PHYS 440: Random Walk Simulation Project #1

Due on Friday, February 6, 2015

Inst. Pavel Snopok, Spring 2015

Michael Lee

February 6, 2015

Abstract

A simple application of the Monte Carlo method is a random walk simulation. We create the random walk through producing a sequence where random numbers are generated and summed with the previous steps. Since spatial dimensions are linearly independent, we can generate points for x, y, z and then create a three dimensional random walk. Our interest in the random walk stems from its usefulness in modeling many physical processes that have random behavior such as the Brownian motion of molecules in a fluid and photons tunneling from a star's core to its surface. Matplotlib provides an robust animation module which we utilized to create a 3-d rendering of the random walk.

1 Introduction

A random walk is a process that uses sequences of random steps to plot a path. Although this behavior may seem trivial, they define processes that enable us to model various physical phenomenon. These models are so ubiquitous that they can found in the most disparate fields. Stock traders use random walks to forecast the fluctuating price of a stock while physicists use them to estimate the time it takes a photon to leave the center of a star. Chemists are interested in the Brownian motion exhibited by molecules while biologists use random walks to determine foraging patterns of wild animals. Random walks are employed in all these fields simply because they are useful and give us insight about stochastic processes which are impalpable to human beings. Our objective will simply be to write Python code that simulates a random walk since further in depth research would be required to accurately create simulations of the examples noted above.

2 Theory and Salient Details

Random Walk

Our random walk simulation has an *artificial walker* that takes sequential steps in a direction independent of its previous step. He takes N step in the Cartesian plane with the path:

$$(0,0,0), (\Delta x_1, \Delta y_1, \Delta z_1), \ ...$$

$$(\Delta x_2, \Delta y_2, \Delta z_2), (\Delta x_N, \Delta y_N, \Delta z_N)$$

Since the x, y, and z directions are independent of each other, we can find the radial distance R through the distance formula:

$$R = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}$$

$$= \sqrt{(\Delta x_1 + \Delta x_2 + ...)^2 + (\Delta y_1 + \Delta y_2 + ...)^2 + (\Delta z_1 + \Delta z_2 + ...)^2}$$

The particle travels in random directions so we can safely assume that the cross terms of the squares cancel over large numbers of random steps. That is,

$$(\Delta x_1 + \Delta x_2 + ... \ )^2 \approx \Delta x_1^2 + \Delta x_2^2 + ...$$

because the cross terms are

$$2\Delta x_1 \Delta x_2 + 2\Delta x_2 \Delta x_3 + 2\Delta x_3 \Delta x_4 + ...$$

and since each direction, positive and negative, is likely to occur, the total averages to 0.

Thus we obtain a value called the root mean square (RMS). This value is defined as the *magnitude of a varying quantity* and also defines standard deviation, a useful measure to understand.

$$R_{rms}^2 = \langle x_1^2 + ... + x_N^2 + y_1^2 + ... + y_N^2 + z_1^2 + ... + z_N^2 \rangle$$

$$R_{rms}^2 = \langle x_1^2 + y_1^2 + z_1^2 \rangle + ... + \langle x_N^2 + y_N^2 + z_N^2 \rangle$$

$$R_{rms}^2 = N \langle r_{rms}^2 \rangle \to r_{rms} = \frac{R_{rms}}{\sqrt{N}}$$

where r_{rms} is the root mean square step size.

As the number of steps gets very large, the average vector distance evens out.

$$\langle \vec{R}_{rms} \rangle = \langle x \rangle \vec{i} + \langle y \rangle \vec{j} + \langle z \rangle \vec{k} \ \approx 0$$

This makes sense since we expect the probabilities of the particle moving up, down, left, or right to be equal. This is to say that there is no preferential direction for the random walk.

The actual scalar distance from the origin that the particle has traveled is the square root of sum of the square of the step sizes.

$$R = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}$$

We see then that although the displacement vector averages to zero, the magnitude of that vector is non-zero, meaning it is unlikely that the walk will result back to the origin. In a probabilistic sense, it is more likely that the artificial walker will take a path leading away from the origin than to it. Summing all three Δx^2, Δy^2, and Δz^2 means that a positive number is more likely to occur than zero for all three directions.

3 Simulations

3.1 Random Walk Simulation

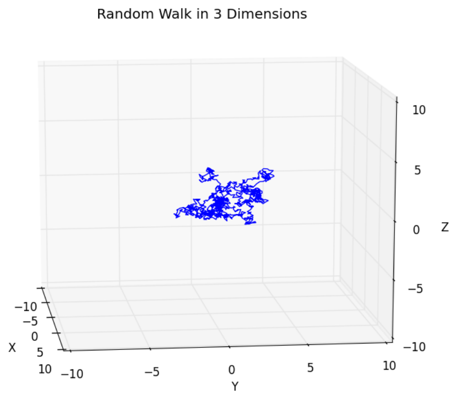

Figure 1

The main crux of the program is this randomwalk function definition that creates an empty multidimensional array, linedata, and populates it with sequences of step. The steps are created by adding a random number, supplied by the numpy library, to the step previous, all in three dimensions. Explicitly, *np.random.rand(dims)* generates three random numbers between zero and one (excluded) in an array and the step is added to the previous.

```
def randomwalk(length, dims=3):
    linedata = np.empty((dims, length))
    linedata[:,0] = np.random.rand(dims)
    for index in range(1, length):
        step = (np.random.rand(dims) \
            - 0.5)*.5
        linedata[:, index] =\
            linedata[:, index-1] + step
    return linedata
```

3.2 Range, Scalar Distance, and RMS

Our program calculates the range of x, y, and z, the scalar distance from the origin, and the RMS.

```
Ranges for plot 1 : X = -1.217054 to
5.682756;
Y = -2.770109 to 3.898981;
Z = -1.920242 to 3.405293
The scalar distance from the origin
for plot 1 is: 5.551387
The RMS (Quadratic mean) step size
for plot 1 is: .2525
```

The quadratic mean step size indicates that the average magnitude of the random steps is 0.2525. This means that the 68.26% of the steps have magnitudes between -0.2525 and 0.2525 and 95.4% have magnitudes between -0.505 and 0.505 or *two standard deviations*. This is indicative of the step variable in our randomwalk function where the maximum possible step is 0.866 from

$$\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2}$$

$$= \sqrt{(.25 + .25)^2 + (.25 + .25)^2 + (.25 + .25)^2}$$

$$= \sqrt{.75} = 0.866$$

As we increase the number of steps, the scalar distance from the origin of the artificial walker increases. This is the expected behavior as we claimed before; it is unlikely that it goes back to the origin. For the RMS, changing the number of steps has the only effect of making it converge to 0.25, as we should expect.

4 Conclusion

We were able to successful simulate random walks using Python and mathplotlib. We learned that the relationship between the scalar distance is almost linear to the number of steps the walker takes except for when the number of steps is low. This is because as the artificial walker moves away from the origin, it has less of a chance of moving in a direction that brings it back in the same direction. Simulating with a low number of steps brings us back to the idea of statistical significance and how a tiny amount of data is insufficient to generalize the behavior of a system. As the number of steps increase, we see that the quadratic mean of the step size converges to 0.25. This is exactly what we want since it means our experimental design satisfied the randomness needed for a random walk.

References

[1] R. Landau, Manual J. Paez, Cristian C. Bordeianu. *A Survey of Computational Physics*, 2010: Princeton University Press.

[2] D. Beazley and Brian K. Jones. *The Python Cookbook*, 2013: O'Reilly.

[3] Newman, M. E. J. *Computational Physics*, 2013: Createspace.

5 Appendix

Random Walk Simulation

```python
"""
Written by Michael Lee.
Adapted from Random Walk simulation by matplotlib.

PHYS440: Computational Physics
Project 1, Random Walk
"""

import numpy as np
import matplotlib
matplotlib.use('TKAgg') # For OSX users
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

def randomwalk(length, dims=3):
    linedata = np.empty((dims, length))
    linedata[:,0] = np.random.rand(dims)
    for index in range(1, length):
        step = (np.random.rand(dims) - 0.5)*.5
        linedata[:, index] = linedata[:, index-1] + step
    return linedata

def animate(num, datalines, lines):
    for line, data in zip(lines, datalines):
        line.set_data(data[0:2, :num])
        line.set_3d_properties(data[2, :num])
    return lines

# Index accounts for the number of random walk simulations in one trial
def walk_range(data, index, dims=3):
    xyzranges = []
    for i in range(index):
        for j in range(0, dims):
            max_range = max(data[i][j])
            min_range = min(data[i][j])
            xyzranges.append((min_range, max_range))
    return xyzranges

# RMS is the _average_ distance from the origin of all the
# steps. The actual scalar distance from the origin is calculated
# and given in the array R.
def distance_from_origin_and_rms(data, steps, index, dims=3):
    sum = 0
    rms_sum = 0
    rms = []
    RMS = []
    r = []
    R = []
    for i in range(index):
        for j in range(dims):
            for k in range(steps-1):
                sum = sum + (data[i][j][k+1] - data[i][j][k])
                rms_sum = rms_sum + (data[i][j][k+1] - data[i][j][k])**2
                r.append(sum)
                rms.append(rms_sum)
                sum = 0
                rms_sum = 0
    for l in range(len(r)/3):
        xyz = r[3*l]**2 + r[3*l+1]**2 + r[3*l+2]**2
        R.append(xyz)
        xyz_rms = (rms[3*l] + rms[3*l+1]+ rms[3*l+2])
        RMS.append(xyz_rms)
    return map(np.sqrt, R), RMS

if __name__ == '__main__':

    fig = plt.figure()
    ax = p3.Axes3D(fig, xlim=(-10,10), ylim = (-10, 10), zlim = (-10,10))
    ax.set_xlabel('X'), ax.set_ylabel('Y'), ax.set_zlabel('Z')
    ax.set_title('Random Walk in 3 Dimensions')

    steps = 1000
    number_of_walks = 1
    data = [randomwalk(steps) for index in range(number_of_walks)]
    lines = [ax.plot(dat[0, 0:1], dat[1, 0:1], dat[2, 0:1])[0] for dat in data]
    rang = walk_range(data, number_of_walks)
    distance = distance_from_origin_and_rms(data, steps, number_of_walks)

    for i in range(number_of_walks):
        print "Ranges for plot %d : X = %f to %f; Y = %f to %f; Z = %f to %f" %\
            (i+1, rang[3*i][0], rang[3*i][1], rang[(3*i)+1][0], rang[(3*i)+1][1],
            rang[(3*i)+2][0], rang[(3*i)+2][1])

    for i in range(number_of_walks):
        print "The scalar distance from the origin for plot %d is: %f" %\
            (i+1, distance[0][i])

        print "The RMS (Quadratic mean) step size for plot %d is: %f" %\
            (i+1, distance[1][i] / np.sqrt(steps))

    ani = animation.FuncAnimation(fig, animate, steps, fargs=(data, lines),
        interval=100, repeat=False)

    plt.show()
```