PROJECT 6 - 2D RANDOM WALKS, DIFFUSION AND ENTROPY – FINAL REPORT

Jakub Palacka - 118403546

PY2105 Introduction to Computational Physics

Contents

Introduction

Random Walks

The purpose of this project is to model diffusion in 2-D using random walks and to calculate entropy as it varies with time. A random walk is a *stochastic* or random process which is used to describe a succession of random steps governed by probability on a mathematical plane which can be make applicable to real-world situations through the definition of a dimensions and a coordinate system. The simplest example is of a 1D walker capable of taking steps of unit length left or right with equal probability of both. The walker begins at the origin $(x = 0)$ and makes a step either left or right with the probability of each being 0.5. From its subsequent position ($x = x_1$) the walker makes another step with identical probability and this process is repeated for a set number of steps or *walks*[1]. The result of a set number of walks would result in a plot similar to the one below:

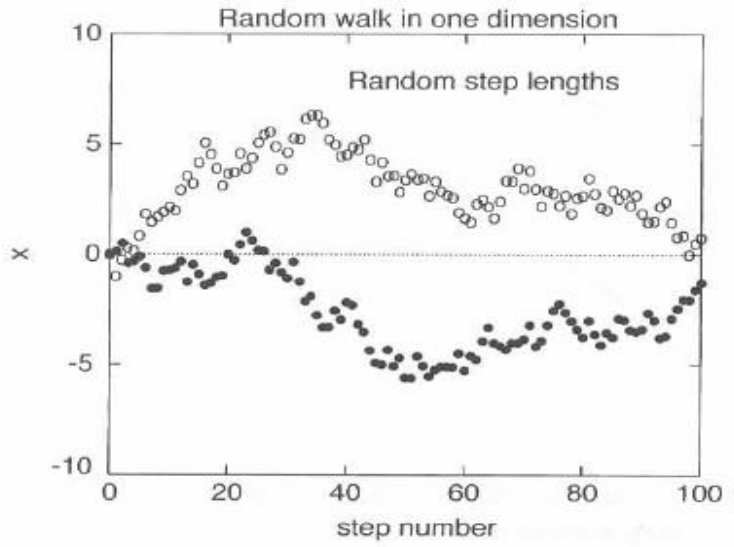


Figure 1. N.J.Giordano & H.Nakanishi – One dimensional random walk 7.3

Diffusion

Diffusion is the movement of particles from an area of high concentration to an area of low concentration and it is a phenomenon which can be most readily observed in mixtures consisting of small particles suspended in fluids. From an atomic or particulate point of view, diffusion can be expressed in terms of random walks of particles as they spread from an area of high concentration to one of low concentration. The random motion of small particles in fluids was first discovered by Robert Browne in 1827 while studying microscopic organisms. The movement of pollen particles was visible through the optical microscope he was using. By comparing this motion with that of particles floating inside water trapped in a quartz crystal he was able to deduce that this motion had no relation to organisms but a result of the pollen interacting with the water molecules.[2]

[1] N.J.Giordano & H.Nakanishi - Computational Physics 7.2

[2] The Brownian Movement". The Feynman Lectures of Physics, Volume I. 41.1

Diffusion is caused by the thermal motion of particles at temperatures above absolute zero. The rate at which this occurs is a function of temperature, viscosity of the fluid and the size (and mass) of the particles in question.[3] Throughout this project the setting of a "drop of cream in coffee" [4] will be used as an analogy for the system being modelled, the "drop" being a high concentration of particles which proceed to diffuse throughout the coffee. Initially 400 particles will be confined to a square "drop" in the centre of a 200x200 unit^2 container. The position of these particles will evolve over time as they disperse throughout the container and their displacement from the origin will also change over time. As the particles disperse and the displacement changes it can be said the solution becomes more "disordered" over time. Initially all the particles are situated in a uniform "drop" with equal spacing between them, as time goes on the spacing becomes unequal and random, the term attributed to the "disorder" of a system is called *Entropy.*

Entropy

The entropy of a state of a system is the measure of the degree of disorder of that state. An example[5] of this would be a monoatomic crystal sublimating into vapour. The initial state of the crystal is highly ordered with the atoms performing small vibrations about their equilibrium positions spaced uniformly throughout the crystal lattice. The final state has the atoms distributed randomly throughout the volume occupied by the vapour. As outlined in the previous paragraph, the "coffee and cream" analogy describes this equally well.

Natural processes (such as diffusion), are ones which are *irreversible* and results in an *increase in entropy*. The direction of a natural process is always towards equilibrium, therefore it can be said that at equilibrium entropy is at its maximum value. In terms of the "coffee and cream" analogy equilibrium and maximum entropy is exists when the "drop" of cream has completely dispersed throughout the coffee.

Minimal Objectives

1. Evolution of Particle Distribution

The evolution of the distribution of positions of the particles was be achieved through a series of scatter plots displaying the specific position of particles following set amounts of steps.

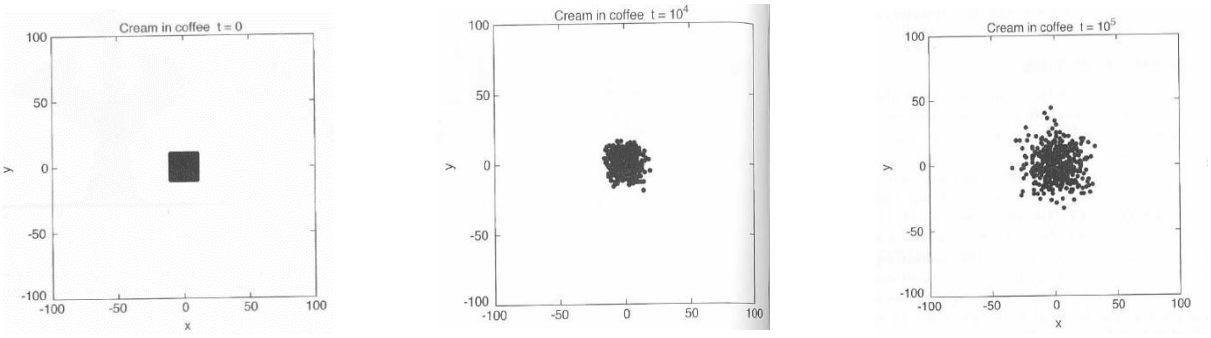


Figure 3. N.J.Giordano & H.Nakanishi – Particle distribution over time 7.13 and 7.14

[3] "Fundamentals of Fluid Flow in Porous Media" A. Kantzas, J. Bryan, S. Taheri

[4] N.J.Giordano & H.Nakanishi - Computational Physics 7.4

[5] "Statistical Physics" F.Mandle Chp 2.

An alternative way to display the distribution would have been through the use of a Gaussian distribution showing densities of particles after a set amount of steps, however since the precise coordinates of the particle positions as well as the sites they occupy at each step was already being recorded it was decided that the previous method was more convenient.

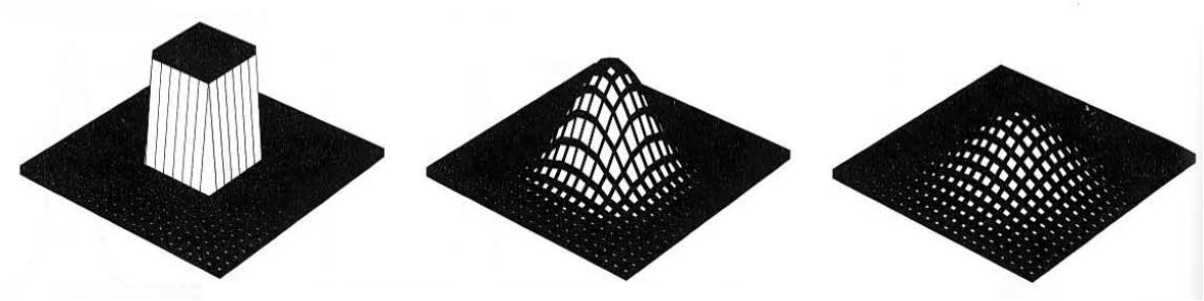


Figure 4. N.J.Giordano & H.Nakanishi – Gaussian distributions of particles at $t_0<t_1<t_2$ respectively. 7.11

2. Root-Mean-Square (RMS) Displacement ($\sqrt{< s^2 >}$)

The first objective in this experiment is to calculate the RMS of the system of particles from the origin and to graph this as a function of time (steps). The root-mean-square (RMS) displacement is defined as the square root of the mean-square displacement from a set reference point. The reference point used in this project will be the centre of the 200x200 container. The exact coordinates of this point will be defined in a later section. A defining feature of diffusion is that the distribution of the particles varies with $\sqrt{t(steps)}$[6]. As a result the graph of RMS versus time (steps) is expected to have the shape of the graph t versus $\sqrt{t}$.

