ABSTRACT

COPPOLA, STEPHEN W. Graduated Random Fill Method for Large-Scale Isotropy in DEM. (Under the direction of Dr. Lawrence Silverberg).

This paper introduces a graduated random fill (GRF) method for achieving large-scale isotropy in DEM models for the case in which the modeling of small-scale properties is relaxed. The soft-sphere DEM formulation accommodates multi-disciplinary/phase problems via a Boscovich force (BF), which is constructed from a simple banded bilinear function. We generated the isotropic body in three stages – filling, curing, and cutting, and then tested its isotropy. The effectiveness of the GRF method was illustrated through comparative stress-strain tests of it and an alternate close pack body. In the comparative tests, the bodies exhibited robust ductile behavior with the GRF body having an isotropic deviation of about 3%.
Graduated Random Fill Method for Large-Scale Isotropy in DEM

by

Stephen W. Coppola

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Mechanical Engineering

Raleigh, North Carolina

2017

APPROVED BY:

Dr. Lawrence Silverberg
Committee Chair

Dr. Hsiao-Ying Shadow Huang

Dr. Mohammed Zikry
BIOGRAPHY

Stephen W. Coppola grew up in Pinehurst, North Carolina, where he attended the O’Neal high school. Because of his love of logic and designing he pursued and then completed a B.S. in Mechanical Engineering from NCSU College of Engineering in Raleigh. He continued his studies at NCSU, where he is pursuing a Master’s of Science in Mechanical Engineering.
# TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................ iv

INTRODUCTION ........................................................................................................... 1

METHODS .................................................................................................................. 3

  Governing Equations ............................................................................................ 3

  Graduated Random Fill Method ........................................................................... 8

RESULTS....................................................................................................................... 10

DISCUSSION ............................................................................................................... 13

  Filling Efficiency .................................................................................................... 13

  Residual Stress ....................................................................................................... 13

  Range of Particle Radii .......................................................................................... 13

  Curing Time ............................................................................................................ 13

  The Boundary ........................................................................................................ 14

  Large-Scale Behavior ............................................................................................. 14

  Isotropy .................................................................................................................. 15

SUMMARY AND CONCLUSIONS.................................................................................... 16

REFERENCES............................................................................................................. 17

APPENDICES............................................................................................................. 19

  Appendix A. Filling, Curing, and Cutting Code .................................................... 19

  Appendix B. Testing Code ...................................................................................... 28
LIST OF FIGURES

Figure 1: The Band Function ........................................................................................................ 5

Figure 2: Particle Interaction ......................................................................................................... 6

Figure 3: GRF Method During Fill ............................................................................................... 10

Figure 4: Cutting ............................................................................................................................ 11

Figure 5: Stress-Stain Curves for GRF and Close Pack Blocks .................................................... 12
1 Introduction

With the advances in parallel computing and nearest neighbor sorting, Discrete Element Method (DEM) has recently resurfaced as a viable computational modeling tool compared to its alternatives, finite element method and finite difference method. DEM simulates large-scale properties with an abundant number of small-scale particles, making it computationally intensive, but its versatility in forming complex geometries and properties and its intuitiveness provides the analyst with the potential capability to approach problems that were otherwise difficult to model. DEM also has a discontinuous nature to it, making it prominent in fracture mechanics [1, 2, 3, 4] and granular materials [5, 6, 7]. It is also used in fluid flow [8, 9, 10], biological materials [11, 12], and multiphase problems [13, 14], because of the particles’ degrees of freedom. Similarly, DEM is used in multiscale [15, 16] and combined model problems [17, 18] for its simplistic and bounded nature. Despite the trending of DEM, and putting aside the computational burden, serious problems are found when trying to generate large-scale properties from smaller-scale particle distributions – even when the interest lies, not in modeling small-scale properties, but only in modeling the large-scale properties. One such case is isotropy. No small-scale crystal, owing to its pattern regularity, produces isotropic behavior. Isotropy is prevalent in fluids and frequent in solids so the question of how to generate isotropic behavior at the large scale is important to DEM. It is known that isotropy requires a destruction of small-scale regularity, achievable through a random distribution of some sort. Random distributions of particles of equal size have been shown to create isotropic behavior [19]. Furthermore, random distributions of unequal size
potentially offer advantages over ones of equal size. More generally, some kind of methodology for filling regions with particles which lead to isotropic behavior has potential advantageous. This paper introduces an algorithm to fill a region with small-scale particles and produce large scale isotropic behavior. To achieve the broadest multi-disciplinary/phase model, we first formulate a Boscovich force (BF) to be used in the DEM. Next, we introduce an algorithm that generates isotropic behavior at the large scale, called the graduated random fill (GRF) method, and then cut the material into blocks. In the results section the isotropy of the blocks of material is tested and then compared with alternative close pack fill blocks of roughly equal number of particles. Finally, the discussion and conclusion sections review the isotropy data and make other observations.
2 Method

Governing equations

The DEM considers a system of \(n\) particles of mass \(m_i\) \((i = 1, 2, \ldots, n)\). The internal force components between the \(i^{th}\) and \(j^{th}\) particles are denoted by \(f_{xij}, f_{yij},\) and \(f_{zij}\) and the external force components acting on the \(i^{th}\) particle are denoted by \(F_{xi}, F_{yi},\) and \(F_{zi}\). The governing equations are:

\[
m_i \ddot{x}_i = \sum_{j=1}^{n} f_{xij} + F_{xi} \quad m_i \ddot{y}_i = \sum_{j=1}^{n} f_{yij} + F_{yi} \quad m_i \ddot{z}_i = \sum_{j=1}^{n} f_{zij} + F_{zi}
\]

We consider a soft sphere model wherein the internal forces are central forces, given by

\[
f_{xij} = f_{ij} \frac{x_j - x_i}{D_{ij}} \quad f_{yij} = f_{ij} \frac{y_j - y_i}{D_{ij}} \quad f_{zij} = f_{ij} \frac{z_j - z_i}{D_{ij}}
\]

where \(D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}\). Next, we define the characteristic length \(D_0\), the characteristic mass \(m_0\), and the characteristic frequency \(\omega_0\). The resulting non-dimensional governing equations are:

\[
\bar{m}_i \ddot{x}'_i = \sum_{j=1}^{n} \bar{f}_{xij} + \bar{F}_{xi} \quad \bar{m}_i \ddot{y}'_i = \sum_{j=1}^{n} \bar{f}_{yij} + \bar{F}_{yi} \quad \bar{m}_i \ddot{z}'_i = \sum_{j=1}^{n} \bar{f}_{zij} + \bar{F}_{zi}
\]

\[
\bar{f}_{xij} = \bar{f}_{ij} \frac{x_j - x_i}{\bar{D}_{ij}} \quad \bar{f}_{yij} = \bar{f}_{ij} \frac{y_j - y_i}{\bar{D}_{ij}} \quad \bar{f}_{zij} = \bar{f}_{ij} \frac{z_j - z_i}{\bar{D}_{ij}}
\]

where \((\cdot)'\) is the derivative with respect to non-dimensional time \(\bar{t} = \omega_0 t\). The non-dimensional quantities are: \(\bar{x} = x / D_0\), \(\bar{y} = y / D_0\), \(\bar{z} = z / D_0\), \(\bar{f} = f / m_0 \omega_0^2 D_0\), and \(\bar{F} = F / m_0 \omega_0^2 D_0\). Denote the non-dimensional local coordinate between the \(i^{th}\) and \(j^{th}\) pair of particles as \(\bar{s}\). The global coordinates are then
Notice that $\vec{s} = \vec{D}_j$ when $\vec{x} = \vec{x}_j, \vec{y} = \vec{y}_j, \vec{z} = \vec{z}_j$. The central force $\vec{f}_{ij}$ between the pair of particles is a patchwork of effects that depend on the dominant physics in the different regions between the pair, written

$$\vec{f}_{ij} = \begin{cases} 
\vec{f}_1 & s_1 < \vec{s} < s_2 \\
\vec{f}_2 & s_2 < \vec{s} < s_3 \\
& \vdots \\
\vec{f}_p & s_p < \vec{s} < s_{p+1}
\end{cases}$$

For computational purposes, it is advantageous to express the patchwork as a smooth function. A simple way to do this is to rewrite Eq. (5) as

$$\vec{f}_{ij} = \sum_{r=1}^{p} \vec{f}_{r} band(\vec{s}_r, \vec{s}_{r+1})$$

in which

$$band(\vec{s}_r, \vec{s}_s) = \frac{1}{2} \left[ erf(\beta(\vec{s} - \vec{s}_r)) - erf(\beta(\vec{s} - \vec{s}_s)) \right]$$

is a band function that captures the dominant physics in each of the regions. The band function goes to zero to the left of $s_r$ and to the right of $s_s$. Referring to Eq. (6) and Fig. 1, the band function smoothly patches together the forces in the different regions, where $\beta$ is a smoothing parameter.
The ability to accommodate the dominant physics in the different regions between and near a pair of particles constitutes one of the important features of DEM. In principle, the banded force enables the analyst to model processes that undergo phase transition, and that are mechanical, electrodynamic, and/or thermal in origin, a unifying theme dating back to the 18th century [23]. A single continuous force that models physical behavior is referred to as the Boscovich force (BF). Arguably, the simplest BF that captures multi-disciplinary/phase behavior is the banded bilinear model introduced below (See Fig. 2): A single continuous force that broadly models physical behavior is referred to as the Boscovich force (BF). Arguably, the simplest central force that captures multi-disciplinary/phase behavior is the banded bilinear model introduced below (See Fig. 2):
\[(8) \quad \vec{f}_{ij} = \vec{f}_{Cij} + \vec{f}_{Tij} \]
\[
\vec{f}_{Cij} = [\vec{c}_{ij} g'_{ij} + \vec{k}_{ij} g_{ij}] Band_C \quad \vec{f}_{Tij} = [\vec{c}'_{ij} g'_{ij} + \vec{k}'_{ij} g_{ij}] Band_T
\]

where

\[(9) \quad Band_C = \frac{1}{2} [\text{erf}(\beta D_{ij}) - \text{erf}(\beta(D_{ij} - (\vec{r}_i + \vec{r}_j)))]
\]
\[
Band_T = \frac{1}{2} [\text{erf}(\beta(D_{ij} - (\vec{r}_i + \vec{r}_j))) - \text{erf}(\beta(D_{ij} - (\vec{R}_i + \vec{R}_j)))]
\]

where \(\vec{r}_i\) and \(\vec{R}_i\) are the non-dimensional physical and attraction radii of the \(i^{th}\) particle. In Eq. (8) the non-dimensional rate of change of the distance is

\[(10) \quad \vec{D}_{ij}' = D_0 \alpha_0 \frac{(\vec{x}_j - \vec{x}_i)(\vec{x}_j' - \vec{x}_i') + (\vec{y}_j - \vec{y}_i)(\vec{y}_j' - \vec{y}_i') + (\vec{z}_j - \vec{z}_i)(\vec{z}_j' - \vec{z}_i')}{\vec{D}_{ij}}
\]

**Figure 2:** Particle Interactions: outside attraction radius (A), contacting attraction radius (B), contacting physical radii (C), overlapping physical radii (D)
Referring to Fig. 2, the $i^{th}$ particle has a compressive stiffness of $k_C$ and a tensile stiffness of $k_T$ as well as a compressive damping of $c_C$ and a tensile damping of $c_T$. They are related to their non-dimensional counterparts by $k_i = \bar{k}_i m_0 \omega_0^2$ and $c_i = \bar{c}_i m_0 \omega_0$, respectively. Notice that the stiffnesses and damping in Eq. (8) are associated with pairs of particles, obtained from the stiffnesses and damping of the individual particles by

$$\frac{1}{k} = \frac{1}{k_i} + \frac{1}{k_j} \quad \frac{1}{c} = \frac{1}{c_i} + \frac{1}{c_j}$$

(12)

The banded bilinear force model is similar to the viscoelastic force model; the difference being that the viscoelastic force model is not banded [20]. Reflecting the underlying physics and for efficiency, it is natural to prescribe physical properties of individual particles and not of pairs of particles explicitly; the properties of pairs of particles are derivable from individual particle properties. For computational purposes, the non-dimensional form of the governing equations are numerically integrated. We defined the characteristic length as the smallest particle diameter, the characteristic mass as the smallest particle mass, and the characteristic frequency as the highest natural frequency of any pair of particles, written

$$D_0 = \min(2r_i) \quad m_0 = \min(m_i) \quad \omega_0 = \sqrt{\frac{2 \max(k_{ij}, k_{Cij})}{m_0}}$$

(13)

This choice of non-dimensional parameters leads to a step size for any system that is on the constant order of $\Delta T = \frac{1}{50} \left( \frac{2 \pi}{\omega_0} \right)[12]$. The banded bilinear form is employed later in the paper.
Graduated Random Fill Method

The aim of the graduated random fill (GRF) method proposed herein is ultimately to generate a model that possesses isotropic properties at the large scale with the caveat that the interest does not lie in reproducing or emulating any particular small-scale behavior. The procedure is applicable to both fluid and solid bodies and divides into a) a filling stage, b) a curing stage, and c) a cutting stage. At the filling stage, the GRF method fills an initial region with particles. The locations of the filled particles constitute the initial state of the system. At the curing stage, the particles settle and the system reaches equilibrium, but deforms the shape of its boundary. At the cutting stage, the desired isotropic body is cut from a larger region. Note that the filling stage is a purely geometric process, independent of material behavior, and that the particles’ material properties govern the effectiveness of the curing stage.

Broadly, the GRF method randomly places particles of decreasing size in a region, one after another with a maximum allowable overlap from other particles. The individual steps in the GRF method are as follows:

A. **Start.** Define the initial boundary of the region to fill. The parameters specified before beginning the iteration are the maximum overlap radius $O_{\text{max}}$, the particle radius decrement $\alpha$, the radius $r_1$ of the first particle placed in the region, and the maximum number $M$ of rejections. Initialize the particle count $i$ to 0 and the rejection count $k$ to 0.

B. **Particle update.** Increase $i$ by 1 and update the particle radius according to $r_i = \alpha r_{i-1}$.

C. **Particle placement.** Calculate a uniformly random set of coordinates for the $i^{th}$ particle.

D. **Determine acceptance.** Accept the particle if it lies in the region and overlaps with any other particle by no more than $O_{\text{max}}$, where $\text{overlap}_{ij} = 1 - D_{ij}/(r_i - r_j)$, otherwise
reject the particle. If the particle is accepted return to step B. If rejected, check whether \( k \) is less than \( M \) and if so increase \( k \) by 1 and then return to step C.

E. **End.** Set \( i \) to \( n \) and stop.

After creating the body, the need arises to test the deviation in the body’s isotropy; in particular, its directional dependence in normal stress as the body is strained. The comprised test measures normal stress of \( l \) specimens oriented at the angles \( \theta_r \) \((r = 1, 2, \ldots, l)\) over \( N \) increments \((s = 1, 2, \ldots, N)\). Denote the normal stress of the \( r^{th} \) specimen at the \( s^{th} \) strain by \( \sigma_r(\varepsilon_s) \) and the average normal stress over the \( l \) specimens at the \( s^{th} \) strain by \( \sigma_{av}(\varepsilon_s) \). The isotropic deviation in the body of the \( r^{th} \) sample is defined as

\[
\delta_r = \frac{1}{N} \left[ \sum_{j=1}^{l} 2 \left( \frac{\sigma_r(\varepsilon_s) - \sigma_{av}(\varepsilon_s)}{\sigma_r(\varepsilon_s) + \sigma_{av}(\varepsilon_s)} \right) \right] \times 100
\]
3 Results

Viewing Figure 3, the GRF method during the filling of a circular region of radius 30\(r_1\) is illustrated, with the parameters \(r_1 = 1\), \(O_{max} = 0.285\), \(\alpha = 0.999925\), and \(M = 10^5\).

![Figure 3: GRF Method During Fill: \(M = 10^2\) (n = 174) in A, \(M = 10^3\) (n = 1065) in B, \(M = 10^4\) (n = 3472) in C, and \(M = 10^5\) (n = 4626) in D](image)

After completing the filling stage, we conducted the curing stage and brought the body to equilibrium. During this process, and later while testing isotropy, computational efficiency was increased by limiting force calculations to an area of influence around each particle, illustrated in figure 4, with the red and black circles denoting the maximum interaction range and the area of influence of a particle. These areas were displayed when running the model in order to ensure error was not produced from limiting the force calculations. Lastly, because of the relatively slow moving particles in DEM solid models [21] the radius of influence around each particle was chosen to be update only every 100 numerical integration steps.
After the curing stage, we formed rectangular sections during the cutting step (See Fig. 4). As shown, we cut $l = 4$ rectangular sections from the circular body at angles of $0^0$, $30^0$, $60^0$, and $90^0$. The resulting rectangular sections served as GRF method testing specimens.

![Figure 4: Cutting: $0^0$ for A, $30^0$ for B, $60^0$ for C, $90^0$ for D](image)

The GRF method testing specimens were compared against close pack (CP) testing specimens (horizontally stacked rows of particles offset by a radius). The CP testing specimens possess the tightest packing density for uniform sized particles and the greatest large-scale stiffness among the different arrangements, and therefore provide a good benchmark against which to compare the GRF method testing specimens. The radius of the CP particles was set to $0.545r_1$ in order to approximately match the total number of particles with the GRF method specimens.
The GRF and CP block specimens underwent the same compression test sampled $N = 1375$ times from zero to 4% strain, shown in Fig. 5. The isotropic deviation for each GRF and CP block are:

**GRF method:**

- $\delta_A = 3.17\%$
- $\delta_B = 2.64\%$
- $\delta_C = 3.60\%$
- $\delta_D = 3.22\%$

**CP:**

- $\delta_A = 11.69\%$
- $\delta_A = 11.64\%$
- $\delta_A = 11.61\%$
- $\delta_A = 11.66\%$
4 Discussion

First consider some observed relationships pertaining to the selection of the initialization parameters in the GRF method ($O_{\text{max}}, \alpha, r_1$ and $M$), and the physical parameters ($k_C, k_T$, and $m_0$).

- **Filling efficiency:** Fig. 3 shows that the rejection rate $\dot{M}$ increases nonlinearly with the number of particles, making higher maximum number of rejections $M$ less computational efficient.

- **Residual stresses:** High maximum overlap $O_{\text{max}}$ increases packing density but also increases residual energy released during the curing phase, which, at the extreme, causes the system to fragment. High compressive stiffness $k_C$ further increases residual energy whereas higher tensile stiffness $k_T$ prevent the body from fragmenting. We let $k_C = 3k_T = 20\pi r_1^2$

- **Range of particle radii:** The range of particle radii decreases as the particle radius decrement $\alpha$ approaches 1 from below and it increases with the number of particles $n$. We let $\eta / r_n = 0.5$.

- **Curing time:** A given particle reaches equilibrium (cures) fastest when its damping is critical and the overall system reaches equilibrium fastest when each of the particles are critically damped. The particles were given the same compressive stiffness and damping constants and the same tensile stiffness and damping constants, whereas the masses decreased in size. Therefore, only one particle could be critically damped, with the smaller particles becoming overdamped and the larger particles becoming
underdamped. To reduce the settling time (curing time), the middle particle was selected to be critically damped, letting $c_C = 2 \sqrt{k_C m_{n/2}}$ $c_T = 2 \sqrt{k_T m_{n/2}}$ where $m_{n/2}$ is the mass of the middle particle.

- **The boundary.** The GRF method can be applied to a boundary of any shape, however, the boundary deforms during the curing stage. Therefore, to obtain a more precise boundary, the desired shape is better obtained via the cutting stage. When testing multiple specimens the cutting stage also serves to provide a single body from which different specimens can be compared. Additionally, the precision of the boundary increases as the particles decrease in size and during the curing stage, the boundary deforms increasingly with larger attraction radii.

- **Large-scale behavior.** Referring to Fig. 5, first observe that the stresses in the stress-strain curves for both the GRF method and CP tests are of the same order of magnitude. This results from the matching of the stiffnesses in the tests. Next, for both the GRF and CP tests, consider the three ranges: (1) –0.25% to 0% strain, (2) 0% to 2% strain, and (3) 2% to 4% strain. In the first range for both the GRF tests and the CP tests, we find that all of the specimens undergo a small expansion resulting from a residual stress release that occurs after the cutting stage. In the second range for both the GRF tests and the CP tests, we observe typical ductile stress-strain behavior of a stress increase, peak, and relaxation. In the third range for both GRF tests and CP tests, we observe a second cycle of stress increase and peak, indicating robust ductility.
• **Isotropy:** Continuing to refer to Fig. 5, we observe low isotropic deviation in the GRF method testing specimens of 2.6% to 3.6%. The CP specimens undergo considerably more isotropic deviation – on the order of 12%. For the CP test, we also observe that the stress-strain curves for the $0^0$ and $60^0$ specimens match closely and, likewise, for the $30^0$ and $90^0$ specimens. This is expected because the CP body has $60^0$ symmetry.
5 Summary and Conclusions

This paper addressed the problem in DEM of generating large-scale isotropy for the case in which the modeling of small-scale properties is relaxed. A broad soft-sphere DEM formulation that accommodates multi-disciplinary/phase problems was set up via the introduction of a Boscovich force (BF), and in particular a banded bilinear force model. Next, we generated the isotropic body in three stages – filling, curing, and cutting, and then tested its isotropy. First, we developed a graduated random fill (GRF) method for the filling stage. The GRF method randomly places particles of decreasing size in a region, after which one cures the resulting body and then cuts it to a desired shape. The paper discussed filling efficiency, the impact of residual stress, the generation of a range of particle radii, and the approach to achieving a precise boundary. Finally, we illustrated the effectiveness of the method through comparative stress-strain tests of GRF and CP body isotropies. In the comparative tests, the GRF and CP bodies exhibited robust ductile behavior and the GRF body had an isotropic deviation of about 3%.

This paper successfully demonstrated a method of generating large-scale isotropy for DEM and, with it, produced a ductile material. The method has the potential of improving two and three dimension DEM models where isotropy is considered. Also this method could produce isotropic flows with different flow patterns from the graduated particle sizes. Lastly because of its good packing density the method could model hard materials as well as save on computational time.
REFERENCES


APPENDICES
Appendix A. Filling, Curing, Cutting Code

% function [] = Particle_Code()
% close all, clc, clear
% ngp = 0;
%
% %Video and Code Configuration
% Trial_Name='10^-1.5'; %Name of trial, change each run if you want to keep data saved
% videoLength = 1; %Length for video, in seconds
% frameRate = 10; %Frame rate for video, in frames per second
% videoName = Trial_Name; %Output file name, dont change
%
% %Particle Setup
% rc = 215; %radius of initial packing region
% m(1) =10^3; %mass l
% c(1) =.15*m(1)*10*5; %damper constant
% Kc(1) = m(1)*20; %Compressive Spring Constant
% Cf(1) = 10^2; %Band Function Constant (beta)
% Ka(1) = m(1)*.7*10; %Tensile Spring Constant
% Al(1) = 1.5; %Tensile Spring Length (in diameters from center) R
% Af(1) = Cf(1); %Band Function Constant (beta)
% d(1)=2*(m(1)/pi)^0.5; %Diameter, set to constant density
% z0(4*1-3)=500-rc+rand*2*rc; %Initial random x posistion
% z0(4*1-2)=0; %Initial x velocity
% z0(4*1-1) = 500+rand*(rc^2-(z0(4*1-3)-500)^2)^.5*(-1)^(round(rand*2+.5)); %initial random y posistion
% z0(4*1)=0; %Initial y velocity
% i=1;
% repeat = 0;
% while repeat < 10^5 %Total random placement attempts
% i=i+1;
% m(i) = m(i-1)*.99985; % Mass Decrement
% c(i)=c(1); %Damper Constant
% Kc(i) = Kc(1)*m(i)/m(1); %Compressive Spring Constant
% Cf(i) = Cf(1); % Beta
% Ka(i) = Ka(1)*m(i)/m(1); %Tensile Spring Constant
% Al(i) = Al(1); %Tensile Spring Length (in diameters) R
% Af(i) = Af(1); %Band Function Constant (beta)
% d(i)=2*(m(i)/pi)^0.5; %Diameter, constant density
% z0(4*i-3)=500-rc+rand*2*rc; %Initial random x posistion
% z0(4*i-2)=0; %Initial x velocity
% z0(4*i-1) = 500-500/2+rand*500/1; %Initial random y posistion square
% z0(4*i)=0; %Initial y velocity
% for j = 1:i-1 %Look at each other particle below it to check if Max overlap is passed
Dis(i,j) = ((z0(4*j-3)-z0(4*i-3))^2+(z0(4*j-1)-z0(4*i-1))^2)^0.5;
% particle distance magnitude
DisC = ((z0(4*i-3)-500)^2+(z0(4*i-1)-500)^2)^.5; %Distance particle from center of build circle
if Dis(i,j) < (2*((m(j)+m(i))/2/pi)^0.5)*.715 || DisC > rc %Distance from each other and Percent overlap
i=i-1;
repeat=repeat+1;
break
end
end

percent = m(i)/m(1)*100 %percent smallest/largest particle
ind = [(i+1)*4 (i+1)*4-1 (i+1)*4-2 (i+1)*4-3]; % indices to be removed
z0(ind) = [];
% remove last particle
nmp1 = i;
%Number of particles
nmp2 = 0;
% number of particles body 2 if you want multiple bodies
nmp = nmp1+nmp2;
%Total number of mobile particles
np = nmp %Number of total particles

k = 2;
% 1st element in list
moveD = d(1)*1.5;
%expected move distance
for i = 1:np %for each particle
Crange(i) = d(i)^.5*d(1)^.5*(Al(i)+Al(1)-2)/2+(d(i)+d(1))/2+moveD;
%Max attraction range for each particle + modeD = cutoff range
for j = i+1:np %Look at each other particle above
Dis(i,j) = ((z0(4*i-3)-z0(4*j-3))^2+(z0(4*i-1)-z0(4*j-1))^2)^0.5; % particle distance magnitude
if Dis(i,j) < Crange(i)
Clist(i,k) = j; %for each particle, a list of particles that are close
k = k+1;
end
end
Clist(i,1) = k-2;
k=2;
end

Accuracy = 50; %checks each osilation
deltaT = 2*pi/(max(Kc)/min(m))^.5/Accuracy %Step size, in seconds
update = 10; %how often clist is updated per second
% z = z0;
% %------------------------------------------------------Setup Done------------------------------------------------------

------
% tic                                 %start timing
% theta = linspace(0,2*pi,12);        %number of steps, rounded to
% % N = round(videoLength/deltaT);     %number of steps, rounded to
% % index = 0;                         %indexing term, used in the
% %     % framing calculation (leave at 0)
% %     %
% % fig = figure(1);
% % for i = 1:N                        % i is the moment in time
% %     time=(i-1)*deltaT;
% % z=integrator('Force_Functions',z,theta,deltaT,m,c,Kc,d,np,ngp,Al,Ka,Af,Cf,n
% % mpl,nmp2,Clst);
% %     %
% %     % r0 = round((1/deltaT)/frameRate); % rounded to ensure an
% %     % integer value
% %     isMultiple = (r0*round(double(i)/r0) == i );
% %     %
% %     % if mod(i, update) == 0          %update clist
% %         z0=z;
% %         clearvars Clst;
% %         k = 2;                       %1st element in list
% %         for ii = 1:np
% %             %for each particle
% %             %for each other particle
% %     k=2;
% %     end
% %     end
% %     if isMultiple == 1             % grab every 'r'th frame after the
% %         index = index+1;            % indexing update
% %         G(index) = getframe(fig);  %'fig' is the variable name of
% %         'fig' is the variable name of
% %         %
% %         %
% %         %
% %         %
% %             %Body 1
% %             xcir = d(j)/2*cos(theta) + z(4*j-3);
% %             ycir = d(j)/2*sin(theta) + z(4*j-1);
% plot(xcir/d(np),ycir/d(np),'Color',[0,0,1]);
% hold on
% end
%
% for j = round(nmpl/2) %Crange region of middle particle
%    xcir=cos(theta)*Crange(round(j))+ z0(4*round(j)-3);
%    ycir=sin(theta)*Crange(round(j))+ z0(4*round(j)-1);
%    plot(xcir/d(np),ycir/d(np),'Color',[0,0,0]);
%    hold on
%
%    % max attraction range largest w/ 2nd largest of same j particle
%    xcir=cos(theta)*(Crange(j)-moveD)+ z(4*round(j)-3);
%    ycir=sin(theta)*(Crange(j)-moveD)+ z(4*round(j)-1);
%    plot(xcir/d(np),ycir/d(np),'Color',[1,0,0]);
%    hold on
%    axis([0,1000/d(np),0,1000/d(np)]); %if you dont want to follow the particle
%    hold off
%    clc
%    percent = percent
%    nmp = nmp
%    done = i/N
% end
% end
% toc
% j=0; % rotate -0 degrees Cuts and stores particle locations to file
% for i=1:np
%    if (z(4*i-3)-500)<0
%        angle = 180+atand((z(4*i-1)-500)/(z(4*i-3)-500));
%        hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
%        angle = angle-0;
%        zshift(4*i-3)=hypotnus*cosd(angle)+500;
%        zshift(4*i-1)=hypotnus*sind(angle)+500;
%    else
%        angle = atand((z(4*i-1)-500)/(z(4*i-3)-500));
%        hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
%        angle = angle-0;
%        zshift(4*i-3)=hypotnus*cosd(angle)+500;
%        zshift(4*i-1)=hypotnus*sind(angle)+500;
%    end
%
% if zshift(4*i-3)> 575 || zshift(4*i-3)<425 %x cut % Cut particles into Blocks
%    zshift(4*i-3)=zshift(4*i-3)+9999;
% end
% if zshift(4*i-1)> 650 || zshift(4*i-1)<350 %y cut
%    zshift(4*i-1)=zshift(4*i-3)+9999;
% end
if zshift(4*i-3)<1001 && zshift(4*i-1)<1001 && zshift(4*i-3) > 0 &&
zshift(4*i-1) > 0;
j=j+1;
z2(4*j-3)=zshift(4*i-3); %converting kept particles to new matrix
z2(4*j-2)=z(4*i-2);
z2(4*j-1)=zshift(4*i-1);
z2(4*j-0)=z(4*i-0);
m2(j)=m(i);
CSC2(j)=Kc(i);% CSC2(j)=Kc(i);
c2(j)=c(i);% c2(j)=c(i);
d2(j)=d(i);% d2(j)=d(i);
TSC2(j)=Ka(i);% TSC2(j)=Ka(i);
TSL2(j)=Al(i);% TSL2(j)=Al(i);
TSF2(j)=Af(i);% TSF2(j)=Af(i);
end
end
fileID = fopen('Box1a.txt','w'); % Writing Conditions to File
fprintf(fileID,'%6.2f %12.8fn',z2);
fprintf(fileID,'%6.2f %12.8fn',m2);
fprintf(fileID,'%6.2f %12.8fn',CSC2);
fprintf(fileID,'%6.2f %12.8fn',c2);
fprintf(fileID,'%6.2f %12.8fn',d2);
fprintf(fileID,'%6.2f %12.8fn',TSC2);
fprintf(fileID,'%6.2f %12.8fn',TSL2);
fprintf(fileID,'%6.2f %12.8fn',TSF2);
fclose(fileID);

j=0; % rotate 30 degrees
for i=1:np
if (z(4*i-3)-500)<0
    angle = 180+atand((z(4*i-1)-500)/(z(4*i-3)-500));
    hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
    angle = angle+30;
    zshift(4*i-3)=hypotnus*cosd(angle)+500;
    zshift(4*i-1)=hypotnus*sind(angle)+500;
else
    angle = atand((z(4*i-1)-500)/(z(4*i-3)-500));
    hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
    angle = angle+30;
    zshift(4*i-3)=hypotnus*cosd(angle)+500;
    zshift(4*i-1)=hypotnus*sind(angle)+500;
end

if zshift(4*i-3)> 575 || zshift(4*i-3)<425 %x cut % Cut particles into Blocks
    zshift(4*i-3)=zshift(4*i-3)+9999;
end
if zshift(4*i-1)> 650 || zshift(4*i-1)<350 %y cut
    zshift(4*i-1)=zshift(4*i-3)+9999;
end
% if zshift(4*i-3)<1001 && zshift(4*i-1)<1001 && zshift(4*i-3) > 0 && zshift(4*i-1) > 0;
% j=j+1;
% z3(4*j-3)=zshift(4*i-3);  %converting kept particles to new matrix
% z3(4*j-2)=z(4*i-2);
% z3(4*j-1)=zshift(4*i-1);
% z3(4*j-0)=z(4*i-0);
% m3(j)=m(i);
% CSC3(j)=Kc(i);
% c3(j)=c(i);
% d3(j)=d(i);
% TSC3(j)=Ka(i);
% TSL3(j)=Al(i);
% TSF3(j)=Af(i);
% end
% end
% fileID = fopen('Box1b.txt','w'); % Writing Conditions to File
% fprintf(fileID,'%6.2f %12.8f
',z3);
% fprintf(fileID,'%6.2f %12.8f
',m3);
% fprintf(fileID,'%6.2f %12.8f
',CSC3);
% fprintf(fileID,'%6.2f %12.8f
',c3);
% fprintf(fileID,'%6.2f %12.8f
',d3);
% fprintf(fileID,'%6.2f %12.8f
',TSC3);
% fprintf(fileID,'%6.2f %12.8f
',TSL3);
% fprintf(fileID,'%6.2f %12.8f
',TSF3);
% fclose(fileID);
% j=0; % rotate +60 degrees 
% for i=1:np
% if (z(4*i-3)-500)<0
% angle = 180+atand((z(4*i-1)-500)/(z(4*i-3)-500));
% hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
% angle = angle+60;
% zshift(4*i-3)=hypotnus*cosd(angle)+500;
% zshift(4*i-1)=hypotnus*sind(angle)+500;
% else
% angle = atand((z(4*i-1)-500)/(z(4*i-3)-500));
% hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
% angle = angle+60;
% zshift(4*i-3)=hypotnus*cosd(angle)+500;
% zshift(4*i-1)=hypotnus*sind(angle)+500;
% end
%
% if zshift(4*i-3)> 575 || zshift(4*i-3)<425 %x cut % Cut particles into Blocks
% zshift(4*i-3)=zshift(4*i-3)+9999;
% end
% if zshift(4*i-1)> 650 || zshift(4*i-1)<350 %y cut
% zshift(4*i-1)=zshift(4*i-1)+9999;
% end
% if zshift(4*i-3)<1001 && zshift(4*i-1)<1001 && zshift(4*i-3) > 0 &&
  zshift(4*i-1) > 0;
  j=j+1;
% z4(4*j-3)=zshift(4*i-3);  %converting kept particles to new matrix
% z4(4*j-2)=z(4*i-2);
% z4(4*j-1)=zshift(4*i-1);
% z4(4*j-0)=z(4*i-0);  m4(j)=m(i);
% CSC4(j)=Kc(i);
% c4(j)=c(i);
% d4(j)=d(i);
% TSC4(j)=Ka(i);
% TSL4(j)=Al(i);
% TSF4(j)=Af(i);
% end
% end

% fileID = fopen('Box1c.txt','w'); % Writing Conditions to File
% fprintf(fileID,'%6.2f %12.8f
',z4);
% fprintf(fileID,'%6.2f %12.8f
',m4);
% fprintf(fileID,'%6.2f %12.8f
',CSC4);
% fprintf(fileID,'%6.2f %12.8f
',c4);
% fprintf(fileID,'%6.2f %12.8f
',d4);
% fprintf(fileID,'%6.2f %12.8f
',TSC4);
% fprintf(fileID,'%6.2f %12.8f
',TSL4);
% fprintf(fileID,'%6.2f %12.8f
',TSF4);
% fclose(fileID);
%
%    j=0; % rotate +90 degrees
% for i=1:np
%     if (z(4*i-3)-500)<0
%         angle = 180+atand((z(4*i-1)-500)/(z(4*i-3)-500));
%         hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
%         angle = angle+90;
%         zshift(4*i-3)=hypotnus*cosd(angle)+500;
%         zshift(4*i-1)=hypotnus*sind(angle)+500;
%     else
%         angle = atand((z(4*i-1)-500)/(z(4*i-3)-500));
%         hypotnus = ((z(4*i-1)-500)^2+(z(4*i-3)-500)^2)^0.5;
%         angle = angle+90;
%         zshift(4*i-3)=hypotnus*cosd(angle)+500;
%         zshift(4*i-1)=hypotnus*sind(angle)+500;
%     end
%
% if zshift(4*i-3)> 575 || zshift(4*i-3)<425 %x cut % Cut particles into
% Blocks
%     zshift(4*i-3) = zshift(4*i-3)+9999;
% end
% if zshift(4*i-1)> 650 || zshift(4*i-1)<350 %y cut
%    zshift(4*i-1) = zshift(4*i-1)+9999;
% end
%
% if zshift(4*i-3)<1001 && zshift(4*i-1)<1001 && zshift(4*i-3) > 0 && 
  zshift(4*i-1) > 0;
%     j=j+1;
% z5(4*j-3)=zshift(4*i-3); %converting kept particles to new matrix
% z5(4*j-2)=z(4*i-2);
% z5(4*j-1)=zshift(4*i-1);
% z5(4*j-0)=z(4*i-0);  m5(j)=m(i);
% CSC5(j)=Kc(i);
% c5(j)=c(i);
% d5(j)=d(i);
% TSC5(j)=Ka(i);
% TSL5(j)=Al(i);
% TSF5(j)=Af(i);
% end
% end
% fileID = fopen('Box1d.txt','w'); % Writing Conditions to File
% fprintf(fileID,'%6.2f %12.8f
',z5);
% fprintf(fileID,'%6.2f %12.8f
',m5);
% fprintf(fileID,'%6.2f %12.8f
',CSC5);
% fprintf(fileID,'%6.2f %12.8f
',c5);
% fprintf(fileID,'%6.2f %12.8f
',d5);
% fprintf(fileID,'%6.2f %12.8f
',TSC5);
% fprintf(fileID,'%6.2f %12.8f
',TSL5);
% fprintf(fileID,'%6.2f %12.8f
',TSF5);
% fclose(fileID);
%
%     % Write Movie File
%     writerObj = VideoWriter(videoName);
%     writerObj.FrameRate = round(frameRate); % rounds if not an integer already
%     writerObj.Quality = 100; % can be adjusted, but 100 is recommended
%     open(writerObj);
%     writeVideo(writerObj,G);
%     close(writerObj);
%     complete = 'Process complete.';
% end
% function znew = 
integrator(file,z,time,deltaT,m,c,Kc,r,np,ngp,Al,Ka,Af,Cf,nmp1,nmp2,Clist)
% 2nd order runge kutta integrator
% eval(['gl=',file,'(z,time,m,c,Kc,r,np,ngp,Al,Ka,Af,Cf,nmp1,nmp2,Clist);'])
%   deltaz1=deltaT*gl;
% eval(['g2=',file,'(z+deltaz1,time+deltaT,m,c,Kc,r,np,ngp,Al,Ka,Af,Cf,nmp1, 
  nmp2,Clist);'])
%   znew=z+0.5*deltaT*(gl+g2);
% end
%
% function [f] = 
Force_Functions(z,time,m,c,Kc,d,np,ngp,Al,Ka,Af,Cf,nmp1,nmp2,Clist) %force function
%
%      Fx =zeros(1,np);
%      Fy =zeros(1,np);
%
%      for i = 1:np                 %for each particle
%          for jj = 1:Clist(i,1)        %Look at x number of close particles
%          j = Clist(i,jj+1);           %converts first particle on list to global numbering of particle
%          Dis(i,j) = ((z(4*j-3)-z(4*i-3))^2+(z(4*j-1)-z(4*i-1))^2)^0.5; % particle distance magnitude
%          Ddot(i,j) = ((z(4*j-3)-z(4*i-3))*(z(4*j-2)-z(4*i-2))+(z(4*j-1)-z(4*i-1)))/Dis(i,j); %particle velocity magnitude
%          % linear compression region (Compression)
%          bandc = .5*(erf((1/Cf(i)+1/Cf(j))^1*(Dis(i,j)-0))-erf((1/Cf(i)+1/Cf(j))^1*(Dis(i,j)-d(j)/2-d(i)/2)))*Ddot(i,j)./Dis(i,j); % spring force
%          Fc(i,j) = ((1/c(i)+1/c(j))^1*Ddot(i,j)+(1/Kc(i)+1/Kc(j))^1*(Dis(i,j)-d(i)+d(j))/2)*bandc; % spring force
%          % attraction
%          banda = .5*(erf((1/Af(i)+1/Af(j))^1*(Dis(i,j)-d(j)/2-d(i)/2)-d(i)^1.5*d(j)^1.5*(Al(i)+Al(j)-2)/2)) % attraction
%          Fa(i,j) = ((1/c(i)+1/c(j))^1*Ddot(i,j)+(1/Ka(i)+1/Ka(j))^1*(Dis(i,j)-d(i)+d(j))/2)*banda; % attraction
%          Fa(j,i) = -Fa(i,j);
%          end
%      end
%
%      for i = 1:np                 %for each particle (Summing Forces with gravity)
%          for jj = 1:Clist(i,1)        %Look at x number of close particles above
%          j = Clist(i,jj+1);           %converts first particle on list to global numbering of particle
%          Fx(i) = Fx(i)+(Fa(i,j)+Fc(i,j))*(z(4*j-3)-z(4*i-3))/Dis(i,j);
%          Fy(i) = Fy(i)+(Fa(i,j)+Fc(i,j))*(z(4*j-1)-z(4*i-1))/Dis(i,j);
%          Fx(j) = Fx(j)+(Fa(j,i)+Fc(j,i))*(z(4*j-3)-z(4*i-3))/Dis(i,j);
%          Fy(j) = Fy(j)+(Fa(j,i)+Fc(j,i))*(z(4*j-1)-z(4*i-1))/Dis(i,j);
%          end
%      end
% end
Appendix B. Testing Code

```matlab
% function [] = Particle_Code()
% close all, clc, clearvars, clear global

% %Video and Code Configuration
% Trial_Name='T1'; % Name of trial, change each run if you want to keep data saved
% videoLength = 5; % Length for video, in seconds
% frameRate = 20; % Frame rate for video, in frames per second
% videoName = Trial_Name; % Output file name, dont change
%
% %Body Setup
% nmp(5) = 0; % total number of particles, counts up starting at 0
% for k = 1:4 % 4 bodies
% if k == 1
% fileID = fopen('Box1a.txt','r');
% elseif k==2
%    fileID = fopen('Box1b.txt','r');
% elseif k==3
%        fileID = fopen('Box1c.txt','r');
% else
%    fileID = fopen('Box1d.txt','r');
% end
% formatSpec = '%f';
% [data,count] = fscanf(fileID,formatSpec);
% fclose(fileID);
% nmp(k)=(count)/11;
% j=0; % local data position, where i is global
% for i=1+nmp(5)*4:nmp(5)*4+nmp(k)*4
%   j=j+1;
%   z0(i) = data(j); % initial position/velocity
%   if (j+2)/4 == round((j+2)/4) % set velocity to 0
%       z0(i) = 0;
%   end
%   if (j+0)/4 == round((j+0)/4) % set velocity to 0
%       z0(i) = 0;
%   end
%   if (j+3)/4 == round((j+3)/4) % move x to correct position
%       z0(i) = z0(i)-625+250*k;
%   end
% end
% j=0;
% for i = 1+nmp(5):nmp(5)+nmp(k)
%   j=j+1;
%   m(i)=data(j+4*nmp(k)); % mass of each mobile particle
%   Kc(i)=data(j+5*nmp(k)); % dampening of each mobile particle
%   c(i)=data(j+6*nmp(k))/2; % stiffness of each mobile particle
%   d(i)=data(j+7*nmp(k)); % diameter of each mobile particle
%   Ka(i)=data(j+8*nmp(k));
%   Al(i)=data(j+9*nmp(k));
```

% function [] = Particle_Code ()
% close all, clc, clearvars, clear global
% % %Video and Code Configuration
% % Trial_Name='T1'; % Name of trial, change each run if you want to keep data saved
% % videoLength = 5; % Length for video, in seconds
% % frameRate = 20; % Frame rate for video, in frames per second
% % videoName = Trial_Name; % Output file name, dont change
%
% %Body Setup
% % nmp(5) = 0; % total number of particles, counts up starting at 0
% % for k = 1:4 % 4 bodies
% % if k == 1
% % fileID = fopen('Box1a.txt','r');
% % elseif k==2
% %    fileID = fopen('Box1b.txt','r');
% % elseif k==3
% %        fileID = fopen('Box1c.txt','r');
% % else
% %    fileID = fopen('Box1d.txt','r');
% % end
% % formatSpec = '%f';
% % [data,count] = fscanf(fileID,formatSpec);
% % fclose(fileID);
% % nmp(k)=(count)/11;
% % j=0; % local data position, where i is global
% % for i=1+nmp(5)*4:nmp(5)*4+nmp(k)*4
% %   j=j+1;
% %   z0(i) = data(j); % initial position/velocity
% %   if (j+2)/4 == round((j+2)/4) % set velocity to 0
% %       z0(i) = 0;
% %   end
% %   if (j+0)/4 == round((j+0)/4) % set velocity to 0
% %       z0(i) = 0;
% %   end
% %   if (j+3)/4 == round((j+3)/4) % move x to correct position
% %       z0(i) = z0(i)-625+250*k;
% %   end
% % end
% % j=0;
% % for i = 1+nmp(5):nmp(5)+nmp(k)
% %   j=j+1;
% %   m(i)=data(j+4*nmp(k)); % mass of each mobile particle
% %   Kc(i)=data(j+5*nmp(k)); % dampening of each mobile particle
% %   c(i)=data(j+6*nmp(k))/2; % stiffness of each mobile particle
% %   d(i)=data(j+7*nmp(k)); % diameter of each mobile particle
% %   Ka(i)=data(j+8*nmp(k));
% %   Al(i)=data(j+9*nmp(k));
end

Accuracy = 50; %checks each oscillation
deltaT = 2*pi/(max(Kc)/min(m))^0.5/Accuracy; %Step size, in seconds
N = round(videoLength/deltaT); %number of steps, rounded to ensure an integer value
direction = (N*2/3+0.5)*2; %time when walls change direction *2 means no change
update = 10; %how often clist is updated once per second
wait = -4; %how long until wall starts crunching set to 1/3 of time
z = z0; %convert to changing z matrix

nmp(5) = nmp(5)+nmp(k); %total number of particles counting up
percent(k) = m(nmp(5))/m(nmp(5)-nmp(k)+1)*100; %percent smallest to largest

end

percent = percent

ncmp
nmp
nmps(1) = 0; % nmps is nmp but summed together
nmps(2) = nmp(1);
nmps(3) = nmp(2)+nmps(2);
nmps(4) = nmp(3)+nmps(3);
nmps(5) = nmp(4)+nmps(4);

for k = 1:4 %finding max and min pos of each body
  Posx(k,1:nmp(k)) = sort(z(4*nmps(k)+1:4*nmps(k)+1:nmp(k)+1));
  Posy(k,1:nmp(k)) = sort(z(4*nmps(k)+3:4*nmps(k)+1:nmp(k)+1));
  xmax(k) = mean(Posx(k,1:round(nmp(k)/5)));
  xmin(k) = mean(Posx(k,1:round(nmp(k)/5)));
  ymax(k) = mean(Posy(k,1:round(nmp(k)/5)));
  ymin(k) = mean(Posy(k,1:round(nmp(k)/5)));
  ymint(k) = min(Posy(k,1:round(nmp(k)/5)));
  W_max0(k) = xmax(k)-xmin(k);
  W_max0t(k) = xmax(k)-xmin(k);
  L_max0(k) = ymax(k)-ymin(k);
  L_max0t(k) = ymax(k)-ymin(k);
end

%Wall Setup
cw = c(1); %temperature absorption
Kcw = Kc(1); %Compressive Spring Constant (Slope)
ytw =
wait+max([yamt(1)+d(1),yamt(2)+d(1),yamt(3)+d(1),yamt(4)+d(1)]);
ybw = -wait+min([yint(1)-d(1),yint(2)-d(1),yint(3)-d(1),yint(4)-d(1)]);
% Characteristic Variables
% do = d(nmp(5)); %smallest diamater
% mo = m(nmp(5)); %smallest mass
% Kc0 = Kc(1)/10^5; %Largest Kc
% wo = (Kc0/mo)^.5;

k = 2; %lst element in list For mobile particles, optimization
moveD = d(1); %expected move distance
for i = 1:nmp(5) %for each mobile particle
  Crange(i) = d(1)*(Al(1)-1)+d(1)+moveD; %Max attraction range for each particle + modeD = cutoff range
  for j = i+1:nmp(5) %Look at each other particle above
    Dis(i,j) = ((z0(4*j-3)-z0(4*i-3))^2+(z0(4*j-1)-z0(4*i-1))^2)^0.5; % particle distance magnitude
    if Dis(i,j) < Crange(i)
      Clist(i,k) = j; %for each particle, a list of particles that are close
      k = k+1;
    end
  end
  Clist(i,1) = k-2;
end

for l = 1:4 %for each block
  j=1;
  k=1;
  for i = nmps(l)+1:nmps(l+1) %for each mobile particle in given block
    if z0(4*i-1)+d(i)/2 > ytw-25 %check y pos for walls
      j = j+1;
      Clisttw(j,l) = i;
    elseif z0(4*i-1)-d(i)/2 < ybw+25
      k = k+1;
      Clistbw(k,l) = i;
    end
  end
  Clisttw(1,l) = j;
  Clistbw(1,l) = k;
end
theta = linspace(0,2*pi,12);

tic
MVelx = zeros(1,4);
MVely = zeros(1,4);
MStress = zeros(1,4);
MTStrainL = zeros(1,4);
MStrainL = zeros(1,4);
MTStrainW = zeros(1,4);
MStrainW = zeros(1,4);
MTV = zeros(1,4);
MV = zeros(1,4);
index = 0; %indexing term, used in the frame-grab calculation (leave at 0)
fig = figure(1);
tstart = [0,0,0,0];
for i = 1:N % i is the moment in time
time = (i-1)*deltaT;
time2(i) = time;

[z,Fg] = integrator('Force_Functions',z,time,deltaT,m,c,Kc,d,Al,Ka,Af,nmp,Clist,Clisttw,Clistbw,ytw,ybw,cw,Kcw); %call integrator

for k = 1:4 %pull out position from z matrix and find min and max
    Velx(k,i) = mean(abs(z(4*nmps(k)+2:4:4*nmps(k+1)))); %find engineering values for plots
    Vely(k,i) = mean(abs(z(4*nmps(k)+4:4:4*nmps(k+1))));
    Posx(k,1:nmp(k)) = sort(z(4*nmps(k)+1:4:4*nmps(k+1)));
    Posy(k,1:nmp(k)) = sort(z(4*nmps(k)+3:4:4*nmps(k+1)));
    xmax(k) = mean(Posx(k,nmp(k)-round(nmp(k)/5):nmp(k)));
    xmin(k) = mean(Posx(k,1:round(nmp(k)/5)));
    ymax(k) = mean(Posy(k,nmp(k)-round(nmp(k)/5):nmp(k)));
    ymin(k) = mean(Posy(k,1:round(nmp(k)/5)));
    W_max(k) = xmax(k) - xmin(k);
    L_max(k) = ymax(k) - ymin(k);
    Stress(k,i) = Fg(k)/(W_max(k)*2);
    StrainL(k,i) = ((L_max0(k)-L_max(k))/L_max0(k));
    TStrainL(k,i) = ((L_max0t(k)-L_max(k))/L_max0t(k));
    TStrainW(k,i) = -(W_max0(k)-W_max(k))/W_max0(k); % TV(k,i) = TStrainW(k,i)/TStrainL(k,i)+.000000001;
    V(k,i) = StrainW(k,i)/StrainL(k,i);
else
    StrainW(k,i) = TStrainW(k,i);
    V(k,i) = TV(k,i);
end

VelxA(i) = mean(Velx(:,i)); %average velocity over each block
VelyA(i) = mean(Vely(:,i));
StressA(i) = mean(Stress(:,i));
TStrainLA(i) = mean(TStrainL(:,i));
StrainLA(i) = mean(StrainL(:,i));
TStrainWA(i) = mean(TStrainW(:,i));
StrainWA(i) = mean(StrainW(:,i));
TVA(i) = mean(TV(:,i));
VA(i) = mean(real(V(:,i)));
for k=1:4 %find mean engineering values for plots
MVelx(k) = MVelx(k)+real((Velx(k,i)-VelxA(i))/((Velx(k,i)+VelxA(i)+.0000001)/2));
Mvely(k) = Mvely(k)+real((Vely(k,i)-VelyA(i))/((Vely(k,i)+VelyA(i)+.0000001)/2));
MStress(k) = MStress(k)+real((Stress(k,i)-StressA(i))/((Stress(k,i)+StressA(i)+.0000001)/2));
MTStrainL(k) = MTStrainL(k)+real((TStrainL(k,i)-TStrainLA(i))/((TStrainL(k,i)+TStrainLA(i)+.0000001)/2));
MTStrainW(k) = MTStrainW(k)+real((TStrainW(k,i)-TStrainWA(i))/((TStrainW(k,i)+TStrainWA(i)+.0000001)/2));
MTV(k) = MTV(k)+real((TV(k,i)-TVA(i))/((TV(k,i)+TVA(i)+.0000001)/2));
MV(k) = MV(k)+real((V(k,i)-VA(i))/((V(k,i)+VA(i)+.0000001)/2));
end
direction = direction-1; %moving the walls
ytw = ytw-10/N*direction/abs(direction)*1;        % y ground down
ybw = ybw+10/N*direction/abs(direction)*1;        % y ground up
if mod(i, update) == 0 %update clist %
z0=z;
clearvars Clist;
kp = 2; %1st element in list
for kk=1:4 %for each box
for ii = nmps(kk)+1:nmps(kk+1) %for each particle in box
for j = ii+1:nmps(kk+1) %Look at each other particle in box above
Dis(ii,j) = ((z0(4*j-3)-z0(4*ii-3))^2+(z0(4*j-1)-z0(4*ii-1))^2)^0.5; % particle distance magnitude
if Dis(ii,j) < Crange(ii)
    Clist(ii,k) = j; %for each particle, a list of particles that are close
    k = k+1;
end
end
end
Clist(ii,1) = k-2;
k=2;
end
end

rnd = round((1/deltaT)/frameRate); % rounded to ensure an integer value MOVIE!
isMultiple = (rnd*round(double(i)/rnd) == 1 );
if isMultiple == 1  % grab every 'r'th frame after the first
    index = index+1;  % indexing update
    G(index) = getframe(fig);  % 'fig' is the variable name of the figure
    for k = 1:4
        for j = nmps(k)+1:nmps(k+1) %plot bodies 1-4
            xcir = d(j)/2*cos(theta) + z(4*j-3);
            ycir = d(j)/2*sin(theta) + z(4*j-1);
            plot(xcir/d(nmp(5)),ycir/d(nmp(5)),'Color',[0,0,0]);
            hold on
        end
    end
    Arange = d(round(j/2))^.5*d(round(j/2))^.5*Al(round(j/2))+d(round(j/2));  % attraction range
    xcir=cos(theta)*Arange+ z0(4*round(j/2)-3);
    ycir=sin(theta)*Arange+ z0(4*round(j/2)-1);
    plot(xcir/d(nmp(5)),ycir/d(nmp(5)),'Color',[1,0,0]);
    hold on
    plotxw(1)= 20;  %plot walls
    plotxw(2) = 980;
    plotytw(1) = ytw;
    plotytw(2) = ytw;
    plotybw(1) = ybw;
    plotybw(2) = ybw;
    plot(plotxw/d(nmp(5)),plotytw/d(nmp(5)),'Color',[0,0,0]);
    plot(plotxw/d(nmp(5)),plotybw/d(nmp(5)),'Color',[0,0,0]);
end
end

%
plotstart = min(tstart*wo);
figure(2);
plot(StrainLA(:,),StressA(:,)/Kc0,'k')
hold on
for k = 1:4 %stress strain plot
plot(StrainL(k,:),Stress(k,:)/Kc0,'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Strain in Length') % x-axis label
ylabel('Stress(Kc0)') % y-axis label
axis([0,inf,0,inf]);
end
figure(3);
plot(TStrainLA(:,),StressA(:,)/Kc0,'k')
hold on
for k = 1:4 %stress strain plot true
plot(TStrainL(k,:),Stress(k,:)/Kc0,'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('True Strain in Length') % x-axis label
ylabel('Stress(Kc0)') % y-axis label
axis([-inf,inf,0,inf]);
end
figure(4);
plot(time2*wo,VA(:,),'k')
hold on
for k = 1:4 %possession ratio plot
plot(time2*wo,V(k,:),,'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('Poisson Ratio') % y-axis label
axis([plotstart,inf,-2,2]);
end
figure(5);
plot(time2*wo,TVA(:,),'k')
hold on
for k = 1:4 %possession ratio plot true
plot(time2*wo,TV(k,:),,'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('TPoisson Ratio') % y-axis label
axis([0,inf,-2,2]);
end
figure(6);
plot(time2*wo,StrainLA(:,),'k')
hold on
for k = 1:4 %StrainL time plot
plot(time2*wo,StrainL(k,:),,'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('StrainL') % y-axis label
axis([plotstart,inf,0,inf]);
end
figure(7);
plot(time2*wo,TStrainLA(:),'k')
hold on
for k = 1:4 %true StrainL total time plot
plot(time2*wo,TStrainL(k,:),'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('Non-Ajusted StrainL') % y-axis label
axis([0,inf,-inf,inf]);
end
figure(8);
plot(time2*wo,StrainWA(:),'k')
hold on
for k = 1:4 %strainW plot
plot(time2*wo,StrainW(k,:),'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('StrainW') % y-axis label
axis([plotstart,inf,-inf,inf]);
end
figure(9);
plot(time2*wo,TStrainWA(:),'k')
hold on
for k = 1:4 %TstrainW plot
plot(time2*wo,TStrainW(k,:),'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('TStrainW') % y-axis label
axis([0,inf,-inf,inf]);
end
figure(10);
plot(time2*wo,StressA(:)/Kc0,'Color','k')
hold on
for k = 1:4 %Stress L plot
plot(time2*wo,Stress(k,:)/Kc0,'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('Stress(Kc0)') % y-axis label
axis([0,inf,-inf,inf]);
end
figure(11);
plot(time2*wo,(VelxA(:)+VelyA(:))/(VelxA(1)+VelyA(1)),'k')
hold on
for k = 1:4 %Stress L plot
plot(time2*wo,(Velx(k,:)+Vely(k,:))/(Velx(k,1)+Vely(k,1)),'Color',[k/3.1-1/3.2,1-k/4,.9])
hold on
xlabel('Time(cycles)') % x-axis label
ylabel('Velocity Magnitude') % y-axis label
axis([0,inf,-inf,inf]);
end
% MVelx = (MVelx/N)
% MVely = (MVely/N)
% MStress = (MStress/N)
% MTStrainL = (MTStrainL/N)
% MStrainL = (MStrainL/N)
% MTStrainW = (MTStrainW/N)
% MStrainW = (MStrainW/N)
% MTV = (MTV/N)
% MV = (MV/N)
% MeanDifference = [MVelx, MVely, MStress, MTStrainL, MStrainL, MTStrainW, MStrainW, MTV, MV]
%
% j=0; % body 1
% for i=1:nmp(5)
% if z(4*i-3)<250 && z(4*i-1)<1001 && z(4*i-3) > 0 && z(4*i-1) > 0;
%     j=j+1;
%     z2(4*j-3)=z(4*i-3); %converting kept particles to new matrix
%     z2(4*j-2)=z(4*i-2);
%     z2(4*j-1)=z(4*i-1);
%     z2(4*j-0)=z(4*i-0);
% m2(j)=m(i);
% CSC2(j)=Kc(i);
% c2(j)=c(i);
% d2(j)=d(i);
% TSC2(j)=Ka(i);
% TSL2(j)=A1(i);
% TSF2(j)=Af(i);
% end
% end
% fileID = fopen('Box2a3.txt','w'); % Writing Conditions to File
% fprintf(fileID,'%6.2f %12.8f
',z2);
% fprintf(fileID,'%6.2f %12.8f
',m2);
% fprintf(fileID,'%6.2f %12.8f
',CSC2);
% fprintf(fileID,'%6.2f %12.8f
',c2);
% fprintf(fileID,'%6.2f %12.8f
',d2);
% fprintf(fileID,'%6.2f %12.8f
',TSC2);
% fprintf(fileID,'%6.2f %12.8f
',TSL2);
% fprintf(fileID,'%6.2f %12.8f
',TSF2);
% fclose(fileID);
%           j=0; % body 2
% for i=1:nmp(5)
% if z(4*i-3)<531.25 && z(4*i-1)<1001 && z(4*i-3) > 250 && z(4*i-1) > 0;
%     j=j+1;
%     z3(4*j-3)=z(4*i-3); %converting kept particles to new matrix
%     z3(4*j-2)=z(4*i-2);
%     z3(4*j-1)=z(4*i-1);
%     z3(4*j-0)=z(4*i-0);
% m3(j)=m(i);
% CSC3(j)=Kc(i);
% c3(j)=c(i);
% d3(j)=d(i);
38

% TSC3(j)=Ka(i);
% TSL3(j)=Al(i);
% TSF3(j)=Af(i);
% end
% end
% fileID = fopen('Box2b3.txt','w'); % Writing Conditions to File
% fprintf(fileID,'%6.2f %12.8f
',z3);
% fprintf(fileID,'%6.2f %12.8f
',m3);
% fprintf(fileID,'%6.2f %12.8f
',CSC3);
% fprintf(fileID,'%6.2f %12.8f
',c3);
% fprintf(fileID,'%6.2f %12.8f
',d3);
% fprintf(fileID,'%6.2f %12.8f
',TSC3);
% fprintf(fileID,'%6.2f %12.8f
',TSL3);
% fprintf(fileID,'%6.2f %12.8f
',TSF3);
% fclose(fileID);
%         j=0; % body 3
% for i=1:nmp(5)
% if z(4*i-3)<718.75 && z(4*i-1)<1001 && z(4*i-3) > 531.25 && z(4*i-1) > 0;
%     j=j+1;
% z4(4*j-3)=z(4*i-3);  %converting kept particles to new matrix
% z4(4*j-2)=z(4*i-2);
% z4(4*j-1)=z(4*i-1);
% z4(4*j-0)=z(4*i-0);
% m4(j)=m(i);
% CSC4(j)=Kc(i);
% c4(j)=c(i);
% d4(j)=d(i);
% TSC4(j)=Ka(i);
% TSL4(j)=Al(i);
% TSF4(j)=Af(i);
% end
% end
% fileID = fopen('Box2c3.txt','w'); % Writing Conditions to File
% fprintf(fileID,'%6.2f %12.8f
',z4);
% fprintf(fileID,'%6.2f %12.8f
',m4);
% fprintf(fileID,'%6.2f %12.8f
',CSC4);
% fprintf(fileID,'%6.2f %12.8f
',c4);
% fprintf(fileID,'%6.2f %12.8f
',d4);
% fprintf(fileID,'%6.2f %12.8f
',TSC4);
% fprintf(fileID,'%6.2f %12.8f
',TSL4);
% fprintf(fileID,'%6.2f %12.8f
',TSF4);
% fclose(fileID);
%         j=0; % body 4
% for i=1:nmp(5)
% if z(4*i-3)<1001 && z(4*i-1)<1001 && z(4*i-3) > 718.75 && z(4*i-1) > 0;
%     j=j+1;
% z5(4*j-3)=z(4*i-3);  %converting kept particles to new matrix
% z5(4*j-2)=z(4*i-2);
% z5(4*j-1)=z(4*i-1);
% z5(4*j-0)=z(4*i-0);
% m5(j)=m(i);
% CSC5(j)=Kc(i);% c5(j)=c(i);% d5(j)=d(i);% TSC5(j)=Ka(i);% TSL5(j)=A1(i);% TSF5(j)=Af(i);% end% end% fileID = fopen('Box2d3.txt','w'); % Writing Conditions to File% fprintf(fileID,'%6.2f %12.8f
',z5);% fprintf(fileID,'%6.2f %12.8f
',m5);% fprintf(fileID,'%6.2f %12.8f
',CSC5);% fprintf(fileID,'%6.2f %12.8f
',c5);% fprintf(fileID,'%6.2f %12.8f
',d5);% fprintf(fileID,'%6.2f %12.8f
',TSC5);% fprintf(fileID,'%6.2f %12.8f
',TSL5);% fprintf(fileID,'%6.2f %12.8f
',TSF5);% fclose(fileID);% % %     % Write Movie File%     writerObj = VideoWriter(videoName);%     writerObj.FrameRate = round(frameRate); % rounds if not an integer already%     writerObj.Quality = 100; % can be adjusted, but 100 is recommended%     open(writerObj);%     writeVideo(writerObj,G);%     close(writerObj);%     toc% end% % function [znew,Fg] = integrator(file,z,time,deltaT,m,c,Kc,r,Al,Ka,Af,Cf,nmp,Clist,Clisttw,Clistbw,ytw,ybw,cw,Kcw) %integrator% [g1,Fgy] = feval(file,z,time,m,c,Kc,d,Al,Ka,Af,Cf,nmp,Clist,Clisttw,Clistbw,ytw,ybw,cw,Kcw);% delaiz1 = deltaT*gl;% % [g2,Fgy2] = feval(file,z+delaiz1,time+deltaT,m,c,Kc,r,Al,Ka,Af,Cf,nmp,Clist,Clisttw,Clistbw,ytw,ybw,cw,Kcw);% znew = z+0.5*deltaT*(gl+g2);% Fg = (Fgy+Fgy2)/2;% % end% % function [f,Fgy] = Force_Functions(z,time,m,c,Kc,d,Al,Ka,Af,Cf,nmp,Clist,Clisttw,Clistbw,ytw, ybw,cw,Kcw) %force function% Fx = zeros(1,nmp(5));
%     Fy = zeros(1,nmp(5));
%     Fw = zeros(1,nmp(5));
%     Fgy = zeros(1,4);  %zero force on wall
%     for i = 1:nmp(5) %for each mobile particle
%     for jj = 1:Clist(i,1)  %Look at x number of close
%     j = Clist(i,jj+1);  %converts first particle on list to global
particles
%     Dis(i,j) = ((z(4*j-3)-z(4*i-3))^2+(z(4*j-1)-z(4*i-1))^2)^0.5;  %
particle distance magnitude
%     Ddot(i,j) = ((z(4*j-3)-z(4*i-3))*(z(4*j-2)-z(4*i-2))+(z(4*j-1)-
%     z(4*i-1))*(z(4*j)-z(4*i)))/Dis(i,j);  %particle velocity magnitude
%     bandc = .5*(erf((1/Cf(i)+1/Cf(j))^-
%     1*(Dis(i,j)-0))-erf((1/Cf(i)+1/Cf(j))^-
%     1*(Dis(i,j)-d(j)/2-d(i)/2)));  % linear compression
%     Fc(i,j) = ((1/c(i)+1/c(j))^-
%     1*Ddot(i,j)+(1/Kc(i)+1/Kc(j))^-
%     1*(Dis(i,j)-(d(i)+d(j))/2))*bandc;  % spring force
%     Fc(j,i) = -Fc(i,j);
%     banda = .5*(erf((1/Af(i)+1/Af(j))^-
%     1*(Dis(i,j)-d(j)/2-d(i)/2)-
%     erf((1/Af(i)+1/Af(j))^-
%     1*(Dis(i,j)-d(j)/2-d(i)/2)-
%     d(i)^.5*d(j)^.5*(Al(i)+Al(j)-2)/2)));  % attraction
%     Fa(i,j) = ((1/c(i)+1/c(j))^-
%     1*Ddot(i,j)+(1/Ka(i)+1/Ka(j))^-
%     1*(Dis(i,j)-(d(i)+d(j))/2))*banda;  % attraction
%     Fa(j,i) = -Fa(i,j);
%     end
%     end
%     for k = 1:4
%     for i = 2:Clisttw(1,k)  %for top wall
%     Disw = abs(z(4*Clisttw(i,k)-1)-ytw);  % particle distance
magnitude from center to wall
%     if Disw < d(Clisttw(i,k))/2;
%     Ddotw = z(4*Clisttw(i,k));  %particle velocity magnitude
%     Fw(Clisttw(i,k)) = (1/c(Clisttw(i,k))+(1/cw)^-
%     1*Ddotw-(1/Kc(Clisttw(i,k))+(1/Kcw)^-
%     1*(d(Clisttw(i,k))/2-Disw));  % spring force
%     Fgy(k) = Fgy(k)+abs(Fw(Clisttw(i,k)));%end
%     end
%     end
%     end
%     for k = 1:4
%     for i = 2:Clistbw(1,k)  %for bottom wall
%     Disw = abs(z(4*Clistbw(i,k)-1)-ybw);  % particle distance
magnitude
%     if Disw < d(Clistbw(i,k))/2
%     Ddotw = z(4*Clistbw(i,k));  %particle velocity magnitude
%     Fw(Clistbw(i,k)) = -(1/c(Clistbw(i,k))+(1/cw)^-
%     1*Ddotw+(1/Kc(Clistbw(i,k))+(1/Kcw)^-
%     1*(d(Clistbw(i,k))/2-Disw));  % spring force
%     Fgy(k) = Fgy(k)+abs(Fw(Clistbw(i,k)));
for i = 1:nmp(5) %for each mobile particle (Summing Forces with
gravity)
    for jj = 1:Clist(i,1) %Look at x number of close
        j = Clist(i,jj+1); %converts first particle on list to global
        numbering of particle
        Fx(i) = Fx(i)+(Fa(i,j)+Fc(i,j))*(z(4*j-3)-z(4*i-3))/Dis(i,j);
        Fx(j) = Fx(j)+(Fa(j,i)+Fc(j,i))*(z(4*j-3)-z(4*i-3))/Dis(j,i);
        Fy(i) = Fy(i)+(Fa(i,j)+Fc(i,j))*(z(4*j-1)-z(4*i-1))/Dis(i,j);
        Fy(j) = Fy(j)+(Fa(j,i)+Fc(j,i))*(z(4*j-1)-z(4*i-1))/Dis(j,i);
    end
    o = 4*i-3;
    f(o)=z(4*i-2);
    f(o+1)=Fx(i)/m(i);
    f(o+2)=z(4*i);
    f(o+3)=(Fy(i)+Fw(i))/m(i);
end