Kernel: SageMath 10.7

Random Walks

MATH 242 Modern Computational Math

Today we will continue our study of 1D random walks.

One-Dimensional Random Walks

Here is our **`randomWalk`** function from last time.

In [5]:
```
# function: randomWalk
# input: number of steps
# output: a random walks, returned as a list of numbers
def randomWalk(numSteps):
    # initialize list of locations
    locations = [0]*numSteps

    # main loop
    for i in range(1, numSteps):
        move = choice( [-1,1] )   # generate a random move
        #print(move)
        locations[i] = locations[i-1] + move  # store the next location

    # return the list of locations
    return locations
```

Run the **`randomWalk`** function:

In [2]:
```
rw = randomWalk(50)
print(rw)
```

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

Today we will explore the frequency with which 1D random walks return to the origin. There are multiple questions we can ask about this. Three questions appear below.

1. What proportion of 1D simple symmetric random walks return to the origin at least once in their first 20 steps?

In [9]:
```
numSteps = 20
numWalks = 1000

returns = [randomWalk(numSteps).count(0)-1 for _ in range(numWalks)]
# print(returns)

# percent that return to the origin at least once
n(numWalks - returns.count(0), digits=3)/numWalks
```

Out[9]: 0.830

In [0]:

In [0]:

2. What proportion of 1D simple symmetric random walks return to the origin at least once in their first N steps? How does this depend on N?

Choose various values of N. For each value of N, compute the proportion of random walks that return to the origin in their first N steps. Make a table and/or a plot to display your results. Can you conjecture a formula for the proportion of random walks that return to the origin in their first N steps?

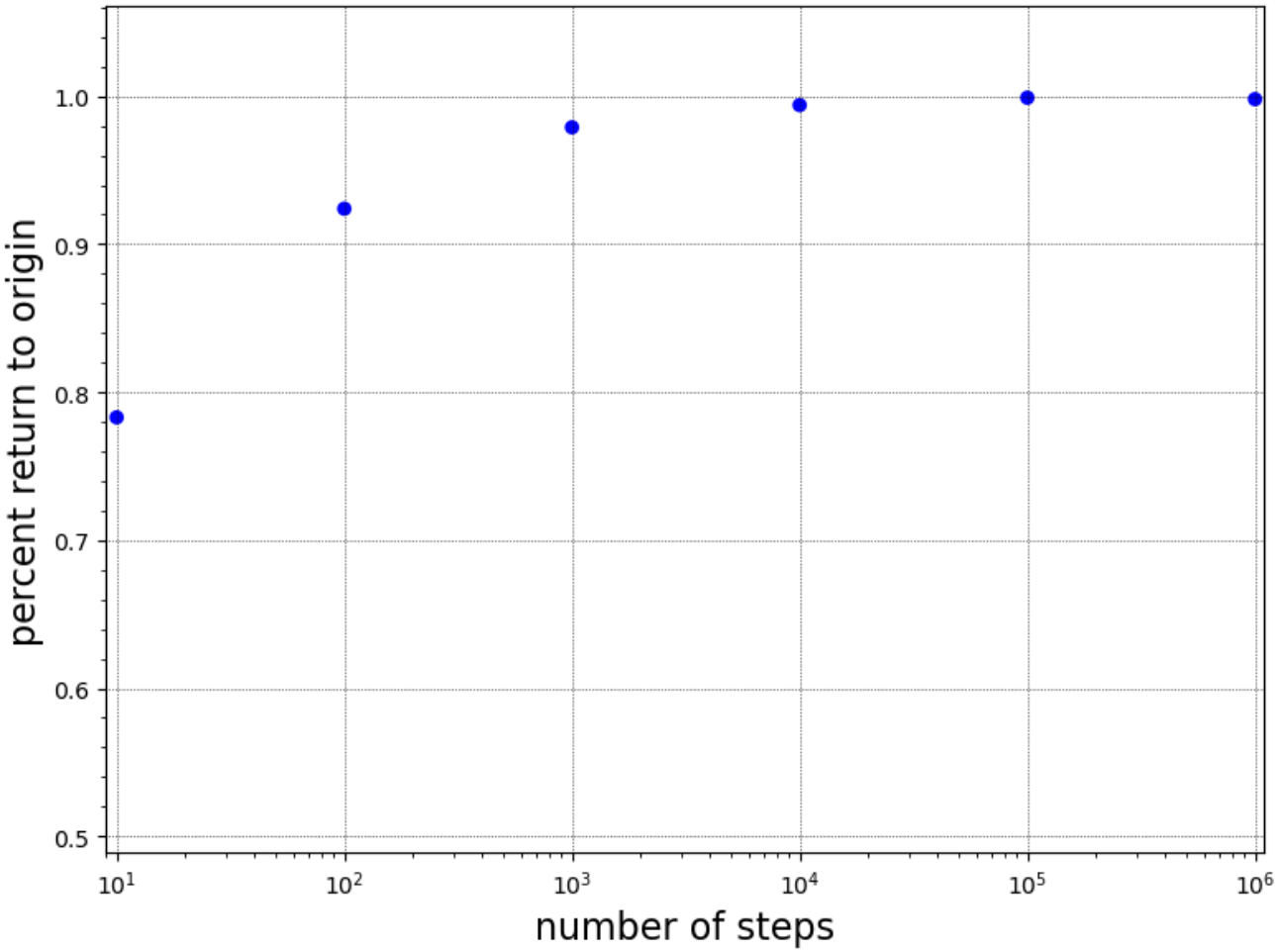
In [1]:
```
# simulates a random walk for up to maxSteps
# return: True if random walk revisits the origin; False otherwise
def doesItReturn(maxSteps):
    location = 0
    for _ in range(maxSteps):
        location = location + choice([-1,1])
        if location == 0:
            return True
    return False

# compute the proportion of random walks of specified number of steps that return to the
origin
def computeProportion(numWalks, maxSteps):
    vals = [doesItReturn(maxSteps) for _ in range(numWalks)]
    return sum(vals)/numWalks
```

In [2]:
```
# specify the numbers of steps to simulate
stepVals = [10^k for k in range(1,7)]

# specify the number of random walks to simulate for each number of steps
numWalks = 1000

# create a list to store the proportions
props = []

# for each number of steps, simulate random walks and store the proportion that return to
the origin
for steps in stepVals:
    props.append( computeProportion( numWalks, steps ) )

# display the proportions
props
```

Out[2]:
```
[783/1000, 231/250, 979/1000, 497/500, 999/1000, 499/500]
```

In [3]:
```
# make a plot of the proportions
list_plot(list(zip(stepVals,props)),
          marker='o', size=40, figsize=8,
          ymin=0.5, ymax=1.05,
          scale="semilogx", frame=True, gridlines=True,
          axes_labels=["number of steps","percent return to origin"])
```

Out[3]:



3. What proportion of 1D simple symmetric random walks do you think will *eventually* return to the origin, if you could simulate them for arbitrarily large numbers of steps? How do your computations above inform your answer?

In [0]:

In [0]:

Now explore your own questions about random walks.

For example, you could explore the following questions:

- On average, how many times does a simple random walk return to the origin in its first 100 steps? ...in its first N steps?
- What is the probability that a simple random walk reaches a distance of 50 from the origin before returning to the origin? ...reaches a distance d before returning to the origin?
- How often does a random walk visit each position on the number line? Start your investigation by making histogram showing the distribution of positions in a random walk. Run your code several times to see histograms for different random walks, with various numbers of steps. How would your distribution be different if you combine the position lists from *many* random walks?

Or explore a different question that interests you!

In [0]:

In [0]:

In [0]:

In [3]:
```
plot(sqrt(x), (x,0,100))
```

Out[3]: