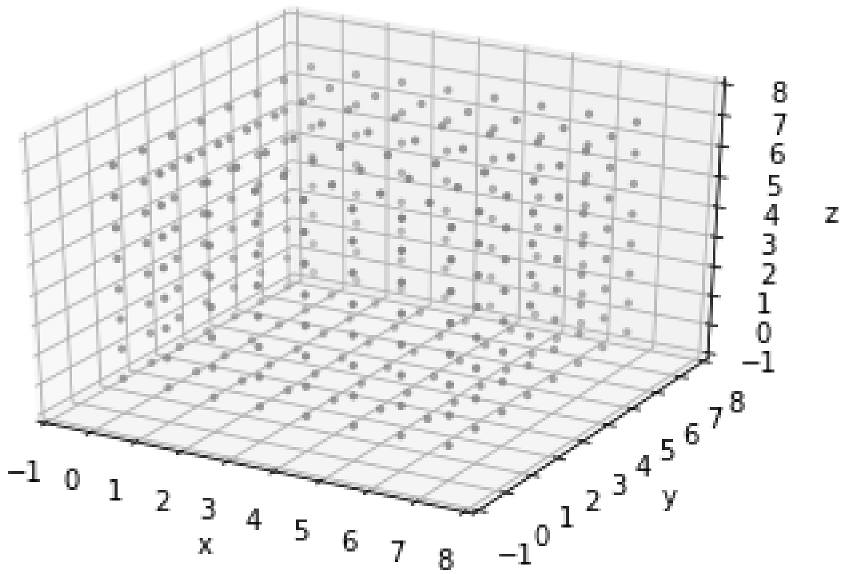
exercise-v

March 20, 2018

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from mpl_toolkits import mplot3d

        n = 8

        xc = n/2
        yc = n/2
        zc = n/2

In [2]: def is_exit(position): # test if position is an absorption site (set A union B)
            i, j, k = position
            return (i == 0 or j == 0 or k == 0 or i == n-1 or j == n-1
                    or is_good_exit(position))

        def is_good_exit(position): # test if position is a good absorption site (the set A)
            i, j, k = position
            return k == n-1

        ax = plt.axes(projection='3d')
        ax.set_xlim(-1, n)
        ax.set_ylim(-1, n)
        ax.set_zlim(-1, n)
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')

        ax.scatter([i for i in range(n) for j in range(n) for k in range(n) if is_exit( (i,j,k
                   [j for i in range(n) for j in range(n) for k in range(n) if is_exit( (i,j,k
                   [k for i in range(n) for j in range(n) for k in range(n) if is_exit( (i,j,k
                   color='grey', marker='.'
                   )
        plt.show()
```




```
In [3]: P = np.zeros( (n,n,n,n,n,n) )  # transition probabilities of the unconditional chain
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if not(is_exit( (i,j,k) )):
                        possible_moves = [(i+1,j,k), (i-1,j,k),
                                          (i,j+1,k), (i,j-1,k),
                                          (i,j,k+1), (i,j,k-1)]

                        for possible_move in possible_moves:
                            xnew, ynew, znew = possible_move
                            P[i,j,k,xnew,ynew,znew] = 1
                        s = np.sum(P[i,j,k,:,:,:])
                        P[i,j,k,:,:,:] = (1.0 / s) * P[i,j,k,:,:,:]
                    else:
                        P[i,j,k,i,j,k] = 1.0
```

```
In [4]: a = np.zeros( (n,n,n,n,n,n) ) # linear system, see the np.linalg.tensorsolve documenta
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if not(is_exit( (i,j,k) )):
                        a[i,j,k,:,:,:] = - P[i,j,k,:,:,:]

                    a[i,j,k,i,j,k] = 1

        b = np.zeros( (n,n,n) ) # boundary conditions: 1 for good exists and 0 for others.
```

```
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if is_good_exit( (i,j,k) ):
                        b[i,j,k] = 1.0


        good_exit_probabilities = np.linalg.tensorsolve(a, b)
```

```
In [5]: P_transformed = np.zeros_like(P)
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if not(is_exit( (i,j,k) )):
                        for x in range(n):
                            for y in range(n):
                                for z in range(n):
                                    P_transformed[i,j,k,x,y,z] = P[i,j,k,x,y,z] * good_exit_pro
                    else: # absorption once it reaches an exit
                        P_transformed[i,j,k,i,j,k] = 1.0
```

```
In [6]: class Point:
            def __init__(self, x, y, z):
                self.x = x
                self.y = y
                self.z = z

        all_points = [Point(i,j,k) for i in range(n) for j in range(n) for k in range(n)]

        # Run the walk and return the positions visited.
        def run_walk():
            x0 = xc
            y0 = yc
            z0 = 1

            current = Point(x0, y0, z0)
            x_visited = []
            y_visited = []
            z_visited = []
            while(True):
                current = np.random.choice(
                    all_points,
                    1, # return one random element
                    p=np.array([P_transformed[current.x, current.y, current.z, p.x, p.y, p.z] f
                          )[0]
                x_visited.append(current.x)
                y_visited.append(current.y)
                z_visited.append(current.z)
```

```
            if is_exit( (current.x, current.y, current.z) ):
                break

        return x_visited, y_visited, z_visited
```

In [7]:
```
ax = plt.axes(projection='3d')
ax.set_xlim(-1, n)
ax.set_ylim(-1, n)
ax.set_zlim(-1, n)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

def run_and_plot_walk(color):
    x_visited, y_visited, z_visited = run_walk()
    ax.plot(x_visited, y_visited, z_visited, color=color)

run_and_plot_walk('green')
run_and_plot_walk('red')
run_and_plot_walk('blue')
run_and_plot_walk('cyan')
run_and_plot_walk('yellow')

ax.scatter([i for i in range(n) for j in range(n) for k in range(n) if is_exit( (i,j,k)
           [j for i in range(n) for j in range(n) for k in range(n) if is_exit( (i,j,k)
           [k for i in range(n) for j in range(n) for k in range(n) if is_exit( (i,j,k)
           color='grey', marker='.'
           )
plt.show()
```