Scientific Computing Lab

Kevin Francis

April 1, 2021

Equations and Numerical Methods

In this lab we will be making a random walk program. To do this random coordinates in the x and y direction will be generated by the built in random function. Since the points should be uniform and in different directions they will be turned into unit vectors. Below x and y represent the random numbers.

$$L = \sqrt{(x^2) + y^2}, \qquad x' = \frac{x}{L}, \qquad y' = \frac{y}{L} \tag{1}$$

The x' and y' in (1) represent the new unit vectors. To analyze the walkers a plot will be made of the number of walkers vs. the root mean square of each walker. Doing this requires that we know the mean square distance for each walker, given by the equation below.

$$\left\langle R^2(N) \right\rangle = \frac{1}{K} \sum_{k=1}^{K} R^2_{(k)}(N) \tag{2}$$

The N and K in (2) represent the number of steps and walkers respectively. Those two variable also have the relationship of $K = \sqrt{N}$. To plot the plot mentioned above the square root of (2) must be taken.

Part 1

Write a function that generates random values for x and y.

Code

```
def randshift():
    r = (random() - 0.5)*2
    return r
```

Listing 1: Function for generating x and y

Part 2

Now the two vectors will be normalized. This is done by two functions, one that finds the magnitude, and another that normalizes x and y.

Code

```
def distance(x,y): #Since the distance is from the origin it is just the
    ending points
    V = sqrt((x*x) + (y*y))
    return V
```

Listing 2: Function for finding magnitude

```
def normalize():
    dx = randshift()
    dy = randshift()
    L = distance(dx,dy)
    x = dx/L #gives x a length of 1 in a random direction
    y = dy/L
    return x,y
```

Listing 3: Function for normalizing

Part 3

The next part is to make a function that takes the position of x and y and saves them into a list, and then returns that list.

Code

```
def walker2(N): #N is the number of steps
    xpos = zeros(N+1)
    ypos = zeros(N+1)
    x = 0
    y = 0
    for i in range (1,N+1):
        dx,dy = normalize()
        x += dx
        y += dy
        xpos[i] = x
        ypos[i] = y
    return xpos, ypos
```

Listing 4: Function for saving the position of each vector at each step

```
def walker3(N): #N is the number of steps
    xpos = zeros(N+1)
    ypos = zeros(N+1)
    zpos = zeros(N+1)
    x = 0
    y = 0
    z = 0
    for i in range (1,N+1):
        dx,dy,dz = normalize3D()
        x += dx
        y += dy
        z += dz
        xpos[i] = x
        ypos[i] = y
        zpos[i] = z
    return xpos, ypos, zpos
def walker2D():
```

Listing 5: Function for saving the position of each vector at each step in 3D

Visualization

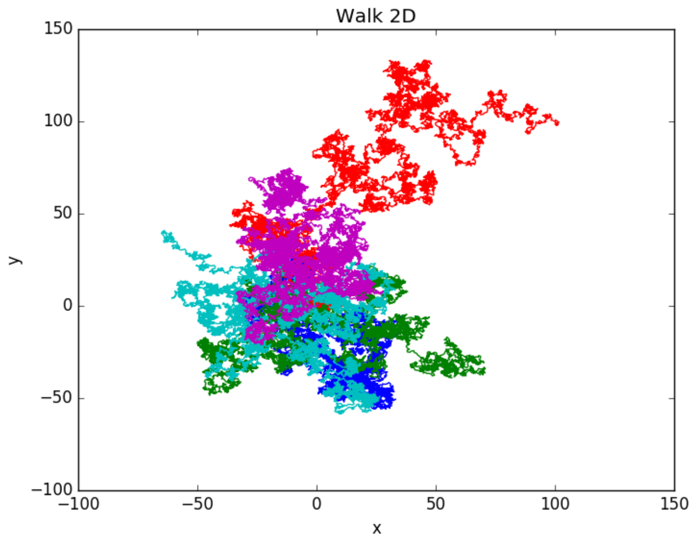

Figure 1: This plot represents 5 walkers with 10,000 steps each

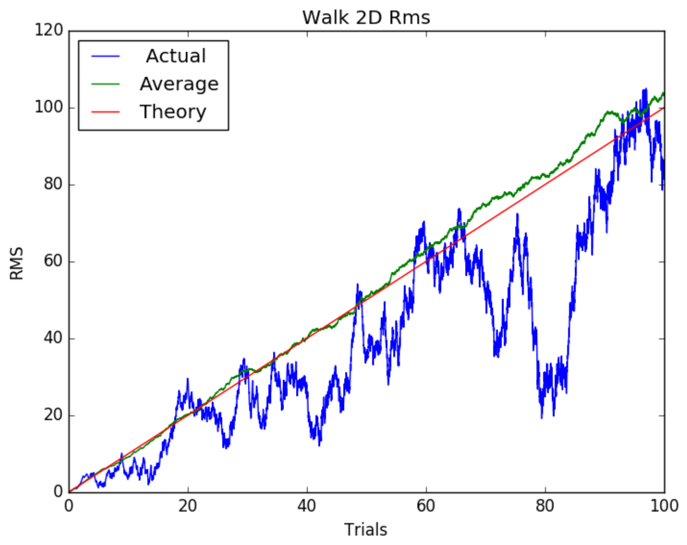

Figure 2: This plot shows the rms vs. number of walkers (trials)

```python
def walker2D():
        nwalkers = 100
        nsteps = nwalkers**2
        Rms = zeros(nsteps+1) #Root mean square and starts it a one
        steps = range(nsteps+1)
        for i in range(nwalkers):
            x, y = walker2(nsteps)
            if i < 5:
                figure(4)
                plot(x,y)
            d = distance(x,y)
            Rms += d**2/nwalkers #Adds every time
            if i == nwalkers/2:
                figure(3)
                plot(sqrt(steps),d,label=" Actual")
                legend(loc="best")
        figure(4)
        xlabel ("x")
        ylabel("y")
        title("Walk 2D")
        Rms = sqrt(Rms)
        figure(3)
        xlabel ("Trials")
        ylabel("RMS")
        title("Walk 2D Rms")
        plot(sqrt(steps),Rms,label="Average")
        plot(sqrt(steps),sqrt(steps),label="Theory")
        legend(loc="best")
```

Listing 6: Function that plots figure 1 and 2

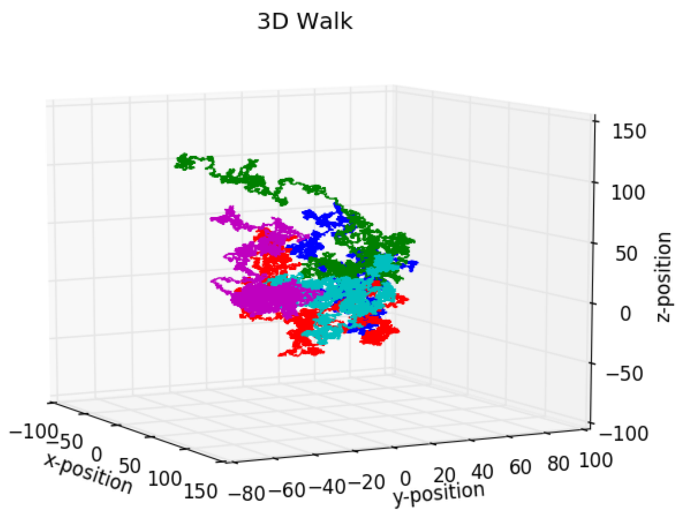

Figure 3: This plot shows 5 walkers in 3D

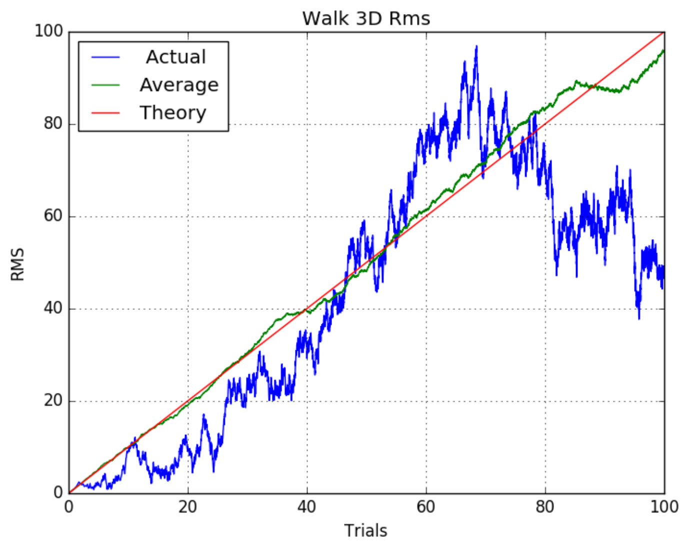

Figure 4: This plot shows the rms vs number of walkers for 3D

```
def walker3D():
        nwalkers = 100
        nsteps = nwalkers**2
        Rms = zeros(nsteps+1) #Root mean square and starts it a one
        steps = range(nsteps+1)
        for i in range(nwalkers):
            x, y, z = walker3(nsteps)
            if i < 5:
                fig = plt.figure(5)
                ax = fig.gca(projection='3d')
                ax.plot(x, y, z)
            d = distance3D(x,y,z)
            Rms += d**2/nwalkers #Adds every time
            if i == nwalkers/2:
                figure(7)
                plot(sqrt(steps),d,label=" Actual")
                legend(loc="best")
        figure(5)
        title('3D Walk')
        xlabel('x-position')
        ylabel('y-position')
        ax.set_zlabel('z-position')
        figure(4)
        xlabel ("x")
        ylabel("y")
        title("Walk 2D")
        Rms = sqrt(Rms)
        figure(7)
        xlabel ("Trials")
        ylabel("RMS")
        title("Walk 3D Rms")
        plot(sqrt(steps),Rms,label="Average")
        plot(sqrt(steps),sqrt(steps),label="Theory")
        legend(loc="best")
```

Listing 7: Function that plots figure 3 and 4

Discussion

The code in 7 and 6 are very similar aside from one being for 2D and the other being for 3D. This is where parts 4 through 9 of the lab occur. Equation (2) can be seen being calculated in line 12 of listing 6. This values are then added to an array and the square root is then taken. Those values are what if being plotted vs the number of walkers in figure 2 and figure 4. As for the accuracy of $2D$ vs $3D$, it was found that less walkers are needed in $3D$ to attain the same accuracy as $2D$. This was done by trail and error until the accuracy of the two was similar.