

Assignment 1 - Prison Break

Group number 9, 1796097, 1732722

1 Summary

This assignment provided a guided set of steps for us to be able to solve the famous "Narrow escape problem". In order to the first two tasks the documentation available online for the numpy function 'random.rand()' was investigated in order to provide the expected results. During the testing phase for task 3, the team would print out the 'get_random_radian(N)' function and using a calculator would check if the magnitude of the x and y values was equal to the expected step size of 0.5. The proper setup of the function was crucial for the rest of the assignment. Initially when dealing with one prisoner, the 'pos' array held all of the steps taken by the prisoner. However, when dealing with 500 prisoners, it was deemed inefficient to hold 500 arrays as the position history of the prisoners doesn't matter. However, later on, when it was asked to bind the prisoners to a particular domain, all that needed to be changed was to have a separate array containing the initial positions of the prisoners. When working on verifying the successful escape of the prisoners initially the code was set up in such a way that although it was vulnerable to false positives and negatives as mentioned in the assignment. Instead, the group chose to create a loop that would check if the straight line between the final and initial position and if it crossed the arc of the boundary, the escape would count as a success. However, it was observed that the second solution, although it was not vulnerable to false positives or negatives, was much slower when compared to the initial solution.

2 Results and discussion

1. After verifying the documentation it was discovered that the value had to be between 0 and 1 by applying function `numpy.random.rand()`. Thus by multiplying the function by 2π the team was able to generate a random radian ranging between 0 to 2π .
2. The documentation showed that 'random.rand()' can take input parameters to generate an array of any size within the range of 0 to 2π . This function was defined as seen in Listing 1. The output array of N random radians collected from function 'get random radian(N)' was displayed as a histogram as seen in Figure 1. The function to identify each N array's standard deviation and mean was also defined. In order to calculate the values, one can copy and run the code located in Listing A (codes/Task 2.py) Figure 1.
3. As mentioned in the summary, the task was completed by having the array pos hold all of the positions that the prisoner had. While this was changed for subsequent tasks, it was beneficial for generating an appropriate animated graph. During testing, the positions were printed and the distance between the points was verified to be the expected value 0.5. This helped ensure that no new mistakes were generated. In order to see that animation one can copy and run the code located

```
1 def get_random_radian(N):  
2     radian_array = 2 * np.pi * np.random.rand(N)  
3     return radian_array
```

Listing 1: Definition of get_random_radian(N)

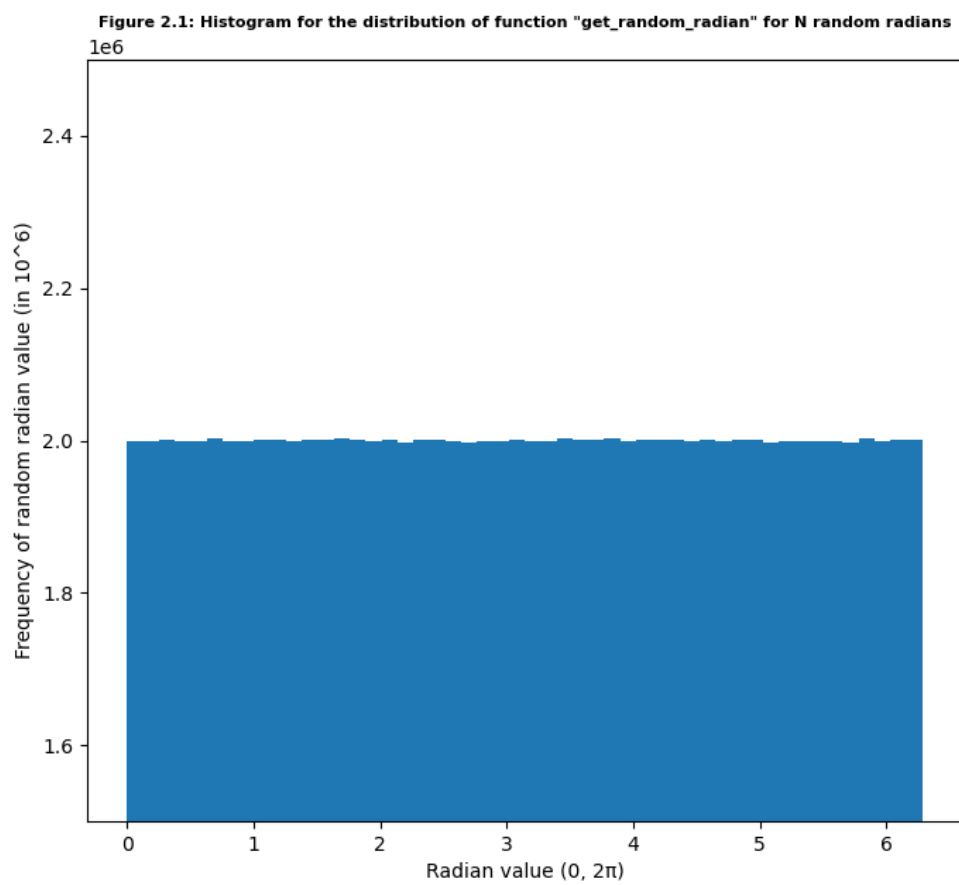


Figure 1: Distribution of the get_random_radian function

```

1 def new_step(N):
2     pos = np.zeros([N, 2])
3     rand_rad = get_random_radian(N) # To use the same direction for a
        pair of xy coordinates
4     x_values = np.cos(rand_rad) * step_size
5     y_values = np.sin(rand_rad) * step_size
6     rand_array = np.column_stack([x_values, y_values]) # Used to
        combine the x and y into seperate columns
7     pos = np.add(pos, rand_array)
8     return rand_array

```

Listing 2: Definition of new_step(N)

```

1 def mean_square_displacement_f(j):
2     displacement = np.array([])
3     for i in range(len(j[:, 0])):
4         displacement = np.append(displacement, np.linalg.norm(pos[i,
            :])**2)
5     mean_square_displacement = np.mean(displacement)
6     return mean_square_displacement

```

Listing 3: Definition of mean_square_displacement(j)

in [Listing A](#) (codes/Task 3.py). As can be seen in the code, first an array of $[0, 0]$ is generated, next a 'for' loop is utilized to generate a new step. Within the loop `np.vstack` is used to add a new row to the existing `pos` array. The animated graph was done by adapting code that was shown during the second lecture of the course.

4. The code had to be slightly altered to now only hold the current position of the 1000 prisoners. This not only allowed the team to make use of the fact that the function 'get random radian(N)' generates an array of size N but also reduced computation time if instead, the group chose to generate an array for each prisoner. A new function was defined which would generate a new step by converting the radian into an x and y component. The function can be seen in [Listing 2](#), however by running the full code found in [Listing A](#) (codes/Task 4.py) one can see the full animation. Moreover, the number of prisoners was assigned to a variable as this simplified testing and ensured proper readability of the code. The 2d histogram with the position of the prisoners can be seen in [Figure 2](#)
5. For this task a new function had to be defined that would find the current mean squared displacement and then calculate it for every step. This new function can be seen in [listing 3](#). Note that to ensure universality, the function was defined to accept any array with 2 columns. The expected diffusion coefficient was calculated by the formula given in the assignment and then was plotted on the same graph as the measured diffusion coefficients. This also verified the proper functionality of the code as the expected diffusion coefficient was always within 0.01 of the expected value.
6. For this task the group used the numpy function 'hist2d' to plot the 2D histograms. If statements were utilized to store the positions at various times. The times chosen were equally spaced and 5 intervals were selected as this properly illustrated the evolution of positions over time. The code for the loop can be seen in [Listing 4](#). While the code is not too elegant, it works correctly.
7. For this task, the group chose to include 5 graphs in equal time intervals to illustrate the pdf over time identical to those chosen in task 5. In order to graph the multivariable function the group used the knowledge gained from lecture 2. Then the histogram from task 6 was plotted on the same figure to compare the actual movement to the expected one. As the figure is partially in 3

Figure 4.2: 2D Histogram for the 500 step paths of 1000 prisoners

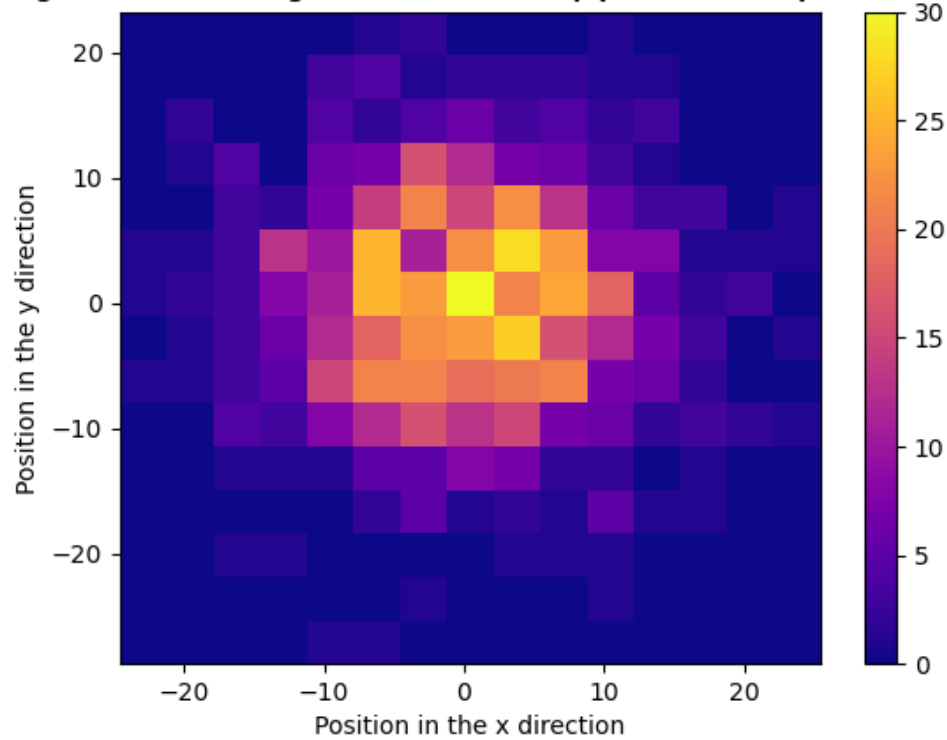


Figure 2: 2D Histogram of the positions of the prisoners after 500 steps

```

1 for i in range(Number_of_Steps):
2     pos = np.add(pos, new_step(Number_of_Prisoners))
3     if i == 99:
4         pos_after_100_steps = pos
5     if i == 199:
6         pos_after_200_steps = pos
7     if i == 299:
8         pos_after_300_steps = pos
9     if i == 399:
10        pos_after_400_steps = pos
11    current_d = mean_square_displacement_f(pos) / (2*
        Number_Of_Dimensions * (i+1))
12    dcoeff_vs_time[i, :] = [i + 1, current_d]
13    line41.set_data(pos[:, 0], pos[:, 1])
14    fig41.canvas.draw()
15    fig41.canvas.flush_events()
16    plt.pause(0.001)

```

Listing 4: Loop to store positions after a certain amount of steps

```

1  for n in range(len(pos[:,0])):
2      if np.linalg.norm(pos[n,:]) >= radius_of_bounds: # First check
3          to see if the new position is outside the bounds
4          pos[n,:] = pos_ini[n,:] # Go back to initial position
5          new_maybe_correct_step = new_step(1)
6          while np.linalg.norm(np.add(pos[n:],
7              new_maybe_correct_step)) >= radius_of_bounds: # Check
            to see if new step is outside the bounds
            new_maybe_correct_step = new_step(1)
            pos[n,:] = np.add(pos[n:], new_maybe_correct_step)

```

Listing 5: Loop that checks if the new position is within the given bounds

dimensions it is not included in this report, however, one can gather it by running the code from [Listing A](#) (codes/Task 7.py).

8. Having task 4 properly defined proved to be beneficial for the completion of this task. First and if statement was made which would iterate over every row of the pos array and check if the magnitude of the position vector was greater than 12. Then if the magnitude was greater than 12 a 'while' loop was implemented for the prisoner to look for a new step that would be within the domain. On the graph, the circular domain was plotted to visually verify that no prisoner spontaneously escaped however, as it is an animation, one must run the code from [Listing A](#) (codes/Task 8.py). Listing 5 shows the loop that was utilized.
9. This task proved to be far more interesting than it appeared. Initially, it was completed by first checking if the magnitude of the position vector was greater than 12 and subsequently laid within the region outside the wedge region. However, this was later improved upon to solve the cases of false positives and negatives. This was done by using sympy and defining the region of the hole as the first function and the straight line connecting the initial and final point as the second function. However, the first function was defined as the wedge and the for the domain of the wedge but 0 elsewhere for all positive values of x. Then, using 'sympy.solve()' the loop checked if and where the two lines met. Then if one of the intersections was on the wedge was counted as a success. This solved the problem if false positives and negatives, however, it did prove to be more computationally heavy. During testing the simulation was often run with less than 10 prisoners to ensure that it was completed in a reasonable amount of time. To reduce some computation a limited number of tries was given and the search for successful escapes was limited to a small region in which an escape could be possible. The code can be seen in listing 6. In order to make the simulation run until every prisoner escaped, a 'while' loop was utilized such that it continues until there are no more elements in the array pos. The histogram can be seen in [Figure 3](#).
10. The function was defined to accept inputs for radius, ϵ , Δt , and the number of dimensions. Next, in order to change the gap width, the angle was multiplied the an arbitrary counter ranging from 1 to 5. This would then run the simulation 5 times in increasing gap size to a quarter circle. The mean, median, and modes were calculated and then plotted on the same graph as the expected residence time. In order to print the graphs sooner. The while loop was exchanged for a 'for' loop to limit the number of steps a prisoner takes. When plotting the histograms it could be observed that after a certain number of steps, the number of successful escapes heavily drops. Thus by limiting the number of steps to a number such as 25000, the code could be executed much faster. The simulation was run with 1000 prisoners overnight as can be seen in [Figure 4](#). It is interesting to observe that the mean and median escape times are always much higher than the expected residence time. However, the mean escape time is nearly always lower than the expected residence time.
11. As was mentioned in the task 9, the edge cases were solved for this task. In order to verify the proper functionality of the new code, print statements were utilized. The system would print out

```

1  for n in range(len(pos[:,0])):
2      if np.linalg.norm(pos[n,:]) >= radius_of_bounds: # Check if
        they hit the boundry
3          if pos[n,0] > boundry_condition - 0.7: # Can be removed
        but its much slower without
4              slope_for_testing = (pos[n,1]-pos_ini[n,1])/(pos[n,0]-
        pos_ini[n,0])
5              y_intercept_for_testing = pos[n,1] - (
        slope_for_testing)*pos[n,0]
6              f2 = slope_for_testing * x_for_checking +
        y_intercept_for_testing ## Draw a straight line
        between new point and initial point
7              solution_to_be_tested = sympy.solve(f1 - f2,
        x_for_checking)
8              if len(solution_to_be_tested)==1 and
        solution_to_be_tested[0] >= boundry_condition:
9                  pos[n,:] = [13,2] ## 13, 2 is just an
        arbitrary position outside the domain and its a
        unique position and we know that we can remove a
        prisoner if their position is 13,2
10             elif len(solution_to_be_tested) == 2 and
        solution_to_be_tested[1] >= boundry_condition:
11                 pos[n,:] = [13,2]
12             else:
13                 pos[n,:] = pos_ini[n,:]
14                 new_maybe_correct_step = new_step(1)
15                 number_of_tries = 0 ## To make it a bit faster we
        only give them 5 tries to make a new move
        otherwise we send them back to their original
        position
16                 while (np.linalg.norm(np.add(pos[n, :],
        new_maybe_correct_step)) >= radius_of_bounds ):
17                     new_maybe_correct_step = new_step(1)
18                     number_of_tries = number_of_tries + 1
19                     if number_of_tries == 5:
20                         break
21                     if number_of_tries != 5:
22                         pos[n,:] = np.add(pos[n,:],
        new_maybe_correct_step)
23                     elif number_of_tries != 5:
24                         pos[n,:] = pos_ini[n,:]
25             else:
26                 pos[n,:] = pos_ini[n,:]
27                 new_maybe_correct_step = new_step(1)
28                 while (np.linalg.norm(np.add(pos[n, :],
        new_maybe_correct_step)) >= 12 ):
29                     new_maybe_correct_step = new_step(1)
30                 pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)

```

Listing 6: Loop to check if the current move was a success

Figure 9.1: Histogram of mean escape times of 1500 prisoners bounded by a fence with a gap from 0 to 0.1 radians

