The Random Walk

BY EESHAN BEOHAR

Introduction

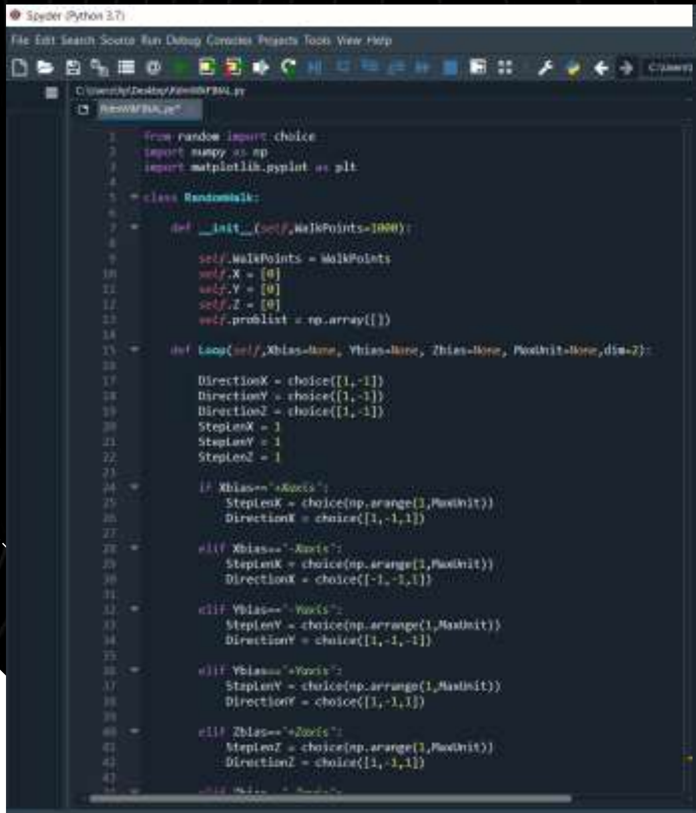
- The Traditional Problem: a drunk wanders off from a lamppost on a street such that he/she is equally likely to take a unit length step towards right or left. The person is sufficiently drunk so that each step is completely independent of the preceding step.
- The question of interest is then, after taking N steps, what is the probability that the person is found at a distance x or what is the probabilistic behavior if N is very large.
- The traditional random walk can be readily generalized to higher dimensions, biased walks with different step probabilities (an inclined street), different unit steps, etc.
- Random Walk is also ideal for understanding the Central Limit Theorem.

Getting Started

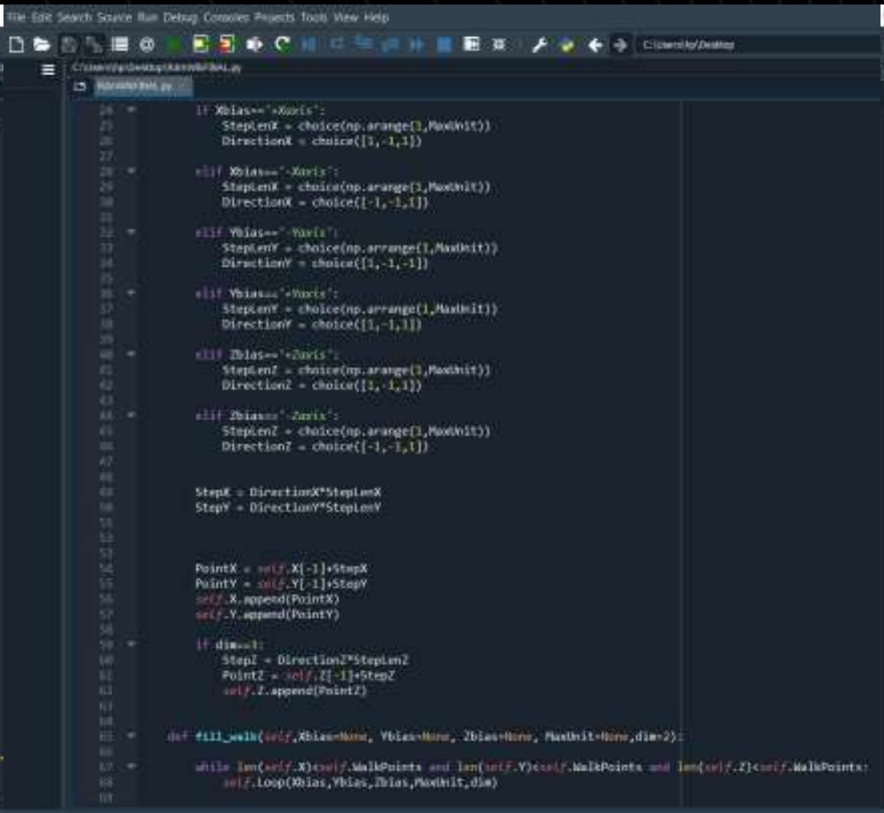
- Modules and Libraries imported are choice from random, numpy and matplotlib.pyplot.
- The entire probabilistic behaviour of the system is regulated by the choice function, which randomly returns an item from a sequence.
- Our aim is to construct a class called RandomWalk() that successfully simulates a 2d or 3d general random walk, and also inherits all the recipes for plotting it.
- We assume that the walk always initiates from an origin (X=Y=Z=0) and lasts for a total number of “WalkPoints”.

