Random Walks on Fractals [*]

J. Marcus Hughes [†]

May 20, 2018

1 Introduction

Imagine a raindrop falling from the sky. It wanders as it falls, ever attracted to the ground by gravity. This simple behavior can be modeled as a three-dimensional random walk with an attracting plane. Even simpler, consider particles in a liquid. Their motion is seemingly random on short time scales as they are attracted by near-by particle and repelled by others. Numerous physical phenomena can be modeled with random walks of various kinds [5, 6]. Random walks exhibit structure in how they traverse space though, structure that can be characterized using a kind of fractal dimension.

Random walk theory is a well explored and old discipline. The name "random walk" is thought to originate with Pearson in 1901 regarding a game of golf [1]. Pearson presented it in Nature as a question asking for help:

> A man starts from a point 0 and walks 1 yards in a straight line; he then turns through any angle whatever and walks another 1 yards in a second straight line. He repeats this process n times. I require the probability that after these n stretches he is at a distance between r and $r + dr$ from his starting point 0. The problem is one of considerable interest, but I have only succeeded in obtaining an integrated solution for two stretches. I think, however, that a solution ought to be found, if only in the form of a series in powers of $1/n$, when n is large

[*] written as a final project for Math 306: Fractals and Chaos Theory at Williams College taught by Cesar Silva

[†] hughes.jmb@gmail.com

Lord Rayleigh had explored this phenomena for gases in the 1880s [3] and responded with to Pearson that the probability of traveling a distance between R and $R + dR$ in N steps as $N \to \infty$ was [4]:

$$P_N(R) \sim \frac{2R}{N} e^{-R^2/N} \tag{1}$$

This began a long study of random walks which would take volumes to completely account. Applications range from statistical physics (also the continuous version of Brownian motion [2]) to search engine algorithms. Despite the long history of study, there remain open questions concerning specific kinds of random walks.

For this paper, I will present a random walk contextualized as a Markov chain. The random walk is an iterated process where the next step only depends on the current location. Random walks occur in some space $\mathbb{X}$ which could be finite, such as on a finite graph, or infinite, such as $\mathbb{R}$. The walker is at some location $p \in \mathbb{X}$, such as (x, y) in the lattice $\mathbb{Z} \times \mathbb{Z}$. It can then transition to any of its neighbor states $N(p) \subset \mathbb{X}$. In the lattice, the $N((x, y)) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$. In the case of the simple random walk, the probabilitiy of attaining any of these subsequent states is equal. However, this does not have to be the case. Let $P(p, p')$ be a function $\mathbb{X} \times \mathbb{X} \to [0, 1]$ denoting the probability of transitioning from p to p'. Then given $p \in \mathbb{X}$, it holds $\sum_{p' \in \mathbb{X}} P(p, p') = 1$. This framework is general enough to admit several interesting applications and questions.

2 Simple random walks

We will begin by considering the simple random walk on the regular lattice in k dimensions and analyzing the fractal dimensino of it. There are many ways to consider the dimension of a geometric object. The topological dimension is the most common in grade school and has applications in linear algebra and other mathematics. It is always a natural number and is defined recursively. A set A has topological dimension of zero if every point $p \in A$ has neighborhoods of arbitrarily small length whose boundary misses A. An example is a finite set of points $\{1, 2, 3, 4\} \subset \mathbb{Z}$. A set A has topological dimension of one if is not of topological dimension 0 and every point in A has arbitrarily small neighborhoods whose boundary meets A in a set of topological dimension zero. The definition for topological dimension is recursively defined from there. This dimension does not admit the rough structure in sets we are examining, e.g. Cantor's set is the same topological dimension as a finite set.

The Hausdorff dimension admits more structure. The Hausdorff dimension is s such for

$$\lambda^t(A) = \inf\{\sum_{j=1}^{\infty} |I_j|^t : A \subset \cup_{j=1}^{\infty} I_j\}$$

it holds that $\lambda^s(A) = \infty$ for $s < t$ and $\lambda^s(A) = 0$ for $s > t$. It captures the fractal nature but is difficult to use in practice. Thus, we tend to consider the Minkowski dimension which can be extracted using the box-counting algorithm. For a compact set A and $\epsilon >, 0$, let $N(A, \epsilon)$ be the smallest number of ϵ-boxes needed to cover A. Then the Minkowski dimension is:

$$d = \lim_{\epsilon \to 0} -\frac{\log N(A, \epsilon}{\log \epsilon}$$

when the limits exists. This definition works quite well for interesting sets such as the Sierpisnki triangle. However, for the random walk on the lattice it is insufficient because when $N(A, \epsilon) < n$ where n is the number of steps in the random walk for $\epsilon < 1$ because each lattice point visited is matched to one box.

Therefore, I will follow the example of Saberi (2011) and use a different measure of the fractal dimension [7]. Thus, the dimension d of a random walk is $M \sim R^d$ where M is the number of unique lattice points visited and R is the radius of gyration for a random walk. The radius of gyration R is defined in terms of the center of mass r_{cm}:

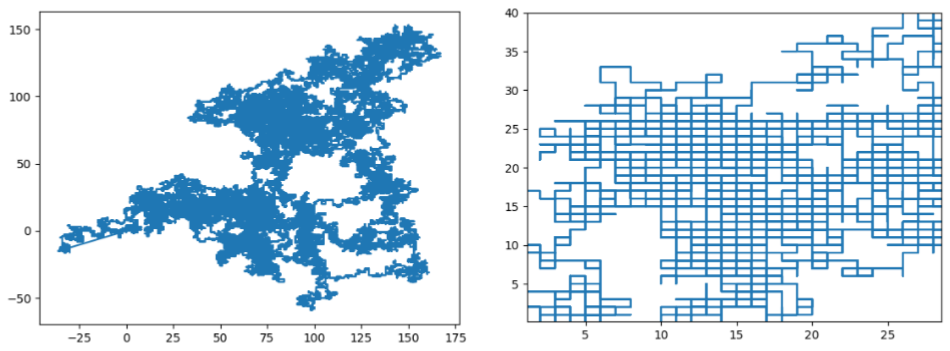
$$r_{cm} = \frac{1}{n+1} \sum_{i=0}^{n} r_i \tag{2}$$

. Here, n is the number of steps in the walk and r_i is the location at the i-th step. From this, we derive R:

$$R = \sqrt{\frac{1}{N+1}\sum_{i=0}^{n}\langle(r_{cm} - r_i)^2\rangle} \tag{3}$$

. In this case, $\langle r_i \rangle$ is the average of the entries of r_i. Thus, the dimension can be found by fitting a line to a log-log plot of M and R.

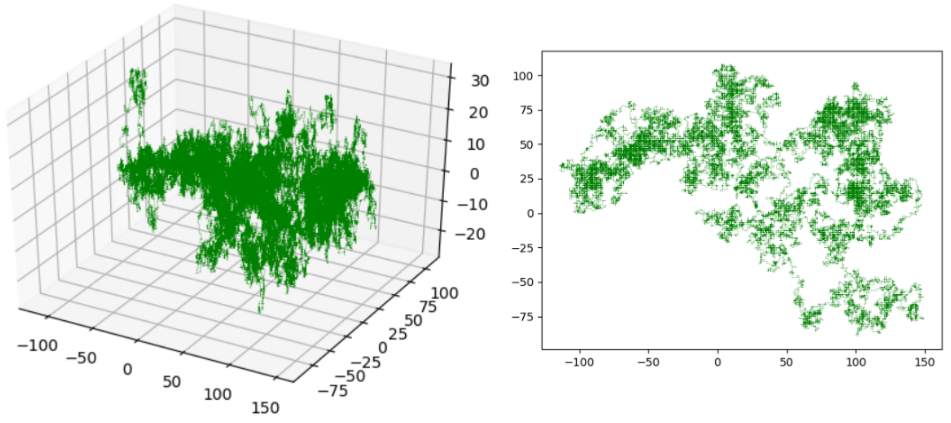
The figure below shows an example of a simple random walk in two dimensions:



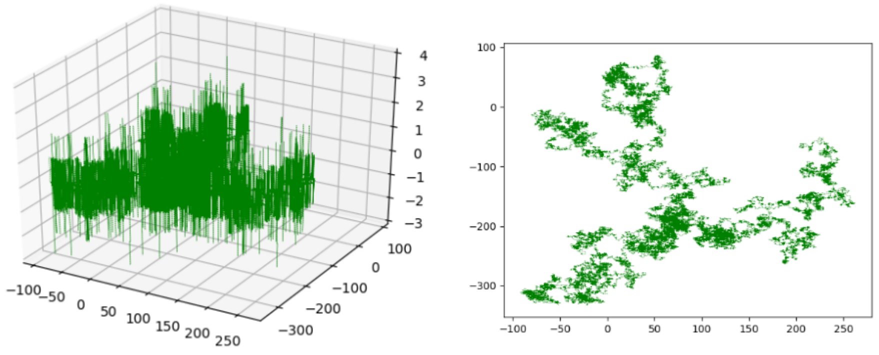
The image at left shows the full walk with 55574 steps while the right zooms in to reveal the underlying lattice structure. The fractal dimension of this motion is rather uninteresting. For $d = 1$, the Hausdorff dimension is known to be 3/2 while it is 2 for $d \geq 2$ [7].

3 Attracted random walkks

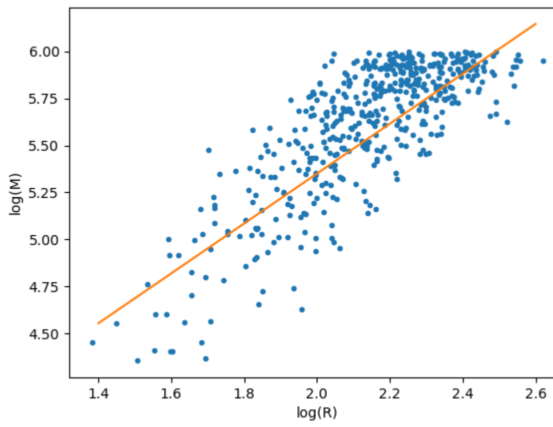
Saberi (2011) propose an attracted random walk where the walker is predisposed attracted, with some attraction coefficient α, toward the xy-plane from a three-dimensional lattice. At a lattice point (x, y, z) the walker has six choices for movement: up, down, left, right, forward, backwards. Let $p = \frac{1}{\alpha+5}$ and $q = \frac{1}{4\alpha+2}$. When the walker is not on the xy-plane, i.e. $z \neq 0$, the walker has probability αq to move toward the plane and p for all other directions. When the walker is on the xy-plane, i.e. $z = 0$, the walker will move left, right, forwards, and backwards with probability of αq and up or down (thus moving off the plane) with probability of q. When, $\alpha = 1$ this is a pure three-dimensional random walk but when $\alpha \to \infty$ this becomes a two-dimensional random walk on the plane. The walker begins at the origin.



Above is an example of this attracted random walk with $\alpha = 1.3$ and 10^4 steps. At left, you can see a side view in three dimensions and at right is the view directly from the top. This is lightly attracted. Below, you can see the result with $\alpha = 10$.



The walker can only get a few steps off the plane before it is pulled back onto it. Most of the interesting behavior occurs in this range of $1 < \alpha < 5$. However, by looking at very high α we can determine the limiting case. Saberi (2011) found this to be 1.83 [7]. However, I could not replicate this result.



My fit line instead had slope of 1.3. I did not have sufficient time to determine why this was different.

4 Code Contribution

Since this project was computational, I will briefly outline my original code contributions. I have developed two independent software packages in Java: jifs and jWalker . These were kept separate as jifs is being developed as a portable Java version of TeraFractal, a beautiful fractal generation tool exclusively for Mac. jifs provides iterated function systems in Java and measuring the Minkowski dimension while jWalker handles the random walks portion. This report will be made available with the jWalker code.

4.1 jifs

4.2 jWalker

5 Summary

References

[1] Carazza, B. The history of the random-walk problem: Considerations on the interdisciplinarity in modern physics. *Rivista Del Nuovo Cimento 7*, 3 (1977), 419–27.

[2] Einstein, A. Über die von der molekularkinetischen theorie der wrme geforderte bewegung von in ruhenden flssigkeiten suspendierten teilchen. *Annalen der physik 322*, 8, 549–560.

[3] F.R.S., L. R. Xii. on the resultant of a large number of vibrations of the same pitch and of arbitrary phase. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 10*, 60 (1880), 73–78.

[4] F.R.S., L. R. O. Xxxi. on the problem of random vibrations, and of random flights in one, two, or three dimensions. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 37*, 220 (1919), 321–347.

[5] Haw, M. Einstein's random walk. *Physics World 18*, 1 (2005), 19.

[6] Mandelbrot, B. B. *The fractal geometry of nature.* W. H. Freeman, New York, 1983.

[7] Saberi, A. A. Fractal structure of a three-dimensional brownian motion on an attractive plane. *Phys. Rev. E 84* (Aug 2011), 021113.