The RMS of the system was calculated after each step for an arbitrarily high number of steps. The number of steps was then calibrated over time to be approximately the amount required for the entropy of the system to reach its maximum (equilibrium) value.

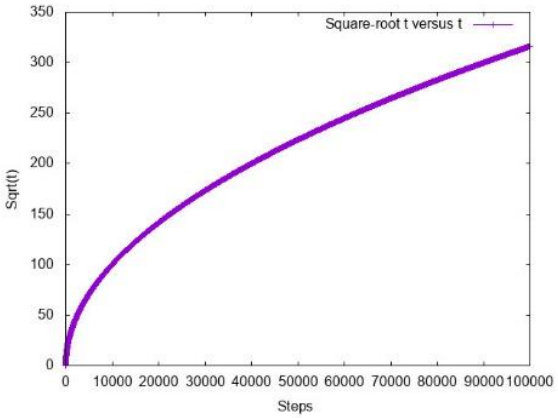


Figure 2. Graph of time (steps) versus $\sqrt{time(steps)}$

[6] N.J.Giordano & H.Nakanishi - Computational Physics 7.4

3. Entropy as a Function of Time

The 2nd Law of Thermodynamics states the change in entropy (ΔS) for a natural process is always ≥ 0. Also, as mentioned in the introduction the direction of a natural process such as the diffusion of particles is always towards equilibrium, or in other words, maximum entropy. Because the particles were initialised in a square "drop" the initial state of the system is very ordered and with low entropy. As the particles are allowed to disperse the entropy of the system was expected to increase rapidly at first, the rate of change would then decrease over time as the particles spread out to fill the entire container more or less evenly.

The equilibrium achieved is a dynamic equilibrium in which the particles are constantly in motion but the RMS displacement from the origin of the system remains approximately constant. The final result of this would be a graph of entropy versus time (steps) with the shape shown below.

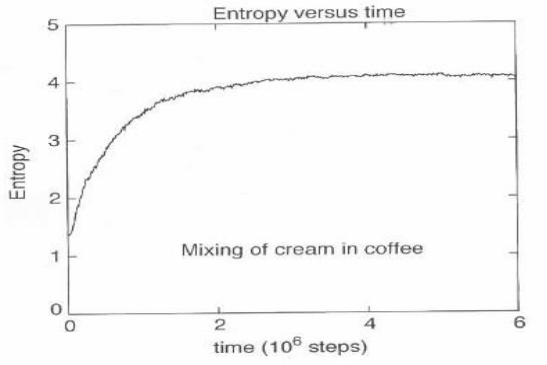


Figure 5. N.J.Giordano & H.Nakanishi – Graph of entropy versus time (steps) 7.6

Additional Objectives

1. Time to Reach Equilibrium

For a square container with constant area the amount of time to reach equilibrium will be relatively similar between multiple runs of the program. However if the area of the container were to be changed the time to reach equilibrium would also change. In order to show this change it was necessary to change the side length of the container and define a new origin in order to keep the initial position of the drop in the centre of the container. Simply by comparing graphs of entropy versus time(steps) for different sizes of the container it is possible to see that the number of steps required to reach entropy varies.

In theory it would be possible to calculate the number of steps at which rate of change of entropy ≈ 0, however due to the constant fluctuations of entropy values the rate of change is never close enough to 0 making this solution unviable. Because of this instead of looking for the number of steps at which rate of change of entropy is ≈ 0 the program will be allowed to run for a sufficiently large number of steps (eg. 10000) and the number of steps at which the rate of change is at a minimum will be determined using the central method of numerical differentiation: $\frac{DS}{DT} = \frac{S(t+\Delta t)-S(t-\Delta t)}{2\Delta t}$

In this case Δt should be small, however the smallest value available is the step size of 1. In addition, due to the large number of points and the relatively large fluctuations in entropy between each individual step getting the rate of change between only 2 points would not result in the most accurate representation of how the slope is changing over time. In this case it was more representative to increase the Δt to a larger value such that a larger gap exists between the 2 points. The appropriate Δt was chosen based on trial and error to be 25.

2. Change in Drop Size

In order to show that the change in drop size varies as $\sqrt{time(steps)}$ it was first necessary to define what constitutes the size of the drop. Additionally it is not fully clear whether "drop size" refers to the area of the circular drop or the radius of the drop. Since the area of a circle varies with r^2, depending of the choice of r, a graph of area versus time(steps) could have essentially the same form as a graph of mean-square displacement versus time. If RMS was chosen as the radius:

- Mean-square displacement = $\frac{(\sum_A displacement)^2}{number\_particles}$
- Area = $\pi r^2 = \pi(RMS)^2 = \pi\left(\sqrt{\frac{(\sum_A displacement)^2}{number\_particles}}\right)^2 = \pi\left(\frac{(\sum_A displacement)^2}{number\_particles}\right)$

As mean-square displacement is known to vary linearly with time(steps), it is impossible for this definition of drop size to vary as $\sqrt{time(steps)}$.

If "drop size" referred to the size of the radius however, it is possible that the size would vary as $\sqrt{time(steps)}$. RMS was found to vary as $\sqrt{time(steps)}$, therefore, if the RMS of the system was chosen as the radius of the drop, the objective would be fulfilled. The only question remaining is if this case is true for a different choice of radius.

Several possible definitions exist, one would be the circle with radius equal to half the distance between the two particles with the largest separation in the container. Another option would be to simply find the furthest particle from the origin, calculate it's displacement from the origin and use this as the radius of the circular drop. A final alternative is to use the RMS of the system as the radius of the circle. In this project the latter two options were considered. As RMS is already being calculated this was chosen as the most convenient method. The maximum displacement from the origin was used for comparison. The area of the circle occupied by the drop was then plotted against $\sqrt{time(steps)}$.

Method – Minimum Objectives

The particles are first initialised in a 5x5 square "drop" from which they are then allowed to disperse. The initial entropy of the system and RMS are then calculated based on the initial positions of the particles. The remainder of the program takes place within a WHILE loop which executes for the desired number of steps. The following general steps are executed during each run of the loop:

- Determine new positions of particles based on random number generation

- Calculate entropy of the system based on probability of container sites being occupied
- Calculate RMS of the system based on displacement from origin at centre of the container

1. Particle Distribution

As required by the project guidelines the particles are initially positioned in square shaped drop of 5x5 particles around the origin (101, 101) which is the centre of the container in which the particles are confined. The origin was set to this value because of the coordinate system being used. The entropy aspect of this program uses the x-coordinates and y-coordinates to mark which sites are occupied in the 2-Dimensional site[i][j] array. As it is not possible to define array elements using negative numbers it was necessary to use a coordinate system in which $x, y \geq 0$ at all times. Because of this it was not possible to use a standard coordinate system with origin (0, 0) and $-100 \leq x \leq 100$, $-100 \leq y \leq 100$.

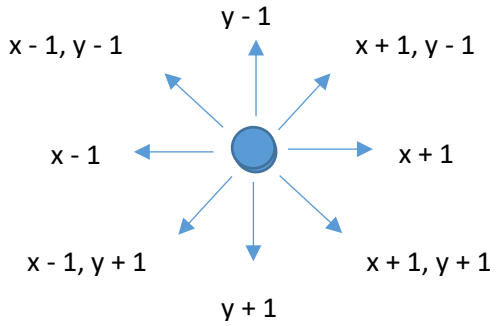
The 2D site[i][j] array contains an element for each position (i, j) in the container. Since the container has a 200x200 area site[i][j] has 200x200 elements. It's possible to imagine the container as a 200x200 grid with each particle occupying a particular "site" on the grid. The site[i][j] either stores the number 1 in element (i, j) if a particle has coordinates (i, j) or it stores 0 in element (i, j) if there is no particle with coordinates (i, j).

In order to ensure that $x, y \geq 0$ at all times, the container was defined to be a 200x200 unit square with values $1 \leq x \leq 201$ and$1 \leq y \leq 201$ with origin (101, 101). This square is the area in which the particles are free to move and diffuse. The full X and Y length of the grid is actually declared to be 202x202, the reason for this will be outlined in Objective 3 Calculation of Entropy versus Time. The particles were initialised into a 5x5 square with 16 particles occupying each of the 25 sites using the following routine:

- Declare variable *l* to use as a coefficient to increase the x-coordinate by the appropriate amount
    - Initialise *l* = 0
- Declare integer arrays *x_coord[]* and *y_coord[]* with size = number_particles (400) which will hold an x-coordinate and y-coordinate for each particle[7]
- Set particle coordinates row by row, i.e. first row of particles will have $y = 99$ and $99 \leq x \leq 103$, second row will have $y = 100$ and $99 \leq x \leq 103$ etc.
- This is done using two nested FOR loops which will do the following:
    - *x_coord[]* = 99 + *l* (*l* increases by 1 after every 16 repetitions of inner loop)
    - *y_coord[]* = 99
- Four additional copies of this loop must be carried out after the first one (One for each row of particles)
    - Reset *l* to *l* = 0 at the beginning of each new loop
- In each subsequent loop the value of y_coord[] will be increased by 1 up to a maximum of y_coord[] = 103

[7] First report proposed that positions would be recorded using a 2D "particles" array, this solution has instead been reassigned to the "sites" array recording occupied lattice sites. X and Y coordinate values are instead stored as explained above.

After being initialised in a square around the origin the particles are allowed to disperse. Each particle can move in 1 of 8 directions (diagram below) every time 1 step in the random walk occurs.



In the previous iterations of the program the particles moved using either one of the two methods indicated below:

These methods were replace by the one above as it is a more accurate representation of the motion of particles during diffusion. An even more accurate representation would be to allow a particle to move in any direction 360° around its current position by a random distance. Implementing this was however not viable as the algorithm used to calculate entropy relies on steps being of unit length.

Every time 1 step is taken in the random walk the position of all 400 particles is changed. The change in position is carried out using the following routine:

- Get random integer step_x between 1 and 8 using the RAND function
    - If step_x = 1 then increase x-coordinate by 1
    - If step_x = 2 then decrease x-coordinate by 1
    - If step_x = 3 increase y-coordinate by 1
    - If step_x = 4 decrease y-coordinate by 1
    - If step_x = 5 increase x-coordinate by 1, increase y-coordinate by 1
    - If step_x = 6 then decrease x-coordinate by 1, increase y-coordinate by 1
    - If step_x = 7 then increase x-coordinate by 1, decrease y-coordinate by 1
    - Else decrease x-coordinate by 1, decrease y-coordinate by 1

If the program is allowed to run for increasingly large amounts of steps the particles will eventually come into contact with the walls of the container. When this occurs it becomes necessary to impose a restriction on where the particles can move to prevent them from exiting the borders of the container. This was done by saving the initial X and Y coordinates of a particle in variables *x_previous* and *y_previous* before its position is changed. After the change in position has occurred the new position coordinates are checked to see if the particle has been taken outside the container outside the container. If the particle is outside the

container, it's position is reverted to the one it occupied before the change occurred. The routine executes as follows:

- Before change in position occurs save $x\_coord[i] = x\_previous \; and \; y\_coord[i] = y\_previous$
- Change position of particle
    - If new position is out of container reset $x\_coord[i] = x\_previous$, $y\_coord[i] = y\_previous$
    - Else if new position is inside container then keep new x_coord[i] and y_coord[i]

This results in particle motion as shown on the diagram below:

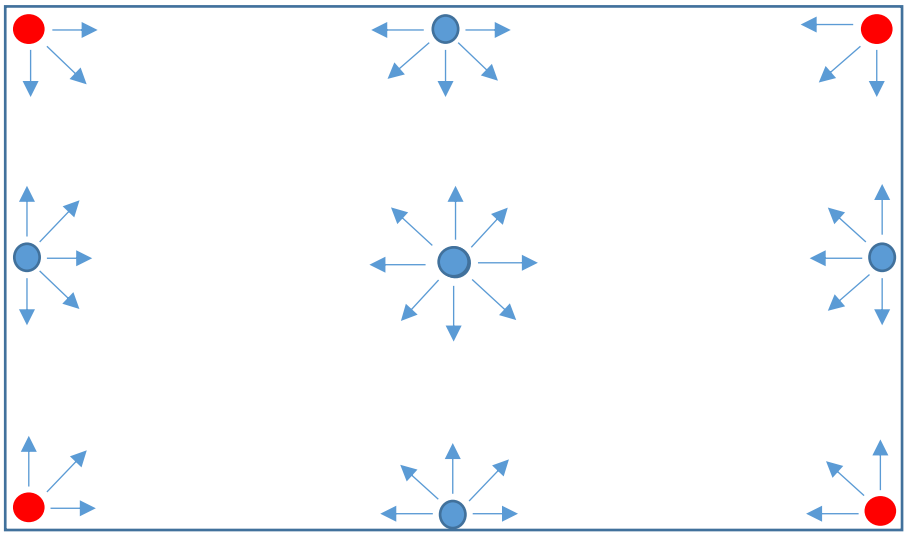


Figure 6. Possible directions of movement for particles in container

This routine is allowed to repeat for the desired amount of steps using a WHILE loop which encompasses the above routine.

- Declare a constant *k* and set *k* = 0 before the commencement of the loop
- Increase *k* by 1 after every repetition of the loop
- Repeat the loop while *k* < desired number of steps
- When *k* ≥ desired number of steps break out of the loop

The evolution of the positions of the particles can be observed by calling gnuplot to print a scatter plot JPEG of the positions after a set number of steps has occurred[8].

- An example of this would be halfway through the total number of steps:
    - If *k* = $(number\ of\ steps)$ call gnuplot to print current positions, entropy and RMS displacement

[8] In previous report it was suggested that the printing of graphs would be based on fractions of the total number of steps occurring, this has since changed to set numbers of steps occurring for increased simplicity.

The program has been set to print graphs of the above values at 200, 500, 1000, 3000 and 7000, 30000, 50000, 100000 steps. A sample of these is shown below. The new gnuplot script which enables manual setting of the scale has been implemented into the program with ranges set to $0 \leq x \leq 201$ and $0 \leq y \leq 201$. This allows the evolution of the positions of the particles to be observed without the automatic scaling which gnuplot usually incorporates into its graphs.

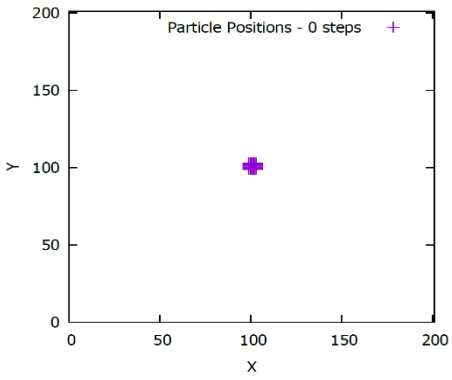


Figure 7a. Initial positions

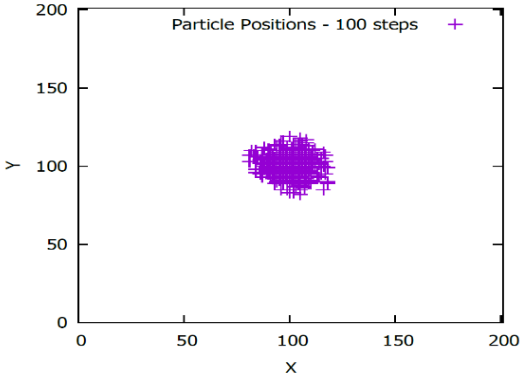


Figure 7b. Positions after 100 steps

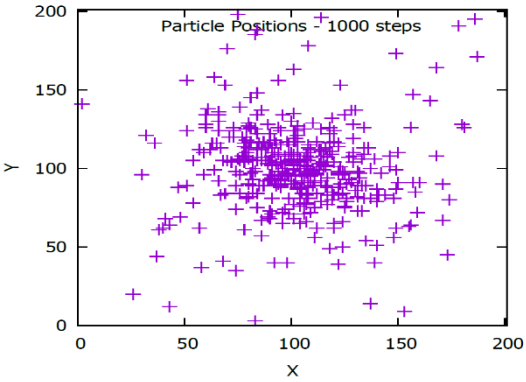


Figure 7c. Positions after 1000 steps

2. RMS

The RMS displacement of each particle was calculated after the positions of the particles have been updated in the WHILE loop. The RMS was calculated as follows:

- Declare timestep[number_steps] array which will store the step values from the main WHILE loop
- Declare rms[number_steps] array which will contain the RMS values of the system at each step.
- (Optional) Declare meansqr[number_steps] array which will contain mean-square displacement values of the system at each step
- Calculate displacement from origin using $s = \sqrt{(x - x\_origin)^2 + (y - y\_origin)^2}$
- Calculate mean – square displacement (meansqr_displacment[number_particles]) using $< s^2 > = \frac{(x-101)^2+(y-101)^2}{n}$
    - n = number of particles (number_particles)
- Find the sum of all the mean – square displacements of all the particles
    - Declare variable mean_square which will contain sum of the mean-square displacements of all the particles.
        - mean_square = mean_square + meansqr_displacement[number_particles]
    - Loop this for the total number of particles

- Find RMS of the system of particles $\sqrt{< s^2 >}$
    - root_mean_square = sqrt(mean_square)
- This calculation is repeated for each step of the main WHILE loop
- Store each root_mean_square value in a
- Call gnuplot to graph values in the rms[number_steps] array versus values in the timestep[number_steps] array.
- (Optional) Call gnuplot to graph values of meansqr[number_steps] versus values of timestep[number_steps].

Mean-Square displacement was shown to vary linearly with time(steps , as suggested by literature[9], while RMS varied with $\sqrt{time(steps)}$, as can be seen below on samples from the resulting graphs.

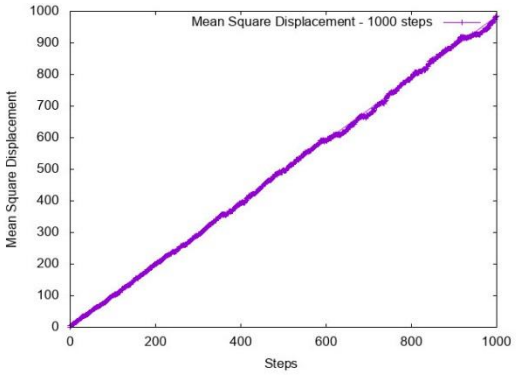


Figure 8a. Mean-Square Displacement versus time(steps)

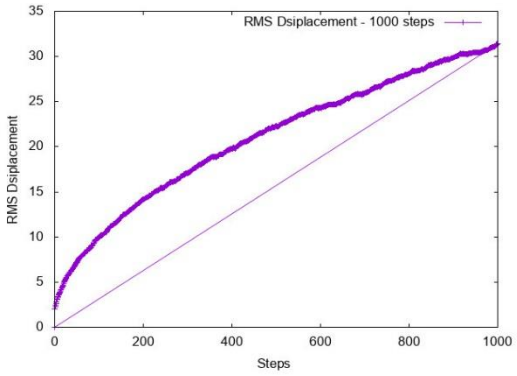
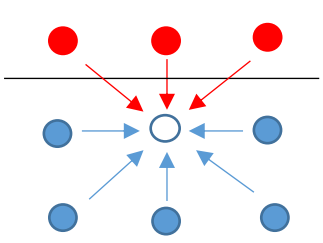


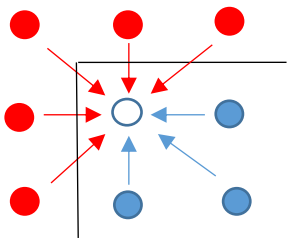
Figure 8b. RMS versus time(steps)

3. Entropy

The entropy of the system is calculated by dividing the lattice into a 202x202 grid and calculating the probability of each position in that grid being occupied in the next step. Each position on the grid is initialised with the value 0 in the site[x_length][y_length]. When the initial positions of the particles are set, site[x_length][y_length] is updated with the locations of the particles and each occupied position is set = 1 and unoccupied sites set = 0. The grid is 202x202 due to the way probability of site occupancy is calculated. Probability of site (i, j) being occupied is calculated by checking the 8 surrounding sites from which particles could



Wall of container



Corner of container

[9] N.J.Giordano & H.Nakanishi - Computational Physics 7.4

approach. Each occupied site adjacent to (i, j) contributes $\frac{1}{8}$ to the overall probability of (i, j) being occupied in the next step. However if (i, j) is on the border, as below, there is only 5 adjacent sites from which particles could approach. If (i, j) was located in a corner of the square lattice there would only be 3 adjacent sites from which particles could approach.

The program requires 8 adjacent sites from which to draw values of 0 or 1 and if the situation above arose then it would attempt to acquire values from locations which are undefined. This would result in the program either using incorrect values which will reduce the accuracy of the graph or cause an error and crash the program. By declaring an additional row and column for each side of the square lattice and setting all sites outside the border = 0 the problem is averted. It would theoretically be possible to prepare specific rules for the possible scenarios in which this could occur (similar to the way movement is restricted when near walls or corners) however this solution is simpler and reduces the risks of errors in the writing of the code.

The entropy of the system is calculated analytically using $S = -\sum_i Pi \ln(Pi)$ where P is the probability of a site being occupied. The probability of each site in the lattice being occupied in the next step using the following routine:

- Declare array probability[x_length][y_length] to contain the values of probability for a site to be occupied in the next step.
- The probability that (i, j) is occupied at step k is

$$P(i,j,n) = \frac{1}{8}[\,P(i+1,j\,,k-1) + P(i-1,j,k-1) + P(i,j+1,k-1) + P(i,j-1,k-1) + P(i+1,j+1,k-1) + P(i+1,j-1,k-1) + P(i-1,j+1,k-1) + P(i-1,j-1,k-1)]$$

  - The value $P(i+1,\ j,\ k-1)$ is determined by the value stored in site[i + 1][j] in the previous step k_{-1}.
  - If that site was occupied then $P = 1$, otherwise $P = 0$.
  - It is the same case for the other values of P. Therefore $0 \leq P(i,j,n) \leq 1$ at all times.
- By incorporating the site[x_length][y_length] array into the calculation, the formula will have the form:

$$probability[i][j] = \frac{1}{8}\,[\,P(i+1,j\,,k-1) + P(i-1,j,k-1) + P(i,j+1,k-1) + P(i,j-1,k-1) + P(i+1,j+1,k-1) + P(i+1,j-1,k-1) + P(i-1,j+1,k-1) + P(i-1,j-1,k-1)]$$

- Next it is necessary to calculate the natural log of the probability values, multiply by -1 and find the sum over all the available sites. As some probabilities will be 0 it is necessary to eliminate these as the ln function is undefined for $ln(0)$.

- Declare *entropy* variable which will contain the entropy of the system at step k
- Declare array probability_sum[x_length][y_length] which will contain $-Pi\ln(Pi)$ values for each site
    - If $probability[i][j] \neq 0$, then $probability\_sum[i][j] = -1(probability[i][j] * log(probability[i][j]))$
    - Else $probability\_sum[i][j] = 0$
    - Sum all $-Pi\ln(Pi)$ values to find entropy: $entropy = entropy + probability\_sum[i][i]$
    - Store this value in array entropy_t[number_steps]
- Repeat this for every step in the loop
- Call gnuplot to plot entropy_t[number_steps] versus timestep[number_steps]

Graphs of entropy versus time (steps) are graphed at several step intervals. After increasing numbers of steps the rate of increase of entropy becomes lower until eventually it is ≈ 0 and the particles have reached an equilibrium distribution in the container.

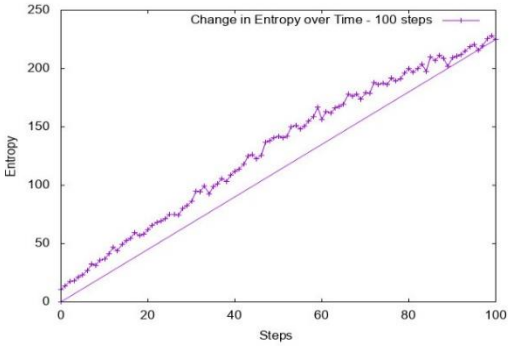


Figure 9a. Entropy - 100 steps

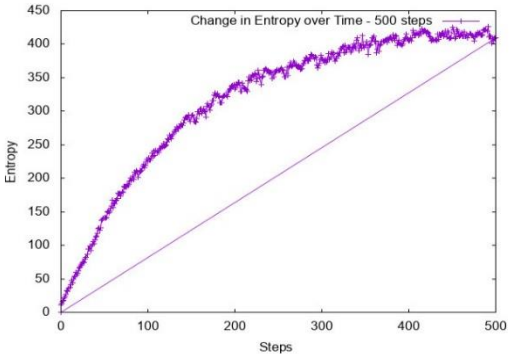


Figure 9b. Entropy - 500 steps

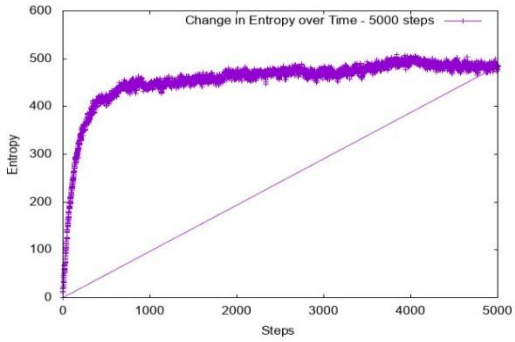


Figure 9c. Entropy - 5000 steps

The entropy of the system can clearly be seen to be reaching a constant value after approximately 3000 - 5000 steps.

Method – Additional Objectives

1. Equilibrium

The amount of steps required to reach equilibrium for varying sizes of the container were calculated using the central method of numerical differentiation. Once the system was allowed to run for the desired number of steps the minimum rate of change of entropy was calculated and the number of steps at which this occurred was recorded. These values were written into a .csv file and the calculation was repeated 10 times to get an average for the number of steps required. The dimensions were increased in quantities of 50 units up to a maximum of 850. The rate of change and minimum rate of change were acquired using the following routine:

- Declare array entropy_prime[number_steps] to hold values of rate of change
- Declare integer variable dt to hold the step size between points from which rate of change is calculated.
- Declare integer variable min_steps to hold value of steps required to reach equilibrium
    - Using a FOR loop Calculate $entropy\_prime = \frac{entropy_{t[i+dt]} - entropy[i-dt]}{2dt}$ for $i < number\_steps$ where $dt = 20$
    - Find minimum value of entropy_prime[number_steps] by comparing the entropy_prime[20] with entropy_prime[i].
        - If entropy_prime[20] < entropy_prime[i]:
            - entropy_prime[20] = entropy_prime[i]
            - min_steps = i
        - Else entropy_prime[20] = entropy_prime[20]
        - min_steps = min_steps
    - Values of entropy_prime[20], min_step and x_length are printed into a .csv
- This procedure is repeated 10 times for a single value of x_length and y_length
- The mean of the values of min_steps is calculated in Excel and plotted against x_length^2
- X_length and y_length are then manually increased by 50 units and the procedure repeated.

An abridged version of the code necessary to calculate the rate of change of entropy, find the step at which it is minimum and printing this to a .csv file has been included. In order to speed up the process of finding the mean of the steps at which equilibrium is reached a FOR loop was wrapped around the main section of the program. This has since been removed as it significantly increases the run time of the other sections of the program. Only 1 value of the steps necessary to reach equilibrium will be printed for the default lattice size of 200x200.

2. Drop Size

The size of the drop was calculated as the area of the circle with the RMS of the system at a particular step as the radius. The area is calculated simply by using the formula for the area of a circle $A = \pi r^2$. The formula is applied as follows:

- Declare constant pi = 3.1415
- Declare drop_size_RMS[number_steps] to hold values of area for r = RMS at each step
    - Calculate initial drop size: drop_size_RMS[0] = pi * rms[0]*rms[0]
    - Calculate drop size at each step k: drop_size_RMS[k] = pi*rms[k]*rms[k]
    - Call gnuplot to plot graph of drop_size_RMS[k] versus time(steps)
    - It isn't necessary to plot RMS versus time(steps) as this is being done already by a different portion of the program
- Declare drop_size_disp[number_steps] to hold values of area for r = max. displacement at each step
    - Calculate displacement from origin for each particle using FOR loop
        - $disp[i] = \sqrt{(x\_coord[i] - x\_origin)^2 + (y\_coord[i] - y\_origin)^2}$

- Find max. displacement from origin by comparing disp[0] with disp[i] using FOR loop
  - If disp[0] < displ [i] then set disp[0] = disp[i]
  - Else keep disp[0] = disp[0]
- Calculate initial drop size: drop_size_disp[0] = pi * rms[0]*rms[0]
- Calculate drop size at each step k: drop_size_disp[k] = pi*rms[k]*[k]
- Call gnuplot to plot graph of drop_size_disp[k] versus time(steps)
- Call gnuplot to plot graph of disp[k] versus time(steps)

Results

1. Particle Distribution

The distribution of the particles evolves in the expected manner for the first few thousand steps. The particles spread outwards from their initial positions towards the edges of the container until eventually the distribution reaches maximum entropy or equilibrium.

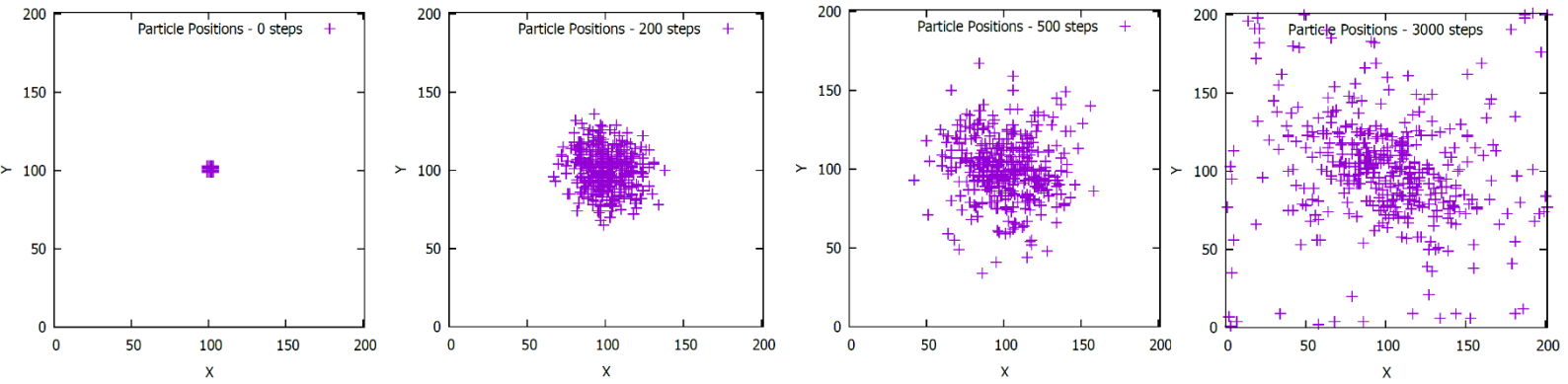


Figure 10. Positions after 0, 200, 500 and 3000 steps

For relatively short term simulations of diffusion the program returns the expected results. However at very large numbers of steps (> 30000). Graphs of entropy versus time show that the entropy of the system reaches a local minimum value at regular step intervals. This minimum value is greater than the initial entropy of the system however it is still significantly

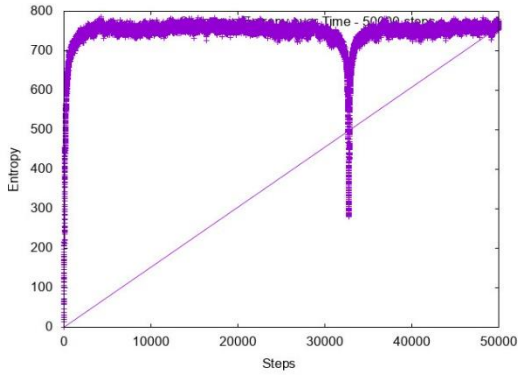


Figure 11a. Entropy - 30000 steps

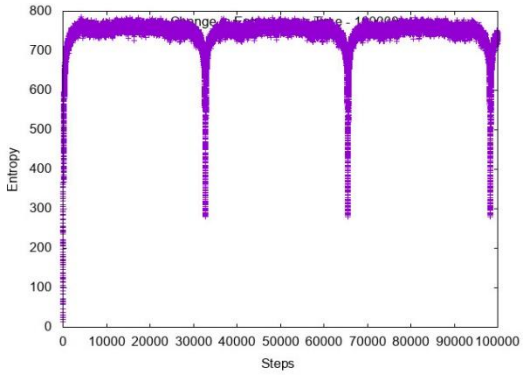


Figure 11b. Entropy - 50000 steps

lower than the entropy which the system has at equilibrium.

From these graphs it was possible to identify the approximate number of steps at which this occurred and obtain the distributions of the particles at these step numbers. The plots show

that at step intervals ≈ 33000 the particles form an oblong shape stretching across the container. This clustering of the particles in such an ordered manner is the reason for the significantly reduced entropy.

The precise reason for this clustering is unknown. However there are at least two factors with a high probability of contributing to this behaviour. The first factor is the way in which particles move at the walls and corners of the container. Near the centre of the container a particle has a $\frac{1}{8}$ probability of moving in any 1 of 8 directions. Depending on where it is located in the container the probability of moving towards the origin is higher or lower. However as long as the particle is not near a wall or a corner the probability of moving away from the origin is higher than moving towards it.

Near walls and corners 3-5 possible directions of movement are eliminated making the probability of moving towards the origin higher than when near the centre of the container. This change in the probability of moving towards the centre could cause the net direction of movement of particles to be towards the centre for a very short time, explaining the momentary clustering. As shown by the entropy graphs this clustering only lasts for a very short time and the particles are quick to disperse again. If the simulation were allowed to run for very large numbers of steps ($> 1 \times 10^6$) these local minima would eventually occupy such a small portion of the distribution as to be negligible.

The second possible factor contributing to the clustering is a limitation of random number generation. True random number generation is nearly impossible to achieve and in the long term a pattern is always revealed. It's possible that the RAND function becomes repetitive after large numbers of uses causing the periodic behaviour shown in the graphs.

2. RMS

The RMS of the system was shown to vary as $\sqrt{time(steps)}$. Graphs of RMS versus time(steps) maintain this trend for steps ≤ 3000.

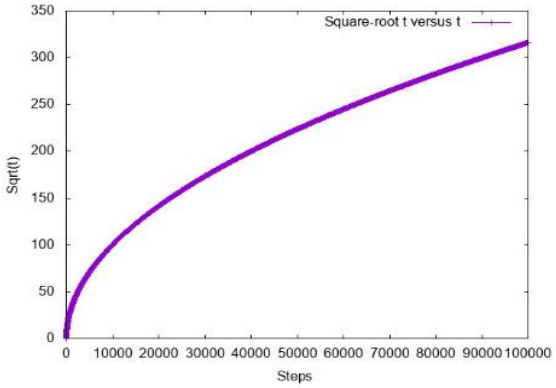


Figure 2. Graph of time (steps) versus $\sqrt{time(steps)}$

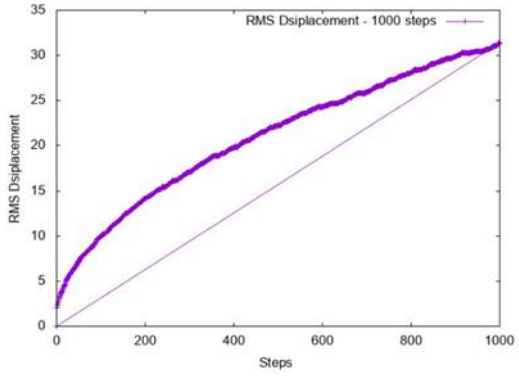


Figure 12a. Graph RMS versus time(steps)

However, as was the case with the distribution of particles and entropy, at increasingly large numbers of steps (≥ 4000) irregular behaviour is seen in the graphs of RMS versus time(steps).

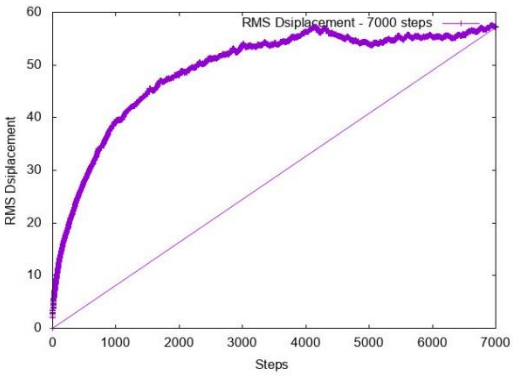


Figure 12b. RMS versus time(steps) - 7000 steps

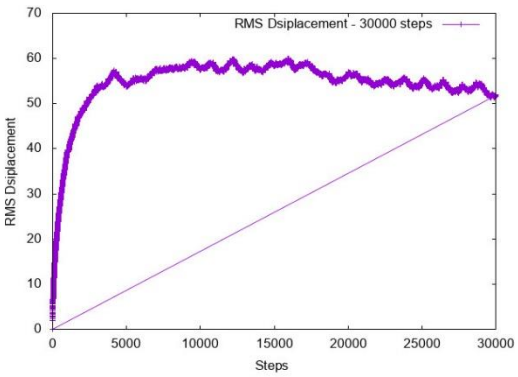


Figure 12c. RMS versus time(steps) – 30000 steps

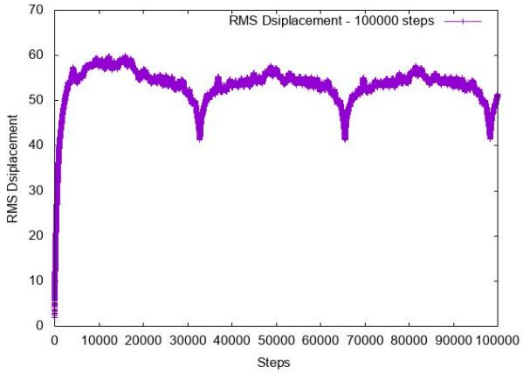


Figure 12d. RMS versus time(steps) – 100000 steps

The RMS quickly reaches a maximum after approx. 3000 steps, this is to be expected as the system has already reached equilibrium by then. At equilibrium the RMS is not expected to experience much change as the particles have more or less spread out as much as possible. Some fluctuation is possible as the particles are constantly moving and at times the net movement in a particular step can be towards the origin causing the RMS to drop. The reverse is also possible where the net movement can be outwards from the origin. These fluctuation are not very noteworthy. The sharp drops which can be seen in *Figure 10c* are however. The step numbers at which they occur correspond roughly to the minima of entropy and the steps at which the particles are seen to cluster together. Therefore it is likely that the reasons for the local minima in RMS are the same as for those in entropy. Namely the changes in probability of movement direction and the limits of the RAND function. As with entropy the minima only exist for very short intervals and occupy a very small portion of the overall graph. At very large numbers of steps the effect of the minima would be even smaller.

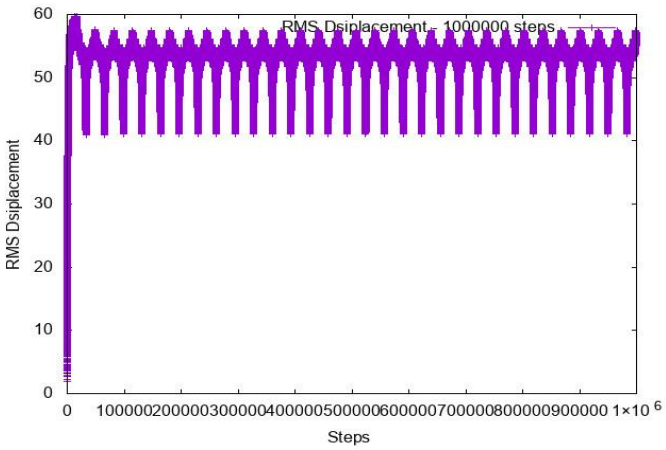


Figure 12e. Graph RMS versus time(steps) - 1×10^6 steps

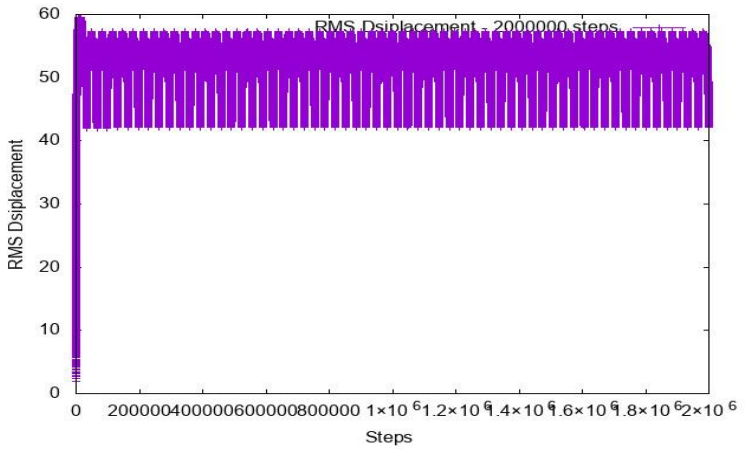
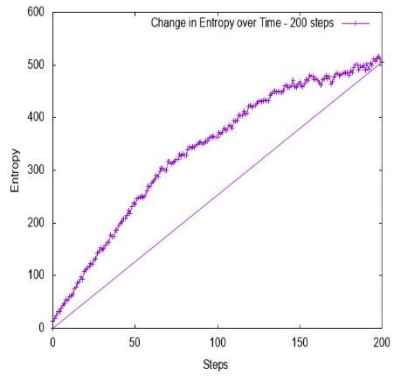


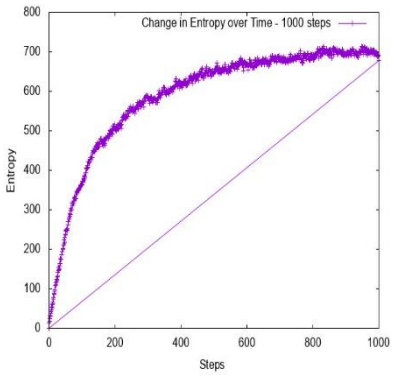
Figure 12f. Graph RMS versus time(steps) - 1×10^6 steps

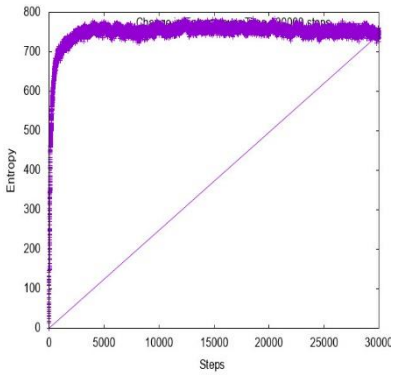
The above graphs show RMS versus displacement at steps 1×10^6 and 2×10^6 respectively. The minimum points are still noticeable due to the large cross shaped plot points used by gnuplot and the way in which the graphs are compressed in order to maintain as specific aspect ratio. As a result the area of the graph occupied by minimum points is exaggerated. This effect could be nullified by only calculating RMS at specific intervals as opposed to calculating it at each step.

3. Entropy

The entropy was seen to initially increase rapidly in an almost a linear manner for the first 200 steps after which point the rate of increase becomes lower over time until equilibrium was reached. Equilibrium is maintained for approx. 30000 steps after which point entropy began to decrease rapidly to a local minimum as outlined in section 1. of Results (1. Particle Distribution).







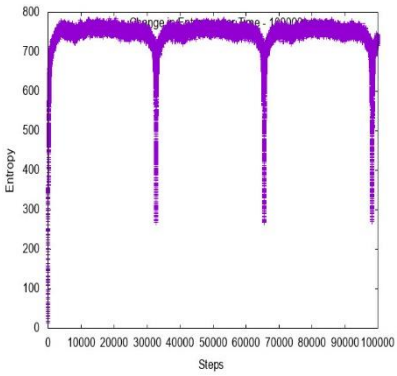
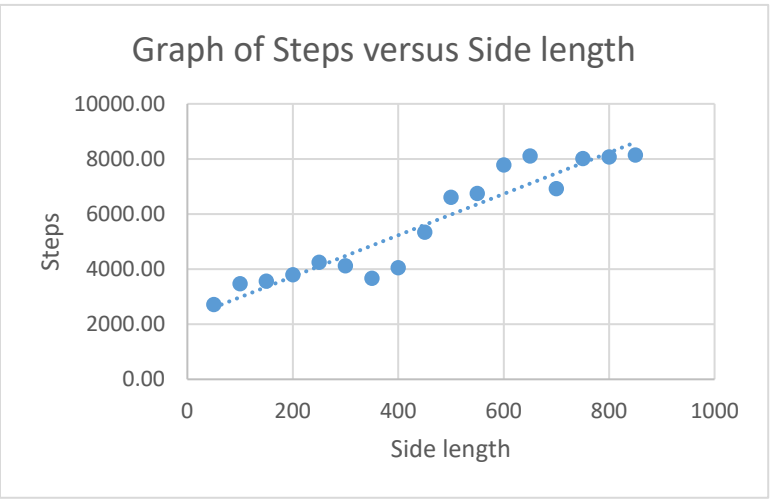


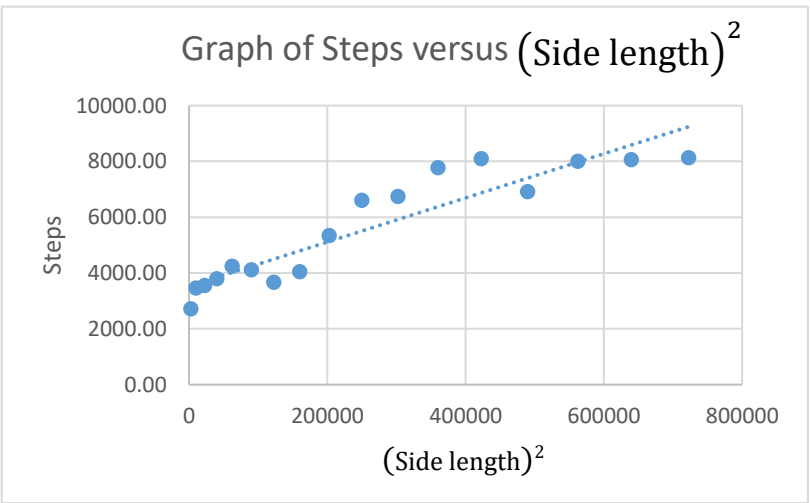
Figure 13a. Entropy-100 steps *Figure 13b. Entropy*-1000 steps *Figure 13c. Entropy*-30000 steps *Figure 13d. Entropy*-10000 steps

Long term graphs of entropy versus time suffer the same problem as those of RMS versus time in that the portions of the graph occupied by the minimum points are exaggerated while the larger sections are compressed. Similarly this could be solved by calculating and plotting entropy at specific intervals instead of at every step.