Simulating the Walk

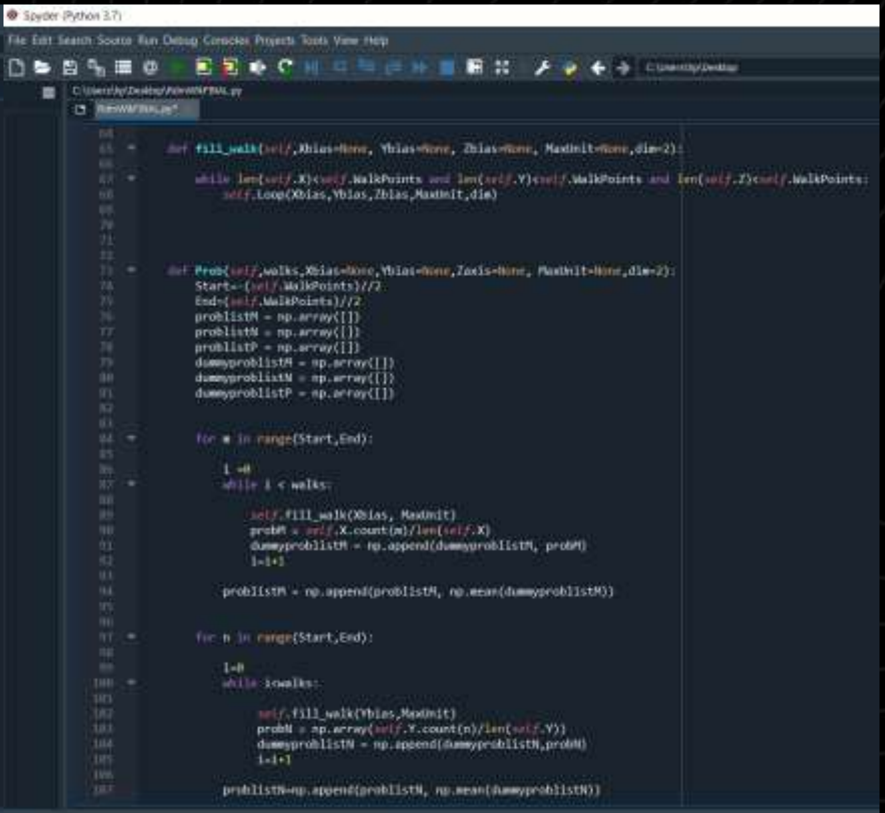
- We create a Loop() function which simulates a point, and takes arguments and parameters depending what sort of walk is required, 2-d or 3-d, biased or unbiased.
- Next, the function fill_walk() puts Loop() under a while loop up until the points are “filled up”


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help

```python
from random import choice
import numpy as np
import matplotlib.pyplot as plt

class RandomWalk:

    def __init__(self,WalkPoints=1000):

        self.WalkPoints = WalkPoints
        self.X = [0]
        self.Y = [0]
        self.Z = [0]
        self.problist = np.array([])

    def Loop(self,Xbias=None, Ybias=None, Zbias=None, MaxUnit=None,dim=2):

        DirectionX = choice([1,-1])
        DirectionY = choice([1,-1])
        DirectionZ = choice([1,-1])
        StepLenX = 1
        StepLenY = 1
        StepLenZ = 1

        if Xbias=='+Xaxis':
            StepLenX = choice(np.arange(1,MaxUnit))
            DirectionX = choice([1,-1,1])

        elif Xbias=='-Xaxis':
            StepLenX = choice(np.arange(1,MaxUnit))
            DirectionX = choice([-1,-1,1])

        elif Ybias=='-Yaxis':
            StepLenY = choice(np.arrange(1,MaxUnit))
            DirectionY = choice([1,-1,-1])

        elif Ybias=='+Yaxis':
            StepLenY = choice(np.arrange(1,MaxUnit))
            DirectionY = choice([1,-1,1])

        elif Zbias=='+Zaxis':
            StepLenZ = choice(np.arange(1,MaxUnit))
            DirectionZ = choice([1,-1,1])

        elif Zbias=='-Zaxis':
```


File Edit Search Source Run Debug Consoles Projects Tools View Help

```python
        if Xbias=='+Xaxis':
            StepLenX = choice(np.arange(1,MaxUnit))
            DirectionX = choice([1,-1,1])

        elif Xbias=='-Xaxis':
            StepLenX = choice(np.arange(1,MaxUnit))
            DirectionX = choice([-1,-1,1])

        elif Ybias=='-Yaxis':
            StepLenY = choice(np.arrange(1,MaxUnit))
            DirectionY = choice([1,-1,-1])

        elif Ybias=='+Yaxis':
            StepLenY = choice(np.arrange(1,MaxUnit))
            DirectionY = choice([1,-1,1])

        elif Zbias=='+Zaxis':
            StepLenZ = choice(np.arange(1,MaxUnit))
            DirectionZ = choice([1,-1,1])

        elif Zbias=='-Zaxis':
            StepLenZ = choice(np.arange(1,MaxUnit))
            DirectionZ = choice([-1,-1,1])


        StepX = DirectionX*StepLenX
        StepY = DirectionY*StepLenY



        PointX = self.X[-1]+StepX
        PointY = self.Y[-1]+StepY
        self.X.append(PointX)
        self.Y.append(PointY)

        if dim==3:
            StepZ = DirectionZ*StepLenZ
            PointZ = self.Z[-1]+StepZ
            self.Z.append(PointZ)


    def fill_walk(self,Xbias=None, Ybias=None, Zbias=None, MaxUnit=None,dim=2):

        while len(self.X)<self.WalkPoints and len(self.Y)<self.WalkPoints and len(self.Z)<self.WalkPoints:
            self.Loop(Xbias,Ybias,Zbias,MaxUnit,dim)
```


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help

```python

    def fill_walk(self,Xbias=None, Ybias=None, Zbias=None, MaxUnit=None,dim=2):

        while len(self.X)<self.WalkPoints and len(self.Y)<self.WalkPoints and len(self.Z)<self.WalkPoints:
            self.Loop(Xbias,Ybias,Zbias,MaxUnit,dim)




    def Prob(self,walks,Xbias=None,Ybias=None,Zaxis=None, MaxUnit=None,dim=2):
        Start=-(self.WalkPoints)//2
        End=(self.WalkPoints)//2
        problistM = np.array([])
        problistN = np.array([])
        problistP = np.array([])
        dummyproblistM = np.array([])
        dummyproblistN = np.array([])
        dummyproblistP = np.array([])


        for m in range(Start,End):

            i =0
            while i < walks:

                self.fill_walk(Xbias, MaxUnit)
                probM = self.X.count(m)/len(self.X)
                dummyproblistM = np.append(dummyproblistM, probM)
                i=i+1

            problistM = np.append(problistM, np.mean(dummyproblistM))


        for n in range(Start,End):

            i=0
            while i<walks:

                self.fill_walk(Ybias,MaxUnit)
                probN = np.array(self.Y.count(n)/len(self.Y))
                dummyproblistN = np.append(dummyproblistN,probN)
                i=i+1

            problistN=np.append(problistN, np.mean(dummyproblistN))
```

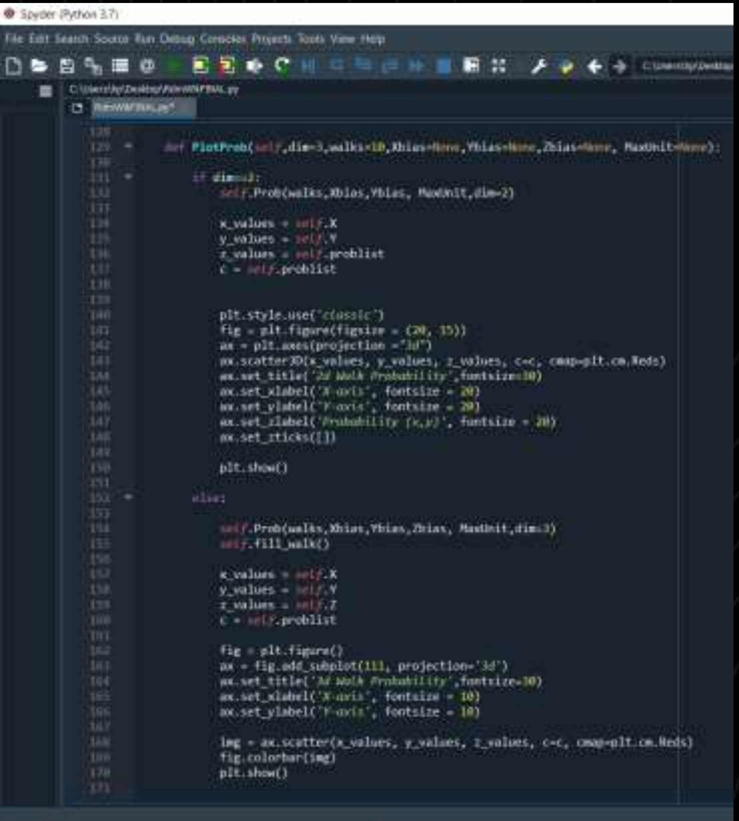
Probability Density

- Prob() takes in the same arguments as fill_walk() with an added parameter walks which determines the number of simulations.
- Prob() calculates the mean probability of a point on the X,Y and Z axis by simulating the walk “walk” times.
- Note that the probabilities of respective dimensions will still be uncorrelated, such that probability of being at (x,y) is just P(x,y)= P(x) x P(y)

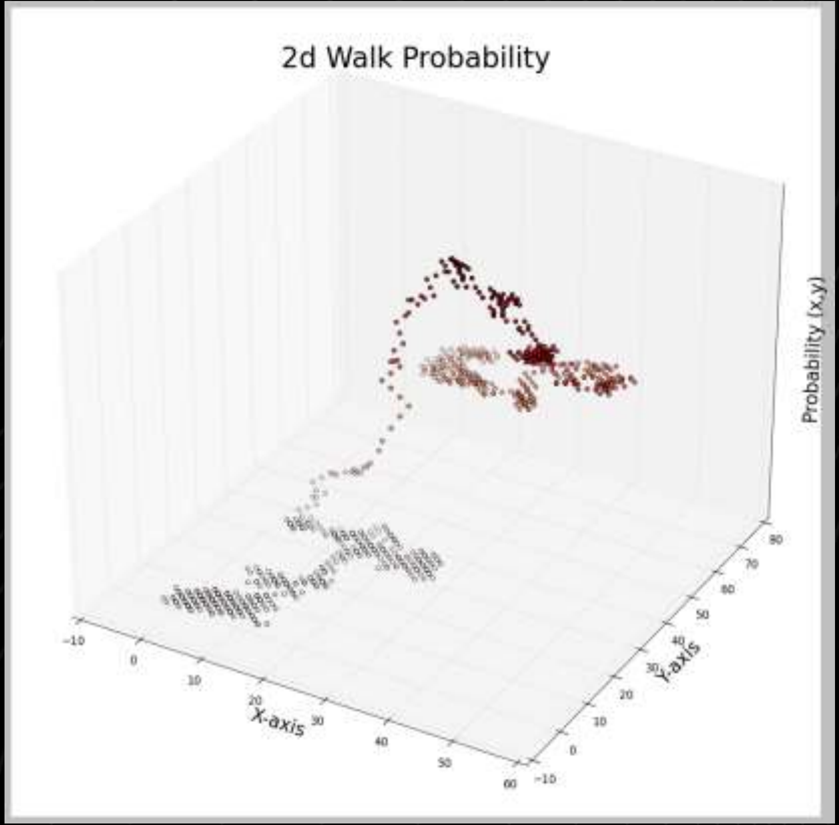
Potting the Probability Densities

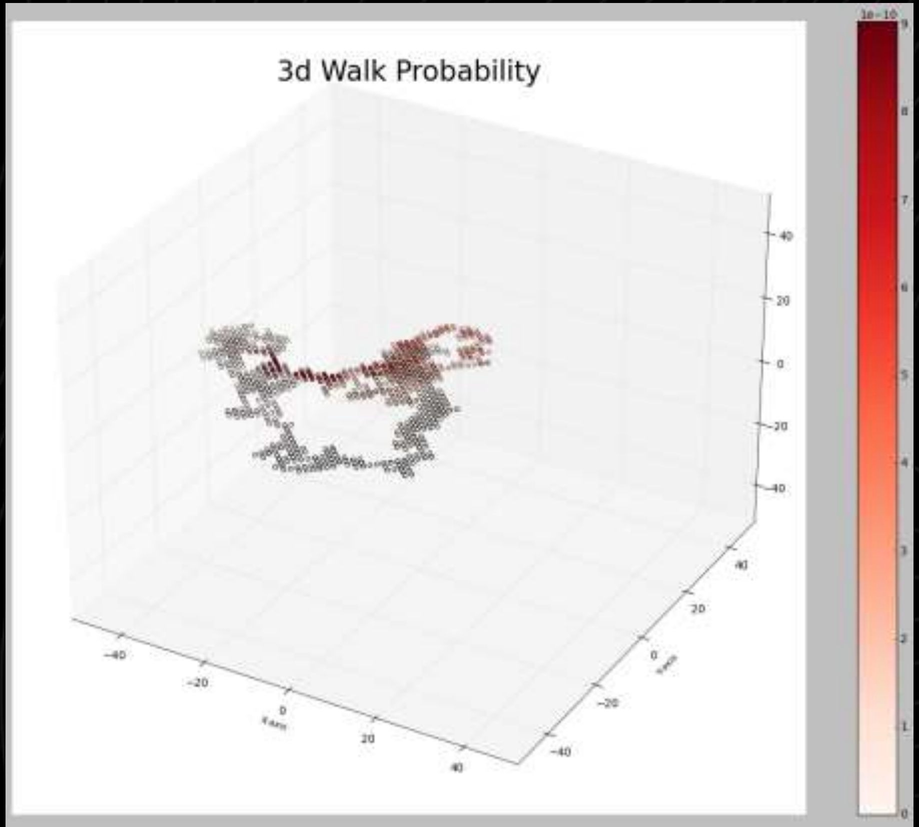
- We create PlotProb() with the aid of Matplotlib to represent probability densities of 2d or 3d walks.

- The Z-axis in 2-d walk plot represents the P(x,y), whereas the colour map in the 3-d walk plot represents the P(x,y,z)


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\hp\Desktop\RWFINAL.py
RWFINAL.py*

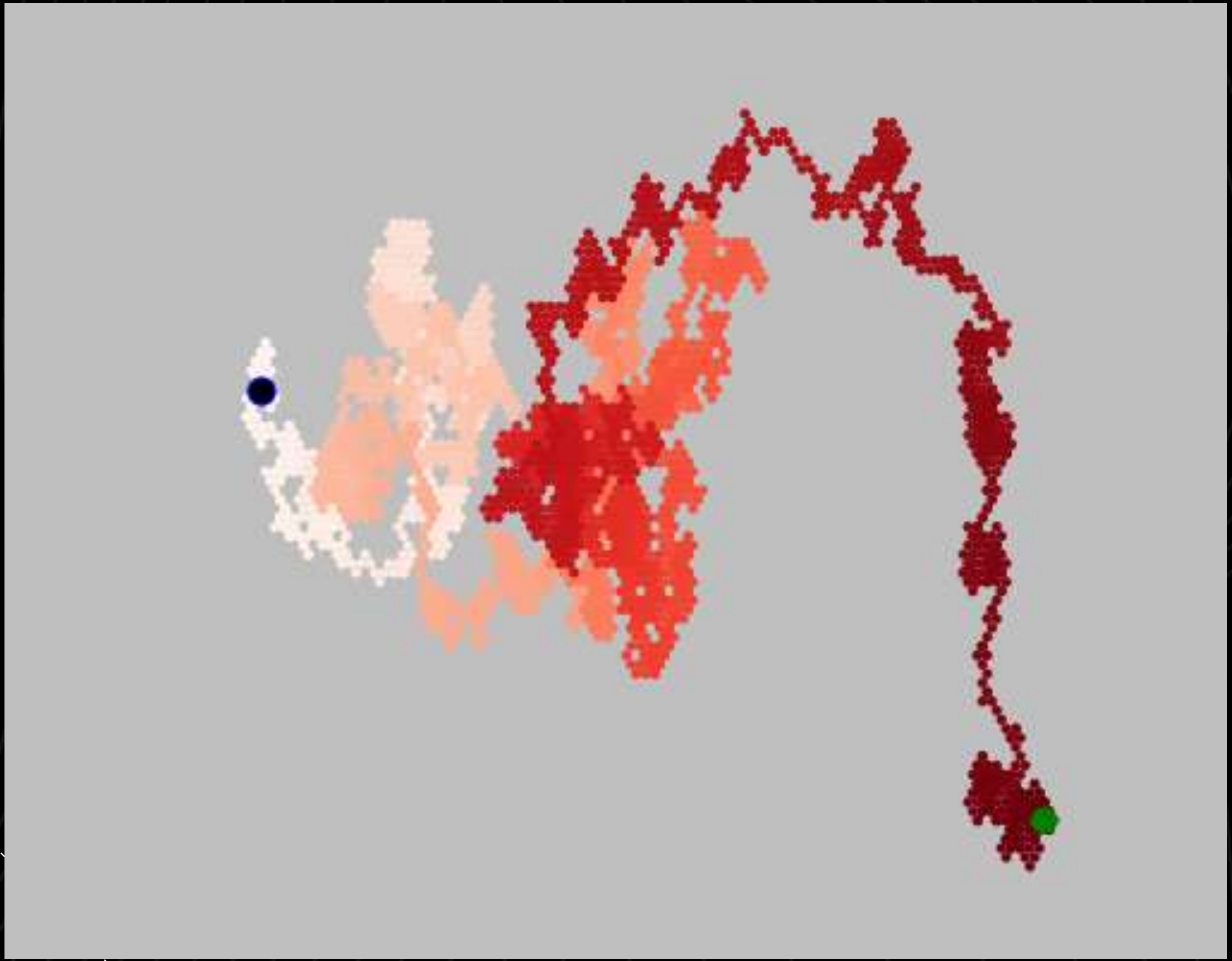
```
    def PlotProb(self,dim=3,walks=10,Xbias=None,Ybias=None,Zbias=None, MaxUnit=None):

        if dim==2:
            self.Prob(walks,Xbias,Ybias, MaxUnit,dim=2)

            x_values = self.X
            y_values = self.Y
            z_values = self.problist
            c = self.problist


            plt.style.use('classic')
            fig = plt.figure(figsize = (20, 15))
            ax = plt.axes(projection ="3d")
            ax.scatter3D(x_values, y_values, z_values, c=c, cmap=plt.cm.Reds)
            ax.set_title('2d Walk Probability',fontsize=30)
            ax.set_xlabel('X-axis', fontsize = 20)
            ax.set_ylabel('Y-axis', fontsize = 20)
            ax.set_zlabel('Probability (x,y)', fontsize = 20)
            ax.set_zticks([])

            plt.show()

        else:

            self.Prob(walks,Xbias,Ybias,Zbias, MaxUnit,dim=3)
            self.fill_walk()

            x_values = self.X
            y_values = self.Y
            z_values = self.Z
            c = self.problist

            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            ax.set_title('3d Walk Probability',fontsize=30)
            ax.set_xlabel('X-axis', fontsize = 10)
            ax.set_ylabel('Y-axis', fontsize = 10)

            img = ax.scatter(x_values, y_values, z_values, c=c, cmap=plt.cm.Reds)
            fig.colorbar(img)
            plt.show()
```

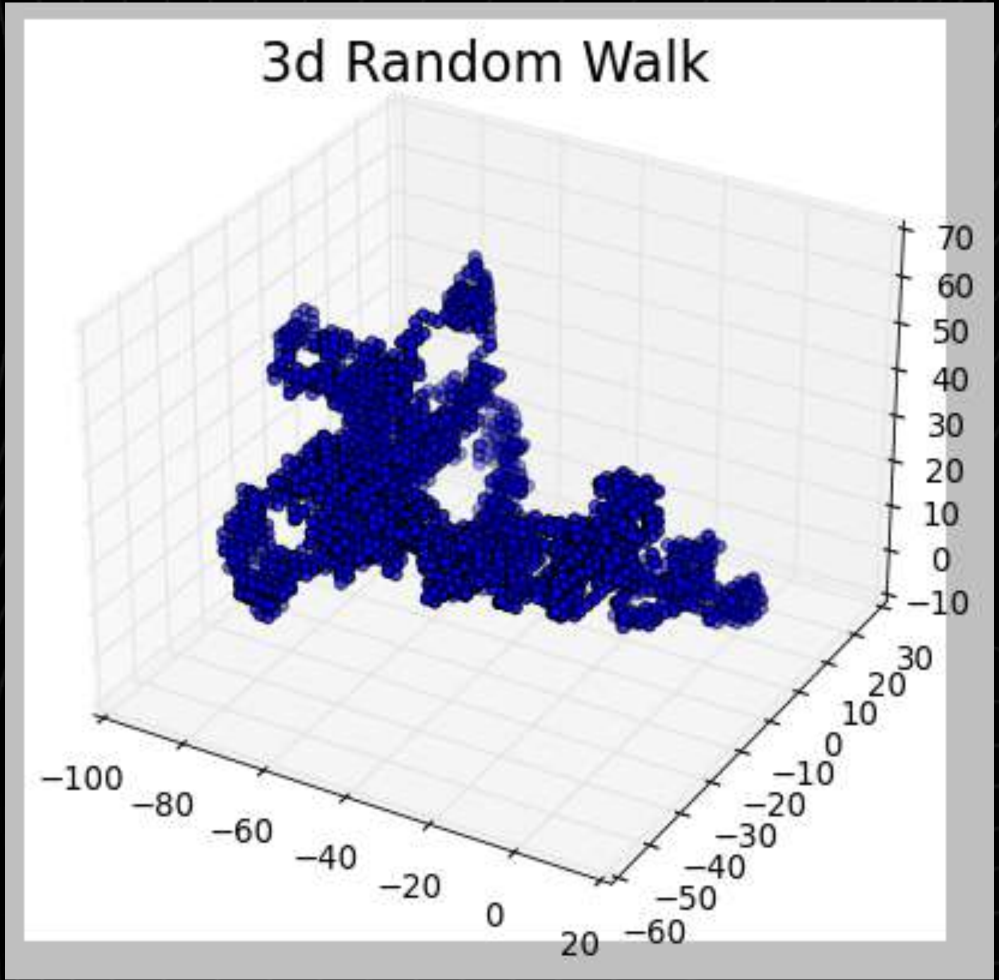




Plotting the Walks

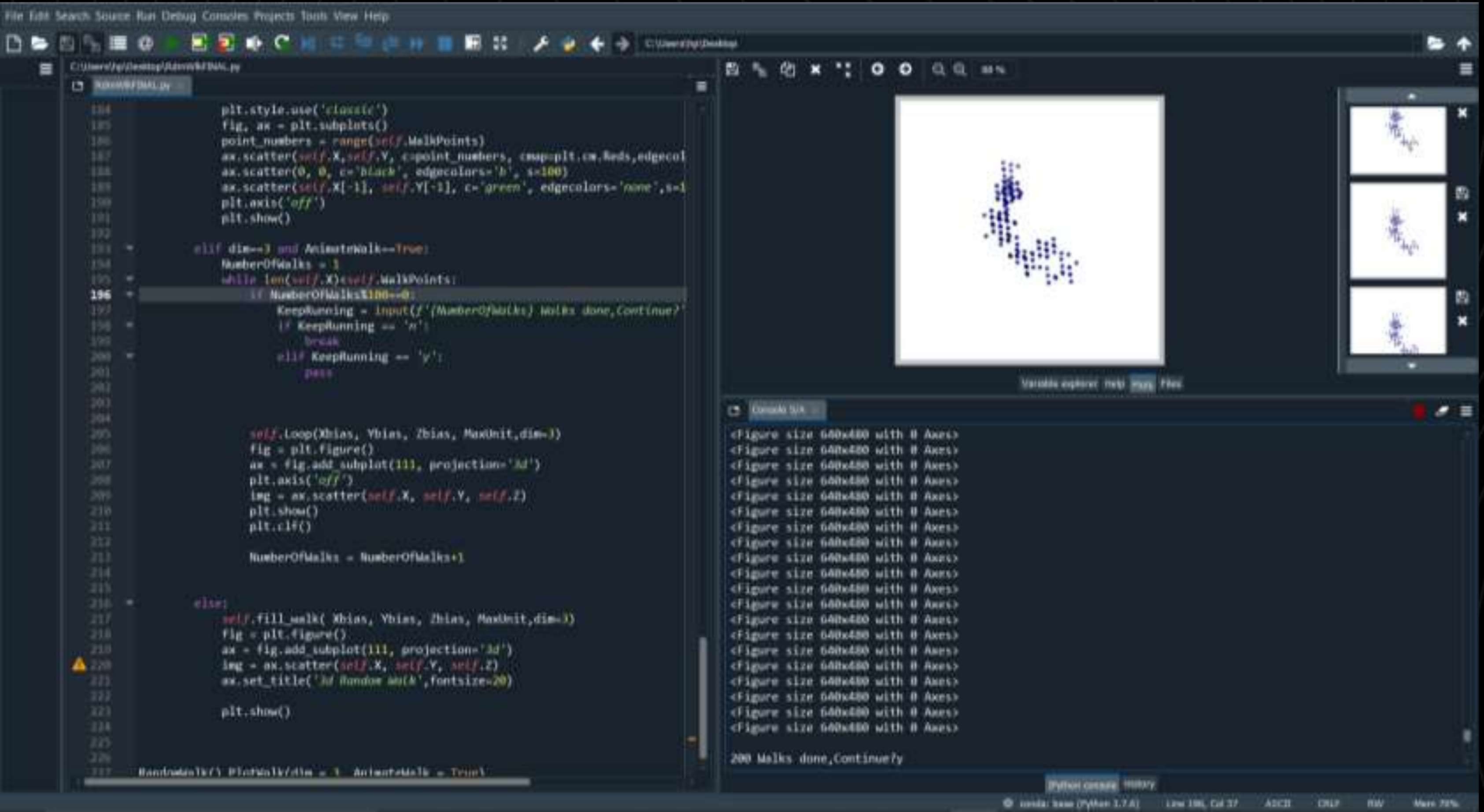
- Again using MatplotLib, PlotWalk() plots the walks, while keeping track of the origin and end points.





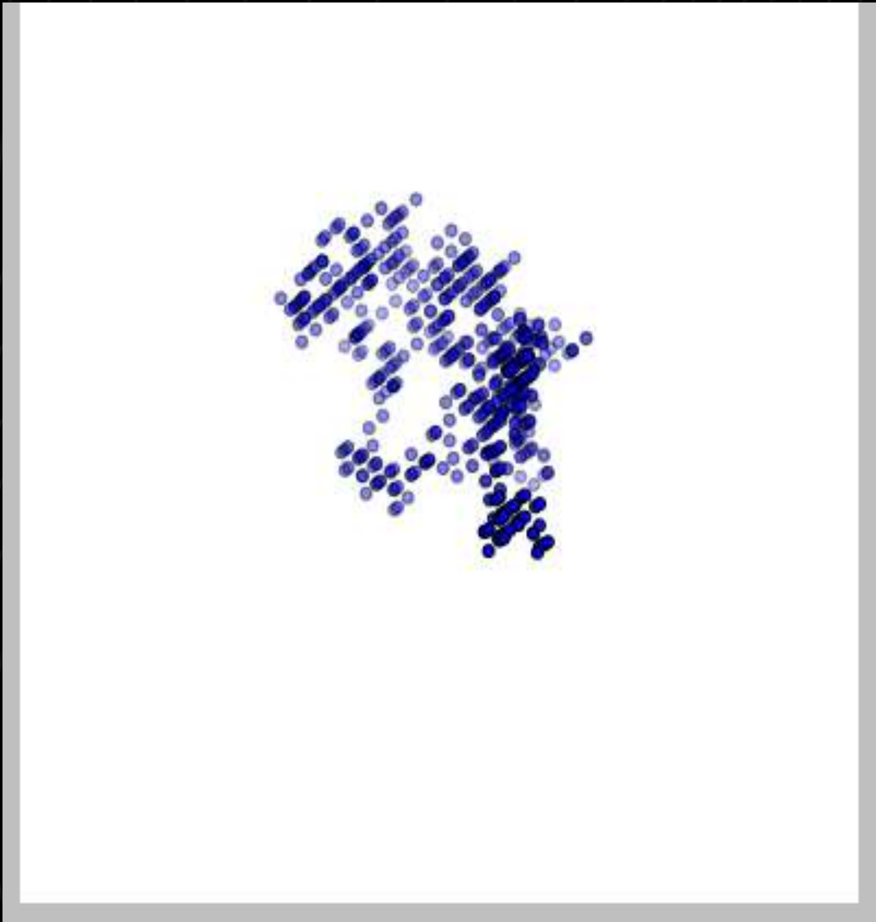
‘DO YOU WALK THE WALK?’

- AnimateWalk() lets you follow the 3-d walk, step by step! Also makes sure every 100 steps whether you want to continue or not.



```
        plt.style.use('classic')
        fig, ax = plt.subplots()
        point_numbers = range(self.WalkPoints)
        ax.scatter(self.X,self.Y, c=point_numbers, cmap=plt.cm.Reds,edgecol
        ax.scatter(0, 0, c='black', edgecolors='b', s=100)
        ax.scatter(self.X[-1], self.Y[-1], c='green', edgecolors='none',s=1
        plt.axis('off')
        plt.show()

    elif dim==3 and AnimateWalk==True:
        NumberOfWalks = 1
        while len(self.X)<self.WalkPoints:
            if NumberOfWalks%100==0:
                KeepRunning = input(f'{NumberOfWalks} Walks done,Continue?'
                if KeepRunning == 'n':
                    break
                elif KeepRunning == 'y':
                    pass



            self.Loop(Xbias, Ybias, Zbias, MaxUnit,dim=3)
            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            plt.axis('off')
            img = ax.scatter(self.X, self.Y, self.Z)
            plt.show()
            plt.clf()

            NumberOfWalks = NumberOfWalks+1


    else:
        self.fill_walk( Xbias, Ybias, Zbias, MaxUnit,dim=3)
        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        img = ax.scatter(self.X, self.Y, self.Z)
        ax.set_title('3d Random Walk',fontsize=20)

        plt.show()



RandomWalk().PlotWalk(dim = 3, AnimateWalk = True)
```

```
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>

200 Walks done,Continue?y
```



THE END