnt11] = yforce [xx11][contactpnt11] + p2 ;
2613.    y [xx11] = fysum [xx11][contactpnt11];
2614.    resultfy [xx11] += y[xx11];
2615.
2616.
2617.    msum [xx11][contactpnt11] = (yforce [xx11][contactpnt11]* ( rightiso
   sceles [qq11][1][0] - rightisosceles [xx11][0][0]) -
    xforce [xx11][contactpnt11] * ( rightisosceles [qq11][1][1] -
    rightisosceles [xx11][0][1]));
2618.    m [xx11] = msum [xx11][contactpnt11];
2619.    resultm [xx11] += m[xx11];
2620.
2621.
2622.    udoty [xx11][contactpnt11]=  ((fysum [xx11][contactpnt11] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2623.    udotysum [xx11] = udoty [xx11][contactpnt11];
2624.    resultudoty [xx11] += udotysum [xx11];
2625.
2626.
2627.    udotx [xx11][contactpnt11] =  ((fxsum [xx11][contactpnt11]* dtime_in
   crement)*1000/ dmass); //mm/sec
2628.    udotxsum [xx11] = udotx [xx11][contactpnt11];
2629.    resultudotx [xx11] += udotxsum [xx11];
2630.
2631.
2632.    thetadot [xx11][contactpnt11] = ((msum [xx11][contactpnt11] * dtime_
   increment)*1000/ ii); //1/s
2633.    thetadotsum [xx11] = thetadot [xx11][contactpnt11];
```

```
2634.    resultthetadot [xx11] += thetadotsum [xx11];
2635.
2636.
2637.    deluy [xx11][contactpnt11] = udoty [xx11][contactpnt11] * dtime_incr
   ement;
2638.    deluysum [xx11] = deluy [xx11][contactpnt11];
2639.    resultdeluy [xx11] += deluysum [xx11];
2640.
2641.
2642.    delux [xx11][contactpnt11] = udotx [xx11][contactpnt11] * dtime_incr
   ement;
2643.    deluxsum [xx11] = delux [xx11][contactpnt11];
2644.    resultdelux [xx11] += deluxsum [xx11];
2645.
2646.
2647.    deltheta [xx11][contactpnt11] = thetadot [xx11][contactpnt11] * dtim
   e_increment;
2648.    delthetasum [xx11] = deltheta [xx11][contactpnt11];
2649.    resultdeltheta [xx11] += delthetasum [xx11];
2650.
2651.
2652.    uy [xx11][contactpnt11] = deluy [xx11][contactpnt11];
2653.    uysum [xx11] = uy [xx11][contactpnt11];
2654.    resultuy [xx11]+= uysum [xx11];
2655.
2656.
2657.    ux [xx11][contactpnt11] = delux [xx11][contactpnt11];
2658.    uxsum [xx11] = ux [xx11][contactpnt11];
2659.    resultux [xx11] += uxsum [xx11];
2660.
2661.
2662.    theta [xx11][contactpnt11] = deltheta [xx11][contactpnt11];
2663.    thetasum [xx11] = theta [xx11][contactpnt11];
2664.    resulttheta [xx11] += thetasum [xx11];
2665.    alpha [xx11] = alpha1 + resulttheta [xx11];
2666.
2667.
2668.
2669.    }
2670.
2671.    }
2672.
2673.    }
2674.
2675.    next11:
2676.    ;}
2677.
2678.
2679.
2680.    int xx12 = 0, qq12 = 0, contactpnt12 = 0, l12, i12;
2681.
2682.
```

```
    for (i12 = 0; i12 < isobox12; i12= i12+1)

    {

    xx12 = isocounterbox12[i12];

    rightisosceles [xx12][0][0] = rightisosceles [xx12][0][0] + resultux
   [xx12] ;

    rightisosceles [xx12][0][1] = rightisosceles [xx12][0][1] + resultuy
   [xx12] ;


    for ( l12 = 0; l12 < isobox12; l12 = l12+1)

    {


    qq12 = isocounterbox12[l12];


    if ( ((pow((rightisosceles [xx12][0][0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) > 0) && ((pow((rightisosceles [xx12][0]
   [0] -
    rightisosceles [qq12][0][0]), 2)) + (pow ((rightisosceles [xx12][0][1] -
    rightisosceles [qq12][0][1]), 2)) <= 800))

    {

    contactpnt12 = contactpnt12 + 1;

    if (contactpnt12 >= 4) {goto next12;}


    if ( rightisosceles [qq12][3][1] >  rightisosceles [xx12][1][1] )

    {


    ydisplat [xx12][contactpnt12] = ydisplacement2  + (deluy [xx12][cont
   actpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][0] -
    rightisosceles [qq12][0][0]));
    xdisplat [xx12][contactpnt12] = xdisplacement2 + (delux [xx12][conta
   ctpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][3][1] - rightisosceles [xx12][0][1])  -
```

```
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][3][1] -
   rightisosceles [qq12][0][1]));
2720.
2721.
2722.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2723.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2724.
2725.
2726.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2727.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2728.
2729.
2730.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2731.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2732.
2733.
2734.    yforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * sin (alpha1)) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha1)));
2735.    xforce [qq12][contactpnt12] =   (((fs [xx12][contactpnt12] + ds [xx1
   2][contactpnt12]) * cos (alpha1)) + ((fn [xx12][contactpnt12] + dn [xx12][
   contactpnt12]) * sin (alpha1)));
2736.
2737.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2738.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2739.
2740.
2741.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2742.    y [xx12] = fysum [xx12][contactpnt12];
2743.    resultfy [xx12] += y[xx12];
2744.
2745.
2746.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2747.    x [xx12] = fxsum [xx12][contactpnt12];
2748.    resultfx [xx12] += x[xx12];
2749.
2750.
2751.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][3][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][3][1] -
    rightisosceles [xx12][0][1]));
2752.    m [xx12] = msum [xx12][contactpnt12];
2753.    resultm [xx12] += m[xx12];
2754.
2755.
```

```
2756.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement) *1000/ dmass); // mm/sec
2757.    udotysum [xx12] = udoty [xx12][contactpnt12];
2758.    resultudoty [xx12] += udotysum [xx12];
2759.
2760.
2761.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2762.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2763.    resultudotx [xx12] += udotxsum [xx12];
2764.
2765.
2766.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2767.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2768.    resultthetadot [xx12] += thetadotsum [xx12];
2769.
2770.
2771.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2772.    deluysum [xx12] = deluy [xx12][contactpnt12];
2773.    resultdeluy [xx12] += deluysum [xx12];
2774.
2775.
2776.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2777.    deluxsum [xx12] = delux [xx12][contactpnt12];
2778.    resultdelux [xx12] += deluxsum [xx12];
2779.
2780.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2781.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2782.    resultdeltheta [xx12] += delthetasum [xx12];
2783.
2784.
2785.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2786.    uysum [xx12] = uy [xx12][contactpnt12];
2787.    resultuy [xx12]+= uysum [xx12];
2788.
2789.
2790.    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
2791.    uxsum [xx12] = ux [xx12][contactpnt12];
2792.    resultux [xx12] += uxsum [xx12];
2793.
2794.
2795.    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
2796.    thetasum [xx12] = theta [xx12][contactpnt12];
2797.    resultttheta [xx12] += thetasum [xx12];
2798.    alpha [xx12] = alpha1 + resultttheta [xx12];
2799.
2800.    }
2801.
```

```
2802.    else
2803.
2804.    {
2805.
2806.
2807.
2808.    ydisplat [xx12][contactpnt12] += ydisplacement2  + (deluy [xx12][con
   tactpnt12] -
    deluy [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][0] - rightisosceles [xx12][0][0])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][0][0] -
    rightisosceles [qq12][0][0]));
2809.    xdisplat [xx12][contactpnt12] += xdisplacement2 + (delux [xx12][cont
   actpnt12] -
    delux [qq12][contactpnt12] + deltheta [xx12][contactpnt12] * ( rightisosc
   eles [qq12][1][1] - rightisosceles [xx12][0][1])  -
    deltheta [qq12][contactpnt12] * (rightisosceles [qq12][1][1] -
   rightisosceles [qq12][0][1]));
2810.
2811.    delus [xx12][contactpnt12] = (xdisplat [xx12][contactpnt12] * cos (a
   lpha1) + ydisplat [xx12][contactpnt12] * sin (alpha1));
2812.    delun [xx12][contactpnt12] = (ydisplat [xx12][contactpnt12] * cos (a
   lpha1) - xdisplat [xx12][contactpnt12] * sin (alpha1));
2813.
2814.    fn [xx12][contactpnt12] =  delun [xx12][contactpnt12] * (-
   1) *  (dstiffness_coefficient/1000);
2815.    fs [xx12][contactpnt12] =  delus [xx12][contactpnt12] *  (dstiffness
   _coefficient/1000);
2816.
2817.    dn [xx12][contactpnt12] = delun [xx12][contactpnt12] * (-
   1) * (ddamping_coefficient/1000);
2818.    ds [xx12][contactpnt12] = delus [xx12][contactpnt12] * (ddamping_coe
   fficient/1000);
2819.
2820.    yforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * sin (alpha [xx12])) -
    ((fn [xx12][contactpnt12] + dn [xx12][contactpnt12]) * cos (alpha [xx12])
   ));
2821.    xforce [qq12][contactpnt12] =  (((fs [xx12][contactpnt12] + ds [xx12
   ][contactpnt12]) * cos (alpha [xx12])) + ((fn [xx12][contactpnt12] + dn [x
   x12][contactpnt12]) * sin (alpha [xx12])));
2822.
2823.    yforce [xx12][contactpnt12] = (yforce [qq12][contactpnt12] * -1);
2824.    xforce [xx12][contactpnt12] = (xforce [qq12][contactpnt12] * -1);
2825.
2826.    fxsum [xx12][contactpnt12] = xforce [xx12][contactpnt12];
2827.    x [xx12] = fxsum [xx12][contactpnt12];
2828.    resultfx [xx12] += x[xx12];
2829.
2830.
2831.    fysum [xx12][contactpnt12] = yforce [xx12][contactpnt12] + p2 ;
2832.    y [xx12] = fysum [xx12][contactpnt12];
```

```
2833.    resultfy [xx12] += y[xx12];
2834.
2835.
2836.    msum [xx12][contactpnt12] = (yforce [xx12][contactpnt12]* ( rightiso
   sceles [qq12][1][0] - rightisosceles [xx12][0][0]) -
    xforce [xx12][contactpnt12] * ( rightisosceles [qq12][1][1] -
    rightisosceles [xx12][0][1]));
2837.    m [xx12] = msum [xx12][contactpnt12];
2838.    resultm [xx12] += m[xx12];
2839.
2840.
2841.    udoty [xx12][contactpnt12]=  ((fysum [xx12][contactpnt12] * dtime_in
   crement)* 1000/ dmass); // mm/sec
2842.    udotysum [xx12] = udoty [xx12][contactpnt12];
2843.    resultudoty [xx12] += udotysum [xx12];
2844.
2845.
2846.    udotx [xx12][contactpnt12] =  ((fxsum [xx12][contactpnt12]* dtime_in
   crement)*1000/ dmass); //mm/sec
2847.    udotxsum [xx12] = udotx [xx12][contactpnt12];
2848.    resultudotx [xx12] += udotxsum [xx12];
2849.
2850.
2851.    thetadot [xx12][contactpnt12] = ((msum [xx12][contactpnt12] * dtime_
   increment)*1000/ ii); //1/s
2852.    thetadotsum [xx12] = thetadot [xx12][contactpnt12];
2853.    resultthetadot [xx12] += thetadotsum [xx12];
2854.
2855.
2856.    deluy [xx12][contactpnt12] = udoty [xx12][contactpnt12] * dtime_incr
   ement;
2857.    deluysum [xx12] = deluy [xx12][contactpnt12];
2858.    resultdeluy [xx12] += deluysum [xx12];
2859.
2860.
2861.    delux [xx12][contactpnt12] = udotx [xx12][contactpnt12] * dtime_incr
   ement;
2862.    deluxsum [xx12] = delux [xx12][contactpnt12];
2863.    resultdelux [xx12] += deluxsum [xx12];
2864.
2865.
2866.    deltheta [xx12][contactpnt12] = thetadot [xx12][contactpnt12] * dtim
   e_increment;
2867.    delthetasum [xx12] = deltheta [xx12][contactpnt12];
2868.    resultdeltheta [xx12] += delthetasum [xx12];
2869.
2870.
2871.    uy [xx12][contactpnt12] = deluy [xx12][contactpnt12];
2872.    uysum [xx12] = uy [xx12][contactpnt12];
2873.    resultuy [xx12]+= uysum [xx12];
2874.
2875.
```

```
    ux [xx12][contactpnt12] = delux [xx12][contactpnt12];
    uxsum [xx12] = ux [xx12][contactpnt12];
    resultux [xx12] += uxsum [xx12];


    theta [xx12][contactpnt12] = deltheta [xx12][contactpnt12];
    thetasum [xx12] = theta [xx12][contactpnt12];
    resulttheta [xx12] += thetasum [xx12];
    alpha [xx12] = alpha1 + resulttheta [xx12];

    }
    }

    }

    next12:
    ;}


    int xx13 = 0, qq13 = 0, contactpnt13 = 0, l13, i13;


    for (i13 = 0; i13 < isobox13; i13= i13+1)

    {

    xx13 = isocounterbox13[i13];


    rightisosceles [xx13][0][0] = rightisosceles [xx13][0][0] + resultux
    [xx13] ;

    rightisosceles [xx13][0][1] = rightisosceles [xx13][0][1] + resultuy
    [xx13] ;


    for ( l13 = 0; l13 < isobox13; l13 = l13+1)

    {


    qq13 = isocounterbox13[l13];


    if ( ((pow((rightisosceles [xx13][0][0] -
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) > 0) && ((pow((rightisosceles [xx13][0]
   [0] -
```

```
    rightisosceles [qq13][0][0]), 2)) + (pow ((rightisosceles [xx13][0][1] -
    rightisosceles [qq13][0][1]), 2)) <= 800))
2923.
2924.    {
2925.
2926.    contactpnt13 = contactpnt13 + 1;
2927.
2928.    if (contactpnt13 >= 4) {goto next13;}
2929.
2930.
2931.    if ( rightisosceles [qq13][3][1] >  rightisosceles [xx13][1][1] )
2932.
2933.    {
2934.
2935.
2936.    ydisplat [xx13][contactpnt13] = ydisplacement2  + (deluy [xx13][cont
   actpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][0] -
    rightisosceles [qq13][0][0]));
2937.    xdisplat [xx13][contactpnt13] = xdisplacement2 + (delux [xx13][conta
   ctpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][3][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][3][1] -
   rightisosceles [qq13][0][1]));
2938.
2939.
2940.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
2941.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
2942.
2943.
2944.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
2945.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
2946.
2947.
2948.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
2949.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
2950.
2951.
2952.    yforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * sin (alpha1)) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha1)));
```

```
2953.    xforce [qq13][contactpnt13] =   (((fs [xx13][contactpnt13] + ds [xx1
   3][contactpnt13]) * cos (alpha1)) + ((fn [xx13][contactpnt13] + dn [xx13][
   contactpnt13]) * sin (alpha1)));
2954.
2955.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
2956.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
2957.
2958.
2959.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
2960.    y [xx13] = fysum [xx13][contactpnt13];
2961.    resultfy [xx13] += y[xx13];
2962.
2963.
2964.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
2965.    x [xx13] = fxsum [xx13][contactpnt13];
2966.    resultfx [xx13] += x[xx13];
2967.
2968.
2969.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][3][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][3][1] -
    rightisosceles [xx13][0][1]));
2970.    m [xx13] = msum [xx13][contactpnt13];
2971.    resultm [xx13] += m[xx13];
2972.
2973.
2974.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement) *1000/ dmass); // mm/sec
2975.    udotysum [xx13] = udoty [xx13][contactpnt13];
2976.    resultudoty [xx13] += udotysum [xx13];
2977.
2978.
2979.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
2980.    udotxsum [xx13] = udotx [xx13][contactpnt13];
2981.    resultudotx [xx13] += udotxsum [xx13];
2982.
2983.
2984.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
2985.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
2986.    resultthetadot [xx13] += thetadotsum [xx13];
2987.
2988.
2989.    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
   ement;
2990.    deluysum [xx13] = deluy [xx13][contactpnt13];
2991.    resultdeluy [xx13] += deluysum [xx13];
2992.
2993.
2994.    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
   ement;
```

```
2995.    deluxsum [xx13] = delux [xx13][contactpnt13];
2996.    resultdelux [xx13] += deluxsum [xx13];
2997.
2998.    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
   e_increment;
2999.    delthetasum [xx13] = deltheta [xx13][contactpnt13];
3000.    resultdeltheta [xx13] += delthetasum [xx13];
3001.
3002.
3003.    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
3004.    uysum [xx13] = uy [xx13][contactpnt13];
3005.    resultuy [xx13]+= uysum [xx13];
3006.
3007.
3008.    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
3009.    uxsum [xx13] = ux [xx13][contactpnt13];
3010.    resultux [xx13] += uxsum [xx13];
3011.
3012.
3013.    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
3014.    thetasum [xx13] = theta [xx13][contactpnt13];
3015.    resulttheta [xx13] += thetasum [xx13];
3016.    alpha [xx13] = alpha1 + resulttheta [xx13];
3017.
3018.    }
3019.
3020.    else
3021.
3022.    {
3023.
3024.
3025.
3026.    ydisplat [xx13][contactpnt13] += ydisplacement2  + (deluy [xx13][con
   tactpnt13] -
    deluy [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][0] - rightisosceles [xx13][0][0])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][0][0] -
    rightisosceles [qq13][0][0]));
3027.    xdisplat [xx13][contactpnt13] += xdisplacement2 + (delux [xx13][cont
   actpnt13] -
    delux [qq13][contactpnt13] + deltheta [xx13][contactpnt13] * ( rightisosc
   eles [qq13][1][1] - rightisosceles [xx13][0][1])  -
    deltheta [qq13][contactpnt13] * (rightisosceles [qq13][1][1] -
   rightisosceles [qq13][0][1]));
3028.
3029.    delus [xx13][contactpnt13] = (xdisplat [xx13][contactpnt13] * cos (a
   lpha1) + ydisplat [xx13][contactpnt13] * sin (alpha1));
3030.    delun [xx13][contactpnt13] = (ydisplat [xx13][contactpnt13] * cos (a
   lpha1) - xdisplat [xx13][contactpnt13] * sin (alpha1));
3031.
3032.    fn [xx13][contactpnt13] =  delun [xx13][contactpnt13] * (-
   1) *  (dstiffness_coefficient/1000);
```

```
3033.    fs [xx13][contactpnt13] =  delus [xx13][contactpnt13] *  (dstiffness
   _coefficient/1000);
3034.
3035.    dn [xx13][contactpnt13] = delun [xx13][contactpnt13] * (-
   1) * (ddamping_coefficient/1000);
3036.    ds [xx13][contactpnt13] = delus [xx13][contactpnt13] * (ddamping_coe
   fficient/1000);
3037.
3038.    yforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * sin (alpha [xx13])) -
    ((fn [xx13][contactpnt13] + dn [xx13][contactpnt13]) * cos (alpha [xx13])
   ));
3039.    xforce [qq13][contactpnt13] =  (((fs [xx13][contactpnt13] + ds [xx13
   ][contactpnt13]) * cos (alpha [xx13])) + ((fn [xx13][contactpnt13] + dn [x
   x13][contactpnt13]) * sin (alpha [xx13])));
3040.
3041.    yforce [xx13][contactpnt13] = (yforce [qq13][contactpnt13] * -1);
3042.    xforce [xx13][contactpnt13] = (xforce [qq13][contactpnt13] * -1);
3043.
3044.    fxsum [xx13][contactpnt13] = xforce [xx13][contactpnt13];
3045.    x [xx13] = fxsum [xx13][contactpnt13];
3046.    resultfx [xx13] += x[xx13];
3047.
3048.
3049.    fysum [xx13][contactpnt13] = yforce [xx13][contactpnt13] + p2 ;
3050.    y [xx13] = fysum [xx13][contactpnt13];
3051.    resultfy [xx13] += y[xx13];
3052.
3053.
3054.    msum [xx13][contactpnt13] = (yforce [xx13][contactpnt13]* ( rightiso
   sceles [qq13][1][0] - rightisosceles [xx13][0][0]) -
    xforce [xx13][contactpnt13] * ( rightisosceles [qq13][1][1] -
    rightisosceles [xx13][0][1]));
3055.    m [xx13] = msum [xx13][contactpnt13];
3056.    resultm [xx13] += m[xx13];
3057.
3058.
3059.    udoty [xx13][contactpnt13]=  ((fysum [xx13][contactpnt13] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3060.    udotysum [xx13] = udoty [xx13][contactpnt13];
3061.    resultudoty [xx13] += udotysum [xx13];
3062.
3063.
3064.    udotx [xx13][contactpnt13] =  ((fxsum [xx13][contactpnt13]* dtime_in
   crement)*1000/ dmass); //mm/sec
3065.    udotxsum [xx13] = udotx [xx13][contactpnt13];
3066.    resultudotx [xx13] += udotxsum [xx13];
3067.
3068.
3069.    thetadot [xx13][contactpnt13] = ((msum [xx13][contactpnt13] * dtime_
   increment)*1000/ ii); //1/s
3070.    thetadotsum [xx13] = thetadot [xx13][contactpnt13];
```

```
    resultthetadot [xx13] += thetadotsum [xx13];


    deluy [xx13][contactpnt13] = udoty [xx13][contactpnt13] * dtime_incr
ement;
    deluysum [xx13] = deluy [xx13][contactpnt13];
    resultdeluy [xx13] += deluysum [xx13];


    delux [xx13][contactpnt13] = udotx [xx13][contactpnt13] * dtime_incr
ement;
    deluxsum [xx13] = delux [xx13][contactpnt13];
    resultdelux [xx13] += deluxsum [xx13];


    deltheta [xx13][contactpnt13] = thetadot [xx13][contactpnt13] * dtim
e_increment;
    delthetasum [xx13] = deltheta [xx13][contactpnt13];
    resultdeltheta [xx13] += delthetasum [xx13];


    uy [xx13][contactpnt13] = deluy [xx13][contactpnt13];
    uysum [xx13] = uy [xx13][contactpnt13];
    resultuy [xx13]+= uysum [xx13];


    ux [xx13][contactpnt13] = delux [xx13][contactpnt13];
    uxsum [xx13] = ux [xx13][contactpnt13];
    resultux [xx13] += uxsum [xx13];


    theta [xx13][contactpnt13] = deltheta [xx13][contactpnt13];
    thetasum [xx13] = theta [xx13][contactpnt13];
    resulttheta [xx13] += thetasum [xx13];
    alpha [xx13] = alpha1 + resulttheta [xx13];



    }

    }

    }

    next13:
    ;}


    int xx14 = 0, qq14 = 0, contactpnt14 = 0, l14, i14;

```

```
3120.    for (i14 = 0; i14 < isobox14; i14= i14+1)
3121.
3122.    {
3123.
3124.    xx14 = isocounterbox14[i14];
3125.
3126.    rightisosceles [xx14][0][0] = rightisosceles [xx14][0][0] + resultux
    [xx14] ;
3127.
3128.    rightisosceles [xx14][0][1] = rightisosceles [xx14][0][1] + resultuy
    [xx14] ;
3129.
3130.
3131.
3132.    for ( l14 = 0; l14 < isobox14; l14 = l14+1)
3133.
3134.    {
3135.
3136.
3137.    qq14 = isocounterbox14[l14];
3138.
3139.
3140.
3141.    if ( ((pow((rightisosceles [xx14][0][0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) > 0) && ((pow((rightisosceles [xx14][0]
   [0] -
    rightisosceles [qq14][0][0]), 2)) + (pow ((rightisosceles [xx14][0][1] -
    rightisosceles [qq14][0][1]), 2)) <= 800))
3142.
3143.    {
3144.
3145.    contactpnt14 = contactpnt14 + 1;
3146.
3147.    if (contactpnt14 >= 4) {goto next14;}
3148.
3149.
3150.    if ( rightisosceles [qq14][3][1] >  rightisosceles [xx14][1][1] )
3151.
3152.    {
3153.
3154.
3155.    ydisplat [xx14][contactpnt14] = ydisplacement2  + (deluy [xx14][cont
   actpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][0] -
    rightisosceles [qq14][0][0]));
3156.    xdisplat [xx14][contactpnt14] = xdisplacement2 + (delux [xx14][conta
   ctpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][3][1] - rightisosceles [xx14][0][1])  -
```

```
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][3][1] -
   rightisosceles [qq14][0][1]));
3157.
3158.
3159.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3160.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3161.
3162.
3163.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3164.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3165.
3166.
3167.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3168.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3169.
3170.
3171.    yforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * sin (alpha1)) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha1)));
3172.    xforce [qq14][contactpnt14] =   (((fs [xx14][contactpnt14] + ds [xx1
   4][contactpnt14]) * cos (alpha1)) + ((fn [xx14][contactpnt14] + dn [xx14][
   contactpnt14]) * sin (alpha1)));
3173.
3174.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3175.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3176.
3177.
3178.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3179.    y [xx14] = fysum [xx14][contactpnt14];
3180.    resultfy [xx14] += y[xx14];
3181.
3182.
3183.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3184.    x [xx14] = fxsum [xx14][contactpnt14];
3185.    resultfx [xx14] += x[xx14];
3186.
3187.
3188.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][3][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][3][1] -
    rightisosceles [xx14][0][1]));
3189.    m [xx14] = msum [xx14][contactpnt14];
3190.    resultm [xx14] += m[xx14];
3191.
3192.
```

```
3193.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement) *1000/ dmass); // mm/sec
3194.    udotysum [xx14] = udoty [xx14][contactpnt14];
3195.    resultudoty [xx14] += udotysum [xx14];
3196.
3197.
3198.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3199.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3200.    resultudotx [xx14] += udotxsum [xx14];
3201.
3202.
3203.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3204.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3205.    resultthetadot [xx14] += thetadotsum [xx14];
3206.
3207.
3208.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3209.    deluysum [xx14] = deluy [xx14][contactpnt14];
3210.    resultdeluy [xx14] += deluysum [xx14];
3211.
3212.
3213.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3214.    deluxsum [xx14] = delux [xx14][contactpnt14];
3215.    resultdelux [xx14] += deluxsum [xx14];
3216.
3217.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3218.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3219.    resultdeltheta [xx14] += delthetasum [xx14];
3220.
3221.
3222.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3223.    uysum [xx14] = uy [xx14][contactpnt14];
3224.    resultuy [xx14]+= uysum [xx14];
3225.
3226.
3227.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3228.    uxsum [xx14] = ux [xx14][contactpnt14];
3229.    resultux [xx14] += uxsum [xx14];
3230.
3231.
3232.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3233.    thetasum [xx14] = theta [xx14][contactpnt14];
3234.    resultttheta [xx14] += thetasum [xx14];
3235.    alpha [xx14] = alpha1 + resultttheta [xx14];
3236.
3237.    }
3238.
```

```
3239.    else
3240.
3241.    {
3242.
3243.
3244.
3245.    ydisplat [xx14][contactpnt14] += ydisplacement2  + (deluy [xx14][con
   tactpnt14] -
    deluy [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][0] - rightisosceles [xx14][0][0])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][0][0] -
    rightisosceles [qq14][0][0]));
3246.    xdisplat [xx14][contactpnt14] += xdisplacement2 + (delux [xx14][cont
   actpnt14] -
    delux [qq14][contactpnt14] + deltheta [xx14][contactpnt14] * ( rightisosc
   eles [qq14][1][1] - rightisosceles [xx14][0][1])  -
    deltheta [qq14][contactpnt14] * (rightisosceles [qq14][1][1] -
   rightisosceles [qq14][0][1]));
3247.
3248.    delus [xx14][contactpnt14] = (xdisplat [xx14][contactpnt14] * cos (a
   lpha1) + ydisplat [xx14][contactpnt14] * sin (alpha1));
3249.    delun [xx14][contactpnt14] = (ydisplat [xx14][contactpnt14] * cos (a
   lpha1) - xdisplat [xx14][contactpnt14] * sin (alpha1));
3250.
3251.    fn [xx14][contactpnt14] =  delun [xx14][contactpnt14] * (-
   1) *  (dstiffness_coefficient/1000);
3252.    fs [xx14][contactpnt14] =  delus [xx14][contactpnt14] *  (dstiffness
   _coefficient/1000);
3253.
3254.    dn [xx14][contactpnt14] = delun [xx14][contactpnt14] * (-
   1) * (ddamping_coefficient/1000);
3255.    ds [xx14][contactpnt14] = delus [xx14][contactpnt14] * (ddamping_coe
   fficient/1000);
3256.
3257.    yforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * sin (alpha [xx14])) -
    ((fn [xx14][contactpnt14] + dn [xx14][contactpnt14]) * cos (alpha [xx14])
   ));
3258.    xforce [qq14][contactpnt14] =  (((fs [xx14][contactpnt14] + ds [xx14
   ][contactpnt14]) * cos (alpha [xx14])) + ((fn [xx14][contactpnt14] + dn [x
   x14][contactpnt14]) * sin (alpha [xx14])));
3259.
3260.    yforce [xx14][contactpnt14] = (yforce [qq14][contactpnt14] * -1);
3261.    xforce [xx14][contactpnt14] = (xforce [qq14][contactpnt14] * -1);
3262.
3263.    fxsum [xx14][contactpnt14] = xforce [xx14][contactpnt14];
3264.    x [xx14] = fxsum [xx14][contactpnt14];
3265.    resultfx [xx14] += x[xx14];
3266.
3267.
3268.    fysum [xx14][contactpnt14] = yforce [xx14][contactpnt14] + p2 ;
3269.    y [xx14] = fysum [xx14][contactpnt14];
```

```
3270.    resultfy [xx14] += y[xx14];
3271.
3272.
3273.    msum [xx14][contactpnt14] = (yforce [xx14][contactpnt14]* ( rightiso
   sceles [qq14][1][0] - rightisosceles [xx14][0][0]) -
    xforce [xx14][contactpnt14] * ( rightisosceles [qq14][1][1] -
    rightisosceles [xx14][0][1]));
3274.    m [xx14] = msum [xx14][contactpnt14];
3275.    resultm [xx14] += m[xx14];
3276.
3277.
3278.    udoty [xx14][contactpnt14]=  ((fysum [xx14][contactpnt14] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3279.    udotysum [xx14] = udoty [xx14][contactpnt14];
3280.    resultudoty [xx14] += udotysum [xx14];
3281.
3282.
3283.    udotx [xx14][contactpnt14] =  ((fxsum [xx14][contactpnt14]* dtime_in
   crement)*1000/ dmass); //mm/sec
3284.    udotxsum [xx14] = udotx [xx14][contactpnt14];
3285.    resultudotx [xx14] += udotxsum [xx14];
3286.
3287.
3288.    thetadot [xx14][contactpnt14] = ((msum [xx14][contactpnt14] * dtime_
   increment)*1000/ ii); //1/s
3289.    thetadotsum [xx14] = thetadot [xx14][contactpnt14];
3290.    resultthetadot [xx14] += thetadotsum [xx14];
3291.
3292.
3293.    deluy [xx14][contactpnt14] = udoty [xx14][contactpnt14] * dtime_incr
   ement;
3294.    deluysum [xx14] = deluy [xx14][contactpnt14];
3295.    resultdeluy [xx14] += deluysum [xx14];
3296.
3297.
3298.    delux [xx14][contactpnt14] = udotx [xx14][contactpnt14] * dtime_incr
   ement;
3299.    deluxsum [xx14] = delux [xx14][contactpnt14];
3300.    resultdelux [xx14] += deluxsum [xx14];
3301.
3302.
3303.    deltheta [xx14][contactpnt14] = thetadot [xx14][contactpnt14] * dtim
   e_increment;
3304.    delthetasum [xx14] = deltheta [xx14][contactpnt14];
3305.    resultdeltheta [xx14] += delthetasum [xx14];
3306.
3307.
3308.    uy [xx14][contactpnt14] = deluy [xx14][contactpnt14];
3309.    uysum [xx14] = uy [xx14][contactpnt14];
3310.    resultuy [xx14]+= uysum [xx14];
3311.
3312.
```

```
3313.    ux [xx14][contactpnt14] = delux [xx14][contactpnt14];
3314.    uxsum [xx14] = ux [xx14][contactpnt14];
3315.    resultux [xx14] += uxsum [xx14];
3316.
3317.
3318.    theta [xx14][contactpnt14] = deltheta [xx14][contactpnt14];
3319.    thetasum [xx14] = theta [xx14][contactpnt14];
3320.    resulttheta [xx14] += thetasum [xx14];
3321.    alpha [xx14] = alpha1 + resulttheta [xx14];
3322.
3323.    }
3324.    }
3325.
3326.    }
3327.
3328.    next14:
3329.    ;}
3330.
3331.
3332.
3333.    int xx15 = 0, qq15 = 0, contactpnt15 = 0, l15, i15;
3334.
3335.
3336.    for (i15 = 0; i15 < isobox15; i15= i15+1)
3337.
3338.    {
3339.
3340.    xx15 = isocounterbox15[i15];
3341.
3342.    rightisosceles [xx15][0][0] = rightisosceles [xx15][0][0] + resultux
    [xx15] ;
3343.
3344.    rightisosceles [xx15][0][1] = rightisosceles [xx15][0][1] + resultuy
    [xx15] ;
3345.
3346.
3347.
3348.    for ( l15 = 0; l15 < isobox15; l15 = l15+1)
3349.
3350.    {
3351.
3352.
3353.    qq15 = isocounterbox15[l15];
3354.
3355.
3356.
3357.    if ( ((pow((rightisosceles [xx15][0][0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) > 0) && ((pow((rightisosceles [xx15][0]
   [0] -
    rightisosceles [qq15][0][0]), 2)) + (pow ((rightisosceles [xx15][0][1] -
    rightisosceles [qq15][0][1]), 2)) <= 800))
```

```
3358.
3359.    {
3360.
3361.    contactpnt15 = contactpnt15 + 1;
3362.
3363.    if (contactpnt15 >= 4) {goto next15;}
3364.
3365.
3366.    if ( rightisosceles [qq15][3][1] >  rightisosceles [xx15][1][1] )
3367.
3368.    {
3369.
3370.
3371.    ydisplat [xx15][contactpnt15] = ydisplacement2  + (deluy [xx15][cont
   actpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][0] -
    rightisosceles [qq15][0][0]));
3372.    xdisplat [xx15][contactpnt15] = xdisplacement2 + (delux [xx15][conta
   ctpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][3][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][3][1] -
   rightisosceles [qq15][0][1]));
3373.
3374.
3375.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3376.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3377.
3378.
3379.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3380.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3381.
3382.
3383.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3384.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3385.
3386.
3387.    yforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * sin (alpha1)) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha1)));
3388.    xforce [qq15][contactpnt15] =   (((fs [xx15][contactpnt15] + ds [xx1
   5][contactpnt15]) * cos (alpha1)) + ((fn [xx15][contactpnt15] + dn [xx15][
   contactpnt15]) * sin (alpha1)));
3389.
```

```
    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);


    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
    y [xx15] = fysum [xx15][contactpnt15];
    resultfy [xx15] += y[xx15];


    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
    x [xx15] = fxsum [xx15][contactpnt15];
    resultfx [xx15] += x[xx15];


    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][3][0] - rightisosceles [xx15][0][0]) -
     xforce [xx15][contactpnt15] * ( rightisosceles [qq15][3][1] -
     rightisosceles [xx15][0][1]));
    m [xx15] = msum [xx15][contactpnt15];
    resultm [xx15] += m[xx15];


    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement) *1000/ dmass); // mm/sec
    udotysum [xx15] = udoty [xx15][contactpnt15];
    resultudoty [xx15] += udotysum [xx15];


    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
    udotxsum [xx15] = udotx [xx15][contactpnt15];
    resultudotx [xx15] += udotxsum [xx15];


    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
    resultthetadot [xx15] += thetadotsum [xx15];


    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
    deluysum [xx15] = deluy [xx15][contactpnt15];
    resultdeluy [xx15] += deluysum [xx15];


    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
    deluxsum [xx15] = delux [xx15][contactpnt15];
    resultdelux [xx15] += deluxsum [xx15];

```

```
3433.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3434.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3435.    resultdeltheta [xx15] += delthetasum [xx15];
3436.
3437.
3438.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3439.    uysum [xx15] = uy [xx15][contactpnt15];
3440.    resultuy [xx15]+= uysum [xx15];
3441.
3442.
3443.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3444.    uxsum [xx15] = ux [xx15][contactpnt15];
3445.    resultux [xx15] += uxsum [xx15];
3446.
3447.
3448.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3449.    thetasum [xx15] = theta [xx15][contactpnt15];
3450.    resulttheta [xx15] += thetasum [xx15];
3451.    alpha [xx15] = alpha1 + resulttheta [xx15];
3452.
3453.    }
3454.
3455.    else
3456.
3457.    {
3458.
3459.
3460.
3461.    ydisplat [xx15][contactpnt15] += ydisplacement2  + (deluy [xx15][con
   tactpnt15] -
    deluy [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][0] - rightisosceles [xx15][0][0])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][0][0] -
    rightisosceles [qq15][0][0]));
3462.    xdisplat [xx15][contactpnt15] += xdisplacement2 + (delux [xx15][cont
   actpnt15] -
    delux [qq15][contactpnt15] + deltheta [xx15][contactpnt15] * ( rightisosc
   eles [qq15][1][1] - rightisosceles [xx15][0][1])  -
    deltheta [qq15][contactpnt15] * (rightisosceles [qq15][1][1] -
   rightisosceles [qq15][0][1]));
3463.
3464.    delus [xx15][contactpnt15] = (xdisplat [xx15][contactpnt15] * cos (a
   lpha1) + ydisplat [xx15][contactpnt15] * sin (alpha1));
3465.    delun [xx15][contactpnt15] = (ydisplat [xx15][contactpnt15] * cos (a
   lpha1) - xdisplat [xx15][contactpnt15] * sin (alpha1));
3466.
3467.    fn [xx15][contactpnt15] =  delun [xx15][contactpnt15] * (-
   1) *  (dstiffness_coefficient/1000);
3468.    fs [xx15][contactpnt15] =  delus [xx15][contactpnt15] *  (dstiffness
   _coefficient/1000);
3469.
```

```
3470.    dn [xx15][contactpnt15] = delun [xx15][contactpnt15] * (-
   1) * (ddamping_coefficient/1000);
3471.    ds [xx15][contactpnt15] = delus [xx15][contactpnt15] * (ddamping_coe
   fficient/1000);
3472.
3473.    yforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * sin (alpha [xx15])) -
    ((fn [xx15][contactpnt15] + dn [xx15][contactpnt15]) * cos (alpha [xx15])
   ));
3474.    xforce [qq15][contactpnt15] =  (((fs [xx15][contactpnt15] + ds [xx15
   ][contactpnt15]) * cos (alpha [xx15])) + ((fn [xx15][contactpnt15] + dn [x
   x15][contactpnt15]) * sin (alpha [xx15])));
3475.
3476.    yforce [xx15][contactpnt15] = (yforce [qq15][contactpnt15] * -1);
3477.    xforce [xx15][contactpnt15] = (xforce [qq15][contactpnt15] * -1);
3478.
3479.    fxsum [xx15][contactpnt15] = xforce [xx15][contactpnt15];
3480.    x [xx15] = fxsum [xx15][contactpnt15];
3481.    resultfx [xx15] += x[xx15];
3482.
3483.
3484.    fysum [xx15][contactpnt15] = yforce [xx15][contactpnt15] + p2 ;
3485.    y [xx15] = fysum [xx15][contactpnt15];
3486.    resultfy [xx15] += y[xx15];
3487.
3488.
3489.    msum [xx15][contactpnt15] = (yforce [xx15][contactpnt15]* ( rightiso
   sceles [qq15][1][0] - rightisosceles [xx15][0][0]) -
    xforce [xx15][contactpnt15] * ( rightisosceles [qq15][1][1] -
    rightisosceles [xx15][0][1]));
3490.    m [xx15] = msum [xx15][contactpnt15];
3491.    resultm [xx15] += m[xx15];
3492.
3493.
3494.    udoty [xx15][contactpnt15]=  ((fysum [xx15][contactpnt15] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3495.    udotysum [xx15] = udoty [xx15][contactpnt15];
3496.    resultudoty [xx15] += udotysum [xx15];
3497.
3498.
3499.    udotx [xx15][contactpnt15] =  ((fxsum [xx15][contactpnt15]* dtime_in
   crement)*1000/ dmass); //mm/sec
3500.    udotxsum [xx15] = udotx [xx15][contactpnt15];
3501.    resultudotx [xx15] += udotxsum [xx15];
3502.
3503.
3504.    thetadot [xx15][contactpnt15] = ((msum [xx15][contactpnt15] * dtime_
   increment)*1000/ ii); //1/s
3505.    thetadotsum [xx15] = thetadot [xx15][contactpnt15];
3506.    resultthetadot [xx15] += thetadotsum [xx15];
3507.
3508.
```

```
3509.    deluy [xx15][contactpnt15] = udoty [xx15][contactpnt15] * dtime_incr
   ement;
3510.    deluysum [xx15] = deluy [xx15][contactpnt15];
3511.    resultdeluy [xx15] += deluysum [xx15];
3512.
3513.
3514.    delux [xx15][contactpnt15] = udotx [xx15][contactpnt15] * dtime_incr
   ement;
3515.    deluxsum [xx15] = delux [xx15][contactpnt15];
3516.    resultdelux [xx15] += deluxsum [xx15];
3517.
3518.
3519.    deltheta [xx15][contactpnt15] = thetadot [xx15][contactpnt15] * dtim
   e_increment;
3520.    delthetasum [xx15] = deltheta [xx15][contactpnt15];
3521.    resultdeltheta [xx15] += delthetasum [xx15];
3522.
3523.
3524.    uy [xx15][contactpnt15] = deluy [xx15][contactpnt15];
3525.    uysum [xx15] = uy [xx15][contactpnt15];
3526.    resultuy [xx15]+= uysum [xx15];
3527.
3528.
3529.    ux [xx15][contactpnt15] = delux [xx15][contactpnt15];
3530.    uxsum [xx15] = ux [xx15][contactpnt15];
3531.    resultux [xx15] += uxsum [xx15];
3532.
3533.
3534.    theta [xx15][contactpnt15] = deltheta [xx15][contactpnt15];
3535.    thetasum [xx15] = theta [xx15][contactpnt15];
3536.    resulttheta [xx15] += thetasum [xx15];
3537.    alpha [xx15] = alpha1 + resulttheta [xx15];
3538.
3539.    }
3540.    }
3541.
3542.    }
3543.
3544.    next15:
3545.    ;}
3546.
3547.
3548.
3549.    int xx16 = 0, qq16 = 0, contactpnt16 = 0, l16, i16;
3550.
3551.
3552.    for (i16 = 0; i16 < isobox16; i16= i16+1)
3553.
3554.    {
3555.
3556.    xx16 = isocounterbox16[i16];
3557.
```

```
3558.
3559.
3560.    rightisosceles [xx16][0][0] = rightisosceles [xx16][0][0] + resultux
    [xx16] ;
3561.
3562.    rightisosceles [xx16][0][1] = rightisosceles [xx16][0][1] + resultuy
    [xx16] ;
3563.
3564.
3565.
3566.    for ( l16 = 0; l16 < isobox16; l16 = l16+1)
3567.
3568.    {
3569.
3570.
3571.    qq16 = isocounterbox16[l16];
3572.
3573.
3574.
3575.    if ( ((pow((rightisosceles [xx16][0][0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) > 0) && ((pow((rightisosceles [xx16][0]
   [0] -
    rightisosceles [qq16][0][0]), 2)) + (pow ((rightisosceles [xx16][0][1] -
    rightisosceles [qq16][0][1]), 2)) <= 800))
3576.
3577.    {
3578.
3579.    contactpnt16 = contactpnt16 + 1;
3580.
3581.    if (contactpnt16 >= 4) {goto next16;}
3582.
3583.
3584.    if ( rightisosceles [qq16][3][1] >  rightisosceles [xx16][1][1] )
3585.
3586.    {
3587.
3588.
3589.    ydisplat [xx16][contactpnt16] = ydisplacement2  + (deluy [xx16][cont
   actpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][0] -
    rightisosceles [qq16][0][0]));
3590.    xdisplat [xx16][contactpnt16] = xdisplacement2 + (delux [xx16][conta
   ctpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][3][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][3][1] -
   rightisosceles [qq16][0][1]));
3591.
3592.
```

```
3593.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3594.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3595.
3596.
3597.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3598.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3599.
3600.
3601.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3602.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3603.
3604.
3605.    yforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * sin (alpha1)) -
     ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha1)));
3606.    xforce [qq16][contactpnt16] =   (((fs [xx16][contactpnt16] + ds [xx1
   6][contactpnt16]) * cos (alpha1)) + ((fn [xx16][contactpnt16] + dn [xx16][
   contactpnt16]) * sin (alpha1)));
3607.
3608.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3609.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3610.
3611.
3612.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3613.    y [xx16] = fysum [xx16][contactpnt16];
3614.    resultfy [xx16] += y[xx16];
3615.
3616.
3617.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3618.    x [xx16] = fxsum [xx16][contactpnt16];
3619.    resultfx [xx16] += x[xx16];
3620.
3621.
3622.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][3][0] - rightisosceles [xx16][0][0]) -
     xforce [xx16][contactpnt16] * ( rightisosceles [qq16][3][1] -
     rightisosceles [xx16][0][1]));
3623.    m [xx16] = msum [xx16][contactpnt16];
3624.    resultm [xx16] += m[xx16];
3625.
3626.
3627.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement) *1000/ dmass); // mm/sec
3628.    udotysum [xx16] = udoty [xx16][contactpnt16];
3629.    resultudoty [xx16] += udotysum [xx16];
3630.
```

```
3631.
3632.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3633.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3634.    resultudotx [xx16] += udotxsum [xx16];
3635.
3636.
3637.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3638.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3639.    resultthetadot [xx16] += thetadotsum [xx16];
3640.
3641.
3642.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3643.    deluysum [xx16] = deluy [xx16][contactpnt16];
3644.    resultdeluy [xx16] += deluysum [xx16];
3645.
3646.
3647.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3648.    deluxsum [xx16] = delux [xx16][contactpnt16];
3649.    resultdelux [xx16] += deluxsum [xx16];
3650.
3651.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3652.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3653.    resultdeltheta [xx16] += delthetasum [xx16];
3654.
3655.
3656.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3657.    uysum [xx16] = uy [xx16][contactpnt16];
3658.    resultuy [xx16]+= uysum [xx16];
3659.
3660.
3661.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3662.    uxsum [xx16] = ux [xx16][contactpnt16];
3663.    resultux [xx16] += uxsum [xx16];
3664.
3665.
3666.    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
3667.    thetasum [xx16] = theta [xx16][contactpnt16];
3668.    resultttheta [xx16] += thetasum [xx16];
3669.    alpha [xx16] = alpha1 + resultttheta [xx16];
3670.
3671.    }
3672.
3673.    else
3674.
3675.    {
3676.
3677.
```

```
3678.
3679.    ydisplat [xx16][contactpnt16] += ydisplacement2  + (deluy [xx16][con
   tactpnt16] -
    deluy [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][0] - rightisosceles [xx16][0][0])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][0][0] -
    rightisosceles [qq16][0][0]));
3680.    xdisplat [xx16][contactpnt16] += xdisplacement2 + (delux [xx16][cont
   actpnt16] -
    delux [qq16][contactpnt16] + deltheta [xx16][contactpnt16] * ( rightisosc
   eles [qq16][1][1] - rightisosceles [xx16][0][1])  -
    deltheta [qq16][contactpnt16] * (rightisosceles [qq16][1][1] -
   rightisosceles [qq16][0][1]));
3681.
3682.    delus [xx16][contactpnt16] = (xdisplat [xx16][contactpnt16] * cos (a
   lpha1) + ydisplat [xx16][contactpnt16] * sin (alpha1));
3683.    delun [xx16][contactpnt16] = (ydisplat [xx16][contactpnt16] * cos (a
   lpha1) - xdisplat [xx16][contactpnt16] * sin (alpha1));
3684.
3685.    fn [xx16][contactpnt16] =  delun [xx16][contactpnt16] * (-
   1) *  (dstiffness_coefficient/1000);
3686.    fs [xx16][contactpnt16] =  delus [xx16][contactpnt16] *  (dstiffness
   _coefficient/1000);
3687.
3688.    dn [xx16][contactpnt16] = delun [xx16][contactpnt16] * (-
   1) * (ddamping_coefficient/1000);
3689.    ds [xx16][contactpnt16] = delus [xx16][contactpnt16] * (ddamping_coe
   fficient/1000);
3690.
3691.    yforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * sin (alpha [xx16])) -
    ((fn [xx16][contactpnt16] + dn [xx16][contactpnt16]) * cos (alpha [xx16])
   ));
3692.    xforce [qq16][contactpnt16] =  (((fs [xx16][contactpnt16] + ds [xx16
   ][contactpnt16]) * cos (alpha [xx16])) + ((fn [xx16][contactpnt16] + dn [x
   x16][contactpnt16]) * sin (alpha [xx16])));
3693.
3694.    yforce [xx16][contactpnt16] = (yforce [qq16][contactpnt16] * -1);
3695.    xforce [xx16][contactpnt16] = (xforce [qq16][contactpnt16] * -1);
3696.
3697.    fxsum [xx16][contactpnt16] = xforce [xx16][contactpnt16];
3698.    x [xx16] = fxsum [xx16][contactpnt16];
3699.    resultfx [xx16] += x[xx16];
3700.
3701.
3702.    fysum [xx16][contactpnt16] = yforce [xx16][contactpnt16] + p2 ;
3703.    y [xx16] = fysum [xx16][contactpnt16];
3704.    resultfy [xx16] += y[xx16];
3705.
3706.
3707.    msum [xx16][contactpnt16] = (yforce [xx16][contactpnt16]* ( rightiso
   sceles [qq16][1][0] - rightisosceles [xx16][0][0]) -
```

```
    xforce [xx16][contactpnt16] * ( rightisosceles [qq16][1][1] -
    rightisosceles [xx16][0][1]));
3708.    m [xx16] = msum [xx16][contactpnt16];
3709.    resultm [xx16] += m[xx16];
3710.
3711.
3712.    udoty [xx16][contactpnt16]=  ((fysum [xx16][contactpnt16] * dtime_in
   crement)* 1000/ dmass); // mm/sec
3713.    udotysum [xx16] = udoty [xx16][contactpnt16];
3714.    resultudoty [xx16] += udotysum [xx16];
3715.
3716.
3717.    udotx [xx16][contactpnt16] =  ((fxsum [xx16][contactpnt16]* dtime_in
   crement)*1000/ dmass); //mm/sec
3718.    udotxsum [xx16] = udotx [xx16][contactpnt16];
3719.    resultudotx [xx16] += udotxsum [xx16];
3720.
3721.
3722.    thetadot [xx16][contactpnt16] = ((msum [xx16][contactpnt16] * dtime_
   increment)*1000/ ii); //1/s
3723.    thetadotsum [xx16] = thetadot [xx16][contactpnt16];
3724.    resultthetadot [xx16] += thetadotsum [xx16];
3725.
3726.
3727.    deluy [xx16][contactpnt16] = udoty [xx16][contactpnt16] * dtime_incr
   ement;
3728.    deluysum [xx16] = deluy [xx16][contactpnt16];
3729.    resultdeluy [xx16] += deluysum [xx16];
3730.
3731.
3732.    delux [xx16][contactpnt16] = udotx [xx16][contactpnt16] * dtime_incr
   ement;
3733.    deluxsum [xx16] = delux [xx16][contactpnt16];
3734.    resultdelux [xx16] += deluxsum [xx16];
3735.
3736.
3737.    deltheta [xx16][contactpnt16] = thetadot [xx16][contactpnt16] * dtim
   e_increment;
3738.    delthetasum [xx16] = deltheta [xx16][contactpnt16];
3739.    resultdeltheta [xx16] += delthetasum [xx16];
3740.
3741.
3742.    uy [xx16][contactpnt16] = deluy [xx16][contactpnt16];
3743.    uysum [xx16] = uy [xx16][contactpnt16];
3744.    resultuy [xx16]+= uysum [xx16];
3745.
3746.
3747.    ux [xx16][contactpnt16] = delux [xx16][contactpnt16];
3748.    uxsum [xx16] = ux [xx16][contactpnt16];
3749.    resultux [xx16] += uxsum [xx16];
3750.
3751.
```

```
    theta [xx16][contactpnt16] = deltheta [xx16][contactpnt16];
    thetasum [xx16] = theta [xx16][contactpnt16];
    resulttheta [xx16] += thetasum [xx16];
    alpha [xx16] = alpha1 + resulttheta [xx16];


    }

    }

    }

    next16:
    ;}

    ;}

    display:


     a_file<< dmass;

    return 0;
    }
```