Simulation of random packing of hard spheres using Monte Carlo method

Sung-Ho Park
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/theses
Part of the Mechanical Engineering Commons

Recommended Citation
https://digitalcommons.njit.edu/theses/1336

This Thesis is brought to you for free and open access by the Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.
Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement.

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation.

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen.
The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.
SIMULATION OF RANDOM PACKING OF HARD SPHERES USING MONTE CARLO METHOD

by

Sung-Ho Park

Thesis submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

1990
APPROVAL SHEET

TITLE OF THESIS: Simulation of Random Packing of Hard Spheres using Monte Carlo Method

NAME OF CANDIDATE: Sung-Ho Park
Master of Science in Mechanical Engineering

THESIS AND ABSTRACT APPROVED: ___________________ DATE: __________

Anthony D. Rosato
Assistant Professor
Mechanical Engineering Department

SIGNATURE OF OTHER MEMBERS OF THESIS COMMITTEE

_________________________ DATE: __________

_________________________ DATE: __________
VITA

Name : Sung-Ho Park

Permanent Address :

Degree and date to be conferred : MSME, 1990

Date of Birth : .

Place of Birth :

Secondary Education : Kyung-gi high school, Seoul, 1975-1977

<table>
<thead>
<tr>
<th>Collegiate institutions attended</th>
<th>Dates</th>
<th>Degree</th>
<th>Date of Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Korea University</td>
<td>1978-82</td>
<td>BSME</td>
<td>1982</td>
</tr>
<tr>
<td>New Jersey Institute of Technology</td>
<td>1987-90</td>
<td>MSME</td>
<td>1990</td>
</tr>
</tbody>
</table>

Major : Mechanical Engineering
ABSTRACT

Title of thesis: Simulation of random packing of hard spheres using Monte Carlo method

Sung-Ho Park, Master of Science in Mechanical Engineering, 1990
Thesis directed by: Dr. Anthony D. Rosato, Assistant Professor
Mechanical Engineering Department.

A computer based method of generating a random packing of hard spheres is described. Using a Monte Carlo method as employed in the field of Computational Statistical Physics, packing of hard spheres are generated and analyzed.

The mean packing fractions for the present assemblies of 1000 spheres are 0.555±0.015 after pouring and 0.582±0.018 after 10 cycles of shaking. These values are approximately 5 to 6 percent lower than the experimental results of G.D.Scott[30], but similar with the result of Visscher & Bolsterli[17].

The mean coordination numbers are 5.97 and 6.33 for the pouring and shaking case, respectively. The radial distribution function was calculated and compared with other published data. The simulated results are similar with those of G.D.Scott.

The pouring simulations with 5 different system sizes verified that the resulting low packing density is independent of the number of particles in the system.

In an attempt to determine the reasons for the 5 to 6 percent difference between existing experimental data of G.D.Scott and the simulation results, two computations were done.

The first case study measured the total void volume formed by
the gaps of the neighboring spheres. It was found that the void volume occupied approximately 0.0017 per cent of the total volume. Therefore the use of the corrected diameter cannot be a factor.

The second series of computations studied the effects of allowing the system to rapidly "cool" to an equilibrated state as opposed to incrementally reducing $T^*$ from a value of 15.8 to 0.00211, whereby the system is allowed to equilibrium at each incremental step. The result shows that the packing density increased from 0.565 to 0.617. This can account for the 5 to 6 per cent difference between the experimental result of G.D. Scott and the result of current simulation.
I am extremely grateful to my advisor Dr. Anthony D. Rosato for his considerable assistance and helpful discussions. This thesis would not have been completed without his help.
TO DEAR MY WIFE
APPENDIX A: ALGORITHMS

A.1 Monte Carlo Simulation ........................................ 55
  (a) Initial Configuration ..................................... 55
  (b) Pouring Simulation ....................................... 56
  (c) Shaking Simulation ....................................... 57
A.2 Coordination Number .......................................... 58
A.3 Packing Fraction ................................................ 59
  (a) Spherical Growth method ................................ 59
  (b) Plane Growth method ................................... 60
A.4 Radial Distribution Function ................................ 61
A.5 Calculating the Nearest Neighbor Distances ................. 62

APPENDIX B: CODE LISTINGS ........................................ 63

B.1 Monte Carlo Simulation ........................................ 63
B.2 Coordination Number .......................................... 112
B.3 Packing Fraction using Spherical Growth Method .......... 116
B.4 Packing Fraction using Plane Growth Method .............. 119
B.5 Radial Distribution Function ................................ 121
B.6 Cumulative Probability of the Normalized Nearest Neighbor
distance r/dia. ...................................................... 125
  (a) Calculating the Nearest Neighbor Distances .......... 125
  (b) Sorting the Neighbor Distances ......................... 128
  (c) Cumulative Probability of the Normalized Distances 130

APPENDIX C: OUTPUT OF THE SIMULATION ......................... 133

C.1 Coordination Number of Pouring Simulation ................. 133
C.2 Coordination Number of Shaking Simulation ................ 133
C.3 Packing Fraction by Spherical Growth Method after
  Pouring .......................................................... 134
C.4 Packing Fraction by Spherical Growth Method after
Shaking

C.5 Packing Fraction by Plane Growth Method after Pouring - 136
C.6 Packing Fraction by Plane Growth Method after Shaking - 137
C.7 Radial Distribution after Pouring - 138
C.8 Radial Distribution after Shaking - 139
C.9 Cumulative Probability of the Normalized Nearest Neighbour Distance $r/d_{ia.}$ after Pouring - 140
C.10 Cumulative Probability of the Normalized Nearest Neighbour Distance $r/d_{ia.}$ after Shaking - 142
LIST OF FIGURES

1.1 Types of Layers 8

3.1 Coordinate System and Periodic Boundary Condition 11

3.2 Periodic Boundary Condition in each case 12
(a) P.B.C. on the sides 12
(b) P.B.C. in the corner 12

3.3 Possible Locations of Sphere in X disrection 13

4.1 Coordination Number at 1.01, 1.05 and 1.1 diameter separation after pouring 17
(a) 1.1 diameter separation 17
(b) 1.05 diameter separation 18
(c) 1.1 diameter separation 18
(d) Comparison of the results 19

4.2 Comparison of the published data and Simulated results 20

4.3 Coordination Number at 1.01, 1.05 and 1.1 diameter separation after shaking 21
(a) 1.1 diameter separation 21
(b) 1.05 diameter separation 21
(c) 1.01 diameter separation 21
(d) Comparison of the results 22

4.4 Comparison of the results between Bernal & Mason's experiment and shaking simulation 23

4.5 The basic algorithm of Spherical Growth Method 24

4.6 Packing Fractions by Spherical Growth Method 26
(a) After Pouring Simulation 26
(b) After 10 cycles of Shaking 26
(c) Comparison of the results between Pouring and Shaking 27
4.7 The volume of a spherical segment

4.8 Packing Fractions by Plane Growth Method
   a After Pouring
   b After 10 cycles of Shaking
   c Comparison of the Results

4.9 Radial Distribution Function
   a Pouring case
   b Shaking case
   c Comparison with the published data

4.10 Packing Densities in each case
   a Packing Density by the number of passes in each case
   b 64 sphere case
   c 125 sphere case
   d 216 sphere case
   e 343 sphere case

4.11 Cumulative Probability Distribution versus the Normalized Distance
   a After Pouring (415,000 passes)
   b After 12 cycles of Shaking

4.12 Distribution of the Coordination Numbers
   a After Pouring
   b After 12 cycles of Shaking
LIST OF TABLES

1.1 The Summarized Previous Works on Random Packing 4
1.2 Data Comparison of Experimental & Computer Simulation 6
3.1 Initial Input Data for the Pouring Simulation 15
3.2 Input Data for the Shaking Simulation 15
4.1 Comparison of the Packing Fractions 29
4.2 r/dia. at Positions of Peaks 33
4.3 Normalized Configuration Energy 36
4.4 The Packing Densities according to $T^*$ 46
1. INTRODUCTION

1.1 Survey of Previous Research

Random packing of hard spheres have been extensively studied, due to their importance as models for particulate systems in a wide variety of fields such as physics, chemistry, biology and engineering.

The methods used to investigate the sphere packings are broadly classified into two groups, i.e. mechanical packings and computer simulations.

G.D. Scott [3] carried out his experiments with 1/8 inch diameter steel balls and obtained two well-defined limits which he called "dense random packing" and "loose random packing". For the dense random packing, the balls in the container were gently shaken down for 2 minutes. For the loose packing the balls filled the container essentially by rolling down a slope of randomly-packed balls. The values for the limiting packing densities were 0.637 for dense random packing and 0.601 for loose random packing. The variation of measurements of the two limits were within ±0.2 per cent.

H. Susskind and W. Becker [34] packed rubber ballons with 0.118 inch diameter glass balls and 0.125 inch diameter steel balls. The beds were packed by dropping balls randomly into the rubber ballons and evacuated the air from the ballons, but in several cases the beds were vibrated for 45 minutes on a shaker before evacuating the air. The average densities of the loosely packed beds were 0.638±0.01 and 0.635±0.01 for the glass and stainless
steel beds, respectively. The average density of the densely packed glass beds was 0.652±0.01.

R.K. McGeary [6] found that a maximum packing density could be obtained when the container diameter was more than about ten times the sphere diameter.

G. Mason [4,31] simulated the random packing of equal spheres on a computer, and found a limiting density of 0.63 to 0.64, close to the experimentally determined value by G.D. Scott [3]. The methods used by Mason essentially assumed a central confining force on the sphere, thereby avoiding effects due to gravity.

D.J. Adams and A.J. Matheson [14] generated a random close packing of hard spheres via a computer simulation. Their method placed a new sphere at the tetrahedral site nearest to the center of packing, thus producing a spherical model. The resulting packing density was 0.628. The fluctuation in the measured packing density was not specified.

C.H. Bennett [12] constructed packings of several thousand equal hard spheres by depositing each sphere, one at a time, at surface sites on a small seed cluster, placing each new sphere in contact with three already presented ones. This yielded the mean packing density of 0.61. The limiting values were bounded from 0.57 to 0.63. Bennett and Matheson's techniques are basically the same, but the choice of sites of which to place the new sphere was different as described above.

Further, W.M. Visscher and M. Bolsterli [15] approached the problem of random packing of spheres by means of a Monte Carlo computer simulation of the physical process of dropping spheres into a bin and found a density of 0.582. E.M. Tory et al.
[10,16,19,32] simulated the very slow settling of spheres from a dilute suspension into a randomly packed bed. To avoid the wall effects, the packing density was measured on the interior 5000 spheres of an assembly of 10,000 monosized spheres. An overall mean packing density of 0.58 was found.

A.J. Matheson [17] generated a homogeneous assembly of randomly closed packed spheres of packing density 0.606 ± 0.006. He used a spherical growth method which involved the selection, from among the large list of available tetrahedral sites, of that one site which is nearest to the origin of the pile of existing spheres.

W.S. Jodrey and E.M. Tory [21] generated 3000 spheres in a cubic container by a relaxation method. The relaxation method eliminated the largest overlap at each step and gradually converged to an overlap-free packing. Their packing achieved density of 0.6366 and coordination number of 5.64.

J. Rodriguez et al. [22] developed an assembly of packing under gravity, particle by particle. A new particle at a randomly chosen position above the already placed particles was dropped and allowed to roll down until it reached a stable position. The resulting packing density was 0.58 ± 0.05. The summarized survey is presented in Table 1.1.;
<table>
<thead>
<tr>
<th>Methods</th>
<th>Procedures</th>
<th>References</th>
</tr>
</thead>
</table>

Table 1.1. The summarized previous works on random packing
1.2. Comparison of Experimental and Computer Simulated Results

Computer simulations of random packings are highly dependent on the assumptions made in the generating algorithm. In experiments, observed results also had a high dependence on the experimental procedures.

The summary described above indicates that the upper limit values of experimental and computer simulated packing densities are 0.637±0.001 and 0.6366±0004, respectively. The lower limiting densities are 0.60 and 0.58, respectively. The coordination numbers ranged from 5.45 to 6.4 at close sphere contacts in experiments. In the case of computer simulation, the coordination numbers ranged from 6.0 to 6.1 at close contacts.

Table 1.2 summarizes the results of experimental and computer simulated random packings.

<table>
<thead>
<tr>
<th>Mean coordination number</th>
<th>Packing density</th>
<th>System</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.601±0.001</td>
<td>Steel balls in a cylinder</td>
<td>Scott [3]</td>
</tr>
<tr>
<td></td>
<td>0.637±0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.625</td>
<td>Steel balls in a glass container</td>
<td>McGeary [6]</td>
</tr>
<tr>
<td></td>
<td>0.6366±0.0004</td>
<td>Steel balls in a cylinder</td>
<td>Finney [13]</td>
</tr>
<tr>
<td>6.1</td>
<td>0.59</td>
<td>Computer Simulation</td>
<td>Tory, Cochrane &amp; Waddell [10]</td>
</tr>
</tbody>
</table>
### Table 1.2 Data comparison of experimental & computer simulation

<table>
<thead>
<tr>
<th>Value</th>
<th>Simulation</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.628</td>
<td>Computer Simulation</td>
<td>Adams &amp; Matheson [14]</td>
</tr>
<tr>
<td>0.61</td>
<td>Computer Simulation</td>
<td>Bennett [12]</td>
</tr>
<tr>
<td>0.582</td>
<td>Computer Simulation</td>
<td>Visscher &amp; Bolster [15]</td>
</tr>
<tr>
<td>0.58</td>
<td>Computer Simulation</td>
<td>Tory, Church, Tam &amp; Ratner [16]</td>
</tr>
<tr>
<td>0.606±0.006</td>
<td>Computer Simulation</td>
<td>Matheson [17]</td>
</tr>
<tr>
<td>0.6099</td>
<td>Statistical Method</td>
<td>Gotoh &amp; Finney [18]</td>
</tr>
<tr>
<td>0.6472</td>
<td>Statistical Method</td>
<td>Gotoh &amp; Finney [18]</td>
</tr>
<tr>
<td>0.59±0.01</td>
<td>Computer Simulation</td>
<td>Powell [20]</td>
</tr>
<tr>
<td>0.58±0.05</td>
<td>Computer Simulation</td>
<td>Rodriguez, Allibert &amp; Chaix</td>
</tr>
<tr>
<td>0.634</td>
<td>Computer Simulation</td>
<td>Mason [31]</td>
</tr>
<tr>
<td>0.582</td>
<td>Computer Simulation</td>
<td>Gotoh, Jodrey &amp; Tory [32]</td>
</tr>
<tr>
<td>0.6366</td>
<td>Computer Simulation</td>
<td>Jodrey &amp; Tory [21]</td>
</tr>
</tbody>
</table>

Table 1.2 Data comparison of experimental & computer simulation

### 1.3 Outline of Thesis

Section 2 describes the packing of monosized spheres. As a first step toward the analysis of random packing of spheres, the regular and random packing arrangements of monosized spheres are
discussed in this section. In section 3, the basic algorithms for converting two dimensional code to three dimensional code are presented. The periodic boundary conditions and geometry checking subroutine are the main parts where that idea is applied. The general concepts of the Monte Carlo Method in the pouring and the shaking simulations are also introduced. Section 4 deals with the analysis of the assemblies which are obtained from the simulation code. Summary and Conclusions are presented in Section 5 with suggestions for further studies.

2. PACKING OF MONOSIZED SPHERES

2.1 Regular Packing of Spheres

A regular packing of spheres may be assembled from layers and rows. The fundamental unit is a row of contacting spheres. These rows can be arranged in the same place, parallel to each other and touching, to form a layer.

The most common packings are built from one or another of the limiting forms. These are the square layer with a 90 degree angle and the triangular or simple rhombic layer with an angle of 60 degree [24]. Those two types of layers are shown in Figure 1.1:
The highest over-all density in a regular packing is achieved in the face-centered cubic (F.C.C.) and hexgonal close-packed (H.C.P.) structures. The FCC structure has four spheres per unit cell and its packing density is calculated as follows:

\[
\frac{V_s}{V_c} = \frac{4 \times \left( \frac{4}{3} \times \pi \times r^3 \right)}{\left( \frac{4 \times r}{\sqrt{2}} \right)^3}
\]

\[
\frac{V_s}{V_c} = 0.7405
\]

where,

- \( V_s \): Total volume of spheres
- \( V_c \): Volume of unit cell
- \( r \): Sphere radius

In the case of HCP structure, each sphere touches three
spheres in the layer below its plane, six spheres in its own plane, and three spheres in the layer above. The packing density is also found to equal 0.7405.

2.2 Random Packing of Spheres

A random packing [23,24] is formed by the haphazard positioning of spheres to form an assembly or a bed. The loose and close random packings characterize the configurations which result when an assembly of spheres is packed in an apparently random manner to its loosest and densest conditions, respectively.

In this work, Monte Carlo method [1,23,25,26] of the type from Computational Statistical Physics is applied to achieve random packings of hard spheres.

2.2.1 Random Loose Packing

This configuration is obtained by packing the spheres so that they roll individually into place over similarly placed spheres by individual random hand packing or by "dropping" the spheres into the container without bouncing.

The most probable value for the packing density of a random loose packing [2,3,10,12,15-17,20,22,32] of monosized spheres is bounded between 0.58 and 0.60.

2.2.2 Random Close Packing

A random close packing for monosized spheres corresponds to
their maximum density without long range order or deformation. These are obtained when the bed is vibrated or vigorously shaken down. Most of the reported experimental values of the packing density for random close packing lies between 0.625 and 0.64 [2,3,6].

In the case of computer simulated techniques, produced packing densities [14,21,31] ranged from 0.628 to 0.6366±0.0004 for monosized spheres.

3. THE SIMULATION CODE AND PROCEDURES

This section outlines the simulation procedure and the Monte Carlo method. The algorithm for this code is presented in Appendix A.1 and the FORTRAN code listing is also found in Appendix B.1. The Monte Carlo method adapted here is commonly used in the field of Computational Statistical Physics. It was developed by von Neumann, Ulman, and Metropolis to study the diffusion of neutrons in fissionable material. The details can be found in [1], [23]-[25],[28], and [29].

3.1 Periodic Boundary Conditions

The two dimensional code is converted to three dimension mainly by modifying the periodic boundary conditions (P.B.C.) and the geometry checking subroutine (GEOMCK). The existing dimensional code has only 6 cases of P.B.C., but 49 cases are considered in three dimension code. The basic idea for establishing the P.B.C. in three dimensions is now described:
(i) X-Y-Z coordinate system is defined in Figure 3.1. Two boundary conditions are established here. One is a hard vertical wall, X-O-Z plane and the others have periodic boundary conditions. A sphere at coordinate \((X, Y, Z)\) reappears at \((X \pm L_x, Y, Z \pm L_z)\) in a periodic boundary condition, so the packing is effectively infinite in horizontal direction.

![Diagram of coordinate system and periodic boundary conditions](image)

**Fig.3.1 Coordinate system and periodic boundary conditions**

(ii) If a new sphere is created on the side of a cell and partially included in the cell as shown in Figure 3.2.(a), the other segment of the sphere appears on opposite side of the cell.
(iii) If a new sphere is created in the corner of a cell and included partially in the cell, the other segments of the sphere appears in three other corners as shown in Figure 3.2.(b).

Fig. 3.2 Periodic Boundary Conditions in each case
(iv) In each case (ii) and (iii), the sphere can lie at seven different locations in X direction as shown in Figure 3.3. Considering the Z direction, combinations of X and Z result in 49 different cases of boundary conditions in this system.

Fig. 3.3 Possible locations of sphere in X direction

By those rules, finally 47 cases of P.B.C. are established and coding is modified to incorporate these cases. In order to check the sphere overlaps, geometry checking subroutine (GEOMCK) is used. All the cases are checked by GEOMCK whether the spheres are overlapped or not. This is effectively done to enforce the hard sphere potential, i.e., spheres can touch without experiencing any attractive or repulsive force, but cannot overlap.
3.2 Pouring Simulation

The "pouring" process starts with moving one sphere at a time according to the following prescription:

\[
\begin{align*}
X &\rightarrow X + \delta \xi_1 \\
Y &\rightarrow Y + \delta \xi_2 \\
Z &\rightarrow Z + \delta \xi_3
\end{align*}
\]

where \( \delta \) is the maximum allowable displacement. \( \xi_1, \xi_2 \) and \( \xi_3 \) are the random numbers between -1 and 1. After moving a sphere, it is equally likely to be anywhere within a cubic of side \( 2\delta \) centered about its original position.

A trial configuration is accepted as the new configuration based on the change of potential energy \( \Delta E \) in the system. If \( \Delta E < 0 \), the new position is allowed by placing the trial sphere in its new position. If \( \Delta E > 0 \), the new position is accepted with probability \( \exp(-\Delta E/kT) \), i.e. compare a random number, and \( 0 \leq J \leq 1 \), with \( \exp(-\Delta E/kT) \); move the sphere to its new position if \( J < \exp(-\Delta E/kT) \). Otherwise, reject the position and keep the sphere at its old location. This process is carried out for all \( N \) particles of the system thereby completing one "pass".

In this simulation, the gravitational potential is permitted only to decrease the configuration energy and no bouncing is permitted. Hence the spheres slowly settle down to the bottom of the container. As the pass number increases, the change of configuration energy becomes smaller. It requires more than a hundred thousand passes to attain an equilibrated state.
The input data for the pouring simulation is presented in Table 3.1.

<table>
<thead>
<tr>
<th>sphere number</th>
<th>1,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>container dimensions (inch)</td>
<td>3.0 x 5.0 x 3.0 (Width x Height x Depth)</td>
</tr>
<tr>
<td>sphere diameter (inch)</td>
<td>0.3</td>
</tr>
<tr>
<td>δ in each pass</td>
<td>1/6 Dia.</td>
</tr>
</tbody>
</table>

Table 3.1 Initial input data for the pouring simulation

3.3 Shaking Simulation

In order to get the densest packing, a shaking procedure is necessary. The spheres are first lifted uniformly by a predefined specific amplitude and then allowed to settle down via the Monte Carlo method without bouncing as described in section 3.2. This completes one cycle.

In this simulation, the shaking amplitude for each case is between one thirds and one sixths of the sphere diameter. Many cycles are required to obtain the "densest" packing. A cycle is halted when the change of the potential energy is less than a predefined tolerance in the input data. Table 3.2 shows the input data for shaking process.

<table>
<thead>
<tr>
<th>amplitude</th>
<th>1/3 - 1/6 Dia.</th>
</tr>
</thead>
<tbody>
<tr>
<td>passes for cycle</td>
<td>40,000</td>
</tr>
<tr>
<td>δ in each pass</td>
<td>1/6 Dia.</td>
</tr>
<tr>
<td>number of cycles</td>
<td>10 cycles</td>
</tr>
</tbody>
</table>

Table 3.2. Input data for the shaking simulation
4. RESULTS

To analyze the sphere assemblies generated, geometrical properties of the assemblies are measured and compared with the published ones. These include the packing fraction, the distribution of coordination numbers and the radial distribution function.

The mean coordination number is computed using three different tolerances, i.e., 1%, 5% and 10% of sphere diameter. The first one included the close contacts within 1% of the sphere diameter in separation. The second and the third one included 5% and 10%, respectively. The comparison of the results with others is based on the 5% diameter separation, because the experimental result of Bernal et al. and the computer simulated result of Matheson are using same tolerance. The details are presented in Section 4.1.

In this work, two methods are used to calculate the packing fraction. The first one is a "Plane Growth Method" and the other one is a "Spherical Growth Method". The details are explained in Section 4.2 and 4.3.

The calculated radial distribution function is presented in Section 4.3 and compared with published results.

In order to obtain the possible factors that effect the low packing densities, three case studies were done and their results are presented in Section 4.4 to 4.6.

All the calculations were carried out using VAX/VMS-8800 computer.
4.1 Coordination Number

The coordination number \([2,21]\) is defined as the mean number of spheres in contact with any given sphere. The expected value of the coordination number seems to be six \([2]\), as each sphere may be generally supported by three others and in turn to support another three spheres.

In order to include all the contacting neighbors, the coordination numbers of the central 563 spheres of the 1000 sphere assembly have been calculated. The coordination number distribution is shown in figure 4.1 for the pouring simulation. The results are computed for the sphere separations of 1.1, 1.05 and 1.01 diameters. The mean coordination numbers are 6.90 at 1.1 diameter separation, 5.97 at 1.05 diameter and 4.98 at 1.01 diameter. These values are measured using the coordination number code located in Appendix B.2. The computed values of the coordination numbers are also presented in Appendix C.1.
(b) 1.05 diameter separation

1000 spheres
1.05 diameter separation
mean number: 5.97

(c) 1.01 diameter separation

1000 spheres
1.01 diameter separation
mean number: 4.98
(d) Comparison of the results

Fig. 4.1 Coordination numbers at 1.01, 1.05 and 1.1 diameter separation after pouring

The experimental result of Bernal & Mason and the computer simulation results of Tory et.al., Jodrey & Tory, Matheson and the current results of pouring simulation are compared in fig 4.2. All the results show a peak value at a coordination of six, except for the result of Bernal et.al.. The results of Tory et.al. and Matheson showed a similar distribution. In comparison with the experimental results by Bernal et.al., the simulated distribution is shifted to the left.
With an amplitude of one sixth of the sphere diameter, 10 cycles (40,000 passes per cycle) of shaking were carried out. Then coordination numbers for each case are computed. The result shows an approximate 6 to 10 per cent increase of coordination number.

The average coordination numbers are 7.55 for 1.1 diameter separation, 6.55 for 1.05 diameter separation and 5.29 for 1.01 diameter separation. Figure 4.3 shows the coordination number histogram for the shaking case. The computed values are found in Appendix C.2.
(a) 1.1 diameter separation

(b) 1.05 diameter separation
Fig. 4.3 Coordination numbers at 1.01, 1.05 and 1.1 diameter separation after 10 cycles of shaking.

(c) 1.01 diameter separation

(d) Comparison of the results
Figure 4.4 shows a comparison of the results between experimental results of Bernal et al. and the current result after 10 cycles of shaking simulation. The mean coordination number of Bernal & Mason's result is 7.99 and the current one is 6.55 for the sphere separation of 1.05 diameter.

Because of the low packing density, the present result shows a configuration shifted to left as compared with the result of Bernal et al..

Fig. 4.4 Comparison of the results between Bernal & Mason's experiment and shaking simulation.
4.2 Packing Fraction

The packing fraction [12-20,24,26,27] or solids fraction is defined as the ratio of the total volume of spheres to the volume containing them. Two methods are used to calculate the packing fraction.

4.2.1 Spherical Growth Method

This method calculated the packing fraction from the 19 spherical samples within the packing. The code may be found in Appendix B.3. The actual volume of solids within each spherical sample is determined by calculating the volume of the spheres totally within the radius plus fractional volume of those of those spheres which intersected the sampling sphere. The details are shown in Figure 4.5.

![Spherical Growth Method Diagram](image)

Fig.4.5 The basic algorithm of Spherical Growth Method
The volume common to two spheres [12] of radii \( a \) and \( b \), with centers a distance \( c \) apart is given by:

\[
V = \frac{\pi}{3} \times \left[ 2 \times a^3 + 2 \times b^3 + c^3 - 3 \times c \times (d^2 + b^2) \right]
\]

where,

\[
|a - b| \leq c \leq a + b
\]

\( V \) : common volume

\( a \) & \( b \) : radii of two spheres

\( c \) : distance of centers between two spheres

\( d \) : \( \frac{a^2 + c^2 - b^2}{2 \times c} \)

A spherical sample containing central 598 spheres is taken from the packing and the packing fraction is calculated for the intervals of 0.05 sphere diameters. The measured mean packing fractions are 0.555±0.015, 0.582±0.018 for the pouring and the shaking simulation, respectively.

Figures 4.6 (a) and (b) show the packing fractions for the pouring and shaking cases versus \( r/dia. \) where \( dia. \) equals the diameter of the sphere and \( r \) is the radial distance measured outward from the center of the packing. There is found a small peak at 1.33 sphere diameter outward from the center of the packing and the result of G.D.Scott[30] shows a similar distribution of packing fraction.
(a) after pouring simulation

1000 spheres
0 cycle
110,000 passes
dia = 0.3"
mean packing fraction : 0.555
configuration energy : 0.1011E+00 J

(b) after 10 cycles of shaking (40000 passes per cycle)

1000 spheres
10 cycles
510,000 passes
dia = 0.3"
mean packing fraction : 0.582
configuration energy : 0.9603E-01 J
% decrease of potential energy : 5.01 %
(c) Comparison of the results between after pouring and shaking

Fig. 4.6 packing fractions by Spherical Growth Method
4.2.2 Plane Growth Method

This method first cuts the packing by a plane and calculates the volume of spheres bounded by that plane and the periodic "walls". The details are shown in Figure 4.7.

![Diagram of a spherical segment](image)

**Figure 4.7 The volume of a spherical segment**

The volume of spherical segment of one base [35] is given by:

\[
V_s = \frac{1}{6} \times \pi \times h \times \left( 3 \times a^2 + h^2 \right)
\]

Where,

\( V_s \): volume of spherical segment
\( h \): height of a spherical segment
\( a \): intersected distance between plane and sphere
\[ = \sqrt{h \times \left( 2 \times R - h \right)} \]
\( R \): radius
The local packing fractions are calculated from the bottom to the top of the packing for the intervals of 0.1 inch. The resulting mean packing fractions are 0.551±0.01 for the pouring case and 0.581±0.006 for 10 cycles of shaking case, which is in a good agreement with the results obtained by spherical growth method. The mean packing fractions by spherical growth method are 0.555±0.015 and 0.582±0.018 for the pouring and shaking, respectively. The published results of Visscher & Bolsterli, Tory et.al., Powell and Gotoh et.al. show similar packing fractions with the current results.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Packing fractions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tory, Cochrane &amp; Waddell [10]</td>
<td>0.59</td>
</tr>
<tr>
<td>Visscher &amp; Bolsterli [15]</td>
<td>0.582</td>
</tr>
<tr>
<td>Tory et.al. [16]</td>
<td>0.58</td>
</tr>
<tr>
<td>Powell [20]</td>
<td>0.59</td>
</tr>
<tr>
<td>Gotoh et.al.[32]</td>
<td>0.582</td>
</tr>
<tr>
<td>Current results</td>
<td>0.581−0.582</td>
</tr>
</tbody>
</table>

Table 4.1 Comparison of the packing fractions

Fig 4.8 shows the distribution of local packing fractions from the bottom of the packing.
(a) After pouring case

(b) After 10 cycles of shaking (40000 passes per cycle)
(c) Comparison of the results

Fig. 4.8 packing fractions by plane growth method
4.3 Radial Distribution Function

The radial distribution function \([4,24,29,30]\) is defined as the number of spheres (or density of sphere centers) as a function of distance from the center of the packing. In other words, it is the average number of sphere centers per unit volume in a spherical shell about a central sphere. By the definition, radial distribution function \(g(r/D)\) is,

\[
g(r/D) = \frac{N_{av}}{4 \pi \times (r/D)^2 \times \Delta(r/D)}
\]

where,

- \(N_{av}\) : average number of sphere centers per interval
- \(\Delta(r/D)\) : interval (= one-fifth of sphere diameter was used)
- \(r\) : radial distance
- \(D\) : sphere diameter

The values of \(g(r/D)\) is plotted versus \(r/D\) and this is shown in Fig. 4.7. The measurement was made for a cluster of 1000 spheres and the code listing is found in Appendix B.5. The computed list of data is also found in Appendix C.7 and C.8. Some published values of the radial distances of the first, second, third, fourth and fifth peaks are presented in the Table 4.2.
The results for the poured and the shaken assemblies are illustrated in figure 4.9. The first peak in the distribution function lies in the interval 1.0 - 1.1. Since the spheres cannot overlap, values of r/D can not occur less than 1.0. The maxima of peaks 2, 3 and 4 of the assembly in pouring case occurred at 1.8, 2.6 and 3.4 sphere diameters. These values are nearly the same as Scott [30] and slightly larger than Matheson’s values. After 10 cycles of shaken, the value of $g$ at r/D = 1 increased from 0.5409 to 0.5806 and this also appears as an increase of the coordination number. Figure 4.9.c presents the comparison of the G.D.Scott’s result with the simulated results.

<table>
<thead>
<tr>
<th>Reference</th>
<th>first</th>
<th>second</th>
<th>third</th>
<th>fourth</th>
<th>fifth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bennett [12]</td>
<td>1.00</td>
<td>1.73</td>
<td>2.68</td>
<td>3.53</td>
<td>4.38</td>
</tr>
<tr>
<td>Finney [13]</td>
<td>1.00</td>
<td>1.73</td>
<td>2.65</td>
<td>3.50</td>
<td>4.35</td>
</tr>
<tr>
<td>Matheson [17]</td>
<td>1.00</td>
<td>1.8</td>
<td>2.78</td>
<td>3.64</td>
<td>4.45</td>
</tr>
<tr>
<td>Scott [30]</td>
<td>1.00</td>
<td>1.83</td>
<td>2.64</td>
<td>3.45</td>
<td>-</td>
</tr>
<tr>
<td>Current result</td>
<td>1.00</td>
<td>1.9</td>
<td>2.7</td>
<td>3.5</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Table 4.2 r/dia. at positions of peaks
(a) Pouring case

(b) Shaking case (10 cycles)
Fig. 4.9 Radial distribution function for a 1,000 sphere configurations

(c) Comparison with the published data [30]
4.4. System Size Dependence

In order to verify that the results are independent of the number of particles, four different cases were done by varying the numbers of spheres. The cell dimension for each case was 3.0" x 3.0" (base area) x 5.0" (height).

In each case, the spheres were poured into the cell to obtain a configuration in the equilibrated state.

In order to measure the packing densities in a similar condition, the packing of each system was made in a similar height by varying the radius of sphere. The sphere diameters used in this simulation were 0.75" for 64 sphere system, 0.55" for 125 sphere system, 0.45" for 216 sphere system and 0.4" for 343 sphere system. Normalized configuration energy versus pass number shows the height of each system in the equilibrated state. Table 4.4 lists these energies for each size system in the equilibrium. Here \( Z_i \) denotes the location of the sphere center above the cell bottom, \( m_i \) is the sphere mass and \( g \) is the gravitational acceleration.

<table>
<thead>
<tr>
<th>cases</th>
<th>( \sum_{i=1}^{n} m_i g Z_i )</th>
<th>( \sum_{i=1}^{n} m_i )</th>
<th>( \sum_{i=1}^{n} g Z_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>0.1060</td>
<td>0.2858</td>
<td>0.3709</td>
</tr>
<tr>
<td>125</td>
<td>0.06475</td>
<td>0.2201</td>
<td>0.2942</td>
</tr>
<tr>
<td>216</td>
<td>0.05622</td>
<td>0.2083</td>
<td>0.2699</td>
</tr>
<tr>
<td>343</td>
<td>0.06701</td>
<td>0.2323</td>
<td>0.2885</td>
</tr>
</tbody>
</table>

Table 4.2 Normalized configuration energy
This simulation was repeated on systems of 125, 216 and 343 spheres and run for 44,000 passes for each case. The final packing densities are 0.553±0.01 for 64 sphere case, 0.535±0.01 for 125 sphere case, 0.545±0.01 for 216 sphere case and 0.564±0.01 for 343 sphere cases. Each size case were carried 3 times to obtain an average value. Comparing these results with the result of 1000 sphere case, the mean packing density of 1000 sphere case (0.555±0.015) lies within these values. The results of four separate cases also show independence between the system size and the packing density. So the resulting low packing density of current simulation is not affected by the system size.

The final packing densities for each case are plotted in Figures 4.10 (a), while (b) - (e) shows the variation versus pass number.
(b) 64 sphere case

64 spheres
sphere dia : 0.75
0 cycle
430000 passes
normalized configuration energy : 0.37094

(c) 125 sphere case

125 spheres
sphere dia : 0.55
0 cycle
440000 passes
normalized configuration energy : 0.29417
Fig. 4.10 packing density in each case

(d) 216 sphere case
- 216 spheres
- sphere dia : 0.45
- 0 cycle
- 440000 passes
- normalized configuration energy : 0.26992

(e) 343 sphere case
- 343 spheres
- sphere dia : 0.4
- 0 cycle
- 440000 passes
- normalized configuration : 0.28855
4.5. Comparison of the Packing Density between Corrected & Uncorrected Values

This work was done to attempt to discover what could account for the 5 to 6 per cent lower density from published experimental data.

Let the distance between one geometric neighbor and its center sphere be \( d \), and let its radii be \( r_1 \). In this case the geometric neighbor is in contact with the center sphere if,

\[
d = 2 \times r_1
\]

In the simulation process, the equality can never be exactly obtained because of the machine error.

To locate the nearest neighbor and calculate the distance, a 1000 sphere configuration was produced, processing 415,000 passes. Since there exists only one nearest neighbor, there are 1000 nearest neighbor distances. These distances fell between 1.000000024124545 and 1.017997631992374 sphere diameter. For all practical purpose, the lowerbound is considered to be 1.0 due to machine error.

Let \( \sigma \) be the diameter of the spheres in the packing and \( P(D) \) be the cumulative probability [36] that the nearest neighbor is located in the range of \( \sigma \leq D \leq \sigma + d\sigma \). Then, for a fixed packing fraction \( \eta \), the median nearest neighbor distance \( D_{mn}(\eta) \) is defined by:

\[
P(D_{mn}) = \frac{1}{2}
\]
The median nearest neighbors are 1.000006586507291 sphere diameter for the pouring case and 1.000096933545690 sphere diameter after 12 cycles of shaking (20,000 passes per cycle).

The cumulative probability versus normalized distance r/dia is plotted in Figure 4.11.

By using the median value \( R_{\text{mnn}} \), to compute the sphere volume, a corrected packing density is calculated as follows:

\[
V_{\text{sp}} = \frac{4}{3} \times \pi \times ( \frac{R_{\text{mnn}}}{2} )^3 \times N
\]

\[
V_{\text{oc}} = X_i \times Y_i \times Z_i
\]

\[
pd_{\text{corr}} = \frac{V_{\text{sp}}}{V_{\text{oc}}}
\]

Where,

- \( V_{\text{sp}} \): total volume of spheres
- \( V_{\text{oc}} \): occupied volume
- \( pd_{\text{corr}} \): corrected packing density
- \( N \): number of spheres
- \( R_{\text{mnn}} \): \( D_{\text{mnn}} \)/2

The corrected packing density was 0.5338. The difference between uncorrected and corrected packing densities is 0.002 per cent in the pouring case.

The same procedure was repeated for the shaking case and the corrected packing density was computed to be 0.57923, a very insignificant increase from the uncorrected value of 0.579. The increase was approximately 0.029 per cent.

Because the volume difference between the corrected and the uncorrected one is not significant, the use of the "corrected" sphere diameter cannot be a significant factor in accounting for the 5 to 6 per cent difference between the experimental data and
After pouring (415,000 passes):

1000 spheres
0 cycle
415000 passes
sphere diameter : 0.3
$R_{\text{ave}} : 1.000006586507291$ sphere diameter
= 0.300001976
Fig. 4.11 The cumulative probability distribution versus the normalized distance

In order to characterize the geometry of these two packings more exactly, the coordination numbers for both cases are also calculated. The mean coordination numbers are 5.91 for the pouring case and 6.46 for the shaking case. The distributions of the coordination numbers are plotted in Figure 4.12.
Fig. 4.12 Distribution of the coordination number

(a) After pouring

- 1000 spheres
- 0 cycle
- 415,000 passes
- mean coordination: 5.91

(b) After 12 cycles of shaking (20,000 passes per cycle)

- 1000 spheres
- 12 cycles
- 655,000 passes
- mean coordination: 6.46

Fig. 4.12 Distribution of the coordination number
4.6. The Annealing Simulation

In the previous simulations, we choose a very large value for $1/k_B T$ as 1.0E+30 where $k_B$ is a Boltzmann constant and $T$ is an absolute temperature. This choice is equivalent to allowing only a downward movement in order to minimize the system potential energy.

Because the value of $1/k_B T$ was so large, the system cooled rapidly and possibly prevented the formation of a greater density. In order to check this factor, the "annealing simulation" was done in a 64 sphere system. The system was heated with a high temperature to an equilibrated state and then slowly cooled by decreasing the temperature of the system.

A normalized temperature, $T^*$, is defined as follows:

$$T^* = \frac{k_B T}{m_s g d_s}$$

where,

- $k_B$: Boltzmann constant ($= 1.380 \times 10^{-23} \text{ JK}^{-1}$)
- $T$: Absolute temperature (K)
- $m_s$: Mass of a sphere
- $g$: Gravitational velocity
- $d_s$: Diameter of a sphere

The normalization is made as a comparison with the gravitational potential energy.

The annealing simulation was started with an initial value of $T^*$ as 15.81. When the system was brought to an equilibrated
state, $T^*$ was changed to a smaller value and again the simulation was run until the system reached another equilibrium state at $T^*$. Table 4.3 shows the sequence of $T^*$. The final configuration was obtained after 680,000 passes. The resulting packing density without using the annealing simulation was 0.595 which was computed after 550,000 passes. Comparison of these two results shows approximately 5.2 per cent difference. This gap is almost the same as the difference between the current result and the result of G.D.Scott. The values are 0.555 and 0.606 for the current result and the result of G.D.Scott, respectively.

The resulting packing densities according to $T^*$ are also shown in Table 4.3.

<table>
<thead>
<tr>
<th>$T^*$</th>
<th>pass number</th>
<th>packing density</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.58</td>
<td>120,000</td>
<td>0.249</td>
</tr>
<tr>
<td>0.158</td>
<td>280,000</td>
<td>0.504</td>
</tr>
<tr>
<td>0.0158</td>
<td>440,000</td>
<td>0.608</td>
</tr>
<tr>
<td>0.00316</td>
<td>560,000</td>
<td>0.617</td>
</tr>
<tr>
<td>0.00211</td>
<td>680,000</td>
<td>0.618</td>
</tr>
</tbody>
</table>

Table 4.3 The packing densities according to $T^*$

Eventually the system reached an equilibrated state and the change in the packing density became less than 0.1 per cent.

The result in the annealing simulation shows that the manner in which the system is dropped to a $T^* \approx 0$. Therefore, this is a significant factor in accounting for the deficit between experimental data and the simulated result.
5. SUMMARY & CONCLUSION

The random packing of spheres is a process of considerable scientific interest and practical importance. A variety of simple models have been developed to obtain a better understanding of technically important processes.

In this work, a Monte Carlo simulation code [1] has been extended from two dimension to three dimension and then to investigate the properties of ranom packing of hard spheres.

With configurations of 1000 spheres obtained from the simulation code, the properties of the assemblies are calculated and compared with other results in many ways such as coordination number, packing density and radial distribution function etc.

The followings are the results and the conclusions:

(1) The coordination numbers for 3 different diameter separations (ie., 1.01, 1.05 and 1.1) are calculated and compared with the published data. All the comparisons are based on using the 1.05 sphere diameter separation because the experimental and computer simulated results of Bernal et.al. and A.J.Matheson uses same tolerances. In the pouring simulation, the peak value occurred at a coordination number of approximately six similar to the result obtained by Jodrey et.al.. In shaking simulation studies, the peak value occurred at seven coordination, but the average number is lower than the experimental results of Bernal et.al.. The average coordination numbers are 6.55 for the shaking simulation and 7.99 for Bernal et.al..

(2) Another way of characterizing the bulk configuration of the system is the packing fraction. This quantity is measured by
two different methods called Spherical Growth Method and Plane Growth Method. A good agreement for the results is obtained by both methods. In the case of pouring simulation, the packing fractions are 0.555±0.015 by the spherical growth method and 0.551±0.01 by the plane growth method. The packing fractions obtained from the shaking simulation give a packing fraction of 0.582±0.018 using the "Spherical Growth Method" and 0.581±0.006 using the "Plane Growth Method". Both in the small and large systems, the results from the plane growth method show a good accuracy.

Comparisons with the published data shows that the resulting packing fractions are approximately 5 to 6 per cent lower than the experimental results of G.D.Scott.

Three case studies are done to find the significant factor in accounting for the 5 to 6 per cent deficit of the packing density between the cited experimental results. The following are the results and conclusions of three case studies.

(4) To check the dependence of the system size, 4 separate cases were simulated. The simulation was repeated 3 times each using an identically sized cell. Each study was allowed to run for 400,000 passes to obtain final equilibrium configurations. The final averaged packing densities for each case do not show the dependence between the system size and the packing density. Hence it is concluded, the simulation results are not system-size dependent.

(5) Comparison of the "corrected" and the "uncorrected packing densities" was done to attempt to discover what could account for the 5 to 6 per cent lower density than published
experimental results. The corrected packing density was computed using the median nearest neighbor distance and compared with the results of the uncorrected packing density. The resulting uncorrected and corrected packing densities in the pouring case were 0.5335 and 0.5338, respectively. In the shaking case, the resulting uncorrected and corrected packing densities were 0.579 and 0.57923, respectively. The differences between two cases were only 0.002 per cent and 0.029 per cent. So the use of the corrected sphere diameter cannot be a significant factor.

(6) The annealing simulation was done in an attempt to determine if the way the system was cooled effected the density. This process excluded the possibility which could prevent the formation of a greater density because of the rapid cooling of the system. The system was first heated with a high temperature then slowly cooled. The results showed approximately 5.2 per cent increase of the packing density as compared with the rapid cooling results. The packing densities are 0.565 for the rapid cooling simulation and 0.618 after annealing simulation. It is found that the method of dropping the system is a critical factor effecting the packing density and this could account for 5 to 6 per cent difference between the experimental data and the simulated results. Many case studies are necessary to verify this. However, the preliminary results cited here indicate that the claim is true.

Some aspects for further research are as follows:

(1) Find an optimized sequence of values $T^*$ from the results of annealing simulation on various system sizes.

(2) The Voronoi diagram may be used in the analysis of the
results. If this diagram is applied in the analysis, the exact value of the coordination number and its distribution are obtained. In this way, the configuration may be looked at on a local level.
REFERENCES


13. J. L. Finney, "Random Packing and the Structure of Simple
Liquids. 1. The Geometry of Random Close Packing,"


15. W.M.Visscher and M.Bolsterli, "Random Packing of Equal and
Unequal Spheres in Two and Three Dimensions," Nature, 239,
504-507(1972)


17. A.J.Matheson, "Computation of a Random Packing of Hard

to Random Packing Density of Equal Spheres," Nature, 252,
202-205(1974)

19. W.S.Jodrey and E.M.Tory, "Simulation of Random Packing of
Spheres," Simulation, 1-12(Jan.1979)

20. M.J.Powell, "Computer-Simulated Random Packing of Spheres,

21. W.S.Jodrey and E.M. Tory, "Computer Simulation of Isotropic,
Homogeneous,Dense Random Packing of Equal Spheres," Powder
Technology, 30, 111-118(1981)

for Random Packing of Spheres of Unequal Size," Powder Technology,
47, 25-33(1986)

23. D.P.Haughey and G.S.G. Beveridge, "Structural Properties of

Elsevier Science, Amsterdam, The Netherlands, Chapter 1 - 3(1987)


36. J.G. Berryman, "Random Close Packing of Hard Spheres and

APPENDIX A:

A.1 Algorithm for Monte Carlo simulation code

(a) Initial configuration

START

READ INPUT DATA

GENERATE RANDOM NUMBER SEED USING SYSTEM CLOCK

RESTART?

YES

READ INFORMATION FROM THE PREVIOUS RUN

'POUR' OR 'NP'?

POUR

GO TO POURING SIMULATION

NP

GO TO SHAKING SIMULATION

GENERATE A SPHERE OF SAME DIAMETER IN A PREFIXED CONTAINER

YES

SPHERE OVERLAPS? (GEOMETRY CHECKING)

NO

FINISH GENERATING N - SPHERES?

YES

CALCULATE THE ENERGY OF THE CONFIGURATION

WRITE OUTPUT DATA

GO TO SIMULATION PART

NO

GO TO SIMULATION PART
(b) Pouring simulation: begin moving spheres and generating new configurations.
(c) Shaking simulation: begin moving spheres with amplitude and generating new configurations
A.2 Algorithm for Coordination Number

START

READ X, Y, Z

CHOOSE ALL SPHERES TOTALLY INSIDE THE CONTAINER?

YES

CALCULATE THE DISTANCE FROM CENTER TO ALL THE OTHER SPHERES

COUNT THE NEAREST ONES (COORDINATION NUMBER) WITHIN PREFIXED TOLERANCE

NO

CHECK ALL THE SPHERES?

YES

ADD FREQUENCY FOR EACH CASE OF COORDINATION NUMBER

WRITE COORDINATION NUMBER AND FREQUENCY

STOP

58
A.3 Algorithm for Packing Fraction

(a) Spherical Growth Method

START

READ X, Y, Z

CHOOSE A SPHERICAL SAMPLE WITH A PREFIXED RADIUS FROM THE CENTER OF THE PACKING

CALCULATE THE VOLUME OF SPHERES INSIDE THAT SPHERICAL SAMPLE

CALCULATE THE VOLUME OF THE SAMPLE

COMPUTE THE DENSITY (= SPHERE VOL./SAMPLE VOL.)

STOP
(b) Plane Growth Method

START

READ X, Y, Z

CHOOSE A SAMPLE BY CUTTING A PACKING WITH A HORIZONTAL PLANE

CALCULATE THE VOLUME OF SPHERES INSIDE THAT SAMPLE

CALCULATE THE VOLUME OF THE SAMPLE

COMPUTE THE DENSITY (= SPHERE VOL./SAMPLE VOL.)

STOP
A.4 Algorithm for Radial Distribution Function

START

READ X, Y, Z

CALCULATE THE DISTANCE FROM CENTER SPHERE TO THE OTHER SPHERES USING P.B.C.

COMPUTE THE NUMBER OF SPHERE CENTERS WITHIN PREDEFINED INTERVAL \((R, R+\Delta R)\)

CALCULATE THE R.D.F. WHICH DEFINED AS THE FREQUENCY DIVIDED BY \(4\pi (R/D\lambda)^2\)

STOP
A.5 Algorithm for calculating Median Nearest Neighbor

START

READ X, Y, Z

CALCULATE ALL THE GEOMETRIC NEAREST NEIGHBOR DISTANCES WITHIN PREFIXED TOLERANCE FROM GIVEN SPHERES.

CHECK ALL THE SPHERES?

YES

CHOOSE THE NEAREST DISTANCE FOR EACH SPHERE

SORT THOSE DISTANCES IN ORDER

NORMALIZED THE DISTANCES (DEVIDE BY DIAMETER OF A SPHERE)

STOP
APPENDIX B:

B.1 Monte Carlo Simulation Code

* ***************************************************************
* MONTE CARLO SIMULATION OF SETTLING OF SPHERES AND SHAKING *
* ***************************************************************

Variable List

beta | "temperature" parameter appearing in Boltzmann distribution

| real-valued array of different diameters in polydisperse systems

delt | maximum allowable sphere displacement
dens | density in (gms/cm**3)
dia | array of sphere diameters
dmax | diameter of largest sphere
dmin | diameter of smallest sphere
dpe | difference in the potential ENERGY between the current and a previous configurations
dsame | character variable: value is 'yes' if all the particles are the same and 'no' if array of diameters are to be read in
diam | disc diameter when all spheres are the same size
eps | tolerance used in conjunction with ebrat
emean | mean value of the ENERGY over "mp1" values
edev | standard deviation of ENERGY array over "mp1" values
ebr | array of averaged ENERGY values; ebr(k) = average of 0 e(1) through e(k)
ebrp | previous value of ebr (at pass k-1)
ebrat | ratio of current average ENERGY to previous average ENERGY
g | 9.8 meters/(sec**2)
height | height of parallelopiped (inches)
iaccpt | number of accepted moves performed to generate one pass
iaccpt | cycle at which restart begins
icycle | cycle counter
id | integer array of size "kn" whose value id(k) represents the diameter dsort(k)
ipaspr | pass print iteration counter
iprint | print counter within each pass
iterpr | integer designating the iteration number in a specific pass at which a printout is desired
iterg | integer specifying frequency at which ENERGY value is output
ix | parameter used by SETRAN
length | length of the parallelopiped (inches)
mass | mass of the sphere in kilograms
maxgen | maximum number of trial to generate initial positions
maxpas | maximum number of complete passes
mode | character variable passed into GEOMCK which designates the mode in which GEOMCK is to operate (either 'generate' or 'simulate')
mp1 | maxpas + 1
n | total number of spheres
nh | number of layers (horizontal) of large spheres generated for the initial configuration (if the layer option is 'yes').
maxcyc | maximum number of "shaking" cycles
nbig | number of large spheres to be located in close-packed configuration on container bottom. This initial configuration will be generated if the variable 'layer' is input as 'yes'.
ns | integer array containing distribution of the number of sphere sizes of diameter array d; Note that the sum of all the entries of this array must be n
nsize | number of diameters in polydisperse systems
nv | number of columns of large spheres (vertical) to be generated in the initial configuration (on the container bottom) if the layer option is read in as 'yes'.
nw | number of spheres along z-axis.
paspr | integer designating the pass number at which to print
pbc | character variable (yes,no) for implementation of periodic boundary conditions or not (no => hard vertical walls)
pe | potential ENERGY of the current configuration
peo | potential ENERGY of the previous configuration
poly | character variable (yes,no) designating polydisperse system S
psintl | designates the pass number at which the program starts. It is nonzero only if the value of "restrt" is yes
ra | sphere radius when all spheres are the same size
restrt | character variable: value is 'yes' if a restart is to be done and 'no' if no restart required
xnew | trial x-coordinate of a particular spheres.
x | array containing x-coordinates of spheres.
ynew | trial y-coordinate of a particular spheres.
y | array containing y-coordinates of spheres.
z | array containing z-coordinates of spheres.
znew | trial z-coordinate of a particular spheres.
ymean | average of the y array
ydev | standard deviation the y array
Subroutines and Functions

AINSRT  This routine inserts the newly generated coordinate into position "isave" of a real-valued array. (4 arguments)
ENERGY  This function computes the potential ENERGY of the current configuration based on the height of each sphere above the datum position. (6 arguments)
GEOMCK  This routine searches the trial configuration for geometric violations (forbidden overlap of spheres). If there is a violation, a value of -1 is returned for "ier". (9 arguments)
IINSRT  This routine is identical to AINSRT except that it works on integer-valued arrays.
SEARCH  This routine searches for the location in which to insert the trial x-coordinate (xtemp) into the existing x array. It returns this location as isave, which is then used by the other routines. (4 arguments)
SHELL   This routine does a shell sort of a one-dimensional array into increasing order (2 arguments).
SDEV    This routine returns the mean and standard deviation of a sample (4 arguments).
YBARF   This function computes the y coordinate of the area centroid (3 arguments).

Input and output file unit numbers

Unit 9  File containing the output from the program. It is opened as 'sxp4639.mc3d.mc3d.out'
Unit 7  File containing the input data. It is opened as 'sxp4639.mc3d.mc3d.dat', status = "old".
Unit 12 File containing the configuration number and ENERGY for each pass. It is defined as 'sxp4639.mc3d.mc3d.erg'.
Unit 31 File containing the restart data. It is opened as 'sxp4639.mc3d.mc3d.res'.
Unit 36 File containing the configuration data to be transferred to the DEC-20 system and then plotted via DISPLA. It is opened as 'sxp4639.mc3d.mc3d.plo'

Beginning of MAIN Program

Implicit Real*8 (a-h, o-z)
Real*8 imax, mass
d(100), dia(2000), rad(2000), yt(50), ebr(5000)
Integer freq(300), id(100), ns(100)
Common diam, dmin, ipass
Character * 3 restrt, dsame, poly, layer, pbc
Character * 8 mode
Character * 4 pour
Integer passpr, psintl, totpas
Integer temdim, select, 1p, j, lor, upbd, lwbd, jor, jtr

Open units 36, 9, 12 and 7

open(unit=36, file='[sxp4639.mc3d]mc3d.plo', status='new')
open(unit=9, file='[sxp4639.mc3d]mc3d.out', status='new')
open(unit=7, file='[sxp4639.mc3d]mc3d.dat', status='old')
open(unit=12, file='[sxp4639.mc3d]mc3d.erg', status='new')

read(7,*) length, height, width, diam, dens, beta, delt
read(7,*) n, maxpas, maxgen, nsize
read(7,*) iterpr, passpr, iterg
read(7,*) eps
read(7,921) yjump0
read(7,*) maxcyc
read (7,902) restrt
read (7,902) poly
read (7,902) pbc

Initialization of parameters, indices and file outputs

g = 9.8D0
ndim = n + 1
n1 = n * (n - 1) * 0.5
mpl = maxpas + 1
ra = diam * 0.5D0
eps1 = 1.0D0 - eps
pi = 3.1415926536D0
icycle = 0
icyc0 = 0
yjump = 0.0D0
totpas = 0
mxc1 = maxcyc + 1
mode = 'generate'

Generate the Random Number Seed (ix) using system clock
rns = SECNDS(0.0)
rns = pi * rns * 1.0D3
ix = IDINT(rns)
C
if ((restrt .eq. 'YES').or.(restrt .eq. 'yes')) go to 110
read(7,920) pour
read(7,902) dsame
read(7,902) layer
if((dsame.eq.'YES').or.(dsame.eq.'yes')) then
dmax = diam
dmin = diam
atem = 4.0D0/3.0D0 * pi * (ra*ra*ra) * ((2.54D+00)**3)
ams = atem * dens * 1.0D-03
do 500 ir = 1, n
   dia(ir) = diam
   mass(ir) = ams
500
   continue
   close(unit=7)
go to 2
endif
C
C
This part of the code will generate an initial configuration
C in which part of the container will contain "nbig" large
C spheres. On top of this, there will be "n - nbig" smaller
C spheres. (There will be 2 layers.)
C
C
IF ( ( layer .eq. 'YES' ) .or. ( layer .eq. 'yes' ) ) THEN
read(7,* ) dbig, dsmall
read(7,* ) nv, nh, nw
dmax = dbig
dmin = dsmall
nbig = nv * nh * nw
do 5550 ir = 1, nbig
   dia ( ir ) = dbig
5550
   continue
   do 5560 ir = nbig + 1, n
      dia ( ir ) = dsmall
5560
   continue
   close ( unit = 7 )
C
rbig = 0.5D0 * dbig
rsmall = 0.5D0 * dsmall
do 5581 k=1, nw
do 5580 i = 1, nv
do 5570 j = 1, nh
   im1=i-1
   jm1 = j - 1
   km1=k-1
   Kx = i + (jm1 * nv)+(km1*nv*nh)
x(kx) = rbig + dbig * jm1
y(kx) = rbig+dbig*im1
z(kx) = rbig + dbig *km1
continue
continue
continue
kdim = (nv * nh * nw) + 11
lmax = length - rsmall
hmax = height - rsmall
wmax = width - rsmall
xtemp = RAN ( 1x ) * length
if (( xtemp .le. rsmall ) .or. ( xtemp .ge. lmax )) go to 5590
ymin = nv * dbig
ytemp = RAN ( 1x ) * height + ymin
if (( ytemp .le. rsmall ) .or. ( ytemp .ge. hmax )) go to 5592
ztemp = RAN(1x)*width
if (( ztemp.le.rsmall).or.(ztemp.ge.wmax)) go to 5593

C

call SEARCH ( xtemp, x, Kdim, isave )
call GEOMCK ( x, y,z, dia, xtemp, ytemp,ztemp, dbig, dsmall, +
              isave, Kdim, 0, mode, ier )
C
if ( ier .eq. 0 ) then
    call AINSRT ( xtemp, x, Kdim, isave )
call AINSRT ( ytemp, y, kdim, isave )
call AINSRT ( ztemp, z, Kdim, isave )
call AINSRT ( dsmall, dia, Kdim, isave )

icount = icount + 1
kdim = kdim + 1
if ( icount .gt. maxgen ) go to 888
if ( kdim .le. n ) then
go to 5590
else
    xpid = p1 * ((2.54D+00)**3) * dens * 1.0D-03
    do 5594 i = 1, n
         mass(i)=4.0DO/3.0DO*xpid*((dia(i)*0.5DO)**3)
5594 continue
go to 377
endif
else
    icount = icount + 1
    if ( icount .gt. maxgen ) go to 888
    go to 5590
end if

C
ENDIF

C

[Either generate a polydisperse array or read in
distribution from unit 7]
C
IF (( poly .eq. 'NO' ) .or. ( poly .eq. 'no' )) THEN
    read(7,*) n2
    do 395 i = 1, n2
    read(7,*) x(i), y(i), z(i), dia(i)
395 continue
read(7,*) dmax, dmin
rmin = dmin * 0.5DO
rmax = dmax * 0.5DO
close(unit=7)
do 392 i = n2+1, n
   dia(i) = diam
392  continue
C [generate the remainder of the spheres of diameter
C 'diam']
icount = 1
kdim = n2 + 1
393  xtemp = RAN(1x) * length
   if((xtemp.le.ra) .or. (xtemp.ge.length-ra)) go to 393
397  ytemp = RAN(1x) * height
   if((ytemp.le.ra).or.(ytemp.ge.height-ra)) go to 397
398  ztemp = RAN(1x) * width
   if((ztemp.le.ra) .or. (ztemp.ge.width-ra)) go to 398
   Call SEARCH(xtemp,x,kdim,isave)
   Call GEOMCK(x,y,z,dia,xtemp,ytemp,ztemp,dmax,diam,diam,isave,
   kdim,0,mode,ier)
   +
   if(ier .eq. 0) then
      Call AINSRT(xtemp,x,kdim,isave)
      Call AINSRT(ytemp,y,kdim,isave)
      Call AINSRT(ztemp,z,Kdim,isave)
      Call AINSRT(diam,dia,kdim,isave)
      kdim = kdim + 1
      icount = icount + 1
      if (icount .gt. maxgen) go to 888
      if(kdim .le. n) then
         go to 393
      else
         xpid = pi * ((2.54D+00)**3) * dens * 1.0D-3
         do 391 ia = 1, n
            tempa=4.0DO/3.0DO*xpid*((dia(ia)*0.5DO)**3)
            mass(ia) = tempa
391  continue
      go to 377
   endif
else
   icount = icount + 1
   if (icount .gt. maxgen) go to 888
   go to 393
endif
ELSE
C [Generate polydisperse system of spheres]
C
jend = 0
icount = 1
kdim = 1
read (7,*) dmax, dmin
dO 2000 K = 1, NSIZE
   read (7, *) ns(k), d(k)
   if ( d(k) .gt. dmax ) dmax = d(k)
   if ( d(k) .lt. dmin ) dmin = d(k)
rad(k) = 0.5DO * d(k)
if (k .eq. 1) then
   jstrt = 1
else
   jstrt = jend + 1
endif
jend = jend + ns(k)
do 551 j = jstrt, jend
   dia(j) = d(k)
551 continue
xtemp = RAN(1x) * length
if((xtemp.le.rad(k)).or.
   (xtemp.ge.length-rad(k))) go to 501
+ 501 ytemp = RAN(1x) * height
if((ytemp.le.rad(k)).or.
   (ytemp.ge.height-rad(k))) go to 502
+ 502 ztemp = RAN(1x) * width
if((ztemp.le.rad(k)).or.
   (ztemp.ge.width-rad(k))) go to 506
if (kdim .eq. 1) then
   isave = 1
   go to 505
endif
Call SEARCH(xtemp, x, Kdim, isave)
505 Call GEOMCK(x,y,z,dia,xtemp,ytemp,ztemp,dmax,d(k),isave,
   Kdim,0,mode,ier)
if (ier .eq. 0) then
   Call AINSRT(xtemp, x, Kdim, isave)
call AINSRT(ytemp, y, kdim, isave)
Call AINSRT(ztemp, z, Kdim, isave)
Call AINSRT(d(k), dia, Kdim, isave)
Kdim = Kdim + 1
icount = icount + 1
if (icount .gt. maxgen) go to 888
if (Kdim .le. jend ) go to 501
else
   icount = icount + 1
   if (icount .gt. maxgen) go to 888
   go to 501
endif
2000 CONTINUE
C
xpid = p1 * ((2.54D+00)**3)* dens * 1.0D-3
do 555 k = 1, n
   tempa = 4.0D/3.0D0 * xpid *((dia(k)*0.5D0)**3)
   mass(k) = tempa
555 continue
C
C This section to generate n-spheres of diameter 'diam'
C
2 psint1 = 0
x(1) = RAN(1x) * length
if((x(1).le. ra).or.(x(1).ge. length-ra)) go to 2
y(1) = RAN(ix) * height
if((y(1).le. ra).or.(y(1).ge. height-ra)) go to 6
z(1) = RAN(ix)*width
if ((z(1).le.ra).or.(z(1).ge.width-ra)) go to 5
xnew = RAN(ix) * length
if((xnew.le.ra).or.(xnew.ge.length-ra)) go to 3
ynew = RAN(ix) * height
if((ynew.le.ra).or.(ynew.ge.height-ra)) go to 7
znew=RAN(ix)*width
if ((znew.le.ra).or.(znew.ge.width-ra)) go to 8
isave = 1
if (x(1).le.xnew) isave = 2
Call GEOMCK(x,y,z, dia, xnew, ynew, znew, dmax, diam, isave, 1    2,0,mode,ier)
    if(ier.eq.-1) go to 3
Call AINSRT(xnew, x, 2, isave)
call AINSRT(ynew, y, 2, isave)
Call AINSRT(znew, z, 2, isave)

icount = 1
Kdim = 3
xtemp = RAN(ix) * length
if((xtemp .le. ra).or.(xtemp .ge. length-ra)) go to 96
ytemp = RAN(ix) * height
if((ytemp .le. ra).or.(ytemp .ge. height-ra)) go to 98
ztemp=RAN(ix)*width
if ((ztemp.le.ra).or.(ztemp.ge.width-ra)) go to 99
Call SEARCH(xtemp, x, Kdim, isave)
Call GEOMCK(x,y,z,dia,xtemp,ytemp,ztemp,dmax,diam,1              1, isave,Kdim,0,mode,ier)
    if (ier.eq.0) go to 91
C
    icount = icount + 1
        if (icount.gt.maxgen) go to 888
        go to 96
91 Call AINSRT(xtemp,x,Kdim,isave)
call AINSRT(ytemp,y,kdim,isave)
Call AINSRT(ztemp,z,kdim,isave)
kdim = kdim + 1
icount = icount + 1
if (icount.gt.maxgen) go to 888
if (kdim.le.n) then
    go to 96
endif
C
377 peo = ENERGY(0, n, mass, g, y, 0.0D0)
e(1) = peo
    if (kdim.gt.n) go to 885
C
C [The following statements will be executed if restrt = 'Yes'.
C File unit 31 contains the information from the previous
C run.]
C
110 open(unit=31,file='[sxp4639.mc3d]mc3d.res',status='old')
do 57 ir = 1, n
     read(31,901) x(ir), y(ir), z(ir)
     read(31,9801) dia(ir), mass(ir)
   57     continue
     read (31,*) psint1, peo
     read(31,*) icyc0
     read (31,*) dmax, dmin
     read(31,* ) yjump0
     read(31,919) pour
     e(1) = peo
     totpas = psint1
     icycle = 1
     yjump = yjump0
     if (yjump .ne. 0.0) pour = 'np'
     close(unit=31)
     rewind(unit=31)
     go to 885

C    ************************************************************************
C    * Start of the Simulation  *
C    ************************************************************************

C Begin moving spheres and generating new configurations

115 if (icyle .le. maxcyc) then
     ix = 1x + 2
     go to 767
else
     go to 998
endif

C -- Lift assembly of spheres by amplitude "yjump" --

767 do 116 i = 1, n
     y(i) = y(i) + yjump
   116     continue

C peo = ENERGY(0, n, mass, g, y, 0.0DO)
e(1) = peo

C mode = 'simulate'
ipass = 0
ipaspr = 0

120 iaccurr = 0
totpas = totpas + 1
ipass = ipass + 1
if (ipass .eq. 20) then
     Call SDEV(e, 20, ebrp, ebrdp)
ebr(1) = ebrp
endif
ipaspr = ipaspr + 1
iprint = 0

C ---- Start of Random Selection Procedure ----
temdim = n
do 3000 J = 1, n
    Temp ( J ) = J
    Trace ( J ) = J
    Origin ( J ) = J
3000 continue
15 if ( temdim .eq. 1 ) then
    select = 1
    ip = temp ( 1 )
    i = trace ( ip )
    temdim = 0
    go to 70
else if ( temdim .eq. 0 ) then
    go to 90
endif
C
4005 select = IFIX(temdim * RAN(1x))
if ( select .eq. 0 ) go to 4005
C
ip = temp ( select )
i = trace ( ip )
C
if ( select .lt. temdim ) then
    do 4000 J = select, temdim
        temp ( J ) = temp ( J + 1 )
4000 continue
endif
C
    temdim = temdim - 1
C
70 iprint = iprint + 1
    xnew = x(1) + delt * (1.0D+00-2.0D+00*RAN(1x))
ynew = y(1) + delt * (1.0D+00-2.0D+00*RAN(1x))
znew = z(1) + delt * (1.0D+00-2.0D+00*RAN(1x))
if (ynew .le. dia(1)*0.5D+00) go to 15
if ((ynew .gt. dia(1)*0.5D+00) .and.
1 (ynew .lt. height-dia(1)*0.5D+00)) then
    go to 65
endif
C
C
C
C
***************
* Impose Hard Vertical Walls *
***************
C
C
65 if (((pbc .eq. 'no').or.(pbc .eq. 'No').or.(pbc .eq. 'nO')
+ .or.(pbc .eq. 'NO')) then
    if (((xnew .gt. dia(1)*0.5DO).and.
1 (xnew .lt. length-dia(1)*0.5DO)).and.
2 ((znew .gt. dia(1)*0.5DO).and.
3 (znew .lt. width-dia(1)*0.5DO))) then
        Call SEARCH(xnew, x, ndim, isave)
        Call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
+ isave,ndim,i,mode,ier)
    if (ier .eq. -1) then
go to 15
else
  go to 69
endif
else
  go to 15
endif
endif

*****************************************************************************
* Imposing Periodic Boundary Conditions *
*****************************************************************************
if (((xnew.gt.0.0).and. (xnew.lt. dia(1)))).and.
1  (znew.ge.width+dia(1)*0.5DO)) then
  znw=znw-width
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
  *          isave,ndim,i,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew+length
  call SEARCH(xnew1,x,ndim,isave1)
  call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
  *          isave1,ndim,i,mode,ier)
  if (ier.eq.-1) go to 15
  znew1=znew+width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
  *          isave,ndim,i,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew+length
  znew1=znew+width
  call SEARCH(xnew1,x,ndim,isave1)
  call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
  *          isave1,ndim,i,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    go to 69
endif
if (ier.eq.-1) go to 15
xnewl=xnew+length
znewl=znew-width

call SEARCH(xnewl, x, ndim, isave1)
call GEOMCK(x,y,z,dia,xnewl,ynew,znewl,dmax,dia(1),
          isave1,ndim,1,mode,ier)
if (ier.eq.-1) then
  go to 15
else
  znew=znewl
  go to 69
endif
else if (xnew.gt.0.0).and.(xnew.lt.dia(1)).and.
1 ((znew.gt.width-dia(1)).and.(znew.lt.width))) then
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
          isave,ndim,1,mode,ier)
if (ier .eq. -1) go to 15
xnewl=xnew+length

call SEARCH(xnewl, x, ndim, isave1)
call GEOMCK(x,y,z,dia,xnewl,ynew,znewl,dmax,dia(1),
          isave1,ndim,1,mode,ier)
if (ier .eq. -1) go to 15
znewl=znew-width

call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znewl,dmax,dia(1),
          isave,ndim,1,mode,ier)
if (ier.eq.-1) go to 15
xnewl=xnew+length
znewl=znew-width

call SEARCH(xnewl,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnewl,ynew,znewl,dmax,dia(1),
          isave1,ndim,1,mode,ier)
if (ier.eq.-1) then
  go to 15
else
  go to 69
endif
else if (xnew.gt.0.0).and.(xnew.lt.dia(1)).and.
1 (znew.ge.dia(1)).and.(znew.le.width-dia(1))) then
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
          isave,ndim,1,mode,ier)
if (ier .eq. -1) go to 15
xnewl=xnew+length

call SEARCH(xnewl, x, ndim, isave1)
call GEOMCK(x,y,z,dia,xnewl,ynew,znew,dmax,dia(1),
          isave1,ndim,1,mode,ier)
if (ier .eq. -1) then
  go to 15
else

go to 69
endif
else if (((xnew.gt.0.0).and. (xnew.lt.dia(1))).and.
           ((znew.gt.0.0).and.(znew.lt.dia(1)))) then
   call SEARCH(xnew, x, ndim, isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
               isave,ndim,1,mode,ier)
   if (ier.eq. -1) go to 15
   xnew1 = length + xnew
   call SEARCH(xnew1, x, ndim, isave1)
   call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
               isave1,ndim,1,mode,ier)
   if (ier.eq. -1) go to 15
   znew1=znew+width
   call SEARCH(xnew, x, ndim, isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
               isave,ndim,1,mode,ier)
   if (ier.eq. -1) go to 15
   xnew1=xnew+length
   znew1=znew+width
   call SEARCH(xnew1,x,ndim,isave1)
   call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
               isave1,ndim,1,mode,ier)
   if (ier.eq. -1) then
      go to 15
   else
      go to 69
   endif
else if (((xnew.gt.0.0).and. (xnew.lt.dia(1))).and.
           ((znew.gt.-dia(1)*0.5D0).and.(znew.le.0.0))) then
   call SEARCH(xnew, x, ndim, isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
               isave,ndim,1,mode,ier)
   if (ier.eq. -1) go to 15
   znew1=znew+width
   call SEARCH(xnew, x, ndim, isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
               isave,ndim,1,mode,ier)
   if (ier.eq. -1) go to 15
   xnew1=xnew+length
   call SEARCH(xnew1,x,ndim,isave1)
   call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
               isave1,ndim,1,mode,ier)
   if (ier.eq. -1) go to 15
   xnew1=xnew+length
   znew1=znew+width
   call SEARCH(xnew1,x,ndim,isave1)
   call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
               isave1,ndim,1,mode,ier)
   if (ier.eq. -1) then
      go to 15
   else
      znew=znew1
go to 69
dendif
else if (((xnew.gt.0.0).and.(xnew.lt.dy(i))).and.
(znew.ge.0.5D0).then
  znew=znew+width
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
            isave,ndim,1,mode,ier)
  if (ier .eq. -1) go to 15
  xnew1=xnew+length
  call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
            isave1,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znew1=znew-width
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
            isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew+length
  znew1=znew-width
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
            isave1,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    go to 69
  endif
else if (((xnew.gt.length-dia(1))).and.(xnew.le.length)).and.
(znew.ge.width+dia(1)*0.5D0)) then
  znew=znew-width
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
            isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew+length
  call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
            isave1,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znew1=znew-width
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
            isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew+length
  znew1=znew-width
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
            isave1,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  endif
else
  go to 69
endif
else if (((xnew.gt.length-dia(1)).and.(xnew.le.length)).and.
1  ((znew.gt.width).and.(znew.lt.width+dia(1)*0.5DO)) then
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
*     isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
*     isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
*     isave1,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  znew1=znew-width
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
  isave1,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    znew=znew1
    go to 69
  endif
else if (((xnew.gt.length-dia(1)).and.(xnew.le.length)).and.
1  ((znew.gt.width-dia(1)).and.(znew.le.width))) then
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
*     isave,ndim,1,mode,ier)
  if (ier .eq. -1) go to 15
  xnewl = xnew-length
  call SEARCH(xnewl, x, ndim, isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
  isave1,ndim,1,mode,ier)
  if (ier .eq. -1) go to 15
  xnewl=xnew-length
  call SEARCH(xnew,l,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
  isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  znew1=znew-width
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
  isave1,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15

else
    go to 69
endif
else if (((xnew.gt.length-dia(1)).and.(xnew.le.length)).and.
1  ((znew.ge.dia(1)).and.(znew.le.width-dia(1)))) then
    call SEARCH(xnew, x, ndim, isave)
    call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
    * isave, ndim, i, mode, ier)
    if (ier.eq.-1) go to 15
    xnew1 = xnew - length
    call SEARCH(xnew1, x, ndim, isave1)
    call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
    * isave1, ndim, i, mode, ier)
    if (ier.eq.-1) then
        go to 15
    else
        go to 69
    endif
else if (((xnew.gt.length-dia(1)).and.(xnew.le.length)).and.
1  ((znew.gt.0.0).and.(znew.lt.dia(1)))) then
    call SEARCH(xnew, x, ndim, isave)
    call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
    * isave, ndim, i, mode, ier)
    if (ier .eq. -1) go to 15
    xnew1 = xnew-length
    call SEARCH(xnew1, x, ndim, isave1)
    call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
    * isave1, ndim, i, mode, ier)
    if (ier .eq. -1) go to 15
    znew1 = znew+width
    call SEARCH(xnew, x, ndim, isave)
    call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(1),
    * isave, ndim, i, mode, ier)
    if (ier .eq. -1) go to 15
    xnew1 = xnew-length
    znew1 = znew+width
    call SEARCH(xnew1, x, ndim, isave1)
    call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(1),
    * isave1, ndim, i, mode, ier)
    if (ier .eq. -1) then
        go to 15
    else
        go to 69
    endif
else if (((xnew.gt.length-dia(1)).and.(xnew.le.length)).and.
1  ((znew.gt.-dia(1)*0.5DO).and.(znew.le.0.0))) then
    call SEARCH(xnew, x, ndim, isave)
    call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
    * isave, ndim, i, mode, ier)
    if (ier .eq. -1) go to 15
    znew1 = znew+width
    call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
    isave,ndim,i,mode,ier)
  +
  if (ier .eq. -1) go to 15
  xnewl=xnew-length
  call SEARCH(xnewl,x,ndim,isave1)
  call GEOMCK(x,y,z,dia,xnewl,ynew,znew,dmax,dia(1),
    isave1,ndim,i,mode,ier)
  +
  if (ier .eq. -1) go to 15
  xnewl=xnew-length
  znewl=znew+width
  call SEARCH(xnewl,x,ndim,isave1)
  call GEOMCK(x,y,z,dia,xnewl,ynew,znew1,dmax,dia(1),
    isave1,ndim,i,mode,ier)
  if (ier .eq. -1) then
    go to 15
  else
    znew=znew1
    go to 69
  endif

C
else if (((xnew.gt.length-dia(1)).and.(xnew.le.length)).and.
  (znew.le.-dia(1)*0.5DO)) then
  znew=znew+width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
    isave,ndim,i,mode,ier)
  +
  if (ier .eq. -1) go to 15
  xnewl=xnew-length
  call SEARCH(xnewl,x,ndim,isave1)
  call GEOMCK(x,y,z,dia,xnewl,ynew,znew1,dmax,dia(1),
    isave1,ndim,i,mode,ier)
  +
  if (ier .eq. -1) go to 15
  znewl=znew+width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
    isave,ndim,i,mode,ier)
  +
  if (ier .eq. -1) go to 15
  xnewl=xnew-length
  znewl=znew-width
  call SEARCH(xnewl,x,ndim,isave1)
  call GEOMCK(x,y,z,dia,xnewl,ynew,znew1,dmax,dia(1),
    isave1,ndim,i,mode,ier)
  if (ier .eq. -1) then
    go to 15
  else
    go to 69
  endif

C
else if (((xnew.gt.-dia(1)*0.5DO).and.(xnew.le.0.0)).and.
  (znew.ge.width+dia(1)*0.5DO)) then
  znew=znew-width
  xnew=xnew+length
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),

80
* if (ier.eq.-1) go to 15
  xnew1=xnew-length
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
               isave1, ndim, i, mode, ier)
* if (ier.eq.-1) go to 15
  znew1=znew+width
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(1),
               isave, ndim, i, mode, ier)
* if (ier.eq.-1) go to 15
  xnew1=xnew-length
  znew1=znew-width
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(1),
               isave1, ndim, i, mode, ier)
* if (ier.eq.-1) then
  go to 15
else
  go to 69
endif

else if (((xnew.gt.-dia(1)*0.5D0).and.(xnew.le.0.0)).and.
1  ((znew.gt.width).and.(znew.lt.width+dia(1)*0.5D0))) then
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
               isave, ndim, i, mode, ier)
* if (ier .eq. -1) go to 15
  znew1=znew-width
  xnew1=xnew+length
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(1),
               isave1, ndim, i, mode, ier)
* if(ier.eq.-1) go to 15
  xnew1=xnew-length
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
               isave1, ndim, i, mode, ier)
* if (ier.eq.-1) go to 15
  znew1=znew-width
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(1),
               isave, ndim, i, mode, ier)
* if (ier.eq.-1) then
  go to 15
else
  znew=znew1
  xnew=xnew1
  isave=isave1
  go to 69
endif

else if (((xnew.gt.-dia(1)*0.5D0).and.(xnew.le.0.0)).and.
((znew.gt.width-dia(1)).and.(znew.le.width))) then
   call SEARCH(xnew, x, ndim, isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
                isave,ndim,1,mode,ier)
   if (ier.eq.-1) go to 15
   xnew1=xnew+length
   call SEARCH(xnew1,x,ndim,isave1)
   call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
                isave1,ndim,1,mode,ier)
   if (ier.eq.-1) go to 15
   znew1=znew-width
   call SEARCH(xnew, x, ndim, isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
                isave,ndim,1,mode,ier)
   if (ier.eq.-1) go to 15
   xnew1=xnew+length
   znew1=znew-width
   call SEARCH(xnew1,x,ndim,isave1)
   call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
                isave1,ndim,1,mode,ier)
   if (ier.eq.-1) then
      go to 15
   else
      xnew=xnew1
      isave=isave1
      go to 69
   endif
   else if (((xnew.gt.-dia(1)*0.5DO).and.(xnew.le.0.0)).and.
((znew.ge.dia(1)).and.(znew.le.width-dia(1)))) then
   call SEARCH(xnew,x,ndim,isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
                isave,ndim,1,mode,ier)
   if (ier.eq.-1) go to 15
   xnew1 = xnew + length
   call SEARCH(xnew1,x,ndim,isave1)
   call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
                isave1,ndim,1,mode,ier)
   if (ier.eq.-1) then
      go to 15
   else
      xnew = xnew1
      isave = isave1
      go to 69
   endif
   else if (((xnew.gt.-dia(1)*0.5DO).and.(xnew.le.0.0)).and.
((znew.gt.0.0).and.(znew.lt.dia(1)))) then
   call SEARCH(xnew, x, ndim, isave)
   call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
                isave,ndim,1,mode,ier)
   if (ier .eq. -1) go to 15
   xnew1=xnew+length
   call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(i),
isave1, ndim, 1, mode, ier)

if (ier.eq.-1) go to 15
znew1=znew+width
call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(i),
isave, ndim, 1, mode, ier)

if (ier.eq.-1) go to 15
znew1=znew+length
xnew1=xnew+length
call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(i),
isave1, ndim, 1, mode, ier)

if (ier.eq.-1) then
  go to 15
else
  xnew=xnew1
  isave=isave1
  go to 69
endif

C

else if (((xnew.gt.-dia(i)*0.5DO).and.(xnew.le.0.0)).and.
1 (((znew.gt.-dia(i)*0.5DO).and.(znew.le.0.0))) then
  call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(i),
isave, ndim, 1, mode, ier)

if (ier.eq.-1) go to 15
xnew1=xnew+length
znew1=znew+width
call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(i),
isave1, ndim, 1, mode, ier)

if (ier.eq.-1) go to 15
xnew1=xnew+length
call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(i),
isave1, ndim, 1, mode, ier)

if (ier.eq.-1) go to 15
znew1=znew+width
call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(i),
isave, ndim, 1, mode, ier)

if (ier.eq.-1) then
  go to 15
else
  xnew=xnew1
  znew=znew1
  isave=isave1
  go to 69
endif

C

else if (((xnew.gt.-dia(i)*0.5DO).and.(xnew.le.0.0)).and.
1 (znew.le.-dia(i)*0.5DO)) then
  znew=znew+width
xnew=xnew+length
call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
isave, ndim, i, mode, ier)

if (ier.eq.-1) go to 15
xnew1=xnew-length
call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
isave1, ndim, i, mode, ier)

if (ier.eq.-1) go to 15
znew1=znew-width
call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(1),
isave, ndim, i, mode, ier)

if (ier.eq.-1) go to 15
xnew1=xnew-length
znew1=znew-width
call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(1),
isave1, ndim, i, mode, ier)

if (ier.eq.-1) then
go to 15
else
go to 69
endif

C else if (((xnew.gt.length).and.(xnew.lt.length+dia(1)*0.5DO)).and.
(1
(znew.ge.width+dia(1)*0.5DO)) then
znew=znew-width
xnew=xnew-length
call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
isave, ndim, i, mode, ier)

if (ier.eq.-1) go to 15
xnew1=xnew+length
call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
isave1, ndim, i, mode, ier)

if (ier.eq.-1) go to 15
znew1=znew+width
 call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(1),
isave, ndim, i, mode, ier)

if (ier.eq.-1) go to 15
xnew1=xnew+length
znew1=znew+width
call SEARCH(xnew1, x, ndim, isave1)
call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(1),
isave1, ndim, i, mode, ier)

if (ier.eq.-1) then
go to 15
else
go to 69
endif
else if (((xnew.gt.length).and.(xnew.lt.length+dia(1)*0.5DO)).and. 
1 ((znew.gt.width).and.(znew.lt.width+dia(1)*0.5DO))) then 
call SEARCH(xnew, x, ndim, isave) 
call GEMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1), 
+ isave,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
xnew1=xnew-length 
znew1=znew-width 
call SEARCH(xnew1, x, ndim, isave1) 
call GEMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1), 
+ isave1,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
xnew1=xnew-length 
call SEARCH(xnew1, x, ndim, isave1) 
call GEMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1), 
+ isave1,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
znew1=znew-width 
call SEARCH(xnew,x,ndim,isave) 
call GEMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1), 
+ isave,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
xnew1=xnew-length 
call SEARCH(xnew1,x,ndim,isave1) 
call GEMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1), 
+ isave1,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
znew1=znew-width 
call SEARCH(xnew,x,ndim,isave) 
call GEMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1), 
+ isave,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
xnew1=xnew-length 
znew1=znew-width 
call SEARCH(xnew1,x,ndim,isave1) 
call GEMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1), 
+ isave1,ndim,1,mode,ier) 
if (ier.eq.-1) then 
go to 15 
else 
xnew=xnew1 
znew=znew1 
isave=isave1 
go to 69 
endif 

else if (((xnew.gt.length).and.(xnew.lt.length+dia(1)*0.5DO)).and. 
1 ((znew.gt.width-dia(1)).and.(znew.le.width))) then 
call SEARCH(xnew, x, ndim, isave) 
call GEMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1), 
+ isave,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
xnew1=xnew-length 
call SEARCH(xnew1,x,ndim,isave1) 
call GEMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1), 
+ isave1,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
znew1=znew-width 
call SEARCH(xnew,x,ndim,isave) 
call GEMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1), 
+ isave,ndim,1,mode,ier) 
if (ier.eq.-1) go to 15 
xnew1=xnew-length 
znew1=znew-width 
call SEARCH(xnew1,x,ndim,isave1) 
call GEMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1), 
+ isave1,ndim,1,mode,ier) 
if (ier.eq.-1) then 
go to 15 

else
  xnew=xnew1
  isave=isave1
  go to 69
endif

C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(1)*0.5DO)).and. 1
  ((znew.ge.dia(1)).and.(znew.le.width-dia(1))).)
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
               isave, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  xnew1 = xnew - length
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
               isave1, ndim, 1, mode, ier)
  if (ier.eq.-1) then
    go to 15
  else
    xnew=xnew1
    isave=isave1
    go to 69
  endif

C
else if (((xnew.gt.length).and.
          (xnew.lt.length+dia(1)*0.5DO)).and.
          (znew.gt.0.0).and.(znew.lt.dia(1))
          )
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(1),
               isave, ndim, i, mode, ier)
  if (ier .eq. -1) go to 15
  xnew1=xnew-length
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(1),
               isave1, ndim, 1, mode, ier)
  if (ier.eq.-1) go to 15
  znew1=znew+width
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(1),
               isave, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew-length
  znew1=znew+width
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(1),
               isave1, ndim, 1, mode, ier)
  if (ier.eq.-1) then
    go to 15
  else
    xnew=xnew1
    isave=isave1
    go to 69
  endif

C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(1)*0.5DO)).and.
1
((znew.gt.-dia(1)*0.5DO).and.(znew.le.0.0))) then
    call SEARCH(xnew, x, ndim, isave)
    call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
    isave,ndim,1,mode,ier)
    if (ier.eq.-1) go to 15
    znew1=znew+width
    xnew1=xnew-length
    call SEARCH(xnew1,x,ndim,isavel)
    call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
    isavel,ndim,1,mode,ier)
    if (ier.eq.-1) go to 15
    xnew1=xnew-length
    call SEARCH(xnew1,x,ndim,isavel)
    call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
    isavel,ndim,1,mode,ier)
    if (ier.eq.-1) go to 15
    znew1=znew+width
    call SEARCH(xnew,x,ndim,isave)
    call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
    isave,ndim,1,mode,ier)
else
    xnew=xnew1
    znew=znew1
    isave=isavel
go to 69
endif
C
else if (((xnew.gt.length).and.(xnew.lt.length+dia(1)*0.5DO)).and.
1
(znew.le.-dia(1)*0.5DO)) then
    xnew=xnew-length
    znew=znew+width
    call SEARCH(xnew,x,ndim,isave)
    call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
    isave,ndim,1,mode,ier)
    if (ier.eq.-1) go to 15
    xnew1=xnew+length
    call SEARCH(xnew1,x,ndim,isavel)
    call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
    isavel,ndim,1,mode,ier)
    if (ier.eq.-1) go to 15
    znew1=znew+width
    call SEARCH(xnew,x,ndim,isave)
    call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
    isave,ndim,1,mode,ier)
    if (ier.eq.-1) go to 15
    xnew1=xnew+length
    znew1=znew-width
    call SEARCH(xnew1,x,ndim,isavel)
    call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
    isavel,ndim,1,mode,ier)
    if (ier.eq.-1) then

87
go to 15
else
go to 69
endif

C
else if (((xnew.le.-dia(1)*0.5D0).and.
1 (znew.ge.width+dia(1)*0.5D0)) then
xnew=xnew+length
znew=znew+width
call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,diag,xnew,ynew,znew,dmax,dia(1),
    isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  call SEARCH(xnewl,x,ndim,isavel)
call GEOMCK(x,y,z,diag,xnewl,ynewl,znewl,dmax,dia(1),
    isavel,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znewl=znew+width
  call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,diag,xnew,ynew,znew,dmax,dia(1),
    isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  znewl=znew+width
call SEARCH(xnewl,x,ndim,isavel)
call GEOMCK(x,y,z,diag,xnewl,ynewl,znewl,dmax,dia(1),
    isavel,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    go to 69
endif

C
else if (((xnew.le.-dia(1)*0.5D0).and.
1 ((znew.gt.width).and.(znew.lt.width+dia(1)*0.5D0))) then
xnew=xnew+length
znew=znew+width
call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,diag,xnew,ynew,znew,dmax,dia(1),
    isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  call SEARCH(xnewl,x,ndim,isavel)
call GEOMCK(x,y,z,diag,xnewl,ynewl,znewl,dmax,dia(1),
    isavel,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znewl=znew+width
call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,diag,xnew,ynew,znewl,dmax,dia(1),
    isavel,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnewl=xnew-length
  znewl=znew+width
call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
           isave1,ndim,1,mode,ier)
if (ier.eq.-1) then
  go to 15
else
  go to 69
endif

c
else if ((xnew.le.-dia(1)*0.5D0).and.
1 ((znew.gt.width-dia(1)).and.(znew.le.width))) then
  xnew=xnew+length
  call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
             isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew-length
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
             isave1,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znew1=znew-length
  call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
             isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew-length
  znew1=znew-length
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
             isave1,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    go to 69
  endif
C
else if ((xnew.le.-dia(1)*0.5D0).and.((znew.ge.dia(1)).and.
1 (znew.le.width-dia(1)))) then
  xnew = xnew + length
  call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
             isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew-length
  call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
             isave1,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    go to 69
  endif
C
else if ((xnew.le.-dia(1)*0.5DO).and.
+ ((znew.gt.0.0).and.(znew.lt.dia(1)))) then
    xnew=xnew+length
    call SEARCH(xnew, x, ndim, isave)
    call GOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
        isave,ndim,i,mode,ier)
    if (ier.eq.-1) go to 15
    xnew1=xnew-length
    call SEARCH(xnew1, x, ndim, isave1)
    call GOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
        isave1,ndim,i,mode,ier)
    if (ier.eq.-1) go to 15
    znew1=znew+width
    call SEARCH(znew1, x, ndim, isave)
    call GOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
        isave,ndim,i,mode,ier)
    if (ier.eq.-1) go to 15
    xnew1=xnew-length
    znew1=znew+width
    call SEARCH(xnew1, x, ndim, isave1)
    call GOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
        isave1,ndim,i,mode,ier)
    if (ier.eq.-1) then
        go to 15
    else
        go to 69
    endif
else if ((xnew.le.-dia(1)*0.5DO).and.
+ ((znew.gt.-dia(1)*0.5DO).and.(znew.le.0.0))) then
    xnew=xnew+length
    znew=znew+width
    call SEARCH(xnew, x, ndim, isave)
    call GOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
        isave,ndim,i,mode,ier)
    if (ier.eq.-1) go to 15
    xnew1=xnew-length
    call SEARCH(xnew1, x, ndim, isave1)
    call GOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
        isave1,ndim,i,mode,ier)
    if (ier.eq.-1) go to 15
    znew1=znew-width
    call SEARCH(xnew,znew,x,ndim,isave)
    call GOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
        isave,ndim,i,mode,ier)
    if (ier.eq.-1) go to 15
    xnew1=xnew-length
    znew1=znew-width
    call SEARCH(xnew1,x,ndim,isave1)
    call GOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
        isave1,ndim,i,mode,ier)
    if (ier.eq.-1) then
        go to 15
    else

else if ( (xnew .le. -dia(i) * 0.5D0) .and. 
(xnew .le. -dia(i) * 0.5D0) ) then 
xnew = xnew + length 
znew = xnew + width 
call SEARCH(xnew, x, ndim, isave) 
call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(i), 
               isave, ndim, i, mode, ier) 
* 
if (ier .eq. -1) go to 15 
xnew1 = xnew + length 
call SEARCH(xnew1, x, ndim, isave1) 
call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(i), 
               isave1, ndim, i, mode, ier) 
* 
if (ier .eq. -1) go to 15 
znew1 = xnew + width 
call SEARCH(xnew, x, ndim, isave) 
call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(i), 
               isave, ndim, i, mode, ier) 
* 
if (ier .eq. -1) go to 15 
xnew1 = xnew + length 
znew1 = xnew + width 
call SEARCH(xnew1, x, ndim, isave1) 
call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(i), 
               isave1, ndim, i, mode, ier) 
* 
if (ier .eq. -1) then 
go to 15 
else 
go to 69 
endif

else if ( (xnew .ge. length + dia(i) * 0.5D0) .and. 
(znew .ge. width + dia(i) * 0.5D0) ) then 
xnew = xnew + length 
znew = xnew + width 
call SEARCH(xnew, x, ndim, isave) 
call GEOMCK(x, y, z, dia, xnew, ynew, znew, dmax, dia(i), 
               isave, ndim, i, mode, ier) 
* 
if (ier .eq. -1) go to 15 
xnew1 = xnew + length 
call SEARCH(xnew1, x, ndim, isave1) 
call GEOMCK(x, y, z, dia, xnew1, ynew, znew, dmax, dia(i), 
               isave1, ndim, i, mode, ier) 
* 
if (ier .eq. -1) go to 15 
znew1 = xnew + width 
call SEARCH(xnew, x, ndim, isave) 
call GEOMCK(x, y, z, dia, xnew, ynew, znew1, dmax, dia(i), 
               isave, ndim, i, mode, ier) 
* 
xnew1 = xnew + length 
znew1 = xnew + width 
call SEARCH(xnew1, x, ndim, isave1) 
call GEOMCK(x, y, z, dia, xnew1, ynew, znew1, dmax, dia(i), 
               isave1, ndim, i, mode, ier)
if (ier.eq.-1) then
  go to 15
else
  go to 69
endif

C
else if (((xnew.ge.length+dia(1)*0.5D0).and.
1     ((znew.gt.width).and.(znew.lt.width+dia(1)*0.5D0))) then
  xnew=xnew-length
  znew=znew-width
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x,y,z, dia, xnew, ynew, znew, dmax, dia(1),
              isave, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  xnew=xnew+length
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x,y,z, dia, xnew1, ynew, znew, dmax, dia(1),
              isave1, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  znew=znew-width
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x,y,z, dia, xnew, ynew, znew1, dmax, dia(1),
              isave, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew+length
  znew1=znew+width
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x,y,z, dia, xnew1, ynew, znew1, dmax, dia(1),
              isave1, ndim, i, mode, ier)
  if (ier.eq.-1) then
    go to 15
else
  go to 69
endif
C
else if (((xnew.ge.length+dia(1)*0.5D0).and.
1     ((znew.gt.width-dia(1)).and.(znew.lt.width))) then
  xnew=xnew-length
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x,y,z, dia, xnew, ynew, znew, dmax, dia(1),
              isave, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  xnew=xnew+length
  call SEARCH(xnew1, x, ndim, isave1)
  call GEOMCK(x,y,z, dia, xnew1, ynew, znew, dmax, dia(1),
              isave1, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  znew=znew-width
  call SEARCH(xnew, x, ndim, isave)
  call GEOMCK(x,y,z, dia, xnew, ynew, znew1, dmax, dia(1),
              isave, ndim, i, mode, ier)
  if (ier.eq.-1) go to 15
  xnew1=xnew+length
  znew1=znew-width

92
call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
isave1,ndim,1,mode,ier)

if (ler.eq.-1) then
   go to 15
else
   go to 69
endif

c
else if (((xnew.ge.length+dia(1)*0.5DO).and.((znew.ge.dia(1))
   .and.(znew.le.width-dia(1)))) then
xnew = xnew - length
call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
isave,ndim,1,mode,ier)

if (ler.eq.-1) go to 15
xnew1=xnew+length
call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
isave1,ndim,1,mode,ier)

if (ler.eq.-1) then
   go to 15.
else
   go to 69
endif

c
else if (((xnew.ge.length+dia(1)*0.5DO).and.((znew.gt.0.0).and.(znew.lt.dia(1)))) then
xnew=xnew-length
call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
isave,ndim,1,mode,ier)

if (ler.eq.-1) go to 15
xnew1=xnew+length
   call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew,dmax,dia(1),
isave1,ndim,1,mode,ier)

if (ler.eq.-1) go to 15
znew1=znew+width
call SEARCH(xnew,x,ndim,isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(1),
isave,ndim,1,mode,ier)

if (ler.eq.-1) go to 15
xnew1=xnew+length
znew1=znew+width
call SEARCH(xnew1,x,ndim,isave1)
call GEOMCK(x,y,z,dia,xnew1,ynew,znew1,dmax,dia(1),
isave1,ndim,1,mode,ier)

if (ler.eq.-1) then
   go to 15
else
   go to 69
endif
else if (((xnew.ge.length+dia(1)*0.5D0).and.
1   ((znew.gt.-dia(1)*0.5D0).and.(znew.le.0.0))) then
xnew=xnew-length
znew=znew+width
call SEARCH(xnew, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
+   isave,ndim,i,mode,ier)
   if (ier.eq.-1) go to 15
xnewl=xnew+length
znewl=znew-width
call SEARCH(xnewl,x,ndim1,isave1)
call GEOMCK(x,y,z,dia,xnewl,ynewl,znewl,dmax,dia(1),
+   isave1,ndim1,i,mode,ier)
   if (ier.eq.-1) go to 15
znewl=znew-width
call SEARCH(xnew,x,ndim,ise)
call GEOMCK(x,y,z,dia,xnew,ynew,znewl,dmax,dia(1),
   isave,ndim,i,mode,ier)
   if (ier.eq.-1) go to 15
xnewl=xnew+length
znewl=znew-width
call SEARCH(xnewl,x,ndim1,isave1)
call GEOMCK(x,y,z,dia,xnewl,ynewl,znewl,dmax,dia(1),
   isave1,ndim1,i,mode,ier)
   if (ier.eq.-1) then
   go to 15
else
   go to 69
endif

else if (((xnew.ge.length+dia(1)*0.5D0).and.
+   (znew.le.-dia(1)*0.5D0)) then
xnew=xnew-length
znew=znew+width
call SEARCH(xnewl, x, ndim, isave)
call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
   isave,ndim,i,mode,ier)
   if (ier.eq.-1) go to 15
xnewl=xnew+length
call SEARCH(xnewl,x,ndim1,isave1)
call GEOMCK(x,y,z,dia,xnewl,ynewl,znewl,dmax,dia(1),
   isave1,ndim1,i,mode,ier)
   if (ier.eq.-1) go to 15
znewl=znew-width
call SEARCH(xnewl,x,ndim1,isave1)
call GEOMCK(x,y,z,dia,xnewl,ynewl,znewl,dmax,dia(1),
   isave1,ndim1,i,mode,ier)
   if (ier.eq.-1) go to 15
xnewl=xnew+length
znewl=znew-width
call SEARCH(xnewl,x,ndim1,isave1)
call GEOMCK(x,y,z,dia,xnewl,ynewl,znewl,dmax,dia(1),
   isave1,ndim1,i,mode,ier)
   if (ier.eq.-1) then
   go to 15

94
else
go to 69
endif

c
else if (((xnew.ge.dia(1))).and.(xnew.le.length-dia(1)))
  and.(znew.ge.width+dia(1)*0.5DO)) then
  znew=znew-width
  call SEARCH(xnew,x,ndim,isave)
  call GOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
               isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znewl=znew+width
  call SEARCH(xnew,x,ndim,isave)
  call GOMCK(x,y,z,dia,xnew,ynew,znewl,dmax,dia(1),
               isave,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    go to 69
  endif

c
else if (((xnew.ge.dia(1))).and.(xnew.le.length-dia(1))).and.
  ((znew.gt.width).and.(znew.lt.width+dia(1)*0.5DO))) then
  call SEARCH(xnew,x,ndim,isave)
  call GOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
               isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znewl = znew - width
  call SEARCH(xnew,x,ndim,isave)
  call GOMCK(x,y,z,dia,xnew,ynew,znewl,dmax,dia(1),
               isave,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    znew = znewl
    go to 69
  endif

c
else if (((xnew.ge.dia(1))).and.(xnew.le.length-dia(1))).and.
  ((znew.gt.width-dia(1))).and.(znew.le.width))) then
  call SEARCH(xnew,x,ndim,isave)
  call GOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
               isave,ndim,1,mode,ier)
  if (ier.eq.-1) go to 15
  znewl = znew - width
  call SEARCH(xnew,x,ndim,isave)
  call GOMCK(x,y,z,dia,xnew,ynew,znewl,dmax,dia(1),
               isave,ndim,1,mode,ier)
  if (ier.eq.-1) then
    go to 15
  else
    go to 69
  endif

C
else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
((znew.gt.0.0).and.(znew.lt.dia(i)))) then
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
               isave,ndim,i,mode,ier)
if (ier.eq.-1) go to 15
znew1 = znew + width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
               isave,ndim,i,mode,ier)
if (ier.eq.-1) go to 15
znew2=znew-width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew2,dmax,dia(i),
               isave,ndim,i,mode,ier)
else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
((znew.gt.-dia(i)*0.5DO).and.(znew.le.0.0))) then
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
               isave,ndim,i,mode,ier)
if (ier.eq.-1) go to 15
znew1 = znew + width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
               isave,ndim,i,mode,ier)
if (ier.eq.-1) then
  go to 15
else
  znew = znew1
  go to 69
endif
  C
else if (((xnew.ge.dia(i)).and.(xnew.le.length-dia(i))).and.
((znew.le.-dia(i)*0.5DO)) then
  znew = znew + width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew,dmax,dia(i),
               isave,ndim,i,mode,ier)
if (ier.eq.-1) go to 15
znew1=znew-width
  call SEARCH(xnew,x,ndim,isave)
  call GEOMCK(x,y,z,dia,xnew,ynew,znew1,dmax,dia(i),
               isave,ndim,i,mode,ier)
if (ier.eq.-1) then
  go to 15
else
  go to 69
endif
  C
else
    call SEARCH(xnew,x,ndim,1save)
    call GEMV(x,y,z,dia,xnew,ynew,znew,dmax,dia(1),
               isave,ndim,i,mode,ier)
    if (ier.eq. -1) then
        go to 15
    else
        go to 69
    endif
endif

******************************************************************************

*Calculating new ENERGY of the system*
******************************************************************************

69  pe = ENERGY(i, n, mass, g, y, ynew)
    dpe = pe - peo
    if (dpe .gt. 0.0D0) go to 80
75  iaccept = iaccept + 1
    peo = pe
C
C    ---- Core section of random selection ------
C
C    tempd = dia ( i )
    tempm = mass ( i )
C
C    if ( isave .gt. 1 ) isave = isave - 1
C
C    IF ( i .lt. isave ) THEN
C
C    ---- Update arrays x, y, z, dia, & mass ------
C
C    jr = isave - 1
    do 9100 j = 1, jr
       ij = 1 + j
       ijm1 = ij - 1
       x ( ijm1 ) = x ( ij )
       y ( ijm1 ) = y ( ij )
       z ( ijm1 ) = z ( ij )
       dia ( ijm1 ) = dia ( ij )
       mass ( ijm1 ) = mass ( ij )
9100    continue
C
C    ---- Update arrays trace and origin ------
C
C    ior = Origin ( 1 )
    upbd = ior
    lwbd = ior
    do 5000 j = i+1, isave
       jor = Origin ( j )
       if ( jor .gt. upbd ) then
          upbd = jor
          go to 5000
         endif
5000   continue
endif
        if ( jor .lt. lwbd ) lwbd = jor

5000      continue
C
    do 6000  j = lwbd, upbd 
          jtr = trace ( j )
          if ( jtr .le. 1 ) go to 6000
          if ( jtr .gt. isave ) go to 6000
          trace ( j ) = jtr - 1
          origin(trace(j)) = j

6000      continue
C
        trace ( ip ) = isave
        origin ( isave ) = ip
C
        else if ( i .gt. isave ) then
C
        c----- Update arrays x, y, z, dia, and mass ----- 
C
        jr = i - isave
        do 9200  j = 1, jr 
          1j = i - j 
          1jp1 = 1j + 1 
          x ( 1jp1 ) = x ( 1j )

          y ( 1jp1 ) = y ( 1j )
          z ( 1jp1 ) = z ( 1j )
          dia ( 1jp1 ) = dia ( 1j )
          mass ( 1jp1 ) = mass ( 1j )

9200     continue
C
        c----- Update arrays trace and origin ----- 
C
        ior = origin ( isave )
        upbd = ior
        lwbd = ior
        do 7000  j = isave + 1, i
          jor = origin ( j )
          if ( jor .gt. upbd ) then
            upbd = jor
            go to 7000
        endif
          if ( jor .lt. lwbd ) lwbd = jor

7000      continue
C
    do 8000  j = lwbd, upbd 
          jtr = trace ( j )
          if ( jtr .ge. 1 ) go to 8000
          if ( jtr .lt. isave ) go to 8000
          trace ( j ) = jtr + 1
          origin ( trace ( j ) ) = j

8000     continue
C
        trace ( ip ) = isave

98
origin ( isave ) = ip

ENDIF

c
x ( isave ) = xnew
y ( isave ) = ynew
z ( isave ) = znew
dia ( isave ) = tempd
mass ( isave ) = tempm

c ****** end of random selection ******
c
if ( iprint .ne. iterpr ) go to 15
iprint = 0
write(9,205) ipass
write(9,206)
write(9,203)
c  do 305 Kp = 1, n
c  write(9,204) Kp, x(Kp), y(kp), z(kp)
c305  continue
   go to 15
C
80  bd = beta * dpe
if (DABS(bd) .ge. 170.0D+00) go to 15
pr = DEXP(-bd)
if (RAN(ix) .le. pr) then
   go to 75
else
   go to 15
  endif
C
90  e(ipass+1) = peo
perctg = DFLOAT(iaccpt)/DFLOAT(n) * 100.0D+00
C
After 10,000 passes, Set the value of delta as 0.06E-02
C
C
if (perctg .lt. 3.0D+00) then
   delt = delt * 0.75D+00
  endif
C
C
***************
C
*Output pass data*
C
***************
C
IF (ipaspr .eq. passpr) THEN
ipass1 = ipass + psint1
ipaspr = 0
icy = icycle + icyc0
write(9,220) yjump
if (pour .eq. 'pour') icy=0
write(9,199) icy
write(9,208) ipass1
write(9,209) e(ipass+1)
write(9,210) perctg
write(9,206)
WRITE(36,*) icy
WRITE(36,*) n
WRITE(36,*) ipass1
WRITE(36,*) beta
WRITE(36,*) e(ipass+1)
WRITE(36,*) dmin
IF (MOD(ipass,100).EQ.0.0) THEN
   ip1 = ipass + 1
   ip2 = ipass/100
   CALL SDEV(e, ip1, emean, edev)
   ebr(ip2) = emean
   ebrat = ebr(ip2)/ebrp
   IF (icycle.EQ.0) THEN
      ebrat = 1.0D0
      ebrp = ebr(ip2)
   ENDIF
   IF (icycle.EQ.0) THEN
      ipass1 = totpas
      psint1 = totpas
      icy = icycle + icyc0
      WRITE(9,220) yjump
      IF (pour.EQ.0) THEN
         icy = 0
         WRITE(9,199) icy
         WRITE(9,208) ipass1
         WRITE(9,209) e(mp1)
         WRITE(9,210) perctg
         WRITE(9,206)
         WRITE(9,203)
         WRITE(9,204) x(jp), y(jp), z(jp), dia(jp), mass(jp)
         WRITE(9,203) icy
         WRITE(9,203) n
         WRITE(9,203) ipass1
write(36,*) beta
write(36,*) e(mp1)
write(36,*) dmin

continue

C
C Compute mean and standard deviation of ENERGY array
C and store values in "emean" and "edev".

Call SDEV(e, mp1, emean, edev)
write(9,907) emean, edev
write(9,916)
1cycle = 1cycle + 1
ebrat = 0.0D0
ebrp = 0.0D0
ebrpdp = 0.0D0
nerg = MOD(maxpas,100) + 5

do 713 inr = 1, nerg
ebr(inr) = 0.0D0

713 continue

if (.not. pour) then
[Shaking Cycle completed: Begin new cycle.]
go to 115
else
[Pouring completed.]
go to 998
endif

ELSE

[Cycle not completed.]
[Continue until (1-eps) <= ebrat <= (1+eps)]
go to 120
ENDIF

... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

Initial Data and Output formats

Call IDATE(month, iday, iyear)
write(9,201)
format(1h,'MAIN: THE MAXIMUM NUMBER OF TRIALS TO GENERATE
* THE INITIAL DISTRIBUTION =',i7,' HAS BEEN EXCEEDED.')
format(16X,'MONTE CARLO SIMULATION SHAKING SEGREGATION',/)
write(9,219) month, iday, iyear
write(9,220) yjump
write(9,199) 1 cyc0
write(9,202) psint1
202 format(11X,'STARTING COORDINATES OF SPHERES, PASS NO: ',i8
+ '/',6X,'-',i8,'/)

if (psint1.eq. 0) then
write(36,*) ratio
write(36,*) amp, etcut
write(36,*) n

101
write(36,*) beta
write(36,*) peo
write(36,*) dmin
    do 405 ij = 1, n
       write(36,*) x(ij), y(ij), z(ij), dia(ij)
c405    continue
dendif
write(9,203)
203  format(4X,'Sphere',4x,'X',10x,'Y',10X,'Z',6x,'Diameter',
      + 3x,'Mass',/)
dc    do 300 K = 1, n
       write(9,204) k, x(K), y(K), z(K), dia(K), mass(K)
c300    continue
write(9,209) peo
write(9,211) delt
204  format(2X,I5,1x,3(D11.5,1X),2x,D10.4,1x,D11.5)
199  format('///,26X,'CYCLE NUMBER:',I7)
205  format('///,9X,'SPHERE COORDINATES OF A CONFIGURATION FOR 
      + PASS NO.:',I6)
208  format('7X,' 
      + ')
208  format('/',9X,'SPHERE COORDINATES AT PASS NUMBER:',I9)
209  format('/',9X,'Configuration ENERGY =',D20.7,1x,
      + 'Newton-meters')
210  format(9X,'Percentage of moves accepted: ',D11.4)
211  format(9X,'maximum absolute value of particle displacement:'
      + ',D14.6)
213  format(5(I5,6x))
218  format(1h , 'Pass Number = ',I9,/
219  format(23X,'Run Date: ',I2,'-',I2,'-',I2)
220  format(11X,'Amplitude of Shaking: ',D11.5)
901  format(1X,3(D22.16,1X))
9001 format(1X,2(D22.16,1X))
902  format(10X,A3)
903  format(18,5x,D19.13)
904  format(1X,'= C / ',D11.5,',' ,D11.5,',' ,D11.5)
905  format(1X,/) 
907  format(1X,'The Mean Energy = ',D13.5,5x,'Standard deviation 
      + D13.5)
908  format(6X,'The approximate packing height = ',D15.8,/,6X,
      + 'The standard deviation is ',D15.8)
909  format(11X,'The mean + standard deviation = ',D15.8,/,11X,
      + 'The mean - standard deviation = ',D15.8)
910  format(6X,'The approximate packing center = ',D15.8)
911  format(11X,'Upper limit = ',D15.8,/,11X,'Lower limit = ',
      + D15.8)
913  format(6X,'The mean of the entire y-array = ',D14.7)
914  format(11X,'Its standard deviation = ',D14.7)
915  format(6X,'The maximum height attained by some sphere =',
      + D14.7)
916  format(4X,'================================================================
      + ==
      + ==',/)
917  format(1X,D12.6,1X,D12.6,1X,11)
format(1h,'Shaking-Segregation Parameters: ',/,'26X,
+ 'yjump = ',D10.4,'26X,'ENERGY cut-off ratio = ',D12.6,)
+ '/,26X,'Number of Cycles = ',I5,'#'
format(1X,A4)
format(10X,A4)
format(10X,D9.3)
go to 115
C
888 write(9, 200) maxgen
go to 999
C
********** write the restart file to unit 31 **********
C
open(unit=31,file='[sxp4639.mc3d]mc3d.res',status='new')
open(unit=33,file='[sxp4639.rdf]radist3.dat',status='new')
C
do 900 ii = 1, n
   write(31,901) x(ii), y(ii), z(ii)
   write(31,9001) dia(ii), mass(ii)
900 continue
   write(31,*') totpas, e(mp1)
   if (pour .eq. 'pour') icy = 0
   write(31,*') icy
   write(31,*') dmax, dmin
   write(31,*') yjump
   write(31,918) pour
C
****** write the r.d.f. data file to unit 33  *******
C
   write(33,1902) n
   write(33,1901) diam
   write(33,*')
   write(33,*')
1902 format(I4)
1901 format(D7.1)
do 1900 ii=1, n
   write(33,1903) x(ii), y(ii), z(ii)
1903 format(3(D22.16, 1X))
1900 continue
C
********************************************************************
C Shell sort the y-array to estimatedhe average packed height.
C Also compute mean and standard deviation of entire y array.
C********************************************************************
C
Call SHELL(y, n)
Call SDEV(y, n, ymean, ydev)
nav = IDINT(length/diam)
  nava = n - nav
  do 400 K = 1, nav
    J = nava + K
    yt(K) = y(J)
400 continue
Call SDEV(yt, nav, ytmn, ytdev)
ycen = ytmean * 0.5DO  
ytmax = ytmean + ytdev  
ytmin = ytmean - ytdev  
ytmaxc = ytmax * 0.5DO  
ytminc = ytmin * 0.5DO  
    write(9,905)  
      write(9,908) ytmean, ytdev  
      write(9,909) ytmax, ytmin  
      write(9,910) ycen  
      write(9,911) ytmaxc, ytminc  
      write(9,913) ymean  
      write(9,914) ydev  
      write(9,915) yt(nav)  
      write(9,905)  
C  
999    close(unit=12)  
    close(unit=9 )  
    close(unit=31)  
    close(unit=33)  
    close(unit=36)  
    stop  
end  

--- End of main Program ---

* BINARY SEARCH SUBROUTINE *

SUBROUTINE SEARCH (xtemp, x, kdim, isave)  

Implicit Real*8 (a-h, o-z)  
Dimension x(kdim)  
Integer high, low, mid  

[SEARCH for location "isave" where xtemp < x(low)]

low = 1  
high = Kdim-1  
mid = 0.5 * ( low + high )  
if(xtemp .lt. x(mid)) high = mid - 1  
if(xtemp .gt. x(mid)) low = mid + 1  
if(xtemp .eq. x(mid)) go to 60  
if(low .le. high) go to 50  
isave = low  
go to 99  
60    isave = mid  
99    return
end

*********************
* REAL-VALUED INSERT SUBROUTINE *
*********************

SUBROUTINE AINSRT(xtemp, x, kdim, isize)
Implicit Real*8 (a-h, o-z)
Dimension x(kdim)

if (isave .eq. kdim) go to 70
  temp1 = x(isave)
  x(isave) = xtemp
  isp1 = isize + 1
  temp2 = x(isp1)
  x(isp1) = temp1
  j = isize + 2
  if (j .gt. Kdim) go to 100
  temp1 = x(j)
  x(j) = temp2
  jpl = j + 1
  if (jpl .gt. Kdim) go to 100
  temp2 = x(jpl)
  x(jpl) = temp1
  j = j + 2
  if (j .gt. Kdim) go to 100
  go to 80
70  x(isave) = xtemp
100  return
end

****************************
* INTEGER VALUED INSERT ROUTINE *
****************************

SUBROUTINE IINSRT(ntemp, nx, kdim, isize)

Insert xtemp in position "isave" and move other elements appropriately. This is identical to AINSRT except the argument are of type integer.

Implicit Real*8 (a-h, o-z)
Dimension nx(Kdim)
Integer temp1, temp2, ntemp

if (isave .eq. Kdim) go to 70
  temp1 = nx(isave)
  nx(isave) = ntemp
  isp1 = isize + 1
  temp2 = nx(isp1)
  nx(isp1) = temp1
  j = isize + 2
  if (j .gt. Kdim) go to 100
GEOMETRY CHECKING SUBROUTINE

This subroutine checks for forbidden overlap of spheres and returns a value of -1 in ier if overlap occurs.

 mode Character variable designating whether the routine is to be used in a 'generation' mode or in the 'simulation' mode.
 ipos array index of the particle that has been moved to a new position (xtemp, ytemp, ztemp). This is used used only in the 'simulate' mode.
 Kdim dimension of the x, y, and z arrays (plus 1).
 dmax maximum sphere diameter in the system of spheres.
 diam diameter of sphere to be inserted in 'generate' mode.
 isave array index, returned by SEARCH, at which the particle is to be placed after its move. ('simulate' mode)
 ier error flag returned as -1 if any discs overlap with the location of the displaced sphere. ('simulate' mode)
 x array of x-coordinates of the spheres.
 y array of y-coordinate of the spheres.
 z array of z-coordinates of the spheres.
 xtemp tentative new x-coordinate of the sphere whose x-coordinate is given by x(ipos).
 ytemp tentative new y-coordinate of the sphere of which y-coordinate is given by y(ipos).
 ztemp tentative new z-coordinate of the sphere whose z-coordinate is given by z(ipos).

SUBROUTINE GEOMCK(x, y, z, d, xtemp, ytemp, ztemp, dmax, diam, 
 isave, kdim, ipos, mode, ier)

Implicit Real*8 (a-h, o-z)
Dimension x(kdim), y(kdim), z(kdim), d(kdim)
Character *8 mode

C
if (mode .eq. 'generate') then
   rtemp = 0.5DO * diam
else
   rtemp = 0.5DO * d(1pos)
endif

C
j = 0
9 if (lsave+j .eq. ipos) j = j + 1
if ( lsave+j .gt. Kdim-1 ) then
   go to 50
endif
isj = lsave + j
dx = DABS(xtemp - x(isj))
dy = DABS(ytemp - y(isj))
dz = DABS(ztemp - z(isj))
dist = rtemp + d(isj)*0.5DO

C
10 if (dx .ge. dmax) then
   go to 50
endif
if (dx .ge. dist) then
   j=j+1
   go to 9
else if (dy .ge. dist) then
   j=j+1
   go to 9
else if (dz .ge. dist) then
   j=j+1
   go to 9
else
   dst =DSQRT(dx*dx + dy*dy +dz*dz)
   if (dst .ge. dist) then
      j = j + 1
      go to 9
   else
      ier = -1
      return
   endif
endif

C
50 j = -1
51 if (lsave+j .eq. ipos) j = j - 1
if ( lsave+j .lt. 1) go to 70
isj = lsave + j
dx = DABS(xtemp - x(isj))
dy = DABS(ytemp - y(isj))
dz = DABS(ztemp - z(isj))
dist = rtemp + d (isj)*0.5DO

C
60 if (dx .ge. dmax) then
   go to 70
endif
if (dx .ge. dist) then
    j = j-1
    go to 51
else if (dy .ge. dist) then
    j = j-1
    go to 51
else if (dz .ge. dist) then
    j = j-1
    go to 51
else
    dst = DSQRT(dx*dx + dy*dy + dz*dz)
    if (dst .ge. dist) then
        j = j-1
        go to 51
    else
        ier = -1
        return
    endif
endif
end
C
70   ier = 0
return
end

*****************************************************************************

*   ENERGY FUNCTION
*****************************************************************************

This function calculates the potential ENERGY of the system
in Joules, or Newton-meters. The y coordinates are
converted to meters. The mks system is used in this
calculation. (1 J = 1 Nm)

REAL*8 FUNCTION ENERGY(k, n, mass, g, y, ynew)

Implicit Real*8 (a-h, o-z)
Real*8 mass
Dimension y(n), mass(n)

temp = 0.0DO
temp1 = 0.0DO
do 10 j = 1, n
    temp = temp + mass(j) * g * y(j) * 0.0254DO
10 continue
energy = temp
if (k .ne. 0) then
    temp1 = mass(k) * g * y(k) * 0.0254DO
    energy = temp - temp1 + mass(k) * g * ynew * 0.0254DO
endif
return
end
Y-CENTROID FUNCTION

This function returns the y-coordinate of area centroid.

REAL*8 FUNCTION YBARF(n, y, dia)

Implicit Real*8 (a-h, o-z)
Dimension y(n), dia(n)

sum1 = 0.0DO
sum2 = 0.0DO
do 10 K = 1, n
   sum1 = sum1 + (dia(k)**2) * y(k)
   sum2 = sum2 + (dia(k)**2)
10
ybarf = sum1/sum2
return
end

SHELL SORT SUBROUTINE

This subroutine takes an array "v" of dimension "n" and sorts it into increasing order.

SUBROUTINE SHELL(v, n)

Implicit Real*8 (a-h, o-z)
Dimension v(n)
Integer gap
gap = n * 0.5
20  do 10 i = gap, n
    j = i - gap
15  if (j .le. 0) go to 10
    if (v(j) .le. v(j+gap)) go to 10
    temp = v(j)
    v(j) = v(j + gap)
    v(j + gap) = temp
    j = j - gap
  go to 15
10  continue
gap = gap * 0.5
if (gap .gt. 0) go to 20
99  return
end
**SUBROUTINE SDEV**

This routine computes the mean and standard deviation of the array "x" and returns them in "xmean" and "xdev" respectively.

SUBROUTINE SDEV(x, n, xmean, xdev)

Implicit Real*8 (a-h, o-z)
Dimension x(n)

n1 = n - 1
xsum = 0.0D0
sum = 0.0D0
do 10 i = 1, n
   xsum = xsum + x(i)
   sum = sum + x(i)
10   if (n1 .eq. 0) n1 = 1
   xmean = xsum / DFCLOAT(n1)
do 20 i = 1, n
   sum = sum + (x(i) - xmean)**2
20   var = sum / DFCLOAT(n1)
xdev = DSQRT(var)
return
end

**HISTOGRAM SUBROUTINE**

This subroutine takes an array "v" which has been presorted by the routine SHELL, and returns the integer array "freq" consisting of "nintv" elements. The latter array contains the frequency distribution of "v" which can be used to plot its histogram. The parameter "dmax" is the supremum of the elements of "v". The routine in effect does a binary SEARCH on the array "v".

SUBROUTINE HSTOGm(v, dmax, deltar, n, nintv, freq)

Implicit Real*8 (a-h, o-z)
Dimension v(n)
Integer freq(300), high, low, mid

do 10 i = 1, nintv + 1
   freq(i) = 0
10   isaveo = 1
   vtemp = deltar
   i = 1
80   if (vtemp .gt. dmax) go to 100
      low = isaveo
      high = n
50   mid = (high + low) * 0.5
if (vtemp .lt. v(mid)) high = mid - 1
if (vtemp .gt. v(mid)) low = mid + 1
if (vtemp .eq. v(mid)) then
    Km = mid + 1
    mid = Km
endif
vt1 = DABS(v(Km) - vtemp)
if (vt1 .gt. 1.0D-06*vtemp) go to 60
    Km = Km + 1
    mid = Km
    go to 30
else
    isave = low
    go to 99
endif
isave = mid
freq(i) = isave - isaveo
isaveo = isave
if (isaveo .gt. n) go to 100
    i = i + 1
    vtemp = vtemp + deltar
    go to 80
return
end
B.2 Coordination Number Code

Co-ordination Number

Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>freq</td>
<td>Integer array containing histogram of intersphere distances.</td>
</tr>
<tr>
<td>(x, y, z)</td>
<td>Arrays of sphere center coordinates.</td>
</tr>
<tr>
<td>n</td>
<td>Number of spheres.</td>
</tr>
<tr>
<td>xlng</td>
<td>Length of &quot;box&quot; containing spheres.</td>
</tr>
<tr>
<td>zlng</td>
<td>Width of &quot;box&quot; containing spheres.</td>
</tr>
<tr>
<td>dx</td>
<td>Difference between x coordinates of two spheres.</td>
</tr>
<tr>
<td>dy</td>
<td>Difference between y coordinates of two spheres.</td>
</tr>
<tr>
<td>dz</td>
<td>Difference between z coordinates of two spheres.</td>
</tr>
<tr>
<td>dia</td>
<td>Diameter of sphere.</td>
</tr>
<tr>
<td>eps</td>
<td>Tolerance of the close contact = (1 + eps) * Diameter. Input by user.</td>
</tr>
<tr>
<td>dst</td>
<td>Intersphere distance.</td>
</tr>
</tbody>
</table>

Input and Output Files

Unit 51 Defined as [SXP4639.codnum]cod_num.dat. This is the file from which the input data is read.
Unit 52 Defined as [SXP4639.codnum]cod_num.out. This is the file to which output is written.

Description

This program calculates the coordination number which is defined as the number of spheres in contact with a given sphere. The coordinates of spheres are read from the input file, then the coordination numbers within predefined tolerances of the diameter separation are calculated.

Beginning of Program

Implicit real*8 (a-h, o-z)
Integer freq(1000), sum(20), freq(1000), tot
Dimension x(1000), y(1000), z(1000)
Real perctg(20)

open(unit=51, file='[SXP4639.codnum]cod_num.dat', status='old')
open(unit=52, file='[SXP4639.codnum]cod_num.out', status='new')

read(51,*), n
read(51,*), dia
read(51,*), xlng, ylng, zlng
do 3 i = 1, n
   read(51,*) x(i), y(i), z(i)
3 continue
C
do 4 i = 1, n
   freq(i) = 0
4 continue
do 5 i = 1, n
   freq(i) = 0
5 continue
C
do 6 i = 1, 13
   sum(i) = 0
6 continue
C
do 7 i = 1, 13
   perctg(i) = 0.0
7 continue
C
eps = 0.05D+00
xlngf = xlng*0.5D+00
ylngf = ylng*0.5D+00
zlngf = zlng*0.5D+00
i = 0
k = 0
C
110 k = k + 1
111 if (((x(k).gt.(xlngf-0.8D0)).and.(x(k).lt.(xlngf+0.8D0)))
   .and.((y(k).gt.0.7D0).and.(y(k).lt.1.8D0)).and.(
   2((z(k).gt.(zlngf-0.8D0)).and.(z(k).lt.(zlngf+0.8D0)))) then
   go to 112
else
   k = k + 1
   if (k .gt. n) then
      go to 220
   else
      go to 111
   endif
   endif
C
112 i = i + 1
n1 = 1
do 38 j = 1, n
   dx = DABS(x(k) - x(j))
   dy = DABS(y(k) - y(j))
   dz = DABS(z(k) - z(j))
   if (dx .gt. xlngf) then
      if (x(j).gt.xlngf) then
         xjp = x(j) - xlng
      else
         xjp = x(j) + xlng
      endif
      dx = DABS(x(k) - xjp)
   endif
C
if (dy .gt. ylngf) then
    if (y(j) .gt. ylngf) then
        yjp = y(j) - ylng
    else
        yjp = y(j) + ylng
    endif
    dy = DABS(y(k) - yjp)
endif

C

if (dz .gt. zlngf) then
    if (z(j) .gt. zlngf) then
        zjp = z(j) - zlng
    else
        zjp = z(j) + zlng
    endif
    dz = DABS(z(k) - zjp)
endif

dst2 = dx**2 + dy**2 + dz**2
if ((dst2 .ge. ((1-eps)*dia)**2)).and.
    (dst2 .le. ((1+eps)*dia)**2)) then
    freq(k) = freq(k) + 1
    freq1(i) = freq(k)
endif

38     continue
    go to 110
C
220    do 43 i = 1, n1
       do 42 j = 1, 12
           if (freq1(i) .eq. 1) then
               sum(i) = sum(i) + 1
           endif
       42          continue
        43     continue
C
          tot = 0
    do 45 k = 1, 12
        tot = tot + sum(k)
    45     continue
C
          do 46 j = 1, 12
        tr = (FLOAT(sum(i)))/(FLOAT(tot))
        perctg(i) = tr * 100.0
    46     continue
C
          write(52,55)
          write(52,56)
          write(52,57)
    do 44 i = 1, 12
        write (52,58) i, sum(i), perctg(i), n1
    44     continue
C
55     format(10X, '-------------------------------------------------')
56     format(10X,'CO.NUM.',5X,'FREQ.',5X,'PERCTG',5X,'INNER SP.')
57     format(10X, '-------------------------------------------------')
58     format(10X,I3,6X,I5,7X,F7.4,5X,I4)
C
close(unit=51)
close(unit=52)
stop
der

C

End of Program

C
B.3 Packing Fraction code using Spherical Growth Method

Variables

(x, y, z) Arrays of sphere center coordinates.
n Number of spheres.
xlng Length of "box" containing spheres.
ylng Height of "box" containing spheres.
zlng Width of "box" containing spheres.
dx Difference between x coordinates of two spheres.
dy Difference between y coordinates of two spheres.
dz Difference between z coordinates of two spheres.
dia Diameter of sphere.
rad 0.5 * Diameter
dst Radius of the container.
dist Distance of interspheres.
tvol Volume of the spherical sample.
svol Volume of the spheres.

Input and Output Files

Unit 51 Defined as [SXP4639.fractn]frac.dat. This is the file from which the input data is read.
Unit 52 Defined as [SXP4639.fractn]frac.out. This is the file to which output is written.

Description

This program calculates the packing fraction from the several spherical samples. The packing fraction is determined by the ratio of the total volume of spheres (svol) to the volume of spherical sample (tvol).

Beginning of Program

Implicit real*8 (a-h,o-z)
Dimension x(2000), y(2000), z(2000), sum(110), tsum(100)

open(unit=51,file='[SXP4639.fractn]frac.dat',status='old')
open(unit=52,file='[SXP4639.fractn]frac.out',status='new')

read(51,*) n
read(51,*) dia
read(51,*) xlng, ylng, zlng

do 3 j = 1, n
    read(51,*) x(j), y(j), z(j)
C continue

rad = dia * 0.5D+00
pi = 3.1415926536D+00
xlngf = xlng*0.5d0
zlngf = zlng*0.5d0

C do 4 l = 1, 100
    sum(l) = 0.0d0
4 continue

C do 5 11 = 1, 100
    tsum(11) = 0.0DO
5 continue

C
i = 0
k = 0
k1 = 1

C k = k + 1
110 if ((x(k).gt.(xlngf-1.2d0)).and.(x(k).lt.(xlngf+1.2d0)))
1. and.((y(k).gt.0.16302D0).and.(y(k).lt.2.56302d0))
2. and.((z(k).gt.(zlngf-1.2d0)).and.(z(k).lt.(zlngf+1.2d0)))
3 then
    go to 112
else
    k = k + 1
    if (k .gt. n) then
        go to 220
    else
        go to 111
    endif
endif

C
112 i=i+1
n1=1
dst = dia

C vol1=0.0d0
vol2=0.0d0
do 38 j = 1, n
    dx = DABS(x(k)-x(j))
    dy = DABS(y(k)-y(j))
    dz = DABS(z(k)-z(j))
    if (dx.gt.xlng*0.5d0) then
        if (x(j).gt.xlng*0.5d0) then
            xjp = x(j)-xlng
        else
            xjp = x(j) + xlng
        endif
    endif
    dx = DABS(x(k)-xjp)
endif
    if (dy .gt. ylng*0.5d0) then
        if (y(j).gt.ylng*0.5d0) then
            yjp = y(j)-ylng
        else
            yjp = y(j) + ylng
        endif
    endif

117
else
  yjp = y(j) + ylng
endif
  dy = DABS(y(k) - yjp)
endif
  if (dz .gt. zlng*0.5d0) then
    if (z(j) .gt. zlng*0.5d0) then
      zjp = z(j) - zlng
    else
      zjp = z(j) + zlng
    endif
  dz = DABS(z(k) - zjp)
endif
  dist = DSQRT(dx*dx + dy*dy + dz*dz)

C
  if (dist .le. (dst-rad)) then
    vol1 = vol1 + 4.0d0/3.0d0*pi*rad**3
  C
  else if ((dist .gt. (dst-rad))
    c 
      d = (dst**2 + dist**2 - rad**2)/(2.0d0*dist)
      vol2 = vol2 + pi/3.0d0*(2*(dist**3)+2*(rad**3) + (dist**3) - 3*dist*(d**2+rad**2))
    c
  endif

C
  continue
C
  svol = vol1 + vol2
tvol = 4.0D0/3.0D0*pi*(dst**3)
pf = svol/tvol
  sum(k1) = sum(k1) + pf
C
  dst = dst + 0.05D0
  if (dst .gt. 1.2D0) then
    k1 = 1
    go to 110
  else
    k1 = k1 + 1
    go to 11
  endif
C
  do 39 k1 = 1, 19
    tsum(k1) = sum(k1)/n1
    write(52,58) tsum(k1), n1
  39 continue
C
  format(7x, F7.4, 10x, I5)
close(unit=51)
close(unit=52)
stop
end

C
C
C
C
C

End of Program
B.4 Packing Fraction code using Plane Growth Method

Packing volume fraction by plane growth method

Variables

(x, y, z) Arrays of sphere center coordinates.
n Number of spheres.
xlng Length of "box" containing spheres.
yhigh Heith of the packing sampled.
zlng Width of "box" containing spheres.
dx Difference between x coordinates of two spheres.
dy Difference between y coordinates of two spheres.
dz Difference between z coordinates of two spheres.
dia Diameter of sphere.
rad Diameter * 0.5
tvol Volume of the container.
svol Volume of the spheres.

description
This method cuts the packing by a plane and calculates the
volume of spheres bounded by that plane and the periodic
"walls". The packing fraction is determined by the ratio of
the volume of spheres (svol) to the volume of container
(tvol) containing them.

beginning of program

Implicit real*8 (a-h,o-z)

open(unit=51,file='[SXP4639.dens]dnsty.dat',status='old')
open(unit=52,file='[SXP4639.dens]dnsty.out',status='new')

read(51,*), n
read(51,*), dia
read(51,*), xlng, yhigh, zlng

do 3 j = 1, n
   read(51,*), x(j), y(j), z(j)
   continue
3

rad = dia * 0.5D+00
pi = 3.1415926536D+00
nu1 = 0
nu2 = 0
nu3 = 0
vol = 0.0D0
C

Volume of Spherical segment of one base

C
do 4 k = 1, n
  if ((y(k).lt.(yhigh+rad)).and.(y(k).ge.yhigh)) then
    nu1 = nu1 + 1
    h = rad - y(k) + yhigh
    vol1 = vol1 + 1.0DO/3.0DO*pi*h*h*(3*rad-h)
  else if (((y(k).lt.yhigh).and.(y(k).gt.(yhigh-rad))) then
    nu2 = nu2 + 1
    h = y(k) + rad - yhigh
    vol2 = vol2 + 4.0DO/3.0DO*pi*rad*rad*rad
  1
  else if (y(k) .le. (yhigh - rad)) then
    nu3 = nu3 + 1
    vol3 = vol3 + 4.0DO/3.0DO*pi*rad*rad*rad
  endif
4 continue

C
tvol = xlng * yhigh * zlng
svol = vol1 + vol2 + vol3
pd = svol/tvol

C
write(52,*), tvol, svol, pd

C
close(unit=51)
close(unit=52)
stop
end

C

==================================================================================================

C
End of Program

C

==================================================================================================
B.5 Radial Distribution Function code

Variables

- freq: Integer array containing histogram of intersphere distances.
- g: Array containing distribution function.
- (x, y, z): Arrays of sphere center coordinates.
- n: Number of spheres.
- delbin: Bin width in units of sphere diameter. It is also redefined as "delbin * dia".
- xlng: Length of "box" containing spheres.
- zlng: width of "box" containing spheres.
- dx: Difference between x coordinates of two spheres.
- dy: Difference between y coordinates of two spheres.
- dz: Difference between z coordinates of two spheres.
- dia: Diameter of sphere.
- rad: Radius of sphere.
- dmax: Maximum distance for which distribution function is to be calculated.
- eps = 1.0D-07: error parameter to account for machine accuracy.
- dst: Intersphere distance.
- kbmax: Total number of bins.
- low: Integer used in binary search to locate correct bin.
- high: Integer used in binary search to locate correct bin.
- mid: Integer used in binary search to locate correct bin.

Input and Output Files

- Unit 51: Defined as [SXP4639.rdf]radist3.dat. This is the file from which the input data is read.
- Unit 52: Defined as [SXP4639.rdf]radist3.out. This is the file to which output is written.

Description

This Fortran 77 code computes the radial distribution function from the coordinates of the center of the sphere (i.e. the configuration). The configuration is read from unit 51. The code computes and returns (to unit 52) the histogram of intersphere distances and the distribution function. This is defined as the number of sphere centers in (r, r+dr) divide by \(4 \times \pi \times (r/\text{dia})^2\) Note that the ring width, "dr", is not included here in the
definition of the distribution function. Also, the
distribution function is computed at the midpoint of each
bin.

===============================================

Beginning of Program
===============================================

Implicit real*8 (a-h,o-z)
Integer freq(600), low, high, mid
Dimension x(2000), y(2000), z(2000), g(600)

open(unit=51,fname='[SXP4639.rdf]radist3.dat',status='old')
open(unit=52,fname='[SXP4639.rdf]radist3.out',status='new')
eps=1.0D-07

read(51,*) n
read(51,*) dia
read(51,*) delbin
read(51,*) xlng, ylng, zlng

do 3 i = 1, n
    read(51,*) x(i), y(i), z(i)
    continue
3

do 4 i = 1, 600
    freq(i) = 0
    continue
4

dmax = xlng
rad = dia * 0.5D0
delbin = delbin * dia
kbmax = dmax/delbin

k = 1
j = k + 1
if (j .gt. n) then
    k = k + 1
    if (k .gt. n) then
        go to 100
    endif
    go to 5
endif

dx = DABS(x(k) - x(j))
dy = DABS(y(k) - y(j))
dz = DABS(z(k) - z(j))
if (dx .gt. xlng*0.5D0) then
    if (x(j) .gt. xlng*0.5D0) then
        xjp = x(j) - xlng
    else
        xjp = x(j) + xlng
    endif
    dx = DABS(x(k) - xjp)
endif
if (dy .gt. ylng*0.5DO) then
  if (y(j) .gt. ylng*0.5DO) then
    yjp = y(j) - ylng
  else
    yjp = y(j) + ylng
  endif
  dy = DABS(y(k) - yjp)
endif

if (dz .gt. zlng*0.5DO) then
  if (z(j) .gt. zlng*0.5DO) then
    zjp = z(j) - zlng
  else
    zjp = z(j) + zlng
  endif
  dz = DABS(z(k) - zjp)
endif

dst = DSQRT(dx*dx + dy*dy + dz*dz)

low = 1
high = kbmax
mid = (low + high)/2
bmid = FLOAT(mid) * delbin
if (dst .lt. bmid) high = mid - 1
if (dst .gt. bmid) low = mid + 1
if ((dst .le. bmid+eps).and.(dst .ge. bmid-eps)) then
  freq(mid) = freq(mid) + 1
  j = j + 1
  go to 6
endif
if (low .le. high) then
  go to 50
else
  isave = low
  freq(isave) = freq(isave) + 1
  j = j + 1
  go to 6
endif

write(52,200)
write(52,201)
write(52,202)

pi = 3.1415926536DO
rs = delbin

do 110 j = 1, kbmax
  rsdl = (rs - 0.5DO*delbin)/dia
  g(j) = 2.0DO*FLOAT(freq(j))/(4.0DO*pi*rsdl*rsdl*FLOAT(n))
  write(52,203) rsdl, freq(j), g(j)
  rs = rs + delbin
110  continue

format(1h ,10x,'RADIAL DISTRIBUTION FUNCTION',/,1h ,10x,
FOR PERIODIC BOUNDARY CONDITIONS

```
1  format(1h,5x,'r/diameter ',5x,'Frequency ',3x,
1         'Distribution Function')
202 format(1h,5x,'=====================================
1         '====='/',
203 format(1h,5x,D10.3,10x,.17,.9x,.D13.6)
C
120 close(unit=51)
close(unit=52)
stop
end

------------------------------
End of Program
------------------------------
```
B.6 Cumulative Probability of the Normalized Nearest Neighbor Distance r/dia.

(a) Calculating the nearest neighbor distances

Variables

(x, y, z) Arrays of sphere center coordinates.
n Number of spheres.
xlng Length of "box" containing spheres.
ylng Heith of "box" containing spheres.
zlng Width of "box" containing spheres.
dx Difference between x coordinates of two spheres.
dy Difference between y coordinates of two spheres.
dz Difference between z coordinates of two spheres.
dia Diameter of sphere.
rad Diameter * 0.5
dst Distance of interspheres.

Input and Output files

Unit 51 Defined as [adr7805.park.dist]dist.dat. This is the file from which the input data is read.

Unit 52 Defined as [adr7805.park.dist]dist.out. This is the file from which the output data is written.

Description

This program chooses the nearest distance between two spheres. [dist.dat] is the coordinates of the spheres.

Beginning of Program

Implicit real*8 (a-h,o-z)
Integer low, high, mid
Dimension x(1100), y(1100), z(1100), dst(1100)

open(unit=51,file='[ADR7805.PARK.dist]dist.dat',status='old')
open(unit=52,file='[ADR7805.PARK.dist]dist.out',status='new')

read(51,*) n
read(51,*) dia
read(51,*) xlng, ylng, zlng
do 3 i = 1, n
   read(51,*), x(i), y(i), z(i)
3 continue
C
do 4 i = 1, n
   dst(i) = 0.0DO
4 continue
C
   k = 1
   j = 1
   j1 = 1
C
   if (j .gt. n) then
      L = 1
      do 66 i1 = 2, 999
         if (dst(i1) .ge. dst(L)) then
            go to 66
         else
            L = i1
         endif
66 continue
   write(52,*), dst(L)
C
   k = k + 1
   if (k .gt. n) then
      go to 100
   endif
   j = 1
   j1 = 1
   go to 67
endif
C
   if (k-j .eq. 0) then
      j = j + 1
   endif
C
   dx = DABS(x(k) - x(j))
   dy = DABS(y(k) - y(j))
   dz = DABS(z(k) - z(j))
C
   if (dx .gt. xlng*0.5DO) then
      if (x(j) .gt. xlng*0.5DO) then
         xjp = x(j) - xlng
      else
         xjp = x(j) + xlng
      endif
      dx = DABS(x(k) - xjp)
   endif
C
   if (dy .gt. ylng*0.5DO) then
      if (y(j) .gt. ylng*0.5DO) then
         yjp = y(j) - ylng
      else
         yjp = y(j) + ylng
      endif
      dy = DABS(y(k) - yjp)
endif

C

if (dz .gt. zlng*0.5DO) then
   if (z(j) .gt. zlng*0.5DO) then
      zjp = z(j) - zlng
   else
      zjp = z(j) + zlng
   endif
   dz = DABS(z(k) - zjp)
endif

C

dst(j1) = DSQRT(dx*dx + dy*dy + dz*dz)

C

j = j + 1
j1 = j1 + 1
go to 6

C

100 close(unit=51)
close(unit=52)
stop
end

C
C
C
C
C

End of program

C
C
(b) Sorting the nearest distances obtained from code (a)

===== Exchange sort method =====

Variables

\( a(i) \)  Arrays of distances obtained from code (a).
\( n \)  Number of distances.
\( \text{dia} \)  Diameter of sphere.

Input and Output files

Unit 5  Defined as \([\text{adr7805.park.dist}]\text{sort.dat}\). This is the file from which the input data is read.

Unit 6  Defined as \([\text{adr7805.park.dist}]\text{sort.out}\). This is the file from which the output data is written.

Description

This program sorts the distances obtained from code (a) and normalize them by diameter of a sphere.

Beginning of Program

implicit real*8 (a-h, o-z)
dimension a(1000)

open (unit=5, file='[\text{adr7805.park.dist}]\text{sort.dat}', status='old')
open (unit=6, file='[\text{adr7805.park.dist}]\text{sort.out}', status='new')

read (5,*) n
read (5,*) dia

do 3 i = 1, n
  read (5,*) a(i)
  continue
3

last = n - 1

do 4 j = 1, n
  l = j
  j1 = j + 1
  do 110 k = j1, n
    if (a(l) .le. a(k)) then
      go to 110
    else
      continue
110
4
l = k
endif
110 continue
C
temp = a(1)
a(1) = a(j)
a(j) = temp
C
if ((a(j) - a(j-1)) .ne. 0.0D0) then
write (6,*) a(j)/dia
endif
C
continue
C
close (unit = 5)
close (unit = 6)
C
stop
end
C
===================================================================
C
End of program
C
===================================================================
(c) Cumulative probability of the normalized distances.

====================================================================
Cumulative probability of the normalized distances
====================================================================

Variables

dist    Arrays of normalized distances from code (b).
n    Number of distances.
dia    Diameter of sphere.
delbin    Bin width in unit of sphere diameter.
kmax    Total number of bins.

.................................................................
Input and Output files

Unit 51 Defined as [adr7805.park.dist]search.dat. This
is the file from which the input data is read.

Unit 52 Defined as [adr7805.park.dist]search.out. This
is the file from which the output data is written

.................................................................
Description
This program computes the cumulative probability of the nor-
malized nearest distances using the result of code (b).

====================================================================
Beginning of Program
====================================================================

Implicit real*8 (a-h,o-z)
Integer freq(2000), sum
Dimension dist(1100)

open(unit=51, file='[adr7805.park.dist]search.dat', status='old')
open(unit=52, file='[adr7805.park.dist]search.out', status='new')

read(51,*), n
read(51,*), delbin

3 do 1 = 1, n
     read(51,*), dist(i)
     continue

4 do 1 = 1, 2000
     freq(i) = 0
     continue

kmax = 0.001DO/delbin
i = 1
110 if (i .eq. 360) then
    write(52,* ) dist(i)
    endif
if (i .gt. n) then
    go to 220
endif
ad = dist(i)-1.0DO
call SEARCH (kbmax, delbin, ad, freq)
i = i + 1
go to 110

220 sum = 0
rs = 1.0DO + delbin
do 40 j =1, kbmax
    sum = sum + freq(j)
    cum = FLOAT(sum)/FLOAT(n)
    write(52,53) rs, sum, cum
    rs = rs + delbin
40 continue

53 format(5X,F12.6,5X,I5,5X,F12.5)
close(unit=51)
close(unit=52)

stop
end

===============================================
                      End of Main Program
===============================================

Subroutine SEARCH (kbmax, delbin, ad, freq)

Implicit Real*8 (a-h, o-z)
Integer freq(2000), high
eps1 = 1.0D-09

low = 1
high = kbmax
mid = (low + high)/2
bmid = FLOAT(mid) * delbin

if (ad .lt. bmid) then
    high = mid - 1
endif
if (ad .gt. bmid) then
    low = mid + 1
endif
if ((ad .le. bmid+eps1).and.(ad .ge. bmid-eps1)) then
    freq(mid) = freq(mid) + 1
    return
endif
if (low .le. high) then
    go to 50

131
else
    isave = low
    freq(isave) = freq(isave) + 1
    return
endif
end
APPENDIX C

C.1 Coordination number of pouring simulation

<table>
<thead>
<tr>
<th>NO. OF CONTACTS</th>
<th>SPHERE SEPARATION BY DIAMETER</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1</td>
<td>1.05</td>
<td>1.01</td>
</tr>
<tr>
<td>3</td>
<td>0.654</td>
<td>2.064</td>
<td>9.455</td>
</tr>
<tr>
<td>4</td>
<td>2.748</td>
<td>8.644</td>
<td>24.57</td>
</tr>
<tr>
<td>5</td>
<td>8.974</td>
<td>23.71</td>
<td>33.78</td>
</tr>
<tr>
<td>6</td>
<td>23.29</td>
<td><strong>32.58</strong></td>
<td>23.78</td>
</tr>
<tr>
<td>7</td>
<td><strong>32.08</strong></td>
<td>24.18</td>
<td>7.232</td>
</tr>
<tr>
<td>8</td>
<td>23.38</td>
<td>7.701</td>
<td>0.797</td>
</tr>
<tr>
<td>9</td>
<td>7.896</td>
<td>1.044</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>0.972</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td><strong>MEAN NUMBER</strong></td>
<td><strong>6.9</strong></td>
<td><strong>5.97</strong></td>
<td><strong>4.98</strong></td>
</tr>
</tbody>
</table>

C.2 Coordination number of shaking simulation

<table>
<thead>
<tr>
<th>NO. OF CONTACTS</th>
<th>SPHERE SEPARATION BY DIAMETER</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.1</td>
<td>1.05</td>
<td>1.01</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>0.178</td>
<td>6.595</td>
</tr>
<tr>
<td>4</td>
<td>1.07</td>
<td>4.635</td>
<td>21.39</td>
</tr>
<tr>
<td>5</td>
<td>4.10</td>
<td>13.90</td>
<td><strong>29.23</strong></td>
</tr>
<tr>
<td>6</td>
<td>12.12</td>
<td>29.06</td>
<td>25.13</td>
</tr>
<tr>
<td>7</td>
<td><strong>29.77</strong></td>
<td><strong>30.48</strong></td>
<td>14.08</td>
</tr>
<tr>
<td>8</td>
<td>31.55</td>
<td>17.29</td>
<td>3.030</td>
</tr>
<tr>
<td>9</td>
<td>17.65</td>
<td>4.099</td>
<td>0.357</td>
</tr>
<tr>
<td>10</td>
<td>3.743</td>
<td>0.357</td>
<td>-</td>
</tr>
<tr>
<td><strong>MEAN NUMBER</strong></td>
<td><strong>7.55</strong></td>
<td><strong>6.55</strong></td>
<td><strong>5.29</strong></td>
</tr>
</tbody>
</table>
C.3 Packing fraction by spherical growth method after pouring

<table>
<thead>
<tr>
<th>spherical distance from the center</th>
<th>packing fraction centers within 2 sphere dia.</th>
<th>3 sphere dia.</th>
<th>4 sphere dia.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.538</td>
<td>0.543</td>
<td>0.535</td>
</tr>
<tr>
<td>0.35</td>
<td>0.573</td>
<td>0.578</td>
<td>0.566</td>
</tr>
<tr>
<td>0.4</td>
<td>0.584</td>
<td>0.587</td>
<td>0.574</td>
</tr>
<tr>
<td>0.45</td>
<td>0.571</td>
<td>0.572</td>
<td>0.558</td>
</tr>
<tr>
<td>0.5</td>
<td>0.558</td>
<td>0.558</td>
<td>0.542</td>
</tr>
<tr>
<td>0.55</td>
<td>0.560</td>
<td>0.560</td>
<td>0.542</td>
</tr>
<tr>
<td>0.6</td>
<td>0.566</td>
<td>0.564</td>
<td>0.544</td>
</tr>
<tr>
<td>0.65</td>
<td>0.569</td>
<td>0.566</td>
<td>0.543</td>
</tr>
<tr>
<td>0.7</td>
<td>0.568</td>
<td>0.562</td>
<td>0.539</td>
</tr>
<tr>
<td>0.75</td>
<td>0.565</td>
<td>0.558</td>
<td>0.532</td>
</tr>
<tr>
<td>0.8</td>
<td>0.564</td>
<td>0.555</td>
<td>0.528</td>
</tr>
<tr>
<td>0.85</td>
<td>0.565</td>
<td>0.554</td>
<td>0.526</td>
</tr>
<tr>
<td>0.9</td>
<td>0.565</td>
<td>0.552</td>
<td>0.523</td>
</tr>
<tr>
<td>0.95</td>
<td>0.563</td>
<td>0.549</td>
<td>0.519</td>
</tr>
<tr>
<td>1.0</td>
<td>0.562</td>
<td>0.545</td>
<td>0.514</td>
</tr>
<tr>
<td>1.05</td>
<td>0.560</td>
<td>0.541</td>
<td>0.510</td>
</tr>
<tr>
<td>1.1</td>
<td>0.558</td>
<td>0.538</td>
<td>0.506</td>
</tr>
<tr>
<td>1.15</td>
<td>0.556</td>
<td>0.534</td>
<td>0.502</td>
</tr>
<tr>
<td>1.2</td>
<td>0.553</td>
<td>0.530</td>
<td>0.498</td>
</tr>
<tr>
<td>average</td>
<td>0.563</td>
<td>0.555</td>
<td>0.532</td>
</tr>
</tbody>
</table>
### C.4 Packing fraction by spherical growth method after shaking

<table>
<thead>
<tr>
<th>Spherical distance from the center</th>
<th>Packing fraction centers within</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 sphere dia.</td>
</tr>
<tr>
<td>0.3</td>
<td>0.574</td>
</tr>
<tr>
<td>0.35</td>
<td>0.609</td>
</tr>
<tr>
<td>0.4</td>
<td>0.615</td>
</tr>
<tr>
<td>0.45</td>
<td>0.597</td>
</tr>
<tr>
<td>0.5</td>
<td>0.584</td>
</tr>
<tr>
<td>0.55</td>
<td>0.590</td>
</tr>
<tr>
<td>0.6</td>
<td>0.598</td>
</tr>
<tr>
<td>0.65</td>
<td>0.600</td>
</tr>
<tr>
<td>0.7</td>
<td>0.597</td>
</tr>
<tr>
<td>0.75</td>
<td>0.593</td>
</tr>
<tr>
<td>0.8</td>
<td>0.593</td>
</tr>
<tr>
<td>0.85</td>
<td>0.595</td>
</tr>
<tr>
<td>0.9</td>
<td>0.596</td>
</tr>
<tr>
<td>0.95</td>
<td>0.593</td>
</tr>
<tr>
<td>1.0</td>
<td>0.590</td>
</tr>
<tr>
<td>1.05</td>
<td>0.587</td>
</tr>
<tr>
<td>1.1</td>
<td>0.585</td>
</tr>
<tr>
<td>1.15</td>
<td>0.582</td>
</tr>
<tr>
<td>1.2</td>
<td>0.579</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.592</strong></td>
</tr>
</tbody>
</table>
C.5 Packing fraction by plane growth method after pouring

<table>
<thead>
<tr>
<th>r/dia.</th>
<th>packing fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td>0.5401192109310428</td>
</tr>
<tr>
<td>2.8</td>
<td>0.5533563455982388</td>
</tr>
<tr>
<td>2.7</td>
<td>0.5600851403260435</td>
</tr>
<tr>
<td>2.6</td>
<td>0.5615047610326287</td>
</tr>
<tr>
<td>2.5</td>
<td>0.5611386025702016</td>
</tr>
<tr>
<td>2.4</td>
<td>0.5599272515142990</td>
</tr>
<tr>
<td>2.3</td>
<td>0.5595471876137151</td>
</tr>
<tr>
<td>2.2</td>
<td>0.5585678623456342</td>
</tr>
<tr>
<td>2.1</td>
<td>0.5570778976261102</td>
</tr>
<tr>
<td>2.0</td>
<td>0.5562386420548790</td>
</tr>
<tr>
<td>1.9</td>
<td>0.5554193648712017</td>
</tr>
<tr>
<td>1.8</td>
<td>0.555645800907506</td>
</tr>
<tr>
<td>1.7</td>
<td>0.5552448467437003</td>
</tr>
<tr>
<td>1.6</td>
<td>0.5530973602248455</td>
</tr>
<tr>
<td>1.5</td>
<td>0.5534938907057115</td>
</tr>
<tr>
<td>1.4</td>
<td>0.5536318213528416</td>
</tr>
<tr>
<td>1.3</td>
<td>0.5520493410288450</td>
</tr>
<tr>
<td>1.2</td>
<td>0.5509418419623947</td>
</tr>
<tr>
<td>1.1</td>
<td>0.5492418546867472</td>
</tr>
<tr>
<td>1.0</td>
<td>0.546423879526412</td>
</tr>
<tr>
<td>0.9</td>
<td>0.5417447435574421</td>
</tr>
<tr>
<td>0.8</td>
<td>0.538573856187314</td>
</tr>
<tr>
<td>0.7</td>
<td>0.5358313991082953</td>
</tr>
<tr>
<td>0.6</td>
<td>0.5289553649019375</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5252080193302946</td>
</tr>
</tbody>
</table>
C.6 Packing fraction by plane growth method after shaking

<table>
<thead>
<tr>
<th>r/dia.</th>
<th>packing fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8</td>
<td>0.5606383468702942</td>
</tr>
<tr>
<td>2.7</td>
<td>0.5766495711363379</td>
</tr>
<tr>
<td>2.6</td>
<td>0.5846561458142102</td>
</tr>
<tr>
<td>2.5</td>
<td>0.5857439493292930</td>
</tr>
<tr>
<td>2.4</td>
<td>0.5855629952082822</td>
</tr>
<tr>
<td>2.3</td>
<td>0.5850296185058922</td>
</tr>
<tr>
<td>2.2</td>
<td>0.5848551228148686</td>
</tr>
<tr>
<td>2.1</td>
<td>0.5851321397674422</td>
</tr>
<tr>
<td>2.0</td>
<td>0.5841343563632578</td>
</tr>
<tr>
<td>1.9</td>
<td>0.5853026573358648</td>
</tr>
<tr>
<td>1.8</td>
<td>0.5848086754926802</td>
</tr>
<tr>
<td>1.7</td>
<td>0.5831994237058778</td>
</tr>
<tr>
<td>1.6</td>
<td>0.5832984725730151</td>
</tr>
<tr>
<td>1.5</td>
<td>0.5815191581614208</td>
</tr>
<tr>
<td>1.4</td>
<td>0.581762005080301373</td>
</tr>
<tr>
<td>1.3</td>
<td>0.583592809004963</td>
</tr>
<tr>
<td>1.2</td>
<td>0.5823838955573167</td>
</tr>
<tr>
<td>1.1</td>
<td>0.5824365402864338</td>
</tr>
<tr>
<td>1.0</td>
<td>0.5832531305759836</td>
</tr>
<tr>
<td>0.9</td>
<td>0.5819327194935322</td>
</tr>
<tr>
<td>0.8</td>
<td>0.5783872140324019</td>
</tr>
<tr>
<td>0.7</td>
<td>0.578244496803609</td>
</tr>
<tr>
<td>0.6</td>
<td>0.5725269548270809</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5728105533288898</td>
</tr>
</tbody>
</table>
### C.7 Radial distribution after pouring

<table>
<thead>
<tr>
<th>r/D</th>
<th>Radial distribution function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>0.540864</td>
</tr>
<tr>
<td>1.3</td>
<td>0.149267</td>
</tr>
<tr>
<td>1.5</td>
<td>0.140551</td>
</tr>
<tr>
<td>1.7</td>
<td>0.195392</td>
</tr>
<tr>
<td>1.9</td>
<td>0.256588</td>
</tr>
<tr>
<td>2.1</td>
<td>0.183299</td>
</tr>
<tr>
<td>2.3</td>
<td>0.165834</td>
</tr>
<tr>
<td>2.5</td>
<td>0.181920</td>
</tr>
<tr>
<td>2.7</td>
<td>0.199566</td>
</tr>
<tr>
<td>2.9</td>
<td>0.190853</td>
</tr>
<tr>
<td>3.1</td>
<td>0.172073</td>
</tr>
<tr>
<td>3.3</td>
<td>0.172922</td>
</tr>
<tr>
<td>3.5</td>
<td>0.181047</td>
</tr>
<tr>
<td>3.7</td>
<td>0.177326</td>
</tr>
<tr>
<td>3.9</td>
<td>0.168625</td>
</tr>
<tr>
<td>4.1</td>
<td>0.166190</td>
</tr>
<tr>
<td>4.3</td>
<td>0.167453</td>
</tr>
<tr>
<td>4.5</td>
<td>0.166276</td>
</tr>
<tr>
<td>4.7</td>
<td>0.162159</td>
</tr>
<tr>
<td>4.9</td>
<td>0.157418</td>
</tr>
<tr>
<td>5.1</td>
<td>0.148973</td>
</tr>
<tr>
<td>5.3</td>
<td>0.133386</td>
</tr>
</tbody>
</table>
### C.8 Radial distribution after shaking

<table>
<thead>
<tr>
<th>r/D</th>
<th>Radial distribution function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>0.580587</td>
</tr>
<tr>
<td>1.3</td>
<td>0.139755</td>
</tr>
<tr>
<td>1.5</td>
<td>0.136803</td>
</tr>
<tr>
<td>1.7</td>
<td>0.209765</td>
</tr>
<tr>
<td>1.9</td>
<td>0.272812</td>
</tr>
<tr>
<td>2.1</td>
<td>0.187088</td>
</tr>
<tr>
<td>2.3</td>
<td>0.163367</td>
</tr>
<tr>
<td>2.5</td>
<td>0.190731</td>
</tr>
<tr>
<td>2.7</td>
<td>0.211617</td>
</tr>
<tr>
<td>2.9</td>
<td>0.196455</td>
</tr>
<tr>
<td>3.1</td>
<td>0.173282</td>
</tr>
<tr>
<td>3.3</td>
<td>0.177219</td>
</tr>
<tr>
<td>3.5</td>
<td>0.188284</td>
</tr>
<tr>
<td>3.7</td>
<td>0.184894</td>
</tr>
<tr>
<td>3.9</td>
<td>0.172203</td>
</tr>
<tr>
<td>4.1</td>
<td>0.170800</td>
</tr>
<tr>
<td>4.3</td>
<td>0.170603</td>
</tr>
<tr>
<td>4.5</td>
<td>0.171321</td>
</tr>
<tr>
<td>4.7</td>
<td>0.165971</td>
</tr>
<tr>
<td>4.9</td>
<td>0.161886</td>
</tr>
<tr>
<td>5.1</td>
<td>0.151402</td>
</tr>
<tr>
<td>5.3</td>
<td>0.136503</td>
</tr>
</tbody>
</table>
C.9 Cumulative probability of the normalized nearest neighbour distance $r/d_{ia.}$ for 1000 sphere after pouring

<table>
<thead>
<tr>
<th>$r/d_{ia.}$</th>
<th>number</th>
<th>cumulative probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000</td>
<td>0</td>
<td>0.00000</td>
</tr>
<tr>
<td>1.000002</td>
<td>139</td>
<td>0.19577</td>
</tr>
<tr>
<td>1.000004</td>
<td>244</td>
<td>0.34366</td>
</tr>
<tr>
<td>1.000006</td>
<td>322</td>
<td>0.45352</td>
</tr>
<tr>
<td>1.000008</td>
<td>413</td>
<td>0.58169</td>
</tr>
<tr>
<td>1.000010</td>
<td>461</td>
<td>0.64930</td>
</tr>
<tr>
<td>1.000012</td>
<td>490</td>
<td>0.73944</td>
</tr>
<tr>
<td>1.000014</td>
<td>525</td>
<td>0.77183</td>
</tr>
<tr>
<td>1.000016</td>
<td>548</td>
<td>0.81268</td>
</tr>
<tr>
<td>1.000018</td>
<td>577</td>
<td>0.83803</td>
</tr>
<tr>
<td>1.000020</td>
<td>595</td>
<td>0.85493</td>
</tr>
<tr>
<td>1.000022</td>
<td>607</td>
<td>0.88310</td>
</tr>
<tr>
<td>1.000024</td>
<td>627</td>
<td>0.89577</td>
</tr>
<tr>
<td>1.000026</td>
<td>636</td>
<td>0.90986</td>
</tr>
<tr>
<td>1.000029</td>
<td>646</td>
<td>0.91972</td>
</tr>
<tr>
<td>1.000030</td>
<td>653</td>
<td>0.92858</td>
</tr>
<tr>
<td>1.000032</td>
<td>680</td>
<td>0.93662</td>
</tr>
<tr>
<td>1.000034</td>
<td>685</td>
<td>0.94085</td>
</tr>
<tr>
<td>1.000036</td>
<td>668</td>
<td>0.94507</td>
</tr>
<tr>
<td>1.000038</td>
<td>671</td>
<td>0.94789</td>
</tr>
<tr>
<td>1.000040</td>
<td>673</td>
<td>0.95634</td>
</tr>
<tr>
<td>1.000042</td>
<td>679</td>
<td>0.96197</td>
</tr>
<tr>
<td>1.000044</td>
<td>683</td>
<td>0.96338</td>
</tr>
<tr>
<td>1.000046</td>
<td>684</td>
<td>0.96761</td>
</tr>
<tr>
<td>1.000048</td>
<td>687</td>
<td>0.97042</td>
</tr>
<tr>
<td>1.000050</td>
<td>689</td>
<td>0.97324</td>
</tr>
<tr>
<td>1.000052</td>
<td>691</td>
<td>0.97746</td>
</tr>
<tr>
<td>1.000054</td>
<td>694</td>
<td>0.97887</td>
</tr>
<tr>
<td>1.000056</td>
<td>695</td>
<td>0.98028</td>
</tr>
<tr>
<td>1.000058</td>
<td>696</td>
<td>0.98028</td>
</tr>
<tr>
<td>1.000060</td>
<td>696</td>
<td>0.98028</td>
</tr>
<tr>
<td>1.000062</td>
<td>697</td>
<td>0.98169</td>
</tr>
<tr>
<td>1.000064</td>
<td>697</td>
<td>0.98169</td>
</tr>
<tr>
<td>1.000066</td>
<td>697</td>
<td>0.98169</td>
</tr>
<tr>
<td>1.000068</td>
<td>698</td>
<td>0.98310</td>
</tr>
<tr>
<td>1.000070</td>
<td>699</td>
<td>0.98451</td>
</tr>
<tr>
<td>1.000072</td>
<td>699</td>
<td>0.98451</td>
</tr>
<tr>
<td>1.000074</td>
<td>700</td>
<td>0.98592</td>
</tr>
<tr>
<td>1.000076</td>
<td>700</td>
<td>0.98592</td>
</tr>
<tr>
<td>1.000078</td>
<td>700</td>
<td>0.98592</td>
</tr>
<tr>
<td>1.000080</td>
<td>700</td>
<td>0.98592</td>
</tr>
<tr>
<td>1.000082</td>
<td>701</td>
<td>0.98732</td>
</tr>
<tr>
<td>1.000084</td>
<td>705</td>
<td>0.99296</td>
</tr>
<tr>
<td>1.000086</td>
<td>705</td>
<td>0.99296</td>
</tr>
<tr>
<td>1.000088</td>
<td>705</td>
<td>0.99296</td>
</tr>
<tr>
<td>1.000094</td>
<td>706</td>
<td>0.99437</td>
</tr>
<tr>
<td>1.000124</td>
<td>707</td>
<td>0.99577</td>
</tr>
<tr>
<td>1.000160</td>
<td>708</td>
<td>0.99718</td>
</tr>
<tr>
<td>1.000166</td>
<td>709</td>
<td>0.99859</td>
</tr>
<tr>
<td>1.000198</td>
<td>710</td>
<td>1.00000</td>
</tr>
</tbody>
</table>
C.10 Cumulative probability of the normalized nearest neighbour distance \( r/dia \) for 1000 sphere after shaking

<table>
<thead>
<tr>
<th>( r/dia )</th>
<th>number</th>
<th>cumulative probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000010</td>
<td>44</td>
<td>0.06162</td>
</tr>
<tr>
<td>1.000020</td>
<td>82</td>
<td>0.11485</td>
</tr>
<tr>
<td>1.000030</td>
<td>119</td>
<td>0.18667</td>
</tr>
<tr>
<td>1.000040</td>
<td>154</td>
<td>0.21569</td>
</tr>
<tr>
<td>1.000050</td>
<td>188</td>
<td>0.26471</td>
</tr>
<tr>
<td>1.000060</td>
<td>219</td>
<td>0.30672</td>
</tr>
<tr>
<td>1.000070</td>
<td>260</td>
<td>0.36415</td>
</tr>
<tr>
<td>1.000080</td>
<td>304</td>
<td>0.42577</td>
</tr>
<tr>
<td>1.000090</td>
<td>332</td>
<td>0.46499</td>
</tr>
<tr>
<td>1.000100</td>
<td>361</td>
<td>0.50560</td>
</tr>
<tr>
<td>1.000110</td>
<td>397</td>
<td>0.55602</td>
</tr>
<tr>
<td>1.000120</td>
<td>422</td>
<td>0.59104</td>
</tr>
<tr>
<td>1.000130</td>
<td>449</td>
<td>0.62885</td>
</tr>
<tr>
<td>1.000140</td>
<td>472</td>
<td>0.66106</td>
</tr>
<tr>
<td>1.000150</td>
<td>490</td>
<td>0.68627</td>
</tr>
<tr>
<td>1.000160</td>
<td>505</td>
<td>0.70728</td>
</tr>
<tr>
<td>1.000170</td>
<td>515</td>
<td>0.72129</td>
</tr>
<tr>
<td>1.000180</td>
<td>528</td>
<td>0.73950</td>
</tr>
<tr>
<td>1.000190</td>
<td>542</td>
<td>0.75910</td>
</tr>
<tr>
<td>1.000200</td>
<td>552</td>
<td>0.77311</td>
</tr>
<tr>
<td>1.000210</td>
<td>564</td>
<td>0.78992</td>
</tr>
<tr>
<td>1.000220</td>
<td>577</td>
<td>0.80812</td>
</tr>
<tr>
<td>1.000230</td>
<td>585</td>
<td>0.81933</td>
</tr>
<tr>
<td>1.000240</td>
<td>595</td>
<td>0.83333</td>
</tr>
<tr>
<td>1.000250</td>
<td>603</td>
<td>0.84454</td>
</tr>
<tr>
<td>1.000260</td>
<td>610</td>
<td>0.85434</td>
</tr>
<tr>
<td>1.000270</td>
<td>612</td>
<td>0.85714</td>
</tr>
<tr>
<td>1.000280</td>
<td>616</td>
<td>0.86275</td>
</tr>
<tr>
<td>1.000290</td>
<td>624</td>
<td>0.87395</td>
</tr>
<tr>
<td>1.000300</td>
<td>627</td>
<td>0.87815</td>
</tr>
<tr>
<td>1.000310</td>
<td>632</td>
<td>0.88515</td>
</tr>
<tr>
<td>1.000320</td>
<td>640</td>
<td>0.89636</td>
</tr>
<tr>
<td>1.000330</td>
<td>647</td>
<td>0.90616</td>
</tr>
<tr>
<td>1.000340</td>
<td>650</td>
<td>0.91035</td>
</tr>
<tr>
<td>1.000350</td>
<td>654</td>
<td>0.91597</td>
</tr>
<tr>
<td>1.000360</td>
<td>655</td>
<td>0.91737</td>
</tr>
<tr>
<td>1.000370</td>
<td>660</td>
<td>0.92437</td>
</tr>
<tr>
<td>1.000380</td>
<td>665</td>
<td>0.93137</td>
</tr>
<tr>
<td>1.000390</td>
<td>667</td>
<td>0.93417</td>
</tr>
<tr>
<td>1.000400</td>
<td>670</td>
<td>0.93938</td>
</tr>
<tr>
<td>1.000410</td>
<td>673</td>
<td>0.94258</td>
</tr>
<tr>
<td>1.000420</td>
<td>676</td>
<td>0.94578</td>
</tr>
<tr>
<td>1.000430</td>
<td>680</td>
<td>0.95238</td>
</tr>
<tr>
<td>1.000440</td>
<td>683</td>
<td>0.95658</td>
</tr>
<tr>
<td>1.000450</td>
<td>686</td>
<td>0.96078</td>
</tr>
<tr>
<td>1.000460</td>
<td>687</td>
<td>0.96218</td>
</tr>
<tr>
<td>1.000470</td>
<td>688</td>
<td>0.96359</td>
</tr>
<tr>
<td>1.000480</td>
<td>689</td>
<td>0.96499</td>
</tr>
<tr>
<td>1.000490</td>
<td>690</td>
<td>0.96640</td>
</tr>
<tr>
<td>1.000500</td>
<td>693</td>
<td>0.97059</td>
</tr>
<tr>
<td>1.000510</td>
<td>695</td>
<td>0.97339</td>
</tr>
<tr>
<td>1.000520</td>
<td>696</td>
<td>0.97479</td>
</tr>
<tr>
<td>1.000530</td>
<td>687</td>
<td>0.97619</td>
</tr>
<tr>
<td>1.000540</td>
<td>698</td>
<td>0.97759</td>
</tr>
<tr>
<td>1.000550</td>
<td>699</td>
<td>0.97899</td>
</tr>
<tr>
<td>1.000560</td>
<td>701</td>
<td>0.98179</td>
</tr>
<tr>
<td>1.000570</td>
<td>702</td>
<td>0.98319</td>
</tr>
<tr>
<td>1.000580</td>
<td>703</td>
<td>0.98459</td>
</tr>
<tr>
<td>1.000590</td>
<td>704</td>
<td>0.98599</td>
</tr>
<tr>
<td>1.000600</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000610</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000620</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000630</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000640</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000650</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000660</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000670</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000680</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000690</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000700</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000710</td>
<td>706</td>
<td>0.98880</td>
</tr>
<tr>
<td>1.000720</td>
<td>707</td>
<td>0.99020</td>
</tr>
<tr>
<td>1.000730</td>
<td>709</td>
<td>0.99300</td>
</tr>
<tr>
<td>1.000740</td>
<td>709</td>
<td>0.99300</td>
</tr>
<tr>
<td>1.000750</td>
<td>710</td>
<td>0.99440</td>
</tr>
<tr>
<td>1.000760</td>
<td>711</td>
<td>0.99580</td>
</tr>
<tr>
<td>1.000780</td>
<td>712</td>
<td>0.99720</td>
</tr>
<tr>
<td>1.000860</td>
<td>713</td>
<td>0.99860</td>
</tr>
<tr>
<td>1.000970</td>
<td>714</td>
<td>1.00000</td>
</tr>
<tr>
<td>1.001000</td>
<td>714</td>
<td>1.00000</td>
</tr>
</tbody>
</table>