Random Walks on Fractals [*]

J. Marcus Hughes [†]

May 18, 2018

1 Introduction

Imagine a leaf falling from a tree. 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. Numerous physical phenomena can be modeled with random walks of various kinds **enumerate**. These random walks are self-similar **why?** and have various fractal dimensions.

2 Background

2.1 Simple random walks

Mandelbrot originally studied random walks as fractals.

2.2 Fractal dimension

The dimension of a geometric object can be interpreted many ways:

- topological dimension
- Hausdorff dimension
- Minkowski dimension

3 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.

3.1 jifs

3.2 jWalker

4 Experiments

5 Summary

References

Appendix A Code

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

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

†hughes.jmb@gmail.com

```java
/**
 * 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);
    }

    /** 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);
    }
}

import java.util.Vector;

/**
 * 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();

    /** Perform a complete walk with the desired number of steps */
    abstract public Vector<Point> walk(int steps);

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

import java.util.Vector;
import java.util.Random;

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());
        }
    }

    /**
     * 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;
    }

    /** Testing method */
    public static void main(String[] args) {
        PlaneAttracted3D walk = new PlaneAttracted3D(1.5);
        Vector<Point> path = walk.walk(100);
        for(Point p : path) {
            System.out.println(p);
        }
    }
}
```