Kernel: SageMath 10.7

Two-Dimensional Random Walks

MATH 242 Modern Computational Math

Today we will start to understand properties of two-dimensional (2D) random walks. For now, we will specify that our 2D random walks start at the origin and take unit steps (of length 1) up, down, left, and right. That is, each position of the random walk will be a pair of integers.

Since each position of the random walk is now an (x, y) coordinate pair, storing the trajectory of the random walk requires us to store a list of pairs. Before we generate 2D random walks, we should practice working with lists of pairs in Sage.

First, how will we generate a random step of a 2D random walk? Since there are only four possible directions (up, down, left, and right), we could simply define a list containing unit steps in these four directions, as in the following code cell.

In [1]:
```
# list of the possible directions
dirs = [[0,1],[0,-1],[1,0],[-1,0]]
```

To choose a random step, we can generate a random index from the set $\{0, 1, 2, 3\}$ and select the direction at that index in our list. Try this out in the next code cell.

In [3]:
```
r = randrange(4)
move = dirs[r]
print("index: ", r)
print("move: ", move)
```

Out[3]:
```
index:  2
move:  [1, 0]
```

Next, we need to think about how to actually make a move. With 1D random walks, we could simply add our current position and our move to obtain our new position. What happens if we try that with 2D positions?

In [4]:
```
current = [3,3]
new = current + move
print("current: ", current)
print("move: ", move)
print("new: ", new)
```

Out[4]:
```
current:  [3, 3]
move:  [1, 0]
new:  [3, 3, 1, 0]
```

That didn't work because in Python, the + operator concatenates lists. We want to *add* the the lists, element by element, to obtain the new position. For 2D positions, the simplest approach is probably just to add the first entries and then add the second entries, as in the next code cell.

In [5]:
```
current = [3,3]
new = [current[0] + move[0], current[1] + move[1]]
print("current: ", current)
```

```
print("move: ", move)
print("new: ", new)
```

Out[5]:
```
current:  [3, 3]
move:  [1, 0]
new:  [4, 3]
```

Now we can generate a 2D random walk. Let's write our code in a function that we can easily call to obtain a random walk of any desired number of steps.

In [1]:
```
# function: randomWalk 2D
# input: number of steps
# output: a 2D random walk, specified as a list of positions
def randomWalk2D(numSteps):
    # store the possible directions
    dirs = [[0,1],[0,-1],[1,0],[-1,0]]

    # set up a list to store the locations visited.
    positions = [[0,0]] * (numSteps+1)
    #print(positions)

    # take steps
    for i in range(1, numSteps+1):
        r = randrange(4)
        move = dirs[r]
        positions[i] = [positions[i-1][0] + move[0], positions[i-1][1] + move[1]]

    # return the random walk as a list of locations
    return positions



# testing
rw = randomWalk2D(20)
print(rw)
```

Out[1]:
```
[[0, 0], [1, 0], [1, -1], [2, -1], [1, -1], [1, 0], [0, 0], [0, 1], [0, 0], [0, 1],
[0, 0], [0, -1], [0, 0], [-1, 0], [-1, 1], [0, 1], [0, 2], [0, 3], [1, 3], [1, 2],
[1, 3]]
```

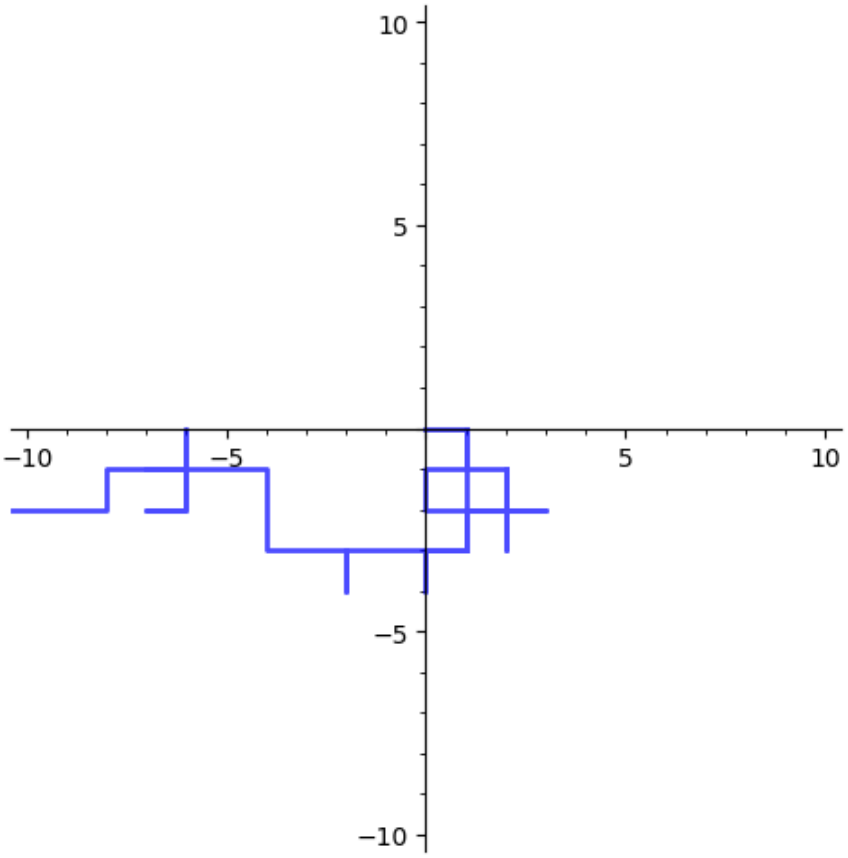
In [7]:
```
[[0,0]]*5
```

Out[7]:
```
[[0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
```

We can also plot the random walk. Here, we will make a 2D plot.

In [12]:
```
rw = randomWalk2D(100)
list_plot(rw, plotjoined=True, thickness=2, alpha=0.7, xmin=-10, xmax=10, ymin=-10,
ymax=10, figsize=[5,5])
```

Out[12]:



In [14]:

```
rw = randomWalk2D(10)
print(rw)

# get list of x-coordinates
xList = [pair[0] for pair in rw]
print(xList)
```

Out[14]:

```
[[0, 0], [1, 0], [1, 1], [0, 1], [-1, 1], [0, 1], [1, 1], [1, 0], [1, 1], [1, 2], [0, 2]]
[0, 1, 1, 0, -1, 0, 1, 1, 1, 1, 0]
```

1. Diameter of a 2D Random Walk

Define the **width** of a 2-D random walk to be the maximum difference between any two x-coordinates attained by the walk. Similarly, define the **height** of the walk to be the maximum difference between any two y-coordinates.

Define the **diameter** of a 2-D random walk to be the maximum of the width and height of the walk. (This isn't the only way to define the diameter of a 2-D random walk, but it's simple. Can you think of other ways?)

What is the average diameter of a 2-D random walk after 100 steps?

What is the average diameter of a 2D random walk after N steps? How does this depend on N?

In [3]:

```
# function to compute the diameter of a given 2D random walk
def diameter2D(aWalk):
    xcoords = [pair[0] for pair in aWalk]
    width = max(xcoords) - min(xcoords)

    ycoords = [pair[1] for pair in aWalk]
    height = max(ycoords) - min(ycoords)

    return max(width, height)
```

```
# testing
rw = randomWalk2D(20)
print(rw)
diameter2D(rw)
```

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

5
```

In [4]:
```
# function to compute the average diameter of many 2D random walks
def avgDiameter2D(numSteps, numWalks):
    diams = [diameter2D(randomWalk2D(numSteps)) for _ in range(numWalks)]
    return sum(diams)/numWalks
```

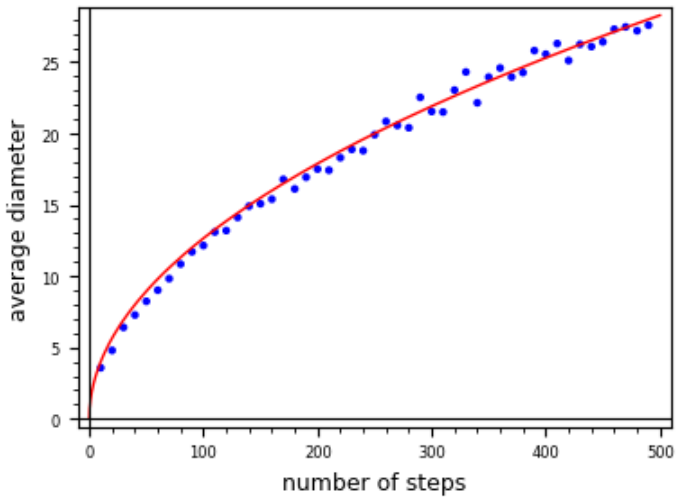
In [5]:
```
# compute average diameters for various numbers of steps
nVals = range(10,500,10)
avgDiams = [avgDiameter2D(n,100) for n in nVals]
print(avgDiams)
```

Out[5]:
```
[357/100, 479/100, 32/5, 727/100, 823/100, 901/100, 491/50, 271/25, 117/10, 607/50,
1309/100, 1319/100, 1411/100, 149/10, 1507/100, 1539/100, 839/50, 1611/100, 1693/100,
35/2, 871/50, 183/10, 472/25, 1879/100, 1991/100, 521/25, 514/25, 102/5, 2253/100,
1077/50, 43/2, 576/25, 2431/100, 443/20, 479/20, 2459/100, 599/25, 2427/100,
2581/100, 639/25, 263/10, 2511/100, 105/4, 2609/100, 1321/50, 2731/100, 2747/100,
136/5, 2759/100]
```

It looks like the average diameter of an N-step 2D random walk is roughly proportional to $\sqrt{N}$:

In [6]:
```
list_plot( list(zip(nVals, avgDiams)), axes_labels=["number of steps","average
diameter"], frame=True) + plot(sqrt(1.6*x), (x, 0, 500), color="red", figsize=4,
fontsize=6)
```

Out[6]:



2. Four Quadrants

How likely is it that a 2D random walk visits all four quadrants of the plane within 100 steps?

How likely is this within N steps? How does this depend on N?

In [0]:

In [0]:

In [0]:

In [0]:

3. Distinct Points Visited

How many distinct points does a 2D random walk visit, on average, in N steps? How does this depend on N?

In [0]:

In [0]:

In [0]:

In [0]: