The Expectation for the Center of Mass of Finite Integer Grids

Finnegan Maximilan Muller Hardy
Bard College

The Expectation for the Center of Mass of Finite Integer Grids

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Finn Hardy

Annandale-on-Hudson, New York
May, 2017

Abstract

The center of mass of an object is the point at which the average distribution of the mass of that object is located. The goal of this project is to determine the expected value for the center of mass of random configurations on a d dimensional finite integer lattice $L^d = \{0, ..., n\}^d$, where at each integral node one of two types of mass m and M is randomly placed with probability p and $1 - p$, respectively. Here, we are assuming that d and n are positive integers, p is a real number in the interval $(0, 1)$ and m and M are nonnegative real numbers. In order to empirically find this expected value, we began by writing a Python code which generated one-, two-, and three-dimensional finite integer lattices, randomly placed fixed masses m and M with fixed probabilities p and $1 - p$, respectively, at each integral node, and then calculated the center of mass of this random configuration. For different values of n, m, M, and p, we ran our code several thousand times and created a number of large data sets containing centers of mass of random configurations. Using statistical analysis, we then estimated the mean of these data sets which seemed to be a number close to $\frac{n}{2}$, regardless of the values of m, M, and p. We then theoretically proved that the expected value for the center of mass of random configurations on L^d is always in the middle of the grid, i.e., the expected value for the center of mass for each component of this lattice is $\frac{n}{2}$.

Contents

Dedication

For my mother.

Acknowledgments

Thank you to Amir Barghi and Hal Haggard for their many hours of patience.

1 Introduction

Imagine you want to balance an elephant on the point of a needle, how would you do it? The clever answer of course is "very carefully"; however, there is a much more systematic and concrete way of solving such a problem. The answer: Center of Mass! To understand the concept of center of mass picture a pen balancing lengthwise on your finger. At each point along the pen there is a slightly different concentration of matter, but near the center of the pen there is an average concentration off all of the matter. Thus, if you place your finger there, voila, it balances. We can, therefore, understand center of mass to be the point in an object or system of objects at which the average distribution of matter is located.

In physics, the principle of center of mass is used to describe the complex motion of an object through space. If you throw a bat into the air it may appear to be moving in an irregular way, however, its center of mass retains a parabolic ark, typical of any object thrown upwards. Engineers use center of mass to maximize performance on sports-cars by designing vehicles with a center of mass extremely close to the ground. This allows for better handling of the car, and ensures the driver feels as connected to the road as possible. Center of mass is also used by divers and gymnasts who wish to master their balance and rotation. When a diver jumps and spins through the air she is maximizing rotating about her center of mass. A gymnast uses the center of mass of his own body in order to coordinate complicated routines on a narrow bar. Even you have used the concept of center of mass in order to maintain your balance as you walk and run.

Another interesting concept described in physics is something you may be much more familiar with, magnets! Magnets have extremely practical uses even beyond holding up photos on your fridge. They are used in power plants to generate the electricity we all use to power our homes. Some trains use magnets to levitate above the track and move along at incredible speeds. Even the magnetic field of the earth is useful for orienting lost travelers using a compass. Formally speaking; magnetism is the attractive and repulsive force between two objects caused by the motion of electric charges within each object, however, for the purposes of this paper we are more specifically interested in how electric spins affect the overall charge of an object. An Ising model is most commonly used to determine this exact phenomenon, and with it we can better understand how and why certain object possess a specific overall charge.

Before we get started it is important that we define a number of terms we will be using throughout this paper.

Definition 1.0.1. A grid is a coordinate plane consisting of small squares with an x-axis, y-axis, and z-axis. Let $d \geq 1$ and let L^d be the grid $[n] \times [n] \times [n] \times ... \times [n] = [n]^d$ where $[n] = \{0, ..., n\}$.

Definition 1.0.2. A vertex can be understood as the common endpoint of two or more rays or line segments. In this sense a grid is essentially a collection of vertices. An integral vertex is an element in $[n]^d$ where $[n]^d = \{(x_1, x_2, ..., x_d) \mid x_i \in [n]\}$.

Definition 1.0.3. The center of mass is the point in an object or system at which the average distribution of matter is located. In the discrete case we define the center of mass as

$$CM = \frac{x_1 m + x_2 m + x_3 m + ... + x_n m}{nm}$$

Where x are the locations of mass, m are the mass values, and n is the number of total locations of mass.

Informally, a random variable X is the outcome an experiment of chance, that takes numeric values. In other words X is a function from a sample space Ω, the set of all possible outcomes, to the real numbers $\mathbb{R}$. We say X is a discrete random variable when the set of all values it takes in $\mathbb{R}$ is either finite or countably infinite. The expected value of a random variable is the long term average outcome of said value as the result of some experiment or function. The standard deviation is a measure of how dispersed the output values of an experiment or function are from the mode output. More formally, we have the following definition:

Definition 1.0.4. Let X be a discrete random variable defined on a sample space Ω, with a probability function $\mathbb{P}$. If we denote the range of X by E, i.e., $E = X(\Omega)$, the expected value of X denoted by $\mathbb{E}(X)$ is $\mu = \sum_{x \in E} x \cdot \mathbb{P}(X = x)$. We define variance as the $Var(X) = \sum_{x \in E}(x - \mu)^2 \cdot \mathbb{P}(X = x)$. We define the standard deviation as $SD(X) = \sqrt{Var(X)} = \sigma$.

1.1 Finding Center of Mass in General

With all these useful application for center of mass, it is important to understand where it is, and how to find it. The first step towards solving these questions is to create a simplified version of an object or system. Let us start with a one dimensional finite integer grid L^1, with coordinate values of 0 through n. Next, we randomly place a mass of either m or M at each coordinate value. We then want to determine where the average concentration of mass is located along the line. To do so we find the total sum of each mass multiplied by its corresponding coordinate, and then divide by the total mass at every point. Thus we have found the coordinate for the mean distribution of mass along the line.

Now let us get a little more complicated. Suppose we have a finite integer two-dimensional grid L^2 with dimensions $(n + 1) \times (n + 1)$, where each coordinate is represented by an x and y value. Again, we randomly generate a mass of either m or M at each point. How do we find the center of mass in this case? At first this may seem more difficult than finding the center of mass for a one dimensional line since we are now dealing with both x and y values, but its not! In the case of a two dimensional grid we apply the same formula we used in the one-dimensional case but find the center of mass for the x coordinate and y coordinate values separately. For example, suppose we have a 3×3 grid whose x coordinate values range from 0 to 2, and y

coordinate values range from 0 to 2. To find the coordinate value of the grid's center of mass we simply multiply the mass at each point by its corresponding x-coordinate, find the sum of all of these value and divide by the total mass. Then we do the same for each mass' corresponding y-coordinate. Thus we have both an x and y coordinate for the center of mass. This is therefore the center of mass of the entire grid.

Applying what we have already discussed for finding the center of mass in two-dimensions, we can similarly find the center of mass for a finite integer three dimensional grid L^3. The only difference between the center of mass for a two-dimensional grid versus a three-dimensional grid is that you apply the center of mass formula for the x,y, and z coordinate values. Once you have a center of mass value for all three, that is the coordinate of the center of mass for your three dimensional space.

1.2 Understand the Ising Model in General

An Ising model is a graphical representation of the magnetic spin states of all the atoms within a certain object which in turn helps to model and describe the ferromagnetism of said object or system. In order to better understand the Ising model, first picture a two dimensional grid. At each vertex on that grid there is a particle with a spin state of either $+1$ or -1. We want to determine the overall charge of our grid and to do so we have to determine the relationship between charges of neighboring vertices. The total charge is determined by the amount of neighboring vertices that have the same spin state. For instance, if a vertex in the top right corner of the grid has a spin of $+1$, and the vertex to its right has a spin of $+1$, then the overall charge of the system gets a $+1$. This process is repeated for every single vertex along the grid. If, for instance, the bottom right most vertex has a spin of -1 and the vertex above it has a particle with a spin of $+1$, then no additional charge is added to the overall state of the system. In this sense we have described the structure of an Ising model.

2 Simulations

2.1 Simulating the Center of Mass

In order to calculate center of mass for a grid L^d for $d \geq 1, 2, 3$ we would have to find the product and sum and quotient of essentially an infinite number of values. To be able to feasibly calculate, in the ballpark of hundreds of thousands of different centers of mass, the only option was to use computers. We decided to use python code in order to simulate a grid of two or three dimensions with any number of vertices. The basic structuring of our code went as follows.

We determined that the best way to simulate a grid was through a list of values that contained the coordinate and mass of every single point within a grid. In order to generate every single coordinate for some n dimensional grid we used three nested loops. The significance of three nested loops was in correspondence with the three dimensions, x,y, and z. Each of these loops iterated through a range of values from 0 to n in accordance with the x-dimension, y-dimension, and z-dimension of the grid. For example, a 10×10 cube would have all three loops ranging from 0 to 10 since this includes 10 total point values. Please refer to the pseudo code below for clarification.

2.1.1 Pseudo Code of Grid Structure

```
+X LOOP: (iterates through $0 \longrightarrow n$)
  Starts with 0, Last value to change
++Y LOOP: (iterates through $0 \longrightarrow n$)
    Starts with 0, Second to last value to change
+++Z LOOP: (iterates through $0 \longrightarrow n$)
      Starts with 0, First value to increase.
Output: $(x, y, z)$
```

$[(0,0,0),(0,0,1),...,(0,0,n),(0,1,0),(0,1,1),...,(0,n,1),...,(0,2,1),...,(0,n,n),...,(n,n,n)]$

In this way we were able to generate every combination of (x, y, z) there was, and thus the entire grid.

Once we constructed these loops we had to design a function that would randomly generate a mass value. Using simple probabilistic practices we knew that when generating two random values, one value would have a probabilistic outcome of m, and the other M, with probability p and $1 - p$, respectively. Thus, our probabilistic mass assigning function incorporated this in addition to the `random.random` built in python function. For practical reasons we had to use rational values for p, instead of real values. In this way, our function took a fractional input and compared it to a value between 0 and 1 that was generated by `random.random` function. Based on the outcome of this comparison, a mass of m or M was placed at each vertex. If for instance, if we input the ratio $\frac{1}{2}$, then there would be a 50 percent chance of there being a mass of m versus a mass of M at each point. Observe the following pseudo code for clarification.

2.1.2 Pseudo Code of Probability Function

```
+WeightFunction (probability input):
++rand = rand.random
  Generates random number between (0,$n$) +++ If rand $\leq$ probability input:
    Generate mass $m$
```

```
+++ Else: Generate mass $M$
```

After successfully writing this function we had to incorporate some way of calculating the center of mass. Since this is a relatively simple summation and product calculation we added a running sum into the nested loops. The purpose of this was as each point was generated and a mass assigned to it, the coordinate values of x,y, and z at that point were each separately multiplied by the mass, and assigned to three different variables each representing the product of the coordinate and mass. These three separate product values corresponding to the three different coordinates were summed over each time the loop iterated over itself again. In this way the program kept track of the total value of the product of each element and its corresponding coordinate. In addition to this, the mass assigned to that point was added to a running sum value that represented to total mass of the system. For example, if for some iteration through our loop a mass of 1 was generated, then that mass of 1 was added to all the previous masses. Once the program had successfully generated every point in the grid we simply called for the function to divide the sum product pair for each coordinate by the total mass. This is exactly how you would find center of mass by hand. This then returned three different coordinates where the center of mass was located for our given grid.

2.1.3 Pseudo Code of Center of Mass Calculation

```
+ X LOOP:
++ Y LOOP:
+++ Z LOOP:
++++ xproduct = $x$-coordinate value $\times$ mass
++++ yproduct = $y$-coordinate value $\times$ mass
++++ zproduct = $z$-coordinate value $\times$ mass
++++ xnumerator = xnumerator + xproduct (Running sum of $x$-coordinate $\times$ mass value)
++++ ynumerator = ynumerator + yproduct (Running sum of $y$-coordinate $\times$ mass value)
++++ znumerator = znumerator + zproduct (Running sum of $z$-coordinate $\times$ mass value)
++++ totalmass = total mass + mass
```

$+\text{xcenterofmass} = \frac{\text{xnumerator}}{\text{totalmass}}$

$+\text{ycenterofmass} = \frac{\text{ynumerator}}{\text{totalmass}}$

$+\text{xcenterofmass} = \frac{\text{znumerator}}{\text{totalmass}}$

Output: Center of mass $= (x, y, z)$

2.2 Simulating the Ising Model

Simulation of the Ising model was done in a similar fashion to our simulation of center of mass in that we again generated a grid and randomly assigned a value to each vertex. However, this time we used only values of +1 or -1 to represent the positive or negative spin states of particles in an object. An added complication was that we had to determine the relationship between the spin of each particle and all of its neighboring particles. Since we were dealing with a two dimensional finite integer grid, we determined that there would have to be nine different conditions overall in order to detect the spin states of neighboring vertices. These nine conditions pertained to the particles at each corner, the particles along the top of the grid excluding the ones in the right and left corners, the particles on the bottom again excluding the corner vertices, the vertices along the sides excluding the top and bottom vertices, and the particles in the middle. The reason why each of these nine parts of the grid required different conditions was because there may not be neighboring vertices in every direction. For example, the bottom right most vertex only has neighbors above and to the left of it, while a vertex in the middle has neighbors all around it. Therefore, if we designed our program to search for every neighboring vertex in every direction we would have major errors in the case where there are no particles to the left, right, above, or below. In addition to this we also had to make sure that particles weren't being counted twice. Thus each vertex only compared its own spin to the spin of particles to the right and below it.

Once we had successfully setup conditions for each of these nine cases we created a variable called "charge", with an initial value of zero. As our code iterated through our grid, if a condition was met such as two neighboring charges being equal, the value of our “charge” variable was give a +1. In this way we were able to find the total charge of an Ising model for a given grid dimension.

2.2.1 Pseudo Code for Ising Model

```
+X LOOP (iterates through $x$-coordinates)
++Y LOOP (iterates through $y$-coordinates)
+++Charge=0
+++Condition 1 (top left corner):
++++If charge to right is = to current charge:
+++++Charge+1
++++If charge below is = to current charge:
+++++Charge+1
+++Condition 2 (top right corner)
++++If charge below is = to current charge:
+++++Charge+1
+++Condition 3 (bottom left corner):
++++If charge to right is = to current charge:
+++++Charge+1
+++Condition 4 (bottom right corner):
++++no output since no charge to right or bottom +++Condition 5 (left side of grid, excluding top left and bottom left charge):
++++If charge to right is = to current charge:
+++++Charge+1
++++If charge below is = to current charge:
+++++Charge+1
+++Condition 6 (right side of grid excluding top right and bottom left charge):
++++If charge below is = to current charge:
```

```
+++++Charge+1
+++Condition 7 (top row excluding top right corner and top left corner):
++++If charge to right is = to current charge:
+++++Charge+1
++++If charge below is = to current charge:
+++++Charge+1
+++Condition 8 (bottom row excluding bottom left corner and bottom right corner):
++++If charge to right is = to current charge:
+++++Charge+1
+++Condition 9 (middle charges):
++++If charge to right is = to current charge:
+++++Charge+1
++++If charge below is = to current charge:
+++++Charge+1
Output: Charge = Charge of Overall grid.
```

2.3 Bootstrap Simulations

We will explain bootstrapping within the context of our own simulations. We started with two data sets. Data set 1 contained the values for 1,000 centers of mass with $m = 1$, and $M = 2$, and Data set 2 contained 10,000 centers of mass with $m = 1$ and $M = 2$. For both sets, each data point was generated by a distinct random configuration of masses. The purpose of using bootstrapping was to approximate the sample distribution of the center of mass. The distribution of the bootstrap mean was a much more narrow distribution, as compared the the standard mean distribution, which in turn ensured a greater level of certainty. In other words, we wanted to estimate, as accurately as possible, the mean center of mass value of each set. To do this we needed to start with a large enough sample of centers of mass of random configurations, in our case Data set 1, and Data set 2. In the case of Data set 1, we took a random sample of

1000 data points, allowing for repeats within the sample. Essentially, we randomly selected 1000 centers of mass from Data set 1, with the potential that we take the *ith* element in Data set 1 multiple times. We repeated this process of random sampling 10^4 times. We then calculated the mean for each of these 10^4 resamples and graphed said mean. This exact process was repeated for Data set 2, but with each resample set containing 10,000, potentially repeated, centers of mass. In this way we were able to gain a more accurate estimate for the expected value for our center of mass.

3
Data Analysis and Results

3.1 Introduction

After running our code numerous times we successfully generated a large amount of data points containing the values for the center of mass for varying configurations. With all this data we decided that it would be important to do some statistical analysis in order to gain some insight into our outputs. The programming language R and the application RStudio contained extremely useful tools that helped us generate QQ-Norm Plots, histograms, and allowed us to perform Bootstrap analysis on our data sets. Due to the symmetry between the x, y, z-coordinates we decided to look exclusively at the output values for our x-coordinate system's center of mass.

3.1.1 Basics of QQ-Norm Plots

Before we begin in is important to understand the basics of a QQ-Norm Plot, (i.e. Quantile Quantile Plot). A QQ-Norm Plot takes a set of data and breaks it up into quantiles. It then compares the quantiles of the data against the quantiles of the normal distribution. In the case of Figure 3.1.1, on the x-axis are the quantiles for some given set of data, and on the y-axis are the center of mass quantities. Each circle on the graph represents a specific center of mass. The diagonal line straight through represents the normal bell shaped distribution curve. The degree to which the data points fall along said straight line indicates how well our data follows a smooth

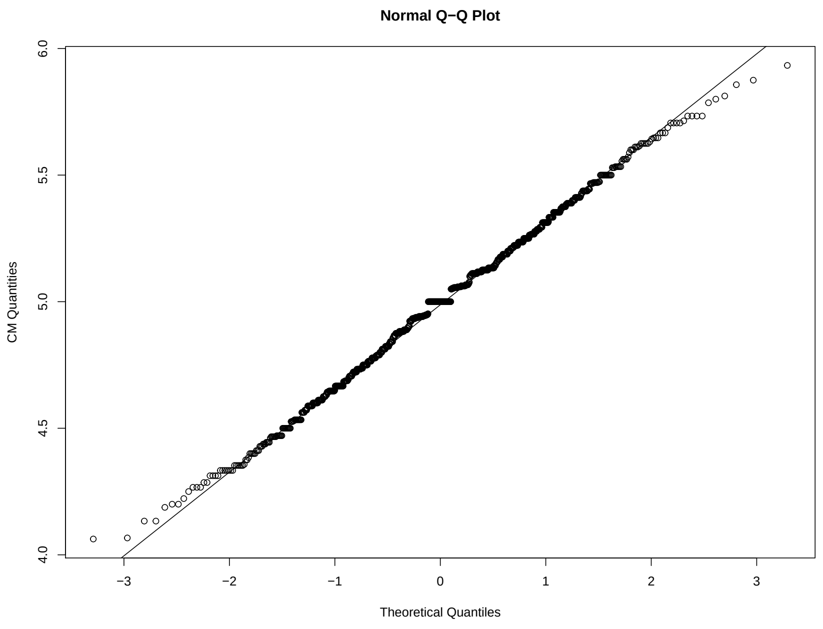

Figure 3.1.1. Basic QQ Plot

distribution. Those points that do not fall along the straight line are ones that fall outside the normal bell shaped curve of distribution. In Figure 3.1.1, the data follows the distribution curve pretty well, with only minor amounts of outliers.

3.1.2 Variables

The independent variables used to manipulate our data output were as follows:

1. Value for m
2. Value for M
3. Probability (between 0 and 1)
4. Dimension of Space (1D, 2D, 3D)
5. Iterations (How many times we ran the code)

Our only dependent variable was the value for the center of mass.

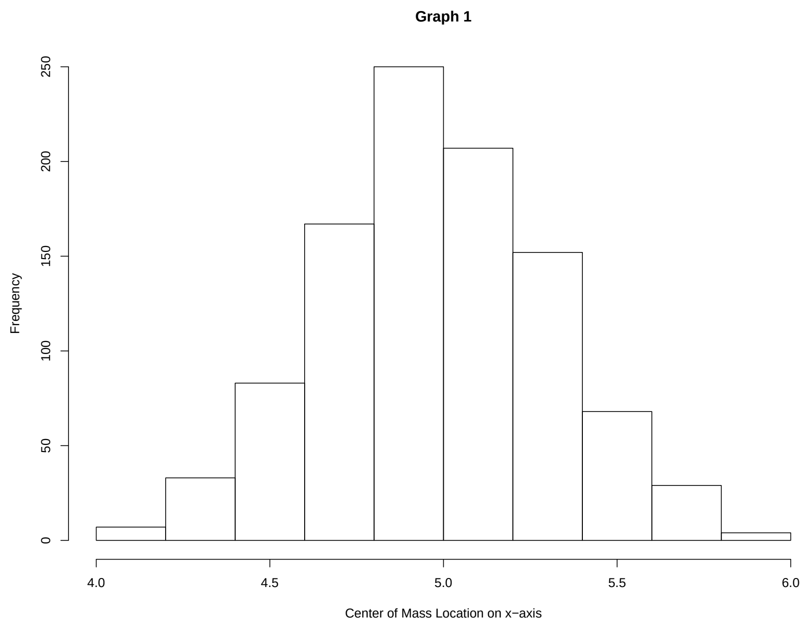

Figure 3.1.2. $m = 1$, $M = 2$, Probability $= \frac{1}{2}$, Dimension $= 10 \times 10$, Iteration $= 1{,}000$

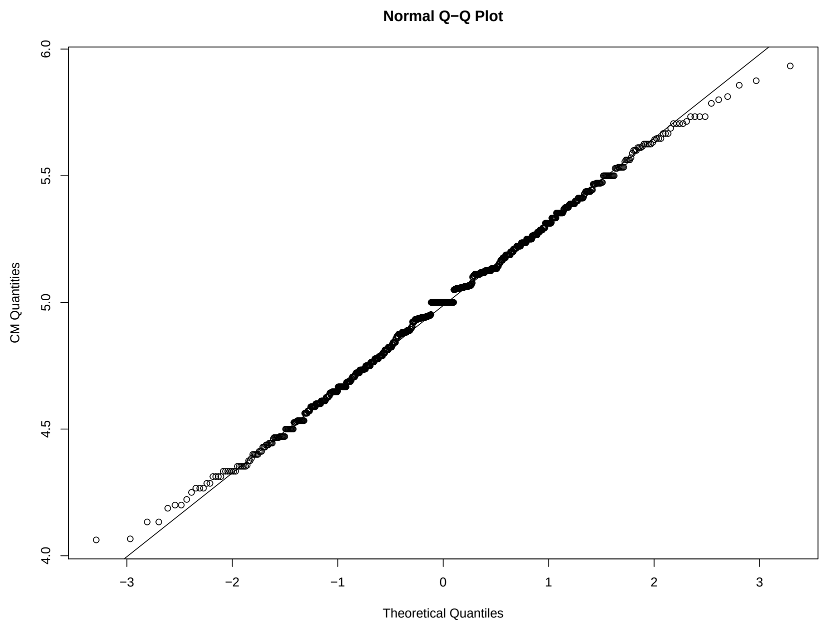

Figure 3.1.3. $m = 1$, $M = 2$, Probability $= \frac{1}{2}$, Dimension $= 10 \times 10$, Iteration $= 1{,}000$

3.2 Getting a Sense of Our Data

We first looked at histograms and QQ-Norm Plots of our data sets in order to get a sense of the general distribution curve for varying configuration of our independent variables. The histogram in Figure 3.1.2 shows the distribution for 1000 iterations, on a 10×10 grid, with probability $\frac{1}{2}$, and mass configurations $m = 1$ and $M = 2$. Here the x-axis represents the x-coordinate value for the center of mass, and the y-axis represents the amount of times that that output occurred. We immediately noticed that our center of mass tended to be centered around $x = 5$, or in other words tended to be located near the center of the grid. We also noticed that despite the fact that our x-coordinates ranged from $x = 0$ to $x = 10$, and we ran the code 1,000 times, our data points lay exclusively between coordinate values $x = 4$ and $x = 6$.

We gathered from Figure 3.1.3 that the distribution of our first data set was quite smooth and followed the curve nicely. On either end of the plot we see there was an increase in outlying center of mass values, which was most likely due to the fact that data points very far from the middle quantile tend to be less frequent and thus less likely to follow the smooth distribution. Further statistical information from Figure 3.1.3:

- Mean = 4.991
- Standard Deviation = 0.3243878

Our relatively low standard deviation told use the distribution of data was also relatively low, hence the fact that the data points lay exclusively between coordinate values $x = 4$ and $x = 6$.

3.3 Adjusting Variables

In order to further investigate the tendency for our center of mass data to be centered around $x = 5$, and the tendency for our overall interval to be quite narrow, we decided to experiment with adjusting our independent variables to determine which had significant affects on our data.

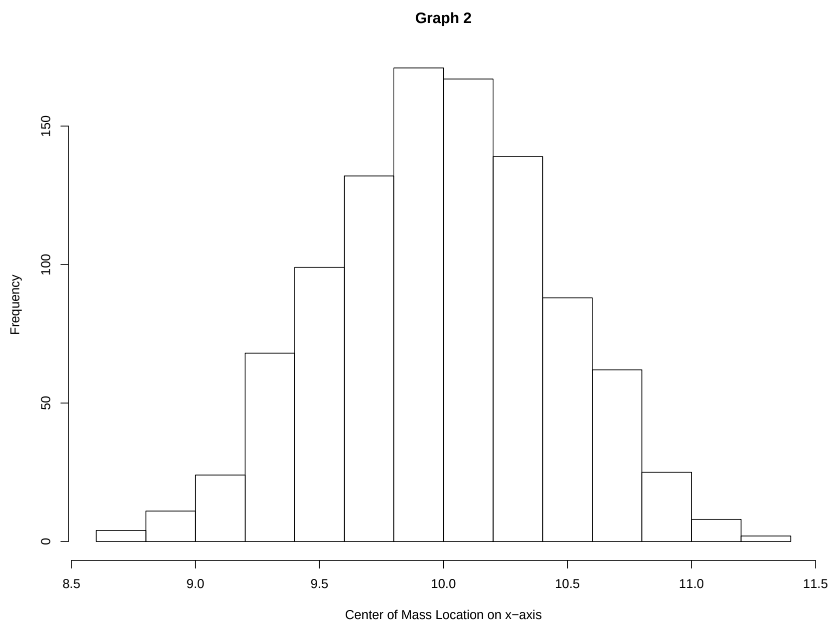

Figure 3.3.1. $m = 1$, $M = 2$, Probability $= \frac{1}{2}$, Dimension $= 20 \times 20$, Iteration $= 1{,}000$

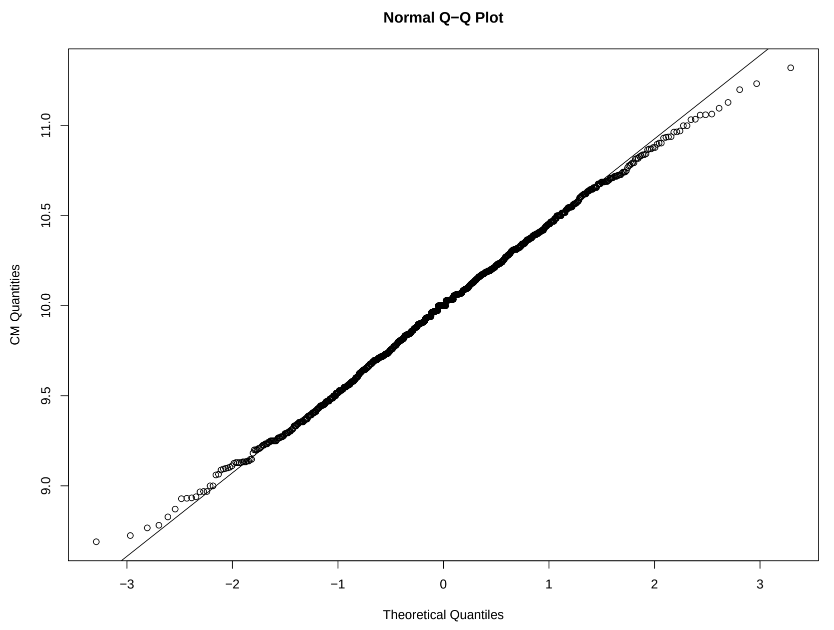

Figure 3.3.2. $m = 1$, $M = 2$, Probability $= \frac{1}{2}$, Dimension $= 20 \times 20$, Iteration $= 1{,}000$

3.3.1 Dimension

We first adjusted the grid size. Figure 3.3.1 is a graph of an identical configuration as in Figure 3.1.2, where $m = 1$, $M = 2$, Probability $= \frac{1}{2}$, Iterations = 1,000, but with a grid size of 20×20. Once again we noticed that the greatest frequency for our center of mass outputs was towards the middle of the grid, or in other words $x = 10$. With our grid size doubled, there was a distribution range of ± 1.5 between the center point, with no data points less that 8.5 or greater than 11.5. Although this is slightly larger than our distribution range of 1 for a 10×10 grid, it is actually proportionally a smaller range as compared to a smaller dimension. There was a similar trend in the data distribution range for dimensions ranging from 10×10 to 1000×1000. Despite the fact that the interval between the middle point of the grid increased as we increased dimension, (maximum at 1000×1000 was ± 7.5), the ratio of interval to dimension decreased. Ultimately we concluded that regardless of dimension, the center of mass tended to be most frequent around the middle of the grid, and always lay within a small margin of said mid point value.

From Figure 3.3.2 we once again saw that our data followed a smooth distribution curve, with only minor amounts of outliers. Further statistical information from Figure 3.3.2:

- Mean = 9.993

- Standard Deviation = 0.4552671

For our dimension manipulated data set the standard deviation was slightly larger, but only by a factor of 0.1. As expected the mean of our data (9.993) remained near the center of the grid.

3.3.2 Iterations

We then adjusted the amount of iterations, or in other words, the amount of center of mass data points. Figure 3.3.3 represents 10,000 center of mass data points, with variable configurations of $m = 1$, $M = 2$, Dimension $= 10 \times 10$, Probability $= \frac{1}{2}$, and Iteration = 10,000. We observed an extremely similar distribution and range of outputs as with the data in Figure 3.1.2, the only

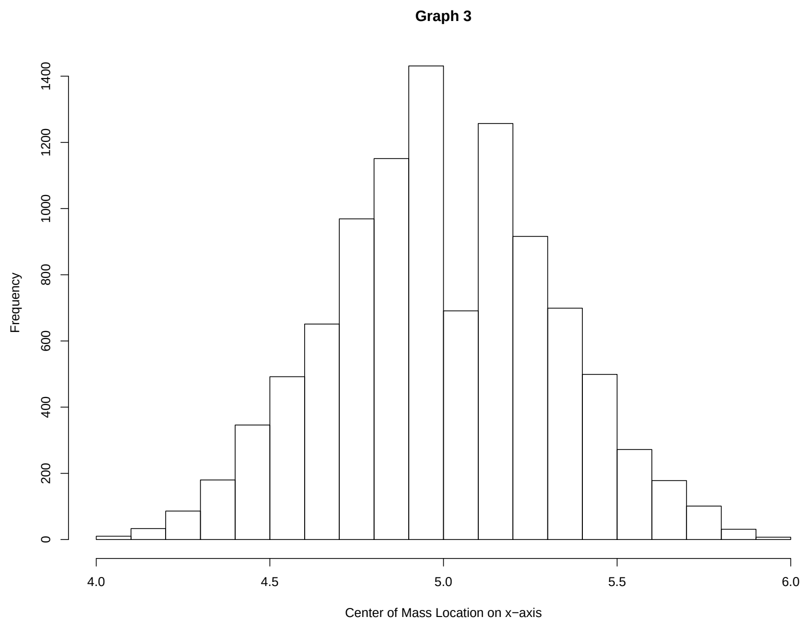

Figure 3.3.3. $m = 1$, $M = 2$, Probability $= \frac{1}{2}$, Dimension $= 10 \times 10$, Iteration $= 10{,}000$

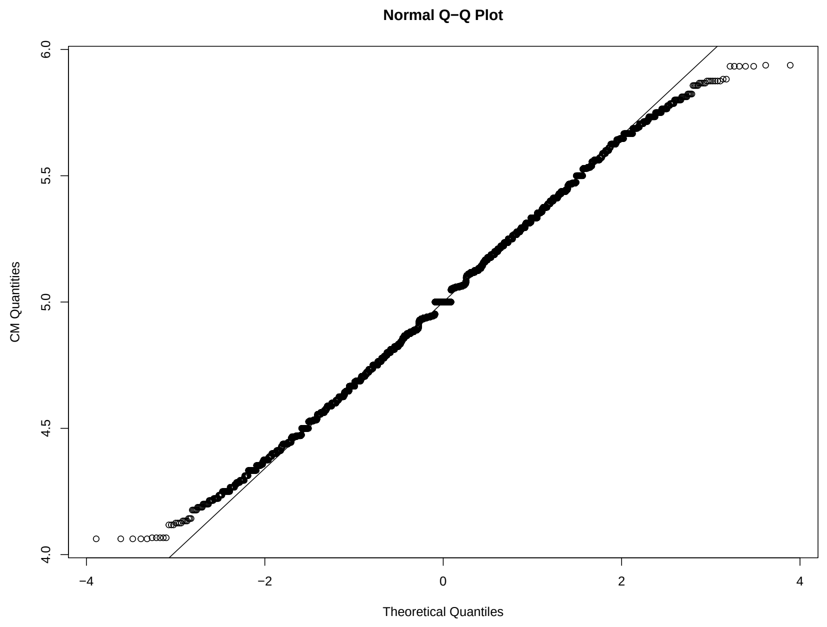

Figure 3.3.4. $m = 1$, $M = 2$, Probability $= \frac{1}{2}$, Dimension $= 10 \times 10$, Iteration $= 10{,}000$

difference being the proportional increase in frequency for each x-coordinate location. Thus we concluded that the Iterations had little to no affect on our output.

We see that in Figure 3.3.4 there is a greater density of points as compared to previous QQ-Plots, which was the direct result of there being 10,000 data points as compared to 1,000. The data followed the distribution curve slightly less perfectly, which was made clear by its divergence towards the ends. This phenomena of the least and greatest quartiles straying slightly from the distribution curve also occurred to a lesser degree from 1,000 iteration data sets. The increase in divergence in Figure 3.3.4 was simply the result of the increased data points making this divergence slightly more dramatic. Further statistical information from Figure 3.3.4:

- Mean = 5.002
- Standard Deviation = 0.3214834

As expected, the increase in iteration did not affect the mean, compared to smaller data sets, and also had little affect on the standard deviation, which was almost identical to our first data set, present in Figure 3.1.2.

3.3.3 Mass

Third, we adjusted Mass using varying combinations of m and M. We found that the difference between masses did not affect the fact that the highest frequency of center of masses tended to be centered around the middle of the grid. However, the ratio of one mass to the other did have a significant impact on the range of x-coordinate frequencies that occurred. We found that the larger the difference between m and M, the wider the range of x-coordinate outputs. For clarification observe Figure 3.3.5 and Figure 3.3.7. Both represent 1,000 centers of mass with variable configurations of probability $\frac{1}{2}$ and grid dimension 10×10. The difference between the two is that Figure 3.3.5 represents a mass configurations of $m = 5$ and $M = 10$, whereas Figure 3.3.7 represents a mass configuration of $m = 1$ and $M = 100$. Again we observe the greatest frequency of outputs to be centered around 5. However, the ratio in Figure 3.3.5 of $\frac{m}{M} = \frac{5}{10} = \frac{1}{2}$ resulted in a smaller range of center of mass outputs, only between $x = 4$ and $x = 6$. Compare

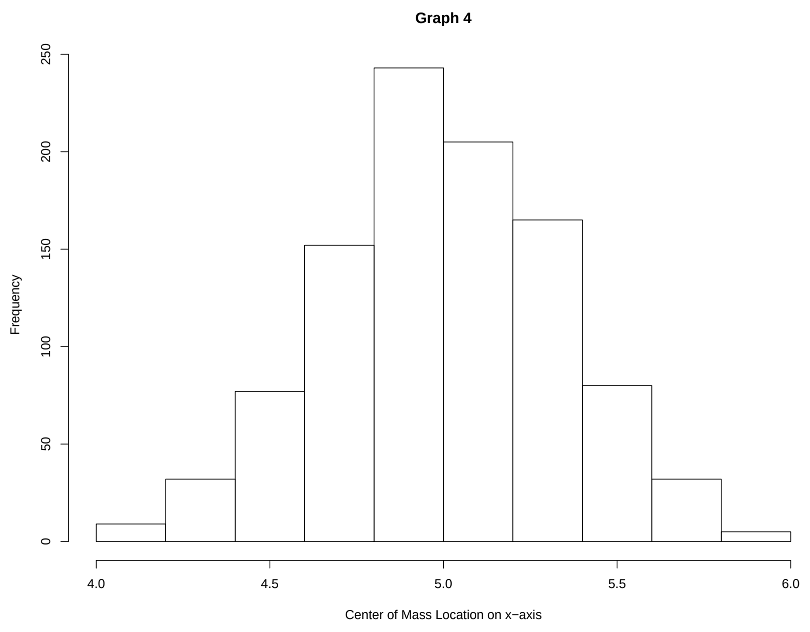

Figure 3.3.5. $m = 5$, $M = 10$, Probability $\frac{1}{2}$, Dimension 10, Iteration = 1,000

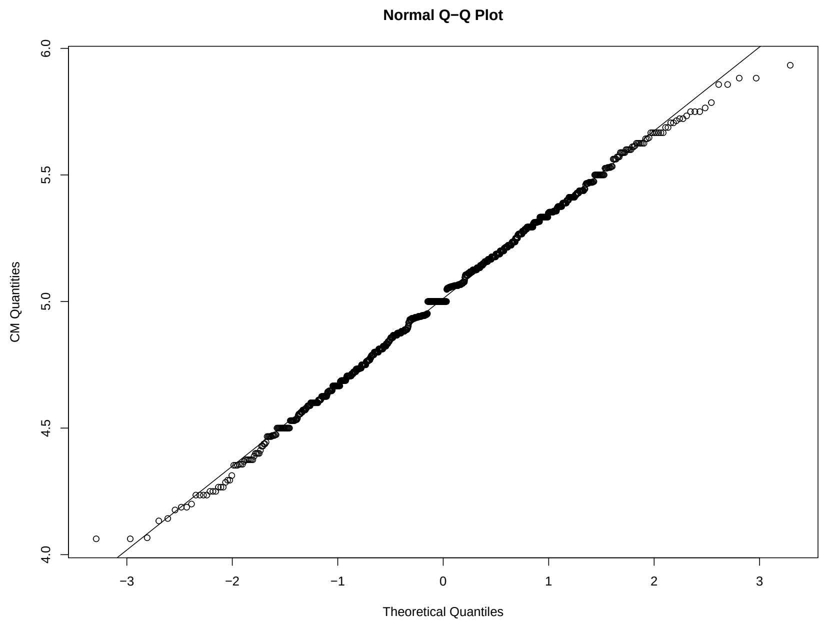

Figure 3.3.6. $m = 5$, $M = 10$, Probability $\frac{1}{2}$, Dimension 10, Iteration = 1,000

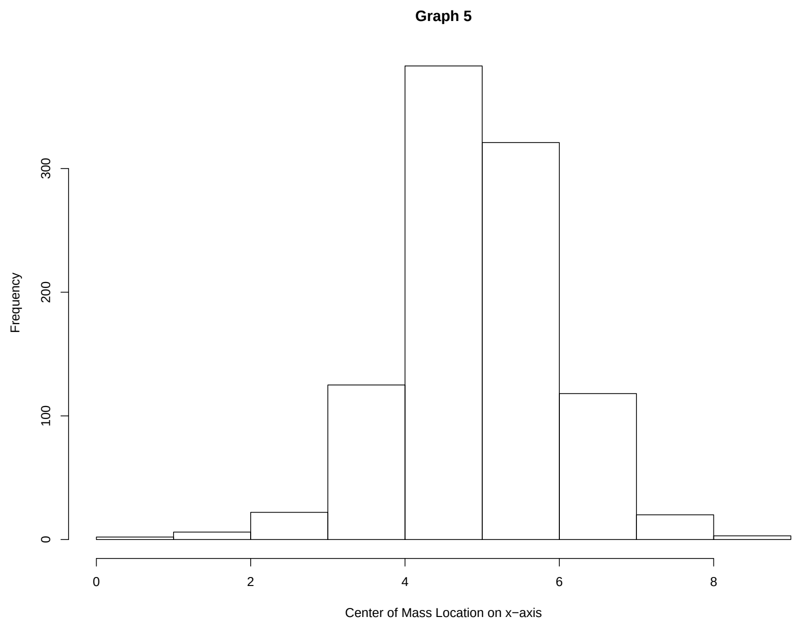

Figure 3.3.7. m=1, M=100, Probability $\frac{1}{2}$, Dimension 10, Iteration = 1,000

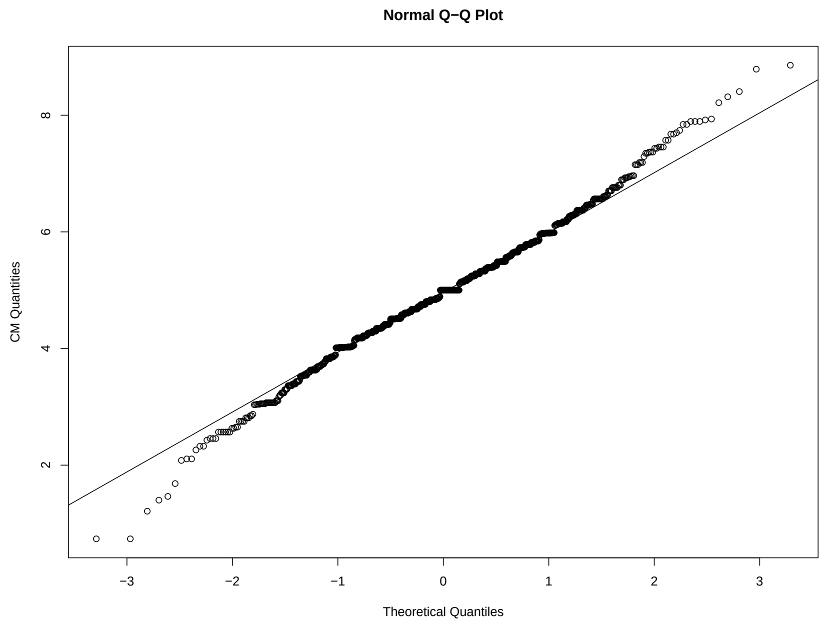

Figure 3.3.8. m = 1, M = 100, Probability $\frac{1}{2}$, Dimension 10, Iteration = 1,000

this to the results in Figure 3.3.7 where the much smaller ratio of $\frac{m}{M} = \frac{1}{100}$, resulted in a wider range of center of mass outputs, between $x = 0$ and $x = 9$. This was most likely due to the fact that the significant difference between m and M, meant that there was a larger likelihood of there being a configuration with enough large masses within an extreme region of ± 3.5 the middle of the grid, that the overall center of mass was off-center.

We see from Figure 3.3.6 that the distribution of data for our $m = 5$ and $M = 10$ configuration was once again very close to the normal distribution. Further statistical information from Figure 3.3.6:

- Mean = 5.012
- Standard Deviation = 0.3330984

We see, similar to previous data sets, that the mean was very close to 5. The standard deviation of 0.3330984 was similar to previous sets as well.

For Figure 3.3.8, the data for our configuration of $m = 1$ and $M = 100$, while still relatively close, did not fit the normal distribution nearly as well as the $m = 5$ and $M = 10$ configuration. This was most likely due to the large disparity in mass values. Further statistical information from Figure 3.3.8:

- Mean = 4.9580
- Standard Deviation = 1.121563

Although the mean for configuration $m = 1$ and $M = 100$, still remained close to 5, the large difference between mass values led to a significant increase in the standard deviation, which we see is 1.121563, as compared to the usual standard deviation of only around 0.3.

3.3.4 Probability

Last, we adjusted the probability to see how it would affect the distribution of our centers of mass. Yet again, the center of mass x-coordinate frequency was greatest around 5. However, we found that the further from a probability of $\frac{1}{2}$ we went, the smaller the distribution interval

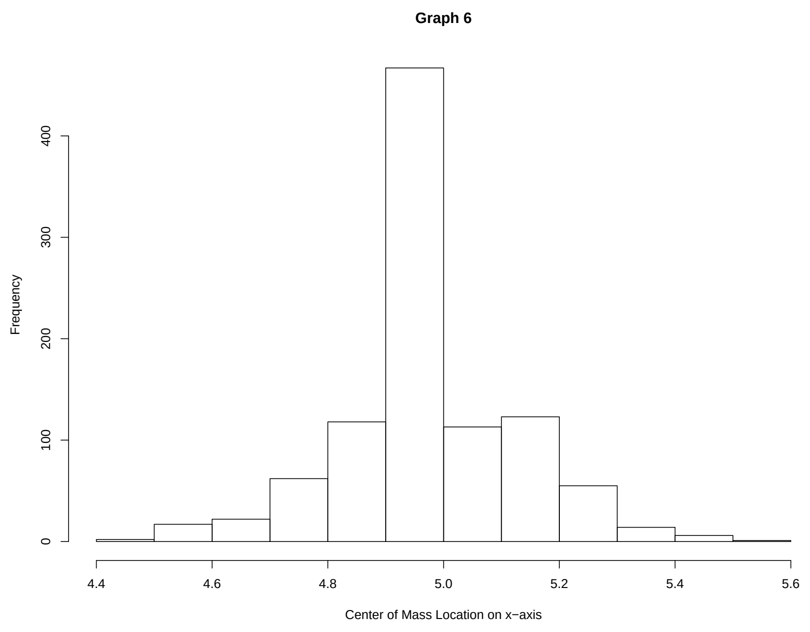

Figure 3.3.9. $m = 1$, $M = 2$, Probability $\frac{1}{10}$, Dimension 10, Iteration = 1,000

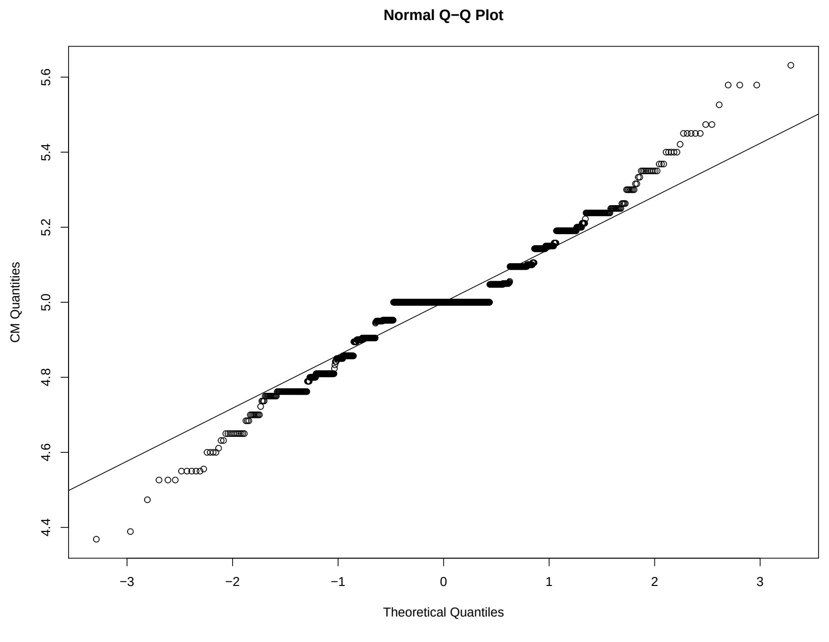

Figure 3.3.10. $m = 1$, $M = 2$, Probability $\frac{1}{10}$, Dimension 10, Iteration = 1,000

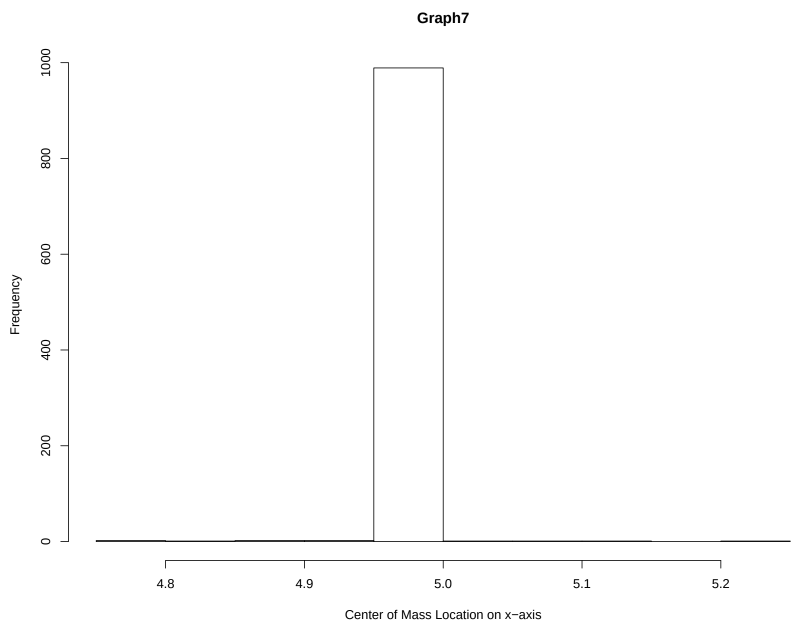

Figure 3.3.11. $m = 5$, $M = 10$, Probability $\frac{1}{1000}$, Dimension 10, Iteration = 1,000

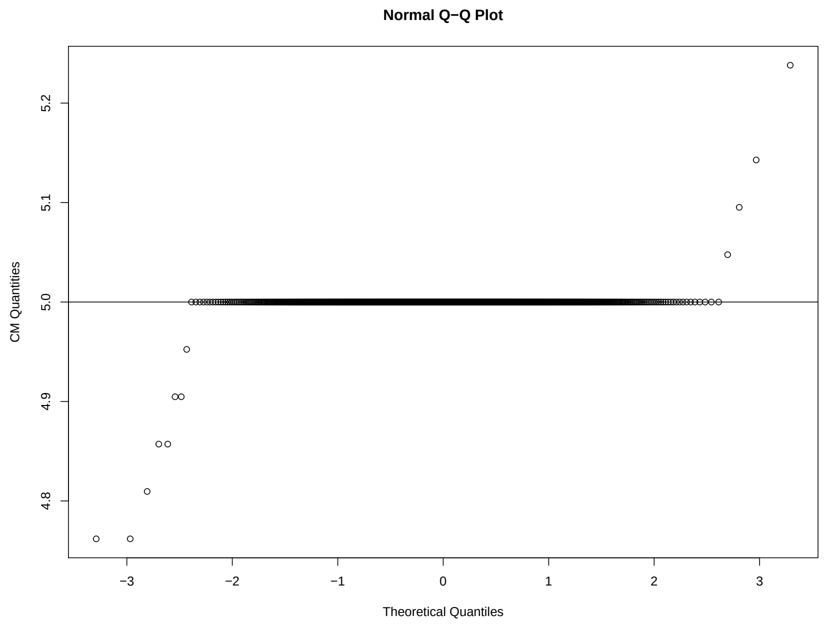

Figure 3.3.12. $m = 1$, $M = 2$, Probability $\frac{1}{1000}$, Dimension 10, Iteration = 1,000

of the data became. Figure 3.3.9 represents 1,000 centers of mass, with variable configuration $m = 1$, and $M = 2$, on a 10×10 grid, with probability $\frac{1}{10}$. Figure 3.3.11 represents 1,000 centers of mass, with the identical variable configuration but with probability $\frac{1}{1000}$. The extremely high frequency of over 400 for values around $x = 5$ in the case of Figure 3.3.9, and the even higher frequency of nearly 1000 for $x = 5$ in Figure 3.3.11, clearly demonstrates that the smaller the fraction, the smaller the distribution of centers of mass.

Figure 3.3.10 exhibits the significant affect probability had on the distribution of our center of mass data points. Although the data still followed a somewhat linear behavior, a significant amount of data points did not precisely follow the normal distribution curve, hence they did not lie directly on the diagonal line. Further statistical information on Figure 3.3.10:

- Mean = 5.001
- Standard Deviation = 0.1643993

Once again, the mean was near 5. The standard deviation however, was significantly smaller than in previous configurations. Only 0.1643993, as compared to the usual 0.3. This makes sense because as we can see from Figure 3.3.9, there was a significantly higher frequency of $CM = 5$, compared to all other frequencies. As such, a majority of the data lay near $x = 5$, and so the standard deviation was significantly smaller than usual.

Figure 3.3.12 is an even more dramatic example of how probability affected distribution. The extreme probability of $\frac{1}{1000}$ resulted in almost all of the data points to be exactly 5. The data was technically normal to the bell curve, since the curve was essentially just a straight vertical line. Further statistical information of Figure 3.3.12:

- Mean = 4.999
- Standard Deviation = 0.01729666

Once again, no surprise that the mean is 5. Our standard deviation in the case of Figure 3.3.12 was extremely low (0.01729666), for the same reason that the standard deviation of the data in

Figure 3.3.12, was low. In this case, essentially every data point was $x = 5$, and so the deviation from 5 was almost 0.

3.3.5 Variable Manipulation to Ensure the Smoothest Distribution

We felt that it may be interesting to find the variable configuration, putting the realistic aspects of the physical world aside, that would give us the most smooth distribution of center of masses across all x-coordinate values. After much experimentation we found the threshold necessary for a smooth curve with some occurrence for every x-coordinate value. Those threshold conditions were as follows:

1. m and M must be opposite in value, (i.e. If m = a, M = - a)
2. Masses must be within a 0.3 difference of one another (i.e. If Mass 1 = 1, then Mass 2 can only be -1.3 or -0.7)
3. Probability must be within a 0.4 range of $\frac{1}{2}$ (i.e. Probability can be at most $\frac{5.4}{10}$ and at least $\frac{4.6}{10}$)
4. Dimensions have no affect on distribution.
5. Iterations have no affect on distribution.

Note: With regard to condition 1, we understand that it does not physically make sense to describe a negative mass, however these conditions were sought out exclusively to find the smoothest distribution.

3.4 Bootstrapping

Based on the work discussed in the previous section we suspected that regardless of the variable configuration, the center of mass was always most frequent towards the middle of the grid. In order to more thoroughly confirm these findings, we decided to use bootstrapping to gain confirm our suspicions and gain a more certain estimate of the expected center of mass value. We used two data sets collected from grids of dimension 10×10 and 20×20. Each set, lets call them

Data10, and Data20 respectively, contained 10,000 centers of mass. All other variables of Data10 and Data20 were identical where the masses were $m = 1$, and $M = 2$, the probability was $\frac{1}{2}$ and the iterations were 10,000.

We first bootstrapped the mean for Data10 using 10^4 resamples. Figure 3.4.1 contains a histogram of the bootstrap means for these 10^4 resamples.

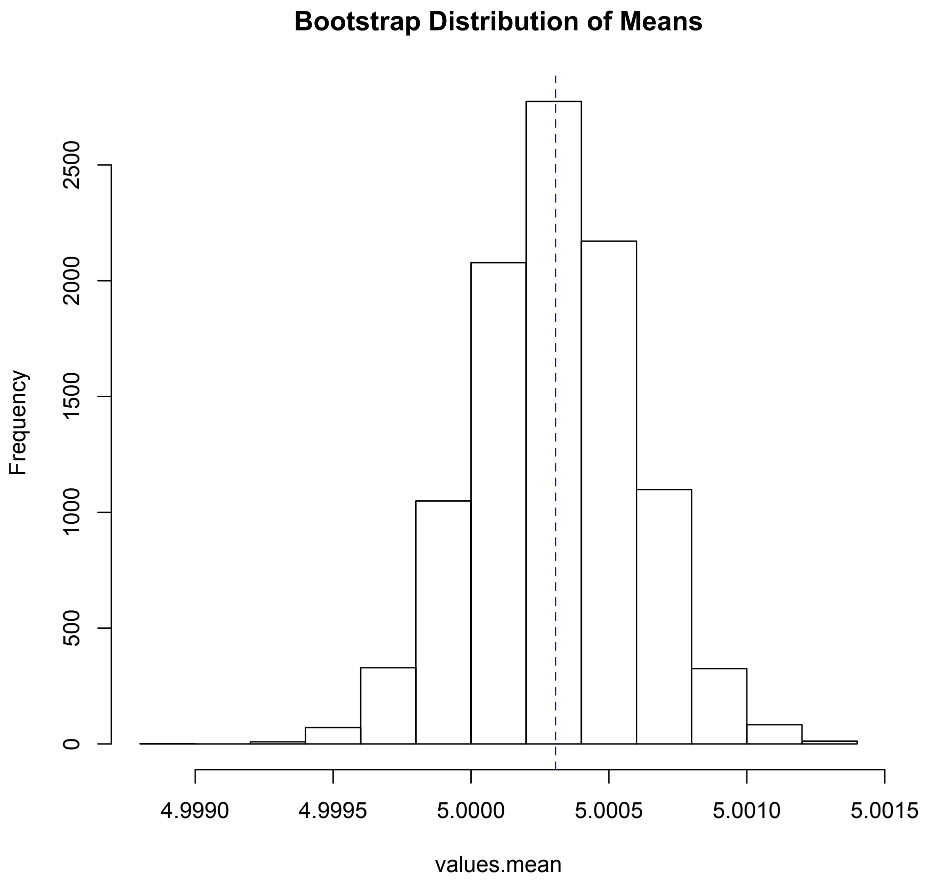

Figure 3.4.1. $m = 1$, $M = 2$, Probability $\frac{1}{2}$, Dimension 10, Iteration = 10,000

When analyzing the results of Figure 3.4.1, we found the following information:

- Bootstrap Mean = 5.000305
- Bias = -1.880041×10^{-6}

As expected, the mean of the data set corresponded with the x-coordinate value for the middle of the 10×10 grid. Here we see that the mean of 5.000305 is extremely close to the grid mid point value of $x = 5$. The bias listed above is the degree to which the bootstrap mean differed from the standard mean of Data1. As we can see the bias was extremely small, however the bootstrap mean is still a more accurate estimate of the expected value for the center of mass.

We then bootstrapped the mean for Data20, again using 10^4 resamples. Figure 3.4.2 is the histogram of our results.

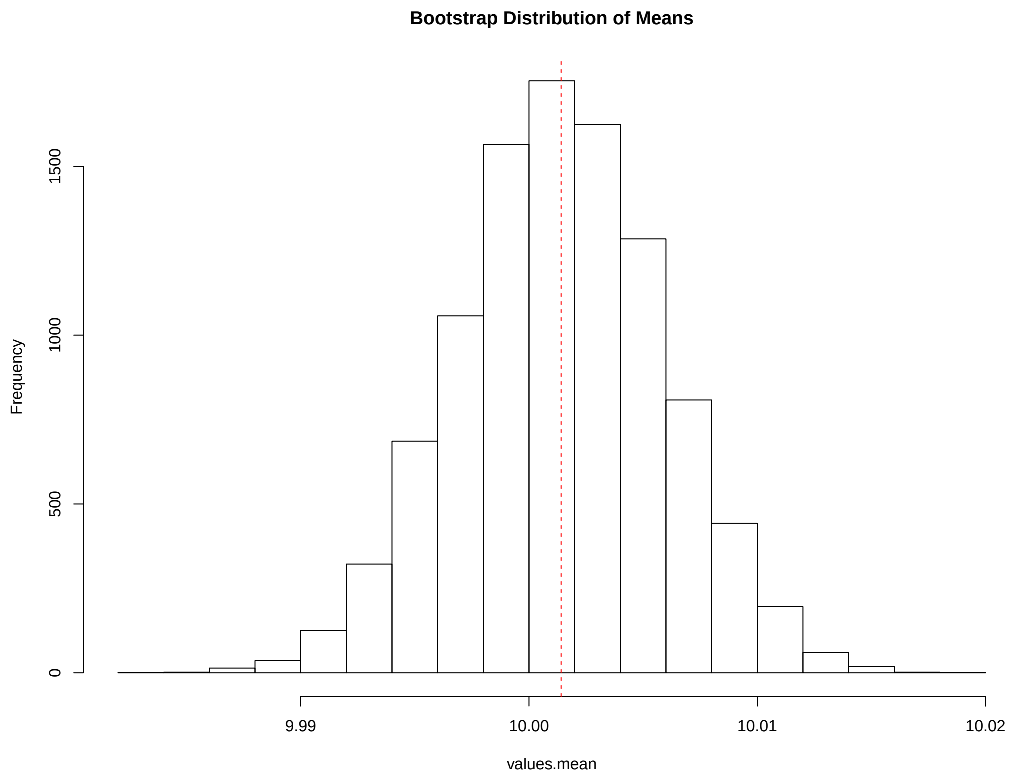

Figure 3.4.2. $m = 1$, $M = 2$, Probability $\frac{1}{2}$, Dimension 20, Iteration = 10,000

When analyzing the results of Figure 3.4.2, we found the following information.

- Bootstrap Mean = 10.00138
- Bias = -2.768852×10^{-5}

Despite the fact that the dimension was doubled, our results from the bootstrap of Data20 further confirmed our previous idea that the center of mass was always in the center of the grid. We saw that our Bootstrap mean of 10.00138 was extremely close to the grid mid point of $x = 10$. Once again our bias was also extremely small, implying that the mean of Data20 and the bootstrap mean of Data20 were very similar, however we felt it couldn't hurt to be as accurate as possible.

By bootstrapping these two data sets we were able to confirm our hypothesis that the expected value for the center of mass of a grid tended to be in the center. The results of the bootstrap analysis confirmed our initial findings, and thus, helped us formulate the theoretical explanation that the expected value for the center of mass was always $\frac{n}{2}$.

4
Theory

We will prove that the center of mass for a 1 dimensional line in both a symmetric and asymmetric configuration is always $\frac{n}{2}$ where n is the dimension of line. We define a symmetric configurations as,

Definition 4.0.1. A symmetric configuration of σ is one where it is the case that for all $X_n = X_{n-i}$ for all $n \in \mathbb{N}$

Similiarly we define an asymmetric configurations as,

Definition 4.0.2. An asymmetric configuration of σ is any configuration in which the center of mass is not exactly at the point $\frac{n}{2}$.

Let $d \geq 1$ and let L^d be the grid $[n] \times [n] \times [n] \times ... \times [n] = [n]^d$ where $[n] = \{0, ..., n\}$. We also have that τ is a configuration on L^d and is a function $\tau : L^d \longrightarrow \{m, M\}$. The center for each component of mass of each configuration L^d is defined as

$$\sum (i - CM) X_i = 0$$

where X_i represents the random variable for the type of mass placed on the ith spot.

Define a random configuration σ of L^d with two types of mass m, M and probability of m being p and probability of M being $1-p$. Let m, M be real numbers such that not both of them are zero. Let $p \in (0,1)$. We define a random configuration $\sigma^d(m, M; p)$ of L^d to be a random function such that

$$\sigma^d : L^d \longrightarrow \{m, M\}$$

$$\sigma^d\left(X_1, ..., X_{(n+1)^d}\right) = \begin{cases} m, & \text{with probability } p, \\ M, & \text{with probability } 1-p, \end{cases}$$

for all $X_i \in [n]$ for $i \in \{1, ..., d\}$.

We denote the set of all possible configurations of L^d by $\mathbb{S}^d$.

Lemma 4.0.3. *Let CM be the random variable that represents the center of mass for a random configuration σ' of $L' = [n]$. Then $\mathbb{E}[CM] = \frac{n}{2}$.*

So this means

$$X_i = \sigma'(i) = \begin{cases} m, & \text{with probability} p, \\ M, & \text{with probability} 1-p, \end{cases}$$

or in other words that we are saying

$$\sum_{i=0}^{n} \left(i - CM\left(\sigma'\right)\right) X_i = 0,$$

which means

$$\tau : L' \longrightarrow \{m, M\}$$

$$\sum_{i=0}^{n} \left(i - CM(\tau)\right) \tau(i) = 0$$

For any configuration σ we have two distinct cases.

Proof. We will prove this lemma by considering two difference cases.

- **Case 1: Symmetric Configurations:**

  We know the center of mass of a configuration σ is the number $CM(\sigma)$ that satisfies the expression

$$\sum_{i=0}^{n} \left(i - CM(\sigma)\right) \sigma(i) = 0.$$

  We claim that $CM(\sigma) = \frac{n}{2}$. In order to prove this claim, we consider two subcases:

1. When n is even we see,

$$\sum_{i=0}^{n}\left(i-\frac{n}{2}\right)X_i=\sum_{i=0}^{\frac{n}{2}-1}\left(i-\frac{n}{2}\right)X_i+\left(\frac{n}{2}-\frac{n}{2}\right)X_{\frac{n}{2}}+\sum_{\frac{n}{2}+1}^{n}\left(i-\frac{n}{2}\right)X_i$$

   Since $\sigma(i)=X_i$ and σ is symmetric we see that $X_i=X_{n-i}$ so we can reduce the previous expression and change variables,

$$\sum_{i=0}^{\frac{n}{2}-1}\left(i-\frac{n}{2}\right)X_i+\sum_{\frac{n}{2}+1}^{n}\left(i-\frac{n}{2}\right)X_i=$$

$$\sum_{i=0}^{\frac{n}{2}-1}\left(i-\frac{n}{2}\right)X_i+\sum_{i=\frac{n}{2}+1-\left(\frac{n}{2}+1\right)}^{n-\left(\frac{n}{2}+1\right)}\left(n-i-\frac{n}{2}\right)X_{n-i}=$$

$$\sum_{i=0}^{\frac{n}{2}-1}\left(i-\frac{n}{2}\right)X_i+\sum_{i=0}^{\frac{n}{2}-1}\left(\frac{n}{2}-i\right)X_{n-i}=0$$

   This shows that $CM(\sigma)=\frac{n}{2}$.

2. When n is odd we have a similar case, the major difference being we are not subtracting a center value of $\frac{n}{2}$ from itself. Again using the fact that symmetry means $X_i=X_{n-i}$, we see

$$\sum_{i=0}^{n}\left(i-\frac{n}{2}\right)X_i=\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor}\left(i-\frac{n}{2}\right)X_i+\sum_{\lceil\frac{n}{2}\rceil}^{n}\left(i-\frac{n}{2}\right)X_i=$$

$$\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor}\left(i-\frac{n}{2}\right)X_i+\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor}\left(n-i-\frac{n}{2}\right)X_{n-i}=\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor}\left(i-\frac{n}{2}\right)X_i+\sum_{i=0}^{\lfloor\frac{n}{2}\rfloor}\left(\frac{n}{2}-i\right)X_{n-i}=0,$$

   which again shows that $CM(\sigma)=\frac{n}{2}$.

- **Case 2: Asymmetric Configurations:**

  We see that for any asymmetric configuration σ there exists a mirrored configuration σ^* such that:

$$\sigma^*=\sigma(n-i)$$

  We claim that $CM(\sigma)+CM(\sigma^*)=n$.

Assume $CM(\sigma) = \frac{n}{2} + c$ for some integer c. So showing that $CM(\sigma^*) = \frac{n}{2} - C$:

$$\sum_{i=0}^{n} \left(i - \left(\frac{n}{2} - c\right)\right) y_i = \sum_{i=0}^{n} \left(i - \frac{n}{2} + c\right) X_{n-i}$$

Using the fact that $\sigma^* = \sigma(n-i)$

$$\sum_{i=0}^{n} \left(i - \frac{n}{2} + c\right) X_{n-i} = \sum_{i=0}^{n} \left(n - i - \frac{n}{2} + c\right) X_{n-i} = \sum_{i=0}^{n} \left(\frac{n}{2} + c - i\right) X_i = 0$$

So when we take the sum $CM(\sigma) + CM(\sigma^*) = \left(\frac{n}{2} + c\right) + \left(\frac{n}{2} - c\right) = n.$ □

Theorem 4.0.4. *The expected value for the center of mass of L^1 is $\frac{n}{2}$.*

Proof. Let $S^1 = S^1_S \cup S^1_A$ be the union of all symmetric and asymmetric configurations where S^1_S denotes a symmetric configuration and S^1_A denotes an asymmetric configuration. We know that the expected value for the random variable CM is

$$\mathbb{E}(CM) = \sum_{\sigma \in S^1} CM(\sigma)\,\mathbb{P}(\sigma).$$

We will first prove that $\mathbb{P}(\sigma) = \mathbb{P}(\sigma^*)$. In other words that

$$\sum_{\sigma \in S^1_A} CM(\sigma)\mathbb{P}(\sigma) = \sum_{\sigma^* \in S^1_A} CM(\sigma^*)\mathbb{P}(\sigma^*).$$

Suppose we have some function

$$f : S^1_A \longrightarrow S^1_A.$$

Since $\sigma, \sigma^* \in S^1_A$, there also exists a function

$$g : \sigma \longrightarrow \sigma^*.$$

Since S^1_A is a finite set and g is a one to one function, there exists a bijections from σ to σ^*, thus it must be the case that $\mathbb{P}(\sigma) = \mathbb{P}(\sigma^*)$, so

$$\sum_{\sigma \in S^1_A} CM(\sigma)\mathbb{P}(\sigma) = \sum_{\sigma^* \in S^1_A} CM(\sigma^*)\mathbb{P}(\sigma^*).$$

Now moving on to the rest of the proof, since $S^1 = S^1_S \cup S^1_A$, by splitting up the sum into two parts, we have

$$\sum_{\sigma \in S^1} CM(\sigma)\,\mathbb{P}(\sigma) = \sum_{\sigma \in S^1_S} CM(\sigma)\,\mathbb{P}(\sigma) + \sum_{\sigma \in S^1_A} CM(\sigma)\,\mathbb{P}(\sigma) =$$

$$\sum_{\sigma\in S_S^1}\frac{n}{2}\mathbb{P}(\sigma)+\frac{1}{2}\left(2\sum_{\sigma\in S_A^1}CM(\sigma)\mathbb{P}(\sigma)\right)=$$

$$\sum_{\sigma\in S_S^1}\frac{n}{2}\mathbb{P}(\sigma)+\frac{1}{2}\left(\sum_{\sigma\in S_A^1}CM(\sigma)\mathbb{P}(\sigma)+\sum_{\sigma\in S_A^1}CM(\sigma)\mathbb{P}(\sigma)\right)$$

$$\sum_{\sigma\in S_S^1}\frac{n}{2}\mathbb{P}(\sigma)+\frac{1}{2}\left(\sum_{\sigma\in S_A^1}CM(\sigma)\mathbb{P}(\sigma)+\sum_{\sigma^*\in S_A^1}CM(\sigma^*)\mathbb{P}(\sigma^*)\right)=$$

$$\sum_{\sigma\in S_S^1}\frac{n}{2}\mathbb{P}(\sigma)+\frac{1}{2}\left(\sum_{\sigma\in S_A^1}\left(\frac{n}{2}+c\right)\mathbb{P}(\sigma)+\sum_{\sigma^*\in S_A^1}\left(\frac{n}{2}-c\right)\mathbb{P}(\sigma^*)\right)=$$

$$\sum_{\sigma\in S_S^1}\frac{n}{2}\mathbb{P}(\sigma)+\frac{1}{2}\left(\sum_{\sigma\in S_A^1}n\mathbb{P}\right)=\sum_{\sigma\in S_S^1}\frac{n}{2}\mathbb{P}(\sigma)+\sum_{\sigma\in S_A^1}\frac{n}{2}\mathbb{P}(\sigma)=\sum_{\sigma\in S}\frac{n}{2}\mathbb{P}(\sigma)$$

Since $\frac{n}{2}$ is a constant, and the sum of the probability all possible configurations must be 1 we see that,

$$\sum_{\sigma\in S}\frac{n}{2}\mathbb{P}(\sigma)=\frac{n}{2}\sum_{\sigma\in S}\mathbb{P}(\sigma)=\frac{n}{2}*1=\frac{n}{2}$$

□

We have now proven for a 1-Dimensional case that the center of mass is always $\frac{n}{2}$, but during our simulations we found the center of mass for 2-D and 3-D, spaces. It is therefore our intent to show that the center of mass for any n-dimensional space is always $\frac{n}{2}$.

Theorem 4.0.5. *Let d be a positive integer. The expected value for the center of mass for each component of L^d is $\frac{n}{2}$.*

Proof. We will prove this by using induction. We have already shown that this statement is true for L^1 by **Theorem 4.0.4**. Since L^2 is the cartesian product of 2 copies of L^1, to find the $\mathbb{E}(CM)$ for each component of L^2, say the first component, we will write this grid as the union of $n+1$ horizontal copies of L^1; in other words,

$$L^2=\{L_0^1,L_2^1,...,L_n^1\}.$$

We know that the $\mathbb{E}(CM)$ for each $L_0^1, L_2^1, ..., L_n^1$ is $\frac{n}{2}$; thus, for the first component of L^2, we have

$$\mathbb{E}\left(CM\left(L_x^2\right)\right) = \sum_{i=0}^{n} \frac{1}{n+1}\mathbb{E}\left(CM\left(L_i^1\right)\right) = (n+1)\frac{n}{2} \cdot \frac{1}{n+1} = \frac{n}{2}.$$

Due to the symmetric nature of this grid, the exact argument can be used to show that $\mathbb{E}\left(CM\left(L_y^2\right)\right) = \frac{n}{2}$ is also true for the second coordinated of L^2.

Now consider L^d. This grid can be represented as a union of $n+1$ copies of L^{d-1}; in other words,

$$L^d = \{L_0^{d-1}, L_2^{d-1}, ..., L_n^{d-1}\}.$$

By inductive hypothesis, we know that the $\mathbb{E}(CM)$ of each of these grids is $\frac{n}{2}$, so for any coordinate of L^d we have

$$\mathbb{E}\left(CM\left(L_{x_j}^d\right)\right) = \sum_{i=0}^{n} \frac{1}{n+1}\mathbb{E}\left(CM\left(L_i^1\right)\right) = (n+1)\frac{n}{2} \cdot \frac{1}{n+1} = \frac{n}{2},$$

where j is between 1 and d. By induction, the expected value for the center of mass for each component of L^d is $\frac{n}{2}$. $\square$

5
Conclusion and Future Work

5.1 Conclusion

The overarching goal of this project was to ultimately determine what the expected value for the center of mass of a d dimensional finite integer grid was. By using Python code to generate 1D, 2D, and 3D spaces, we were able to simulate and calculate the center of mass for thousands of different mass configurations. With some valuable software in RStudio we were able to thoroughly analyze these data sets and determine how the variables of mass, dimension, probability, and iteration affected the trends in our data. In addition to experimenting with distinct variable configurations, we also used bootstrap analysis to gain an accurate estimate for our center of mass. Through these various statistical analyses we were able to determine that regardless of the specific variable assignment, the center of mass for any configuration tended to be near the center of the grid. We then ultimately proved, for both symmetric, and asymmetric configurations, that the expected value for the center of mass of any n-dimensional grid was always $\frac{n}{2}$.

5.2 Future Work

When we began this project we had ambitiously hoped to simulate grids that were more than just rectangular. Simulating center of mass on grids that are triangular or hexagonal in shape has the potential to reinforce the conclusions made in this paper, and may even lead to valuable new information about center of mass in various polygonal spaces. Most objects in the natural world are not perfect rectangles, and although we can generalize their shapes to such a space, more flexible grid shapes like hexagons, or even triangles, have the potential to create a more sophisticated representation of various objects. With this in mind, we propose using the work done in the paper as a basis for calculating the center of mass of more complex grid spaces that do not restrict themselves to symmetry, and thus can more effectively represent objects of any shape.

Furthermore, the theoretical component of this paper exclusively explored the expected value for the center of mass. A further theoretical exploration of the expected value for the variance of our data sets may grant an interesting insight into the behavior of large center of mass data sets.

Finally, although we explored the ising model within the context of simulation, we did not focus on statistically and theoretically exploring the data generated. For future work we suggest not only creating more sophisticated simulations, such as ones that include asymmetric spaces, but also to bootstrap the data generated by these simulations, and use the results of this analysis to mathematically prove potential trends in the behavior of ising data.

Appendix

5.3 Center of Mass Python Code

```
import random
import math
import matplotlib
from collections import defaultdict
from bisect import bisect_left
import numpy as np
import csv



def weight(num):
        rand = random.random()
        if rand <= num:
                return 1
        else:
                return -1


centerofmassxlist=[]
centerofmassylist=[]
centerofmasszlist=[]

def runcode(p,q):
        grid=[]

        #p, q = raw_input("Enter two numbers: ").split()
        if q == "pi":
                q = math.pi
        else:
                q = float(q)

        if p == "pi":
                p = math.pi
        else:
                p = float(p)

        num = p/q

        topx=0
        topy=0
        topz=0
        bottom=0


        for x in range(0,11):
                for y in range(0,11):
                        for z in range(0,11):
                                cluster=(x,y,z,weight(num))
                                a=cluster[0]
                                b=cluster[1]
                                c=cluster[2]
                                m=cluster[3]
                                grid.append(cluster)
                                topx=topx+(a*m)
                                topy=topy+(b*m)
                                topz=topz+(c*m)
                                bottom=bottom+m

        centerofmassx=float(topx)/float(bottom)
        centerofmassy=float(topy)/float(bottom)
        centerofmassz=float(topz)/float(bottom)

        centerofmassxlist.insert(0,centerofmassx)
        centerofmassylist.insert(0,centerofmassy)
        centerofmasszlist.insert(0,centerofmassz)


xvalues=[]
yvalues=[]
zvalues=[]


#for p in range(1,10,1):
for f in range(1000):
        runcode(5,10)
        xvalues.append(centerofmassxlist[0])
        yvalues.append(centerofmassylist[0])
        zvalues.append(centerofmasszlist[0])

#xvalues.sort()
#yvalues.sort()
#zvalues.sort()

centerofmassxlist.sort()
centerofmassylist.sort()
centerofmasszlist.sort()


#print ('X Values')
#print centerofmassxlist
'''
with open('data10.csv','w') as fp:
        a = csv.writer(fp)
        data=[['Values'],centerofmassxlist]
        a.writerows(data)
'''
#print ('Y Values')
#print centerofmassylist

#print ('Z Values')
#print centerofmasszlist




def xfrange(start, stop, step):
    i = 0
    while start + i * step < stop:
        yield start + i * step
        i += 1





rightranges=list(xfrange(0,10,0.1))
leftranges=list(xfrange(1,11,0.1))

xdata=[]
for l,r in zip(rightranges,leftranges):
        xbar=(sum(l<=value<r for value in xvalues))
        xdata.append(xbar)
```

```
ydata=[]
for l,r in zip(rightranges,leftranges):
        ybar=(sum(l<=value<r for value in yvalues))
        ydata.append(ybar)

zdata=[]
for l,r in zip(rightranges,leftranges):
        zbar=(sum(l<=value<r for value in zvalues))
        zdata.append(zbar)




import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

increments=list(xfrange(0,10,0.1))
stringincrements=[]

for num in increments:
        new=str(num)
        stringincrements.append(new)

objects = (stringincrements)
y_pos = np.arange(len(objects))
output = xdata

plt.bar(y_pos, output, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('amount')
plt.title('range')

plt.show()

objects1 = (stringincrements)
y_pos = np.arange(len(objects1))
y_pos1 = np.arange(len(objects1))
output1 = ydata

plt.bar(y_pos1, output1, align='center', alpha=0.5)
plt.xticks(y_pos1, objects1)
plt.ylabel('amount')
plt.title('range')

plt.show()
```

```
objects2 = (stringincrements)
y_pos = np.arange(len(objects2))
y_pos2 = np.arange(len(objects2))
output2 = zdata

plt.bar(y_pos2, output2, align='center', alpha=0.5)
plt.xticks(y_pos2, objects2)
plt.ylabel('amount')
plt.title('range')

plt.show()
```

5.4 Ising Model Python Code

```
import random
import math
import matplotlib
from collections import defaultdict
from bisect import bisect_left



def weight(num):
        rand = random.random()
        if rand <= num:
                return 1
        else:
                return -1


centerofmassxlist=[]
centerofmassylist=[]
centerofmasszlist=[]
grid=[]


def runcode(p,q):


        #p, q = raw_input("Enter two numbers: ").split()

        if q == "pi":
                q = math.pi
        else:
                q = float(q)

        if p == "pi":
                p = math.pi
        else:
                p = float(p)

        num = p/q

        topx=0
        topy=0
        topz=0
        bottom=0


        for x in range(-10,11):
                for y in range(-10,11):
                        cluster=(x,y,weight(num))
                        grid.append(cluster)



runcode(5,10)
#print(grid)

charge=0

for i in grid:

        #top left corner
        if i[0]==-10 and i[1]==10:
                for j in grid:
                        if j[0]==i[0]+1 and j[1]==i[1] and j[2]==i[2]:
                                charge=charge+1
                        if j[1]==i[1]-1 and j[0]==i[0] and j[2]==i[2]:
                                charge=charge+1
        #bottom left corner
        elif i[0]==-10 and i[1]==-10:
                for j in grid:
                        if j[0]==i[0]+1 and j[1]==i[1] and j[2]==i[2]:
                                charge=charge+1
                        if j[1]==i[1]+1 and j[0]==i[0] and j[2]==i[2]:
                                charge=charge+1

        #top right corner
        elif i[0]==10 and i[1]==10:
                for j in grid:
                        if j[1]==i[1]-1 and j[0]==i[0] and j[2]==i[2]:
                                charge=charge+1

        #bottom right corner
        elif i[0]==10 and i[1]==-10:
```

```
        for j in grid:
            if j[1]==i[1]+1 and j[0]==i[0] and j[2]==i[2]:
                charge=charge+1

    #top condition
    elif -9<=i[0]<=9 and i[1]==10:
        for j in grid:
            if j[0]==i[0]+1 and j[1]==i[1] and j[2]==i[2]:
                charge=charge+1
            if j[1]==i[1]-1 and j[0]==i[0] and j[2]==i[2]:
                charge=charge+1

    #right side condition
    if i[0]==10 and -9<=i[1]<=9:
        for j in grid:
            if j[0]==i[0]+1 and j[1]==i[1] and j[2]==i[2]:
                    charge=charge+1
            if j[1]==i[1]-1 and j[0]==i[0] and j[2]==i[2]:
                charge=charge+1

    #left side condition
    elif i[0]==-10 and -9<=i[1]<=9:
        for j in grid:
            if j[0]==i[0]-1 and j[1]==i[1] and j[2]==i[2]:
                charge=charge+1
            if j[1]==i[1]-1 and j[0]==i[0] and j[2]==i[2]:
                charge=charge+1

    #bottom condition
    elif -9<=i[0]<=9 and i[1]==-10:
        for j in grid:
            if j[0]==i[0]+1 and j[1]==i[1] and j[2]==i[2]:
                charge=charge+1
            if j[1]==i[1]+1 and j[0]==i[0] and j[2]==i[2]:
                charge=charge+1

    #everywhere else
    elif -8<=i[0]<=7 and -7<=i[0]<=8:
        for j in grid:
            if j[0]==i[0]+1 and j[1]==i[1] and j[2]==i[2]:
                charge=charge+1
            if j[1]==i[1]-1 and j[0]==i[0] and j[2]==i[2]:
                charge=charge+1


print(charge)
```

```
'''for x in range(10):
        if grid[x][0]==grid[][0] and grid[x][1]



if grid[n][2]==grid[n+1][2]:
                print ('plus 1')
for n in range(0,6):
        if grid[n][2]==grid[n+3][2]:
                print('plus 1 under')

'''











'''
xvalues=[]
yvalues=[]
zvalues=[]


#for p in range(1,10,1):
for f in range(1000):
        runcode(5,10)
        xvalues.append(centerofmassxlist[0])
        yvalues.append(centerofmassylist[0])
        zvalues.append(centerofmasszlist[0])




righttranges=list(range(-10,10))
leftranges=list(range(-9,11))

xdata=[]
for l,r in zip(righttranges,leftranges):
        xbar=(sum(l<=value<r for value in xvalues))
        xdata.append(xbar)
```

```
ydata=[]
for l,r in zip(rightranges,leftranges):
        ybar=(sum(l<=value<r for value in yvalues))
        ydata.append(ybar)

zdata=[]
for l,r in zip(rightranges,leftranges):
        zbar=(sum(l<=value<r for value in zvalues))
        zdata.append(zbar)
```

```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt

increments=list(range(-10,10))
stringincrements=[]

for num in increments:
        new=str(num)
        stringincrements.append(new)

objects = (stringincrements)
y_pos = np.arange(len(objects))
output = xdata

plt.bar(y_pos, output, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('amount')
plt.title('range')

plt.show()

objects1 = (stringincrements)
y_pos = np.arange(len(objects1))
y_pos1 = np.arange(len(objects1))
output1 = ydata

plt.bar(y_pos1, output1, align='center', alpha=0.5)
plt.xticks(y_pos1, objects1)
plt.ylabel('amount')
plt.title('range')

plt.show()
```

```
objects2 = (stringincrements)
y_pos = np.arange(len(objects2))
y_pos2 = np.arange(len(objects2))
output2 = zdata

plt.bar(y_pos2, output2, align='center', alpha=0.5)
plt.xticks(y_pos2, objects2)
plt.ylabel('amount')
plt.title('range')

plt.show()


 '''
```

5.5 R Studio Bootstrap Code

```
view(data20)
hist(data20)
n<-length(data20)
N<-10^4
values.mean <- numeric(N)
values.var <- numeric(N)
for (i in 1:N)
   {     x<- sample(data20, n, replace = TRUE)
         values.mean[i] <- mean(x)
         values.var[i] <- var(x)
   }
hist(values.mean, main = "Bootstrap Distribution of Means")
abline(v=mean(data20), col="red", lty=2)
hist(values.var, main = "Bootstrap Distribution of Variance")
abline(v = var(values1), col="red", lty=2)
summary(values.var)
```

Bibliography

[1] Halliday and Resnick, *Fundamentals of Physics Extended*, Wiley, Hoboken, NJ, 2014.

[2] Laura Chihara and Tim Hersterberg, *Mathematical Statistics*, Wiley, Hoboken, NJ, 2011.