Kernel: SageMath 10.7

Random Walks

MATH 242 Modern Computational Math

Imagine you are standing at the origin of a number line. You flip a fair coin. If the coin lands heads, you move to +1. If the coin lands tails, you move to -1. You flip the coin again and again. Whenever it lands heads, you move 1 unit in the positive direction. Whenever it lands tails, you move 1 unit in the negative direction. Your path on the number line is called a **one-dimensional random walk**.

What does a one-dimensional random walk look like? Let's create some with Python!

1. Building a Random Walk

Use `choice([-1,1])` to simulate one step of the random walk. Try it out:

In [1]:
```
choice([-1,1])
```

Out[1]:
```
-1
```

Now simulate 100 steps of the random walk. Create a list called `locations` consisting of one hundred 0s. The first 0 is the starting location; the other 0s are placeholders for later locations. Compute each location after the first by choosing a +1 or -1 step at random and adding it to the previous location.

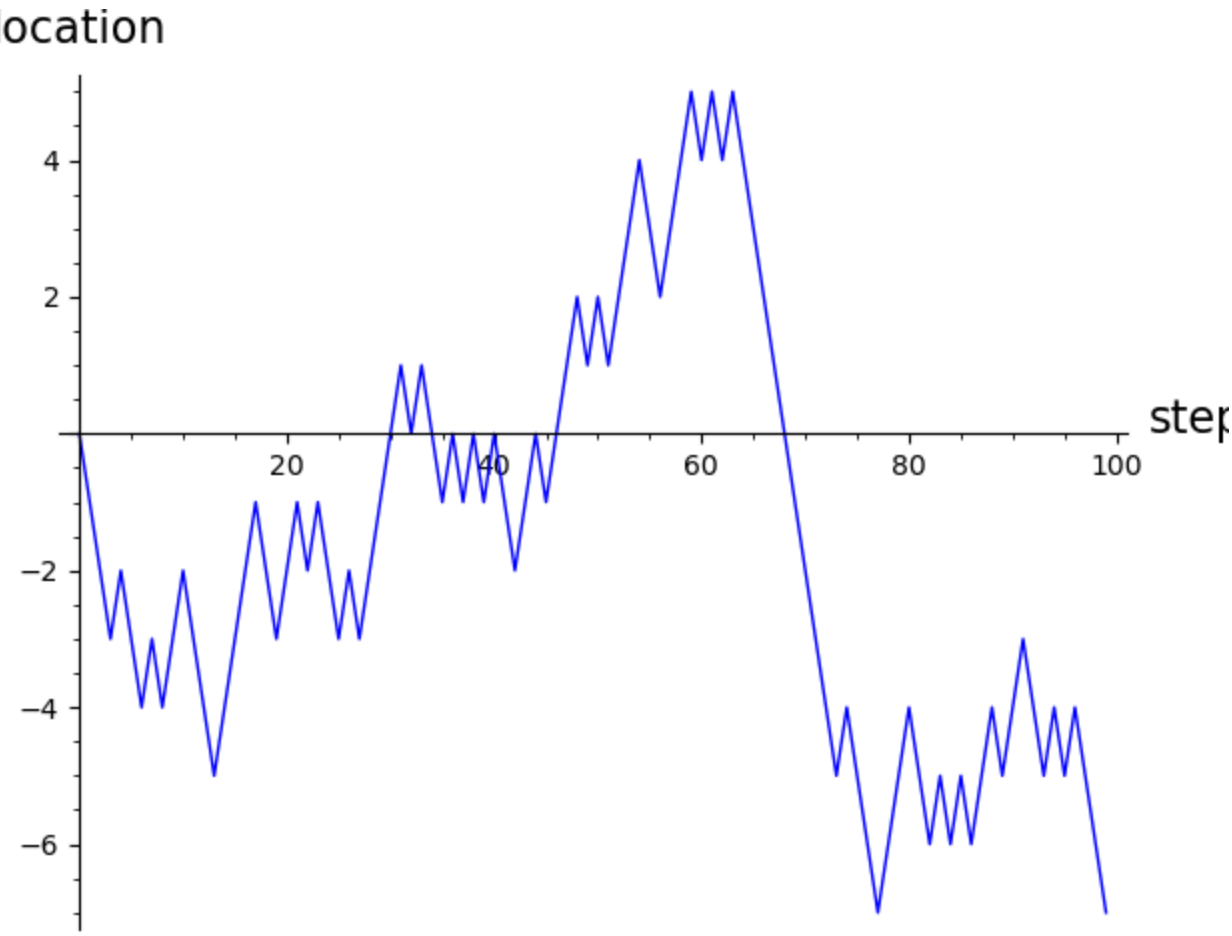
In [3]:
```
locations = [0]*100  # list of 100 zeros
for i in range(1,100):
    move = choice([-1,1])   # current move
    locations[i] = locations[i-1] + move  # store the next location
print( locations )
```

Out[3]:
```
[0, -1, -2, -3, -2, -3, -4, -3, -4, -3, -2, -3, -4, -5, -4, -3, -2, -1, -2, -3, -2,
-1, -2, -1, -2, -3, -2, -3, -2, -1, 0, 1, 0, 1, 0, -1, 0, -1, 0, -1, 0, -1, -2, -1,
0, -1, 0, 1, 2, 1, 2, 1, 2, 3, 4, 3, 2, 3, 4, 5, 4, 5, 4, 5, 4, 3, 2, 1, 0, -1, -2,
-3, -4, -5, -4, -5, -6, -7, -6, -5, -4, -5, -6, -5, -6, -5, -6, -5, -4, -5, -4, -3,
-4, -5, -4, -5, -4, -5, -6, -7]
```

Next, make a plot of the random walk. The horizontal axis will give the number of steps, and the vertical axis will give the location at each step.

In [6]:
```
list_plot( locations , plotjoined=True, axes_labels=["steps","location"])
```

Out[6]:



We will need to generate a lot of random walks. To make this easy, write a function that returns a random walk. Here is the specification for your function:

Function: `randomWalk`

Input: number of steps

Output: a random walk, returned as a list of positions

In [1]:
```
def randomWalk(numSteps):
    locations = [0]*numSteps  # list of 100 zeros
    for i in range(1,numSteps):
        move = choice([-1,1])   # current move
        locations[i] = locations[i-1] + move  # store the next location
    return locations
```

In [9]:
```
rw = randomWalk(200)
print(rw)
```

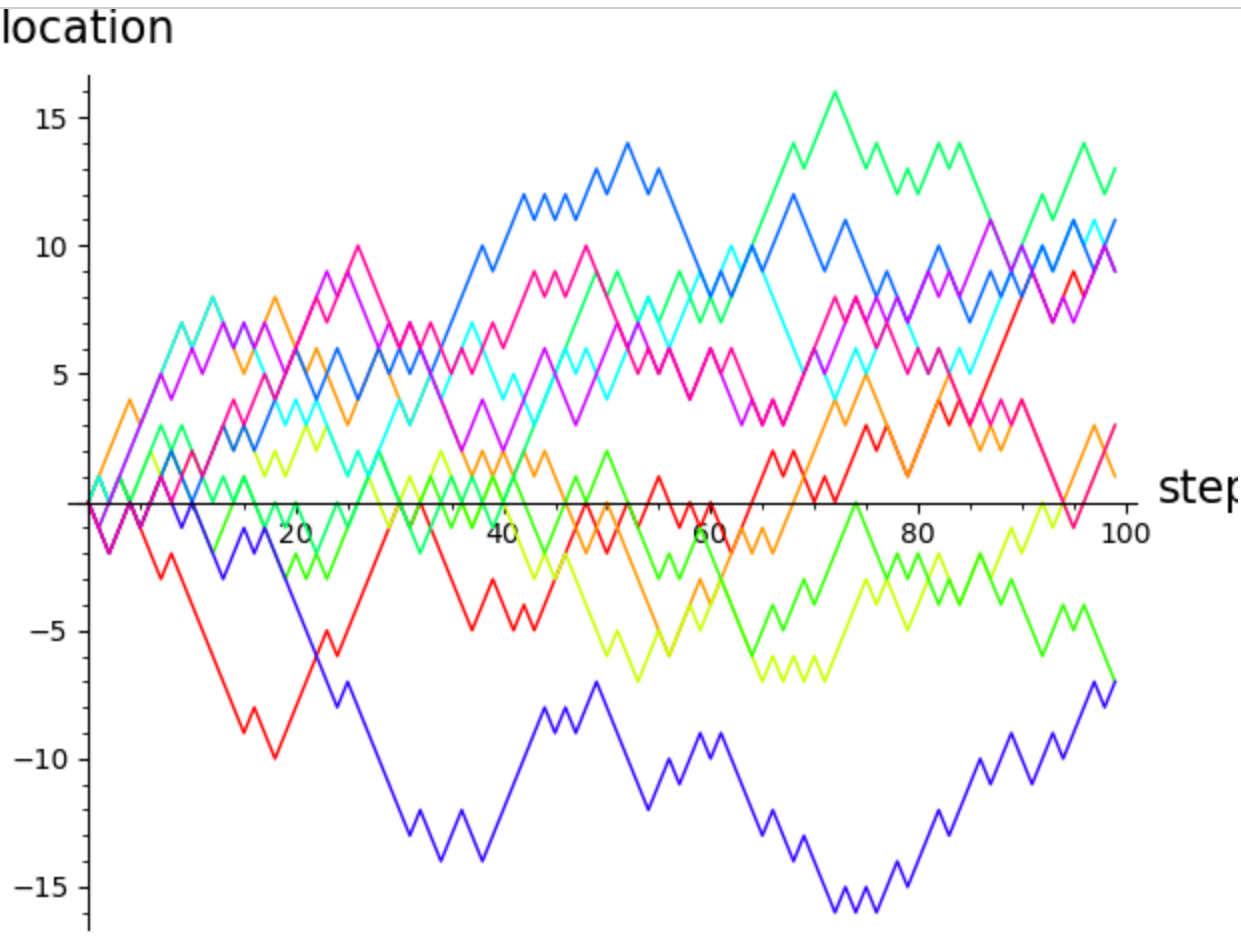
Out[9]:
```
[0, 1, 2, 3, 2, 1, 2, 1, 0, -1, -2, -1, 0, -1, 0, -1, 0, -1, -2, -1, -2, -3, -4,
-5, -6, -5, -4, -5, -4, -3, -4, -3, -4, -5, -6, -7, -6, -5, -6, -5, -6, -5, -4, -5,
-4, -3, -4, -5, -4, -3, -4, -5, -6, -5, -4, -5, -6, -5, -4, -5, -4, -5, -4, -5, -4,
-3, -4, -3, -4, -3, -4, -5, -4, -5, -4, -5, -4, -5, -6, -5, -4, -3, -2, -1, 0, -1,
-2, -1, -2, -1, -2, -1, 0, 1, 2, 1, 0, 1, 2, 3, 2, 1, 0, -1, -2, -1, 0, -1, -2, -3,
-4, -5, -4, -3, -2, -3, -4, -3, -2, -1, -2, -1, 0, 1, 0, 1, 0, -1, 0, 1, 2, 3, 2,
3, 2, 1, 0, 1, 0, 1, 0, -1, 0, 1, 0, -1, -2, -1, -2, -3, -2, -3, -4, -5, -4, -5,
```

```
-4, -3, -4, -5, -4, -5, -6, -7, -8, -9, -8, -9, -10, -9, -8, -7, -6, -5, -6, -7,
-8, -7, -6, -7, -8, -7, -8, -7, -6, -7, -8, -7, -8, -7, -8, -9, -8, -7, -8, -7, -6,
-5, -6, -5]
```

In [11]:
```
rwplots = [list_plot(randomWalk(100), hue=i/10, plotjoined=True) for i in
range(10) ]
rwplot = sum(rwplots)

show(rwplot, axes_labels=["steps","location"])
```

Out[11]:



2. Diameter of a Random Walk

The **diameter** of a random walk is the difference between the maximum and minimum locations in the walk.

Write a function that computes the diameter of a random walk. The input to your function should be a random walk (i.e., a list), and your function should return the diameter of the walk.

Note that Python has built-in functions **`min`** and **`max`** that return the minimum and maximum values in a list.

In [2]:
```
def diameter(aWalk):
    return max(aWalk) - min(aWalk)
```

Test your **`diameter`** function and confirm that it works as expected.

In [3]:
```
rw = randomWalk(30)
print(rw)
diameter(rw)
```

Out[3]:
```
[0, 1, 2, 3, 4, 3, 2, 1, 2, 1, 0, 1, 2, 1, 2, 1, 0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 0,
1, 2, 1]

5
```

Now find the average diameter for random walks of a specified number of steps. Write a function **`avgDiam`** that takes two parameters: the number of steps in a random walk, and the number of random walks to simulate. The function then returns the average diameter of those random walks.

In [4]:
```
def avgDiam(numSteps, numWalks):
    diams = [ diameter(randomWalk(numSteps)) for i in range(numWalks) ]
    return sum(diams)/numWalks
```

In [6]:
```
avgDiam(20, 10).n()
```

Out[6]:
```
6.80000000000000
```

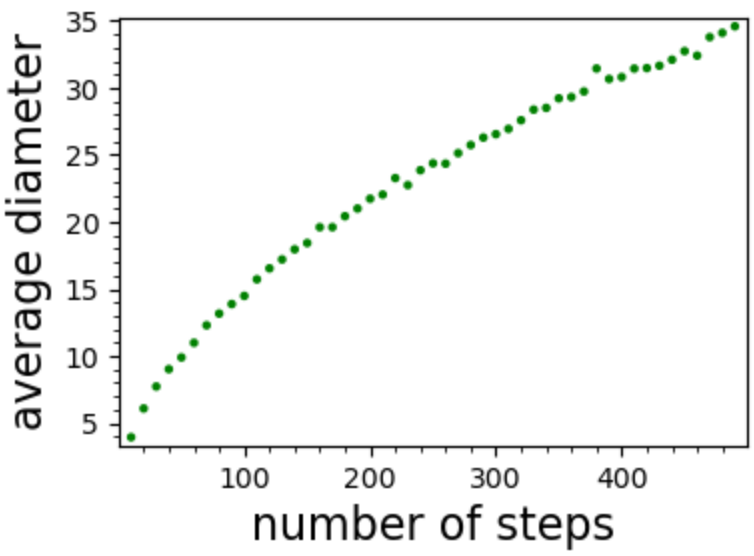
How does the diameter depend on the number of steps?

Create a plot that shows the average diameter of a random walk as a function of the number of steps. Make a conjecture for the growth rate of the average diameter.

In [7]:
```
nvals = range(10,500,10)
avgDiams = [avgDiam(n,500) for n in nvals]
```

In [8]:
```
list_plot(list(zip(nvals,avgDiams)), color="green", axes_labels=["number of
steps","average diameter"], frame=True, figsize=4)
```

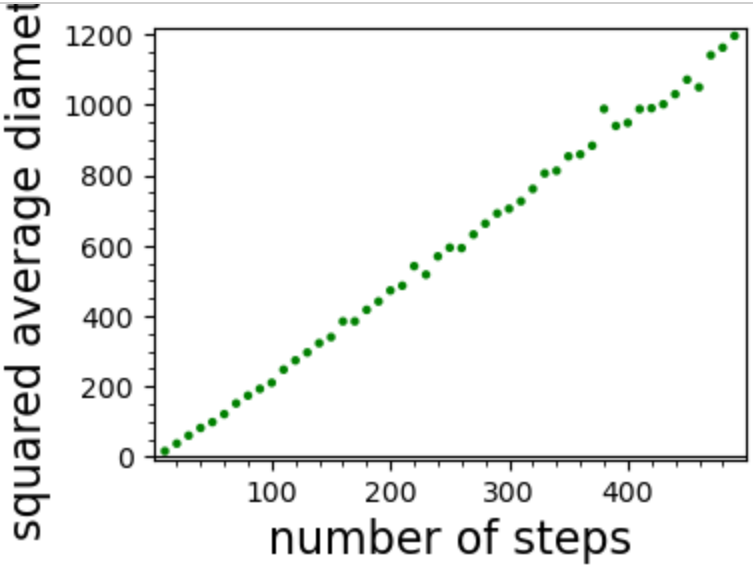
Out[8]:



Could the previous graph have the shape of a square-root function? Let's square the values to find out!

In [9]:
```
sqDiams = [d^2 for d in avgDiams]
list_plot(list(zip(nvals,sqDiams)), color="green", axes_labels=["number of
steps","squared average diameter"], frame=True, figsize=4)
```

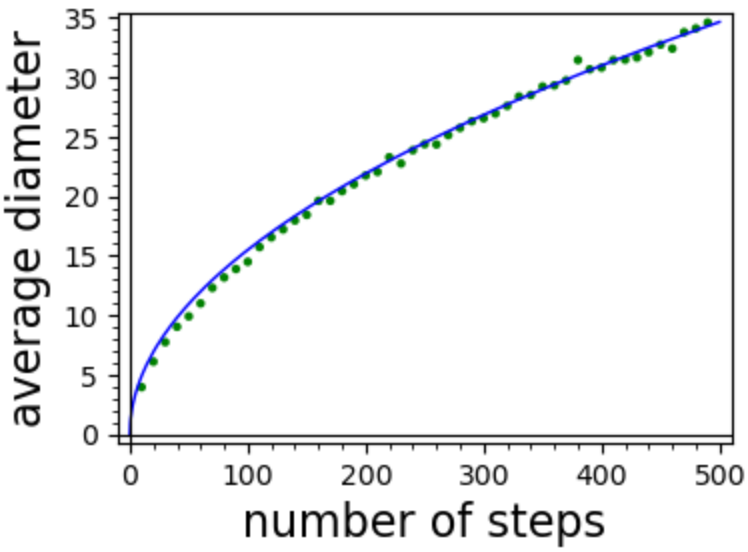
Out[9]:



The squared average diameters are nearly linear! It loks like the slope is approximately $\frac{1200}{500} = 2.4$. Let's plot the average diameters along with the graph of $y = \sqrt{2.4x}$.

In [10]:
```
list_plot(list(zip(nvals,avgDiams)), color="green", axes_labels=["number of
steps","average diameter"], frame=True, figsize=4) + \
    plot(sqrt(2.4*x), (x, 0, 500))
```

Out[10]:



The function $y = \sqrt{2.4x}$ appears to fit our data points fairly well! Thus, it appears that the average diameter is proportional to the square root of the number of steps!

3. Investigate your own questions!

What questions do you have about one-dimensional random walks? Investigate!

In [0]:

In [0]:

In [0]: