Kernel: SageMath 10.7

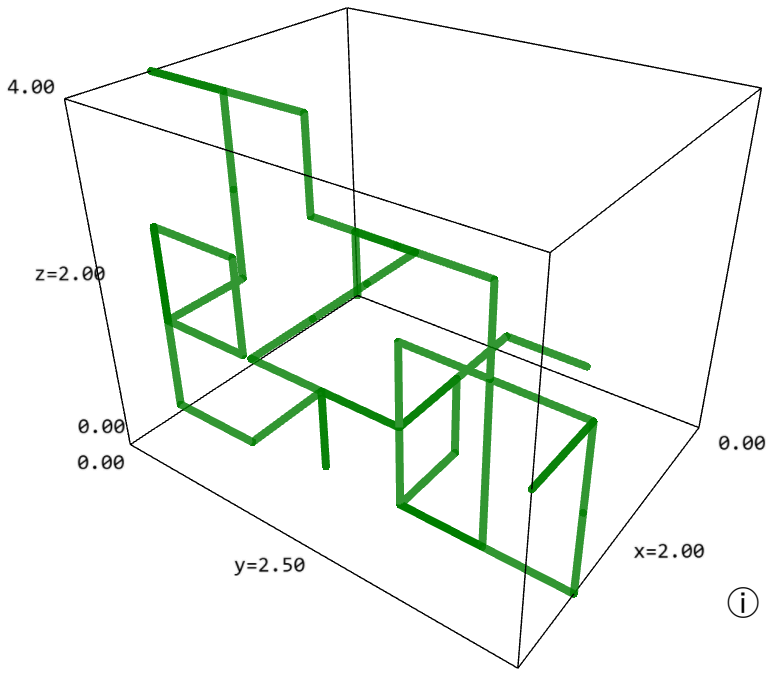
Three-Dimensional Random Walks

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 [2]:
```
# define the possible moves at each step of the random walk
dirs = [[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 = 50

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

# take steps
for i in range(1, numSteps+1):
    r = randrange(6)
    move = dirs[r]  # direction to move
    locations[i] = [locations[i-1][j] + move[j] for j in range(3)]  # next
location

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

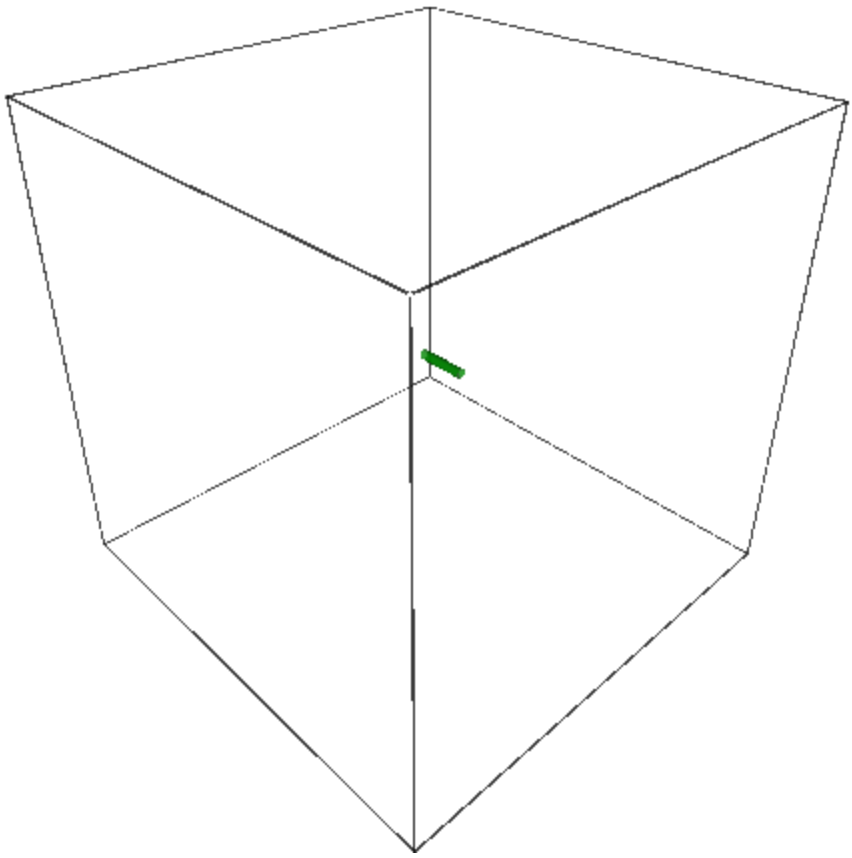
Out[2]:



You can also make an animation of the random walk using Sage. The next code cell creates a simple animation.

In [3]:
```
# define the number of steps
numSteps = 100

# define the possible moves at each step of the random walk
dirs = [[1,0,0],[-1,0,0],[0,1,0],[0,-1,0],[0,0,1],[0,0,-1]]

# set up a list to store the locations visited
locations = [[0,0,0]]

# define axis bounds
bound = 5

# function to generate each frame of the animation
def frame():
    global locations

    # make a move
    r = randrange(6)
    move = dirs[r]  # direction to move
    locations.append( [locations[-1][j] + move[j] for j in range(3)] )

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

    # add invisible corner points to specify the bounding box
    bbox = point3d([[-bound,-bound,-bound], [bound,bound,bound]], size=1,
opacity=0, color="white")

    return path + bbox

# now generate the animation
frames = [frame() for _ in range(numSteps) ]
anim = animate(frames)
anim.show(delay=20)
```

Out[3]:



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 [8]:

```
# function: stepsToReturn3D
# generates a 3D random walk until it returns to the origin or maxSteps steps is
reached
# input: max number of steps to simulate
# return:
#    number of steps to return to the origin, if random walk does return to the
origin
#    False, if random walk does not return to the origin within maxSteps steps
# note: 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 = [[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
```

```
    location = [0,0,0]

    # take steps
    for i in range(1, maxSteps+1):
        r = randrange(6)
        move = dirs[r]  # direction to move
        location = [location[j] + move[j] for j in range(3)]  # next location
        #print(f"{i}: {location}")

        # is the random walk at the origin?
        if location == [0,0,0]:
            return i

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

# testing
stepsToReturn3D(10)
```

Out[8]:
```
2
```

In [7]:
```
stepsToReturn3D(20)
```

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

4
```

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

In [14]:
```
#examples of integer-to-boolean comparisons
if 2:
    print("2 is true")
if not 0:
    print("0 is false")
```

Out[14]:
```
2 is true
0 is false
```

In [0]:
```

```

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

In [0]:
```

```

In [0]:
```

```

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

In [0]:

In [0]:

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

In [0]:

In [0]:

Do you think that 3D simple symmetric random walks return to the origin with 100% probabilty? Why or why not?

In [0]:

Try 10 million steps! Distributed computing activity

In [11]:
```
results = [ stepsToReturn3D(10^7) for i in range(10) ]
results
```

Out[11]: [False, 2, False, 2, False, False, 2, 6, False, False]

```
allResults = [4,4,1,3,4,3,6,4,3,3,4,3,2,4,5,5,6,4,5,3]
sum(allResults)/(10*len(allResults)).n()
```

Out[13]: 0.380000000000000