4. Equilibrium

As expected the amount of steps required to reach equilibrium increases with increasing lattice size. The method employed revealed a roughly linear relationship between the time taken to reach equilibrium and the size of the lattice which can be seen on the graph below. The Excel spreadsheets containing the values relevant to the graph will be included with this report.





Even though the relationship appears linear it is possible that in the long term this is not the case. For increasingly large lattice sizes it's possible that the relationship will turn out to be exponential as opposed to linear. To verify this it would be necessary to use a more accurate way of determining the equilibrium point of a particular lattice size. The fluctuation between entropy values and the significant number of outliers present in the current method of

calculating the equilibrium has too high an error to produce more than a very rough estimate of the true behaviour.

The best way to determine this relationship accurately would be to write the values of entropy versus time(steps) into plotting software which could fit a curve to the graph and generate an equation for it. Doing so would eliminate the need to calculate a mean value for the number of steps at which rate of change of entropy is = 0, this value being very vulnerable to being affected by outliers.

5. Drop Size

The variation of drop size over time(steps) showed interesting results across the 4 cases under consideration. The graph of RMS versus time varied as $\sqrt{time(steps)}$, which was already shown by an earlier portion of the program:

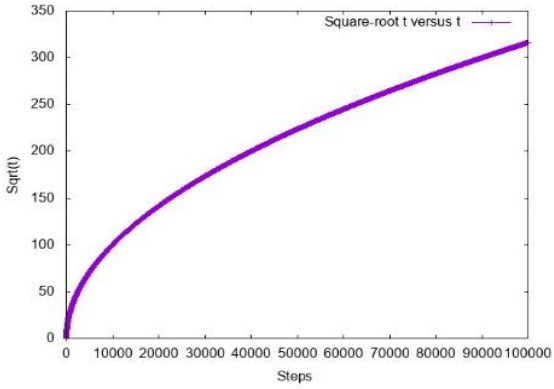


Figure 2. Graph of time (steps) versus $\sqrt{time(steps)}$

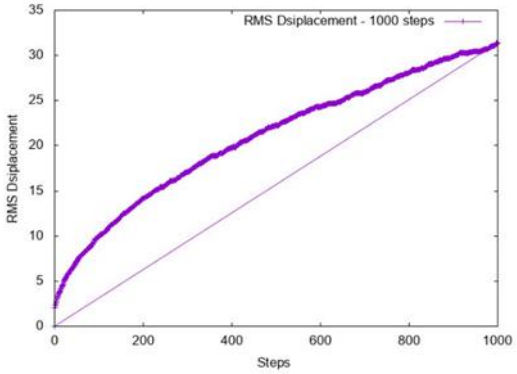


Figure 12a. Graph RMS versus time(steps)

Once the particles spread out sufficiently and the drop size ≈ size of the container RMS stopped varying as $\sqrt{time(steps)}$ and instead fluctuated near a constant value as shown in Figure 12c. As long term behaviour had the particle clustering periodically the RMS and size of the drop mirrored this periodic behaviour as can be seen in Figure 12d.

The graph of max. displacement varied linearly before reaching a constant value in a similar manner to the graph of RMS versus time(steps):

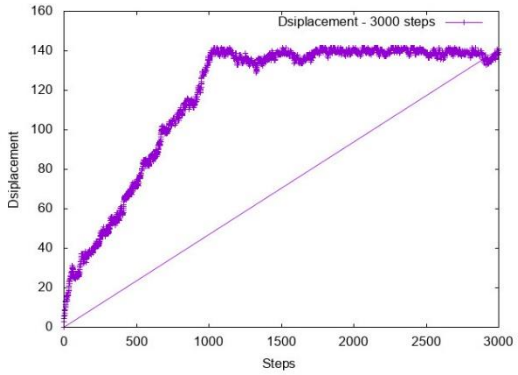


Figure 14a. Graph of Displacement versus time(steps)

As predicted the size of the drop when defined in terms of a circle of area πr^2 where $r = RMS$ essentially results in a graph of mean-square displacement versus time(steps) which

behaves linearly while the drop is smaller than the container. After the drop has grown sufficiently the behaviour changes in a similar way to the previous 2 cases.

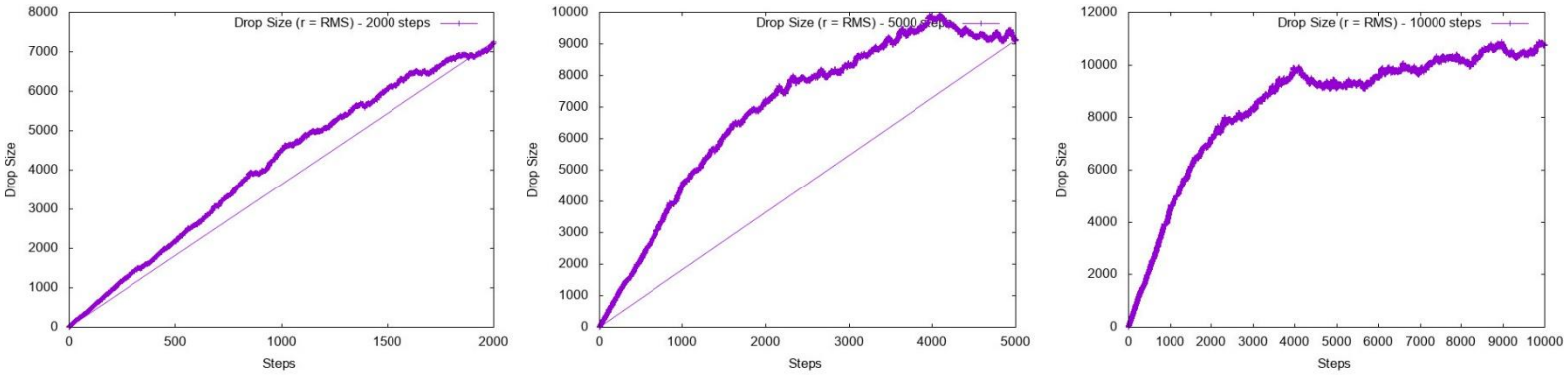


Figure 14b. Graphs of drop size (r=RMS) versus time(steps)

Finally a plot of drop size defined in terms of a circle with radius $r = max.displacement$ resulted in a graph which initially had a quadratic shape which then peaked at a constant value once the drop acquired a size similar to that of the container.

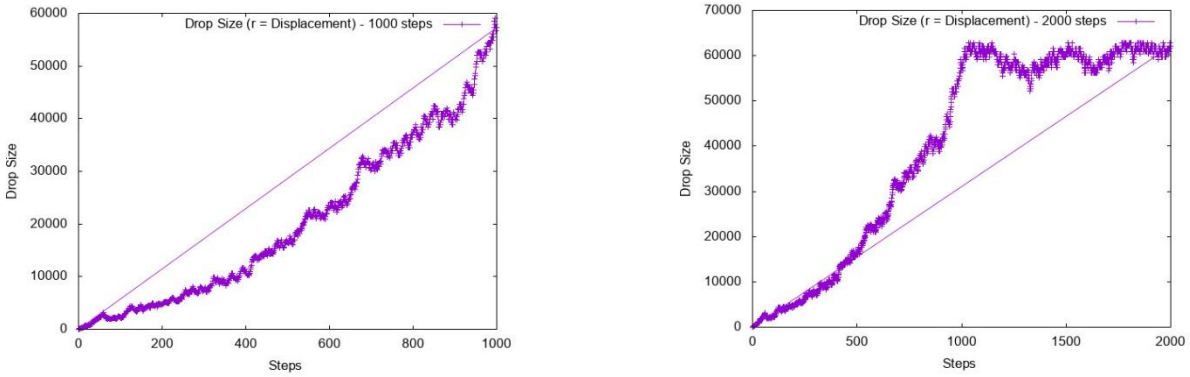


Figure 14c. Graphs of drop size (r=max. displacement) versus time(steps)

The conclusion which can be drawn is that while the size of the drop is smaller than that of the container it experiences a pattern of growth related to the choice of radius. The size then maintains a fluctuating maximum value once the drop has reached the borders of the container at which it stays, with exception to the clustering effect observed at large step numbers.