Kernel: SageMath 10.7

Three-Dimensional Random Walks

MATH 242 Modern Computational Mathematics

The following code cell generates a 3D random walk and draws it using the Sage **`line3d`** function. Note that the drawing is interactive -- you can rotate it and zoom in and out.

In [1]:
```
# define the possible moves at each step of the random walk
dirs = matrix( [[1,0,0],[-1,0,0],[0,1,0],[0,-1,0],[0,0,1],[0,0,-1]] )

# define the number of steps to take
numSteps = 200

# set up a numpy 2-D array to store the locations visited.
locations = zero_matrix(numSteps, 3) # numSteps rows, 2 columns
#print(locations)

# take steps
for i in range(numSteps):
    r = randrange(6)
    move = dirs[r]  # direction to move
    locations[i] = locations[i-1] + move  # next location

# draw the random walk
line3d( locations, opacity=0.8, thickness=4, color="green")
```

Out[1]:

How frequently do 3D random walks return to the origin?

This next code cell generates a 3D random walk, stopping when it either returns to the origin or when it reaches some specified maximum number of steps.

If the random walk returns to the origin, then this function returns the number of steps that the random walk took to get back to the origin.

If the random walk does not return to the origin in **`maxSteps`** steps, then this function returns **`False`**.

In [2]:
```
# generate a 3D random walk until it returns to the origin or maxSteps
steps is reached
# return value:
#    number of steps, if random walk returns to the origin
#    False, if random walk does not return to the origin within maxSteps
steps
# note that this function does NOT store the list of locations visited
def stepsToReturn3D(maxSteps):
    # define the possible moves at each step of the random walk
    dirs = matrix( [[1,0,0],[-1,0,0],[0,1,0],[0,-1,0],[0,0,1],[0,0,-1]] )

    # initialize a vector to store the current location
    loc = vector([0,0,0])

    # take steps
    for i in range(maxSteps):
        r = randrange(6)
        move = dirs[r]  # direction to move
        loc = loc + move  # next location
        #print(loc)

        # is the random walk at the origin?
        if loc[0] == 0 and loc[1] == 0 and loc[2] == 0:
            return i

    # if we get here, then random walk did not return to the origin
    return False
```

In [4]:
```
stepsToReturn3D(20)
```

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

```
(0, -3, -3)
(0, -3, -4)
(0, -4, -4)
```

```
False
```

What proportion of 3D random walks return to the origin within 10 steps?

In [0]:

In [0]:

What proportion of 3D random walks return to the origin within 100 steps?

In [4]:
```
numWalks = 1000
results = [stepsToReturn3D(100) for i in range(numWalks)]
#print(results)
numReturn = sum( [1 for r in results if r > 0] )
numReturn/numWalks.n()
```

Out[4]: 0.308000000000000

In [0]:

What proportion of 3D random walks return to the origin within 1000 steps?

In [5]:
```
numWalks = 1000
results = [stepsToReturn3D(1000) for i in range(numWalks)]
#print(results)
numReturn = sum( [1 for r in results if r > 0] )
numReturn/numWalks.n()
```

Out[5]: 0.334000000000000

In [0]:

What proportion of 3D random walks return to the origin within 10,000 steps?

In [0]:

In [0]:

Try 10 million steps

In-class "distributed computing" activity

In [6]:
```
results = [stepsToReturn3D(10^7) for i in range(10)]
```

```
results
```

Out[7]:
```
[False, False, False, 115, False, False, False, False, 5, False]
```

```
nums = [2,3,2,1,0,3,3,5,1,4,4,4,3,3]
sum(nums)/(10*len(nums)).n()
```

Out[10]:
```
0.271428571428571
```

```

```