Kernel: SageMath 10.4

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 [3]:
```
[ choice( [-1, 1] ) for i in range(20) ]
```

Out[3]: [-1, -1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, 1, 1, 1, 1, -1, 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 [4]:
```
numSteps = 1000
locations = [0]*numSteps

for i in range(1, numSteps):
    move = choice( [-1,1] )
    #print(move)
    locations[i] = locations[i-1] + move

print(locations)
```

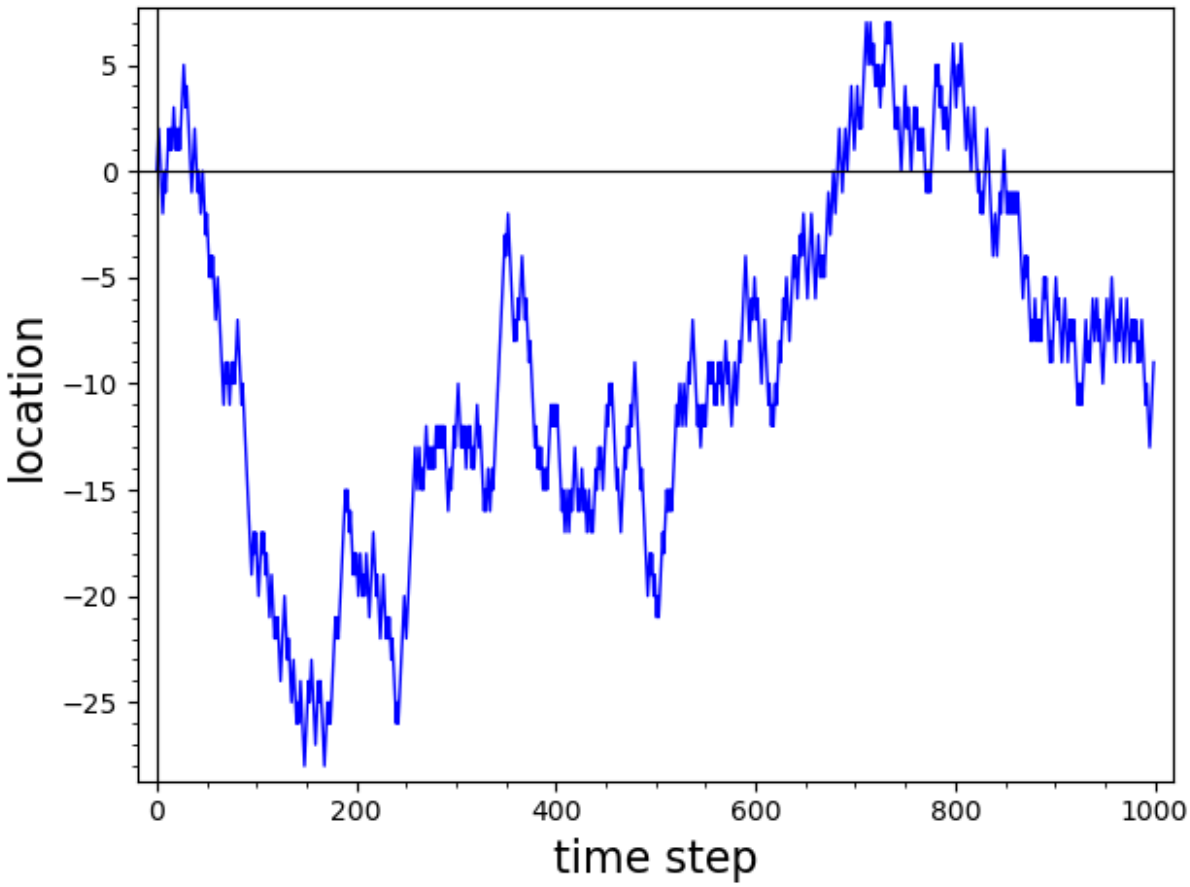
Out[4]:
```
[0, 1, 2, 1, 0, -1, -2, -1, 0, -1, 0, 1, 2, 1, 2, 1, 2, 3, 2, 1, 2, 1,
2, 1, 2, 3, 4, 5, 4, 3, 4, 3, 2, 1, 0, -1, 0, 1, 2, 1, 0, -1, 0, -1, -2,
-1, 0, -1, -2, -3, -2, -3, -4, -5, -4, -5, -4, -5, -6, -7, -6, -5, -6,
-7, -8, -9, -10, -11, -10, -9, -10, -9, -10, -11, -10, -9, -10, -9, -10,
-9, -8, -7, -8, -9, -10, -11, -10, -11, -12, -13, -14, -15, -16, -17,
-18, -19, -18, -17, -18, -17, -18, -19, -20, -19, -18, -17, -18, -17,
-18, -19, -18, -19, -20, -21, -20, -19, -20, -21, -22, -21, -22, -21,
-22, -23, -24, -23, -22, -21, -20, -21, -22, -23, -22, -23, -24, -25,
-24, -23, -24, -25, -26, -25, -26, -25, -24, -25, -26, -27, -28, -27,
-26, -25, -24, -25, -24, -23, -24, -25, -26, -27, -26, -25, -24, -25,
-24, -25, -26, -27, -28, -27, -26, -25, -26, -25, -26, -25, -24, -23,
-22, -21, -22, -21, -22, -21, -20, -19, -18, -17, -16, -15, -16, -15,
-16, -17, -16, -17, -18, -19, -18, -19, -18, -19, -20, -19, -18, -19,
-20, -19, -20, -19, -18, -19, -20, -21, -20, -19, -18, -17, -18, -19,
-20, -19, -20, -21, -22, -21, -20, -19, -20, -21, -22, -21, -22, -21,
-22, -23, -22, -23, -24, -25, -26, -25, -26, -25, -24, -23, -22, -21,
-20, -21, -22, -21, -20, -19, -18, -17, -16, -15, -14, -13, -14, -15,
-14, -13, -14, -15, -14, -15, -14, -13, -12, -13, -14, -13, -14, -13,
-14, -13, -14, -13, -12, -13, -12, -13, -12, -13, -12, -13, -12, -13,
-14, -15, -16, -15, -14, -15, -14, -13, -12, -13, -12, -11, -10, -11,
-12, -13, -12, -13, -12, -13, -14, -13, -12, -13, -12, -13, -14, -13,
-14, -13, -12, -11, -12, -13, -12, -13, -14, -15, -16, -15, -16, -15,
-14, -15, -16, -15, -14, -15, -14, -13, -12, -11, -10, -9, -8, -7, -6,
-5, -4, -3, -4, -3, -2, -3, -4, -5, -6, -7, -8, -7, -8, -7, -6, -7, -6,
-5, -4, -5, -6, -7, -6, -7, -8, -9, -8, -9, -10, -11, -12, -13, -12,
-13, -14, -13, -14, -13, -14, -15, -14, -15, -14, -15, -14, -13, -12,
-11, -12, -11, -12, -11, -12, -11, -12, -13, -14, -15, -16, -15, -16,
-17, -16, -15, -16, -17, -16, -15, -16, -15, -14, -13, -14, -15, -16,
-15, -16, -15, -14, -15, -16, -15, -16, -17, -16, -15, -16, -17, -16,
-17, -16, -15, -14, -15, -14, -13, -14, -13, -14, -15, -14, -13, -12,
-11, -12, -11, -10, -11, -10, -11, -12, -13, -14, -15, -14, -15, -16,
-17, -16, -15, -14, -13, -14, -13, -12, -13, -12, -11, -12, -11, -10,
-9, -10, -11, -12, -13, -14, -15, -14, -15, -16, -17, -18, -19, -20,
-19, -18, -19, -18, -19, -20, -19, -20, -21, -20, -21, -20, -19, -18,
-17, -18, -17, -16, -15, -16, -15, -16, -15, -16, -15, -14, -13, -12,
-11, -12, -11, -10, -11, -12, -11, -10, -11, -12, -11, -10, -9, -10, -9,
-8, -7, -8, -9, -10, -11, -12, -11, -12, -13, -12, -11, -12, -11, -12,
-11, -10, -9, -10, -9, -10, -9, -10, -11, -10, -11, -10, -9, -10, -9,
-10, -11, -10, -9, -8, -9, -10, -9, -10, -11, -12, -11, -10, -9, -10,
-11, -10, -9, -8, -9, -8, -7, -6, -5, -4, -5, -6, -7, -8, -7, -6, -7,
-6, -5, -6, -7, -6, -7, -8, -9, -10, -9, -8, -7, -8, -9, -10, -11, -10,
-11, -12, -11, -12, -11, -10, -11, -10, -9, -8, -9, -8, -7, -6, -7, -6,
-5, -6, -7, -8, -7, -6, -5, -4, -5, -4, -5, -6, -5, -4, -3, -4, -3, -2,
-3, -4, -5, -6, -5, -4, -3, -2, -3, -4, -5, -6, -5, -4, -3, -4, -5, -4,
-5, -4, -5, -4, -3, -2, -1, -2, -3, -2, -1, 0, -1, -2, -1, 0, 1, 2, 1,
0, -1, 0, 1, 2, 1, 0, 1, 2, 3, 4, 3, 2, 1, 2, 3, 4, 3, 2, 3, 2, 3, 4, 5,
6, 7, 6, 5, 6, 7, 6, 5, 6, 5, 4, 5, 4, 5, 4, 3, 4, 5, 4, 5, 6, 7, 6, 7,
6, 7, 6, 5, 4, 3, 2, 3, 2, 3, 2, 1, 0, 1, 2, 3, 4, 3, 2, 3, 2, 1, 0, 1,
2, 3, 2, 3, 2, 1, 2, 1, 2, 1, 2, 1, 0, -1, 0, -1, 0, -1, 0, 1, 2, 3, 4,
5, 4, 5, 4, 3, 4, 3, 2, 3, 2, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 4, 5, 4,
5, 6, 5, 4, 3, 2, 1, 2, 3, 2, 1, 0, 1, 2, 3, 2, 1, 0, -1, 0, -1, -2, -1,
-2, -1, 0, 1, 2, 1, 0, -1, -2, -3, -4, -3, -2, -3, -4, -3, -2, -1, -2,
-1, 0, 1, 0, -1, -2, -1, -2, -1, -2, -1, -2, -1, -2, -1, -2, -1, -2, -3,
-4, -5, -6, -5, -4, -5, -4, -5, -6, -7, -8, -7, -8, -7, -6, -7, -8, -7,
-8, -7, -8, -7, -6, -5, -6, -5, -6, -7, -8, -9, -8, -9, -8, -7, -6, -5,
-6, -7, -6, -7, -8, -9, -8, -7, -6, -7, -8, -9, -8, -7, -8, -7, -8, -7,
-8, -9, -10, -11, -10, -11, -10, -11, -10, -9, -8, -7, -8, -9, -8, -9,
-8, -7, -6, -7, -8, -7, -6, -7, -8, -7, -8, -9, -10, -9, -8, -7, -6, -7,
-8, -7, -6, -5, -6, -7, -8, -9, -8, -7, -8, -7, -6, -7, -8, -9, -8, -7,
```

```
-6, -7, -8, -9, -8, -7, -8, -7, -8, -7, -8, -9, -8, -9, -8, -7, -8, -9,
-10, -11, -10, -11, -12, -13, -12, -11, -10, -9]
```

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 [5]:
```
list_plot( locations, plotjoined=True, axes_labels=["time
step","location"], frame=True)
```

Out[5]:



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 [6]:
```
def randomWalk(numSteps):
    # initialize list of locations
    locations = [0]*numSteps

    # main loop
    for i in range(1, numSteps):
        move = choice( [-1,1] )
        #print(move)
        locations[i] = locations[i-1] + move
```

```
    # return the list of locations
    return locations
```

In [7]:
```
# testing
rw = randomWalk(100)
print(rw)
```

Out[7]:
```
[0, -1, -2, -3, -4, -5, -4, -3, -2, -1, -2, -1, -2, -3, -4, -5, -6, -7,
-6, -5, -6, -5, -4, -3, -2, -3, -4, -3, -4, -5, -6, -7, -8, -7, -6, -5,
-6, -7, -8, -9, -8, -9, -8, -7, -8, -7, -8, -7, -6, -5, -6, -5, -4, -5,
-4, -3, -2, -1, 0, 1, 2, 3, 4, 3, 2, 3, 2, 1, 0, 1, 0, -1, 0, -1, 0, 1,
0, -1, 0, -1, 0, 1, 0, 1, 2, 3, 2, 1, 0, -1, 0, -1, -2, -1, -2, -1, 0,
1, 2, 1]
```

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 [8]:
```
def diameter(aWalk):
    return max(aWalk) - min(aWalk)
```

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

In [0]:
```

```

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 [9]:
```
def avgDiam(numSteps, numWalks):
    diams = [diameter(randomWalk(numSteps)) for _ in range(numWalks)]
    return sum(diams)/numWalks
```

In [10]:
```
avgDiam(100, 100)
```

Out[10]:
```
739/50
```

In [0]:
```

```

How does the diameter depend on the number of steps?

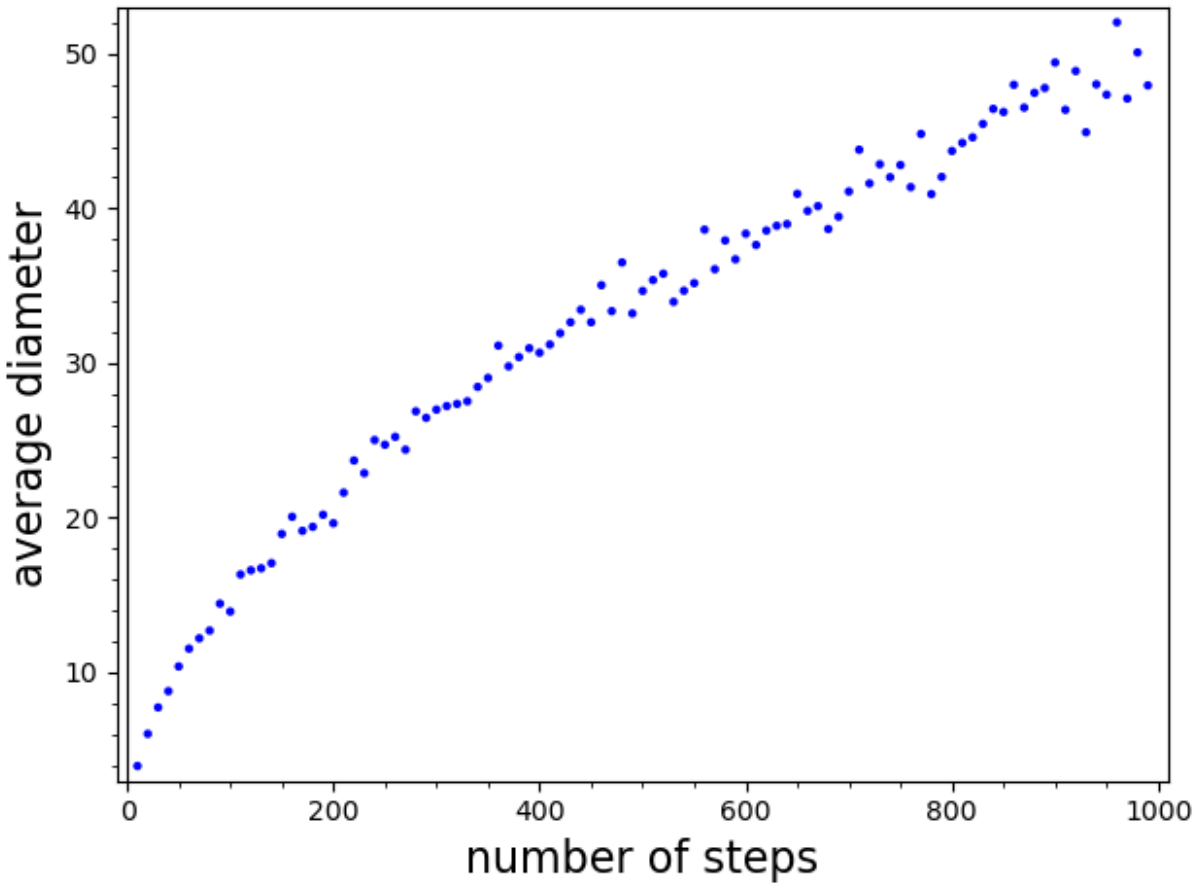
Create a plot that shows the average diameter of a random walk as a function of the number of steps. Make a conjecture for the growth rate of the average diameter.

In [11]:
```
stepVals = range(10, 1000, 10)
diamVals = [avgDiam(s, 100) for s in stepVals]
diamVals
```

Expand

In [12]:
```
list_plot( list(zip(stepVals, diamVals)) , axes_labels=["number of steps","average diameter"], frame=True )
```

Out[12]:



3. Investigate your own questions!

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

In [0]:

In [0]:

In [0]: