One Dimensional Brownian Motion

Matthew Moore
MAT335H1 — Chaos, Fractals, and Dynamics
Professor R. Pyke

(Submitted 24 April 2002)

TABLE OF CONTENTS

INTRODUCTION:

Most mathematical concepts today have their foundations from over 2000 years ago. These concepts are the basis of current definitions of physical phenomena in areas such as physics, chemistry, and engineering. Our current understandings of mathematical properties are described by formulas and are dependent on scales. For example, a circle in the plane R^2 is described by the formula $x^2+y^2=r^2$, where x and y represent the cardinal directions of the plane which are spaced by some numerical factor. The radius is represented by r which is expressed by a numerical factor relative to the cardinal directions of the plane. In the past 20 years a new area of mathematics has surfaced in professional areas where, instead of using formulae for modeling, algorithms are used. This area of mathematics is called fractal geometry, a term coined by its developer Benoit B. Mandelbrot when working at the IBM T.J. Watson Research Center. Mandelbrot geometry has provided a better mathematical model description of complex forms in nature. Fractals are most notable for their lack of scale, self similarity, as well as their randomness. Random processes are the main focus of research by analyzing dynamical systems in order to understand how these 'chaotic' (or undetermined) systems function. One such dynamical system is Brownian Motion named after Robert Brown, regarding his work on the random movement of particles.

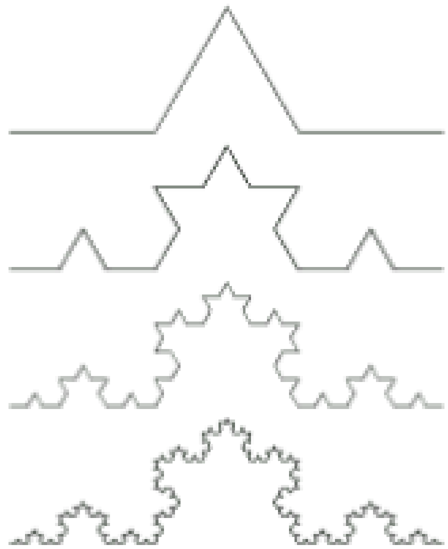
Brownian motion is the erratic movement of microscopic particles. This property was first observed by botanist Robert Brown in 1827, when Brown conducted experiments regarding the suspension of microscopic pollen samples in liquid solution. The experiments showed that the motion of the particles, in a given time interval, was related to heat, the viscosity of the liquid, and the particle size. Temperature is the only factor that has a direct relationship to particle motion whereas viscosity and size have indirect relationships. Particles move more rapidly as the temperature increases, as the viscosity decreases, or when the average particle size is lowered.

At the end of the 19th century, the Kinetic theory of matter was developed which gave further understanding to Brown's observations. Atoms or molecules are in constant motion, where the velocity of the particle is determined by the temperature of the entire system. Thus each particle collides with a neighbouring particle affecting the velocities of the parties involved in the collision. The overall effect of the collisions is the erratic, random, motion of particles. In 1905, Albert Einstein "succeeded in stating the mathematical laws governing the movements of particles on the basis of the principles of the kinetic molecular theory of heat." [1]. Thus the kinetic molecular theory is a practical mathematical model used to describe Brownian motion phenomena. Another mathematical model that describes random processes is fractal mathematics.

The word fractal (short for "fractal dimension") is a term used to describe objects made up of smaller shapes that re-occur at various scales. The re-occurring property of fractals means that at any magnification of an image, an identical replica of the whole image will be seen. Mandelbrot describes this property as self similarity. Another property of a fractals is its dimension. The dimension of a fractal measures how rough fractal curves or surfaces are, where roughness is defined as an increase in dimension. For example a curve may be extremely rough to a point where it fills part of a surface. Thus the dimension of a rough curve would be slightly higher than 1 but less than 2.

There are many ways to generate a fractal. For example a formula may be used and iterated over and over again using the previous value as input. This algorithmic process would produce a random set of values. Another method is to have a set of transformations, each of which has a certain probability of occuring. The transformations as a whole make up an image and upon iteration of that image using the transformations, a replica of the image will re-appear. An example of such a process is the von koch curve.

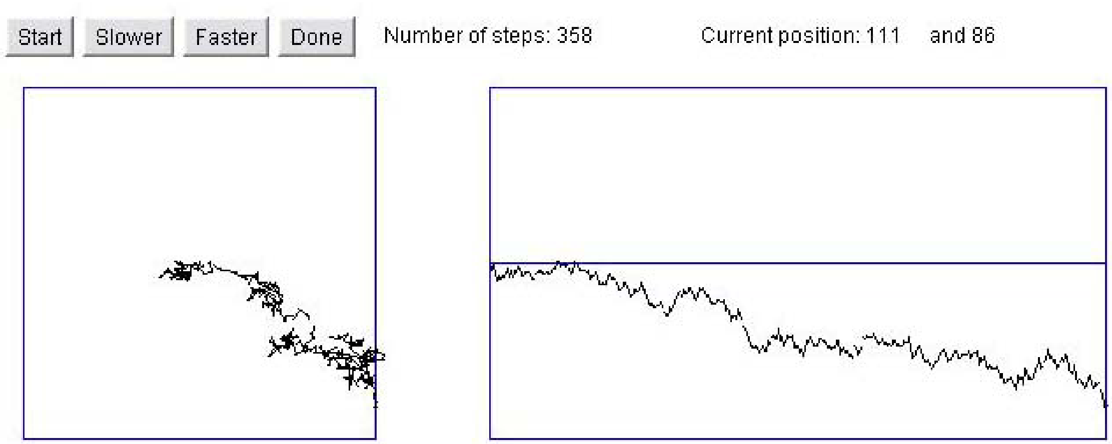
The von koch curve is constrcuted from a line (on the interval [0,1]) where the middle third of the line is removed and replaced by to lines forming an upper part of a triangle. Each segment has a length of 1/3 of the original line segment. Once we iterate this curve a more complex curve emerges:



[2]

Notice that on smaller scales, a replica of the original image can always be seen. Also the fractal dimension, rounded to two decimal places, of the von koch curve is 1.26 . The idea of iterating a function (or more specifically using an algorithm to generate an image) is directly related to reproducing Brownian motion, which will be discussed later on.

When brown conducted his experiments it was done in two dimensions where the motion of a particle is mapped in R^2. However in this form it is difficult to study behaviour in that the path of the particle may overlap (left image). Thus if the displacements from the vertical axis are mapped against time the result would be as follows (right image):



[3]

BROWNIAN MOTION

A Brownian Curve is defined to be a set of random variables of time (in a probability space) which have the following properties:

1. For every h > 0, the displacements $X(t+h)-X(t)$ have Gaussian distribution.
2. The displacements $X(t+h)-X(t)$, $0 < t_1 < t_2 < \ldots < t_n$, are independent of past displacements
3. The mean displacement is zero.

To accomplish generating a Brownian curve, a sample of random data will be created and these values will be plotted. One method to generate a Brownian Curve is to move along one point at a time where each point is displaced vertically upward or downward. Another method is "generate an array of values and integrate them from left to right"[4]. The latter is known as a Fourier Transform. Finally another popular method is Random Midpoint Displacement. This method involves dividing an interval in half and randomly displacing the midpoint vertically upward or downward. Afterwards, the subsequent line segments are divided in half and their respective midpoints are displaced.

Let X(t) be a random variable of time in a sample of random variables. A value for the X is computed for times, t, between the interval 0 and 1. Set X(0) = 0 and set X(1) as a particular sample of a Gaussian random variable with mean zero and initial variance σ^2 equal to the difference between X(1) and X(0).

$$\sigma^2 = \operatorname{var}(X(1)-X(0))$$

$$|t_2 - t_1|\sigma^2 = \operatorname{var}(X(t_2)-X(t_1)) \quad \text{since the mean is zero.}$$

With the midpoint at X(1/2), we now displace this value by some amount D_1 in the vertical direction which has mean zero and variance Δ_1^2. Please note that Δ_1^2 is the variance of the displacement D_1 only.

$$X\left(\frac{1}{2}\right) - X(0) = \frac{1}{2}\left(X(1) - X(0)\right) + D_1$$

$$\mathrm{var}\left(X\left(\frac{1}{2}\right) - X(0)\right) = \frac{1}{4}\mathrm{var}\left(X(1) - X(0)\right) + \Delta_1^2$$

$$\begin{aligned}
|t_2 - t_1|\sigma^2 + \Delta_1^2 &= \mathrm{var}\left(X(t_2) - X(t_1)\right) \\
\frac{1}{4}|1 - 0|\sigma^2 + \Delta_1^2 &= \frac{1}{2}\mathrm{var}\left(X(1) - X(0)\right) \\
\frac{1}{4}\sigma^2 + \Delta_1^2 &= \frac{1}{2}\sigma^2 \\
\Delta_1^2 &= \frac{1}{4}\sigma^2
\end{aligned}$$

Now, when continuing iteration for the line segments $l_1 \in [0, \frac{1}{2}]$, and $l_2 \in [\frac{1}{2}, 1]$ the variance for each displacement would be:

$$X\left(\frac{1}{4}\right) - X(0) = \frac{1}{2}\left(X\left(\frac{1}{2}\right) - X(0)\right) + D_2$$

$$\mathrm{var}\left(X\left(\frac{1}{4}\right) - X(0)\right) = \frac{1}{4}\mathrm{var}\left(\left(X\left(\frac{1}{2}\right) - X(0)\right)\right) + \Delta_2^2$$

$$\begin{aligned}
|t_2 - t_1|\sigma^2 + \Delta_2^2 &= \mathrm{var}\left(X(t_2) - X(t_1)\right) \\
\frac{1}{4}\left|\frac{1}{2} - 0\right|\sigma^2 + \Delta_2^2 &= \mathrm{var}\left(X\left(\frac{1}{2}\right) - X(0)\right) \\
\frac{1}{8}\sigma^2 + \Delta_2^2 &= \frac{1}{4}\sigma^2 \\
\Delta_2^2 &= \frac{1}{8}\sigma^2
\end{aligned}$$

Thus the variance of the displacement for a given random variable may be summarized in the following formula:

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

Note that the displacement of each variable is not completely random. Rather "the movement from one position to the next is selected, at random, from a set of rules, each having a fixed probability of being chosen." [1]. This selection process is achieved using a Gaussian random number generator, which has mean 0 and maximum probability of 1. Generating random numbers is repeated numerous times in the calculations of displacements which are then added to interval midpoints. This repetitive process of diplacing midpoints is synonymous with iterative algorithms used to generate fractal images. Using the Java Programming Language, an algorithm may now be developed in order to visualize the behaviour of Brownian motion.

The following two procedures are responsible for generating the sample data that will be plotted later on. The first algorithm, Generate, creates the starting line segment and determines the variances of each sample value. The next procedure, Divide, actually calculates the midpoint of the line and displaces the midpoint by some random value.

```
public void Generate(int n, double v) {
       int maxpos;
       Double d;
       Var=v;

       d = ( new Double (Math.pow(2,n)) );
       maxpos=d.intValue();

       delta = new double[maxpos+1];

       for(int i = 1; i<n; i++){
            delta[i] = Var*Math.pow(0.5, (i+1)/2);
       }

       Data= new double[maxpos+1];
       Data[0]=0;
       Data[maxpos]=Var*rand.nextGaussian();
       Divide(Data, 0, maxpos, 1, n);

}/* Generate */
```

[5]

The main objective of this method is to calculate the vertical displacements of each sample. This is accomplished, by first calculating the variances of the displacements. To see the full source code refer to Appendix A.

The Generate method executes the following steps:

1. Determine the number of samples to generate based on the number of iterations to carried out.
2. Calculate the variances for each displacement
3. Set the initial sample points X(0) and X(1) to create line segment.

The number of samples to generate is related to the iteration depth, which means how many subsequent divisions of line segments do we wish to have. Since the division is by 2 the number of samples to generate would have to be a multiple of two. Specifically 2^n+1 samples (an extra sample is added because the sample in begins at index zero). Now we are ready to divide the line segment and displace the midpoints using the next method divide.

```
private void Divide(double X[], int a, int b, int level, int n) {
        int c;
        c=(a+b)/2;
        X[c]=(X[a]+X[b])/2 + delta[level]*rand.nextGaussian();

        if (level < n) {
           Divide(X, a, c, level+1, n);
           Divide(X, c, b, level+1, n);
        }
}/* Divide */
```

[5]

Divide simply takes the sample data, divides it in two, and displaces the midpoint value. Each division is tallied so when the maximum number of divisions occur, the process halts. It is a recursive process that displaces the midpoint in each iteration.

FRACTIONAL BROWNIAN MOTION

Fractional Brownian motion is another way to produce brownian motion. Similar to regular Brownian motion, it has the following properties with X(t) representing random variable in a probability space with mean zero and variance σ^2:

1. For every h > 0 $\mathrm{X}(t+h)-\mathrm{X}(t)$ have a Gaussian distribution.
2. The displacements $\mathrm{X}(t+h)-\mathrm{X}(t)$, $0 < t_1 < t_2 < \ldots < t_n$, are dependent on parameter $H \in [0,1]$.

The main difference in fractional brownian motion versus regular brownian motion is the dependence of the displacements on parameter H where the variances have the following relation:

$$\mathrm{var}(\mathrm{X}(t_2)-\mathrm{X}(t_1)) = \sigma^2 |t_2 - t_1|^{2H}$$

Using midpoint methods as before, the variances of the displacements can be obtained:

$$\begin{aligned} \sigma^2 &= \mathrm{var}(\mathrm{X}(1)-\mathrm{X}(0)) \\ |t_2 - t_1|^{2H}\sigma^2 &= \mathrm{var}(\mathrm{X}(t_2)-\mathrm{X}(t_1)) \end{aligned}$$

$$\mathrm{X}\left(\frac{1}{2}\right) - \mathrm{X}(0) = \frac{1}{2}(\mathrm{X}(1)-\mathrm{X}(0)) + D_1$$

$$\mathrm{var}\left(\mathrm{X}\left(\frac{1}{2}\right) - \mathrm{X}(0)\right) = \frac{1}{2^{2H}}\mathrm{var}(\mathrm{X}(1)-\mathrm{X}(0)) + \Delta_1^2$$

$$\begin{aligned} |t_2 - t_1|^{2H}\sigma^2 + \Delta_1^2 &= \mathrm{var}(\mathrm{X}(t_2)-\mathrm{X}(t_1)) \\ \frac{1}{4}\left|\frac{1}{2} - 0\right|^{2H}\sigma^2 + \Delta_1^2 &= \frac{1}{2^{2H}}\mathrm{var}(\mathrm{X}(1)-\mathrm{X}(0)) \\ \frac{1}{4}\sigma^2 \frac{1}{2^{2H}}\sigma^2 + \Delta_1^2 &= \frac{1}{2^{2H}}\sigma^2 \\ \Delta_1^2 &= \frac{\sigma^2}{2^{2H}}\left(1 - 2^{2H-2}\right) \end{aligned}$$

In general the variances of the displacements are:

$$\Delta_n^2 = \frac{\sigma^2}{(2^n)^{2H}}\left(1 - 2^{2H-2}\right)$$

Thus as the parameter H changes, so does the correlation between increments.

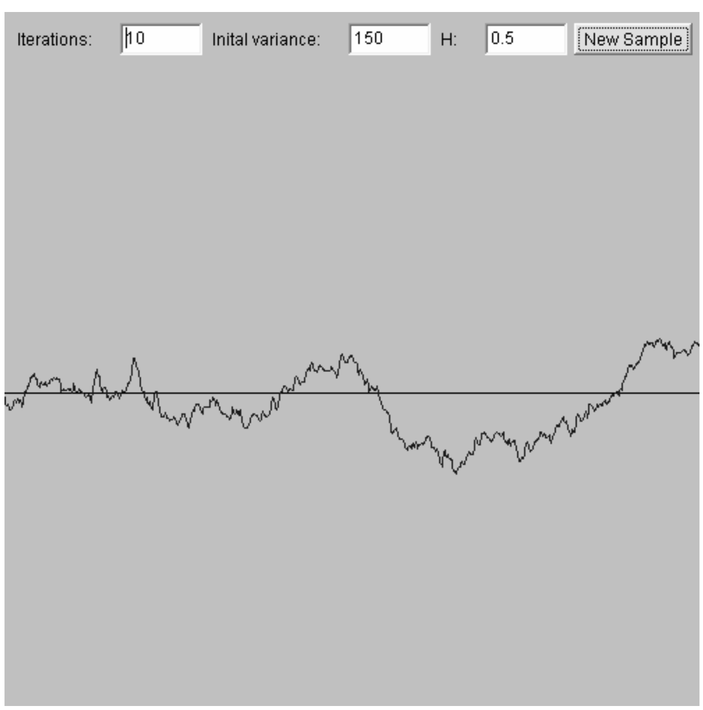
$H = \frac{1}{2}$	Regular Brownian motion
$H < \frac{1}{2}$	Negative correlation meaning the displacements tend to decrease
$H > \frac{1}{2}$	Positive correlation meaning the displacements tend to increase.

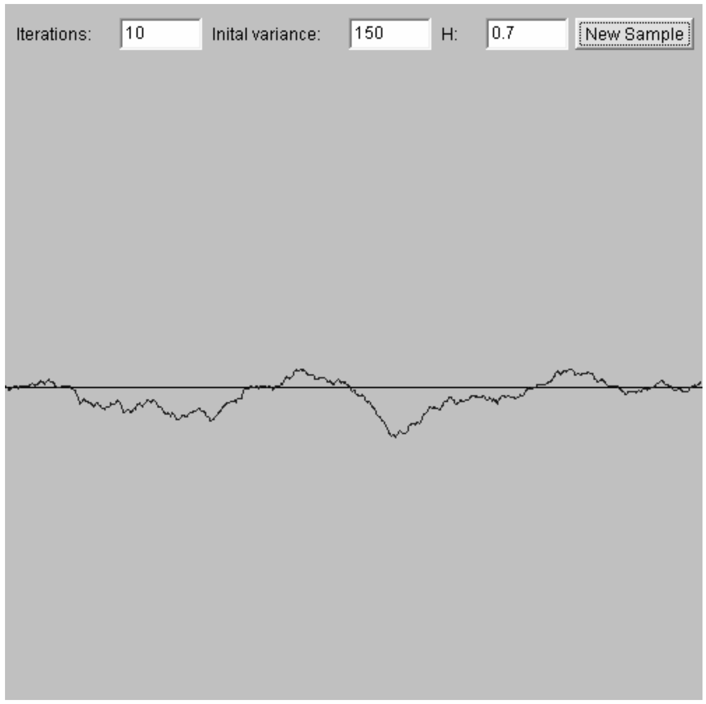
Dependence occurs when the initial displacement is either positive or negative. If the displacement is positive then the displacements will increase, but if negative, the displacements will decrease.

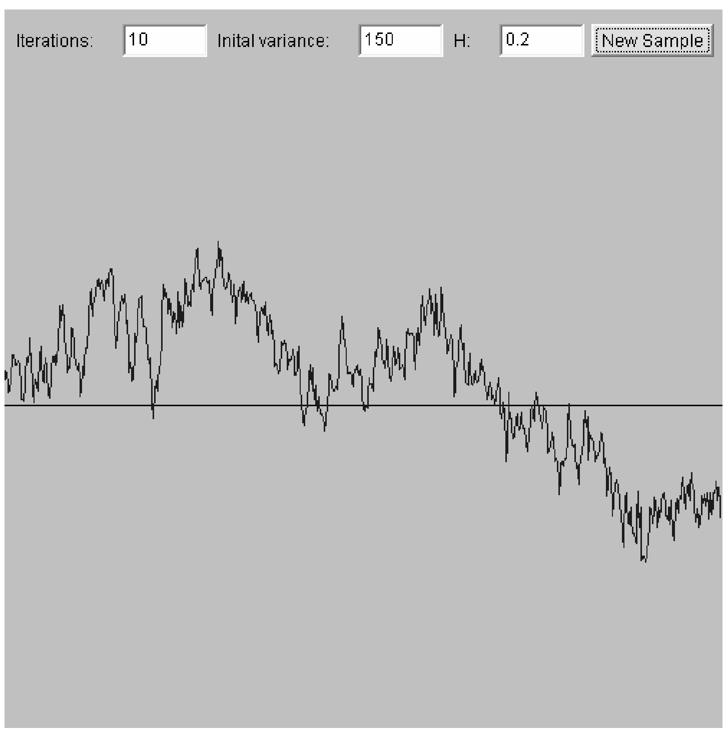
Brownian motion observed naturally is seen on a surface with a topological dimension of two. The dimension of the curve would therefore depend on its roughness which is dependent on the parameter H. Thus the dimension of the brownian curve D_B would be:

$$D_B = 2 - H$$

With this relation, the fractal dimension of regular brownian motion (H=1/2) would be 1.5. When H<1/2 the curve is rougher and closer to filling a surface, and when H>1/2 the curve is smoother.







Therefore the parameter H, called the Hurst exponent [6], describes the roughness of a fractional Brownian curve.

Fractional Brownian motion may also be created using the same midpoint methods for regular brownian motion. The procedure Generate would again be used with modification (Appendix A). This modification incorporates the parameter H when calculating the variances of the displacements.

CONCLUSION

Random processes are very prevalent in nature from Brown's original experiment with particles to stock market analysis. The methods used to recreate Brownian motion are vital tools in learning how certain processes behave in the hope that predicting behaviour may be possible to serve current scientific needs. One main use of brownian motion is in the entertainment industy where the resulting Brownian curves, created in 2 dimensions, model very realistic landscapes which have fractal dimension between 2 and 3 . Fractal geometry has given rise to an area of math that gives futher understanding of how dynamic systems work and as the research continues in this area, more practical uses will be discovered.

APPENDIX A — Java Source Code

```java
/*  PROGRAM:        Brownian Motion Applet
 *  FILE:           BrownianApplet.java
 *  AUTHOR:         Matthew Moore
 *  DESCRIPTION:    Interactive programs that creates a Brownian Curve
 *                  representing random movement over time.
 *  ENVIRONMENT:    JBuilder 3, Java v.1.3.1
 *  NOTES:          Adapted from code found at web site
 *                   http://hobbe.lub.lu.se/~hakan/papers/fract/fract/node12.html
**/

import java.awt.*;
import java.awt.Graphics;
import java.applet.*;
import BrownianCurve;

public class BrownianApplet extends Applet {

   /* Structures used in GUI panel interface*/

   BrownianCurve B;         // Brownian motion curve
   TextField Iterations;    // Number of iterations to perform
   TextField IniVar;        // initial standard deviation


   /*   description of "init"
    *   parameters: none
    *   actions:    initializes applet in web page
    *   returns:    none
   **/
   public void init () {
      //Create user interface
      this.setBackground(Color.lightGray);
      Panel p = new Panel();
      add ("Usr", p);
      p.add(new Label("Iterations:"));
      Iterations=new TextField("10", 5);
      p.add(Iterations);
      p.add(new Label("Inital variance:"));
      IniVar=new TextField("150", 5);
      p.add(IniVar);
      p.add(new Button("New Sample"));
      resize(500,500);

      // Initializes Browninan motion sample
      B = new BrownianCurve();

      // awaits input from user to generate brownian sample
      start();
   }

    /*  description of "start"
     *  parameters: none
     *  actions:    Creates a new Brownian motion sample using the values
     *              in textfields
     *  returns:    none
    **/
    public void start() {
      B.Generate(getIter(), getVar());
    }

    /*  description of "action", boolean
     *  parameters: (Event, Object)
     *  actions:    handles the event of when button is pushed
     *  returns:    true if button is pushed.
    **/
   public boolean action (Event evt, Object arg) {
      if ("New Sample".equals(arg)) {
         start();
         repaint();
         return true;
      }
```

```
   return false;
}

 /*  description of "paint"
  *  parameters: g - image in panel to draw
  *  actions:    Draws the current Brownian sample whenever neccesery
  *  returns:    none
 **/
public void paint(Graphics g) {

     /* The start and end points in a
        line segments (between two sample values)  */

     int i, x1, x2, y1, y2;
     double maxval, minval, facw, fach;

     /* Responsible for canvas where curve is drawn */
     int x, y, h, w;
     x=0;y=50;w=getSize().width-1;h=getSize().height - 51;


     g.drawLine(0,h/2+51,w,h/2+51);
     g.setColor(Color.blue);


    // Locate maximum and minimum values of sample
    maxval=B.Data[getIter()]; minval=B.Data[0];
    for (i=1; i < B.Data.length; i++) {
       if (maxval < B.Data[i]) maxval=B.Data[i];
       if (minval > B.Data[i]) minval=B.Data[i];
    }

    //Calculate scaling factors
    facw=new Double(w).doubleValue()/
         new Double(B.Data.length).doubleValue();
    fach=new Double(h).doubleValue()/
         new Double(Math.abs(maxval-minval)).doubleValue();

    //Rescale and draw
    x1=0; y1=0;
    for (i=1; i < B.Data.length; i++) {
       x2=x + new Double(facw*i).intValue();
       y2=y+h/2-new Double(facw*B.Data[i]).intValue();
      if (i > 1) g.drawLine(x1,y1,x2,y2);
       x1=x2; y1=y2;
    }

}

 /*  description of "getIter"
  *  parameters: none
  *  actions:    read integer value from text field
  *  returns:    the value of text field
 **/
public int getIter() {
   return new Integer(Iterations.getText()).intValue();
}


 /*  description of "getVar"
  *  parameters: none
  *  actions:    reads the variance values in text field
  *  returns:    the value of the text field
 **/
public double getVar() {
   return new Double(IniVar.getText()).doubleValue();
}

 /*  description of "setIter"
  *  parameters: v - the integer in the text field (int)
  *  actions:    sets the text field to entered integer value
```

```
     *  returns:    none
    **/
   public void setIter(int v) {
      Iterations.setText(new String().valueOf(v));
   }


    /*  description of "setVar"
     *  parameters: v - the integer in the text field (double)
     *  actions:    sets the text field to entered integer value
     *  returns:    none
    **/
   public void setVar(double v) {
      IniVar.setText(new String().valueOf(v));
   }


}
```

```
/*  PROGRAM:        Brownian Motion Applet
 *  FILE:           BrownianCurve.java
 *  AUTHOR:         Matthew Moore
 *  DESCRIPTION:    Interactive programs that creates a Brownian Curve
 *                  representing random movement over time.
 *  ENVIRONMENT:    JBuilder 3, Java v.1.3.1
 *  NOTES:          Adapted from code found at web site
 *                  http://hobbe.lub.lu.se/~hakan/papers/fract/fract/node12.html
**/

import java.util.Random;
import java.awt.Graphics;

public class BrownianCurve {
   static double Data[];  /* Array to store sample */
   static double delta[]; /* Array to hold new variances */
   static double Var;     /* initial variance */

   Random rand = new Random();  /* Random number generator*/


    /*  description of "BrownianCurve", constructor
     *  parameters: none
     *  actions:    none
     *  returns:    none
    **/
   public BrownianCurve() {
      Generate(1, 1);
   }

    /*  description of "Generate"
     *  parameters: n - the number of iterations to perform (int)
     *              v - the initial variance (double)
     *  actions:    calculates the new variances and initializes the sample
     *  returns:    none
    **/
   public void Generate(int n, double v) {
       int maxpos;  // Number of samples to generate
       Double d;
       Var=v;       // Gets the initial variance

       d = ( new Double (Math.pow(2,n)) );
       maxpos=d.intValue();

       delta = new double[maxpos+1];
       /* initializes array for new variances */

       for(int i = 1; i<n; i++){
            delta[i] = Var*Math.pow(0.5, (i+1)/2);
       }
       /* Creates variances for sample data */


       Data= new double[maxpos+1];
       Data[0]=0;
       Data[maxpos]=Var*rand.nextGaussian();
       /* initializes array for sample */

       Divide(Data, 0, maxpos, 1, n);
       /* Split and displace midpoint n times */

   }

    /*  description of "Divide"
     *  parameters: X[] - array to hold sample data (double)
     *              a - starting index of sample (int)
     *              b - end index of sample (int)
     *              level - depth of iteration (int)
     *              n - maximum number of iterations (int)
     *  actions:    Creates a new Brownian motion sample
     *  returns:    none
```

```
    **/
    private void Divide(double X[], int a, int b, int level, int n) {
       int c;  //Midpoint position in Data

       c=(a+b)/2;
       X[c]=(X[a]+X[b])/2 + delta[level]*rand.nextGaussian();
       // Midpoint gets displaced

       if (level < n) {
          Divide(X, a, c, level+1, n);
          Divide(X, c, b, level+1, n);
       }
   }/*Divide*/


}/*Brownian Curve*/
```

```
/*  PROGRAM:        fractional Brownian Motion Applet
 *  FILE:           fBrownianApplet.java
 *  AUTHOR:         Matthew Moore
 *  DESCRIPTION:    Interactive programs that creates a Brownian Curve
 *                  representing random movement over time.
 *  ENVIRONMENT:    JBuilder 3, Java v.1.3.1
 *  NOTES:          Adapted from code found at web site
 *                  http://hobbe.lub.lu.se/~hakan/papers/fract/fract/node12.html
**/

import java.awt.*;
import java.awt.Graphics;
import java.applet.*;
import fBrownianCurve;

public class fBrownianApplet extends Applet {

   /* Structures used in GUI panel interface*/

   fBrownianCurve B;        // Brownian motion curve
   TextField Iterations;    // Number of iterations to perform

   TextField IniVar;        // initial standard deviation
   TextField Hurst;         // Hurst exponent

   /*   description of "init"
    *   parameters: none
    *   actions:    initializes applet in web page
    *   returns:    none
   **/
   public void init () {
      //Create user interface
      this.setBackground(Color.lightGray);
      Panel p = new Panel();
      add ("Usr", p);

      p.add(new Label("Iterations:"));
      Iterations=new TextField("10", 2);
      p.add(Iterations);

      p.add(new Label("Inital variance: "));
      IniVar=new TextField("150", 2 );
      p.add(IniVar);


      p.add(new Label("H: "));
      Hurst=new TextField("0.5", 2);
      p.add(Hurst);


      p.add(new Button("New Sample"));
      resize(500,500);

      //Create initial Browninan motion sample
      B = new fBrownianCurve();

      //awaits input from user to generate brownian sample
      start();
   }

    /*  description of "start"
     *  parameters: none
     *  actions:    Creates a new Brownian motion sample using the values
     *              in textfields
     *  returns:    none
    **/
   public void start() {
      B.Generate(getIter(), getVar(), getHurst());
   }
```

```
 /*  description of "action", boolean
  *  parameters: (Event, Object)
  *  actions:    handles the event of when button is pushed
  *  returns:    true if button is pushed.
 **/
public boolean action (Event evt, Object arg) {
   if ("New Sample".equals(arg)) {
      start();
      repaint();
      return true;
   }
   return false;
}

 /*  description of "paint"
  *  parameters: g - image in panel to draw
  *  actions:    Draws the current Brownian sample whenever neccesery
  *  returns:    none
 **/
public void paint(Graphics g) {

     int i, x1, x2, y1, y2;
     double maxval, minval, facw, fach;

     int x, y, h, w;
     x=0;y=50;w=getSize().width-1;h=getSize().height - 51;


     g.drawLine(0,h/2+51,w,h/2+51);
     g.setColor(Color.blue);


    // Locate maximum and minimum valus of the Browninan motion sample
    maxval=B.Data[getIter()]; minval=B.Data[0];
    for (i=1; i < B.Data.length; i++) {
       if (maxval < B.Data[i]) maxval=B.Data[i];
       if (minval > B.Data[i]) minval=B.Data[i];
    }

    //Calculate scaling factors
    facw=new Double(w).doubleValue()/
         new Double(B.Data.length).doubleValue();
    fach=new Double(h).doubleValue()/
         new Double(Math.abs(maxval-minval)).doubleValue();

    //Rescale and draw
    x1=0; y1=0;
    for (i=1; i < B.Data.length; i++) {
       x2=x + new Double(facw*i).intValue();
       y2=y+h/2-new Double(facw*B.Data[i]).intValue();
      if (i > 1) g.drawLine(x1,y1,x2,y2);
       x1=x2; y1=y2;
    }
}

 /*  description of "getIter"
  *  parameters: none
  *  actions:    read integer value from text field
  *  returns:    the value of text field
 **/
public int getIter() {
   return new Integer(Iterations.getText()).intValue();
}

 /*  description of "getVar"
  *  parameters: none
  *  actions:    read integer value from text field
  *  returns:    the value of text field
 **/
public double getVar() {
   return new Double(IniVar.getText()).doubleValue();
```

```
   }

    /*  description of "getHurst"
     *  parameters: none
     *  actions:    read integer value from text field
     *  returns:    the value of text field
    **/
   public double getHurst() {
      return new Double(Hurst.getText()).doubleValue();
   }


    /*  description of "setIter"
     *  parameters: v - the integer in the text field (int)
     *  actions:    sets the text field to entered integer value
     *  returns:    none
    **/
   public void setIter(int v) {
      Iterations.setText(new String().valueOf(v));
   }

    /*  description of "setVar"
     *  parameters: v - the integer in the text field (double)
     *  actions:    sets the text field to entered integer value
     *  returns:    none
    **/
   public void setVar(double v) {
      IniVar.setText(new String().valueOf(v));
   }

    /*  description of "setHurst"
     *  parameters: v - the integer in the text field (int)
     *  actions:    sets the text field to entered integer value
     *  returns:    none
    **/
   public void setHurst(double v) {
      Hurst.setText(new String().valueOf(v));
   }


}/* fBrownianApplet */
```

```
/*  PROGRAM:        fractional Brownian Motion Applet
 *  FILE:           fBrownianApplet.java
 *  AUTHOR:         Matthew Moore
 *  DESCRIPTION:    Interactive programs that creates a Brownian Curve
 *                  representing random movement over time.
 *  ENVIRONMENT:    JBuilder 3, Java v.1.3.1
 *  NOTES:          Adapted from code found at web site
 *                  http://hobbe.lub.lu.se/~hakan/papers/fract/fract/node12.html
**/

import java.util.Random;
import java.awt.Graphics;

public class fBrownianCurve {
   static double Data[];  // Array to store Brwinian motion sample
   static double delta[]; // Array to hold new variances
   static double Var;     // sample variance
   static double H;       // Hurst exponent

   Random rand = new Random();  // Random number generator

    /*  description of "fBrownianCurve", constructor
     *  parameters: none
     *  actions:    none
     *  returns:    none
    **/
   public fBrownianCurve() {
      Generate(1, 1, 0.5);
   }

    /*  description of "Generate"
     *  parameters: n - the number of iterations to perform (int)
     *              v - the initial variance (double)
     *              h - the initial value of the hurst exponent
     *  actions:    calculates the new variances and initializes the sample
     *  returns:    none
    **/
   public void Generate(int n, double v, double h) {
       int maxpos;  // Number of samples to generate
       Double d;
       Var=v;       // Gets the initial variance
       H = h;

       d = ( new Double (Math.pow(2,n)) );
       maxpos=d.intValue();

       delta = new double[maxpos+1];
       /* initializes array for new variances */

       for(int i = 1; i<n; i++){
            delta[i] = Var*Math.pow(0.5, i*H)*Math.sqrt(1 - Math.pow(2,2*H-2));
       }
       /* Creates variances for sample data */


       Data= new double[maxpos+1];
       Data[0]=0;
       Data[maxpos]=Var*rand.nextGaussian();
       /* initializes array for sample */


       Divide(Data, 0, maxpos, 1, n);
       /* Split and displace midpoint n times */

   }


    /*  description of "Divide"
     *  parameters: X[] - array to hold sample data (double)
     *              a - starting index of sample (int)
     *              b - end index of sample (int)
```

```
     *              level - depth of iteration (int)
     *              n - maximum number of iterations (int)
     *  actions:    Creates a new Brownian motion sample
     *  returns:    none
    **/
    private void Divide(double X[], int a, int b, int level, int n) {
       int c;  //Midpoint position in Data

       c=(a+b)/2;
       X[c]=(X[a]+X[b])/2 + delta[level]*rand.nextGaussian();
       // Midpoint gets displaced

       if (level < n) {
          Divide(X, a, c, level+1, n);
          Divide(X, c, b, level+1, n);
       }
   }/*Divide*/


}/*fBrownian Curve*/
```

REFERENCES

[1] "Brownian Motion – The Research Goes On." Online. http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol4/yk1/report.html. April 19, 2002.

[2] "The Koch Curve."Online. http://www.jimloy.com/fractals/koch.htm. April 21, 2002.

[3] Zhou, Mai. "Two and one dimentional Brownian motion." Online. http://www.ms.uky.edu/~mai/java/stat/brmo.html.

[4] "CS184 Lecture 25 Summary." Online. http://www.cs.berkeley.edu/~jfc/cs184f98/lec25/lec25.html. March 21 2002.

[5] Ardo, Hakan. "Fractal Landscapes." Online. http://hobbe.lub.lu.se/~hakan/papers/fract/fract/node12.html. March 19 2002.

[6] "Fractional Brownian Motion." Online. http://classes.yale.edu/math190a/Fractals/RandFrac/fBm/fBm.html. April 21, 2002.