Computational Cost of Brownian Motion

Zachary Schmidt
MNSU-Mankato
zachary.schmidt@mnsu.edu

December 16, 2013

Abstract

Brownian motion, or random motion in some number of dimensions, occurs frequently in the study of particle theory and fractals. It was first observed as pollen moving in a fluid in 1827 by Robert Brown and formalized mathematically in 1905 by Albert Einstein. Brownian motion has applications in biology, biophysics, cellular biology, and stellar dynamics. Numerous algorithms have been created that claim to generate Brownian motion on a computer, but the inherent computational complexities and empirical accuracy of these models have not been discussed. An accurate and cheap model could serve to solve open problems such as the narrow escape problem in biology, or used to simulate large scale Brownian motion that occurs when large stellar bodies respond to gravitational forces from nearby bodies. In this research, two algorithms used to generate Brownian motion, brown noise developed by C.W. Gardiner and midpoint displacement developed by Fournier, Fussell and Carpenter at SIGGRAPH 1982, are implemented and assessed. Programs to generate the motion in one and two dimensions have been implemented and assessed via convergence in mean and variance utilizing several different GNUplot functions. This research also addresses the suitable number of samples to create a normal standard distribution from a uniform distribution using the Central Limit Theorem, and draws a relationship between Einstein's equation and the mathematical definition of Brownian motion. The experimental results from this work confirm the relationships defined by Einstein, and shows that the midpoint displacement algorithm has a smaller mean square displacement versus brown noise.

1 Preliminaries

The concept of Brownian motion was first observed by botanist Robert Brown in 1827 while he was observing pollen samples suspended in a liquid. He noted that the motion of the particles was related to heat, the viscosity of the liquid, and the particle size [7]. He was unable to explain how the pollen was moving, and it took Albert Einstein to formulate an equation in 1905 [4]. The equation allows one to predict, at the end of a given time τ, the mean square $\overline{\Delta_x^2}$ of displacement Δ_x of a spherical particle in a given direction x as a function of radius a of the particle, viscosity μ of the liquid, and the absolute temperature T:

$$\overline{\Delta_x^2} = \frac{RT}{N}\frac{1}{3\pi\mu a}\tau$$

where R is the ideal gas constant and N is Avogadro's constant.

The act of generating Brownian motion on a computer is not a trivial task. For several methods, a random number is required (presenting complexities outside of the scope of this paper), and others require evaluation of integrals (requiring an approximation via a Riemann sum). For these reasons, and others, creating Brownian motion is typically a trade off between speed and accuracy. What other papers [3] have

not mentioned are the computational complexities and empirical (or theoretical!) accuracies of various algorithms.

2 Theory

There are two basic types of Brownian Motion which exist, a "regular" type that is not dependent on past displacement, and another type that is.

Brownian Motion: A stochastic process $\{B(t) : t \geq 0\}$ defined on probability space $(\Omega, \mathcal{F}, P)$ (where $\Omega \neq \emptyset$ is the sample space, $\mathcal{F} \subseteq 2^{\Omega}$ is the set of all events, and P is the probability measure such that $P(\Omega) = 1$) is called Brownian Motion if it satisfies:

1. $B(0) = 0$
2. The system $\{B(t) : t \geq 0\}$ is Gaussian on $(\Omega, \mathcal{F}, P)$, and for any t and h with $t+h > 0, B(t+h) - B(t)$ has mean 0, variance $|h|$, and $E\left[|B(t+h) - B(t)|^2\right] \propto |(t+h) - t|$
3. The displacements $B(t+h) - B(t), 0 \leq t_1 \leq t_2 \leq \cdots \leq t_n$, are independent of past displacements.

Fractional Brownian Motion: A stochastic process $\{B(t) : t \geq 0\}$ defined on probability space $(\Omega, \mathcal{F}, P)$ (where $\Omega \neq \emptyset$ is the sample space, $\mathcal{F} \subseteq 2^{\Omega}$ is the set of all events, and P is the probability measure such that $P(\Omega) = 1$) is called Fractional Brownian Motion if it satisfies:

1. $B(0) = 0$
2. The system $\{B(t) : t \geq 0\}$ is Gaussian on $(\Omega, \mathcal{F}, P)$, and for any t and h with $t+h > 0, B(t+h) - B(t)$ has mean 0, variance $|h|$, and $E\left[|B(t+h) - B(t)|^2\right] \propto |(t+h) - t|$
3. The displacements $B(t+h) - B(t), 0 \leq t_1 \leq t_2 \leq \cdots \leq t_n$, are dependent on parameter $H \in [0, 1]$

Definition 2 differs from "regular" Brownian Motion only in that it is dependent on past displacements, meaning that by simply altering H (the Hurst parameter [1], not at all related to h from the definitions above), a correlation will appear between increments. These correlations can be broken down in to cases as follows [10]:

$H = \frac{1}{2}$	Regular Brownian Motion
$H < \frac{1}{2}$	Negative ρ
$H > \frac{1}{2}$	Positive ρ

Fractional Brownian Motion can be easily generated with only a slight modification to the methods given below, as will be shown.

2.1 Implementation

For the purposes of this paper, two different "classes" of methods will be used [3], each of which require a call to a random number generator:

1. Integral of White Noise
2. Midpoint Displacement Method

To remove any uncertainty brought on with the randomness of pseudo-random number generators (RNG's), the RANLUX algorithm will be used. RANLUX generates random numbers uniformly distributed over the open interval (0,1). Each call produces an array of single-precision real numbers of which 24 bits of mantissa are random. It uses a lagged-fibonacci-with-skipping algorithm to produce different "luxury"

levels of random numbers, the highest of which, in addition to passing all known levels of randomness, is theoretically *guaranteed* to be chaotic (uncorrelated) in all 24 bits. [9] For all methods used in this paper, the RNG will be the highest level RANLUX. In all luxury levels, this RNG has a period of about 10^{171} [5].

Due to the choice of uniform RNG mentioned above, it is easy to derive several metrics about the distribution, namely the mean and variance. The probability density funciton (pdf) of any uniform distribution is given by:

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{Otherwise} \end{cases}$$

Since we know $a = 0$ and $b = 1$ for this particular choice of RNG, it is trivial to state the pdf and cumulative distribution function (cdf) for this distribution:

$$f(x) = \begin{cases} \frac{1}{1-0} = 1 & 0 < x < 1 \\ 0 & \text{Otherwise} \end{cases}$$

$$F(x) = \begin{cases} 0 & x \leq 0 \\ \int 1 \; dx = x & 0 < x < 1 \\ 1 & x \geq 1 \end{cases}$$

Finally, calculating the mean and variance will prove useful for later computations:

$$\begin{aligned} E(X) = \mu &= \int_0^1 x \; f(x) \; dx \\ &= \frac{1}{2} \end{aligned}$$

$$\begin{aligned} E((X-\mu)^2) = \sigma^2 = \text{var}(X) &= \int_0^1 (x-\mu)^2 \; f(x) \; dx \\ &= \frac{1}{12} \end{aligned}$$

The **Central Limit Theorem (CLT)** says that the mean and the sum of a random sample of n independent, identically distributed random variables $X_1, X_2, \ldots, X_n$ have approximately normal distribution as $n \to \infty$.

Standardizing a Random Variable: A random variable X is standardized by subtracting its expected value and dividing the difference by its standard deviation:

$$Z = \frac{X - \text{E}(X)}{\sqrt{\text{var}(X)}}$$

The resulting random variable $Z \sim N(0, 1)$.

Since we know the expected value and the variance, it is now easy to state the standardized value Z:

$$Z = \frac{X - \frac{1}{2}}{\sqrt{\frac{1}{12}}}$$

Now, to apply the Central Limit Theorem, we let Z_n denote the standardized sum of any n samples of our random variable X. That is:

$$Z_n = \frac{\sum_{i=1}^n X_i - \sum_{i=1}^n \frac{1}{2}}{\sqrt{\sum_{i=1}^n \frac{1}{12}}} = \frac{\sum_{i=1}^n X_i - \frac{n}{2}}{\sqrt{\frac{n}{12}}}$$

This Z_n now fits the criteria of Brownian Motion, from **Definition 1** with $\mu = 0$ and $\sigma^2 = |h| = 1$.

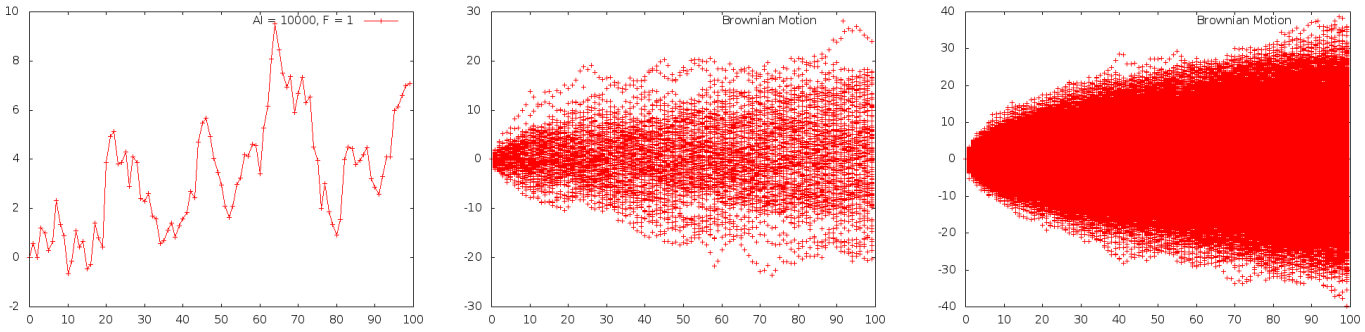
Using the above information, it is now time to state the first algorithm to be used:

```
#define N 100                  \\The number of samples that are desired
#define AI 100                \\The number used to approximate infinity (for CLT)

double B[N];           \\array to hold the y values for the motion, index is the x value
double randn = 0;    \\the is the temporary holding area for the calculation of the gaussian number

B[0] = 0;                \\from the definition of Brownian Motion
for(int i=0; i<N; i++)          \\This will loop from B[0]...B[N]
{
    for(int j = 0; j<AI; j++)
    {
          randn = randn + gsl_rng_uniform_pos (rng);
    }
     B[i+1] = B[i] + (randn - .5*AI)/(sqrt(AI/12));
     cout<<i<<" "<<B[i]<<endl;     \\This outputs the data
}
```

Above, in the statement of the different methods of generating Brownian Motion, I had introduced this as the "Integral of White Noise". When looking at the code above, it can be daunting to spot the actual integration, which occurs discreetly in the line

```
B[i+1] = B[i] + (randn - .5*AI)/(sqrt(AI/12));
```

Thanks to the definition of Brownian Motion, this can be thought of intuitively in a recursive fashion, where the base case is `B[0] = 0` (from the definition), and the subsequent additions simply add the previous displacement in to a "total", which represents the area under the curve, or the integral.



(a) One Iteration (b) One Hundred Iterations (c) Ten Thousand Iterations

Figure 1: Examples of Repeated Iterations of Brown Noise

From the above, it should be fairly easy to see that, given enough iterations, the points will be distributed along the y-axis in a Gaussian distribution. The evidence is deferred for a later section.

The second algorithm works by setting the initial endpoints of the plot, and then recursively displacing the midpoint by an amount returned from a Gaussian distribution. Due to the recursion, and the similarity to merge sort, it is best to state the endpoint as a power of 2, namely $2^{\text{levels}}+1$, where "levels" is the desired number of recursive levels. From the definition, we know that the left endpoint needs to be set to 0, that is, set $B(0) = 0$, and then set the right endpoint by a single call to the Gaussian distribution function with mean 0 and variance 1, $B(2^{\text{levels}}+1) = \text{Gauss}()$. In order to retain some similarity to the previous algorithm, I have decided to use $129 = 2^7 + 1$ points.
Since I am using the Gauss() function developed in a previous section, we know that $\sigma^2 = 1$, and from the

definition of Brownian Motion, we know $E\left[|B(t+h)-B(t)|^2\right]=\sigma^2=1$. Now, the statement $E\left[|B(t+h)-B(t)|^2\right]\propto|(t+h)-t|$ can be easily seen to have a proportionality constant of $\sigma^2=1$, therefore, $E\left[|B(t+h)-B(t)|^2\right]=|(t+h)-t|$ in our case.[1] The recursive aspect of the algorithm then says that we set the midpoint of the line between $B(t+h)$ and $B(t)$ to be the average, plus (or minus) some Gaussian offset D_1 with mean 0 and some new variance Δ_1^2:

$$B(t+\frac{h}{2})-B(t)=\frac{1}{2}\left(B(t+h)+B(t)\right)+D_1$$

Since we know $E\left[|B(t+h)-B(t)|^2\right]=|(t+h)-t|$, the implication is that $E\left[|B(t+\frac{h}{2})-B(t)|^2\right]=|(t+\frac{h}{2})-t|=|\frac{h}{2}|$. Therefore, we can definitively say what D_1's variance is:

$$E\left[|B(t+\frac{h}{2})-B(t)|^2\right]=E\left[\frac{1}{2}|B(t+h)-B(t)|^2+D_1\right]=\frac{1}{4}E\left[|B(t+h)-B(t)|^2\right]+\Delta_1^2$$

$$\Rightarrow|\frac{h}{2}|=\frac{1}{4}|h|+\Delta_1^2\Rightarrow\Delta_1^2=\frac{h}{4}$$

Using the same method, it is trivial to expand this out to the n^{th} level, yielding the result:

$$\Delta_n^2=\frac{h}{2^{n+1}}$$

In the following algorithm, these values are stored in the array named `std[]`.

```
#define N 129   //this defines the number of points, N = 2^{maxlevel}+1

void MidPointRecursion(double a[], int left, int right, int level, int maxlevel, double std[])
{
  //------CLT approximation--------------
  double randn = 0;
  for(int k = 0; k<10; k++)
  {
       randn = randn + gsl_rng_uniform_pos (rng);
  }
       randn=(randn - 5)/(2.8867513);
  //----Done with CLT approximation-------
  int m = (left+right)/2;
  a[m] = ((a[left]+a[right])/2) + std[level] * randn;
  //------------Begin recursion-----------
  if(level<maxlevel)
  {
    MidPointRecursion(a, left, m, level+1, maxlevel, std);
    MidPointRecursion(a, m, right, level+1, maxlevel, std);
  }
}

int main()
{
  double *b = new double[N];
  double std[8];
```

[1] This actually verifies the calculation from the CLT section, $E\left[|B(t+h)-B(t)|^2\right]=|(t+h)-t|=1\rightarrow|h|=\sigma^2=1$, which is stated in the definition of Brownian Motion.

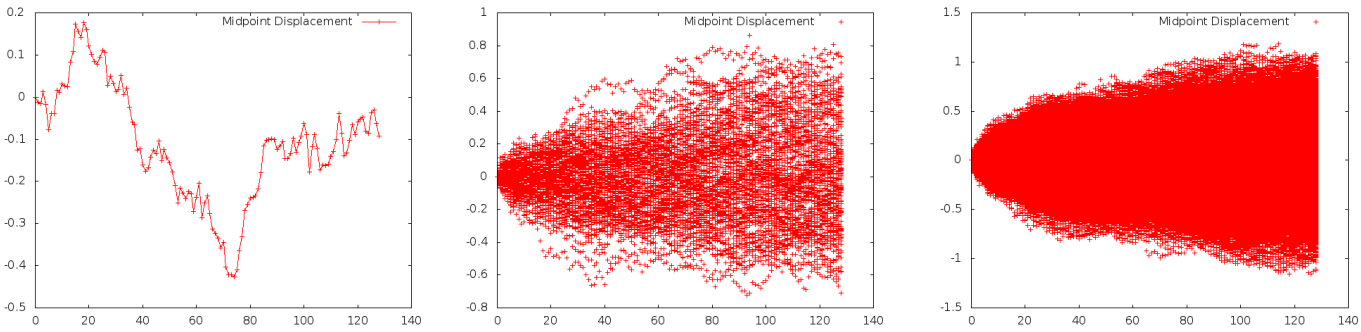
```
  int maxlevel = 7;

  for(int i=1; i<=7; i++)
  {
    std[i] = pow(.5, (i+1)/2);  \\this array holds the variances for each level
  }

  b[0] = 0;

  double randn = 0;
  for(int k = 0; k<10; k++)
  {
    randn = randn + gsl_rng_uniform_pos (rng);
  }
  b[N-1]=(randn - 5)/(2.8867513);

  MidPointRecursion(b, 0, N-1, 1, maxlevel, std);
}
```

The above algorithm makes it exceedingly simple to create fractional Brownian Motion. The only change was in the definition of the motion, which equates out to a slight change in the above development as follows:

$$E\left[|B(t+h) - B(t)|^2\right] = |(t+h) - t|^{2H}\sigma^2$$

Where H is the aforementioned parameter, called the Hurst parameter. The only change in the algorithm occurs in how the array `std[]` is filled:

```
for(int i=1; i<=7; i++)
  {
    std[i] = pow(.5, (i*H)*sqrt(1-power(2,2*H-2);
  }
```



(a) One Iteration (b) One Hundred Iterations (c) Ten Thousand Iterations

Figure 2: Examples of Repeated Iterations of Midpoint Displacement

One thing that is immediately evident is that the Midpoint Displacement algorithm has a much, much smaller variance, as will be shown later.

3 Computational Complexity

It is quite trivial to state the complexity if we stare at the main algorithm contained in the Brown Noise generating program for long enough:

```
for(int i=0; i<N; i++)
{
    for(int j = 0; j<AI; j++)
    {
          randn = randn + gsl_rng_uniform_pos (rng);
    }
     B[i+1] = B[i] + (randn - .5*AI)/(sqrt(AI/12));
     cout<<i<<" "<<B[i]<<endl;
}
```

All of the cost is contained in four things,

1. The outer loop from 0 to N
2. The inner loop from 0 to AI
3. The cost of a call to `ranlux`
4. The cost of a call to `sqrt()`

Therefore, the cost of the Brown Noise algorithm can be stated generally as $O(N \cdot AI \cdot c(\texttt{ranlux}()) \cdot c(\texttt{sqrt}()))$, where $c(fcn())$ represents the cost of a function $fcn()$. This allows for different RNG's and methods of finding square roots to be used while not abstracting over their costs. It is useful to note that if AI is replaced with a constant of sufficient size, the complexity can be stated as $O(N \cdot c(\texttt{ranlux}()))$. In the next section, it becomes apparent that AI can be replaced with a number such that `sqrt(AI/12)` can be replaced with a constant and the accuracy of the model is not affected.
Additionally, it is trivial to extend this algorithm to work in n dimensions, in which case the code would simply be modified to have as many 'for' loops as there are dimensions. In this case, the complexity would be $O(N^n \cdot AI \cdot c(\texttt{ranlux}()) \cdot c(\texttt{sqrt}()))$.

Due to the divide and conquer nature of the algorithm, the cost of the Midpoint Displacement algorithm involves $\log_2$ in a couple of places:

- The array `std[]` has $\log_2 N$ slots, each of which need filling
- Much like the merge-sort algorithm, in order to visit each index, the algorithm needs to recur $\log_2 N$ times

At each recursion, there is a call to the $c(\texttt{ranlux}())$ RNG, therefore the total cost is stated as $\log_2 N + c(\texttt{ranlux}()) \cdot \log_2 N$, which grows no faster than $O(c(\texttt{ranlux}()) \cdot \log_2 N)$. From this analysis, it is plain to see that the Midpoint Displacement algorithm is much quicker than the Brown Noise algorithm.[2]

4 Accuracy

To determine the accuracy of the Brown Noise algorithm, I used both Microsoft Excel and GNUplot to perform a linear regression on the data (it never hurts to double check!) and I then decided that a good metric of accuracy would be to add the absolute value of the calculated slope to the absolute value of the calculated y-intercept. Finally, I plotted this data in two plots, one where the number of iterations is held fixed, and the other where the number used to approximate infinity was held fixed.

[2] Anecdotally, I was able to compute 100,000 instances of Midpoint Displacement, whereas I was only able to do 10,000 with the Brown Noise algorithm.

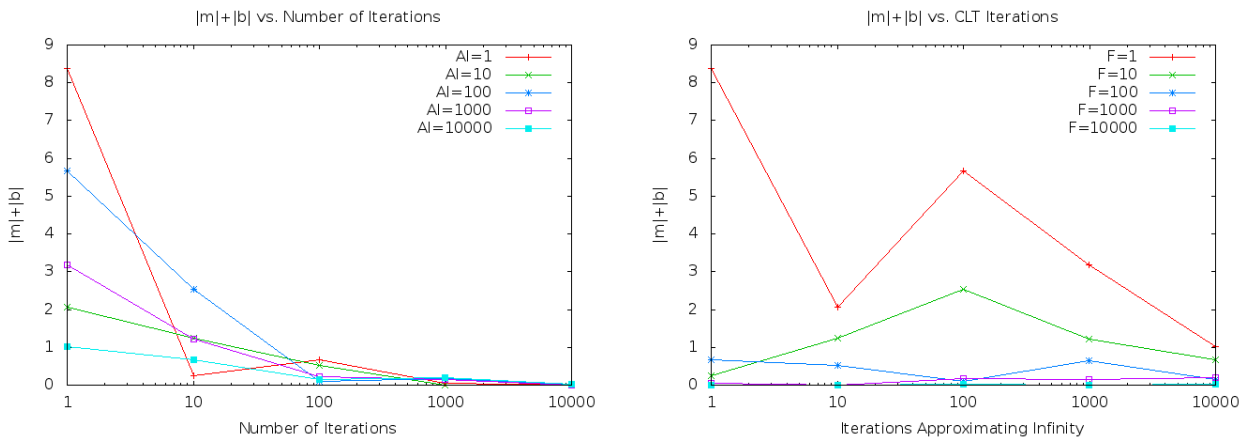
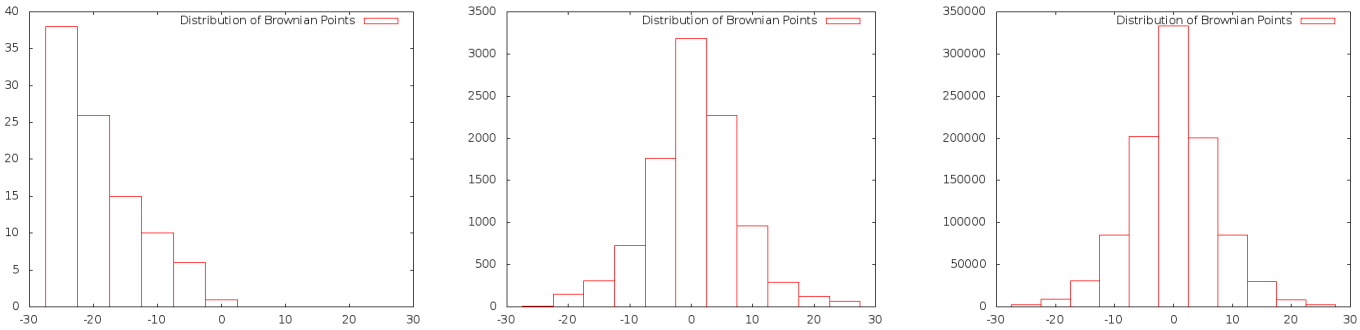


Figure 3: abs(m) + abs(b) versus iterations

From the above graphs, it should be obvious that the number of iterations approximating infinity doesn't matter nearly as much as the number of files that went in to the computation. It is for this reason that the second algorithm does not have a variable for `AI`, I simply used a fixed number, 10. Another thing these plots make obvious is an empirical validation of the Central Limit Theorem. Looking at the plot on the left, the average (or line of best fit) through all of the lines is monotonically decreasing, and obviously bounded below by 0. Therefore, as the number of iterations in increased to infinity, the sum of $|m| + |b|$ is convergent to 0, meaning that the mean tends to zero.[3] The next set of plots make this even more clear:



(a) One Iteration (b) One Hundred Iterations (c) Ten Thousand Iterations

Figure 4: Distributions of Repeated Iterations of Brown Noise

The above plots of the distribution of points generated via the Brown Noise algorithm make it very obvious that even though the mean is zero, the variance is rather large, seemingly converging to 50, as this table and these plots will demonstrate:

[3]Note that this says nothing about the convergence of variance.

Iterations	Mean	Variance
1	4.60152	14.79306683
10	-0.024189102	47.308555
100	0.450178955	43.97742386
1000	0.243488	52.38356
10000	0.000218648	49.7795

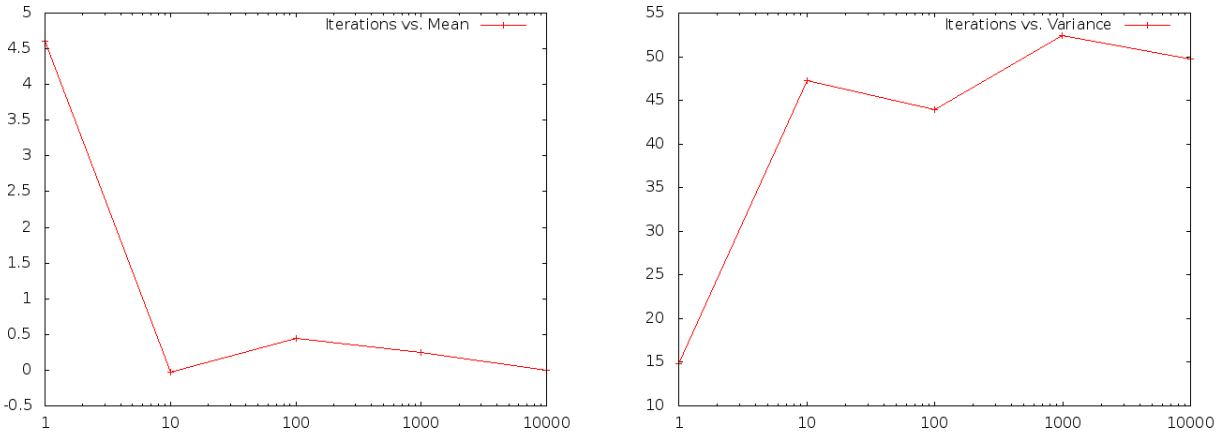
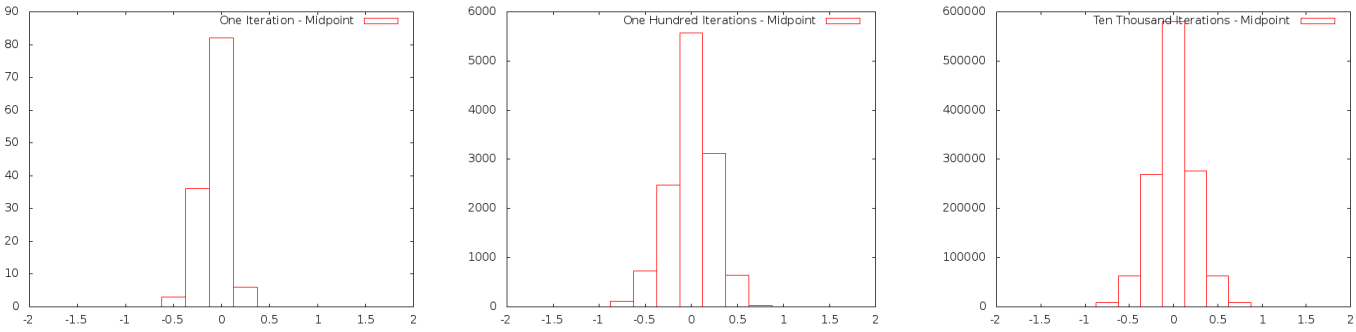


Figure 5: Iterations versus Mean and Variance for Brown Noise

Much like the results from the Brown Noise algorithm, the mean converges to zero as the number of iterations increases, but the variance does not converge to what I had anticipated, this time appearing to want to converge near 0.055.



(a) One Iteration (b) One Hundred Iterations (c) Ten Thousand Iterations

Figure 6: Distributions of Repeated Iterations of Midpoint Displacement

Iterations	Mean	Variance
1	-0.23456	0.014774
10	-0.07847	0.046408
100	0.004321	0.059228
1000	0.003683	0.055179
10000	-0.00074	0.055568

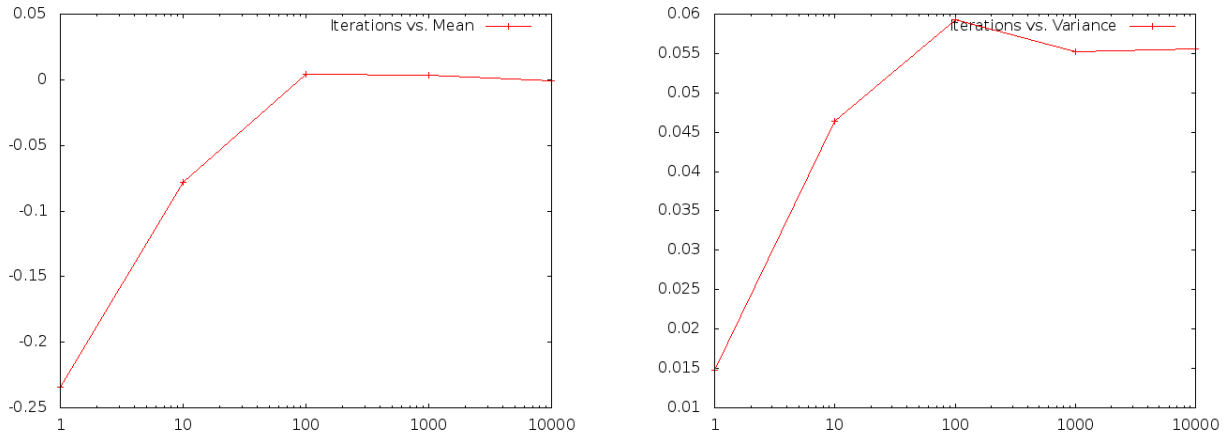


Figure 7: Iterations versus Mean and Variance for Midpoint Displacement

5 Verifying Einstein's Equation

The last part of the analysis of the algorithms is a quick exploration of Einstein's equation. For this, I wanted to replicate some of the parameters of Robert Brown's initial experiment. To that end, I performed two computations; one where I fixed temperature and viscosity, and one where I fixed particle size.
Since the parameters R and N are constants, it becomes necessary to fix some other variables, and then solve for the remaining one. I wrote a short program that calculated the mean squared displacement for each time τ, eliminating two variables. The results of this program were that the Midpoint Displacement algorithm had a mean displacement on the order of 10^{-9} until about time 20, at which point it settled on the order of 10^{-7} until the max time. The Brown Noise algorithm had a mean displacement on the order of 10^{-5} again until about time 20, finally settling on the order of 10^{-4} until its max time. To calculate the size of the particle, I used water's viscosity at 20°C, because it is very nearly 1.[4] The resulting particle size was calculated by manipulating the aforementioned equation to solve for a:

$$a = \frac{4.04737 \times 10^{-21}\tau}{3\pi\,(1.002)\,\overline{\Delta_y^2}}$$

When inserting the values from Midpoint Displacement, the particles are sized on the order of 10^{-13}, or roughly the size of an electrons radius, while Brown Noise resulted in particles the same size as up and down quarks.[5]

For the next calculation, I fixed particle size to be that of a pollen grain [2] (a nod to Brown), and then calculated the ratio of absolute temperature to viscosity:

$$\frac{T}{\mu} = \frac{3\pi\overline{\Delta_y^2}aN}{R\tau}$$

Inserting values from either algorithm resulted in a ratio on the order of 10^9 to 10^{13}. Inserting an everyday temperature in to the ratio to solve for viscosity tells us that viscosity is roughly 10^{-11}. The only compounds that exhibit such viscosity are superfluids. Going the other way, if we decide to fix viscosity, the temperature will need to be incredibly high. So high, in fact, that the viscosity of the liquid would be approaching 0, thus raising the requisite temperature in a divergent fashion.

[4] (1.002, to be exact [6])
[5] I have literally *no idea* what this means or could be used for.

6 Applications

As was mentioned briefly above, the Brownian Motion algorithms can be extended to n dimensions. It just so happens that when the motion is graphed in two dimensions (with a cost of $O(n^2 \cdot c(\texttt{ranlux}()))$), it portrays a quite convincing model of scenery:

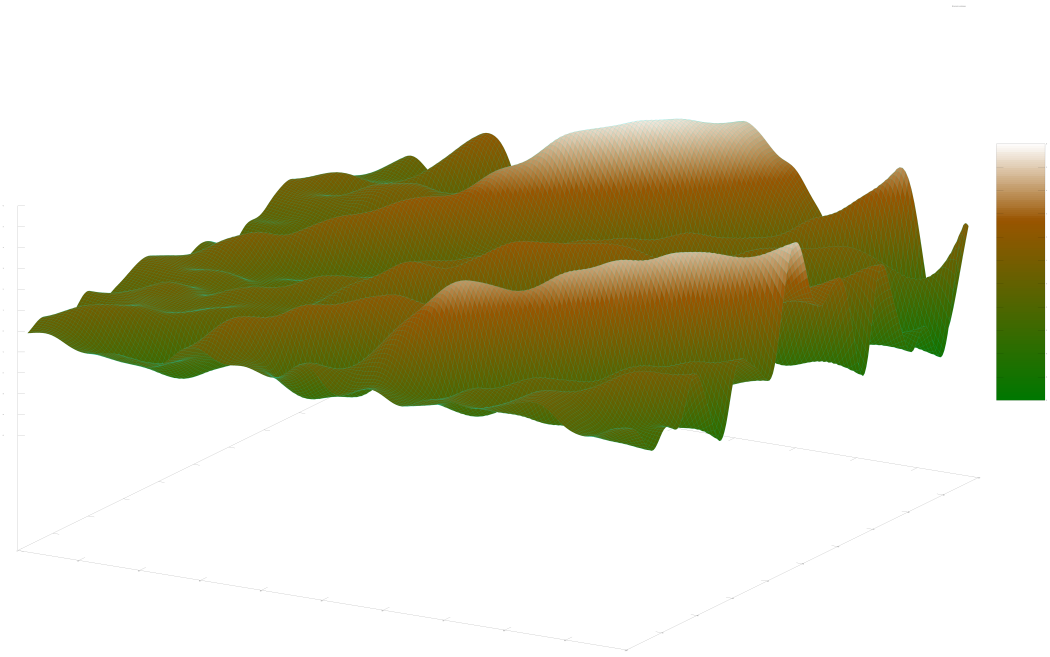


Figure 8: Brown Noise generated landscape

Using GNUplot's built in `dgrid3d` function and smoothing algorithms, one can simply try different values for the average to vary the "smoothness", going from very jagged landscapes to smooth rolling hills.

7 Future Work and Comments

There are several other methods of generation, namely Fourier filtering, and spectral analysis [3] which work in completely different ways than the methods mentioned here, and they would have been fascinating to explore. Additionally, it would have been fun to compare different, less costly, RNG's in order to see how they would alter the results. It would also be interesting to verify the predictor of the ultimate maximum of Brownian Motion that was recently published. [8]

I was bothered that neither of my algorithms exhibited a variance of 1, being that they followed the standard normal distribution. I can perhaps see why the Brown Noise algorithm's variance was larger than 1, due to the integration, but I cannot fathom why the Midpoint Displacement algorithm failed to produce a variance of 1.

I do not understand where I went wrong when verifying Einstein's equation. In my mind, I should have been able to pick three times τ_1, τ_2, τ_3 and solved a system of equations involving them. I could not do this; the numbers did not equate. I have great faith in Einstein and his equation, and it therefore makes me suspect that my algorithms or understanding were incorrect.

References

[1] R. G. Clegg. A practical guide to measuring the Hurst parameter. *ArXiv Mathematics e-prints math/0610756*, October 2006. http://adsabs.harvard.edu/abs/2006math.....10756C.

[2] Jan Derksen and Elisabeth Pierson. Shape, Color and Size [of Pollen]. *Radboud University Nijmegen*, October 2011. http://www.vcbio.science.ru.nl/en/virtuallessons/pollenmorphology/.

[3] Dietmar Saupe. Algorithms for Random Fractals. *Springer New York*, 1988.

[4] Albert Einstein. On the Motion of Small Particles Suspended in a Stationary Liquid, as Required by the Molecular Kinetic Theory of Heat. *Annalen der Physik*, 17, July 1905.

[5] F. James. RANLUX: A Fortran implementation of the high-quality pseudo-random number generator of Lüscher. *Computer Physics Community*, 79, 1994.

[6] Mordechai Sokolov Joseph Kestin and William A. Wakeham. Viscosity of Liquid Water in the Range -8°c to 150°c. *Journal of Physical and Chemical Reference Data*, 7(3), 1978.

[7] M. P. Langevin. On the Theory of Brownian Motion. *C. R. Acad. Sci.*, 146, 1908.

[8] Jesper Lund Pedersen. Optimal Prediction of the Ultimate Maximum of Brownian Motion. *University of Copenhagen*.

[9] M. Lüscher. A portable high-quality random number generator for lattice field theory simulations. *Computer Physics Community*, 79, 1994.

[10] Matthew Moore. One Dimensional Brownian Motion. *University of Toronto*, April 2002.

8 Appendix

This appendix contains all source code for both algorithms, and the 3D Brown Noise algorithm. It also has a sequence diagram for the Brown Noise program, illustrating how the shell scripts interfaced with GNUplot and the main C++ program.[6]

8.1 Sequence Diagram of the Brown Noise program

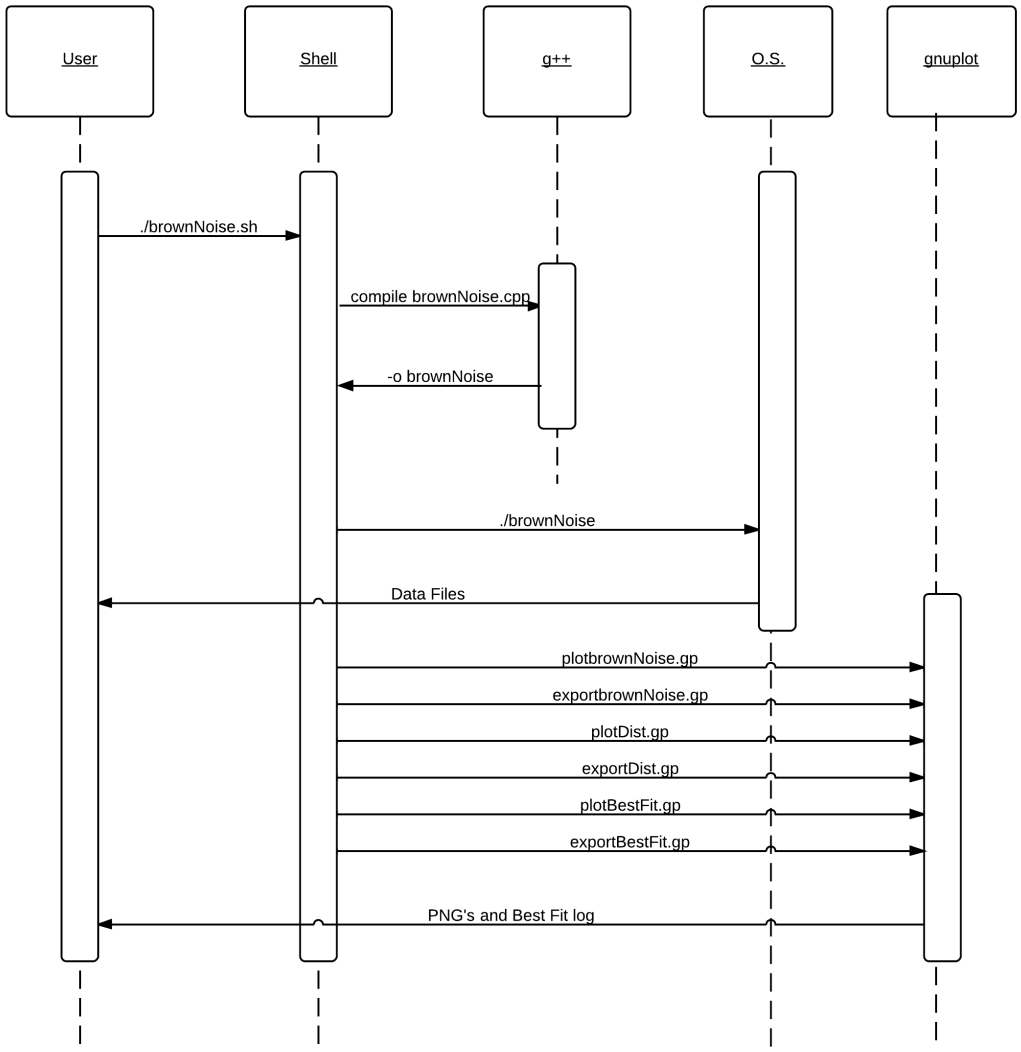


Figure 9: Brown Noise Sequence Diagram

[6]I did not write a shell script for the Midpoint Displacement algorithm, as I formatted the output files differently.

8.2 C++ Code

8.2.1 Brown Noise Algorithm

```
#include <iostream>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <time.h>
#include <math.h>
#include <fstream>

using namespace std;

#define N 100   //this defines the number of points
#define AI 10000   //this approximates infinity
#define F 10000   //this defines the number of files

int main()
{
  ofstream datafile, distfile;

  int points = 0;

  datafile.open("data.txt");
  distfile.open("dist.txt");

  //the following four lines just sets up the variables used in dist.txt
  int zero = 0, posten = 0, postwen = 0, posthir = 0, posfor = 0;
  int posfif = 0, negten = 0, negtwen = 0, negthir = 0, negfor = 0, negfif = 0;
  int one = -25, two = -20, three = -15, four = -10, five = -5;
  int six = 0, seven = 5, eight = 10, nine = 15, ten = 20, eleven = 25;

  int total = 0;

  double randn;

  double b[N] = {0};


  gsl_rng *rng;
  rng = gsl_rng_alloc (gsl_rng_ranlxs2);
  gsl_rng_set (rng, time(NULL)); //setting up the random number gen


  for(int j = 0; j<F; j++)  //recall F is the number of 'files' (iterations)
  {

b[0] = 0;

   datafile<<0<<"\t"<<0<<"\n";  //because b[0] = 0

   for(int i=1; i<N; i++)  //N is the number of points per iteration
```

```
   {
randn = 0;
for(int k = 0; k<AI; k++)
{
     randn = randn + gsl_rng_uniform_pos (rng);
}

     b[i] = b[i-1] + (randn - .5*AI)/(sqrt((1.0/12)*AI));

     datafile<<i<<"\t"<<b[i]<<"\n";
points++;  //the total number of points created


    //the following is for calculating the output for dist.txt
if(b[i] <= 2.5 && b[i] >=-2.5)
{
total++;
zero++;
}
else if(b[i] < -2.5 && b[i] >=-7.5)
{
total++;
negten++;
}
else if(b[i] < -7.5 && b[i] >=-12.5)
{
total++;
negtwen++;
}
else if(b[i] < -12.5 && b[i] >=-17.5)
{
total++;
negthir++;
}
else if(b[i] < -17.5 && b[i] >=-22.5)
{
total++;
negfor++;
}
else if(b[i] < -22.5 && b[i] >=-27.5)
{
total++;
negfif++;
}
else if(b[i] <= 7.5 && b[i] >2.5)
{
total++;
posten++;
}
else if(b[i] <= 12.5 && b[i] >7.5)
{
```

```
total++;
postwen++;
}
else if(b[i] <= 17.5 && b[i] >12.5)
{
total++;
posthir++;
}
else if(b[i] <= 22.5 && b[i] >17.5)
{
total++;
posfor++;
}
else if(b[i] <= 27.5 && b[i] >22.5)
{
total++;
posfif++;
}

   }

   }


gsl_rng_free (rng);  //this frees the random number generator
  //this outputs the distribution to dist.txt
  distfile<<one<<"\t"<<negfif<<endl;
  distfile<<two<<"\t"<<negfor<<endl;
  distfile<<three<<"\t"<<negthir<<endl;
  distfile<<four<<"\t"<<negtwen<<endl;
  distfile<<five<<"\t"<<negten<<endl;
  distfile<<six<<"\t"<<zero<<endl;
  distfile<<seven<<"\t"<<posten<<endl;
  distfile<<eight<<"\t"<<postwen<<endl;
  distfile<<nine<<"\t"<<posthir<<endl;
  distfile<<ten<<"\t"<<posfor<<endl;
  distfile<<eleven<<"\t"<<posfif<<endl;

  //the following cout tells how many points were captured in the dist.txt file
  cout<<"Captured "<<total<<" points out of a possible "<<points;
  cout<<" for a total of "<<((float)total/points)*100<<"% of all points"<<endl;

  datafile.close();
  distfile.close();
  return 0;
}
```

8.2.2 Midpoint Displacement

```
#include <iostream>
#include <gsl/gsl_rng.h>
```

```
#include <gsl/gsl_randist.h>
#include <time.h>
#include <math.h>
#include <fstream>
#include <sstream>

using namespace std;

#define N 129   //this defines the number of points

using namespace std;


void MidPointRecursion(double a[], int left, int right, int level, int maxlevel, double std[])
{
//---------Setting up RNG--------------
gsl_rng *rng;
   rng = gsl_rng_alloc (gsl_rng_ranlxs2);
   gsl_rng_set (rng, rand()%time(NULL));
//-------Done setting up---------------


//------CLT approximation--------------
double randn = 0;
for(int k = 0; k<10; k++)
{
     randn = randn + gsl_rng_uniform_pos (rng);
}
     randn=(randn - 5)/(2.8867513);
//----Done with CLT approximation-------

int m = (left+right)/2;
a[m] = ((a[left]+a[right])/2) + std[level] * randn;

//------------Begin recursion-----------
if(level<maxlevel)
{
MidPointRecursion(a, left, m, level+1, maxlevel, std);
MidPointRecursion(a, m, right, level+1, maxlevel, std);
}

//gsl_rng_free (rng);
}

int main()
{
//-------Setting up RNG-----------
gsl_rng *rng;
   rng = gsl_rng_alloc (gsl_rng_ranlxs2);
   gsl_rng_set (rng, rand());
//---------Done setting up---------
```

```
int F;
string filename;
stringstream ss;

cout<<"Enter desired number of files: ";
cin>>F;

ss<<F;
ss>>filename;
filename="F=" + filename + ".txt";

ofstream outfile;
outfile.open(filename.c_str());

double *b = new double[N];

double std[8];

int maxlevel = 7;

for(int i=1; i<=7; i++)
{
std[i] = pow(.5, (i+1)/2);
}

for(int file=0; file<F; file++)
{

b[0] = 0;

double randn = 0;
for(int k = 0; k<10; k++)
{
     randn = randn + gsl_rng_uniform_pos (rng);
}

     b[N-1]=(randn - 5)/(2.8867513);

MidPointRecursion(b, 0, N-1, 1, maxlevel, std);



for(int i =0; i<N; i++)
{
outfile<<i<<"\t"<<b[i]<<endl;
}
}

gsl_rng_free (rng);
```

```
return 0;
}
```

8.3 Shell Script for Execution

```
#!/bin/bash
g++ -Wall brownNoise.cpp -lgsl -lgslcblas -lm -o brownNoise
./brownNoise
gnuplot plotBrownNoise.gp exportBrownNoise.gp
gnuplot plotDist.gp exportDist.gp
gnuplot plotBestFit.gp exportBestFit.gp
```

8.4 Plotting Procedures for GNUplot

8.4.1 Plotting Brown Noise

```
#save as plotBrownNoise.gp
set origin 0, 0
plot "data.txt" u 1:2 t "Brownian Motion"
```

8.4.2 Plotting the Distribution

```
#save as plotDist.gp
set origin 0, 0
plot "dist.txt" u 1:2 t "Distribution of Brownian Points" w boxes
```

8.4.3 Calculating/Plotting the Best Fit

```
#save as plotBestFit.gp
set origin 0, 0
f(x) = m*x+b
fit f(x) "data.txt" u 1:2 via m,b
plot "data.txt" u 1:2 w d t "", f(x) t "Best Fit"
```

8.5 Export Procedure for GNUplot

8.5.1 Exporting Brown Noise

```
#save exportBrownNoise.gp
set terminal push     #save the current terminal settings
set terminal png      #change terminal to PNG
set output "output.png" #set the output filename to the first option
replot                        #repeat the most recent plot command
set output              #restore output to interactive mode
set terminal pop      #restore the terminal
```

8.5.2 Exporting the Distribution

```
#save as exportDist.gp
set terminal push     #save the current terminal settings
set terminal png       #change terminal to PNG
set output "dist.png"       #set the output filename to the first option
replot                         #repeat the most recent plot command
set output                #restore output to interactive mode
```

```
set terminal pop      #restore the terminal
```

8.5.3 Exporting the Best Fit

```
#save as exportBestFit.gp
set terminal push     #save the current terminal settings
set terminal png       #change terminal to PNG
set output "bestFit.png"      #set the output filename to the first option
replot                        #repeat the most recent plot command
set output               #restore output to interactive mode
set terminal pop      #restore the terminal
```