Appendix A Code

A.1 Point

```java
import java.util.Vector;
import java.lang.Double;

/**
 * A representation of a point in Euclidean space.
 * @author J. Marcus Hughes
 */
public class Point {
    private Vector<Double> coord;
    private int dimension;

    /** Build a point from arbitrarily many double values */
    Point(Double...  values) {
        this.dimension = values.length;
        this.coord = new Vector<Double>();
        for (int i = 0; i < this.dimension; i++) {
            coord.addElement(values[i]);
        }
    }

    /** Built a point from arbitrarily many integer values */
    Point(Integer...  values) {
        this.dimension = values.length;
        this.coord = new Vector<Double>();
        for (int i = 0; i < this.dimension; i++) {
            coord.addElement((double) values[i]);
        }
    }

    /** Built a point from a <code> Vector </code> of coordinate values */
    Point(Vector<Double> values) {
        this.coord = values;
        this.dimension = values.size();
    }

    /**
     * Number of coordinates in point
     * <p>
     * For example, (0,0) is dimension 2 and (0,0,0) is dimension 3
     * </p>
     */
    public int dimension() {
        return coord.size();
    }

    /** Returns the point as a vector of doubles */
    public Vector<Double> getCoord() {
        return this.coord;
    }

    /**
     * Create a string version of a point
     * <p>
     * For example (3, 4, 5)
     * </p>
     */
    public String toString() {
        String s = "(";
        for (Double v : coord) {
            s += v;
            s += ", ";
        }
        s = s.substring(0, s.length()-2);
        s += ")";
        return s;
    }

    /**
     * Add a point to another point elementwise
     * @param other a point with the same dimension
     * @throws RuntimeException when points are not of the same dimension
     * @return the elementwise sum of points
     */
    public Point add(Point other) {
        if (other.dimension() != this.dimension()) {
            throw new RuntimeException("Points must have same dimension to add");
        }
        Vector<Double> result = new Vector<Double>();
        for (int i = 0; i < this.dimension; i++) {
            result.addElement(this.coord.get(i) + other.coord.get(i));
        }
        return new Point(result);
    }

    /**
     * Scales a point by a scalar double
     * @param k number to scale by
     */
    public Point scale(double k) {
        Vector<Double> result = new Vector<Double>();
        for (int i = 0; i < this.dimension; i++) {
            result.addElement(this.coord.get(i) * k);
        }
        return new Point(result);
    }


    /**
     * Subtract other point
     * @param other point to subtract
     * @throws RuntimeException when points are not of the same dimension
     * @return this - other
     */
    public Point subtract(Point other) {
        return this.add(other.scale(-1.0));
    }

    /**
     * Multiply points
     * @param other point to multiple
     * @throws RuntimeException when points are not of the same dimension
     * @return this * other
     */
    public Point multiply(Point other) {
        if (other.dimension() != this.dimension()) {
            throw new RuntimeException("Points must have same dimension to add");
        }
        Vector<Double> result = new Vector<Double>();
        for (int i = 0; i < this.dimension; i++) {
            result.addElement(this.coord.get(i) * other.coord.get(i));
        }
        return new Point(result);
    }

    /** Squares */
    public Point square() {
        return this.multiply(this);
    }

    public double mean() {
        double sum = 0.0;
        for (double v : coord) {
            sum += v;
        }
        return sum / dimension;
    }

    /** testing method */
    public static void main(String[] args) {
        System.out.println("Testing for point");
        Point p = new Point(1, 2, 3);
        System.out.println(p);
    }
}
```

A.2 RandomWalk

```
import java.util.Vector;
import java.util.Set;
import java.util.HashSet;
import static java.lang.Math.sqrt;

/**
 * A generic representation of a random walk
 * @author J. Marcus Hughes
 */
public abstract class RandomWalk {
    int dimension; // the number of coordinates in a point, e.g. (0,0) is dimension 2
    Vector<Point> path; // the collection of points traveled by the random walk

    /**
     * A generic random walk starting from <code> source </code>
     * @param source where the random walk begins from
     */
    RandomWalk(Point source) {
        this.path = new Vector<Point>();
        this.path.addElement(source);
        this.dimension = source.dimension();
    }

    /** Take one step, i.e. move one iteration */
    abstract public void step();

    /**
     * Take <code> steps </code> and return the path
     * @param steps number of movements to make
     * @return path of random walk
     */
    public Vector<Point> walk(int steps) {
        for (int i = 0; i < steps; i++) {
            step();
        }
        return this.path;
    }

    /** Return path */
    public Vector<Point> getPath() {
        return this.path;
    }

    /** Number of points in path */
    public int length() {
        return this.path.size();
    }

    /**
     * Calculates the center of mass
     * @throws RuntimeException if the path is empty
     */
    public Point centerOfMass() {
        if (this.length() == 0) {
            throw new RuntimeException("Path must be populated before calculating");
        }
        Vector<Double> point = new Vector<Double>();
        for (int i = 0; i < this.dimension; i++) {
            point.addElement(0.0);
        }
        Point sum = new Point(point);
        for (Point p : path) {
            sum = sum.add(p);
        }
        return sum.scale(1.0 / (1.0 + length()));
    }

    /**
     * Calculates the radius of gyration
     * @throws RuntimeException if the path is empty
     */
    public double radiusOfGyration() {
        Point center = centerOfMass();
        double sum = 0.0;
        for (Point p : path) {
            sum += p.subtract(center).square().mean();
        }
        return sqrt((1.0 / (length() + 1.0)) * sum);
    }

    /**
     * Number of unique points
     */
    public int numUnique() {
        Set<Point> s = new HashSet<Point>(path);
        return s.size();
    }
}
```

A.3 PlaneAttracted3D

```
import java.util.Vector;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;

public class PlaneAttracted3D extends RandomWalk {

    private double alpha;
    private Random rng; // random number generator
    private double p; // probability of moving when on plane
    private double q; // probability of moving when off plane, p' in paper

    /**
     * Only constructor for <code> PlaneAttracted3D </code>
     * @param alpha attraction parameter
     */
    PlaneAttracted3D(double alpha) {
        super(new Point(0,0,0));
        this.rng = new Random();

        // attraction to plane
        this.alpha = alpha;
        if (this.alpha < 1.0) {
            throw new RuntimeException("alpha must be greater than 1");
        }

        // parameters for movement
        this.p = 1.0 / (alpha + 5.0);
        this.q = 1.0 / (4.0 * alpha + 2.0);
```

```
}

/** Determine where to move next when on plane */
private Point onPlane() {
    Point prev = this.path.lastElement();
    float p = this.rng.nextFloat(); // random number

    // determine next step based on probability and selection
    // on sum compared to generated p
    if (p < this.q) {
        return prev.add(new Point(0, 0, -1)); // move off plane
    } else if (p < 2.0 * this.q) {
        return prev.add(new Point(0, 0, 1));  // move off plane
    } else if (p < (2.0 + this.alpha) * this.q) {
        return prev.add(new Point(1, 0, 0));  // move in plane
    } else if (p < (2.0 + 2.0 * this.alpha) * this.q) {
        return prev.add(new Point(-1, 0, 0)); // move in plane
    } else if (p < (2.0 + 3.0 * this.alpha) * this.q) {
        return prev.add(new Point(0, 1, 0));  // move in plane
    } else {
        return prev.add(new Point(0, -1, 0)); // move in plane
    }
}

/** Determine java documentationwhere to move next when off plane */
private Point offPlane() {
    Point prev = this.path.lastElement();
    float p = this.rng.nextFloat();

    // determine next step based on probability and selection
    // on sum compared to generated p
    if (p < this.p) {
        return prev.add(new Point(1, 0, 0));
    } else if(p < 2.0 * this.p) {
        return prev.add(new Point(-1, 0, 0));
    } else if(p < 3.0 * this.p) {
        return prev.add(new Point(0, 1, 0));
    } else if(p < 4.0 * this.p) {
        return prev.add(new Point(0, -1, 0));
    } else if(p < 5.0 * this.p) {
        if (prev.getCoord().lastElement() < 0) {  // z < 0
            return prev.add(new Point(0, 0, -1)); // move away from plane
        } else {                                  // z > 0
            return prev.add(new Point(0, 0, 1));  // move toward plane
        }
        } else {
            if (prev.getCoord().lastElement() < 0) {  // z < 0
                return prev.add(new Point(0, 0, 1));  // move toward plane
            } else {                                  // z > 0
                return prev.add(new Point(0, 0, -1)); // move away from plane
            }
        }
    }

    /** Take one step */
    public void step() {
        Point prev = this.path.lastElement();
        Integer z = prev.getCoord().lastElement().intValue();

        if (z == 0) { // on plane
            this.path.addElement(onPlane());
        } else {      // z != 0, so off plane
            this.path.addElement(offPlane());
        }
    }

    /** Testing method */
    public static void main(String[] args) {
        Random rng = new Random();
        int iterations = Integer.parseInt(args[0]);
        double alpha   = Double.parseDouble(args[1]);

        int iMin = 10000;
        int iMax = 1000000;
        int length = ThreadLocalRandom.current().nextInt(iMin, iMax);

        PlaneAttracted3D rw;

        for (int i = 0; i < iterations; i++) {
            length = ThreadLocalRandom.current().nextInt(iMin, iMax);
            rw = new PlaneAttracted3D(alpha);
            rw.walk(length);
            System.out.printf("%d %f \n", rw.numUnique(),  rw.radiusOfGyration());
        }
    }
}
```

A.4 Simple2D

```java
import java.util.Vector;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;

public class Simple2D extends RandomWalk {
    private Random rng;

    /**
     * Only constructor for <code> Simple3D </code>
     * @param alpha attraction parameter
     */
    Simple2D() {
        super(new Point(0,0));
        this.rng = new Random();
    }

    /** Take one step */
    public void step() {
        Point prev = this.path.lastElement();
        float p = this.rng.nextFloat();

        // determine next step based on probability and selection
        // on sum compared to generated p
        if (p < 1.0/4.0) {
            this.path.addElement(prev.add(new Point(1, 0)));
        } else if (p < 2.0/4.0) {
            this.path.addElement(prev.add(new Point(-1, 0)));
        } else if (p < 3.0/4.0) {
            this.path.addElement(prev.add(new Point(0, 1)));
        } else {
            this.path.addElement(prev.add(new Point(0, -1)));
        }
    }

    /** Testing method */
    public static void main(String[] args) {
        int iterations = Integer.parseInt(args[0]);

        int iMin = 10000;
        int iMax = 1000000;
        int length = ThreadLocalRandom.current().nextInt(iMin, iMax);

        Simple2D rw;
        for (int i = 0; i < iterations; i++) {
            length = ThreadLocalRandom.current().nextInt(iMin, iMax);
            rw = new Simple2D();
            Vector<Point> ww = rw.walk(length);
            // if (i == 0) {
            //     for(Point p : ww) {
            //         System.out.println(p);
            //     }
            //     System.out.println("-----");
            // }
            System.out.printf("%d %f \n", rw.numUnique(),  rw.radiusOfGyration());
        }
    }
}
```

A.5 Simple3D

```
import java.util.Vector;
import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;

public class Simple3D extends RandomWalk {
    private Random rng;

    /**
     * Only constructor for <code> Simple3D </code>
     * @param alpha attraction parameter
     */
    Simple3D() {
        super(new Point(0,0,0));
        this.rng = new Random();
    }

    /** Take one step */
    public void step() {
        Point prev = this.path.lastElement();
        float p = this.rng.nextFloat();

        // determine next step based on probability and selection
        // on sum compared to generated p
        if (p < 1.0/6.0) {
            this.path.addElement(prev.add(new Point(1, 0, 0)));
        } else if (p < 2.0/6.0) {
            this.path.addElement(prev.add(new Point(-1, 0, 0)));
        } else if (p < 3.0/6.0) {
            this.path.addElement(prev.add(new Point(0, 1, 0)));
        } else if (p < 4.0/6.0) {
            this.path.addElement(prev.add(new Point(0, -1, 0)));
        } else if (p < 5.0/6.0) {
            this.path.addElement(prev.add(new Point(0, 0, 1)));
        } else {
            this.path.addElement(prev.add(new Point(0, 0, -1)));
        }
    }

    /** Testing method */
    public static void main(String[] args) {
        int iterations = Integer.parseInt(args[0]);

        int iMin = 10000;
        int iMax = 1000000;
        int length = ThreadLocalRandom.current().nextInt(iMin, iMax);

        Simple3D rw;

        for (int i = 0; i < iterations; i++) {
            length = ThreadLocalRandom.current().nextInt(iMin, iMax);
            rw = new Simple3D();
            rw.walk(length);
            System.out.printf("%d %f \n", rw.numUnique(),  rw.radiusOfGyration());
        }
    }
}
```

A.6 plotFig1

```
import numpy as np
import matplotlib.pyplot as plt
import argparse

def get_args():
    ap = argparse.ArgumentParser()
    ap.add_argument("path", help="path to text file to open")
    return vars(ap.parse_args())

if __name__ == "__main__":
    args = get_args()
    dat = np.loadtxt(args['path'])
    m3d, rg = dat[:,0], dat[:,1]
```

```
fig, ax = plt.subplots()
ax.plot(np.log10(rg),
        np.log10(m3d),
        ".")
plt.show()
```