ABSTRACT

DEM SIMULATED FLOOR PRESSURE INDUCED BY A GRANULAR COLUMN

by
Shawn A. Chester

At the end of the 19th century, H. A. Janssen discovered that the bottom floor pressure in a cylindrical container of granular material asymptotes exponentially to a value less than the weight of the material i.e., the pressure becomes independent of the fill height of the column. This phenomenon is investigated using discrete element simulations of inelastic, frictional spheres in a cylindrical vessel having a particle-to-cylinder diameter ratio at approximately 13.3 or 26.6, with varying bed heights in both cases. The axial pressure profile and the load experienced by a piston that is supporting the granular column are computed. In order to activate frictional forces at the wall contacts either the piston (or equivalently the cylinder wall), is slowly displaced at a rate so as to maintain quasi-static conditions. Various combinations of wall and inter-particle friction coefficients are examined. The simulated behavior of the load vs. fill level was found to fit well to the functional form of Janssen's theory. Moreover, quantitative comparisons are in agreement with experimental measurements from the literature. Results are critically discussed in the framework of the assumptions implicit in Janssen's theory.

DEM SIMULATED FLOOR PRESSURE INDUCED BY A GRANULAR COLUMN

by
Shawn A. Chester

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Mechanical Engineering

Department of Mechanical Engineering

August 2006

APPROVAL PAGE

DEM SIMULATED FLOOR PRESSURE INDUCED BY A GRANULAR COLUMN

Shawn A. Chester

Dr. Anthony Rosato, Thesis Advisor — Date
Professor of Mechanical Engineering, NJIT

Dr. Pushpendra Singh, Committee Member — Date
Professor of Mechanical Engineering, NJIT

Dr. Ian Fischer, Committee Member — Date
Professor of Mechanical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Shawn A. Chester

Degree: Master of Science

Date: August 2006

Undergraduate and Graduate Education:

- Master of Science in Mechanical Engineering
  New Jersey Institute of Technology, Newark, NJ, 2006

- Bacholor of Science in Mechanical Engineering
  New Jersey Institute of Technology, Newark, NJ, 2005

Major: Mechanical Engineering

Presentations and Publications:

Shawn A. Chester, Anthony D. Rosato, Otis R. Walton,
"Discrete Element Simulations of Floor Pressure due to a Granular Material in a Cylindrical Vessel", the 5th World Congress on Particle Technology, Orlando FL, USA, 2006.

Shawn A. Chester, Anthony D. Rosato,
"Discrete Element Simulations of Floor Pressure of a Granular Material In a Cylindrical Vessel", the Second Conference on Frontiers in Applied and Computational Mathematics, Newark, NJ, USA, 2005.

Shawn A. Chester, Anthony D. Rosato,
"Discrete Element Simulations of Floor Pressure of a Granular Material In a Cylindrical Vessel", the Sigma Xi Annual Meeting and Student Research Conference, Montreal, Quebec, Canada, 2004.

To my parents.

ACKNOWLEDGEMENT

I would like to express my sincere appreciation to my advisor Dr. Anthony Rosato, for guidance and support throughout this research. Special thanks are given to Dr. Pushpendra Singh and Dr. Ian Fischer for their active participation in my thesis committee. The author is also thankful to Dr. Otis Walton for helpful discussions, along with Dr. David Horntrop and Dr. Dennis Blackmore for helpful comments.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

TABLE OF FIGURES

TABLE OF FIGURES
(Continued)

Figure **Page**

TABLE OF FIGURES
(Continued)

Figure **Page**

TABLE OF FIGURES
(Continued)

Figure **Page**

LIST OF SYMBOLS

d	Particle diameter
D	Cylinder diameter
ϕ	$\phi \equiv D/d$, Diameter ratio
ν	Solids fraction
σ_{rr}	Radial stress
σ_{zz}	Vertical stress
τ_w	Wall shear stress
K	Janssen constant
μ_w	Limiting wall-particle friction coefficient
μ_p	Limiting particle-particle friction coefficient
ρ	Particle mass density
$\hat{z}$	Height, measured from piston upward
h	$h \equiv \hat{z}/d$, Normalized measure of height in particle diameters
$\underset{\sim}{F}_N$	Normal contact force vector
$\underset{\sim}{F}_T$	Tangential contact force vector
K_1	Normal loading stiffness
K_2	Normal unloading stiffness
α	Relative overlap between contacting bodies
$\underset{\sim}{x}_i$	Position vector of particle i
Δt	Time step

$\underset{\sim}{F}_{ij}$	Net force vector acting on particle *i* due to particle *j*
$\underset{\sim}{e}_r$	Unit vector from cylinder axis to contact point of two particles
$\underset{\approx}{P}$	Stress tensor
P_{zz}	Axial component of the stress tensor
P_{rr}	Radial component of the stress tensor
K_A	Janssen's constant for the active case
K_P	Janssen's constant for the passive case
F_z	Actual computed piston load in Newton's
F^*	$F^* \equiv \frac{6F_z}{\rho g \pi d^3}$, Piston force normalized by a single particle weight
$\hat{F}$	$\hat{F} \equiv \frac{F_z}{m_{total} g}$ Piston force as a fraction of the material weight

CHAPTER 1

INTRODUCTION AND LITERATURE SURVEY

1.1 Overview

Near the end of the 19th century, H.A. Janssen [1, 2] proposed a model to predict the vertical stress profile in a container of granular materials. A feature of his prediction is that the stress on the floor asymptotes to a value that is less than the weight of the confined material. Despite the fact that some of the assumptions in the model are not valid in practical situations, Janssen's model is heavily used in industry to design silos, bunkers, and other storage containment vessels because the general trends of Janssen's predictions have been corroborated for appropriately initialized granular beds. It is well known that under dynamic unloading, in converging hoppers, and for asymmetric flows, stresses can exceed the values predicted by Janssen's theory by large factors. Thus real silo designs require appropriate modifications at specific locations.

As a consequence, there has been renewed interest in Janssen's theory [2-14] and also in applications where the direction of the wall motion is opposite to the assumptions of Janssen (i.e., a piston pushing a granular material up a vertical column) [7, 8]. The underlying motivation in many of the recent studies reported in the literature [3-9, 11-14] has been to identify which of Janssen's assumptions may not be valid so that an improved model can be developed. Clearly, differences between predictions of the theory and experimental measurements can occur under conditions where assumptions of the theory do not represent the real distribution of stress in the material. Unfortunately, the evolution of stress in a slowly deformed granular material is extremely complex and not

well-understood. In an attempt to bridge the knowledge gap here, simulations have been done that involve the numerical solution of the equations of motion of systems of interacting, dissipative particles (eg., discrete element simulations) under various loading conditions. However, discrepancies between discrete element simulation results and the measured behavior of real materials can occur when simplifying assumptions used in the simulations inadvertently neglect important features of the particle-scale physics. In this thesis, extensive discrete element simulation studies are done to model the pressure distribution in a column of granular material, with an emphasis on the load on the supporting floor. The results, which are presented within the context of Janssen's theory, show good general agreement of the numerical data with the theoretical predictions and experiments in the literature.

1.2 Janssen's Theory

The physical geometry of the cylinder having a diameter D is depicted in Figure 1.1, where the origin is at the top with the z-coordinate positive downward. Janssen's model is based on the following assumptions [2]:

1. Frictional forces are fully 'activated' at the wall.
2. The material is cohesionless and the bulk solids fraction ν is uniform throughout the bed.
3. The radial σ_{rr} and vertical σ_{zz} stresses are principal and exhibit a constant ratio, such that,
$$\sigma_{rr} = K\sigma_{zz} \tag{1.1}$$
4. The stressed are uniform across any horizontal cross section.

To begin, consider a granular fill consisting of uniform spheres of mass density ρ whose limiting (i.e., Coulomb) wall friction coefficient is denoted by μ_w. Following the method of Janssen, the governing differential equation is easily derived from a force balance on a differential slice of thickness *dz*, whose midplane lies at *z* below the surface of the packed bed. The forces acting on this slice are shown in Figure 1.2.

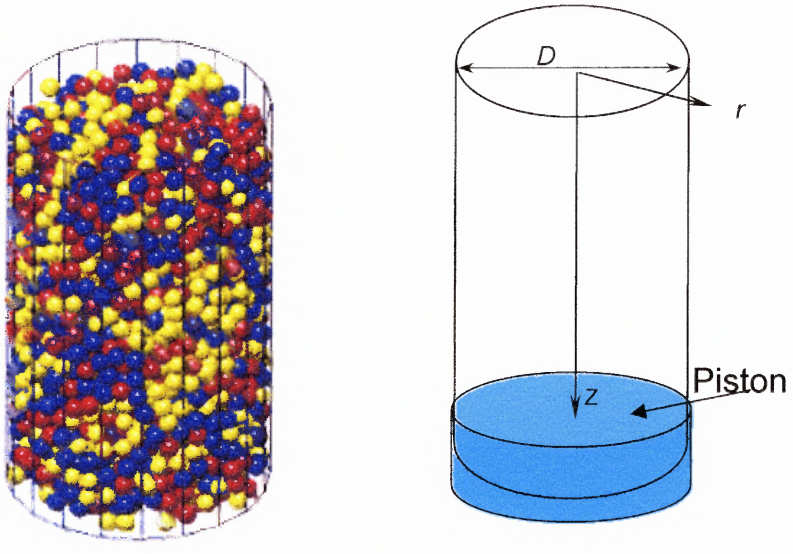

Figure 1.1 Geometry of the simulated system.

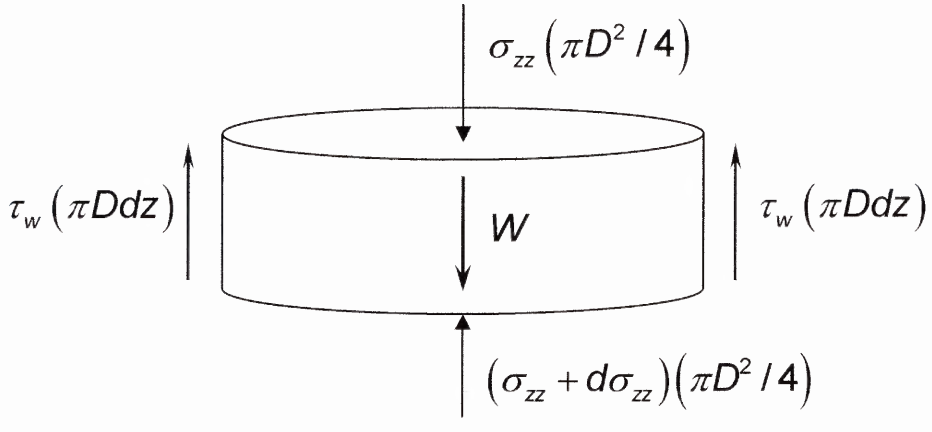

Figure 1.2 Forces acting on a horizontal differential slice of thickness *dz* at a depth *z* below the surface of the packed bed.

In Figure 1.2 the wall shear stress is shown acting upwards since the downward movement of the piston, or upward motion of the wall, causes the material to slide in the positive z-direction. A force balance in the z-direction yields Equation (1.2).

$$\sigma_{zz}\left(\frac{\pi D^2}{4}\right)+\nu\rho g dz\left(\frac{\pi D^2}{4}\right)=\left(\sigma_{zz}+d\sigma_{zz}\right)\left(\frac{\pi D^2}{4}\right)+\tau_w \pi D dz \tag{1.2}$$

By applying the assumptions that the friction at the wall is fully mobilized at the Coulomb limit μ_w, and the proportionality $\sigma_{rr}=K\sigma_{zz}$ between the stresses, the differential equation governing the pressure σ_{zz} is given by,

$$\frac{d\sigma_{zz}}{dz}+\frac{4\mu K}{D}\sigma_{zz}=\nu\rho g \tag{1.3}$$

The latter first-order ordinary differential equation is solved for σ_{zz} using standard methods to yield,

$$\sigma_{zz}(z)=\frac{\nu\rho g}{\beta}\left[1-e^{-\beta z}\right] \tag{1.4}$$

where $\beta=4\mu K/D$. Upon setting $z = H$ (the fill height of the material) in Equation (1.4) and multiplying the result by the piston area, the normal force on the piston is given by,

$$F(H)=\frac{\nu\rho g\pi D^2}{4\beta}\left[1-e^{-\beta H}\right] \tag{1.5}$$

Equation (1.5) predicts that *F(H)* asymptotes to a value that is less than the actual weight of the material in the cylinder – a behavior that is quite different from ordinary fluids.

In the case where the piston is moving upwards, a similar equation can be derived following the procedure outlined above, with the only difference being the direction of the wall shear stress, i.e.,

$$F(H) = \frac{\nu\rho g\pi D^2}{4\beta}\left[e^{\beta H} - 1\right] \tag{1.6}$$

However, the prediction is quite different as the piston load increases exponentially with *H*. Although the focus of this thesis is the phenomenon described by Equation (1.5), preliminary simulations had been done for the case when the piston is displaced upwards (i.e., Equation (1.6)). Here, very large force fluctuations in the simulated piston load were found so that results were not definitive. Unfortunately, the additional simulations needed to obtain consistent behavior could not be carried out within a reasonable time frame on the university's computing systems.

A derivation for Janssen's constant *K* is easily produced via a Mohr-Coulomb failure analysis. A brief sketch is given here for completeness, while further details can be found in [2]. The Coulomb yield criterion for a cohesionless material takes the form $\tau = \mu\sigma + c$, where τ is the shear stress, μ is the coefficient of friction, σ is the normal stress, and c is the cohesion. This criterion imposes a limit on the shear stress that can exist within a granular material. Used in conjunction with Mohr's circle, it provides the basis for the Mohr-Coulomb failure analysis. Two cases arise with Mohr's circle, the active case and the passive case. In the active case the axial stress is greater than the radial stress, while in the passive case the radial stress is larger than the axial stress. Mohr's circle for the active state, along with the Coulomb yield criterion, denoted by *IYL* (Internal Yield Locus), is shown in Figure 1.3 below.

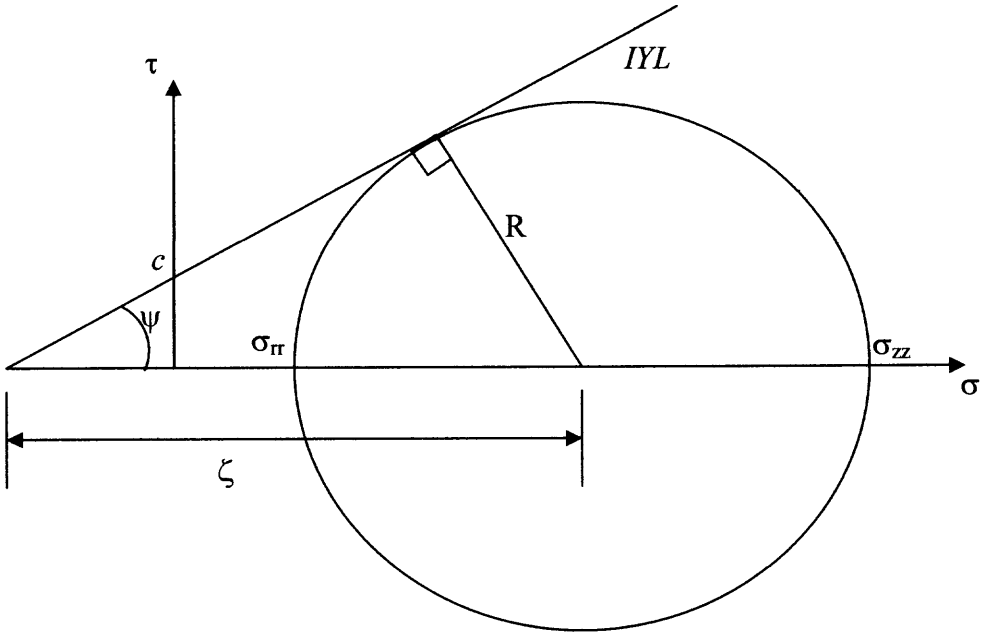

Figure 1.3 Mohr's circle for the stresses in the active state.

For the case shown in Figure 1.3 a geometrical analysis from Mohr's circle will show that the radial and axial stresses are given in Equation (1.7) and Equation (1.8), respectively, where ψ is the internal angle of friction ($\psi = \tan^{-1}\mu$), and c is the cohesion.

$$\sigma_{rr} = \zeta - R - c\cot\psi \tag{1.7}$$

$$\sigma_{zz} = \zeta + R - c\cot\psi \tag{1.8}$$

Additionally, from Mohr's circle, one can find that $R = \zeta\sin\psi$. Then by eliminating ζ and R by combining the equations for the axial and radial stress, and assuming a cohesionless material (i.e., $c \to 0$), one will arrive at Equation (1.9), which relates the radial and axial stresses.

$$\sigma_{rr} = \frac{1-\sin\psi}{1+\sin\psi}\sigma_{zz} \tag{1.9}$$

Thus, for the active case Janssen's constant is found to be $K_A = \frac{1-\sin\psi}{1+\sin\psi}$. A similar analysis will show that for the passive case (in which $\sigma_{rr} > \sigma_{zz}$), one finds that $K_P = \frac{1+\sin\psi}{1-\sin\psi}$.

1.3 Review of Published Literature

H.A. Janssen [1] (English translation is given by [15]) provided the first theoretical model to predict the vertical pressure profile in a granular column. Since 1895, many investigations have attempted to further understand this phenomenon, and in fact, more than 330 published papers have cited Janssen since 1980 [15]. Table 1.1 below gives a brief overview the literature that is most relevant to this thesis.

In his treatment of Janssen's equation, Nedderman [2] points out two questionable assumptions of the theory. The first is that the axial σ_{zz} and radial σ_{rr} stresses are principal stresses as depicted on Figure 1.3, and proportional according to Equation (1.1). It can be shown very easily that this assumption is incorrect because there is no shear stress acting on a principal stress plane, which contradicts another of Janssen's model assumptions, i.e., that frictional forces are fully activated at the wall so that $\tau_w = \mu_w \sigma_{rr}$. The second assumption is that the stresses across any horizontal cross section are uniform. The analysis here is much more complex and the reader is referred to [2] for the details.

Table 1.1 Overview of Selected Published Material

First Author	*Year*	*Methods*
Nedderman [2]	1992	Analytical
Kolb [16]	1999	Experiment
Vanel [13]	1999	Experiment
Vanel [14]	2000	Experiment
Marconi [9]	2000	Analytical
Ovarlez [12]	2001	Experiment
Ovarlez [11]	2003	Experiment
Landry [8]	2003	Simulation
Bertho [4]	2003	Experiment
Arroyo-Cetto [3]	2003	Experiment
Landry [7]	2004	Simulation
Landry [6]	2004	Simulation
Walton [17]	2004	Experiment
Bratberg [5]	2005	Experiment

In 1999, Kolb et al. [16] performed a series of experiments on pushing a two-dimensional granular column upwards with a piston. They reported on the resistance force encountered by the piston for a wide range of parameters such as the piston velocity, and the particular granular fill material used. The main conclusions drawn concerns the variability of the results and the complex features depending on the experimental conditions. The authors were unable to provide a clear description of a mechanism for the rather large fluctuations in the measured piston force.

In 1999, Vanel et al. [13] reported on the static pressure at the bottom of a granular column. Two types of experiments were performed. The first is referred to as a "descent experiment", and the second is referred to as a "tapping experiment". A descent experiment is intended to probe the effect of a series of downward motions of the piston. A tapping experiment is designed to probe the effect of a changing granular density. They report that for every set of data, the Janssen model systematically underestimated the results from the experiments. The authors propose a two parameter model to account for a hydrostatic-like region located at the top of the column, while the remainder of the column behaves according to Janssen's model. Their two-parameter model equations for the apparent mass M_a, expressed in units of fill mass M, are

$$\begin{aligned} M_a &= M, \ for\ M \le M_0 \\ M_a &= M_0 + M_\infty^C \left[1 - \exp\left(-\frac{M - M_0}{M_\infty^C} \right) \right], for\ M > M_0 \end{aligned} \tag{1.10}$$

where the fitting parameter M_0 represents the mass of the hydrostatic zone at the top of the column. Figure 1.4 depicts the parameters of the model. In Equation (1.10), M_∞^C is a fitting parameter that represents the difference between the 'saturation mass' and M_0.

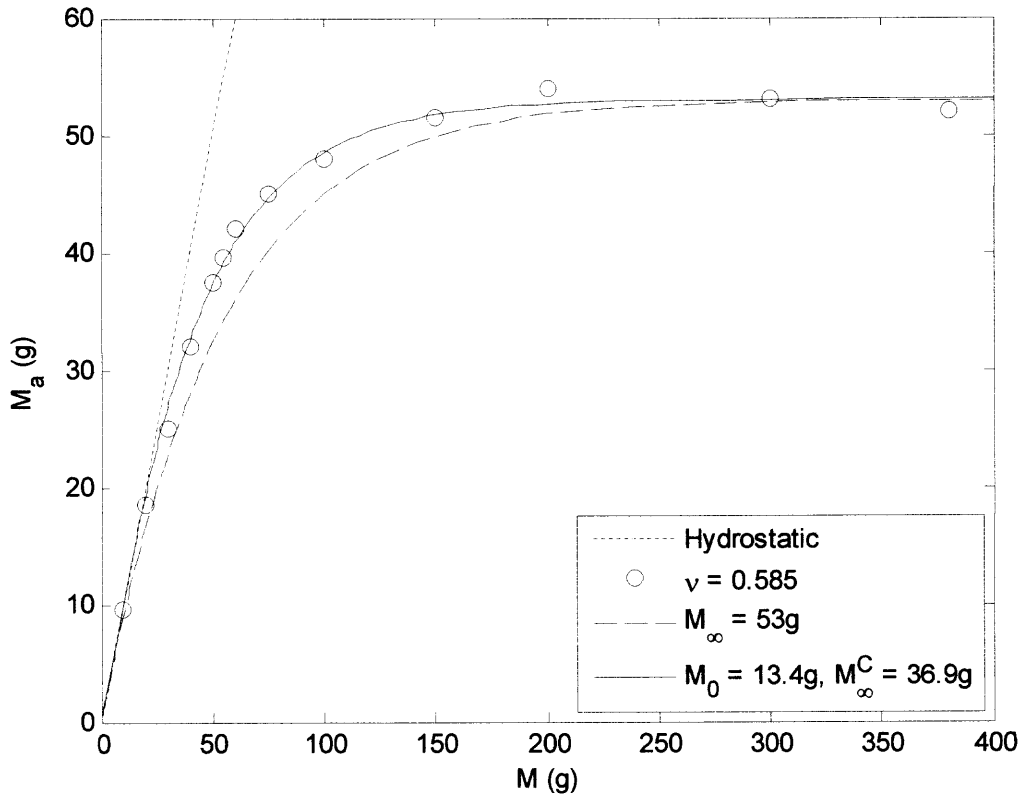

Figure 1.4 Apparent mass M_a as a function of the filling mass M for a packing fraction 0.585 ± 0.005. The straight dotted line indicates a hydrostatic behavior; the dashed curve is a fit with Janssen's prediction; the solid curve is a fit with the two parameter model. Redrawn from [13].

In 2000, Vanel et al. [14] performed experiments to measure the mean pressure at the bottom of a column of cohesionless granular material. Here, Janssen's form was found to be satisfactory and in accordance to their earlier experiments [13]. In addition, they carried out separate experiments by placing an overload on the top surface of the packed bed. Fore this case, the data could not be fit to Janssen's model. Rather, a simple hyperbolic form having two fitting parameters was able to reproduce the experimentally measured stress response of the material.

In 2000, Marconi et al. [9] analytically studied the mechanisms underpinning Janssen's law. They consider a q-model [18] for an assembly of particles on a two-dimensional lattice which simply verified the coupling between the vertical and lateral forces as a necessary feature for Janssen's model. However, this simply restates the constitutive assumption that axial pressure is transferred into the lateral (or radial) direction in a granular material. However, their model did suggest that a non-uniform pressure profile across a horizontal plane results in a deviation from Janssen's model.

In 2001, Ovarlez et al. [12] reported on the rheology of a granular material slowly driven in a cylindrical container. They observed blocking enhancement, aging, and dynamical hardening effects at slow driving velocities. Their analysis showed that the properties exhibited are due to the solid on solid friction at the particle to wall contacts. However, a quantitative analysis in the context of Janssen's model indicated that the dynamical restructuring effects in the bulk cannot be excluded. Additionally, they report a very strong dependence on the relative humidity. In a subsequent paper, the authors [11] report on extended experiments with the same configuration. Here, the piston was translated downward at a constant velocity $V_0 = 1.5\mu m/s$. The fill material consisted of monodisperse particles of diameter $d = 1.5mm$ and thus this corresponded to a non-dimensional piston velocity $V_0 = 0.001\, d/s$ (particle diameters per second). Results conformed with Janssen's model. However, when an overload was placed on the top of the packed bed, Janssen's model failed, as was also found in [14].

Landry et al. [8] reported on discrete element simulations of static granular packings confined to a cylindrical container, in which the vertical stress profile along the height of the granular fill, and the distribution of forces within the static column are computed. They found that the majority of the particle - wall contact forces were at or near the Coulomb failure limit, while particle - particle contact forces in the bulk were far from this limit. Their results confirm the experimental findings of Vanel et al. [13] regarding the hydrostatic region at the top of bed. They further demonstrate from their simulated data that this hydrostatic region is principally due to the forces at the wall being far from the Coulomb yield criteria.

In 2003, Bertho et al. [4] reports on experiments measuring the apparent mass of a granular packing inside of a cylinder that was translated upward. Using a diameter ratio $\phi = 15$ (ϕ is the ratio of the cylinder to particle diameter), they found that Janssen's model was valid for a broad range of velocities, up to several centimeters per second. This is shown in Figure 1.5, in which Bertho's original data is redrawn. Most interesting, is that measurements taken after the cylinder motion was halted contained more dispersion, thereby yielding a poorer fit to Janssen's model. The authors further report that the average solids fraction of the packing remained fairly constant during the piston motion.

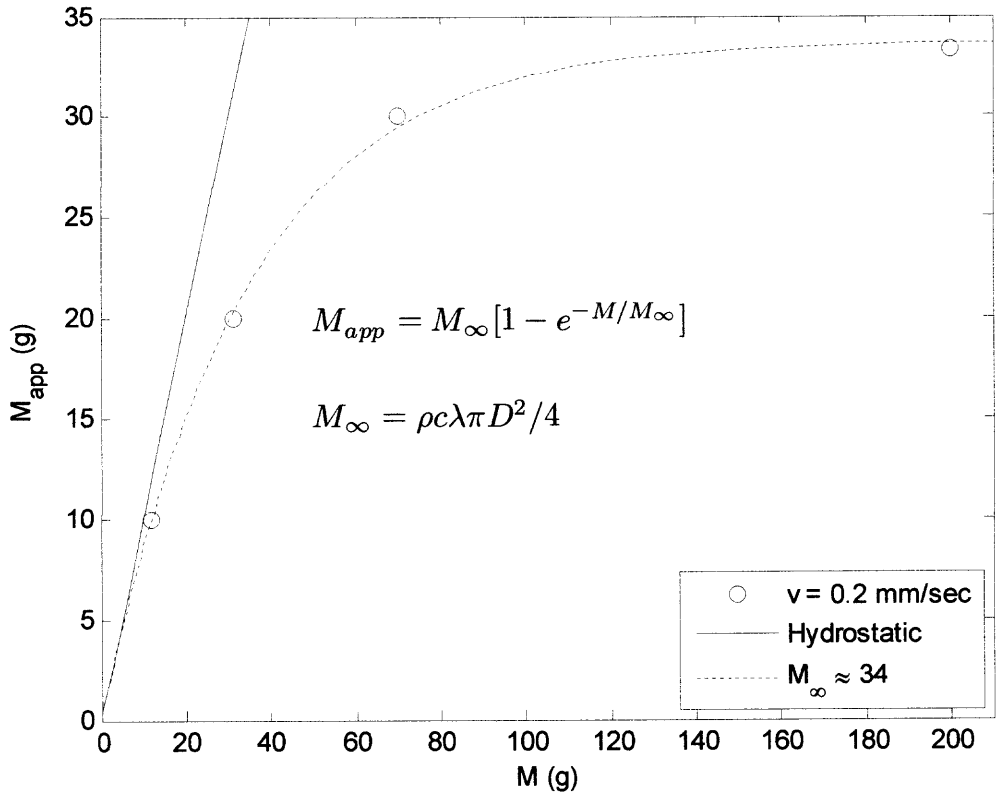

Figure 1.5 Apparent mass M_{app} as a function of the total mass M for a cylinder velocity of 0.2mm/sec (redrawn from [4]).

Arroyo-Cetto et al. [3] described their experiments measuring the force required to push a granular column upwards within a cylinder. To do this, the piston was fixed and the cylinder walls were translated downwards. They reported that this force rapidly increased with fill height, in accordance with Janssen's theory as per the form of Equation (1.6). In addition they found that the force also increases with the velocity of the cylindrical container. However, large fluctuations in the measurements were observed (see Figure 2 in [3]) analogous to what was seen in the experiments of Kolb [16].

Recently, Landry et al. [7] reported on discrete element simulations for both two dimensional and three dimensional static systems. For the two dimensional simulations, they found a clear hydrostatic region at the top of the packing followed by a region not well described by Janssen's functional form. More significantly, their results indicate that a full three dimensional model is needed to reproduce Janssen's prediction. In a sequent paper, the authors [6] present simulation results in which the cylinder is translated upward at a uniform velocity. Computations of the vertical stress profile, the Coulomb criterion at the walls (i.e., the value of the wall friction force against the Coulomb limit), and the packing structure are described. As in experiments [13], the initial 'poured' configuration exhibited a hydrostatic region at the top of the packing that then crosses over to a Janssen like behavior. As the cylinder is moved upward the vertical pressure rapidly changes, and then reaches a steady state value until the motion is stopped. The system was then allowed to relax (or equilibrate) and the vertical pressure computed and fit to Janssen's model. For slower cylinder velocities, the agreement of the static (relaxed) pressure profiles deviated from the model predictions. This was attributed to a reduction in the number of particle-particle and particle-wall contacts at the Coulomb limit. However, it is not clear if this is the case since their force model applies friction to the center of the particle rather than the contact surface.

Walton carried our a series of experiments using 3 mm glass beads in a 4 cm acrylic tube with a diameter ratio $\phi = 13.3$. The results, reported in [17], verified the exponential functional forms of Janssen's model (see Equations (1.5) and (1.6)) for the piston force versus fill height. Figure 1.6 summarizes the main findings of the experiments. The most interesting aspect of the work, shown graphically in Figure 1.7, is

the linear variation of the product $\mu\sigma_{rr}$ (i.e., force required to move the cylinder) against applied axial stress σ_{zz}. However, the line does not intersect the origin in accordance with Janssen's constitutive assumption given by Equation (1.1).

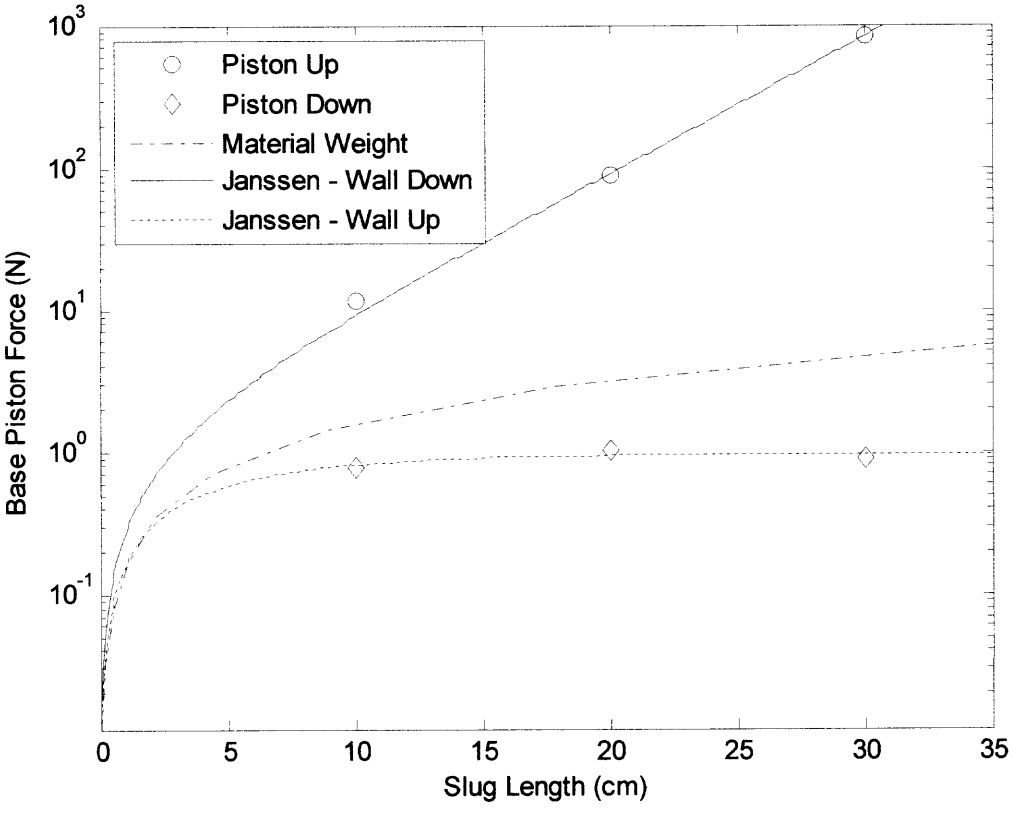

Figure 1.6 Experimental results reported by Walton verifying the functional form of Janssen's model (redrawn from [17]).

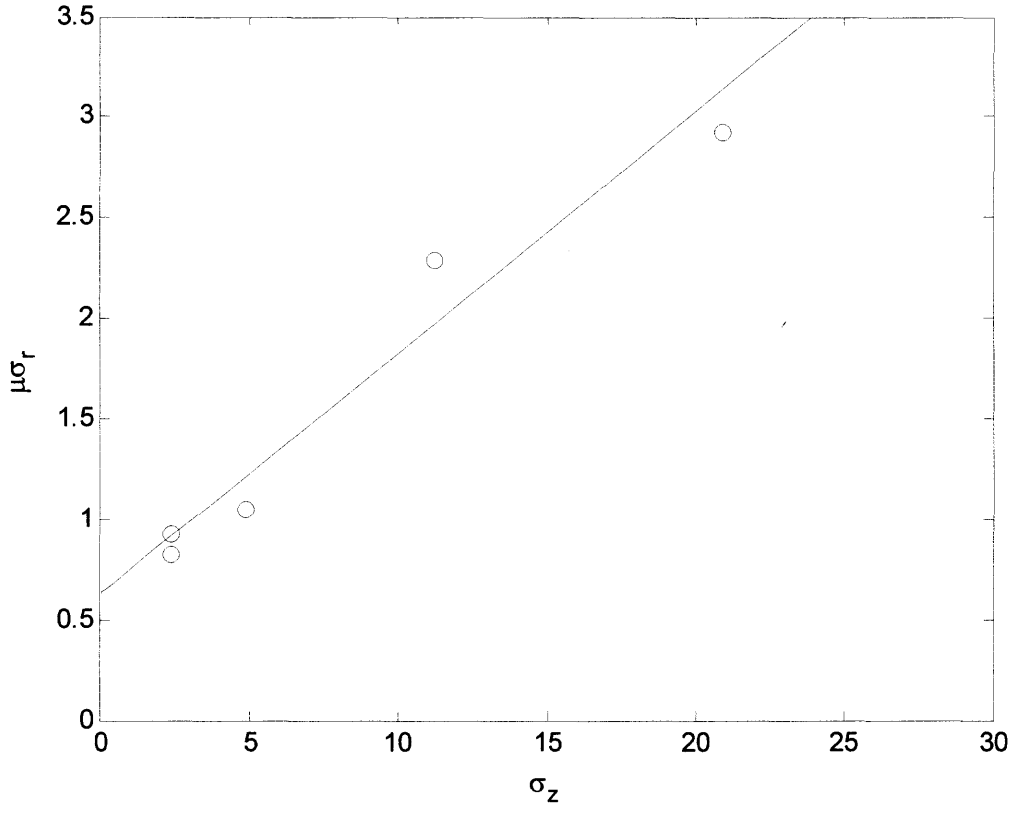

Figure 1.7 Experimental results reported by Walton invalidating the constitutive assumption in Janssen's model (redrawn from [17]).

In 2005, Bratberg et al. [5] performed a series of experiments with small cylinder aspect ratios to test the validity of Janssen's model in this configuration. They used ratios between $\phi = 1.9$ and $\phi = 3.5$ that were well below what had been done previously. The apparent mass and the height of the granular fill was measured for both dynamic and static situations. They reported that for an upward motion of the piston, the apparent mass increased monotonically until a slip-stick regime was reached. Additionally, for a downward motion of the piston, a steady state apparent mass was achieved after an almost linear decrease in the apparent mass. However, Janssen's model did not satisfactory fit the results with only a single parameter. More importantly they observed

that the exponential decay characteristic of Janssen's model was not observed for a column so narrow that each particle had only two near neighbors. This was attributed to the coupled rotations of the particles in this configuration, an effect which is sited as contrary to a necessary 'rotational frustration' to produce the exponential decay.

1.4 Objective

In this thesis, the results of a particle-level numerical investigation of the load on a piston supporting a monodisperse granular material consisting of frictional, inelastic spheres of diameter d, within a cylindrical vessel of diameter D is presented. This is done via discrete element simulations where wall friction in activated by moving the piston down very slowly or displacing the cylinder upwards. A review of the literature indicated that computer simulated investigations have considered diameter-ratios $\phi \equiv D/d$ ranging from 20 to 40 with wall friction activated by the motion of the cylinder walls. In comparison, relatively small diameter-ratios (i.e. $\phi = 13.3$ and $\phi = 26.6$) are selected with the goal of comparing results with experiments and Janssen's model predictions. Results are discussed within the framework of the assumptions implicit in Janssen's theory.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 describes the simulation method and its application. Chapter 3 presents the simulated results and a discussion of their significance in the context of Janssen's theory. Chapter 4 contains the conclusions and recommendations for further work.

CHAPTER 2

DISCRIPTION OF THE DISCRETE ELEMENT METHOD SIMULATION

2.1 Background

The discrete element method is a computational scheme in which the equations of motion of a system of dissipative, interacting particles are numerically integrated to determine the phase space (i.e., positions and velocities). The technique was introduced by Cudall and Strack in the 1970's [19] and has been employed extensively for granular flows (see for example [20-34]). The numerical methods are identical to those used by molecular dynamics simulations. However, engineering mechanics models of contact forces are used in place of intermolecular force relations.

2.2 Overview

The DEM code used in this thesis employs the inelastic, frictional soft sphere models of Walton and Braun [33]. This code was first developed for the purpose of simulating granular shear. Since its inception, the DEM code has undergone many modifications. The specific DEM code used was previously designed for a rectangular geometry with an oscillating floor. Therefore, significant modifications were required to obtain a cylindrical geometry with a moving piston and moving cylinder wall. The reader is referred to Appendix A for details on the modifications to the DEM code. In Appendix A the reader will find excerpts from the code along with a description of the purpose, variables, etc. for that section of code.

General flow of the DEM code:

1. Read the input file, initialize simulation parameters
2. Assign particle positions
3. Update linked list of near neighbors
4. Compute inter-particle forces
5. Integrate to obtain the positions and velocities of the particles
6. Repeat from step 3 until the simulation is finished

Initially, the code will read the input file, *i3ds* (Sample input file shown in Appendix D), using the subroutine *datain.f* (All subroutines are described in section 2.3 below). If specified in the input file that the simulation is a continuation of a previous simulation, the subroutine *dumpread.f* will read the dumpfile, *d3ds1000*, from the previous simulation. The dumpfile, *d3ds1000*, is just like an input file, except that it contains information such as particle coordinates and velocities from where the previous simulation left off.

Following the input of simulations parameters, the subroutine *init.f* will initialize the simulation by assigning initial coordinates to all of the particles. In addition *init.f* will call the subroutine *bound.f* to initialize the boundaries of the simulation. For this thesis, the boundary conditions consist of a cylindrical container of infinite length with a floor, i.e., the piston. In this thesis, both the piston and the cylindrical container posses the ability to translate along the direction of the cylindrical axis. Furthermore, all particle initial positions are randomly generated in this thesis. The subroutine *findrad.f* is then used to assign the particles a dimension. An overlap detection routine is used to confirm

that none of the simulation particles are overlapping each other or a boundary when the simulation starts. Lastly, *init.f* will call two more subroutines, *initcum1.f* and *initcum2.f* both of which will initialize the short and long term cumulative averages, respectively.

After the particle positions and boundaries are initialized, the simulation can begin to loop. Subsequently, the subroutine *update.f* will create the linked list of near neighbors. The subroutine *update.f* is called henceforth only when the cumulative maximum displacement of a particle from each time step is greater than half the search radius for near neighbors. In addition each time *update.f* is called another subroutine *deletem.f* is called to remove particles from the linked list that have traveled outside of the search radius for that particle. More information on the linked list is given in Section 2.4 below.

The next step in the simulation loop is in preparation of the integration routines. The subroutine *initstep.f* is called to initialize parameters for the subroutines that compute the inter-particle forces and the integration routines. Subsequently, the subroutine *forces.f* computes the inter-particle forces due to contacts using the force model described in Section 2.5 below. Following the computation of the contact forces, the subroutine *integ1.f* computes the velocities at the current time step. Next the subroutine *diagnos2.f* computes the diagnostics such as the potential component of the stress tensor at the current time step for the simulation. At this point a test for writing data to file is performed. Data is written to file at user input time intervals *dtout*. If the change in time since the previous output is at or just above the value of *dtout*, then the subroutine *datasav2.f* is called to perform the write to file. Lastly, the subroutine *integ2.f* computes the velocity at the next half time step and the position at the next half time step, and also

increments the time step to march the simulation forward in time. Then the process begins all over again from the start of the integration routine.

The remaining Sections of Chapter 2 are as follows: Section 2.3 Description of Subroutines, 2.4 Contact Detection and the Linked List, 2.5 Force Model, 2.6 Integration Method and Time Step, and lastly 2.7 Diagnostic Computations.

2.3 Description of Subroutines

The following subroutines are the building blocks for the DEM code.

- *bound.f*

  bound.f initializes the boundary particles used in the simulation.

- *datain.f*

  datain.f reads the input file.

- *datasav2.f*

  datasav2.f writes the output to a corresponding output file at a user defined frequency *dtout*.

- *deletem.f*

  deletem.f removes particle *j* from the linked list of particle *i*, if particle *j* has moved beyond a specified distance of particle *i*.

- *diagnos2.f*

  diagnos2.f is used to compute the simulation diagnostics.

- *dumpread.f*

  dumpread.f is used when restarting the simulation. All the information required to restart the simulation from a previous simulation is read in by this subroutine.

- *findrad.f*

  findrad.f is used to assign the particle radii.

- *forces.f*

  forces.f computes the inter-particle forces, along with the potential components of the stress tensor.

- *init.f*

  init.f initializes the simulation, i.e., time step, boundary dimensions, initial particle positions, particle masses, etc are initialized.

- *initcum1.f*

  initcum1.f initializes the short term variables.

- *initcum2.f*

  initcum2.f initializes the long term variables.

- *initstep.f*

  initstep.f initializes the variables used in the numerical integration.

- *integ1.f*

  integ1.f performs the numerical integration at the current time step.

- *integ2.f*

  integ2.f computes the coordinates of the particles at the end of the time step, along with the velocities at that same time.

- *update.f*

  update.f loops through all particles to create, or update, the linked list for particle *i*. Only those particles that fall within a specified distance of particle *i* are added to the list.

2.4 Contact Detection and the Linked List

Contact detection is a very simple process in which the geometry and position of particles are checked against each other. In essence, if the distance between two particle centers is greater than the sum of the particle radii, then no contact. If on the other hand, the distance between the particle centers is less than or equal to the sum of the particle radii, then the pair is contacting.

The detection of contacting pairs of particles is very important in the DEM simulations. However, the process is extremely expensive computationally to check for contact between all pairs in the simulation. Thus, a linked list is implemented to reduce the time required to check for contacts throughout the simulation. The linked list keeps track of all particles that fall within a near neighbor radius, also known as the *search* radius, of the particle under consideration. Therefore, when checking for contacting pairs, the algorithm only needs to check the particles that belong to the linked list of the particular particle, and not check all particles in the system. More detailed information about the linked list can be found in [35].

2.5 Force Model

A soft sphere approach is used in which collisions occur over a finite length of time. The contacting particles are allowed to slightly overlap and a linear stiffness in the force model computes a contact force in proportion to the amount of overlap. Any single particle may be contacting many other particles at the same time. The net force on the particle is simply the vector sum of all the forces applied to the particle.

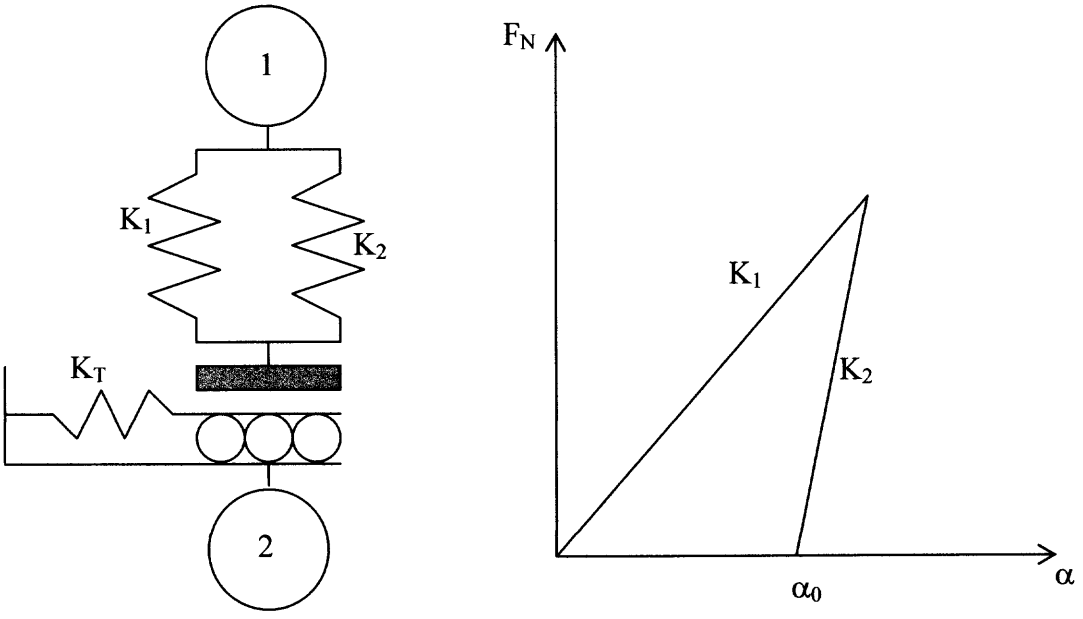

Figure 2.1 Partially latching spring force model.

The force model implemented is the partially latching spring model developed by Walton and Braun [33, 36-38] for an elastic-plastic material and shown pictorially in Figure 2.1. The normal force model operates by incorporating different normal stiffness for loading and unloading, K_1 and K_2, respectively as shown in Figure 2.1. As shown in Figure 2.1 the normal force between colliding particles is a function of the relative overlap α. The magnitude of the normal force is given by Equation (2.1). Where α_0 is the remaining overlap at which point the unloading force goes to zero due to plastic deformation of the contact. Additionally, the normal force acts along line connecting the particle centers. Further details are presented in [33, 36-38].

$$F_N = \begin{cases} K_1\alpha & \text{for loading} \\ K_2(\alpha - \alpha_0) & \text{for unloading} \end{cases} \tag{2.1}$$

This model leads to a constant coefficient of restitution e, which can be shown [39] to be given by Equation (2.2) for the partially latching spring model.

$$e = \sqrt{\frac{K_1}{K_2}} \tag{2.2}$$

Additionally, it can be shown through a Hertzian contact analysis [39] that a good estimate of the normal loading stiffness is given by Equation (2.3). In this equation, E is the elastic modulus, and ν is the Poisson ratio of the material. In addition, d is the diameter, m is the mass of a particle, and v_{max} is the maximum estimated impact velocity.

$$K_1 = \left(\frac{4E\sqrt{dv_{max}}}{15m\left(1-\nu^2\right)} \right)^{4/5} \tag{2.3}$$

The tangential force model also developed by Walton and Braun [33, 36-38] is patterned after Mindlin's theory [40] via a contact stiffness K_T that decreases with displacement until full sliding occurs at the friction limit. The tangential force is applied at the contact, thus allowing particles to rotate. The effective tangential stiffness K_T is given by Equation (2.4). Where F_T is the total tangential force, μ is the coefficient of friction, N is the total normal force, γ is a fixed parameter set to 1/3, and F_T^* is the loading reversal value. The loading reversal value is initially set to zero, and then subsequently set to the value of the total tangential force whenever the rate of change in magnitude changes from increasing to decreasing, or vice versa.

$$K_T = \begin{cases} K_0\left(1-\dfrac{F_T - F_T^*}{\mu N - F_T^*}\right)^{\gamma} & \text{for increasing } F_T \\ K_0\left(1-\dfrac{F_T^* - F_T}{\mu N + F_T^*}\right)^{\gamma} & \text{for decreasing } F_T \end{cases} \tag{2.4}$$

The value of K_0 is given by Equation (2.5) below, where τ is the ratio of normal to tangential stiffness, in this case set to 0.8.

$$K_0 = \begin{cases} \tau K_1 & \text{for loading} \\ \tau K_2 & \text{for unloading} \end{cases} \tag{2.5}$$

The tangential force $\underset{\sim}{F}_T$ is then computed according to Equation (2.8). The tangential force is given as the sum of perpendicular and parallel components as shown in Equation (2.6) and Equation (2.7), respectively. Where $\Delta S^{\perp}$ is the amount of relative surface displacement in the direction perpendicular to the old tangential force, and $\Delta S^{\parallel}$ is the amount of relative surface displacement in the direction parallel to the old tangential force. The model assumes that the relative displacements between time steps are relatively small. Lastly, the total tangential force is checked against the friction limit μN and scaled back, if necessary.

$$F_T^{\perp} = K_0 \Delta S^{\perp} \tag{2.6}$$

$$F_T^{\parallel} = F_T^{old} + K_T \Delta S^{\parallel} \tag{2.7}$$

$$\underset{\sim}{F}_T = \underset{\sim}{F}_T^{\parallel} + \underset{\sim}{F}_T^{\perp} \tag{2.8}$$

The net force $\underset{\sim}{F}_{ij}$ on particle i due to collision with particle j simply is the vector sum of the normal and tangential contact forces descried above. Therefore the net contact force, acting on the contact location, is given by,

$$\underset{\sim}{F}_{ij} = \underset{\sim}{F}_N + \underset{\sim}{F}_T \tag{2.9}$$

2.6 Integration Method and Time Step

Time in the simulations is stepped forward by discrete time steps on the order of $\Delta t \approx 3.7\times10^{-5}$ seconds. The time step is computed via Equation (2.10) which is derived from the finite contact time in the soft sphere model. Where e is the coefficient of restitution, K_1 is the normal loading stiffness, m is the mass of a particle, and n is the number of time steps per single collision. More detail on the derivation of the time step can be found in [33, 37, 38].

$$\Delta t = \frac{\pi e\sqrt{\frac{m}{2K_1}}}{n} = \frac{\pi e\sqrt{\frac{\pi d^3\rho}{12K_1}}}{n} \tag{2.10}$$

A Verlet leap-frog algorithm is used to numerically integrate the equations of motion for the particles. The translational equations of motion are shown below in Equation (2.11), where the superscripts refer to the time step. More details on the integration method derivation and algorithm can be found in [35]. The rotational equations of motion are analogous.

$$\begin{cases} \left.\begin{aligned} \underset{\sim}{v}_i^{t+1/2} &= \underset{\sim}{v}_i^{t-1/2} + \frac{\underset{\sim}{F}_i^t}{2m_i}\Delta t \\ \underset{\sim}{x}_i^{t+1} &= \underset{\sim}{x}_i^t + \underset{\sim}{v}_i^{t+1/2}\Delta t \end{aligned}\right\} t>0 \\ \underset{\sim}{v}_i^{-1/2} = \underset{\sim}{v}_i^0 + \frac{\underset{\sim}{F}_i^0}{4m_i}, t=0 \end{cases} \tag{2.11}$$

2.7 Diagnostic Computations

2.7.1 Solids Fraction

The solids fraction is defined as the volume ratio of solid particles to the total volume the particles are enclosed within. The solids fraction is computed in two forms, the first form is as a solids fraction averaged over the entire computational volume. The second form is as a series of local solids fraction to give the solids fraction in control volumes. A typical result in Figure 2.2 shows ν versus dimensionless distance h, measured in particle diameters from the piston. Typically, the bulk solids fraction is $\nu \approx 0.6$. Monitoring the solids fraction indicates no significant change over the duration of the simulation.

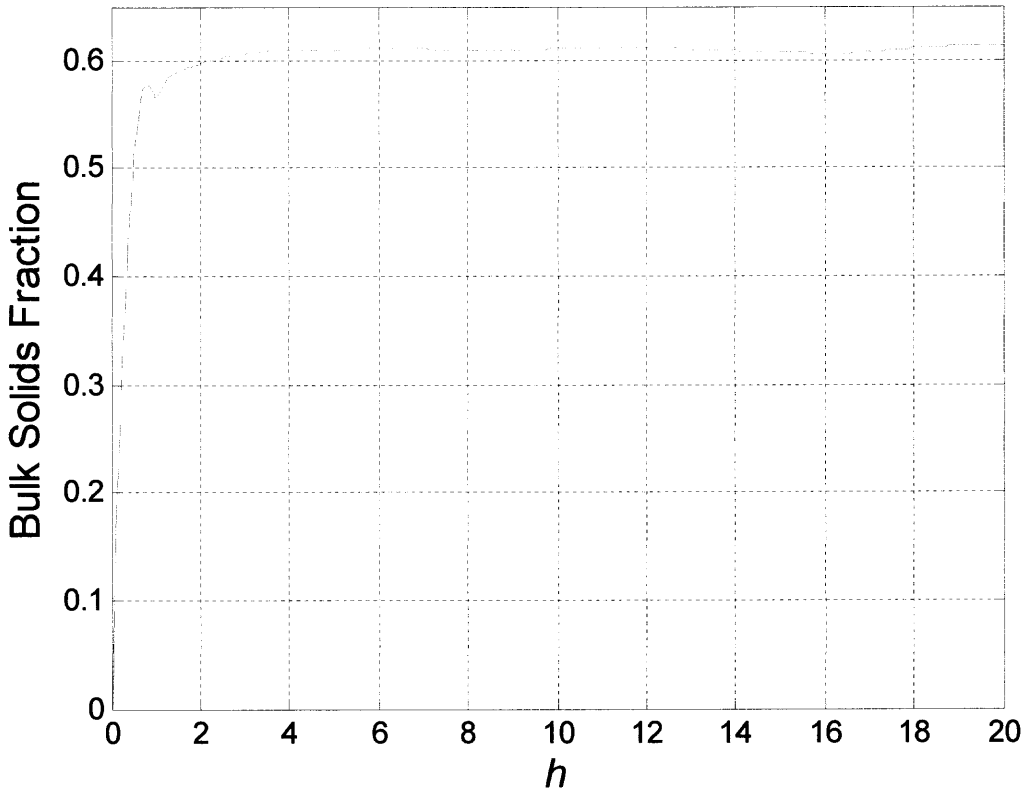

Figure 2.2 Bulk solids fraction ν as a function of the dimensionless distance h from the piston.

The computation of the bulk solids fraction is fairly simple and given by Equation (2.12) below. Where V_P is the volume of particles inside the computational volume V.

$$\nu = \frac{V_P}{V} \tag{2.12}$$

Several cases can arise when computing the solids fraction in control volumes. In one case the particle can be sliced by a plane, and in another case the particle can be sliced by a cylindrical surface. In the first case of the particle being intersected by a plane several sub-cases arise which are described in great detail in [41].

In the second case when the particle is intersected by a cylindrical surface the approach is much more complex. This method approximates a double integral with a double summation. The geometry of the problem is defined by local annular volumes. Each volume has an inner and outer radius, along with a z thickness as shown in Figure 2.3. Beginning at the minimum z coordinate of a specific control volume a series of arc lengths (shown as bold lines) are computed at increasing radii with a step size δr as shown in Figure 2.4.

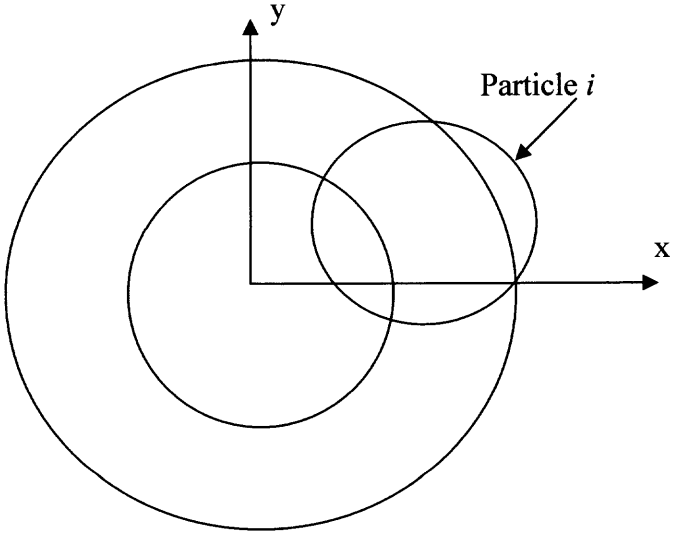

Figure 2.3 Representation of a particle intersected by an annular region.

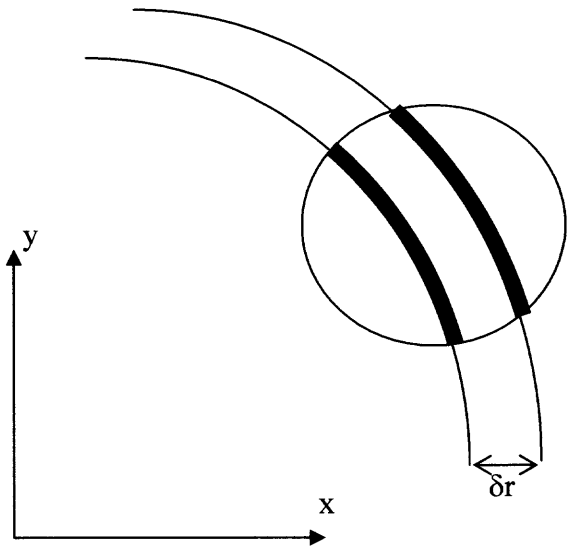

Figure 2.4 Representation of the method used to compute the included sphere volume.

These arc lengths are multiplied by the step size δr and summed together to yield an area at that specific z coordinate. Similarly, this process is repeated along the axial direction at a step size δz. The summation of the step size δz multiplied by the areas yield the volume of a single particle inside of the control volume. This process is then repeated for all particles inside of the control volume to obtain a solids fraction for that control volume. The implementation of this process is via a subroutine *volume.f*. The FORTRAN code for the subroutine *volume.f* along with a more detailed description of the method is given in Appendix E.

2.7.2 Piston Force Extraction

The computation of the piston load in the DEM simulation is fairly straightforward. The individual particle contact force axial component that is computed in the *forces.f* subroutine is summed over all of the particles. This value is computed every time step and saved when the subroutine *datasav2.f* is called. This computed load is also stored and used in a time average; however the time averaged piston load is used only for cases with no relative translations between the granular material and the cylinder wall.

The force experienced by the piston rapidly decays and then fluctuates as it moves downwards. Consequently, it was necessary to allow the system to equilibrate for approximately 2000 seconds (or a total displacement of ~2 particle diameters of the piston) in order to compute a value for the load. The load is extracted from the data using two methods that yielded statistically equivalent values (i.e., in most cases to within three significant digits). In the first method, short and long term averages of the actual force F_z are computed until their values are the same to within a tolerance of 10^{-4}. The second method finds the parameters A, B, and C through a least squares fit of the data to the form of Equation (2.13) below.

$$\hat{F}(t) = A + Be^{-C\,t} \tag{2.13}$$

Where $\hat{F}$ is simply the piston load as a fraction of the fill weight in the cylinder. The limit as $t \to \infty$ yields the equilibrium value A of the normalized load $\hat{F}$. Figure 2.5 is a representative plot of $\hat{F}$, where the functional form is shown as the solid line. Fluctuations are underscored in the inset of the figure. The value computed from the curve fit agreed to within three significant digits with the statistical average. For the case

shown, the normalized piston load was determined to be 0.607±0.0011. Appendix B contains the MATLAB code that does the post-processing.

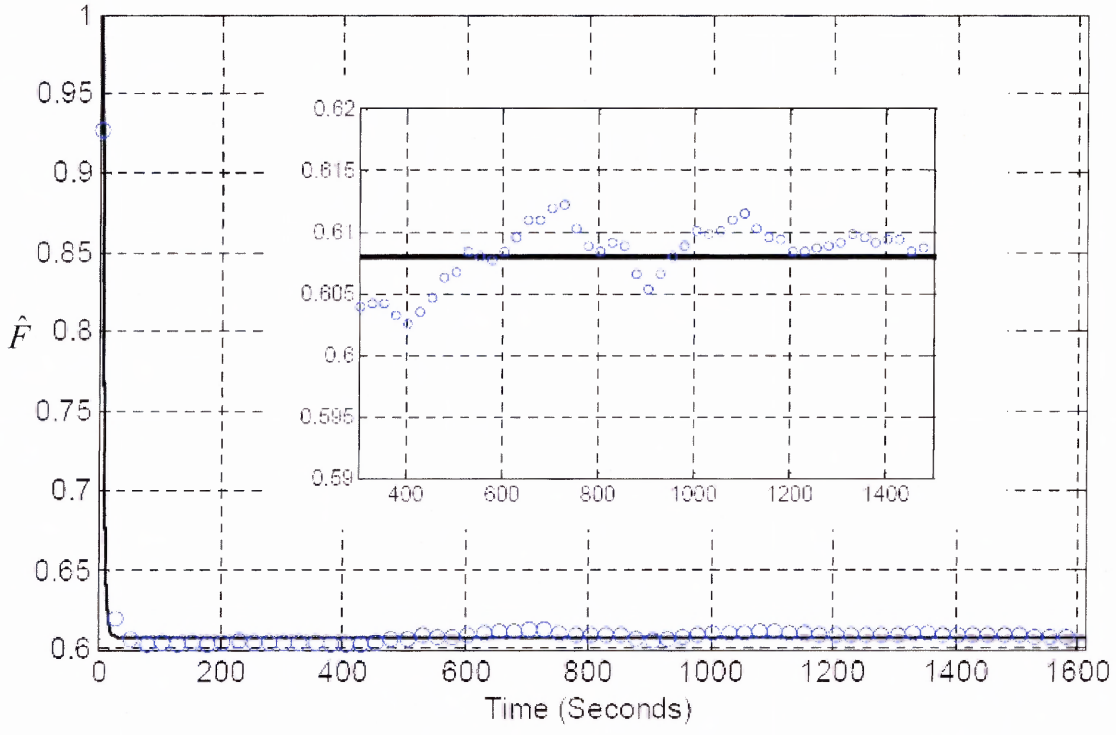

Figure 2.5 Sample data of the evolution of the dimensionless load $\hat{F}$ on the piston as a percent of the total fill weight in the cylinder. The symbols represent every 50^{th} data point from the simulation, while the solid line is the least squares fit to $\hat{F}(t) = A + Be^{-C\,t}$. Fluctuations are highlighted in the insert.

2.7.3 Stress Tensor

In granular materials the stress tensor is composed of two components. The first component is an inertial component, also known as the kinetic component, and the second term is a collisional component, also known as the potential component. The stress tensor averaged over a volume V in Cartesian coordinates is given by Equation (2.14) [33].

$$\underset{\sim}{\underset{\sim}{P}} = \underset{\sim}{\underset{\sim}{P}}_K + \underset{\sim}{\underset{\sim}{P}}_P = \frac{1}{V}\left(\sum_i m_i \left(\underset{\sim}{v}_i - \underset{\sim}{u}_i\right)\left(\underset{\sim}{v}_i - \underset{\sim}{u}_i\right) + \sum_{i>j} \underset{\sim}{F}_{ij} \underset{\sim}{R}_{ij} \right) \tag{2.14}$$

Where $\underset{\sim}{u}_i$ is the mean velocity field, $\underset{\sim}{F}_{ij}$ is the inter-particle contact force, and $\underset{\sim}{R}_{ij}$ is the vector connecting particle centers. The first term in Equation (2.14) is the kinetic contribution to the stress tensor and it depends on velocity fluctuations about the mean velocity field. In this thesis, the motion is quasi-static, and therefore the kinetic component of the stress tensor is negligible. The second term is the potential contribution, which is dominant for the quasi-static system considered in this thesis. Therefore, the computation of the stress tensor in Cartesian coordinates can be computed as,

$$\underset{\sim}{\underset{\sim}{P}} = \underset{\sim}{\underset{\sim}{P}}_P = \frac{1}{V} \sum_{j \neq i} \underset{\sim}{F}_{ij} \underset{\sim}{R}_{ij} \tag{2.15}$$

The calculation of the radial stress P_{rr} is needed in order to validate one of the main assumptions of the Janssen' theory, namely that, $\sigma_{rr} = K\sigma_{zz}$. To do this, it is necessary to compute the radial stress as a function of the axial coordinate and radial distance from the center of the cylinder. A text on molecular dynamics simulations [42] gives a thorough introduction to the derivation of the stress calculation. In order to compute P_{rr}, the contact force between particles i and j in the radial direction $\underset{\sim}{F}_{rij}$ needs to be determined. Beginning with the Cartesian components of the known net contact force $\underset{\sim}{F}_{ij}$,

$$\underset{\sim}{F}_{ij} = F_x \underset{\sim}{e}_x + F_y \underset{\sim}{e}_y + F_z \underset{\sim}{e}_z \tag{2.16}$$

one can find $\underset{\sim}{F}_{rij}$ as follows. The vector connecting the centers of particles i and j is given by as

$$\underset{\sim}{r}_{ij} = (x_j - x_i)\underset{\sim}{e}_x + (y_j - y_i)\underset{\sim}{e}_y + (z_j - z_i)\underset{\sim}{e}_z \tag{2.17}$$

Because the particles are monodisperse, the vector starting from the origin of the coordinate system, and ending at the contact location of particles *i* and *j*, denoted by $\underset{\sim}{c}$ is shown in Figure 2.6, and given by,

$$\underset{\sim}{c} = \frac{\underset{\sim}{i} + \underset{\sim}{j}}{2} = \left(\frac{x_i + x_j}{2}\right)\underset{\sim}{e}_x + \left(\frac{y_i + y_j}{2}\right)\underset{\sim}{e}_y + \left(\frac{z_i + z_j}{2}\right)\underset{\sim}{e}_z \tag{2.18}$$

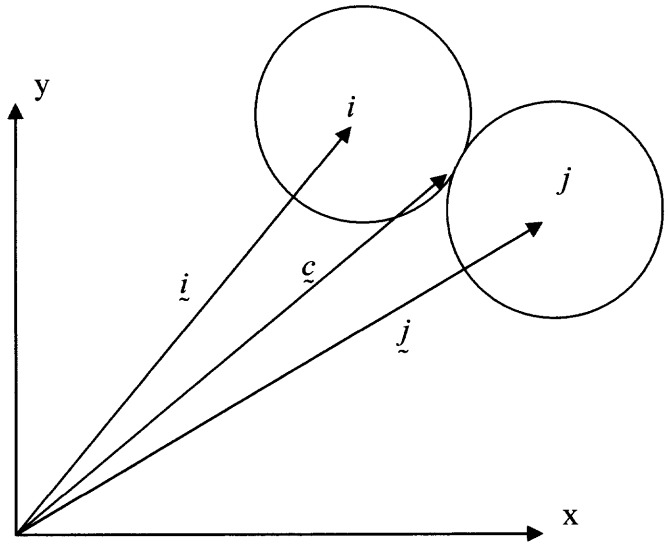

Figure 2.6 Geometry used in the computation of $\underset{\sim}{c}$.

In the usual manner, the unit vector $\underset{\sim}{e}_c$ in the $\underset{\sim}{c}$ direction is,

$$\underset{\sim}{e}_c = \frac{\underset{\sim}{c}}{\|\underset{\sim}{c}\|} = \frac{\left(\frac{x_i + x_j}{2}\right)\underset{\sim}{e}_x + \left(\frac{y_i + y_j}{2}\right)\underset{\sim}{e}_y + \left(\frac{z_i + z_j}{2}\right)\underset{\sim}{e}_z}{\sqrt{\left(\frac{x_i + x_j}{2}\right)^2 + \left(\frac{y_i + y_j}{2}\right)^2 + \left(\frac{z_i + z_j}{2}\right)^2}} \tag{2.19}$$

Then the unit vector in the radial direction (i.e., no z-component) that starts at the cylindrical axis and ends at the contact point of particles i and j is shown in Figure 2.7 and given by,

$$\underset{\sim}{e}_r = \left(\underset{\sim}{e}_c \bullet \underset{\sim}{e}_x\right)\underset{\sim}{e}_x + \left(\underset{\sim}{e}_c \bullet \underset{\sim}{e}_y\right)\underset{\sim}{e}_y = \frac{\left(\frac{x_i + x_j}{2}\right)\underset{\sim}{e}_x + \left(\frac{y_i + y_j}{2}\right)\underset{\sim}{e}_y}{\sqrt{\left(\frac{x_i + x_j}{2}\right)^2 + \left(\frac{y_i + y_j}{2}\right)^2 + \left(\frac{z_i + z_j}{2}\right)^2}} \tag{2.20}$$

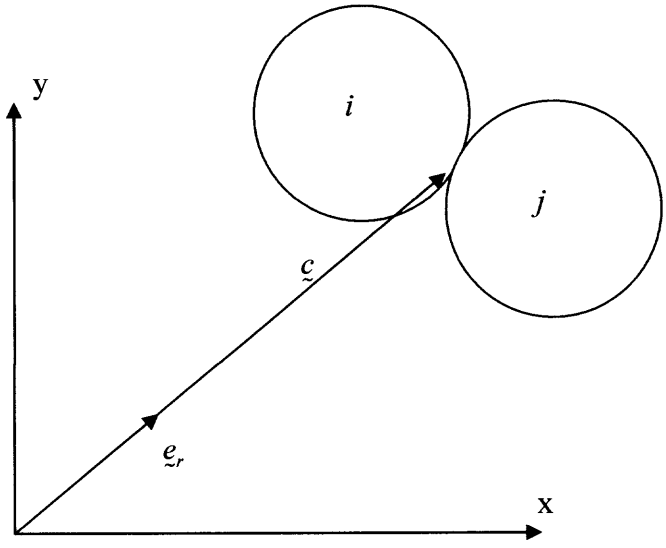

Figure 2.7 Definition of the unit vector in the radial direction $\underset{\sim}{e}_r$.

The radial force acting between colliding particles i and j is simply the projection of $\underset{\sim}{F}_{ij}$ onto the radial direction, i.e.,

$$\underset{\sim}{F}_{rij} = \left(\underset{\sim}{F}_{ij} \bullet \underset{\sim}{e}_r\right)\underset{\sim}{e}_r = \left(\frac{F_x\left(\frac{x_i + x_j}{2}\right) + F_y\left(\frac{y_i + y_j}{2}\right)}{\sqrt{\left(\frac{x_i + x_j}{2}\right)^2 + \left(\frac{y_i + y_j}{2}\right)^2 + \left(\frac{z_i + z_j}{2}\right)^2}}\right)\underset{\sim}{e}_r \tag{2.21}$$

Let δ_{ij} be the radial distance between the centers of the colliding particles i and j, is given by,

$$\delta_{ij} = \sqrt{x_j^2 + y_j^2} - \sqrt{x_i^2 + y_i^2} \tag{2.22}$$

Then the computation of the P_{rr} component of the stress tensor is completed via Equation (2.23), where V is the control volume over which averaging takes place.

$$P_{rr} = \frac{1}{V} F_{rij} \delta_{ij} \tag{2.23}$$

2.7.4 Particle Rotations

It was suspected that at larger wall friction coefficients particles in contact with the wall may be rolling. Thus, the load on the piston would be greater than expected if only sliding took place. This hypothesis is supported by other simulations and measurements of granular material deformation behavior, which have demonstrated that the angle of repose is less sensitive to inter-particle friction as the friction coefficient increases. This behavior has been attributed to the larger fraction of rolling contacts that exist in assemblies with high inter-particle friction [43-45].

To test this hypothesis some simulations were run and the magnitudes of the angular velocity of the particles were analyzed. The magnitudes of the angular velocity of the particles were then normalized against the maximum angular velocity possible due to the translation of either the piston or the cylinder wall, ω_p. The normalized values are then plotted against radial location and trends may be spotted. If the particles contacting the wall have an $|\omega / \omega_p|$ at or near 1, then the particles are rolling and the piston load is

greater than expected. If the value of $|\omega/\omega_p|$ is low then, the particles are sliding and rolling. On the other hand if the value of $|\omega/\omega_p|$ is zero, Janssen's assumption that friction is fully activated at the walls would be confirmed.

The computation of ω_p is fairly simple. The angular velocity in a single direction is given by $\omega = (2v)/d$. Assuming that the maximum velocity is due to the translation of the piston of the cylinder wall, the maximum angular velocity then becomes $\omega = (2v_p)/d$. Where v_p is the translational velocity of the piston or the cylinder wall. Then, for all three coordinate directions the maximum angular velocity due to the translation of the piston or cylinder wall ω_p is given by,

$$\omega_p = \sqrt{\sum_{i=x,y,z} \omega_i^2} = \sqrt{3}\left(\frac{2v_p}{d}\right) \tag{2.24}$$

The results for the normalized angular velocity $|\omega/\omega_p|$ are spatially averaged in annular zones. The annular zones are defined in a post-processing MATLAB code. Appendix C contains the MATLAB code to compute the spatial average, and impose the restriction that $|\omega/\omega_p| \leq 1$.

CHAPTER 3

SIMULATION RESULTS AND ANALYSIS

3.1 Dimensionless Quantities

It is convenient to represent the results of the simulations in terms of dimensionless quantities. The dimensionless force $F^* = \frac{6F_z}{\rho g \pi d^3}$ is simply the force on the piston normalized by the weight of a single particle that has a mass density ρ, and a diameter *d*. The normalized height *H/D* is the amount of filling material in the cylinder of diameter *D*. Janssen's model (Equation (1.5)) is easily expressed in terms of F^* and *H/D* as

$$F^*(H/D) = \frac{1.5\nu\phi^3}{\lambda}\left[1 - e^{-\lambda(H/D)}\right] \tag{3.1}$$

$$\lambda = \beta D = 4\mu K \tag{3.2}$$

3.2 Static Simulations

A comprehensive set of static simulations were performed in which neither the piston nor the cylindrical wall was translated. Therefore, in the static simulations the frictional force is not activated. The parameters of those simulations, called static simulations, are shown in Table 3.1 below.

Table 3.1 Parameters Used in the Various Cases for Static Simulations

Case	H/D	μ_w	μ_p	ϕ
S1	0.750	0.1	0.1	13.33
S2	0.937	0.1	0.1	13.33
S3	1.125	0.1	0.1	13.33
S4	1.312	0.1	0.1	13.33
S5	1.499	0.1	0.1	13.33
S6	1.687	0.1	0.1	13.33
S7	1.875	0.1	0.1	13.33
S8	2.062	0.1	0.1	13.33
S9	0.375	0.1	0.1	26.66
S10	0.422	0.1	0.1	26.66
S11	0.469	0.1	0.1	26.66
S12	0.515	0.1	0.1	26.66
S13	0.375	0.4	0.4	26.66
S14	0.422	0.4	0.4	26.66
S15	0.469	0.4	0.4	26.66
S16	0.515	0.4	0.4	26.66
S17	0.750	0.4	0.1	13.33
S18	0.937	0.4	0.1	13.33
S19	1.125	0.4	0.1	13.33
S20	1.312	0.4	0.1	13.33
S21	1.499	0.4	0.1	13.33
S22	1.687	0.4	0.1	13.33
S23	1.875	0.4	0.1	13.33
S24	2.062	0.4	0.1	13.33

The piston load was computed for simulations S1 – S24. Cases S1 – S8 were conducted at a diameter ratio ϕ of 13.33, an inter-particle friction coefficient μ_p of 0.1, and a wall friction coefficient μ_w of 0.1. The results of simulations S1 – S8 are shown in Figure 3.1 below. Cases S9 – S16 were done using a larger diameter ratio $\phi = 26.66$. Here, the inter-particle friction coefficient $\mu_p = 0.1$, and a wall friction coefficient μ_w of 0.1. For S13 – S16, $\mu_p = 0.4$, and $\mu_w = 0.4$ of 0.4. The results of cases S9 – S16 are presented in Figure 3.2 below.

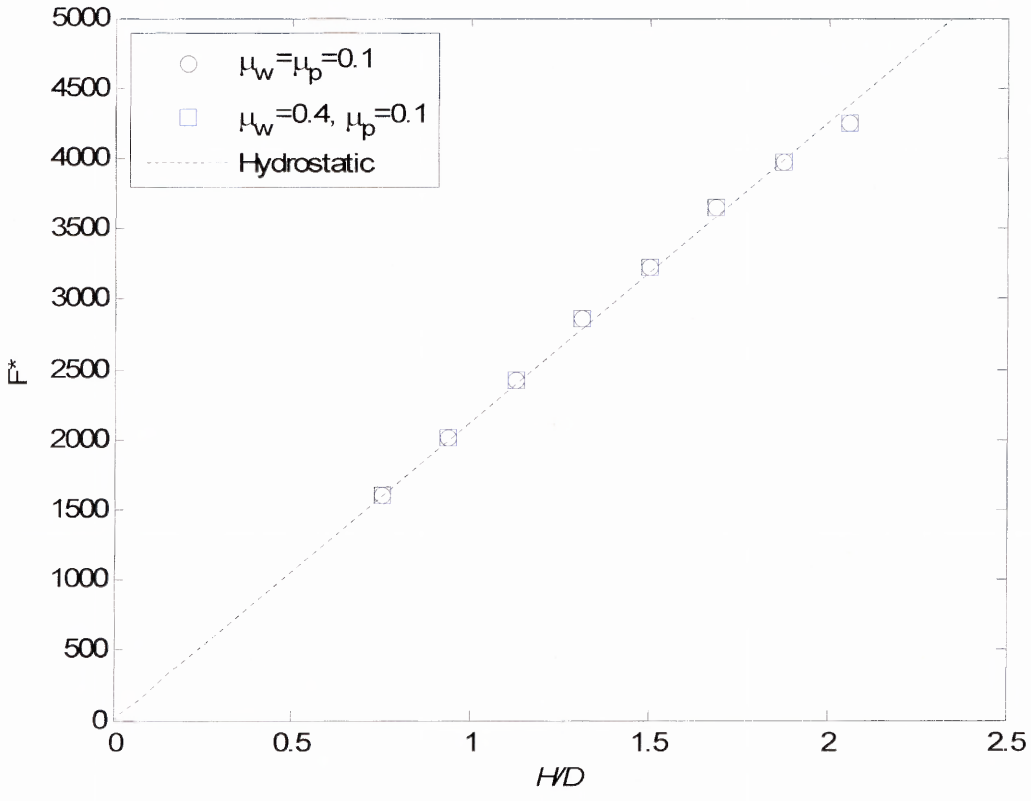

Figure 3.1 Piston load for static simulations S1 – S8 (o), and S17 – S24 (□). The hydrostatic curve is given by the dashed black line.

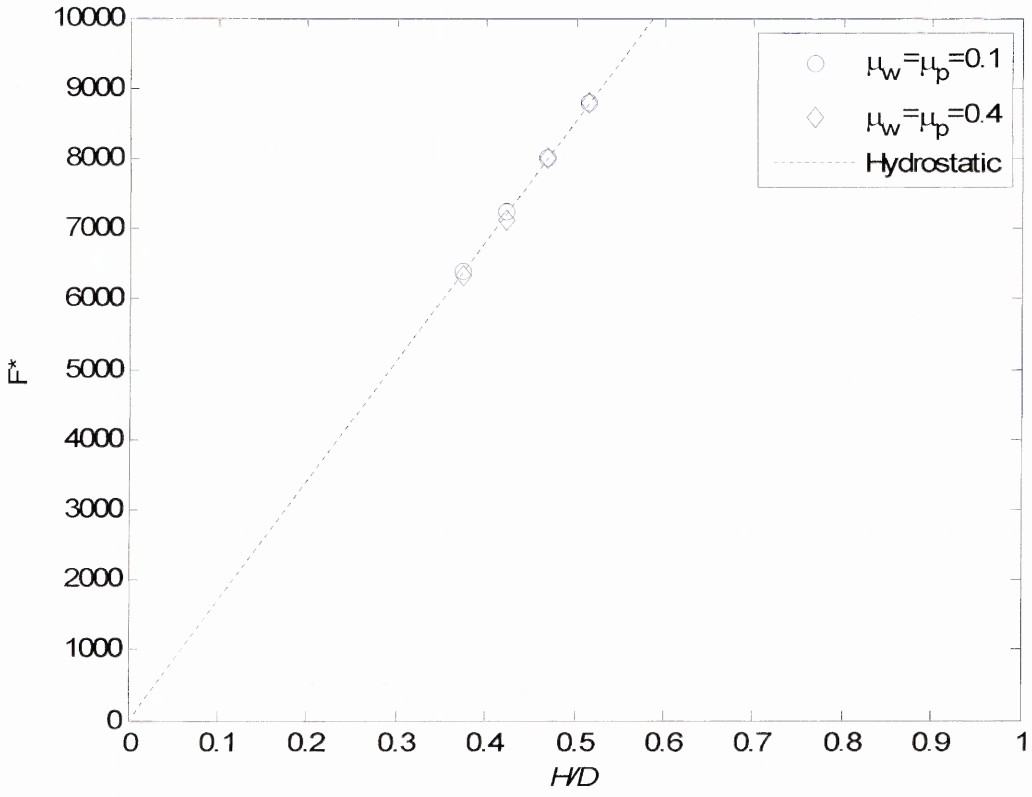

Figure 3.2 Piston load for simulations S9 – S12 (o), and S13 – S16 (◊). The hydrostatic curve is given by the dashed black line.

As seen from both Figure 3.1 and Figure 3.2, the load vs. fill height behavior mimics that of a hydrostatic material. A comparison of results S1 - S8 against S1 – S24, where the only difference is in the wall friction coefficient, shows no change in the behavior. This suggests that for the assemblies in the static simulations, wall friction must be activated to eliminate a hydrostatic behavioral region. That is, contacts of particles with the walls must be at the Coulomb limit and this situation occurs only when there is some motion of the assembly relative to the cylinder. In Subsection 0 it is shown that for the same values of *H*/*D* and similar simulation parameters, a Janssen behavior is clearly evident for the dynamic case, when friction is activated However, for a static case

a Janssen behavior will follow a hydrostatic region [8] for sufficiently large values of the fill depth *H/D.*

3.3 Dynamic Simulations

A comprehensive set of dynamic simulations were performed in which the piston was translated in the positive z- direction, or the wall translated in the negative z-direction. The parameters of these simulations are shown in Table 3.2, and Table 3.3. For all dynamic cases, the velocity of either the piston or the cylinder wall was $v_p = 0.001$ diameters per second. This rate was identified by performing a series of simulated test studies over a wide range of piston velocities. Subsequent to doing this, it was found that value was consistent with that in other studies reported in the literature [6] so that dynamic/inertial effects were not present.

Table 3.2 Parameters Used for Dynamic Simulations

Case	H/D	μ_w	μ_p	ϕ
D1	1.499	0.4	0.4	13.33
D2	1.687	0.4	0.4	13.33
D3	1.875	0.4	0.4	13.33
D4	2.062	0.4	0.4	13.33
D5	1.499	0.8	0.8	13.33
D6	1.687	0.8	0.8	13.33
D7	1.875	0.8	0.8	13.33
D8	2.062	0.8	0.8	13.33
D9	0.375	0.4	0.4	26.66
D10	0.422	0.4	0.4	26.66
D11	0.469	0.4	0.4	26.66
D12	0.515	0.4	0.4	26.66
D13	1.499	0.1	1.0	13.33
D14	1.687	0.1	1.0	13.33
D15	1.875	0.1	1.0	13.33

Table 3.3 Parameters Used for Dynamic Simulations (Continued)

Case	H/D	μ_w	μ_p	ϕ
D16	2.062	0.1	1.0	13.33
D17	0.750	0	0.1	13.33
D18	1.125	0	0.1	13.33
D19	1.500	0	0.1	13.33
D20	1.875	0	0.1	13.33
D21	2.250	0	0.1	13.33
D22	2.625	0	0.1	13.33
D23	3.000	0	0.1	13.33
D24	3.375	0	0.1	13.33
D25	0.749	1.2	0.1	13.33
D26	1.125	1.2	0.1	13.33
D27	1.500	1.2	0.1	13.33
D28	1.875	1.2	0.1	13.33
D29	0.094	0.8	0.1	26.66
D30	0.141	0.8	0.1	26.66
D31	0.234	0.8	0.1	26.66
D32	0.187	0.8	0.1	26.66
D33	0.187	0.12	0.1	13.33
D34	0.375	0.12	0.1	13.33
D35	0.750	0.12	0.1	13.33
D36	1.125	0.12	0.1	13.33
D37	1.500	0.12	0.1	13.33
D38	1.875	0.12	0.1	13.33
D39	2.250	0.12	0.1	13.33
D40	2.812	0.12	0.1	13.33
D41	0.187	0.4	0.1	13.33
D42	0.375	0.4	0.1	13.33
D43	0.750	0.4	0.1	13.33
D44	1.125	0.4	0.1	13.33
D45	1.500	0.4	0.1	13.33
D46	1.875	0.4	0.1	13.33
D47	2.250	0.4	0.1	13.33
D48	2.812	0.4	0.1	13.33
D49	0.375	0.8	0.1	13.33
D50	0.750	0.8	0.1	13.33
D51	1.125	0.8	0.1	13.33
D52	0.375	0.8	0.1	13.33
D53	0.750	0.8	0.1	13.33
D54	1.125	0.8	0.1	13.33
D55	1.500	0.8	0.1	13.33

3.3.1 Comparison Between Piston and Cylinder Translation

Simulations D49 – D55 are used to compare the difference between translating the piston down or moving the cylinder wall upwards. In simulations D49, D50, and D51, the piston was translated downward in the positive z-direction. In simulations D52, D53, D54, and D55, the cylinder wall was translated upward in the negative z-direction. The compiled results for these cases (Figure 3.3) indicate no significant difference in the dimensionless piston load F^* versus H/D (i.e., the least squares curve fits to the data are almost the same). However, simulation D55 was also performed moving the cylinder wall, and the additional data point is included in Figure 3.4.

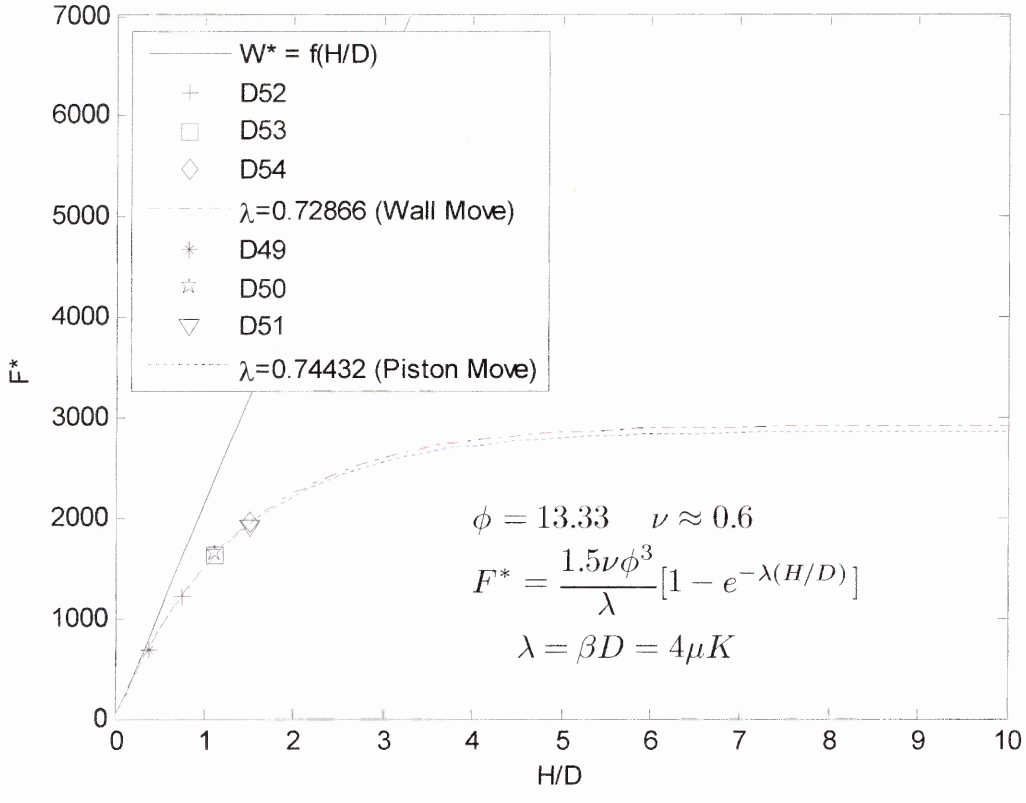

Figure 3.3 Results comparing translating the piston or the wall to activate friction. D52 (+), D53 (□), D54 (◊), the dash-dot red line is a fit of D52 – D54 to Equation (3.1). D49 (*), D50 (☆), D51 (▽), and lastly the dotted blue line is a fit of D49 – D51 to Equation (3.1).

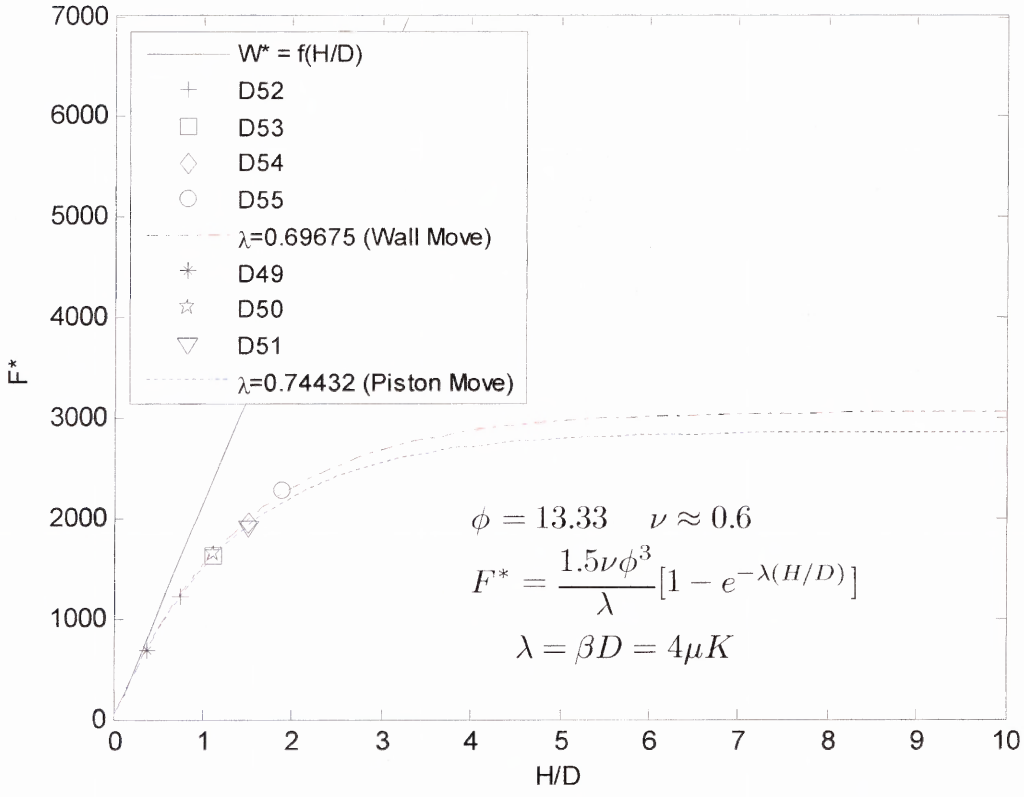

Figure 3.4 Results comparing translating the piston or the wall to activate friction with more data points. D52 (+), D53 (□), D54 (◊), D55 (o), and the dash-dot red line is a fit of D52 – D55 to Equation (3.1). D49 (*), D50 (☆), D51 (▽), and lastly the dotted blue line is a fit of D49 – D51 to Equation (3.1).

It is observed that the single additional point produces a greater discrepancy between the least square fits. This could be a consequence of the regression numerics rather than a real difference in the physical behavior (i.e.., piston motion versus cylinder wall motion). The only way to test this is to carry out extended cases at larger *H/D* values. However, the system sizes to validate this hypothesis would require significantly greater computing resources than was available.

3.3.2 Particle Rolling and Effect of Friction Coefficient

Janssen's theory assumes that the frictional force is fully activated. To test this assumption the normalized angular velocity $|\omega/\omega_p|$ was analyzed to determine if the particles contacting the wall were rolling rather than sliding as Janssen's theory assumes. A preliminary analysis indicated that an isolated few particles in the system were spinning at rates 3 orders of magnitude larger than expected. A closer examination revealed that the tangential force is more sensitive to the time step Δt in some cases than the normal force. Figure 3.5 below shows $|\omega/\omega_p|$ plotted against $|r/R|$ for several identical simulations, run at different time steps, where $|r/R|$ is the radial distance normalized by the cylinder radius R.

It is evident from Figure 3.5 that decreasing the time step would allow the angular velocities of all the particles to become more stable. However, since each simulation is comprised of thousands of particles and the high angular velocity was noted for only a few particles, those few outliers are omitted from the data presented hereafter. Only values of $|\omega/\omega_p|$ that satisfy the condition $|\omega/\omega_p| \leq 1$ are used. To run the simulations at a smaller time step would have been computationally unfeasible with the resources at the time.

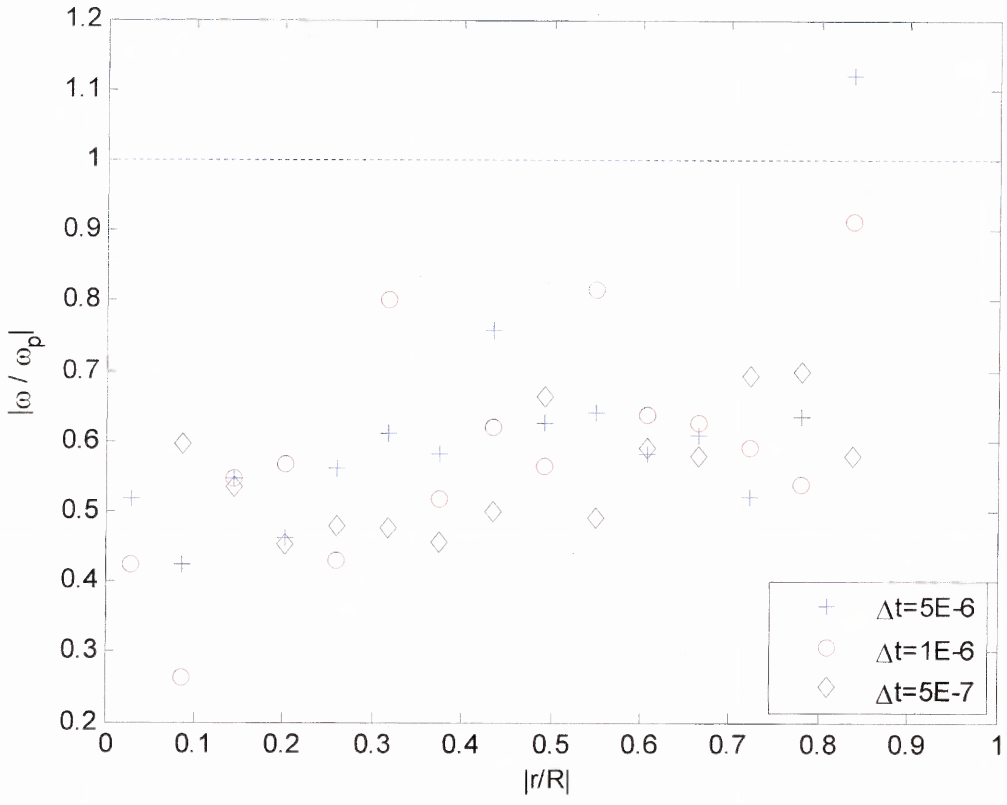

Figure 3.5 Effect of time step on angular velocity. In all cases the friction coefficients $\mu_w = \mu_p = 0.4$ with a diameter ratio $\phi = 7.5$ and a fill height (H/D) = 1.317. Results are shown for a time step of $\Delta t = 5\times10^{-6}$ seconds (+), $\Delta t = 1\times10^{-6}$ seconds (o), $\Delta t = 5\times10^{-7}$ seconds (◊).

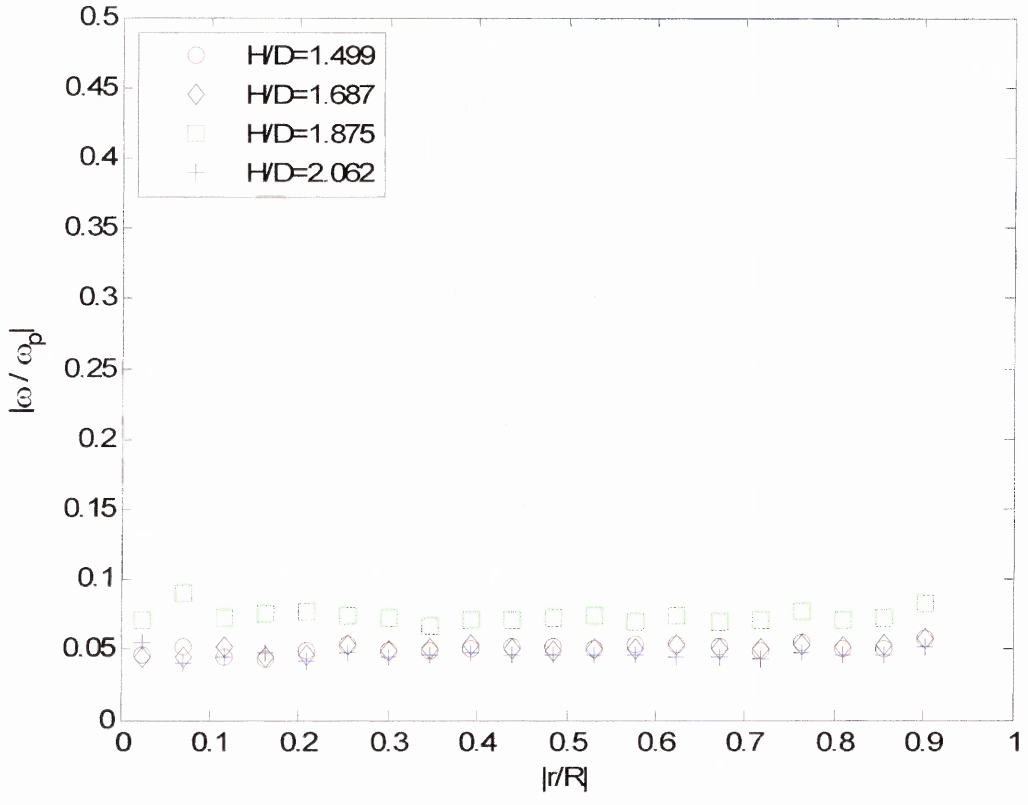

Figure 3.6 $|\omega / \omega_p|$ versus $|r / R|$ for the case of μ_w=0.4, μ_p=0.1 and ϕ=13.33. H/D = 1.499 (o), H/D = 1.687 (◊), H/D = 1.875 (□), and H/D = 1.499 (+).

Figure 3.6 shows $|\omega / \omega_p|$ plotted against $|r / R|$ for several different fill heights. The wall friction coefficient was set to 0.4, and the inter-particle friction coefficient was set to 0.1. As seen from Figure 3.6, the average normalized angular velocity is far from the condition of maximum rolling. Thus, the particles near the wall and throughout the bulk for friction coefficients typical of those used here are not fully rolling. The particles in the packed bed are sliding and rolling.

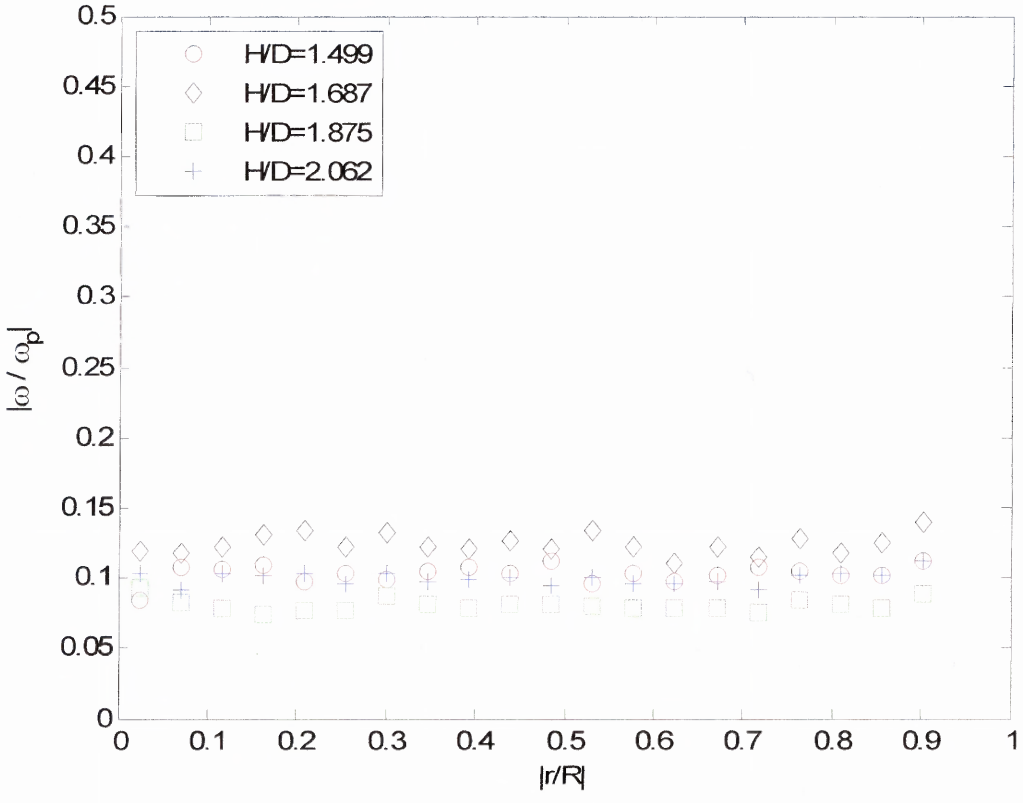

Figure 3.7 $|\omega/\omega_p|$ versus $|r/R|$ for the case of μ_w=0.8, μ_p=0.1 and ϕ=13.33. H/D = 1.499 (○), H/D = 1.687 (◊), H/D = 1.875 (□), and H/D = 1.499 (+).

Figure 3.7 shows $|\omega/\omega_p|$ plotted against $|r/R|$ for several different fill heights. The wall friction coefficient was set to 0.8, and the inter-particle friction coefficient was set to 0.1. As seen from Figure 3.7, the average normalized angular velocity is far from the condition of maximum rolling. Thus, the particles near the wall and throughout the bulk for friction coefficients typical of those used here are not fully rolling. The particles in the packed bed are sliding and rolling. It is noted that the results shown in Figure 3.7 indicate a larger value of $|\omega/\omega_p|$ than the results shown in Figure 3.6. These results

indicate that an increase in the wall friction coefficient increased the normalized angular velocity.

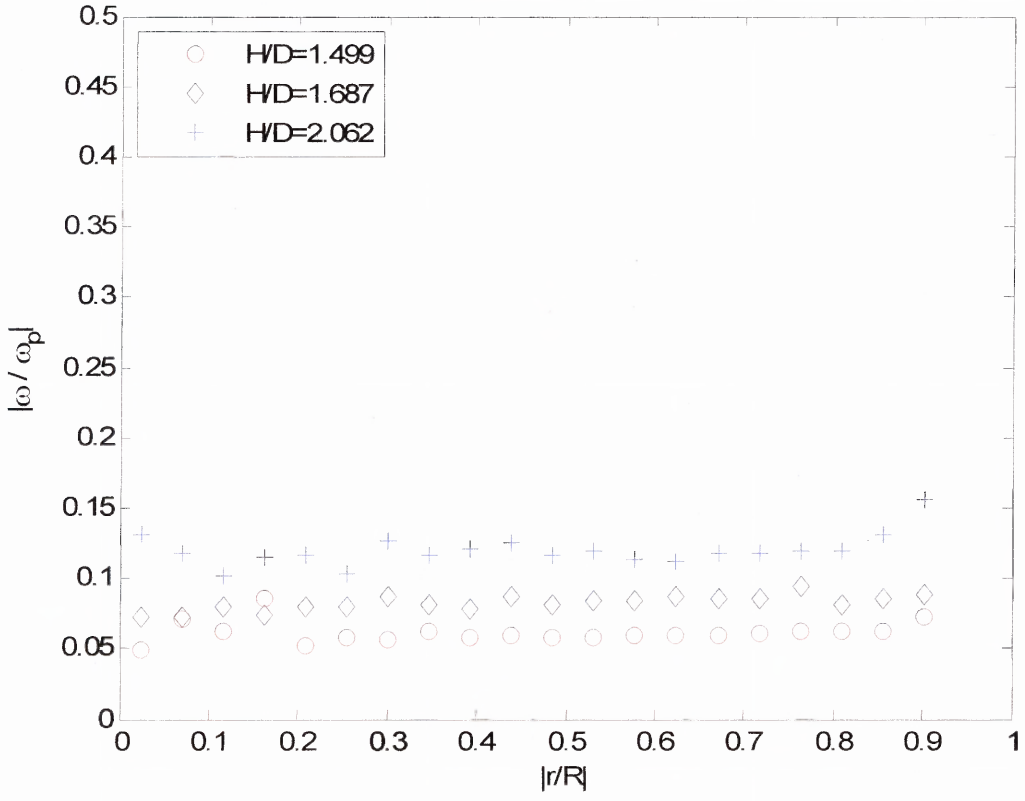

Figure 3.8 $|\omega/\omega_p|$ versus $|r/R|$ for the case of μ_w=0.4, μ_p=0.4 and ϕ=13.33. . H/D = 1.499 (o), H/D = 1.687 (◊), and H/D = 1.499 (+).

Figure 3.8 shows $|\omega/\omega_p|$ plotted against $|r/R|$ for several different fill heights. The wall friction coefficient was set to 0.4, and the inter-particle friction coefficient was set to 0.4. As seen from Figure 3.8, the average normalized angular velocity is far from the condition of maximum rolling. Thus, the particles near the wall and throughout the bulk for friction coefficients typical of those used here are not fully rolling. The particles in the packed bed are sliding and rolling. It is interesting to note that for different fill levels the amount of particle rotation increases with fill height.

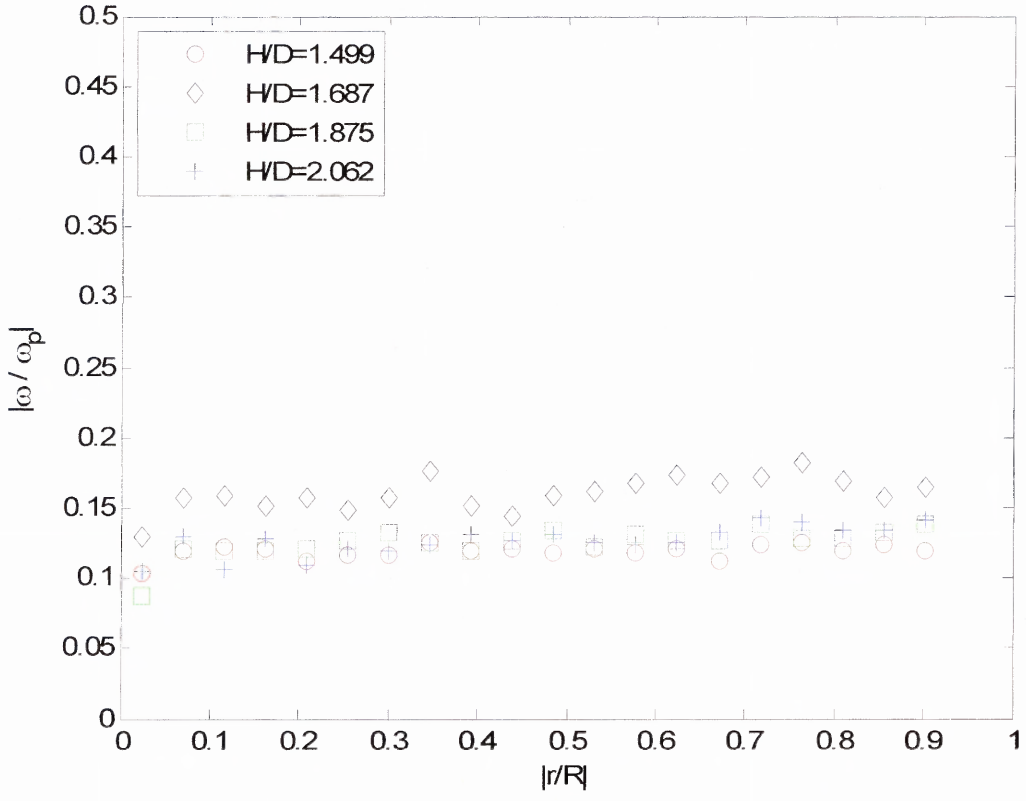

Figure 3.9 $|\omega/\omega_p|$ versus $|r/R|$ for the case of μ_w=0.8, μ_p=0.8 and ϕ=13.33. . H/D = 1.499 (o), H/D = 1.687 (◊), H/D = 1.875 (□), and H/D = 1.499 (+).

Figure 3.9 shows $|\omega/\omega_p|$ plotted against $|r/R|$ for several different fill heights. The wall friction coefficient was set to 0.8, and the inter-particle friction coefficient was set to 0.8. As seen from Figure 3.9, the average normalized angular velocity is far from the condition of maximum rolling. Thus, the particles near the wall and throughout the bulk for friction coefficients typical of those used here are not fully rolling. The particles in the packed bed are sliding and rolling.

As seen from the results above particles are rolling in the simulated assembly, both at the wall and in the bulk of the granular material. If Janssen's assumption were true particles would not roll, however simply slide at the friction limit. Therefore, the

simulated normalized angular velocity behavior indicates a violation of Janssen's assumption that frictional forces are fully activated in the assembly

In the simulated data increasing the wall friction coefficient simply caused the frictional force to create a larger torque, and therefore larger angular velocity onto the particle. The resistance to the linear motion was not increased. A closer examination of the data shows that as the wall friction coefficient increases the normalized angular velocity of the particles increases. This result is consistent with experiments [43] that show an increase in the friction coefficient may not increase the shear strength of the granular material. Also, the additional particle rolling causes the particles at the wall to act like lubrication, and therefore the piston load is larger than expected. The effect of the wall friction coefficient on the piston load is most evident in Figure 3.21. In the figure, it is observed that the difference between asymptotic piston loads for $\mu_w = 0.12$ and $\mu_w = 0.4$ is large, while the difference between $\mu_w = 0.4$ and $\mu_w = 0.8$ is much smaller. Indicating that a change from $\mu_w = 0.4$ to $\mu_w = 0.8$ did not increase the shear strength of the granular material, however the particle rolling acted as a sort of lubrication.

3.3.3 Piston Load

The piston load was computed for all dynamic simulations. The results are shown in Figure 3.10 through Figure 3.20 below, for a range of input parameters such as the wall and inter-particle friction coefficients and fill levels. In all the plots, the load vs. fill level data is fit to the functional form of Equation (3.1).

Figure 3.10 shows the results for simulations D1 – D4. In these simulations, the inter-particle and wall friction coefficients were set to 0.4. Additionally, the diameter ratio ϕ was set to 13.33, and in these simulations the piston was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model.

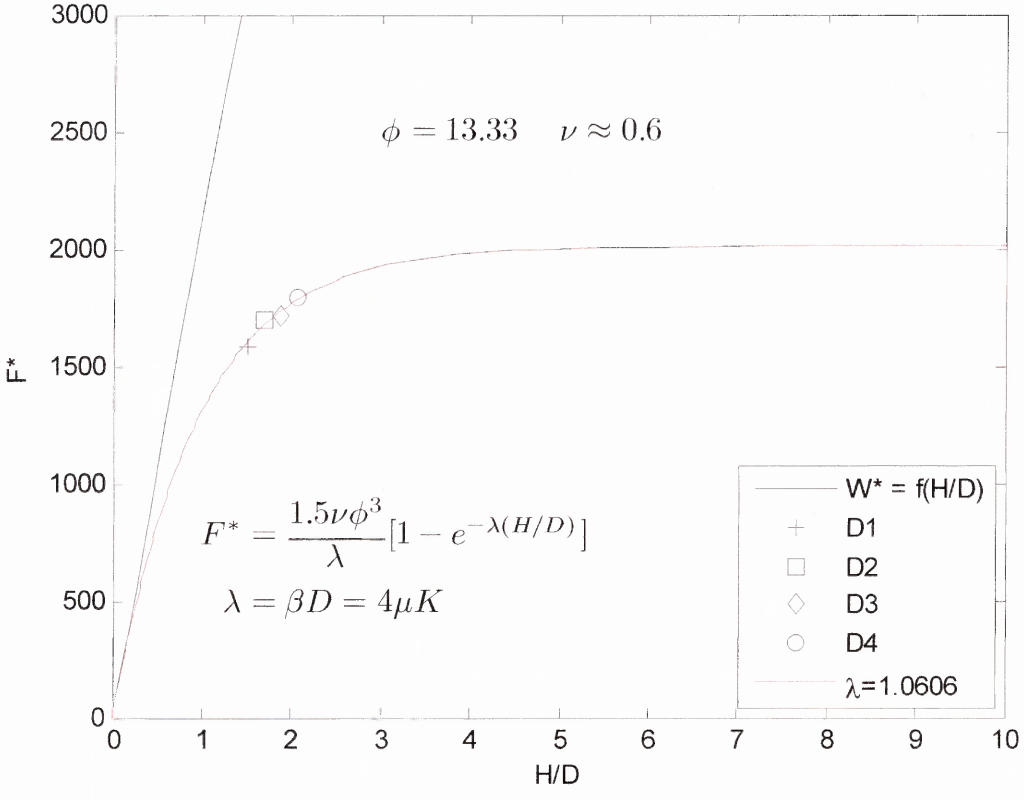

Figure 3.10 Simulated results for D1 (+), D2 (□), D3 (◊), D4 (o). The solid red line corresponds to a fit of D1 – D4 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

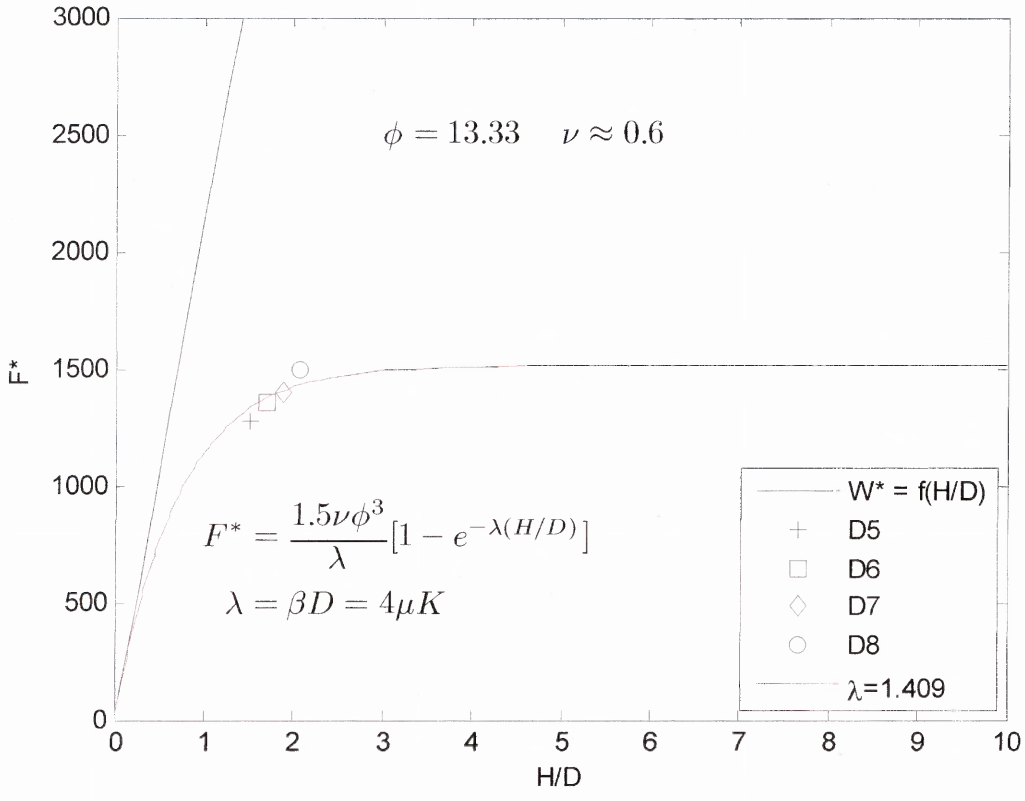

Figure 3.11 Simulated results for D5 (+), D6 (□), D7 (◊), D8 (o). The solid red line corresponds to a fit of D5 – D8 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.11 shows the results for simulations D5 – D8. In these simulations, the inter-particle and wall friction coefficients were set to 0.8, the diameter ratio ϕ was set to 13.33, and in these simulations the piston was translated to activate friction. As seen from the plot, the data is not a very good fit to the functional form of Janssen's model. The apparent discrepancy with the simulated data and Janssen's theory may be attributed to particles rolling at the increased friction coefficients as discussed in Subsection 3.3.2.

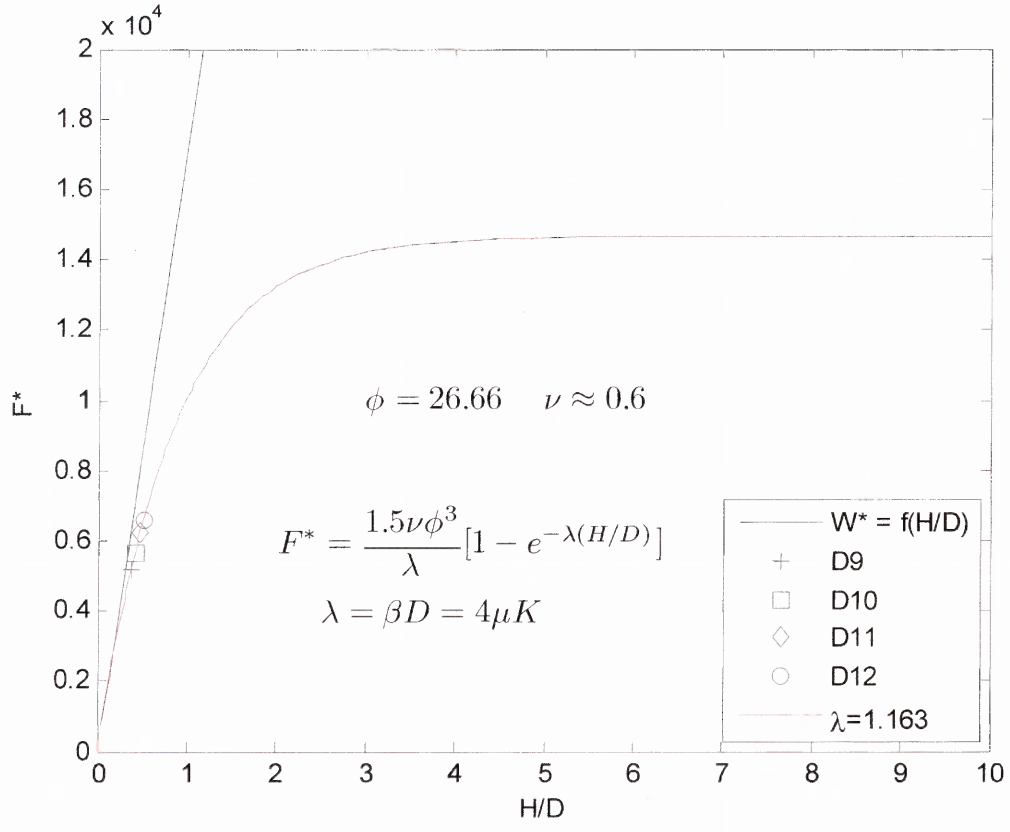

Figure 3.12 Simulated results for D9 (+), D10 (□), D11 (◊), D12 (o). The solid red line corresponds to a fit of D9 – D12 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.12 shows the results for simulations D9 – D12. In these simulations, the inter-particle and wall friction coefficients were set to 0.4, the diameter ratio ϕ was set to 26.66, and in these simulations the piston was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model.

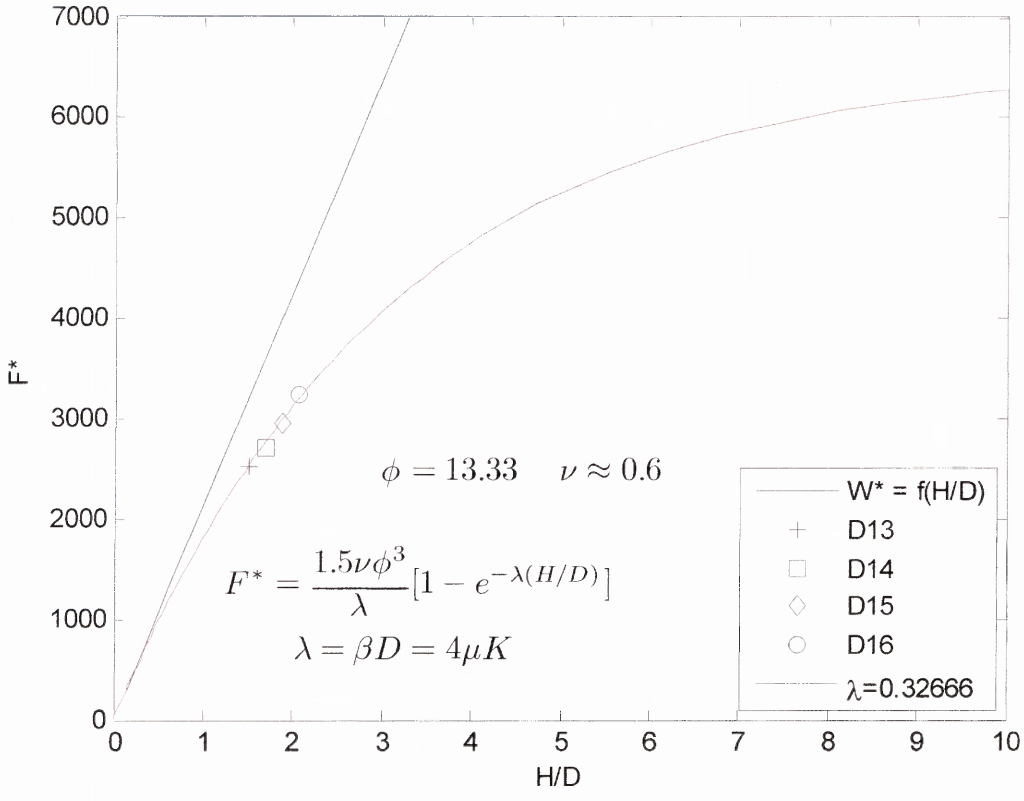

Figure 3.13 Simulated results for D13 (+), D14 (□), D15 (◊), D16 (o). The solid red line corresponds to a fit of D13 – D16 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.13 shows the results for simulations D13 – D16. In these simulations, the inter-particle friction coefficient was set to 1.0, and wall friction coefficient was set to 0.1, and the diameter ratio ϕ was set to 13.33. Also, in these simulations the piston was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model.

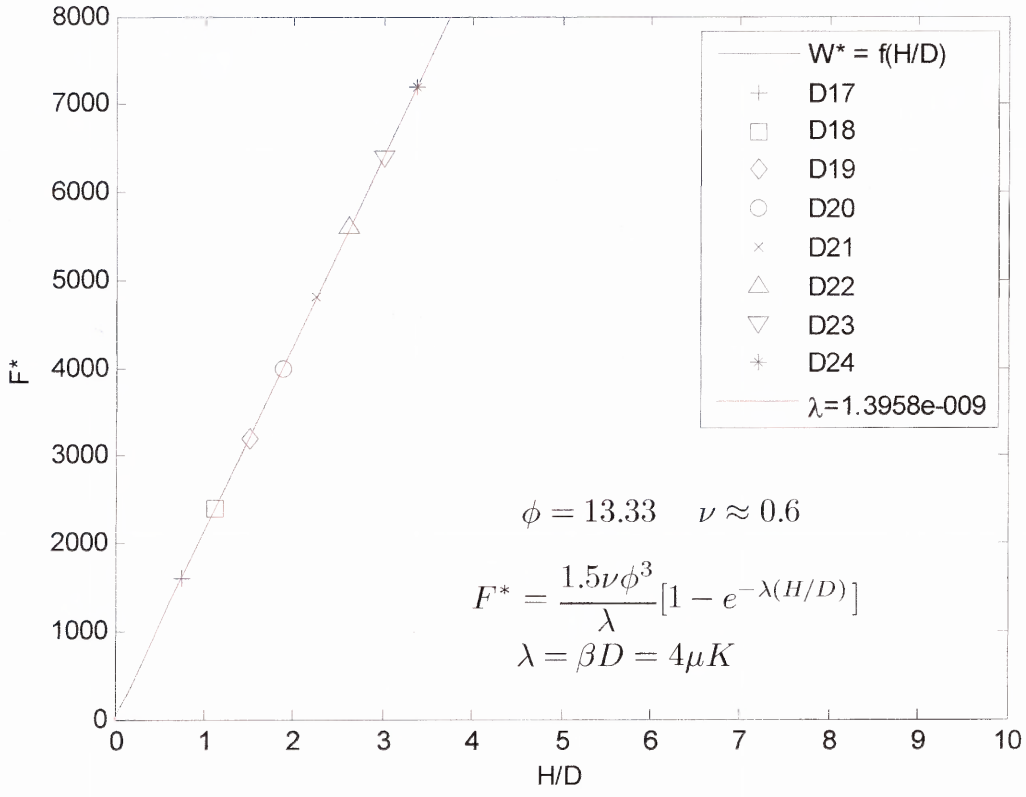

Figure 3.14 Simulated results for D17 (+), D18 (□), D19 (◊), D20 (o), D21 (x), D22 (△), D23 (▽), D24 (*). The solid red line corresponds to a fit of D17 – D24 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.14 shows the results for simulations D17 – D24. In these simulations, the wall friction coefficient μ_w was set to zero. The inter-particle friction coefficient μ_p was set to 0.1, the diameter ratio ϕ was set to 13.33. Also, in these simulations the piston was translated to activate friction. The corresponding data points are clearly along the hydrostatic curve. These results unmistakably indicate that the wall friction is the primary mechanism that creates the Janssen effect. Additionally, it is noted that the force fluctuations on the piston as discussed in Subsection 2.7.2 (Figure 2.5) were

not present in this simulation. Therefore, the fluctuations in the simulated piston force are caused by restructuring within the packed bed due to interactions with the wall.

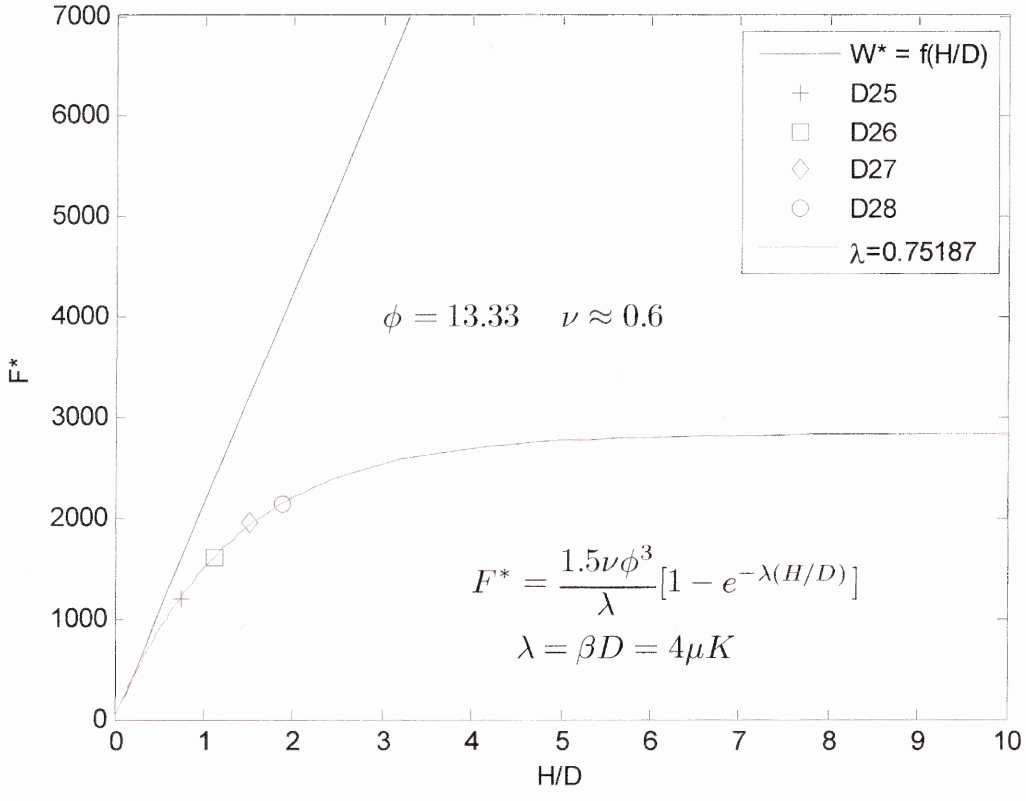

Figure 3.15 Simulated results for D25 (+), D26 (□), D27 (◊), D28 (o). The solid red line corresponds to a fit of D25 – D28 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.15 shows the results for simulations D25 – D28. In these simulations, the wall friction coefficient μ_w was set to 1.2 while the inter-particle friction coefficient μ_p was set to 0.1. Additionally, the diameter ratio ϕ was set to 13.33, and in these simulations the piston was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model.

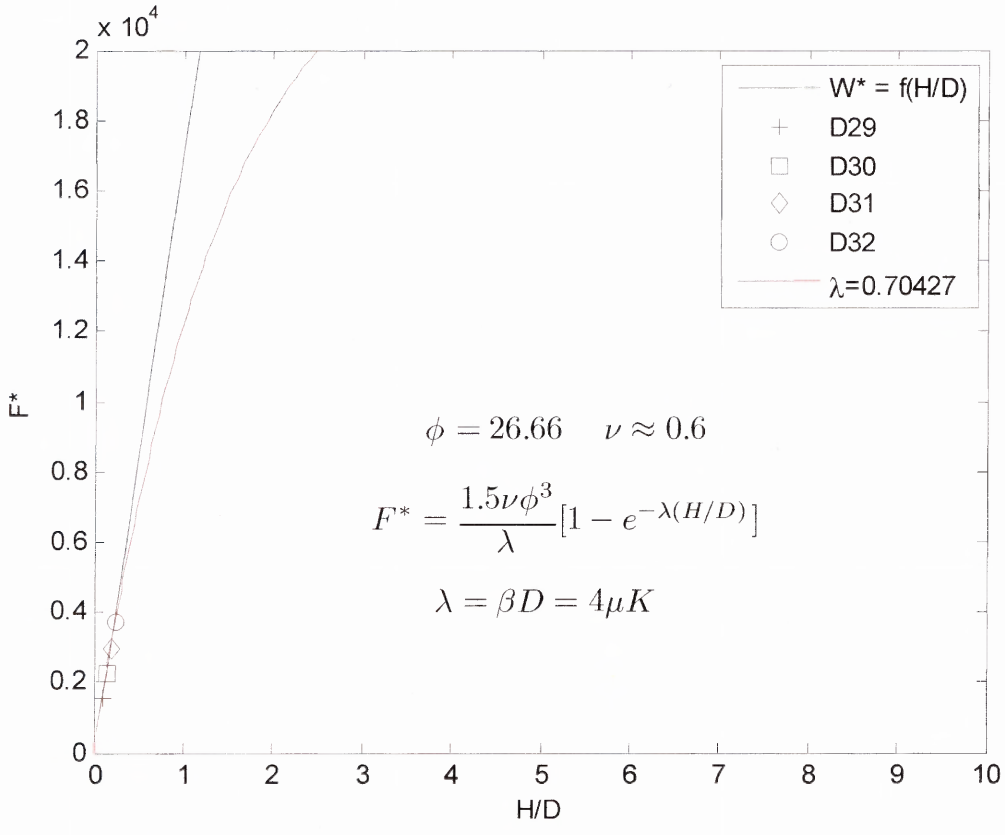

Figure 3.16 Simulated results for D29 (+), D30 (□), D31 (◊), D32 (o). The solid red line corresponds to a fit of D29 – D32 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.16 shows the results for simulations D29 – D32. In these simulations, the wall friction coefficient μ_w was set to 0.8, while the inter-particle friction coefficient μ_p was set to 0.1. Additionally, the diameter ratio ϕ was set to 26.66, and in these simulations the piston was translated to activate friction. The relatively small values of *H/D* show the need for more simulations at larger values of *H/D* to obtain a better fit of the data.

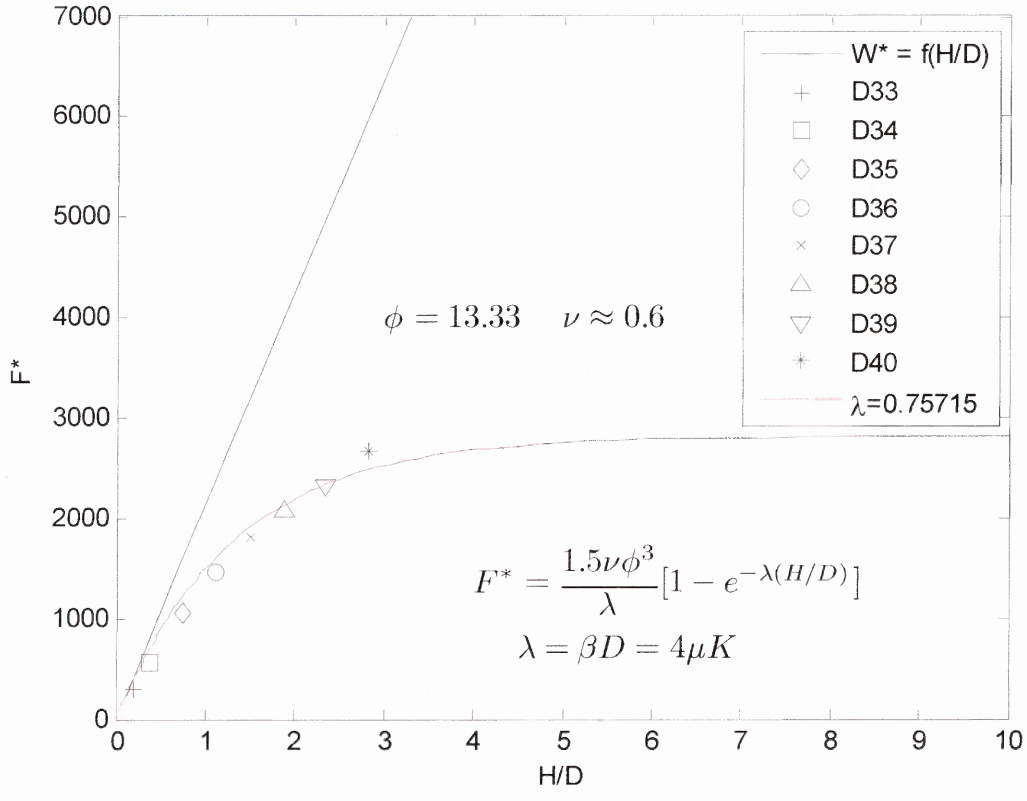

Figure 3.17 Simulated results for D33 (+), D34 (□), D35 (◊), D36 (o), D37 (x), D38 (△), D39 (▽), D40 (∗). The solid red line corresponds to a fit of D33 – D40 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.17 shows the results for simulations D33 – D40. In these simulations, the wall friction coefficient μ_w was set to 0.12, while the inter-particle friction coefficient μ_p was set to 0.1. Additionally, the diameter ratio ϕ was set to 13.33, and in these simulations the piston was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model

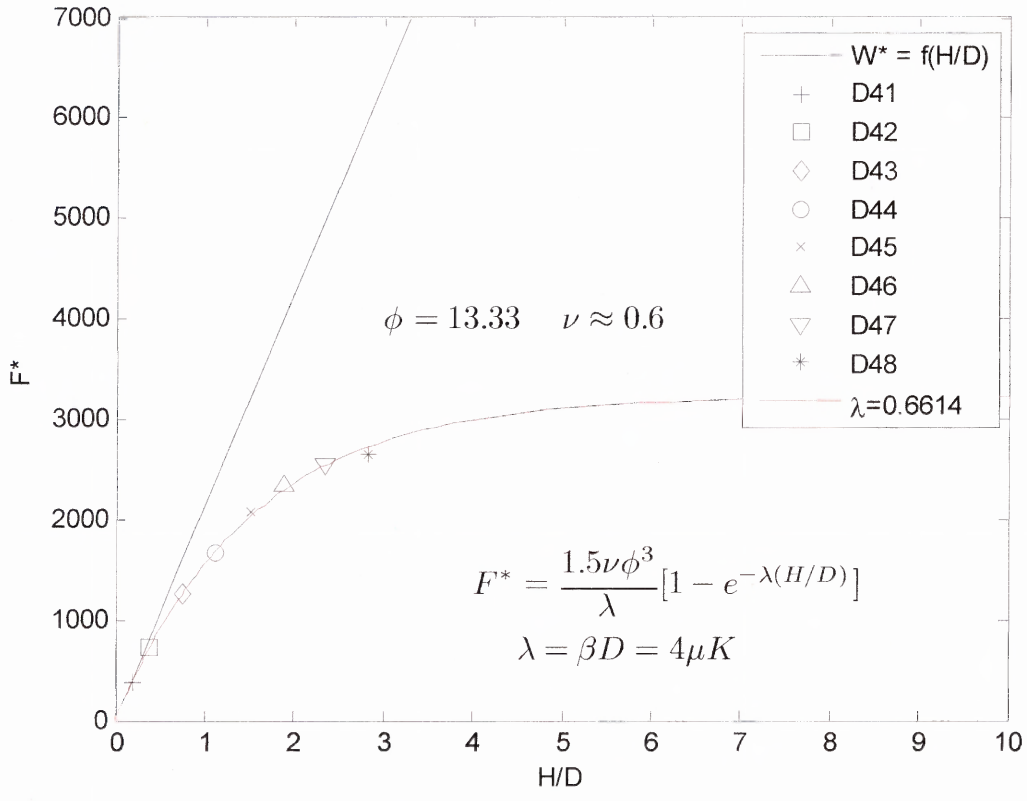

Figure 3.18 Simulated results for D41 (+), D42 (□), D43 (◊), D44 (o), D45 (x), D46 (△), D47 (▽), D48 (∗). The solid red line corresponds to a fit of D41 – D48 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.18 shows the results for simulations D41 – D48. In these simulations, the wall friction coefficient μ_w was set to 0.4, while the inter-particle friction coefficient μ_p was set to 0.1. Additionally, the diameter ratio ϕ was set to 13.33, and in these simulations the piston was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model.

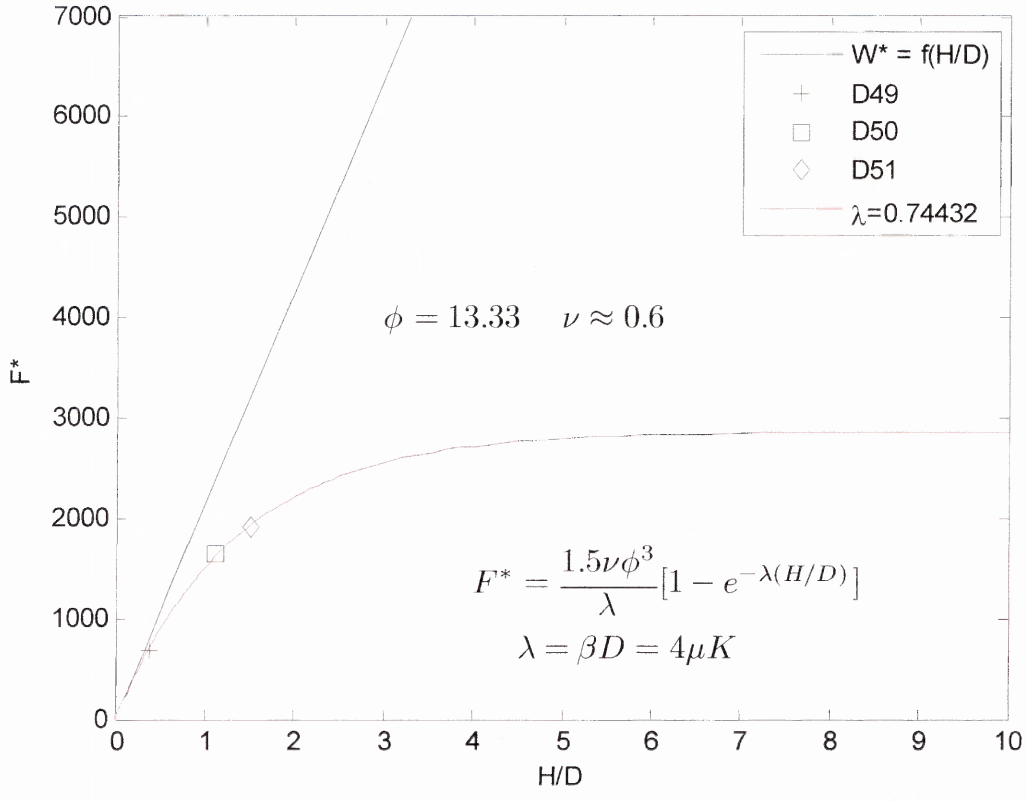

Figure 3.19 Simulated results for D49 (+), D50 (□), D51 (◊). The solid red line corresponds to a fit of D49 – D51 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.19 shows the results for simulations D49 – D51. In these simulations, the wall friction coefficient μ_w was set to 0.8, while the inter-particle friction coefficient μ_p was set to 0.1. Additionally, the diameter ratio ϕ was set to 13.33, and in these simulations the piston was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model.

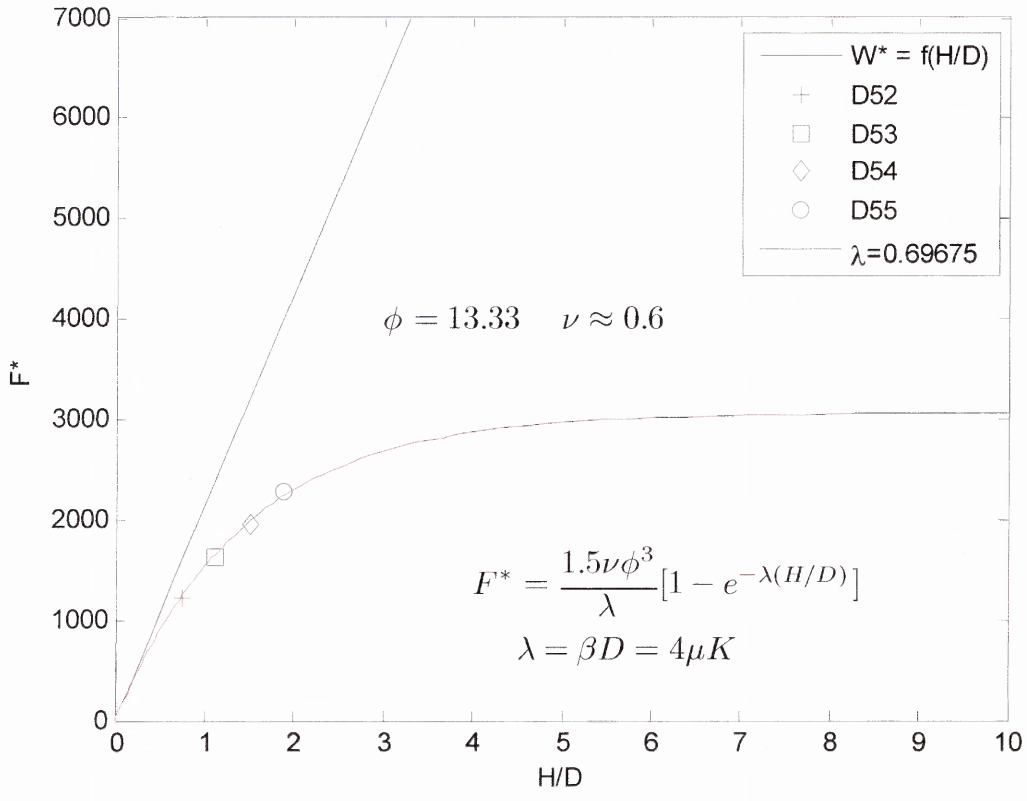

Figure 3.20 Simulated results for D52 (+), D53 (□), D54 (◊), D55 (o). The solid red line corresponds to a fit of D52 – D55 to the functional form of Equation (3.1). The solid black line represents the hydrostatic curve.

Figure 3.20 shows the results for simulations D52 – D55. In these simulations, the wall friction coefficient μ_w was set to 0.8, while the inter-particle friction coefficient μ_p was set to 0.1. Additionally, the diameter ratio ϕ was set to 13.33, and in these simulations the cylinder wall was translated to activate friction. As seen from the plot, the data is a good fit to the functional form of Janssen's model.

3.3.4 Comparison with Experiments from Literature

As mentioned before in Section 1.3 (Review of Published Literature), Walton [17] presents preliminary results for experiments that verify the exponential form of the piston vs. fill height relationship of Equations (1.5) and (1.6). The experiments were performed with 3mm glass beads in a 4cm acrylic tube, note that the diameter ratio used in these experiments is $\phi = 13.3$, the same as used in most of the simulations performed. Figure 3.21 shows a comparison of the simulated data from D33 – D51 against experimental data from Walton [17]. Simulations D33 – D40, D41 – D48, and D49 – D51 differ only with respect to one parameter, μ_w. It is easily observed from Figure 3.21 that with the careful selection of friction coefficients quantitatively accurate results may be obtained. Additionally, as discussed in Subsection 3.3.2, Figure 3.21 also highlights the effect of the friction coefficient.

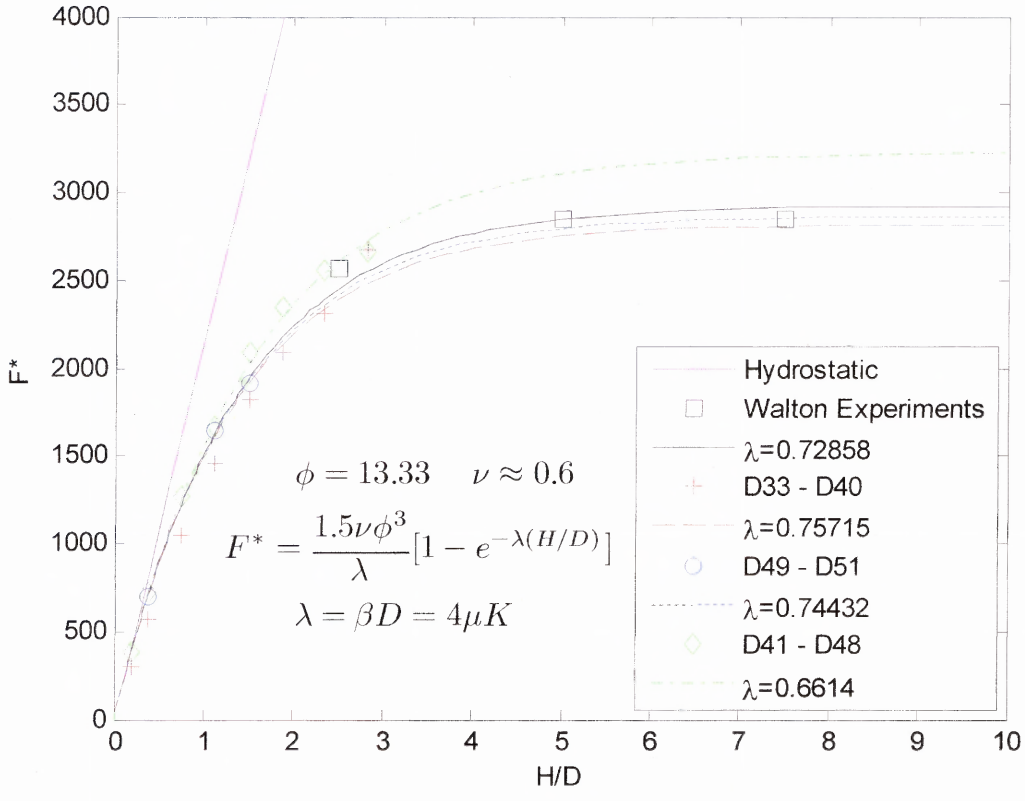

Figure 3.21 Comparison of simulated results to experimental results from literature. The solid magenta line corresponds to the hydrostatic curve. Experiments from Walton [17] are given by (□), and the solid black line is a fit of the experimental data [17] to Equation (3.1). Simulated results from D33 – D40 are given by (+), and the dashed red line is a fit of D33 – D40 to the functional form of Equation (3.1). Simulated results from D49 – D51 are given by (o), and the dashed blue line is a fit of D49 – D51 to the functional form of Equation (3.1). Simulated results from D41 – D48 are given by (◊), and the dashed green line is a fit of D41 – D48 to the functional form of Equation (3.1).

CHAPTER 4

CONCLUSIONS AND FURTHER WORK

4.1 Conclusions

Discrete element simulations were performed to measure the load on a piston supporting a monodisperse granular material of frictional inelastic spheres of diameter d, within a cylindrical vessel of diameter D. The diameter ratio ϕ was set to either 13.33 or 26.66 for the simulations. Both the inter-particle friction coefficient μ_p and the wall friction coefficient μ_w were used as simulation parameters and range from 0.0 to 1.2 in the simulations. Piston load values were extracted by performing careful statistical averaging over long duration runs.

Static simulations were performed in which frictional forces at the wall were not activated. For these simulations the load vs. fill height behavior was found to be hydrostatic in nature for values of H/D less than approximately 2. The behavior for values of H/D larger than approximately 2 is speculated to follow a Janssen type behavior as reported in the literature [13].

Dynamic simulations were performed in which the wall friction was activated by a relative displacement between the cylinder wall and the packed bed. The piston load vs. fill height behavior, the difference between translating the cylinder wall or the piston to activate wall frictional forces, and the effect of the wall friction coefficient and particle rotations were analyzed.

The simulated behavior of the piston load vs. fill level was found to fit well with the functional form of Janssen's theory for the diameter ratios simulated. Additionally, the simulated fill levels at which the loads asymptotes and the associated loads compare well with physical experiments reported in the literature [17]. Thus, indicating that the proper selection of friction coefficients will yield quantitatively accurate results.

Simulations were carried out with the intent of comparing results to test the differences in activating wall friction by either translation of the cylinder wall, or translation of the piston. With the exception of translating the piston, or the cylinder wall, the simulations are performed using the same simulation parameters. As mentioned previously the addition of more data is required for a more accurate comparison. However, it is speculated that a translation of either the floor piston or the cylinder wall to activate frictional forces will not affect the simulated load vs. fill height behavior.

The effect of a tangential force applied to the surface of the particles at the contact point rather than the center of the particle is also analyzed. As shown previously the tangential force causes a torque on the particle, which in consequently causes a rotation of the particle. The rate of rotation is proportionate to the wall friction coefficient. It was shown that an increase in the wall friction coefficient may not increase the resistance to sliding. However, the increase in wall friction coefficient causes a higher particle angular velocity. These rotations cause a violation of Janssen's assumption that frictional forces are fully activated.

Janssen's assumption that the lateral stress is a fixed fraction of the axial stress may not be true for the assemblies simulated in this thesis. A linear elastic solid would exhibit such behavior, as well as a granular assembly failing at the Coulomb failure limit.

There is no reason to expect the bulk material modeled behaves like either of these special cases. Additionally, the stress distribution across a horizontal layer may not be uniform in a granular assembly.

4.2 Further Work

The work from this thesis provides some direction for further work to better understand the assumptions of Janssen's theory. Listed below are some of the areas of most interest.

1. Simulations to compute the radial stress and axial stresses as a function of both radial and axial location so that the radial and axial stresses can be compared against Janssen's assumption. Additionally, Janssen's assumption that the stresses across any horizontal layer are uniform may also be verified through this computation. This type of simulation may prove extremely useful since physical experiments cannot measure the stresses in the bulk of the material.
2. Overload simulations to compare with experiments will provide another test of the Janssen model to predict the loads associated with a granular material.

APPENDIX A

MODIFICATIONS TO THE DEM CODE

This appendix details many of the changes the DEM code has undergone throughout this thesis. Changes are described by a brief header that indicates the subroutine the modification took place, the lines within that subroutine, and a concise outline of the purpose of the modification.

Main Program *3dshear.f* Lines 213 – 217
The following lines of code open the new output files required for a simulation. The file *zfloorforce* contains the current value of the piston load, the long term time average of the piston load, and the current time. The file zprr contains the radial pressure.

```
      open(38,file='zfloorforce',status='unknown')
      open(39,file='zprr',status='unknown')
      open(40,file='zprrcell',status='unknown')
      open(41,file='zcylforce',status='unknown')
      open(42,file='zrolling',status='unknown')
```

Main Program *3dshear.f* Lines 239 – 249
The following lines of code initialize parameters for the restarted run. The old values of tmax, istart, tstart, nout, dsump, dtdump, nrun, vamp, and frq are replaced by the new values. These variables are the only ones allowed to change during a restart.

```
c-----initalize parameters for new run
         tmax = tmax1
         istart = istart1
         tstart = tstart1
         nout = nout1
         dtout = dtout1
         ndump = ndump1
         dtdump = dtdump1
         nrun  = nrun1
         vamp  =vamp1
         frq   = frq1
```

Main Program *3dshear.f* Lines 293 – 294
The following lines of code are used to create the column headings for the output files zfloorforce and zcylforce. This section of code is located here so that it only writes once at the beginning of the simulation to make a column header, not every time step. This makes it easier for data analysis.

```
      write(38,*) '   force    ,  time avg   ,     time'
      write(41,*) '  radforce  ,  time avg   ,     time'
```

Subroutine *datain.f* Lines 21 – 35

The following lines of code refer to the namelist. The namelist is a way of reading data from the input file. If you want to add another variable to the input file, it must also be added here. Variables added for a cylindrical boundary include: nzcyl, nycyl, ncomp, compforce, t2move,t2stop,vyfloor

```
      namelist /var/ np,bdry,nxby0,nzby0,nxby1,nzby1
     + ,nxbz0,nybz0,nxbz1,nybz1,nybx0,nzbx0,nybx1,nzbx1
     + ,nfix,nzcyl,nycyl,ncomp,ncmax,nout
     + ,nczero,ntcol,nvel,ndump,nyzone,mzcell,nycell,itervm
     1 ,icoord,itty,ixyz,istart,ialtk
     2 ,tmax,tpour,dt,dtout,dtdump,tzero,pack
     + ,vave,vxzero,vyzero,vzzero,vseed
     3 ,skn1,elast,slope,ratk,fmu,power,tstart,rmassz
     4 ,xcell,ycell,zcell,xyrat,zyrat,gravx,gravy,gravz
     5 ,vxby0,vxby1,vyby0,vyby1,t2move,t2stop,vyfloor
     6 ,search,xmax,draddt,ystart,ystop,compforce
     7 ,radz,number,radius,x,y,z,skn1b,elastb,slopeb
     8 ,fmub,vx,vy,vz,wx,wy,wz
     9 ,vamp,frq
     + ,finis
```

Subroutine *datain.f* Lines 109 – 126

The following lines of code refer to the new variables nzcyl, nycyl, and ncomp. The values from the input file will override the values in *datain.f*. ncomp can be used to set a particle on top of the packed bed (y-cylinder only) that will exert an axial compression of compforce onto the bed.

```
c-----number of cylinders parallel to the z axis
c     if nzcyl is "1" and rad(ind1cz) is negative, then assume
c     that this zcylinder is the outer boundary. Note: Cannot have
c     both z-cylinders and y-cylinders.
      nzcyl = 0
c
c-----added by Shawn Chester, NJIT**************************
c-----------------------------------------------------------------------
c-----number of cylinders parallel to the y axis
c     if nycyl is "1" and rad(ind1cy) is negitive, then assume
c     that this y-cylinder is the outer boundry. Note: Cannot have
c     both z-cylinder and y-cylinder.
      nycyl = 0
c-----------------------------------------------------------------------
c
c-----number of particles used for axial compression
c     this should only be one
      ncomp = 0
```

Subroutine *datain.f* Lines 311 – 321
The following lines of code refer to the new variables t2move, t2stop, vyfloor. The values are overwritten by what is specified in the input file These variables can be used to move either the floor or the cylinder. More details in the *integ1.f* and *integ2.f* subroutines.

```
c-----t2move =  the time at which the floor will move with the
c               velocity vyfloor
      t2move = 0.0
c
c-----t2stop = the time at which the floor will stop moving
      t2stop = 0.0
c
c-----vyfloor=  the velocity in which the floor will move vertically
c               after the time is greater than t2move
c               note: positive moves the floor up, negitive down
      vyfloor = 0.0
```

Subroutine *datain.f* Lines 343 – 345
The following lines of code refer to the new variable compforce, which will provide the amount of force to exert on the top of the packed bed.

```
c-----compforce = force applied to the top of the packing
c-----to provide an axial compression of the packed bed
      compforce = 0.0
```

Subroutine *datain.f* Lines 365 – 370
The following lines of code ensure initialization of the particle locations and velocities. (minor modification)

```
        x(i)      = 0.
        y(i)      = 0.
        z(i)      = 0.
        vx(i)     = 0.
        vy(i)     = 0.
        vz(i)     = 0.
```

Subroutine *datain.f* Line 394
The following line of code ensures that the simulation dosent have both a z, and y cylinder. A simple error check.

```
      if((nzcyl.ge.1).and.(nycyl.ge.1)) go to 996
```

Subroutine *datasav2.f* Line 11
The following line of code initializes the variables needed to compute the CPU time used in the simulation. The time is added cumulatively.

```
      real  etime,timearray(2),tarray(2)
```

Subroutine *datasav2.f* Lines 30 – 33
The following line of code computes the CPU time used in the simulation. The time is added cumulatively.

```
c----------CPU Time------------------------------------------------
      cputime=etime(timearray)
      cputime=timearray(1)+timearray(2)
      write(3,304) t,cputime
```

Subroutine *datasav2.f* Lines 80 – 81
The following lines of code add a counter for long term time averaging.

```
c--modified by shawn chester
      savet = savet + 1
```

Subroutine *datasav2.f* Lines 108 – 111
The following lines of code add the current value of the stress to the running time averaged value. For all 9 components of the stress tensor.

```
      do 5 j=1,9
         pnnkt(j) = pnnkt(j) + pnnk(j)
         pnnpt(j) = pnnpt(j) + pnnp(j)
 5    continue
```

Subroutine *datasav2.f* Lines 202 – 214
The following lines of code in loop '25' add the current value of the stress to the running time averaged value, for all 9 components of the stress tensor in the y-zones. The lines of code in loop '26' do the same for the radial stress (Both kinetic and potential components). The line of code after loop '26' adds the current potential components of the radial stress to the old value for computing a long term average.

```
      do 25 j=1,9
      do 25 i=1,nyzone
         ypnnkt(i,j) = ypnnkt(i,j) + ypnnk(i,j)
         ypnnpt(i,j) = ypnnpt(i,j) + ypnnp(i,j)
 25   continue
c
c--modification by shawn chester for radial stress
      do 26 i=1,nyzone
         yprrpt(i) = yprrpt(i) + yprrp(i)
 26   continue
c
      prrpt = prrpt + prrp
```

Subroutine *datasav2.f* Lines 374 – 382
The first line of code below writes the particle coordinates to file 13, *zposition*. The lines of code in loop '109' keep the long term average of the particle angular velocity. The lines of code following loop '109' write the values of the angular velocity to file *zrolling*. The radial location *raddist* is printed, along with the current value of the particle angular velocity *angvel*, and the long term time average *angvelt*.

```
      write(13,312) (i,x(i),y(i),z(i),i=1,np)
c----print the angular velocity and radial location
      do 109 i=1,np
         angvel(i) = angvel(i) + dangvel(i)
         angvelt(i) = angvelt(i) + angvel(i)
 109  continue
      write(42,202) (i,raddist(i),angvel(i),angvelt(i)*saveti,i=1,np)
 202  format(/,4x,"i",4x,"raddist(i)",6x,"angvel(i)",5x,"angvelt(i)",
     1         1p,/,(1x,i4,3e15.7))
```

Subroutine *datasav2.f* Line 385
The line of code below calls the subroutine *packfrac.f* to compute the packing fraction as a function of height. The line is commented out to save computational time.

```
c      call packfrac
```

Subroutine *datasav2.f* Lines 401 – 412
The lines of code in loop '85' write the potential component of the stress tensor to the file *ztensor*. The next few lines of code write the radial component of the potential component of the stress in the y-zones to file. The last lines of code write the radial stress to file for the entire computational cell.

```
      do 85 j=1,9
         write(15,320) (j,labz(i),ypnnp(i,j),ypnnpt(i,j)*saveti
     1                  ,i=1,nyzone)
         write(15,320) j,labcell,pnnp(j),pnnpt*saveti
 85   continue
c
         write(39,200) (i,labz(i),yprrp(i),yprrpt(i)*saveti
     1                   ,i=1,nyzone)
 200  format(1x,"stress : pot. (",i4,")  ", a8, "  = ",1p,2e12.4)
c
         write(40,201) pnnp,pnnpt*saveti
 201  format (1x,"radial stress for entire cell = ",1p,2e12.4)
```

Subroutine *datasav2.f* Lines 415 – 418

The following lines of code are used to obtain the piston force. pywall is the current value of the load on the piston, and pywalt is the long term average load on the piston. The values of the current piston load, the long term time averaged piston load, and the time are written to file.

```
        pywall = pywall + dpywal
        pywalt = pywalt + pywall
           write(38,199) pywall,pywalt*saveti,t
 199    format(e12.5," ,",e12.5," ,",e12.5)
```

Subroutine *datasav2.f* Lines 420 – 423

The following lines of code are used to obtain the force exerted on the cylindrical boundary. prcyl is the current value of the load on the wall, and prcylt is the long term average load on the wall. The values of the current wall load, the long term time averaged wall load, and the time are written to file.

```
        prcyl = prcyl + dprcyl
        prcylt = prcylt + prcyl
        write(41,198) prcyl,prcylt*saveti,t
 198    format(e12.5," ,",e12.5," ,",e12.5)
```

Subroutine *deletem.f* Lines 37 – 38
The following lines of code are used to modify collision detection for the cylindrical boundary.

```
        rsum = abs(rad(i)) + abs(rad(j))
        if((j.ge.ind1cy).and.(j.le.ind2cy)) rsum = rad(i)
```

Subroutine *deletem.f* Lines 47 – 69
The lines of code below are used to determine the vector connecting the contacting particle centers. In the case of a free particle contacting the cylinder the modifications are as follows. The y component of the vector is set to zero to create the cylinder wall. Then contact forces are directed along the vector from the center of the cylinder to the particle, however opposite direction, such that particles are contained within the cylinder.

```
c-----z cylinders
        if((j.ge.ind1cz).and.(j.le.ind2cz)) rz = 0.
c-------------------------------------------------------------------
c-----added by Shawn Chester 2003, NJIT ******************************
c-------------------------------------------------------------------
c-----y cylinders
        if((j.ge.ind1cy).and.(j.le.ind2cy)) then
          ry = 0.0
c------see forces.f for an explaination of the statements below
          contactangle = atan(z(i)/x(i))
          xcp = sign(rad(j)*cos(contactangle),x(i))
          zcp = sign(rad(j)*sin(contactangle),z(i))
           if(z(i).eq.0) then
              xcp = sign(rad(j),x(i))
              zcp = 0.0
           endif
           if(x(i).eq.0) then
              xcp = 0.0
              zcp = sign(rad(j),z(i))
           endif
          rx = xcp - x(i)
          rz = zcp - z(i)
        endif
```

Subroutine *deletem.f* Lines 89 – 92
The lines of code below are used to determine the vector connecting the contacting particle centers. In the case of a free particle contacting the particle used for axial compression the modifications are as follows. The x and z components are set to zero so that the contact is perfectly in the axial direction. This particle will act as a wall on the top of the packed bed.

```
         elseif((j.eq.indcomp).and.(ncomp.eq.1)) then
            rx = 0.0
            rz = 0.0
        endif
```

Subroutine *diagnos2.f* Lines 346 – 349
The lines of code below are used to compute the radial distance from the cylinder axis to the center of a particle. Then the magnitude of the angular velocity of that particle is calculated. This is used for particle rotations as discussed in Section 2.7.4.

```
      do 151 i=1,np
         raddist(i) = sqrt(x(i)*x(i) + z(i)*z(i))
         dangvel(i) = sqrt(wx(i)*wx(i) + wy(i)*wy(i) + wz(i)*wz(i))
151   continue
```

Subroutine *diagnos2.f* Lines 352 – 354
The lines of code below are used in the computation of the radial stress in the y zones. The current value of the potential component of the radial stress is computed.

```
      do 155 i=1,nyzone
         yprrp(i) = yprrp(i) + dyprrp(i)*voyzoni(i)
155   continue
```

Subroutine *diagnos2.f* Lines 356
The line of code below is used in the computation of the radial stress in the entire computational cell. The current value of the potential component of the radial stress is computed.

```
      prrp = prrp + dprrp*vcelli
```

Subroutine *diagnos2.f* Lines 359 – 363
The lines of code below are used in the computation of the radial stress in the annular zones. The annular zones have an index for radial zone and an index for y zone.

```
      do 180 n=1,nrzone
      do 181 m=1,nyzone
         anprrp(n,m) = anprrp(n,m) + danprrp(n,m)*vanzone(n,m)
181   continue
180   continue
```

Subroutine *forces.f* Line 73
The line of code below is used with the cylindrical boundary contact detection.

```
        if((j.ge.ind1cy).and.(j.le.ind2cy)) rsum = rad(i)
```

Subroutine *forces.f* Lines 87 – 113
The following lines of code deal with computing the vector the contact force lies along between a free particle and the cylinder wall. See modifications from subroutine *deletem.f* for a description.

```
        if((j.ge.ind1cz).and.(j.le.ind2cz)) rz = 0.
c-------------------------------------------------------------------
c-----added by Shawn Chester 2003, NJIT ******************************
c-------------------------------------------------------------------
c-----y cylinders
        if((j.ge.ind1cy).and.(j.le.ind2cy)) then
           ry = 0.0
c------the contactanle variable is the angle at which the free particle
"i"
c        is going to hit the cylinder boundary.  In other words it is
used to
c      find the x and y coordinate of the contact point, xcp and zcp,
c      on the cylinder where the particle will contact.
c
           contactangle=atan(abs(z(i)/x(i)))
              xcp = sign(rad(j)*cos(contactangle),x(i))
              zcp = sign(rad(j)*sin(contactangle),z(i))
           if(z(i).eq.0) then
                  xcp = sign(rad(j),x(i))
                  zcp = 0.0
           endif
           if(x(i).eq.0) then
               xcp = 0.0
               zcp = sign(rad(j),z(i))
           endif
           rx = xcp - x(i)
           rz = zcp - z(i)
        endif
```

Subroutine *forces.f* Lines 136 – 139
The following lines of code deal with computing the vector the contact force lies along for the case of the axial compression particle contacting a free particle. The axial compression particle has only an axial component of force, i.e., it acts like a wall.

```
         elseif((j.eq.indcomp).and.(ncomp.eq.1)) then
           rx = 0.0
           ry = 0.0
        endif
```

Subroutine *forces.f* Lines 193 – 195

The following lines of code compute the square of the distance between particle centers for the case of a free particle contacting the cylinder wall. The distance squared is computed as the radial distance to the particle center subtracted from the cylinder radius quantity squared.

```
        if((j.ge.ind1cy).and.(j.le.ind2cy)) then
           rijsq = (abs(rad(j)) - sqrt(x(i)**2 + z(i)**2))**2
        endif
```

Subroutine *forces.f* Lines 498 – 546

The following lines of code compute the potential component of the radial stress component. Note that neither the piston, nor the particle used for axial compression adds to the radial stress. The radial distance from the center of the cylinder to the center of the particle is computed for each contacting pair. Then the difference is found as *raddistij*. *xc*, *yc*, and *zc* are the x,y, and z coordinates of the contact point between particles *i* and *j*. That location is used to compute the radial distance to the contact point. *xck*, and *zck* are the components of the unit vector that determines the direction of the radial force between contacting pairs. The magnitude of the radial force is then computed by projecting the contact force onto this vector. Then the stress increment can be computed. The last few lines of code increment the radial stress component in the y zones.

```
c--if floor or compression particle -> no effect on prr
         if((j.eq.ind1y0).or.(j.eq.indcomp)) goto 60
c
         if(j.eq.ind1cy) goto 61
c
         raddistj = sqrt((x(j)*x(j))+(z(j)*z(j)))
         raddisti = sqrt((x(i)*x(i))+(z(i)*z(i)))
         raddistij = raddistj - raddisti
c
c     fix this later for checking **************
c         if(radjsq.eq.radisq) goto 60
c
         xc = (x(j) + x(i))/2
         zc = (z(j) + z(i))/2
         yc = (y(j) + y(i))/2
c
         contrad = sqrt((xc*xc)+(zc*zc)+(yc*yc))
c
c--then calculate the unit vectors for the radial direction
c---using the x and z directions to do this
c
         xck = xc/contrad
         zck = zc/contrad
c
c--calculate the force in radial direction
c
         radfx = ftotx*xck
         radfz = ftotz*zck
         radialforce = sqrt((radfx*radfx)+(radfz*radfz))
c
```

```
c--stress increment
c
         rrfr = radialforce*raddistij
 61      continue
         if(j.eq.indlcy) then
c------just a test to see about ignoring the wall effect
            rrfr = 0.0
c            rrfr = fnij*rad(i)
         endif
c
c--increment the radial stress for cell (not useful, or good)
         dprrp = dprrp + rrfr
c--increment the radial stress (potential part) for the y zones
         dyprrp(npos(i)) = dyprrp(npos(i)) + half*rpos(i)*rrfr
         dyprrp(nmid(i)) = dyprrp(nmid(i)) + half*rmid(i)*rrfr
         dyprrp(nneg(i)) = dyprrp(nneg(i)) + half*rneg(i)*rrfr
         dyprrp(npos(j)) = dyprrp(npos(j)) + half*rpos(i)*rrfr
         dyprrp(nmid(j)) = dyprrp(nmid(j)) + half*rmid(i)*rrfr
         dyprrp(nneg(j)) = dyprrp(nneg(j)) + half*rneg(i)*rrfr
```

Subroutine *forces.f* Lines 548 – 554

The following lines of code are used to compute the radial stress increment used for the annular zones. The stress increment is mass weighted such that the fraction of the mass of particle "i" in a zone is the fraction of the stress increment for that zone. This part of the code is only partially complete.

```
c     radial stress increments in the annular zones
         do 28 n=1,nrzone
            do 29 m=1,nyzone
            danprrp(rzone,yzone) = vfrac(n,m,i)*rrfr
 29         continue
 28      continue
```

Subroutine *forces.f* Lines 565 – 572

The following lines of code are used to increment the potential components of the stress tensor for the y zones.

```
c-----increment stress tensor components (potential) for y zones
         dypnnp(npos(i),k) = dypnnp(npos(i),k) + half*rpos(i)*rnfn(k)
         dypnnp(nmid(i),k) = dypnnp(nmid(i),k) + half*rmid(i)*rnfn(k)
         dypnnp(nneg(i),k) = dypnnp(nneg(i),k) + half*rneg(i)*rnfn(k)
         dypnnp(npos(j),k) = dypnnp(npos(j),k) + half*rpos(j)*rnfn(k)
         dypnnp(nmid(j),k) = dypnnp(nmid(j),k) + half*rmid(j)*rnfn(k)
         dypnnp(nneg(j),k) = dypnnp(nneg(j),k) + half*rneg(j)*rnfn(k)
 30   continue
```

Subroutine *forces.f* Lines 574 – 579

The following lines of code are used to increment the force on the lower boundaries. In this thesis only the lower y boundary (piston) is considered.

```
c-----increment wall forces on lower y-boundary particles
```

```
         if((j.ge.ind1y0).and.(j.le.ind2y0)) then
            dpxwal = dpxwal + ftotx
            dpywal = dpywal + ftoty
            dpzwal = dpzwal + ftotz
         endif
```

Subroutine *forces.f* Lines 582 – 590
The following lines of code are used to increment the radial force on the cylinder wall. In this thesis only the total radial force on the wall is considered.

```
         if(j.eq.ind1cy) then
            dprcyl = dprcyl + radialforce
c--for the y-zones
c              do 31 i=1,nyzone
c              dyprcyl(npos(i)) = dyprcyl(npos(i)) + rpos(i)*radialforce
c              dyprcyl(nmid(i)) = dyprcyl(nmid(i)) + rmid(i)*radialforce
c              dyprcyl(nneg(i)) = dyprcyl(nneg(i)) + rneg(i)*radialforce
c 31           continue
         endif
```

Subroutine *init.f* Lines 145 – 147
The following lines of code are used to find the free particle indices. The free particles are from *ind1* until *ind2*.

```
      ind1 = 1
      ind2 = np - nby0 - nby1 -  nbz0 - nbz1 - nbx0 - nbx1 - nfix
     + - nzcyl - nycyl - ncomp
```

Subroutine *init.f* Lines 235 – 242
The following lines of code are to index the z cylinder particle. The z cylinder particles are from *ind1cz* until *ind2cz*.

```
      if(nzcyl.ge.1) then
       ind1cz = max(ind2, ind2y0, ind2y1, ind2x0
     + , ind2x1, ind2z0, ind2z1, ind2fx) + 1
       ind2cz = ind1cz + nzcyl - 1
       else
        ind1cz = 0
        ind2cz = 0
      endif
```

Subroutine *init.f* Lines 248 – 255
The following lines of code are to index the y cylinder particle. The y cylinder particles are from *ind1cy* until *ind2cy*.

```
      if(nycyl.ge.1) then
       ind1cy = max(ind2, ind2y0, ind2y1, ind2x0
     + , ind2x1, ind2z0, ind2z1, ind2fx, ind2cz) + 1
       ind2cy = ind1cy + nycyl - 1
       else
        ind1cy = 0
        ind2cy = 0
      endif
```

Subroutine *init.f* Lines 258 – 261
The following lines of code are to index the axial compression particle. The axial compression particle is *indcomp*.

```
      if(ncomp.eq.1.) then
         indcomp = max(ind2, ind2y0, ind2y1, ind2x0
     + , ind2x1, ind2z0, ind2z1, ind2fx, ind2cz, ind2cy) + 1
      endif
```

Subroutine *init.f* Lines 319 – 320
The following lines of code are used to compute the volume of the z and y cylinders.

```
         if((i.ge.ind1cz).and.(i.lt.ind1cy)) vol(i)=pi*radz(i)^2*zcell
         if(i.ge.ind1cy) vol(i) = pi*radz(i)*radz(i)*ycell
```

Subroutine *init.f* Lines 354 – 359
The following lines of code are used to compute the mass of the particle used for axial compression from the user input axial force.

```
      if(ncomp.ge.1) then
         pmass(indcomp) = compforce / 9.81
         if(pmass(indcomp).le.0.0) pmass(indcomp) = trifle
         rmass(indcomp) = compforce / 9.81
         if(rmass(indcomp).le.0.0) rmass(indcomp) = trifle
      endif
```

Subroutine *init.f* Lines 478 – 486
The following lines of code are used to convert the negative radius flag on the cylinder boundary to a positive value *rycyl* used in computations.

```
c-----If wanted boundry is a cylinder in the y direction
c-----you need to have one y-cylinder with a negitive radius
c-----the following converts it into a positive radius
c-----added by Shawn Chester 2003, NJIT ****************************
c-------------------------------------------------------------------------
----
         if((nycyl.eq.1).and.(radz(indlcy).lt.0)) then
           rycyl = abs(radz(indlcy))
         endif
```

Subroutine *init.f* Line 504
The following line of code is used to initialize the velocity of the axial compression particle.

```
      if(ncomp.eq.1) vy(indcomp) = 0.0
```

Subroutine *init.f* Lines 524 – 538
The following lines of code are used to randomly place the free particles inside the cylinder.

```
      IF((nycyl.eq.1).and.(radz(indlcy).lt.0)) then
        delr = rycyl - radz(i)
        xminp = x(indlcy) - delr
        xmaxp = x(indlcy) + delr
        zminp = z(indlcy) - delr
        zmaxp = z(indlcy) + delr
        x(i) = xminp + rand(0.)*(xmaxp - xminp)
        z(i) = zminp + rand(0.)*(zmaxp - zminp)
        delx = x(i) - x(indlcy)
        delz = z(i) - z(indlcy)
        if((nby0.gt.1).or.(nby1.gt.1)) then
           yminp = radz(i)
           ymaxp = ycell - radz(i)
        endif
        y(i) = yminp + rand(0.)*(ymaxp - yminp)
```

Subroutine *init.f* Lines 588 – 592
The following lines of code are used to place the axial compression particle atop the packed bed.

```
      if(ncomp.eq.1) then
         x(indcomp) = 0.0
         z(indcomp) = 0.0
         y(indcomp) = ymaxp + rad(indcomp)
      endif
```

Subroutine *init.f* Lines 611 – 615
The following lines of code are used to check for particle overlap during the random placement of free particles with the cylinder.

```
         if((j.ge.ind1cy).and.(j.le.ind2cy)) then
          ry = 0.0
          rx = abs(rad(j)) - x(i)
          rz = abs(rad(j)) - z(i)
         endif
```

Subroutine *init.f* Lines 635 – 638
The following lines of code are used to check for particle overlap during the random placement of free particles with the axial compression particle.

```
         elseif((j.eq.indcomp).and.(ncomp.eq.1)) then
            rx = 0.0
            rz = 0.0
         endif
```

Subroutine *init.f* Lines 722 – 726
The following lines of code are used to initialize the particle sizes for boundary and other, non-free particles.

```
      if((nfix.gt.0).or.(nycyl.gt.0).or.(ncomp.eq.1)) then
        do 29 i=ind1fx, indcomp
                rad(i) = radz(i)
 29     continue
      endif
```

Subroutine *init.f* Lines 742 – 745
The following lines of code are used to initialize the cell volume for a cylindrical boundary.

```
      if((np.ge.ind1cy).and.(rad(ind1cy).lt.0)) then
        vcell = vol(ind1cy)
        vcelli = 1.0/vcell
      endif
```

Subroutine *init.f* Line 750
The following line of code computes the width of the annular zones. This portion of the code is partially complete.

```
      drzone = rycyl/nrzone
```

Subroutine *init.f* Lines 750 – 834
The following lines of code are used to compute the volume in the y zones.

```
      if((nycyl.eq.1).and.(rad(ind1cy).lt.0.)) then
         voyzone(1) = rycyl*rycyl*pi*dyzone - vbound
         voyzoni(1) = 1./voyzone(1)
         do 37 i=2,nyzone
            voyzone(i) = rycyl*rycyl*pi*dyzone
            voyzoni(i) = 1./voyzone(i)
37       continue
      endif
```

Subroutine *initcum1.f* Lines 47 – 56
The following lines of code initialize the current values of the stress tensor and the potential component of the radial stress.

```
      do 5 j=1,9
         pnnk(j) = 0.
         pnnp(j) = 0.
 5    continue
      prrp = 0.
```

Subroutine *initcum1.f* Lines 62 – 64
The following lines of code initialize the current value of the magnitude of the angular velocity of each particle.

```
      do 17 i=1,np
         angvel(i) = 0.
 17   continue
```

Subroutine *initcum1.f* Lines 105 – 115
The following lines of code initialize the current values of the stress tensor and the potential component of the radial stress for the y zones.

```
      do 15 j=1,9
      do 15 i=1,nyzone
         ypnnk(i,j) = 0.
         ypnnp(i,j) = 0.
 15   continue
c
c      modification by shawn chester for the radial stress
c
      do 16 i=1,nyzone
         yprrp(i) = 0.
 16   continue
```

Subroutine *initcum2.f* Lines 47 – 54
The following lines of code initialize the long term time averaged values of the stress tensor and the potential component of the radial stress.

```
      do 5 j=1,9
         pnnkt(j) = 0.
         pnnpt(j) = 0.
 5    continue
c
c--modification by shawn chester for the radial stress
c
      prrpt = 0.
```

Subroutine *initcum2.f* Lines 57 – 59
The following lines of code initialize the long term time averaged value of the magnitude of the angular velocity of each particle.

```
      do 17 i=1,np
         angvelt(i) = 0.
 17   continue
```

Subroutine *initcum2.f* Lines 98 – 107
The following lines of code initialize the long term time averaged values of the stress tensor and the potential component of the radial stress for the y zones.

```
      do 15 j=1,9
      do 15 i=1,nyzone
         ypnnkt(i,j) = 0.
         ypnnpt(i,j) = 0.
 15   continue
c
c--modification by shawn chester for radial stress
      do 16 i=1,nyzone
         yprrpt(i) = 0.
 16   continue
```

Subroutine *initcum2.f* Lines 113 – 115
The following lines of code initialize the long term time averaged values of the lower boundary forces.

```
      pxwalt = 0.
      pywalt = 0.
      pzwalt = 0.
```

Subroutine *initstep.f* Lines 43 – 46
The following lines of code initialize the increment values of the stress tensor.

```
      do 5 j=1,9
         dpnnk(j) = 0.
         dpnnp(j) = 0.
5     continue
```

Subroutine *initstep.f* Lines 75 – 79
The following lines of code initialize the increment values of the stress tensor in the y zones.

```
      do 15 j=1,9
      do 15 i=1,nyzone
         dypnnk(i,j) = 0.
         dypnnp(i,j) = 0.
15    continue
```

Subroutine *initstep2.f* Lines 113 – 115
The following lines of code initialize the increment values of the lower boundary forces.

```
      dpxwal = 0.
      dpywal = 0.
      dpzwal = 0.
```

Subroutine *initstep.f* Lines 89 – 91
The following lines of code initialize the increment value of the magnitude of the angular velocity of each particle.

```
      do 16 i=1,np
         dangvel(i) = 0.
16    continue
```

Subroutine *initstep.f* Lines 94 – 97
The following lines of code initialize the increment value of the radial stress potnetial component.

```
      dprrp = 0.
      do 17 i=1,nyzone
         dyprrp(i) = 0.
17    continue
```

Subroutine *integ1.f* Lines 95 – 101
The following lines of code are implemented to give the cylinder wall an axial velocity. The wall will start to move with a user input velocity *vyfloor*, at a user input *t2move*, and stop at user input *t2stop*.

```
      if((t.ge.t2move).and.(t.le.t2stop)) then
         vy(ind1cy) = vyfloor
         dvy(ind1cy) = 0.0
      else
         vy(ind1cy) = 0.0
         dvy(ind1cy) = 0.0
      endif
```

Alternate code to move the piston

```
      if((t.ge.t2move).and.(t.le.t2stop)) then
         vy(ind1cy) = vyfloor
         dvy(ind1cy) = 0.0
      else
         vy(ind1cy) = 0.0
         dvy(ind1cy) = 0.0
      endif
```

Subroutine *integ1.f* Lines 106 – 117
The following lines of code are implemented to give the axial compression particle the ability to move under gravity. This will ensure that the axial force is always applied to the top of the packed bed.

```
      if(ncomp.eq.1) then
      dvy(indcomp) = (fy(indcomp)*rmasi + gravy )*dt
         if(t.le.0.) then
c-----first time step
c-----trans. velocities at half time step before time zero
            vhy(indcomp) = vy(indcomp) - half*dvy(indcomp)
         else
c-----other than first time step
c-----trans. velocities at start of current time step
            vy(indcomp) = vhy(indcomp) + half*dvy(indcomp)
         endif
      endif
```

Subroutine *integ2.f* Line 23
The following line of code is used to allow the computation of the position and velocity of the particles in the simulation. By changing the upper limit in this loop the compression particle along with the piston particle are included in this calculation. This allows those particles to translate.

```
do 30 i=ind1,indcomp
```

Subroutine *integ2.f* Lines 93 – 101
The following lines of code are implemented to solve for the new position of the cylinder wall under translation, and the velocity of the wall.

```
if(vy(ind1cy).ne.0.) then
   dy(ind1cy) = vyfloor*dt
   y(ind1cy) = y(ind1cy) + dy(ind1cy)
   vy(ind1cy) = vyfloor
endif
```

Subroutine *packfrac.f*

The following subroutine computes the packing via the plane growth method. This code was modified from a standalone code and integrated into the main program.

```
=====================================================================
C             Packing volume fraction by plane growth method
C
=====================================================================
C
C     Variables
C
C     height  height of the box
C     rad     radius of the sphere
C     tvol    Volume of the container
C     svol    Volume of the spheres
C     zvol    zone volume inside the cointainer
C   psvol   partial volume of the spheres in the current partial volume
C     top     top plane of the zone
C     bottom  bottom plane of the zone
C     inc     amount of distance to increment the top plane
C   ninc    number of times to increment the top plane, this value must
C             be an integer value
C      ---------------------------------------------------------------
--
C     This modification will calculate the packing volume fraction for
C     a cylindrical boundry and not a rectangular boundry.  In addition
c     the packing fraction is calculated in zones.
C
=====================================================================
C                         Beginning of Program
C
=====================================================================
C
      subroutine packfrac
      include 's3dscmm'
c
      integer nu,n1,n2,n3,n4,n5,n6,ninc
      double precision inc,top,zvol,psvol,dzone,radcy,test
c
      Pi = 3.141592653589798
      radcy = abs(rad(ind1cy))
c
      top = 0.0D0
      bottom = y(ind1y0)
      dzone = 5*rmin
      height = (2.*rmax*rmax*rmax*ind2)/
     :         (0.6*radcy*radcy)
      tpsvol = 0.0D0
      test = height/dzone
      ninc = dnint(test)
c
      write(53,201) t
 201  format(/,/,"Time = ",e12.4)
C
C----begins loop of slicing the packing volume
c
```

```
      DO 1 i = 1, ninc
c
       top = (y(ind1y0)+rad(ind1y0))+(dzone*i)
       bottom = (y(ind1y0)+rad(ind1y0))+(dzone*(i-1))
       nu = 0
       n1 = 0
       n2 = 0
       n3 = 0
       n4 = 0
       n5 = 0
       n6 = 0
       v1 = 0.0D0
       v2 = 0.0D0
       v3 = 0.0D0
       v4 = 0.0D0
       v5 = 0.0D0
       v6 = 0.0D0
       psvol = 0.0D0
C
C----begins loop of particle volume calculation
c
      DO 4 k = ind1, ind2
C
c---case 1) when the particle lies totally out of the zone
c
         IF((y(k).gt.(top+rad(k))).or.
     1     (y(k).lt.(bottom-rad(k)))) GOTO 4
c
C---case 2) when the center of the sphere is above the zone, but
c           a little of the sphere inside of the zone
c
         IF ((y(k).le.(top+rad(k))).and.
     1        (y(k).gt.top)) THEN
c
                n2 = n2 + 1
                h = top + rad(k) - y(k)
                v2 = v2 + ((1.0D0/3.0D0)*Pi*h*h*(3*rad(k)-h))
c
C---case 3) when the center of the sphere is below the top, but
c           a little of the sphere is above the zone
c
         ELSE IF ((y(k).le.top).and.
     1            (y(k).gt.(top - rad(k)))) THEN
c
                 n3 = n3 + 1
                 h = y(k)+rad(k)-top
                 v3=v3+((4.0D0/3.0D0)*Pi*rad(k)*rad(k)*rad(k)
     1              -((1.0D0/3.0D0)*Pi*h*h*(3*rad(k)-h)))
c
C---case 4) when all of the sphere is inside sample element
c
         ELSE IF ((y(k).le.(top-rad(k))).and.
     1            (y(k).gt.(bottom+rad(k)))) THEN
c
                 n4 = n4 + 1
                 v4 = v4+((4.0D0/3.0D0)*Pi*rad(k)*rad(k)*rad(k))
c
```

```
C---case 5) when the center of the sphere is just above the bottom,
c           and a little of the sphere is outside the zone
c
         ELSE IF ((y(k).gt.bottom).and.
     1            (y(k).le.(bottom+rad(k)))) THEN
                  n5 = n5 + 1
                  h = bottom + rad(k) - y(k)
                  v5 = v5 + ((4.0D0/3.0D0)*Pi*rad(k)*rad(k)*
     1                 rad(k)-(1.0D0/3.0D0)*Pi*h*h*(3*rad(k)-h))
c
C---case 6) when the center of the sphere is just below the bottom,
c           but some of the sphere is inside the zone
c
         ELSE IF ((y(k).le.bottom).and.
     1            (y(k).gt.(bottom-rad(k)))) THEN
c
                  n6 = n6 + 1
                  h = y(k) + rad(k) - bottom
                  v6 = v6 + ((1.0D0/3.0D0)*Pi*h*h*(3*rad(k)-h))
c

          ENDIF
c
 4      CONTINUE
c
         n1 = ind2 - n2 - n3 - n4 - n5 - n6
         nu = n3 + n4 + n5
c
         zvol = radcy*radcy*Pi*dzone
         psvol = v2 + v3 + v4 + v5 + v6
         tpsvol = tpsvol + psvol
         pd = psvol/zvol
c
       write(53,210) i
 210   format("Zone",i3,/,8x,"n1",3x,"n2",3x,"n3",3x,"n4",
     :       3x,"n5",3x,"n6", 5x,"nu")
       write(53,211) n1,n2,n3,n4,n5,n6,nu
 211   format(7x,i3,2x,i3,2x,i3,2x,i3,2x,i3,2x,i3,2x,i5)
       write(53,212)
 212   format(6x,"v1",11x,"v2",11x,"v3",10x,"v4",10x,"v5",10x,"vol")
       write(53,213) v1, v2, v3, v4, v5, psvol
 213   format(e12.4,1x,e12.4,1x,e12.4,1x,e12.4,1x,e12.4,1x,e12.4)
c
        write(53,222) top,bottom,zvol,psvol,pd
c
 222    format("The top of the zone = ",e12.4,/,
     :   "The bottom of the zone = ",e12.4,/,
     :   "Volume of zone = ",e12.4,/,"Particle Volume= ",e12.4,/,
     :   "Packing Fraction= ",e12.4,/)
c
 1    CONTINUE
c
c      tvol = xlength * height * width
c      vol = (4.0D0/3.0D0)*Pi*ra*ra*ra
c      write(6,*)"Volume of 1 particle",vol," Radius=",ra
c
c      svol = np * vol
```

```
c       pd = svol/tvol
c       write(6,*)"tpsvol=",tpsvol," svol=",svol
C
c       write(53,333) tvol, svol, pd
 333    format("Total Volume=",e12.4,/,"Total Particle Volume="
     : ,e12.4,/,"Total Packing Fraction=",e12.4)
c
       RETURN
       END
C
C
=======================================================================
C                              End of Program
C
=======================================================================
```

Subroutine *update.f* Lines 33 – 35
The following lines of code are used for contact detection with the cylindrical boundary.

```
      if((j.ge.ind1cy).and.(j.le.ind2cy)) then
         rsum = rad(i)
      endif
```

Subroutine *update.f* Lines 50 – 66
The following lines of code are described in the modifications to *deltem.f*.

```
      if((j.ge.ind1cy).and.(j.le.ind2cy)) then
        ry = 0.0
c-------see forces.f for an explaination of the below statements
        contactangle = atan(z(i)/x(i))
        xcp = sign(rad(j)*cos(contactangle),x(i))
        zcp = sign(rad(j)*sin(contactangle),z(i))
          if(z(i).eq.0) then
             xcp = sign(rad(j),x(i))
             zcp = 0.0
          endif
          if(x(i).eq.0) then
             xcp = 0.0
             zcp = sign(rad(j),z(i))
          endif
        rx = xcp - x(i)
        rz = zcp - z(i)
      endif
```

Subroutine *update.f* Lines 84 – 87
The following lines of code are used for collision detection with the axial compression particle.

```
      elseif((j.eq.indcomp).and.(ncomp.eq.1)) then
        rx = 0.0
        rz = 0.0
      endif
```

APPENDIX B

MATLAB CODE FOR PISTON FORCE EXTRACTION

This appendix gives the MATLAB code for extracting the piston load as described in Section 2.7.2.

```
%---This code will solve for the average force
% a method of comparing long term averages with short term
% averages is used. Along with a limit method.
%
%   savg=short term average
%   lavg=long term average
%
%   input some parameters
total_particles=input('Input the total number of free particles:  ');
radius=input('Inmput the radius of the free particles:  ');
weight=total_particles*106840.71*radius^3;
%
%  import the raw data file, the path must be set in MATLAB so
%  that the data file is found by the program
%
%extract the data from the raw data
force=abs(data(:,1))/weight;
t=data(:,2);
inc=input('Input the interval between outputs:  ');
%
% dx is the number of points to include in the short term average
% dx must be a positive integer, but a double data type
range=input('Enter the number of sections to break the data into: ');
dx=(length(t)/range);
d=uint32(dx);
dx=double(d);
%
%--neglect starting force fluctuations
zero=input('At what integer time is the data useful:  ');
start=uint32(zero/inc);
start=double(start);
%
%Input the tolerance in which the value is good
tol=input('What percent difference is acceptable:  ');
if(tol > 1)%convert to decimal if incorrect input
    tol=tol/100;
end
%--stop looking once you are at the end of the data
stop=length(t);
%
%plot the normalized force as a function of time
figure(1)
plot(t,force,'o')
xlim([2.5 max(t)])
set(gca,'YGrid','on')
xlabel('Time (s)')
```

```
ylabel('Force Normalized by Total Weight')
%
%-----begin the process
g=1;
for i=start:dx:stop-dx
    halt=i+dx;
    lsum=0;
    for j=start:1:halt
        lsum=lsum+force(j);
    end
    lavg=lsum/(halt-start);
    ssum=0;
    for k=i:1:halt-1
        ssum=ssum+force(k);
    end
    savg=ssum/(halt-i);

    diff=abs(lavg-savg)/lavg;
%--if the percent difference is less than tol than save that value
    if(diff <= tol)
        force1(g)=savg;
        g=g+1;
    end
end
r=1;
%--if the values are the close they must be the value
for q=2:1:g-1
    if(abs(force1(q-1)-force1(q)) <= 0.1)
        avg_force(r)=(force1(q)+force1(q-1))/2;
        r=r+1;
    end
end
%output the results to the screen
if(r > 1)
    value=mean(avg_force)
    plus=max(avg_force)-mean(avg_force)
    minus=mean(avg_force)-min(avg_force)
else
    '****** No values found *******'
end
%what if I could find the curve fit and then find the limit of that
curve?
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%code to find the curve fit and then find the limit as time -> infinity
%
%the first 200 data points are deleted because I only want the curve
that
%is in the form A+[Bexp(-Ct)]
j=1;
start1=5/inc;
for i=start1:1:length(t)
    time(j)=t(i);
    force1(j)=force(i);
    j=j+1;
end
%
```

```
Starting=rand(1,2,3);
options=optimset('Display','iter','TolX',1E-4);
Estimates=fminsearch('fit_function',Starting,options,time,force1);
%
figure(2)
plot(time,force1,'o','MarkerSize',3)
hold on;
fplot('curve_function',[0
(max(time)+10)],[],[],'k',Estimates(1),Estimates(2),Estimates(3))
%
syms q  %allows the symbolic limit calculation below
limit_force=limit(Estimates(1)+(Estimates(2)*exp(-
Estimates(3)*q)),q,inf,'left');
limit_force=double(limit_force)


function G=fit_function(params,Input,Actual_Output)
A=params(1);
B=params(2);
C=params(3);
Fitted_Curve=(A)+(B*exp(-Input*C));
Error_Vector=Fitted_Curve - Actual_Output;
G=sum(Error_Vector.^2);


function R=curve_function(t,A,B,C)
R=A+(B*exp(-C*t));
```

APPENDIX C

MATLAB CODE FOR PARTICLE ROTATION

This appendix gives the MATLAB code for processing the rotations of the particles described in Section 2.7.4.

```
%this will get the data for the rolling test of wall touching particles
%node10 -> fmub=0.4,fmu=0.1,phi=13.33
%node11 -> fmub=0.8,fmu=0.1,phi=13.33
%node12 -> fmub=fmu=0.4,phi=13.33
%node13 -> fmub=fmu=0.8,phi=13.33
%node15 -> fmub=fmu=0.4,phi=7.5
%    15.0 - dt=1e-6 sec
%    15.1 - dt=5e-7 sec
%    15.2 - dt=7.5e-7 sec
%    15.3 - dt=5e-6 sec
%
r = .05;    %particle radius
R = .6667; %cylinder radius
R15 = .375;
%w_p = (.0001/.05); %w_p = pistion velocity / particle radius
w_p = sqrt(3)*(.0001/.05); %pistion velocity / particle radius 3-
directions
%
%the data files (modified zrolling) needs to be placed into the correct
%matlab working folder to obtain the angular velocity as a function of
%radial distance
node10_0data = dlmread('node10_0data.txt');
node10_1data = dlmread('node10_1data.txt');
node10_2data = dlmread('node10_2data.txt');
node10_3data = dlmread('node10_3data.txt');
node11_0data = dlmread('node11_0data.txt');
node11_1data = dlmread('node11_1data.txt');
node11_2data = dlmread('node11_2data.txt');
node11_3data = dlmread('node11_3data.txt');
node12_0data = dlmread('node12_0data.txt');
node12_1data = dlmread('node12_1data.txt');
node12_2data = dlmread('node12_2data.txt');
node12_3data = dlmread('node12_3data.txt');
node13_0data = dlmread('node13_0data.txt');
node13_1data = dlmread('node13_1data.txt');
node13_2data = dlmread('node13_2data.txt');
node13_3data = dlmread('node13_3data.txt');
node15_0data = dlmread('node15_0data.txt');
node15_1data = dlmread('node15_1data.txt');
node15_2data = dlmread('node15_2data.txt');
node15_3data = dlmread('node15_3data.txt');
%
%get the data into workable arrays
%normalize by respective parameters
r10_0 = node10_0data(:,2)/R;
w10_0 = node10_0data(:,3)/w_p;
```

```
r10_1 = node10_1data(:,2)/R;
w10_1 = node10_1data(:,3)/w_p;
r10_2 = node10_2data(:,2)/R;
w10_2 = node10_2data(:,3)/w_p;
r10_3 = node10_3data(:,2)/R;
w10_3 = node10_3data(:,3)/w_p;
r11_0 = node11_0data(:,2)/R;
w11_0 = node11_0data(:,3)/w_p;
r11_1 = node11_1data(:,2)/R;
w11_1 = node11_1data(:,3)/w_p;
r11_2 = node11_2data(:,2)/R;
w11_2 = node11_2data(:,3)/w_p;
r11_3 = node11_3data(:,2)/R;
w11_3 = node11_3data(:,3)/w_p;
r12_0 = node12_0data(:,2)/R;
w12_0 = node12_0data(:,3)/w_p;
r12_1 = node12_1data(:,2)/R;
w12_1 = node12_1data(:,3)/w_p;
r12_2 = node12_2data(:,2)/R;
w12_2 = node12_2data(:,3)/w_p;
r12_3 = node12_3data(:,2)/R;
w12_3 = node12_3data(:,3)/w_p;
r13_0 = node13_0data(:,2)/R;
w13_0 = node13_0data(:,3)/w_p;
r13_1 = node13_1data(:,2)/R;
w13_1 = node13_1data(:,3)/w_p;
r13_2 = node13_2data(:,2)/R;
w13_2 = node13_2data(:,3)/w_p;
r13_3 = node13_3data(:,2)/R;
w13_3 = node13_3data(:,3)/w_p;
r15_0 = node15_0data(:,2)/R15;
w15_0 = node15_0data(:,3)/w_p;
r15_1 = node15_1data(:,2)/R15;
w15_1 = node15_1data(:,3)/w_p;
r15_2 = node15_2data(:,2)/R15;
w15_2 = node15_2data(:,3)/w_p;
r15_3 = node15_3data(:,2)/R15;
w15_3 = node15_3data(:,3)/w_p;
%
%node 10 results
figure(1)
plot(r10_0,w10_0,'ro');
hold on;
plot(r10_1,w10_1,'kd');
hold on;
plot(r10_2,w10_2,'gs');
hold on;
plot(r10_3,w10_3,'b+');
ylim([0 20]);
title('Results from node 10');
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
hold off;
%
%node 11 results
figure(2)
plot(r11_0,w11_0,'ro');
```

```
hold on;
plot(r11_1,w11_1,'kd');
hold on;
plot(r11_2,w11_2,'gs');
hold on;
plot(r11_3,w11_3,'b+');
ylim([0 20]);
title('Results from node 11');
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
hold off;
%
%node 12 results
figure(3)
plot(r12_0,w12_0,'ro');
hold on;
plot(r12_1,w12_1,'kd');
hold on;
plot(r12_2,w12_2,'gs');
hold on;
plot(r12_3,w12_3,'b+');
ylim([0 20]);
title('Results from node 12');
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
hold off;
%
%node 13 results
figure(4)
plot(r13_0,w13_0,'ro');
hold on;
plot(r13_1,w13_1,'kd');
hold on;
plot(r13_2,w13_2,'gs');
hold on;
plot(r13_3,w13_3,'b+');
ylim([0 20]);
title('Results from node 13');
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
hold off;
%
%node 15 results
figure(5)
plot(r15_0,w15_0,'ro');
hold on;
plot(r15_1,w15_1,'kd');
hold on;
plot(r15_2,w15_2,'gs');
hold on;
plot(r15_3,w15_3,'b+');
ylim([0 20]);
title('Results from node 15');
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
hold off;
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
%The code form here down will take an average over a band at different
r/R
%
n_bands = 20; %the number of bands to average over
band_width = ((R-r)/R)/n_bands; %length of the band over r/R the first
                                %term takes into account wall effect
n_bands15 = 15;
band_width15 = ((R15-r)/R15)/n_bands15;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% for the first data set 10.0
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w10_0t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position10_0(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r10_0)
        if(r10_0(i)>=start && r10_0(i)<finish && w10_0(i)<15) %into the
avg
            w10_0t(j) = w10_0t(j) + w10_0(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w10_0avg(k) = w10_0t(k)/counter(k);
end
%
%
%again for the next data set 10.1
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w10_1t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position10_1(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r10_1)
        if(r10_1(i)>=start && r10_1(i)<finish && w10_1(i)<15) %into the
avg
            w10_1t(j) = w10_1t(j) + w10_1(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
```

```
    w10_1avg(k) = w10_1t(k)/counter(k);
end
%
%
%again for the next data set 10.2
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w10_2t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position10_2(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r10_2)
        if(r10_2(i)>=start && r10_2(i)<finish && w10_2(i)<15) %into the
avg
            w10_2t(j) = w10_2t(j) + w10_2(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w10_2avg(k) = w10_2t(k)/counter(k);
end
%
%
%again for the next data set 10.3
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w10_3t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position10_3(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r10_3)
        if(r10_3(i)>=start && r10_3(i)<finish && w10_3(i)<15) %into the
avg
            w10_3t(j) = w10_3t(j) + w10_3(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w10_3avg(k) = w10_3t(k)/counter(k);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% for the first data set 11.0
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
```

```
    w11_0t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position11_0(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r11_0)
        if(r11_0(i)>=start && r11_0(i)<finish && w11_0(i)<15) %into the
avg
            w11_0t(j) = w11_0t(j) + w11_0(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w11_0avg(k) = w11_0t(k)/counter(k);
end
%
%
%again for the next data set 11.1
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w11_1t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position11_1(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r11_1)
        if(r11_1(i)>=start && r11_1(i)<finish && w11_1(i)<15) %into the
avg
            w11_1t(j) = w11_1t(j) + w11_1(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w11_1avg(k) = w11_1t(k)/counter(k);
end
%
%
%again for the next data set 11.2
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w11_2t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position11_2(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r11_2)
        if(r11_2(i)>=start && r11_2(i)<finish && w11_2(i)<15) %into the
avg
            w11_2t(j) = w11_2t(j) + w11_2(i);
```

```
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w11_2avg(k) = w11_2t(k)/counter(k);
end
%
%
%again for the next data set 11.3
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w11_3t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position11_3(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r11_3)
        if(r11_3(i)>=start && r11_3(i)<finish && w11_3(i)<15) %into the
avg
            w11_3t(j) = w11_3t(j) + w11_3(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w11_3avg(k) = w11_3t(k)/counter(k);
end
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% for the first data set 12.0
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w12_0t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position12_0(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r12_0)
        if(r12_0(i)>=start && r12_0(i)<finish && w12_0(i)<15) %into the
avg
            w12_0t(j) = w12_0t(j) + w12_0(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w12_0avg(k) = w12_0t(k)/counter(k);
```

```
end
%
%
%again for the next data set 12.1
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w12_1t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position12_1(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r12_1)
        if(r12_1(i)>=start && r12_1(i)<finish && w12_1(i)<15) %into the
avg
            w12_1t(j) = w12_1t(j) + w12_1(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w12_1avg(k) = w12_1t(k)/counter(k);
end
%
%
%again for the next data set 12.2
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w12_2t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position12_2(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r12_2)
        if(r12_2(i)>=start && r12_2(i)<finish && w12_2(i)<15) %into the
avg
            w12_2t(j) = w12_2t(j) + w12_2(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w12_2avg(k) = w12_2t(k)/counter(k);
end
%
%
%again for the next data set 12.3
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w12_3t(j) = 0;
```

```
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position12_3(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r12_3)
        if(r12_3(i)>=start && r12_3(i)<finish && w12_3(i)<15) %into the
avg
            w12_3t(j) = w12_3t(j) + w12_3(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w12_3avg(k) = w12_3t(k)/counter(k);
end
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% for the first data set 13.0
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w13_0t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position13_0(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r13_0)
        if(r13_0(i)>=start && r13_0(i)<finish && w13_0(i)<15) %into the
avg
            w13_0t(j) = w13_0t(j) + w13_0(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w13_0avg(k) = w13_0t(k)/counter(k);
end
%
%
%again for the next data set 13.1
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w13_1t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position13_1(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r13_1)
        if(r13_1(i)>=start && r13_1(i)<finish && w13_1(i)<15) %into the
avg
```

```
            w13_1t(j) = w13_1t(j) + w13_1(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w13_1avg(k) = w13_1t(k)/counter(k);
end
%
%
%again for the next data set 13.2
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w13_2t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position13_2(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r13_2)
        if(r13_2(i)>=start && r13_2(i)<finish && w13_2(i)<15) %into the
avg
            w13_2t(j) = w13_2t(j) + w13_2(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w13_2avg(k) = w13_2t(k)/counter(k);
end
%
%
%again for the next data set 13.3
counter = 0;
start = 0.0;
%
for j=1:1:n_bands
    w13_3t(j) = 0;
    counter(j) = 0;
    finish = j*band_width; %set the finish location
    position13_3(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r13_3)
        if(r13_3(i)>=start && r13_3(i)<finish && w13_3(i)<15) %into the
avg
            w13_3t(j) = w13_3t(j) + w13_3(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands %get the average value for each band
    w13_3avg(k) = w13_3t(k)/counter(k);
end
```

```
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% for the first data set 15.0
counter = 0;
start = 0.0;
%
for j=1:1:n_bands15
    w15_0t(j) = 0;
    counter(j) = 0;
    finish = j*band_width15; %set the finish location
    position15_0(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r15_0)
        if(r15_0(i)>=start && r15_0(i)<finish) %element goes into the
avg
            w15_0t(j) = w15_0t(j) + w15_0(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands15 %get the average value for each band
    w15_0avg(k) = w15_0t(k)/counter(k);
end
%
%
%again for the next data set 15.1
counter = 0;
start = 0.0;
%
for j=1:1:n_bands15
    w15_1t(j) = 0;
    counter(j) = 0;
    finish = j*band_width15; %set the finish location
    position15_1(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r15_1)
        if(r15_1(i)>=start && r15_1(i)<finish) %element goes into the
avg
            w15_1t(j) = w15_1t(j) + w15_1(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands15 %get the average value for each band
    w15_1avg(k) = w15_1t(k)/counter(k);
end
%
%
%again for the next data set 15.2
counter = 0;
start = 0.0;
%
for j=1:1:n_bands15
```

```
    w15_2t(j) = 0;
    counter(j) = 0;
    finish = j*band_width15; %set the finish location
    position15_2(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r15_2)
        if(r15_2(i)>=start && r15_2(i)<finish) %element goes into the
avg
            w15_2t(j) = w15_2t(j) + w15_2(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands15 %get the average value for each band
    w15_2avg(k) = w15_2t(k)/counter(k);
end
%
%
%again for the next data set 15.3
counter = 0;
start = 0.0;
%
for j=1:1:n_bands15
    w15_3t(j) = 0;
    counter(j) = 0;
    finish = j*band_width15; %set the finish location
    position15_3(j) = (finish+start)/2; %locate the center of averaged
area
    for i=1:1:length(r15_3)
        if(r15_3(i)>=start && r15_3(i)<finish) %element goes into the
avg
            w15_3t(j) = w15_3t(j) + w15_3(i);
            counter(j) = counter(j) + 1;
        end
    end
    start = finish; %reset the starting location
end
for k=1:1:n_bands15 %get the average value for each band
    w15_3avg(k) = w15_3t(k)/counter(k);
end
%
%%%%%%%%%%%% plots of results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%node10 -> fmub=0.4,fmu=0.1,phi=13.33
%node11 -> fmub=0.8,fmu=0.1,phi=13.33
%node12 -> fmub=fmu=0.4,phi=13.33
%node13 -> fmub=fmu=0.8,phi=13.33
%
figure(6)
plot(position10_0,w10_0avg,'ro')
hold on;
plot(position10_1,w10_1avg,'kd')
hold on;
plot(position10_2,w10_2avg,'gs')
hold on;
```

```
plot(position10_3,w10_3avg,'b+')
xlim([0 1]);
ylim([0 .5]);
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
legend('H/D=1.499','H/D=1.687','H/D=1.875','H/D=2.062'...
    ,'Location','northwest')
hold off;
%
figure(7)
plot(position11_0,w11_0avg,'ro')
hold on;
plot(position11_1,w11_1avg,'kd')
hold on;
plot(position11_2,w11_2avg,'gs')
hold on;
plot(position11_3,w11_3avg,'b+')
xlim([0 1]);
ylim([0 .5]);
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
legend('H/D=1.499','H/D=1.687','H/D=1.875','H/D=2.062'...
    ,'Location','northwest')
hold off;
%
figure(8)
plot(position12_0,w12_0avg,'ro')
hold on;
plot(position12_1,w12_1avg,'kd')
hold on;
%plot(position12_2,w12_2avg,'gs')
%hold on;
plot(position12_3,w12_3avg,'b+')
xlim([0 1]);
ylim([0 .5]);
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
legend('H/D=1.499','H/D=1.687','H/D=2.062'...
    ,'Location','northwest')
hold off;
%
figure(9)
plot(position13_0,w13_0avg,'ro')
hold on;
plot(position13_1,w13_1avg,'kd')
hold on;
plot(position13_2,w13_2avg,'gs')
hold on;
plot(position13_3,w13_3avg,'b+')
xlim([0 1]);
ylim([0 .5]);
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
legend('H/D=1.499','H/D=1.687','H/D=1.875','H/D=2.062'...
    ,'Location','northwest')
hold off;
%
```

```
figure(10)
plot(position15_3,w15_3avg,'b+')
hold on;
plot(position15_0,w15_0avg,'ro')
hold on;
plot(position15_1,w15_1avg,'kd')
hold on;
%plot(position15_2,w15_2avg,'gs')
%hold on;
fplot(inline('1'),[0,1],'b:')
xlim([0 1]);
xlabel('|r/R|');
ylabel('|\omega / \omega_p|');
legend('\Deltat=5E-6','\Deltat=1E-6','\Deltat=5E-7'...
    ,'Location','southeast')
hold off;
```

APPENDIX D

SAMPLE INPUT FILE

This appendix gives a sample input file.

```
$var  np =        3203   $Total number of particles in cell
 $var  bdry =      1      $flag for boundry type (1;cubic, 2;tringular)
 $var  nxby0 =     1      $No of boundary particles in x-dir. at y = 0
 $var  nzby0 =     1      $No of boundary particles in y-dir. at y = 0
 $var  nxby1 =     0      $No of boundary particles in x-dir. at ycell
 $var  nzby1 =     0      $No of boundary particles in x-dir. at ycell
 $var  nxbz0 =     0      $
 $var  nybz0 =     0      $
 $var  nxbz1 =     0      $
 $var  nybz1 =     0      $
 $var  nybx0 =     0      $No of boundary particles in y-dir. at x = 0
 $var  nzbx0 =     0      $No of boundary particles in z-dir. at x = 0
 $var  nybx1 =     0      $No of boundary particles in y-dir. at xcell
 $var  nzbx1 =     0      $No of boundary particles in z-dir. at xcell
 $var  nfix =      0     $ number of fixed particles
 $var  nzcyl =     0      $ number of fixed cylinders parallel to z-axis
 $var  nycyl =     1      $ number of fixed cylinders parallel to y-axis
 $var  ncomp =     1       $ the particle used for axial compression
 $var  ncmax =     0       $number of collisions during entire run
 $var  nout =      0     $No. of time to print out results
 $var  nczero =    0       $number of collisions before start cum. ave.
 $var  ntcol =    40       $number of time steps during a collision
 $var  nvel =     20       $number of intervals for vel. distrib.
 $var  nyzone =    5        $number of y zones
 $var  mzcell =    4        $number of z cells
 $var  nycell =    1       $numner of y cells
 $var  itervm =    1       $max iterations per time step
 $var  icoord =    0       $flag for coordinates print out
 $var  itty =      0     $flag for tty interaction
 $var  ixyz =      0      $flag to read init coords of fxd & bnd particles
 $var  istart =    0     $to restart the code rename d3ds to d3ds1000 and set istart=1000
 $var  tmax =  30.00         $max time for calculation
 $var  tpour = 0.5          $time for pouring
 $var  dt =    0.          $time step
 $var  dtout =  0.5         $time interval for printing out results
 $var  dtdump =  30.0         $time interval for dumping
 $var  tzero =   10.0       $restart long-term cum. ave.
 $var  search =  0.05       $search distance for near neighbors
```

```
$var ycell =  3.00       $cell height (m)
$var xyrat =  100.0      $ratio used to compute xcell
$var zyrat =  100.0      $ratio used to compute zcell
$var vave =   0.0        $average deviatoric transl. velocity
$var vseed =  0.9        $seed for random initial particle velocities
$var vxzero = 0.0        $initial velocity in the x-direction (ave)
$var vyzero = 0.0        $initial velocity in y-direction (ave)
$var vzzero = 0.0        $loading stiffness K1
$var skn1 =   2.8e+06    $normal force coefficient
$var elast =  0.971      $coefficient of restitution
$var slope =  0.         $alternative parameter for unloading
$var ratk =   0.8        $ratio of tangential/normal stiffness
$var fmu =    0.1        $coefficient of friction
$var fmub =   0.00       $friction for boundary and fixed particles
$var power =  0.3333333  $tangential force exponent
$var rmassz = 10891      $mass of unit sphere
$var tstart=  0.1        $time to initialize long term averages
$var gravx =  0.0        $acceleration of gravity in x direction
$var gravy =  -9.81      $acceleration of gravity in y direction
$var gravz =  0.0        $acceleration of gravity in z direction
$var vxby0 =  0.0        $x velocity of real boundary at y = zero
$var vxby1 =  0.0        $x velocity of real boundary at y = ycell
$var vyby0 =  0.0        $y velocity of real boundary at y = zero
$var vyby1 =  0.0        $y velocity of real boundary at y = ycell
$var t2move = 70.0       $time to start moving the floor/wall
$var t2stop = 2000.0     $time to stop the floor from moving
$var vyfloor = 0.0001    $velocity of the floor/wall when moving
$var draddt =  20.       $rate of increase of particle radii
$var compforce = 0.00001 $axial force to be applied (N)
$var number(1) = 3200    $number of particles in group 1
$var radius(1) = 0.05    $particle radii for group 1
$var number(2) = 1       $number of particles in group
$var radius(2) = 0.05    $radius of particles in group
$var number(3) = 1       $number of particles in group
$var radius(3) = -0.6667 $radius of cylindrical boundry
$var number(4) = 1       $number of particles in group
$var radius(4) = 0.05    $radius of large particle
$var skn1b = 2.8e+06     $
$var elastb= 0.83        $
$var slopeb= 0.          $
$var vamp = 0.           $velocity amplitude of boundary
$var frq = 0             $boundary frequency
$var finis =  1.         $end
```

APPENDIX E

VOLUME.F SUBROUTINE

This appendix gives the FORTRAN code to compute the volume of a sphere inside of a control volume. The control volume has a planar bottom and top, specified by the y-zone. And the inner and outer radii are specified by the radial zone.

```
c subroutine volume.f that will calculate the volume of a sphere inside
c an annular layer created 6/30/06 by s chester
c
      subroutine volume
      include 's3dscmm'
      integer*4 iter
      real*8 deltar,deltay,area,ylocation,rady,rp,r,p
c
c
c     ref. this paper:
c        C.R.A. Abreu, A Monte Carlo simulation of the packing and
c        segregation of spheres in cylinders.  Brazilian Journal of
c        chemical engineering. volume 16 issue 4, 1999
c
c     ref. any calculus book for the volumes by cross section
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c
      iter = 500
c
c
      do 10 n=1,nrzone
      rstart = (n-1)*drzone
      rstop = n*drzone
c
      do 11 m=1,nyzone
      ybot = (m-1)*dyzone
      ytop = m*dyzone
c
      vanzone(n,m) = pi*((rstop*rstop)-(rstart*rstart))*dyzone
c
      deltar = (rstop - rstart)/iter
      deltay = (ytop - ybot)/iter
c
      do 1 i=ind1,ind2
c
c--------initialize values for particle i
         area = 0.0
         volume(i) = 0.0
         p = 0.0
         totv(i) = 4*third*pi*rad(i)*rad(i)*rad(i)
c
c--------compute the radial distance to the particle center
         rp = sqrt(x(i)*x(i) + z(i)*z(i))
```

```
c
c--------first check to see if particle i is in this zone
         if(((y(i).gt.ytop+rad(i)).and.(y(i).lt.ybot-rad(i))).and.
     1     ((rp.gt.rstop+rad(i)).and.(rp.lt.rstart+rad(i)))) goto 1
c--------second check if the particle is totally inside the zone
         if((((y(i).gt.ybot+rad(i)).and.(y(i).lt.ytop-rad(i))).and.
     1     ((rp.gt.rstart+rad(i)).and.(rp.lt.rstop-rad(i))))) then
            volume(i) = 4*third*pi*rad(i)*rad(i)*rad(i)
            goto 1
         endif
c
c   loop thru all the acive particles,
c    first 'integrate' along the radial direction to get an area at a
specific y
c   then 'integrate' along the axial direction to get volume
c
         do 2 j=1,iter
c
            ylocation = ybot + (j*deltay)
            volume(i) = volume(i) + area*deltay
            area = 0.0
c
c-----------compute the radius of the circle cut into the sphere at
this y
            rady=sqrt(rad(i)*rad(i)-(ylocation-y(i))*(ylocation-y(i)))
c
         do 3 k=1,iter
c
            r = rstart + (k*deltar)
c
c
c-----begin testing for what case of intersection this is
c
            if(r+rady.le.rp) then
c              type a
               theta1 = 0.0
               theta2 = 0.0
            elseif((r.gt.rady).and.(r-rady.ge.rp)) then
c              type b
               theta1 = 0.0
               theta2 = 0.0
            elseif((r.lt.rady).and.(rady-r.ge.rp)) then
c              type c
               theta1 = 0.0
               theta2 = 2*pi
            elseif((rp-rady.gt.r).and.(rp+rady.lt.r)) then
c              type d
               delta = ((r-rady)*(r-rady) - (x(i)+z(i))*(x(i)+z(i))*
     1                 ((x(i)+z(i))*(x(i)+z(i))-(r+rady)*(r+rady)))
               num1 = 2*z(i)*r - sqrt(delta)
               num2 = 2*z(i)*r + sqrt(delta)
               dem = y(i)*y(i) - rady*rady + (x(i)+r)*(x(i)+r)
               theta1 = 2*arctan(num1/dem)
               theta2 = 2*arctan(num2/dem)
               if(r.lt.rady) then
c                 type e,f
                  thetam = (theta1 + theta2)/2
```

```
                  delta2        =        sqrt((x(i)-r*cos(thetam))*(x(i)-
r*cos(thetam))
     1                      + (z(i)-r*sin(thetam))*(z(i)-r*sin(thetam)))
               endif
            endif
c
            theta = theta2 - theta1
            p = min(theta*r,(2*pi-theta)*r)
            if(delta2.lt.r) then
               p = theta*r
            elseif(delta2.gt.r) then
               p = (2*pi-theta)*r
            endif
c
         area = area + p*deltar
c
 3       continue
 2       continue
c
            vfrac(n,m,i) = volume(i) / totv(i)
c
 1       continue
 11      continue
 10      continue
c$$$
c$$$
c$$$    END OF SUBROUTINE
c$$$
c$$$
         return
         end
```

APPENDIX F

MATLAB CODE TO REDRAW FIGURES FROM LITERATURE

This appendix gives the MATLAB code to redraw figures from literature. The data that was taken from literature is given in each case and referenced.

```
%Redraw figures from literature
Starting=rand(2); %starting 'guess' = random number
starting=rand(1);
options=optimset('Display','iter','TolX',1E-12,'MaxIter',500); %set
options
%Walton experiments 2004
length = [10 20 30];
forcedown = [.798 1.025 0.908];
forceup = [11.5 90.5 825.8];
figure(1)
semilogy(length,forceup,'ko')
hold on;
semilogy(length,forcedown,'kd')
xlim([0 35])
ylim([0 1000])
xlabel('Slug Length (cm)')
ylabel('Base Piston Force (N)')

hold on;
fplot(inline('.16027*x'),[0 35],'k-.')
hold on;

esta =
fminsearch('find_janssen_walton_exp1',Starting,options,length,forceup);
estb =
fminsearch('find_janssen_walton_exp2',Starting,options,length,forcedown
);

fplot('curve_fit_walton_exp1',[0 35],[],[],'k-',esta(1),esta(2))
hold on;
fplot('curve_fit_walton_exp2',[0 35],[],[],'k:',estb(1),estb(2))
hold off;

legend('Piston Up','Piston Down','Material Weight','Janssen - Wall
Down',...
    'Janssen - Wall Up','Location','Northwest')


sz = [2.366469 2.366469 4.901404 11.2048 20.9418];
msr = [0.924403 0.827188 1.046684 2.289523 2.926684];
figure(2)
plot(sz,msr,'ko')
hold on;
fplot(inline('0.12*x + 0.63'),[0 30],'k')
```

```
text('Interpreter','latex','String',...
    '$$\mu\sigma_r = A\sigma_z+B,\ where\ B\neq0$$',...
       'Position',[10 1],'FontSize',12)

xlim([0 30])
ylim([0 3.5])
ylabel('\mu\sigma_r')
xlabel('\sigma_z')
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%
%Bertho Experiments 2003
ma = [10 20 30 33.33];
m = [12 31 70 200];
figure(3)
plot(m,ma,'ko')
hold on;
fplot(inline('x'),[0 210],'k')
xlim([0 210])
ylim([0 35])
ylabel('M_a_p_p (g)')
xlabel('M (g)')
est1 = fminsearch('find_janssen_bertho_exp',starting,options,m,ma);
fplot('curve_fit_bertho_exp',[0 210],[],[],'k:',est1)
label = ['M_\infty \approx ',num2str(int8(est1))];
legend('v = 0.2 mm/sec','Hydrostatic',label,'Location','Southeast')
text('Interpreter','latex','String',...
    '$$M_a_p_p=M_\infty]}$$',...
       'Position',[50 20],'FontSize',12)
text('Interpreter','latex','String',...
    '$$M_\infty=\rho{c}\lambda\pi{D}^{2}/4$$',...
       'Position',[50 15],'FontSize',12)
hold off;
%
%
%%%%% vanel 1999
experiments%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mfill = [10 20 30 40 50 55 60 75 100 150 200 300 380];
mapp = [9.5 18.5 25 32 37.5 39.5 42 45 48 51.5 54 53 52];
figure(4)
fplot(inline('x'),[0 210],'k:')
hold on;
plot(mfill,mapp,'ko')
hold on;
%plot the two parameter fit
fplot(inline('53*(1-exp(-x/53))'),[0 400],'k--')
hold on;
fplot(inline('x'),[0 13.4],'k')
hold on;
fplot(inline('13.4+(39.6*(1-exp(-(x-13.4)/39.6)))'),[13.4 400],'k')
%end the two parameter fit
label2 = ['M_\infty = 53g'];
label3 = ['M_0 = 13.4g, M_\infty^{C} = 36.9g'];
```

```
%label the single parameter fit
text('Interpreter','latex','String',...
    '$$M_a_p_p=M_\infty{[1-e^{-M/M_\infty}]}$$',...
       'Position',[40 10],'FontSize',12)
text('Interpreter','latex','String',...
    '$$M_\infty=\rho{c}\lambda\pi{D}^{2}/4$$',...
       'Position',[40 5],'FontSize',12)
%label the two parameter fit
text('Interpreter','latex','String',...
    '$$for M \leq M_0 , M_a_p_p=M$$',...
       'Position',[75 30],'FontSize',12)
text('Interpreter','latex','String',...
    '$$for M > M_0 , M_a_p_p=M_0+M_\infty^{C}\left[1-e^{\left(\frac{-M-
M_0}{M_\infty^{C}}\right)}\right]$$',...
       'Position',[75 27],'FontSize',12)
%
legend('Hydrostatic','\nu =
0.585',label2,label3,'Location','Southeast')
xlim([0 400])
ylim([0 60])
ylabel('M_a (g)')
xlabel('M (g)')
hold off;
```

APPENDIX G

MATLAB CODE TO PROCESS STATIC DATA

This appendix gives the MATLAB code to create the figures used in the static simulation cases.

```
%this MATLAB code will process the data for the static simulations
%node5 -> fmub=fmu=0.1,phi=13.33
%node6 -> fmub=fmu=0.4,phi=13.33
%node7 -> fmub=fmu=0.1,phi=26.66
%node8 -> fmub=fmu=0.4,phi=26.66
%node16-> fmub=fmu=0.1,phi=13.33
%
d = 2*.05;
D1 = 2*.6667;
D2 = 4*.6667;
density = 2600;
weight = 9.81*density*(pi/6)*d^3;
norm = weight/(d^2);
%
np_node5 = [3200 3600 4000 4400];
np_node6 = [3200 3600 4000 4400];
np_node7 = [6400 7200 8000 8800];
np_node8 = [6400 7200 8000 8800];
np_node16 = [1600 2000 2400 2800 3200 3600 4000 4400];
%
force_node5 = [.45234E5 .49620E5 .52952E5 .56792E5];
force_node6 = [.47108E5 .45991E5 .45481E5 .46991E5];
force_node7 = [.85493E5 .96465E5 .10681E6 .11745E6];
force_node8 = [.84474E5 .94974E5 .10694E6 .11733E6];
force_node16 = [.21448E5 .26860E5 .32370E5 .38094E5 ...
    .43077E5 .48670E5 .53096E5 .56791E5];
%
%some post processing to get H/D and F*
H_over_D_node5 = (np_node5*(d^3))/(.9*(D1^3));
H_over_D_node6 = (np_node6*(d^3))/(.9*(D1^3));
H_over_D_node7 = (np_node7*(d^3))/(.9*(D2^3));
H_over_D_node8 = (np_node8*(d^3))/(.9*(D2^3));
H_over_D_node16 = (np_node16*(d^3))/(.9*(D1^3));
%
fstar_node5 = force_node5/weight;
fstar_node6 = force_node6/weight;
fstar_node7 = force_node7/weight;
fstar_node8 = force_node8/weight;
fstar_node16 = force_node16/weight;
fstar_node17 = force_node16/weight;
%
%some processing on the axial stress profile
stress_node16a0 = [3.6233E6 2.8026E7 5.9989E7 9.3035E7 1.2514E8 ...
    1.5786E8 1.9059E8]/norm;
height_node16a0 = (1.25/(D1*8))*[.5 1.5 2.5 3.5 4.5 5.5 6.5];
stress_node16a1 = [2.2617E6 2.3470E7 5.3853E7 8.6407E7 1.1768E8 ...
```

```
    1.4998E8 1.7969E8 2.09442E8 2.4461E8]/norm;
height_node16a1 = (1.5/(D1*10))*[.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5];
stress_node16a2 = [2.1213E6 2.1982E7 5.1743E7 8.2437E7 1.1319E8 ...
    1.4313E8 1.7388E8 2.038E8 2.3656E8 2.6253E8 2.9667E8]/norm;
height_node16a2 = (1.75/(D1*12))*[.5 1.5 2.5 3.5 4.5 5.5 6.5 7.5 ...
    8.5 9.5 10.5];
stress_node16a3 = [3.1060E6 2.6246E7 5.7424E7 9.0554E7 1.2297E8 ...
    1.5565E+08 1.8678E+08 2.2022E+08 2.5232E+08 ...
    2.8531E+08 3.0985E+08 3.5316E+08]/norm;
height_node16a3 = (2.0/(D1*13))*[.5 1.5 2.5 3.5 4.5 5.5 ...
    6.5 7.5 8.5 9.5 10.5 11.5];
stress_node16b0 = [2.3649E+06 2.3618E+07 5.4863E+07 8.5812E+07 ...
    1.1851E+08 1.5007E+08 1.8113E+08 2.1235E+08 2.4445E+08 ...
    2.7555E+08 3.0802E+08 3.3949E+08 3.6523E+08 4.0019E+08]/norm;
height_node16b0 = (2.25/(D1*15))*[.5 1.5 2.5 3.5 4.5 5.5 ...
    6.5 7.5 8.5 9.5 10.5 11.5 12.5 13.5];
stress_node16b1 = [1.8162E+06 2.1233E+07 5.1785E+07 8.2267E+07 ...
    1.1309E+08 1.4551E+08 1.7564E+08 2.0730E+08 2.3582E+08 2.7027E+08
...
    3.0122E+08 3.3455E+08 3.6339E+08 3.9458E+08 4.1964E+08 ...
    4.5324E+08]/norm;
height_node16b1 = (2.5/(D1*17))*[.5 1.5 2.5 3.5 4.5 5.5 ...
    6.5 7.5 8.5 9.5 10.5 11.5 12.5 13.5 14.5 15.5];
stress_node16b2 = [2.3049E+06 2.4014E+07 5.5409E+07 8.7534E+07 ...
    1.2036E+08 1.5172E+08 1.8311E+08 2.1645E+08 2.4904E+08 2.8123E+08
...
    3.1544E+08 3.4720E+08 3.7660E+08 4.0996E+08 4.3234E+08 4.6365E+08
...
    4.9740E+08]/norm;
height_node16b2 = (2.75/(D1*18))*[.5 1.5 2.5 3.5 4.5 5.5 ...
    6.5 7.5 8.5 9.5 10.5 11.5 12.5 13.5 14.5 15.5 16.5];
stress_node16b3 = [2.2863E+06 2.3328E+07 5.4311E+07 8.6168E+07 ...
    1.1783E+08 1.4974E+08 1.8213E+08 2.1439E+08 2.4505E+08 2.7965E+08
...
    3.0949E+08 3.4036E+08 3.6620E+08 3.9649E+08 4.2205E+08 4.5234E+08
...
    4.7604E+08 4.9814E+08 5.3229E+08]/norm;
height_node16b3 = (3.0/(D1*20))*[.5 1.5 2.5 3.5 4.5 5.5 ...
    6.5 7.5 8.5 9.5 10.5 11.5 12.5 13.5 14.5 15.5 16.5 17.5 18.5];
%
labela0 = ['H/D=',num2str(H_over_D_node16(1))];
labela1 = ['H/D=',num2str(H_over_D_node16(2))];
labela2 = ['H/D=',num2str(H_over_D_node16(3))];
labela3 = ['H/D=',num2str(H_over_D_node16(4))];
labelb0 = ['H/D=',num2str(H_over_D_node16(5))];
labelb1 = ['H/D=',num2str(H_over_D_node16(6))];
labelb2 = ['H/D=',num2str(H_over_D_node16(7))];
labelb3 = ['H/D=',num2str(H_over_D_node16(8))];
%
%%%%%%%%%%%%%%%%%%%% Plots of Results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot the piston load results for phi=13.33
figure(1)
%plot(H_over_D_node5,fstar_node5,'r+');
%hold on;
%plot(H_over_D_node6,fstar_node6,'gd');
%hold on;
```

```
plot(H_over_D_node16,fstar_node16,'ko');
hold on;
fplot(inline('(36/40)*(13.33^3)*x'),[0 2.5],'k:')%plot weight=f(H/D1)
ylim([0 5E3]);
xlabel('\it{H/D}')
ylabel('F*')
legend('\mu_w=\mu_p=0.1','Hydrostatic','Location','northwest')
hold off;
%
%plot the piston load resutls for phi=26.66
figure(2)
plot(H_over_D_node7,fstar_node7,'bo');
hold on;
plot(H_over_D_node8,fstar_node8,'kd');
hold on;
fplot(inline('(36/40)*(26.66^3)*x'),[0 2.5],'k:')%plot weight=f(H/D2)
ylim([0 10E3]);
xlim([0 1.0]);
xlabel('\it{H/D}')
ylabel('F*')
legend('\mu_w=\mu_p=0.1','\mu_w=\mu_p=0.4','Hydrostatic'...
    ,'Location','northeast')
hold off;
%
%plot stress results for all of node 16
figure(3)
plot(height_node16a0,stress_node16a0,'ko')
hold on;
plot(height_node16a1,stress_node16a1,'r+')
hold on;
plot(height_node16a2,stress_node16a2,'bs')
hold on;
plot(height_node16a3,stress_node16a3,'gd')
hold on;
plot(height_node16b0,stress_node16b0,'m*')
hold on;
plot(height_node16b1,stress_node16b1,'yp')
hold on;
plot(height_node16b2,stress_node16b2,'ch')
hold on;
plot(height_node16b3,stress_node16b3,'rx')
hold on;
fplot(inline('0.6*2600*9.81*x'),[0 3],'k:')
legend(labela0,labela1,labela2,labela3,labelb0,labelb1,labelb2,labelb3,
...
    'Location','northwest')
%ylim([0 6E5]);
%xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
hold off;
%
%plot the results for node 16a.0
figure(4)
plot(height_node16a0,stress_node16a0,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
```

```
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labela0,'Location','northwest')
hold off;
%
%plot the results for node 16a.1
figure(5)
plot(height_node16a1,stress_node16a1,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labela1,'Location','northwest')
hold off;
%
%plot the results for node 16a.2
figure(7)
plot(height_node16a2,stress_node16a2,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labela2,'Location','northwest')
hold off;
%
%plot the results for node 16a.3
figure(8)
plot(height_node16a3,stress_node16a3,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labela3,'Location','northwest')
hold off;
%
%plot the results for node 16b.0
figure(8)
plot(height_node16b0,stress_node16b0,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labelb0,'Location','northwest')
hold off;
%
%plot the results for node 16b.1
figure(9)
plot(height_node16b1,stress_node16b1,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labelb1,'Location','northwest')
hold off;
%
%plot the results for node 16b.2
figure(10)
```

```
plot(height_node16b2,stress_node16b2,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labelb2,'Location','northwest')
hold off;
%
%plot the results for node 16b.3
figure(11)
plot(height_node16b3,stress_node16b3,'ko-')
ylim([0 6E5]);
xlim([0 2.5]);
xlabel('\it{z/D}');
ylabel('\sigma_z_z^*');
%legend(labelb3,'Location','northwest')
hold off;
```

APPENDIX H

MATLAB CODE TO PROCESS DYNAMIC DATA

This appendix gives the MATLAB code to create the figures used in the dynamic simulation cases.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%Post processing code for the results of dynamic simulations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%node1 -> fmub=fmu=0.4,phi=13.33
%node2 -> fmub=fmu=0.8,phi=13.33
%node3 -> fmub=fmu=0.4,phi=26.66
%node4 -> fmub=0.1,fmu=1.0,phi=13.33
%
%and of course the old data (prior to Lemieux.psc.edu) which
%can be found in the WCPT5 paper
%
r = .05;
d = 2*r;
R1 = .6667;
D1 = 2*R1;
R2 = 2*R1;
D2 = 2*R2;
weight = (pi/6)*2600*9.81*d^3;
%
%%%%%%%%%%%%%%% data from lemieux.psc.edu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fhat1 = [.4952 .4716 .4285 .4080];
fhat2 = [.3997 .3768 .3490 .3394];
fhat3 = [.8067 .7841 .7809 .7477];
fhat4 = [.7931 .7551 .7399 .7389];
%
np1 = [3200 3600 4000 4400];
np2 = [3200 3600 4000 4400];
np3 = [6400 7200 8000 8800];
np4 = [3200 3600 4000 4400];
%
weight1 = np1*weight;
weight2 = np2*weight;
weight3 = np3*weight;
weight4 = np4*weight;
%
H_by_D_1 = .00046868*np1;
H_by_D_2 = .00046868*np2;
H_by_D_3 = ((d^3)/(.9*D2^3))*np3;
H_by_D_4 = .00046868*np4;
%
F1=fhat1.*weight1;
F2=fhat2.*weight2;
```

```
F3=fhat3.*weight3;
F4=fhat4.*weight4;
%
Fstar1 = F1./weight;
Fstar2 = F2./weight;
Fstar3 = F3./weight;
Fstar4 = F4./weight;
%%%%%%%%%%%%%%%%%%%%%%   end of lemieux.psc.edu data
%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%%%  start of old data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
np12old = [400 800 1600 2400 3200 4000 5000 6000];
np4old = [400 800 1600 2400 3200 4000 5000 6000];
np8old = [800 2400 3200];
%np8wallold = [1600 2400 3200];%for 3 data points
np8wallold = [1600 2400 3200 4000];%for 4 data points
%
weight12old = weight*np12old;
weight4old = weight*np4old;
weight8old = weight*np8old;
weight8wallold = weight*np8wallold;
%
H_by_D_12old = np12old*((d^3)/(0.9*D1^3));
H_by_D_4old = np4old*((d^3)/(0.9*D1^3));
H_by_D_8old = np8old*((d^3)/(0.9*D1^3));
H_by_D_8wallold = np8wallold*((d^3)/(0.9*D1^3));
%
F_12old = [3996.39 7626.45 13962.7 19524.4 24228.3 27839.1 30900.5
35683.8];
F_4old = [.9379 .8967 .79 .6959 .6514 0.5858 0.51 0.4429].*weight4old;
F_8old = [0.863 0.685 0.598].*weight8old;
%F_8wallold = [0.765 0.679 0.61].*weight8wallold;%for 3 data points
F_8wallold = [0.765 0.679 0.61 0.569].*weight8wallold;%for 4 data
points
%
Fstar12old = F_12old/weight;
Fstar4old = F_4old/weight;
Fstar8old = F_8old/weight;
Fstar8wallold = F_8wallold/weight;
%
%%%%%%%%%%%%%%%%%%%%%%%%   end of old data
%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%%% start of "floor-down" data
%%%%%%%%%%%%%%%%%%%%%%
npfd2thru9 = [1600 2400 3200 4000 4800 5600 6400 7200];
npfd10thru13 = [1600 2400 3200 4000];
npfd14thru17 = [1600 2400 3200 4000];
%
weight_fd2thru9 = weight*npfd2thru9;
weight_fd10thru13 = weight*npfd10thru13;
weight_fd14thru17 = weight*npfd14thru17;
%
F_fd2thru9 = [1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0].*weight_fd2thru9;
F_fd10thru13 = [.745 .672 .608 .533].*weight_fd10thru13;
```

```
F_fd14thru17 = [.959 .944 .921 .931].*weight_fd14thru17;
%
HbyD_fd2thru9 = npfd2thru9*((d^3)/(0.9*D1^3));
HbyD_fd10thru13 = npfd10thru13*((d^3)/(0.9*D1^3));
HbyD_fd14thru17 = npfd14thru17*((d^3)/(0.9*D2^3));
%
Fstar_fd2thru9 = F_fd2thru9/weight;
Fstar_fd10thru13 = F_fd10thru13/weight;
Fstar_fd14thru17 = F_fd14thru17/weight;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%the next area of code finds the least squares fit of the data to the
%Janssen model.  Estimates is the value of the fitting parameter.
%
Starting=rand(1); %starting 'guess' = random number
options=optimset('Display','iter','TolX',1E-12,'MaxIter',500); %set
options
%
%find fitting parameter for lemieux data
Estimates1 =
fminsearch('find_janssen_13',Starting,options,H_by_D_1,Fstar1);
Estimates2 =
fminsearch('find_janssen_13',Starting,options,H_by_D_2,Fstar2);
Estimates3 =
fminsearch('find_janssen_26',Starting,options,H_by_D_3,Fstar3);
Estimates4 =
fminsearch('find_janssen_13',Starting,options,H_by_D_4,Fstar4);
%find fitting parameter for old data
Estimates12old = fminsearch('find_janssen_13',Starting,options,...
    H_by_D_12old,Fstar12old);
Estimates4old = fminsearch('find_janssen_13',Starting,options,...
    H_by_D_4old,Fstar4old);
Estimates8old = fminsearch('find_janssen_13',Starting,options,...
    H_by_D_8old,Fstar8old);
Estimates8wallold = fminsearch('find_janssen_13',Starting,options,...
    H_by_D_8wallold,Fstar8wallold);
%find fitting parameter for 'floor-down' data
Estimates_fd2thru9 = fminsearch('find_janssen_13',Starting,options,...
    HbyD_fd2thru9,Fstar_fd2thru9);
Estimates_fd10thru13 =
fminsearch('find_janssen_13',Starting,options,...
    HbyD_fd10thru13,Fstar_fd10thru13);
Estimates_fd14thru17 =
fminsearch('find_janssen_26',Starting,options,...
    HbyD_fd14thru17,Fstar_fd14thru17);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
%The next section of code plots the results.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
figure(1) %plot of all the data
plot(H_by_D_1,Fstar1,'ro')
hold on;
plot(H_by_D_2,Fstar2,'bd')
```

```
hold on;
plot(H_by_D_3,Fstar3,'gs')
hold on;
plot(H_by_D_4,Fstar4,'m+')
hold on;
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k.-') %plot the
weight=f(H/D1)
hold on;
fplot(inline('x*(36/40)*(26.66)^3'),[0 10],'ko-'); %plot the
weight=f(H/D2)
hold on;
fplot('curve_fit_13',[0 10],[],[],'r',Estimates1); %plot a curve fit
hold on;
fplot('curve_fit_13',[0 10],[],[],'b-.',Estimates2); %plot a curve fit
hold on;
fplot('curve_fit_26',[0 10],[],[],'g:',Estimates3); %plot a curve fit
hold on;
fplot('curve_fit_13',[0 10],[],[],'m--',Estimates4); %plot a curve fit
%
%   these are plots of the old data
plot(H_by_D_12old,Fstar12old,'bd')
hold on;
fplot('curve_fit_13',[0 10],[],[],'b-',Estimates12old)
hold on;
plot(H_by_D_4old,Fstar4old,'bs')
hold on;
fplot('curve_fit_13',[0 10],[],[],'b-',Estimates4old)
hold on;
plot(H_by_D_8old,Fstar8old,'bo')
hold on;
fplot('curve_fit_13',[0 10],[],[],'b-',Estimates8old)
hold on;
plot(H_by_D_8wallold,Fstar8wallold,'b+')
hold on;
fplot('curve_fit_13',[0 10],[],[],'b-',Estimates8wallold)
hold on;
%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$
% bertho experiments at phi = 15
exph = [.3694 .963 2.22 3.1 6.2];
expf = [943.3 1886.5 2829.8 2994.8 3143.9];
est1 = fminsearch('find_janssen_15',Starting,options,exph,expf);
plot(exph,expf,'g*')
hold on;
fplot('curve_fit_15',[0 10],[],[],'g-',est1)
hold on;
%  walton experiments at phi = 13.33
exphh = [2.5 5 7.5];
expff = [2565 2850 2850];
est2 = fminsearch('find_janssen_13',Starting,options,exphh,expff);
plot(exphh,expff,'g*')
hold on;
fplot('curve_fit_13',[0 10],[],[],'g-',est2)
hold on;
%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$
%
```

```
ylim([0 1E4])  %set the y-axis limits
xlim([0 3])
ylabel('F*')
xlabel('H/D')
label1 = ['\lambda=',num2str(Estimates1)];
label2 = ['\lambda=',num2str(Estimates2)];
label3 = ['\lambda=',num2str(Estimates3)];
label4 = ['\lambda=',num2str(Estimates4)];
label12old = ['\lambda=',num2str(Estimates12old)];
label4old = ['\lambda=',num2str(Estimates4old)];
label8old = ['\lambda=',num2str(Estimates8old)];
label8wallold = ['\lambda=',num2str(Estimates8wallold)];
label_fd2thru9 = ['\lambda=',num2str(Estimates_fd2thru9)];
label_fd10thru13 = ['\lambda=',num2str(Estimates_fd10thru13)];
label_fd14thru17 = ['\lambda=',num2str(Estimates_fd14thru17)];
%
legend('node1 \phi=13.33','node2 \phi=13.33','node3 \phi=26.66'...
    ,'node4 \phi=13.33','weight \phi=13.33','weight \phi=26.66'...
    ,label1,label2,label3,label4)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[.5 5000],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[.5 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[.5 5000],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[.5 4000],'FontSize',12)
hold off;
%
%
%
figure(2) %plot of node 1 data
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_1(1),Fstar1(1),'k+')
hold on;
plot(H_by_D_1(2),Fstar1(2),'ks')
hold on;
plot(H_by_D_1(3),Fstar1(3),'kd')
hold on;
plot(H_by_D_1(4),Fstar1(4),'ko')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r',Estimates1);
ylim([0 3E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* = f(H/D)','D1','D2','D3','D4',label1,'Location','southeast')
text('Interpreter','latex','String',...
```

```
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[1 800],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[1.25 500],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 2500],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[1 800],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[1.25 500],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 2500],'FontSize',12)
hold off;
%
%
%
figure(3) %plot of node 2 data
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_2(1),Fstar2(1),'k+')
hold on;
plot(H_by_D_2(2),Fstar2(2),'ks')
hold on;
plot(H_by_D_2(3),Fstar2(3),'kd')
hold on;
plot(H_by_D_2(4),Fstar2(4),'ko')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r',Estimates2);
ylim([0 3E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* = f(H/D)','D5','D6','D7','D8',label2,'Location','southeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[1 800],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[1.25 500],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 2500],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[1 800],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
```

```
       'Position',[1.25 500],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 2500],'FontSize',12)
hold off;
%
%
%
figure(4) %plot of node 3 data
fplot(inline('x*(36/40)*(26.66)^3'),[0 10],'k-') %plot the
weight=f(H/D2)
hold on;
plot(H_by_D_3(1),Fstar3(1),'k+')
hold on;
plot(H_by_D_3(2),Fstar3(2),'ks')
hold on;
plot(H_by_D_3(3),Fstar3(3),'kd')
hold on;
plot(H_by_D_3(4),Fstar3(4),'ko')
hold on;
fplot('curve_fit_26',[0 10],[],[],'r',Estimates3);
ylim([0 20E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* =
f(H/D)','D9','D10','D11','D12',label3,'Location','southeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[2 6000],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[2.5 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=26.66}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 10000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[2 6000],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[2.5 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=26.66}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 10000],'FontSize',12)
hold off;
%
%
%
figure(5) %plot of node 4 data
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_4(1),Fstar4(1),'k+')
hold on;
```

```
plot(H_by_D_4(2),Fstar4(2),'ks')
hold on;
plot(H_by_D_4(3),Fstar4(3),'kd')
hold on;
plot(H_by_D_4(4),Fstar4(4),'ko')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r',Estimates4);
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* =
f(H/D)','D13','D14','D15','D16',label4,'Location','southeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[1.25 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[2 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 2500],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[1.25 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[2 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 2500],'FontSize',12)
hold off;
%
%
%
figure(6) %plot of fmub=0.12, fmu=0.1
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_12old(1),Fstar12old(1),'k+')
hold on;
plot(H_by_D_12old(2),Fstar12old(2),'ks')
hold on;
plot(H_by_D_12old(3),Fstar12old(3),'kd')
hold on;
plot(H_by_D_12old(4),Fstar12old(4),'ko')
hold on;
plot(H_by_D_12old(5),Fstar12old(5),'kx')
hold on;
plot(H_by_D_12old(6),Fstar12old(6),'k^')
hold on;
plot(H_by_D_12old(7),Fstar12old(7),'kv')
hold on;
plot(H_by_D_12old(8),Fstar12old(8),'k*')
hold on;
```

```
fplot('curve_fit_13',[0 10],[],[],'r-',Estimates12old)
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* = f(H/D)','D33','D34','D35','D36','D37','D38','D39',...
    'D40',label12old,'Location','northeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
hold off;
%
%
%
figure(7) %plot of fmub=0.4, fmu=0.1
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_4old(1),Fstar4old(1),'k+')
hold on;
plot(H_by_D_4old(2),Fstar4old(2),'ks')
hold on;
plot(H_by_D_4old(3),Fstar4old(3),'kd')
hold on;
plot(H_by_D_4old(4),Fstar4old(4),'ko')
hold on;
plot(H_by_D_4old(5),Fstar4old(5),'kx')
hold on;
plot(H_by_D_4old(6),Fstar4old(6),'k^')
hold on;
plot(H_by_D_4old(7),Fstar4old(7),'kv')
hold on;
plot(H_by_D_4old(8),Fstar4old(8),'k*')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r-',Estimates4old)
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* = f(H/D)','D41','D42','D43','D44','D45','D46','D47',...
```

```
    'D48',label4old,'Location','northeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
hold off;
%
%
%
figure(8) %plot of fmub=0.8, fmu=0.1
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_8old(1),Fstar8old(1),'k+')
hold on;
plot(H_by_D_8old(2),Fstar8old(2),'ks')
hold on;
plot(H_by_D_8old(3),Fstar8old(3),'kd')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r-',Estimates8old)
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* =
f(H/D)','D49','D50','D51',label8old,'Location','northeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
```

```
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
hold off;
%
%
%
figure(9) %plot of fmub=0.8, fmu=0.1, wall move
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_8wallold(1),Fstar8wallold(1),'k+')
hold on;
plot(H_by_D_8wallold(2),Fstar8wallold(2),'ks')
hold on;
plot(H_by_D_8wallold(3),Fstar8wallold(3),'kd')
hold on;
plot(H_by_D_8wallold(4),Fstar8wallold(4),'ko')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r-',Estimates8wallold)
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* =
f(H/D)','D52','D53','D54','D55',label8wallold,'Location','northeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
hold off;
%
%
%
%plot floor-down2 thru 9 data
figure(10)
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
```

```
plot(HbyD_fd2thru9(1),Fstar_fd2thru9(1),'k+')
hold on;
plot(HbyD_fd2thru9(2),Fstar_fd2thru9(2),'ks')
hold on;
plot(HbyD_fd2thru9(3),Fstar_fd2thru9(3),'kd')
hold on;
plot(HbyD_fd2thru9(4),Fstar_fd2thru9(4),'ko')
hold on;
plot(HbyD_fd2thru9(5),Fstar_fd2thru9(5),'kx')
hold on;
plot(HbyD_fd2thru9(6),Fstar_fd2thru9(6),'k^')
hold on;
plot(HbyD_fd2thru9(7),Fstar_fd2thru9(7),'kv')
hold on;
plot(HbyD_fd2thru9(8),Fstar_fd2thru9(8),'k*')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r-',Estimates_fd2thru9)
ylim([0 8E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* =
f(H/D)','D17','D18','D19','D20','D21','D22','D23','D24',...
    label_fd2thru9,'Location','northeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
hold off;
%
%
%
%plot floor-down10 thru 13 data
figure(11)
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(HbyD_fd10thru13(1),Fstar_fd10thru13(1),'k+')
hold on;
plot(HbyD_fd10thru13(2),Fstar_fd10thru13(2),'ks')
hold on;
```

```
plot(HbyD_fd10thru13(3),Fstar_fd10thru13(3),'kd')
hold on;
plot(HbyD_fd10thru13(4),Fstar_fd10thru13(4),'ko')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r-',Estimates_fd10thru13)
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* = f(H/D)','D25','D26','D27','D28',...
    label_fd10thru13,'Location','northeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
hold off;
%
%
%
%plot floor-down14 thru 17 data
figure(12)
fplot(inline('x*(36/40)*(26.66)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(HbyD_fd14thru17(1),Fstar_fd14thru17(1),'k+')
hold on;
plot(HbyD_fd14thru17(2),Fstar_fd14thru17(2),'ks')
hold on;
plot(HbyD_fd14thru17(3),Fstar_fd14thru17(3),'kd')
hold on;
plot(HbyD_fd14thru17(4),Fstar_fd14thru17(4),'ko')
hold on;
fplot('curve_fit_26',[0 10],[],[],'r-',Estimates_fd14thru17)
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
legend('W* = f(H/D)','D29','D30','D31','D32',...
    label_fd14thru17,'Location','northeast')
text('Interpreter','latex','String',...
```

```
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=26.66}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=26.66}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[3 4000],'FontSize',12)
hold off;
%
%
%
%comparision between moving the floor or the wall
figure(13)
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'k-') %plot the
weight=f(H/D1)
hold on;
plot(H_by_D_8wallold(1),Fstar8wallold(1),'k+')
hold on;
plot(H_by_D_8wallold(2),Fstar8wallold(2),'ks')
hold on;
plot(H_by_D_8wallold(3),Fstar8wallold(3),'kd')
hold on;
plot(H_by_D_8wallold(4),Fstar8wallold(4),'ko')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r-.',Estimates8wallold)
hold on;
plot(H_by_D_8old(1),Fstar8old(1),'k*')
hold on;
plot(H_by_D_8old(2),Fstar8old(2),'kp')
hold on;
plot(H_by_D_8old(3),Fstar8old(3),'kv')
hold on;
fplot('curve_fit_13',[0 10],[],[],'b:',Estimates8old)
ylim([0 7E3])  %set the y-axis limits
xlim([0 10])
ylabel('F*')
xlabel('H/D')
walllabel=[label8wallold,' (Wall Move)'];
pistonlabel=[label8old,' (Piston Move)'];
legend('W* =
f(H/D)','D52','D53','D54','D55',walllabel,'D49','D50','D51',pistonlabel
,'Location','northwest')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
```

```
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[4 2000],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[4 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
       'Position',[4.5 700],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
       'Position',[4 2000],'FontSize',12)
hold off;
%
%
%comparision between walton experiments and simulated data
figure(14)
%weight
fplot(inline('x*(36/40)*(13.33)^3'),[0 10],'m-') %plot the
weight=f(H/D1)
hold on;
%walton experiments
plot(exphh,expff,'ks')
hold on;
fplot('curve_fit_13',[0 10],[],[],'k-',est2)
hold on;
%simulated data set 1
plot(H_by_D_12old,Fstar12old,'r+')
hold on;
fplot('curve_fit_13',[0 10],[],[],'r--',Estimates12old)
hold on;
%simulated data set 2
plot(H_by_D_8old,Fstar8old,'bo')
hold on;
fplot('curve_fit_13',[0 10],[],[],'b:',Estimates8old)
hold on;
%simulated data set 3
plot(H_by_D_4old,Fstar4old,'gd')
hold on;
fplot('curve_fit_13',[0 10],[],[],'g-.',Estimates4old)
ylim([0 4E3])
xlim([0 10])
ylabel('F*')
xlabel('H/D')
explabel=['\lambda=',num2str(est2)];
legend('Hydrostatic','Walton Experiments',explabel,'D33 -
D40',label12old,...
    'D49 - D51',label8old,'D41 - D48',label4old,'Location','southeast')
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
       'Position',[1.25 1000],'FontSize',12)
```

```
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
      'Position',[2 600],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
      'Position',[2 1400],'FontSize',12)
text('Interpreter','latex','String',...
    '$$F^{*}=\frac{1.5\nu\phi^{3}}{\lambda}{[1-e^{-
\lambda{(H/D)}}]}$$',...
      'Position',[1.25 1000],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\lambda}={\beta}D=4{\mu}K$$',...
      'Position',[2 600],'FontSize',12)
text('Interpreter','latex','String',...
    '$${\phi=13.33}\hspace{12pt}{\nu\approx0.6}$$',...
      'Position',[2 1400],'FontSize',12)
hold off;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%                          END OF PROGRAM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

APPENDIX I

SCRIPT FILES IMPLEMENTED ON LEMIEUX.PSC.EDU

This appendix gives the script files needed to run simulations on lemieux.psc.edu.

A combination of schester.sh and msims.sh are used to run four jobs on the four processors of a node on lemieux.psc.edu

Script file schester.sh
This script is the main script that is submitted to the queue via the qsub command. It contains information required by lemieux.psc.edu such as an 18 hour wall time, one node with four processors, combined output and error files. The body of the script begins by changing the current directory to $SCRATCH, and then using one node and four processors running another script msims.sh (described below).

```
#!/bin/csh
#PBS -l walltime=18:00:00
#PBS -l rmsnodes=1:4
#PBS -j oe
cd $SCRATCH
prun -N ${RMS_NODES} -n ${RMS_PROCS} ./msims.sh
```

Script file msims.sh
This script file is called by schester.sh. The ${RMS_RANK} command lets the user run on a specific processor of a node. When called this script will loop through all the processors on the node the schester.sh will run on. When called the script will change the directory to SCRATCH/1.${RMS_RANK}. Where RMS_RANK is the processor number, it will then run the executable file schesterfpe in the directory SCRATCH/1.${RMS_RANK}.

```
#!/bin/csh
cd $SCRATCH/1.${RMS_RANK}
./schesterfpe
```

Script file copy.sh

This script file is here primarily as an example of another type of script available in the csh shell. This script was used to create a backup of important information in the $HOME directory from the original, but volatile data in the $SCRATCH directory. To run this script type "csh copy.sh" at the command prompt.

```
#! /bin/csh
#
echo -n "What node do you want to backup files in? "
set node = $<
#
echo -n "Backup files in node ${node} (y,n): "
set modfiles = $<
switch ($modfiles)
    case y:
        echo "Files in node ${node} will get copied"
        breaksw
    default:
        echo "Script stopped"
        goto error
        breaksw
endsw
switch ($node)
    case 1:
      cd $SCRATCH/1.0
      cp i3ds d3ds1000 floorforce $HOME/1.0/.
      cd $SCRATCH/1.1
      cp i3ds d3ds1000 floorforce $HOME/1.1/.
      cd $SCRATCH/1.2
      cp i3ds d3ds1000 floorforce $HOME/1.2/.
      cd $SCRATCH/1.3
      cp i3ds d3ds1000 floorforce $HOME/1.3/.
      echo 'node 1 backed up'
      breaksw
    case 2:
      cd $SCRATCH/2.0
      cp i3ds d3ds1000 floorforce160 $HOME/2.0/.
      cd $SCRATCH/2.1
      cp i3ds d3ds1000 floorforce160 $HOME/2.1/.
      cd $SCRATCH/2.2
      cp i3ds d3ds1000 floorforce160 $HOME/2.2/.
      cd $SCRATCH/2.3
      cp i3ds d3ds1000 floorforce160 $HOME/2.3/.
      echo 'node 2 backed up'
      breaksw
    case 3:
      cd $SCRATCH/3.0
      cp i3ds d3ds1000 floorforce $HOME/3.0/.
      cd $SCRATCH/3.1
      cp i3ds d3ds1000 floorforce $HOME/3.1/.
      cd $SCRATCH/3.2
      cp i3ds d3ds1000 floorforce $HOME/3.2/.
      cd $SCRATCH/3.3
      cp i3ds d3ds1000 floorforce $HOME/3.3/.
      echo 'node 3 backed up'
      breaksw
```

```
case 4:
  cd $SCRATCH/4.0
  cp i3ds d3ds1000 floorforce $HOME/4.0/.
  cd $SCRATCH/4.1
  cp i3ds d3ds1000 floorforce $HOME/4.1/.
  cd $SCRATCH/4.2
  cp i3ds d3ds1000 floorforce $HOME/4.2/.
  cd $SCRATCH/4.3
  cp i3ds d3ds1000 floorforce $HOME/4.3/.
  echo 'node 4 backed up'
  breaksw
case 5:
  cd $SCRATCH/5.0
  cp i3ds d3ds1000 floorforce $HOME/5.0/.
  cd $SCRATCH/5.1
  cp i3ds d3ds1000 floorforce $HOME/5.1/.
  cd $SCRATCH/5.2
  cp i3ds d3ds1000 floorforce $HOME/5.2/.
  cd $SCRATCH/5.3
  cp i3ds d3ds1000 floorforce $HOME/5.3/.
  echo 'node 5 backed up'
  breaksw
case 6:
  cd $SCRATCH/6.0
  cp i3ds d3ds1000 floorforce $HOME/6.0/.
  cd $SCRATCH/6.1
  cp i3ds d3ds1000 floorforce $HOME/6.1/.
  cd $SCRATCH/6.2
  cp i3ds d3ds1000 floorforce $HOME/6.2/.
  cd $SCRATCH/6.3
  cp i3ds d3ds1000 floorforce $HOME/6.3/.
  echo 'node 6 backed up'
  breaksw
case 7:
  cd $SCRATCH/7.0
  cp i3ds d3ds1000 floorforce $HOME/7.0/.
  cd $SCRATCH/7.1
  cp i3ds d3ds1000 floorforce $HOME/7.1/.
  cd $SCRATCH/7.2
  cp i3ds d3ds1000 floorforce $HOME/7.2/.
  cd $SCRATCH/7.3
  cp i3ds d3ds1000 floorforce $HOME/7.3/.
  echo 'node 7 backed up'
  breaksw
case 8:
  cd $SCRATCH/8.0
  cp i3ds d3ds1000 floorforce $HOME/8.0/.
  cd $SCRATCH/8.1
  cp i3ds d3ds1000 floorforce $HOME/8.1/.
  cd $SCRATCH/8.2
  cp i3ds d3ds1000 floorforce $HOME/8.2/.
  cd $SCRATCH/8.3
  cp i3ds d3ds1000 floorforce $HOME/8.3/.
  echo 'node 8 backed up'
  breaks
case 10:
  cd $SCRATCH/10.0
```

```
  cp i3ds d3ds1000 rolling $HOME/10.0/.
  cd $SCRATCH/10.1
  cp i3ds d3ds1000 rolling $HOME/10.1/.
  cd $SCRATCH/10.2
  cp i3ds d3ds1000 rolling $HOME/10.2/.
  cd $SCRATCH/10.3
  cp i3ds d3ds1000 rolling $HOME/10.3/.
  echo 'node 10 backed up'
  breaksw
case 11:
  cd $SCRATCH/11.0
  cp i3ds d3ds1000 rolling $HOME/11.0/.
  cd $SCRATCH/11.1
  cp i3ds d3ds1000 rolling $HOME/11.1/.
  cd $SCRATCH/11.2
  cp i3ds d3ds1000 rolling $HOME/11.2/.
  cd $SCRATCH/11.3
  cp i3ds d3ds1000 rolling $HOME/11.3/.
  echo 'node 11 backed up'
  breaksw
case 12:
  cd $SCRATCH/12.0
  cp i3ds d3ds1000 rolling $HOME/12.0/.
  cd $SCRATCH/12.1
  cp i3ds d3ds1000 rolling $HOME/12.1/.
  cd $SCRATCH/12.2
  cp i3ds d3ds1000 rolling $HOME/12.2/.
  cd $SCRATCH/12.3
  cp i3ds d3ds1000 rolling $HOME/12.3/.
  echo 'node 12 backed up'
  breaksw
case 13:
  cd $SCRATCH/13.0
  cp i3ds d3ds1000 rolling position $HOME/13.0/.
  cd $SCRATCH/13.1
  cp i3ds d3ds1000 rolling position $HOME/13.1/.
  cd $SCRATCH/13.2
  cp i3ds d3ds1000 rolling position $HOME/13.2/.
  cd $SCRATCH/13.3
  cp i3ds d3ds1000 rolling position $HOME/13.3/.
  echo 'node 13 backed up'
  breaksw
case 14:
  cd $SCRATCH/14.0
  cp i3ds d3ds1000 rolling position $HOME/14.0/.
  cd $SCRATCH/14.1
  cp i3ds d3ds1000 rolling position $HOME/14.1/.
  cd $SCRATCH/14.2
  cp i3ds d3ds1000 rolling position $HOME/14.2/.
  cd $SCRATCH/14.3
  cp i3ds d3ds1000 rolling position $HOME/14.3/.
  echo 'node 14 backed up'
  breaksw
case 15:
  cd $SCRATCH/15.0
  cp i3ds d3ds1000 rolling position $HOME/15.0/.
  cd $SCRATCH/15.1
```

```
      cp i3ds d3ds1000 rolling position $HOME/15.1/.
      cd $SCRATCH/15.2
      cp i3ds d3ds1000 rolling position $HOME/15.2/.
      cd $SCRATCH/15.3
      cp i3ds d3ds1000 rolling position $HOME/15.3/.
      echo 'node 15 backed up'
      breaksw
    case 16a:
      cd $SCRATCH/16a.0
      cp i3ds d3ds1000 floorforce tensor $HOME/16a.0/.
      cd $SCRATCH/16a.1
      cp i3ds d3ds1000 floorforce tensor $HOME/16a.1/.
      cd $SCRATCH/16a.2
      cp i3ds d3ds1000 floorforce tensor $HOME/16a.2/.
      cd $SCRATCH/16a.3
      cp i3ds d3ds1000 floorforce tensor $HOME/16a.3/.
      echo 'node 16a backed up'
      breaksw
    case 16b:
      cd $SCRATCH/16b.0
      cp i3ds d3ds1000 floorforce tensor $HOME/16b.0/.
      cd $SCRATCH/16b.1
      cp i3ds d3ds1000 floorforce tensor $HOME/16b.1/.
      cd $SCRATCH/16b.2
      cp i3ds d3ds1000 floorforce tensor $HOME/16b.2/.
      cd $SCRATCH/16b.3
      cp i3ds d3ds1000 floorforce tensor $HOME/16b.3/.
      echo 'node 16b backed up'
      breaksw
    case 17a:
      cd $SCRATCH/17a.0
      cp i3ds d3ds1000 floorforce tensor $HOME/17a.0/.
      cd $SCRATCH/17a.1
      cp i3ds d3ds1000 floorforce tensor $HOME/17a.1/.
      cd $SCRATCH/17a.2
      cp i3ds d3ds1000 floorforce tensor $HOME/17a.2/.
      cd $SCRATCH/17a.3
      cp i3ds d3ds1000 floorforce tensor $HOME/17a.3/.
      echo 'node 17a backed up'
      breaksw
    case 17b:
      cd $SCRATCH/17b.0
      cp i3ds d3ds1000 floorforce tensor $HOME/17b.0/.
      cd $SCRATCH/17b.1
      cp i3ds d3ds1000 floorforce tensor $HOME/17b.1/.
      cd $SCRATCH/17b.2
      cp i3ds d3ds1000 floorforce tensor $HOME/17b.2/.
      cd $SCRATCH/17b.3
      cp i3ds d3ds1000 floorforce tensor $HOME/17b.3/.
      echo 'node 17b backed up'
      breaksw
    default:
      echo 'script stopped'
      goto error
endsw
#
    error:
```

Script file d3dsmv.sh

This script file is here primarily as an example of a script available in the bash shell. To run this script type "bash d3dsmv.sh" at the command prompt.

```
#! /bin/bash
#
# user must input the node to modifiy files in
# this script will not change the input files
#
echo -n "What node do you want to modify files in (1,2,3,4): "
set node = $<
#
# test that the input is in range
if ($node < 1 || $node > 4) then
    echo "Script stopped"
    goto error
endif
#
# make sure you want to do this
#
echo -n "Modify files in node ${node} (y,n): "
set modfiles = $<
switch ($modfiles)
    case y:
      echo "Files in node ${node} will get modified"
      breaksw
    default:
      echo "Script stopped"
      goto error
      breaksw
endsw
#
# if everything is good then ....
#
switch ($node)
    case 1:
      cd $SCRATCH/1.0
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/1.1
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/1.2
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/1.3
      mv d3ds d3ds1000
      rm z* *~
      breaksw
    case 2:
      cd $SCRATCH/2.0
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/2.1
      mv d3ds d3ds1000
      rm z* *~
```

```
      cd $SCRATCH/2.2
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/2.3
      mv d3ds d3ds1000
      rm z* *~
      breaksw
    case 3:
      cd $SCRATCH/3.0
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/3.1
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/3.2
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/3.3
      mv d3ds d3ds1000
      rm z* *~
      breaksw
    case 4:
      cd $SCRATCH/4.0
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/4.1
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/4.2
      mv d3ds d3ds1000
      rm z* *~
      cd $SCRATCH/4.3
      mv d3ds d3ds1000
      rm z* *~
      breaksw
    default:
      echo "Script stopped"
      goto error
endsw
#
    error:
```

REFERENCES

1 Janssen, H. A. and Z. Vereins, Deutsh Eng., 1895. **39**: p. 1045.

2 Nedderman, R.M., *Statics and Kinematics of Granular Materials.* 1992, Cambridge, U.K.: Cambridge University Press. 352.

3 Arroyo-Cetto, D., et al., *Commpaction force in a confined granular column.* Physical Review E, 2003. **68**: p. 051301.

4 Bertho, Y., F. Giorgiutti-Dauphine, and J.-P. Hulin, *Dynamic Janssen effect on granular packing with moving walls.* Physical Review Letters, 2003. **90**: p. 144301.

5 Bratberg, I., K.J. Maloy, and A. Hansen, *Validity of the Janssen law in narrow granular columns.* European Physical Journal E, 2005. **18**: pp. 245-252.

6 Landry, J.W. and G.S. Grest, *Granular packings with moving side walls.* Physical Review E, 2004. **69**: p. 031303.

7 Landry, J.W., G.S. Grest, and J. Plimpton, *Discrete element simulations of stress distribution in silos: crossover from two to three dimensions.* Powder Technology, 2004. **138**: pp. 233-239.

8 Landry, J.W., et al., *Confined granular packings: structure, stress and forces.* Physical Review E, 2003. **67**: p. 041303.

9 Marconi, U.M.B., A. Petri, and A. Vulpiani, *Janssen's law and stress fluctuations in confined dry granular materials.* Physica A, 2000. **280**: pp. 279-288.

10 Ovarlez, G. and E. Clement, *Elastic medium confined in a column versus the Janssen experiment.* European Physical Journal E, 2005. **16**: pp. 421-438.

11 Ovarlez, G., C. Fond, and E. Clement, *Overshoot effect in the Janssen granular column: A crucial test for granular mechanics.* Physical Review E, 2003. **67**: p. 060302.

12 Ovarlez, G., E. Kolb, and E. Clement, *Rheology of a confined granular material.* Physical Review E, 2001. **64**: p. 060302.

13 Vanel, L. and E. Clement, *Pressure screening and fluctuations at the bottom of a granular column.* European Physical Journal B, 1999. **11**: pp. 525-533.

14 Vanel, L., et al., *Stresses in silos: Comparison between theoretical models and new experiments.* Physical Review Letters, 2000. **84**: pp. 1439-1442.

15 Sperl, M., *experiments on corn pressure in silo cells - translation and comment of Janssen's paper from 1895.* Granular Matter, 2006. **8**(2): pp. 59-65.

16 Klob, E., T. Mazozi, and E.C. Duran, *Force fluctuations in a vertically pushed granular column.* European Physical Journal B, 1999. **8**: pp. 483-491.

17 Walton, O.R., *Potential discrete element simulation applications ranging from airborne fines to pellet beds.* SAE 2004 Transactions: J. Aerospace, 2004. **2004-01-2329**: pp. 1-14.

18 Coppersmith, S.N., et al., *Model for force fluctuations in bead packs.* Physical Review E, 1996. **53**(5): p. 4673.

19 Cundall, P.A. and O.D.L. Strack, *A discrete numerical model for granular assemblies.* Geotechnique, 1979. **29**(1): pp. 47-65.

20 Campbell, C.S., *Self-diffusion in granular shear flows.* J. Fluid Mech., 1997. **348**: pp. 85-101.

21 Gallas, J.A.C., et al., *Molecular dynamics simulation of size segregation in three dimensions.* J. Stat. Phys., 1996. **82**(1/2): pp. 443-450.

22 Goldhirsch, I., M.L. Tan, and G. Zanetti, *Molecular dynamical studies of granular fluids I: The unforced granular gas in two dimensions.* J. Scientific Computing, 1993. **8**: pp. 1-40.

23 Kondic, L. and R.P. Behringer, *Elastic energy, fluctuations and temperature for granular materials.* Europhysics Letters, 2004. **67**(2): pp. 205-211.

24 Lan, Y. and A.D. Rosato, *Convection related phenomena in vibrated granular beds.* Physics of Fluids, 1997. **9**(12): pp. 3615-3624.

25 Liffman, K., et al., *A segregation mechanisms in vertically shaken bed.* Granular Matter, 2001. **3**: pp. 205-214.

26 Louge, M.Y., J.T. Jenkins, and M.A. Hopkins, *Computer simulations of rapid granular shear flows between parallel bumpy boundaries.* Physics of Fluids A, 1990. **2**(6): pp. 1042-1044.

27 Luding, S., *Granular materials under vibration: simulations of rotating spheres.* Physical Review E, 1995. **52**(4): pp. 4442-4457.

28 Lun, C.K.K., *Granular dynamics of inelastic spheres in Couette flow.* Phys. fluids, 1996. **8**(11): pp. 2868-2883.

29 McCarthy, J.J. and J.M. Ottino, *Particle dynamics simulation: a hybrid technique applied to granular mixing.* Powder Technology, 1998. **97**: pp. 91-99.

30 Poschel, T. and H.J. Herrmann, *Size segregation and convection.* Europhysics Letters, 1995. **29**(2): pp. 123-128.

31 Rosato, A., et al., *Experimental, simulation and nonlinear dynamics analysis of Galton's board.* Int. J. Nonlin. Sci. and Num. Simulation, 2004. **5**: pp. 289-312.

32 Savage, S.B., *Numerical simulations of Couette flow of granular materials: spatio-temporal coherence and 1/f noise*, in *Physics of Granular Media*, J. Dodds and D. Bideau, Editors. 1991, Nova Scientific Publishers: New York. pp. 343-362.

33 Walton, O.R., *Numerical simulation of inelastic, frictional particle-particle interactions*, in *Particulate Two-Phase Flow*, M.C. Roco, Editor. 1992, Butterworths: Boston. pp. 884-911.

34 Wassgren, C.R., et al., *Dilute granular flow around an immersed cylinder.* Physics of Fluids, 2003. **15**(11): pp. 1-13.

35 Sweetman, M., *Addition of a chain-cell search method and a van der Waals force model to a particle dynamics code*, in *Mechanical Engineering*. 2003, New Jersey Institute of Technology: Newark, NJ.

36 Walton, O.R. *Numerical simulation of inclined chute flows of monodisperse, inelastic, frictional spheres.* in *Second U.S.-Japan Seminar on Micromechanics of Granular Materials*. 1991. Potsdam, NY: Elsevier.

37 Walton, O.R. and R.L. Braun, *Viscosity and temperature calculations for assemblies of inelastic, frictional disks.* Journal of Rheology, 1986. **30**(5): pp. 949-980.

38 Walton, O.R. and R.L. Braun, *Stress calculations for assemblies of inelastic spheres in uniform shear.* Acta Mechanica, 1986. **63**: pp. 73-86.

39 Lan, Y., *Particle dynamics modeling of vibrating granular beds*, in *Mechanical Engineering Department*. 1994, New Jersey Institute of Technology: Newark.

40 Mindlin, R.D. and H. Deresiewicz, *Elastic spheres in contact under varying oblique forces.* J. Appl. Mech., 1953. **20**: p. 327.

41 Kim, H.J., *Particle Dynamics Modeling of Boundary Effects in Granular Couette Flow*, in *Mechanical Engineering*. 1992, New Jersey Institute of Technology: Newark, NJ.

42 Haile, J.M., *Molecular Dynamics Simulation: Elementary Methods*. 1992: J. Wiley and Sons. 489.

43 Skinner, A.E., *A note on the influence of interparticulate friction on the shearing strength of a random assembly of spherical particles.* Geotechnique, 1969. **19**(1): pp. 150-157.

44 Thornton, C., *Numerical simulations of deviatoric shear deformation of granular media.* Geotechnique, 2000. **0**: pp. 43-53.

45 Thornton, C. and S.J. Antony, *Quasi-static deformation of particulate media.* Phil. Trans. Roy. Soc. London A, 1998. **356**: pp. 2763-2782.