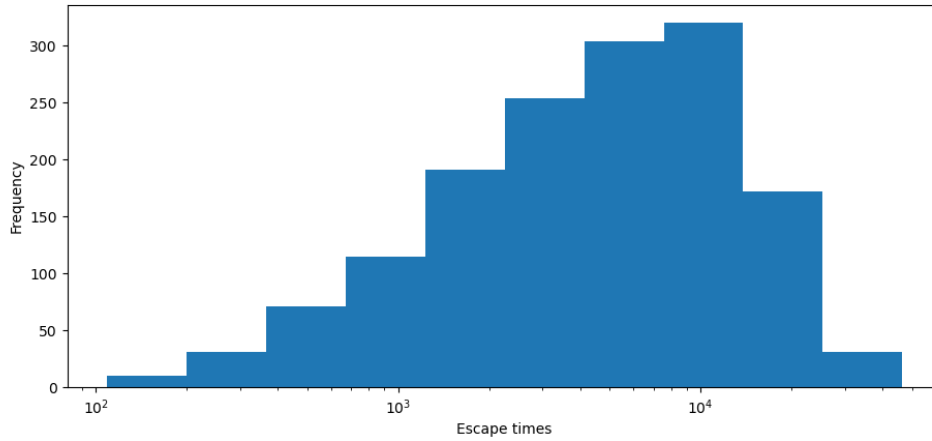


Figure 3: Histogram of 1500 prisoner escape times

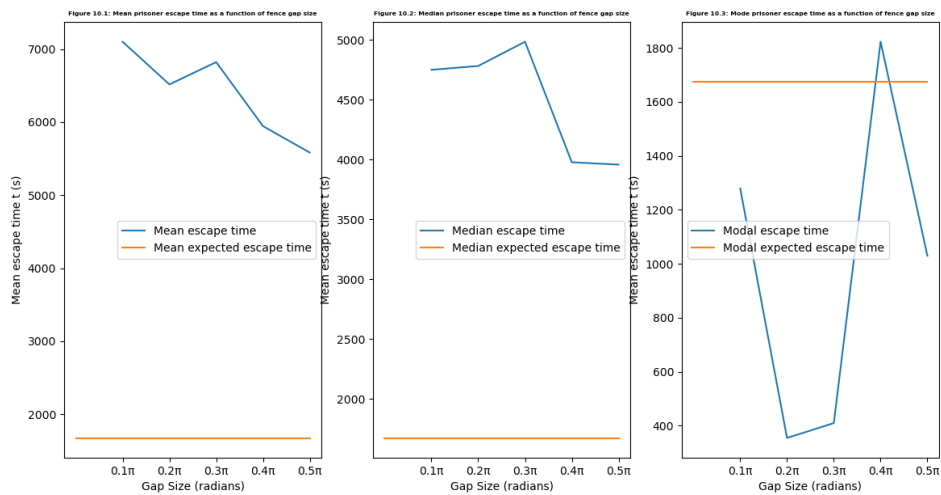


Figure 4: Mean, median, and mode, escape times vs expected escape time.

the initial position and the next step along with either 'Success' or 'Fail'. The points were then plotted using desmos which could then be used to visually verify the proper functionality of the code.

3 Reflection

This assignment helped the team gather information on using numpy arrays and how to use loops in python. The skill of debugging and verification was developed during the writing of the code. Most of the trouble shooting was done during task 9 as it was difficult to implement a solution that was able to correctly assign true positives and true negatives. To minimize the testing time, most code was written in a separate, simplified file which reduced computation time. Moreover, the use of online internet forums gave sometimes gave us guidance as to how a particular solution could be achieved. When working with loops, the group came upon an issue that accidentally created an infinite loop. This initially not detected however, after carefully reading every line of code, the error was found and fixed. Overall the assignment is a success however, the code runs quite slowly on when working with a large number of prisoners.

References

A Full code listings


```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing

```

codes/Task 1.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing
14
15 #Task 2
16 #From numpy.org Parameters:
17
18 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
19
20
21 #Add documentation and maybe add case where if not an integer give
   error
22 def get_random_radian(N):
23     '''Input must be an integer and output is an array of N columns
   with random radians'''
24     radian_array = 2 * np.pi * np.random.rand(N)
25     return radian_array
26
27
28 Random_radian_big_number = 10**8 #Bigger exponent = more time
29
30 fig21, ax21 = plt.subplots(1, 1, figsize=(8,7))
31 ax21.hist(get_random_radian(Random_radian_big_number), bins=50)
32
33 #To make graph nicer
34 ax21.set_title('Figure 2.1: Histogram for the distribution of function
   "get_random_radian" for N random radians', size=8, weight='bold')
35 ax21.set_xlabel('Radian value (0, 2 )')
36 ax21.set_ylabel('Frequency of random radian value (in 10^6)')
37 ax21.set_ylim([1.5*10**6, 2.5*10**6])
38
39 plt.show()
40 plt.savefig("plot2.png")
41 #Mean
42 Random_Radian_f_mean = np.mean(get_random_radian(
   Random_radian_big_number))
43
44 #Standard Deviation
45 Random_Radian_f_stddev = np.std(get_random_radian(
   Random_radian_big_number))

```

46

47

```
print('The random radian array of size',Random_radian_big_number, 'has  
a median of', Random_Radian_f_mean, 'and its standard deviation is  
, Random_Radian_f_stdev, '.')
```

codes/Task 2.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing
14
15 #Task 2
16 #From numpy.org Parameters:
17
18 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
19
20 #Add documentation and maybe add case where if not an integer give
   error
21 def get_random_radian(N):
22     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
23     return radian_array
24
25 # Prereq from previous tasks
26
27 step_size = 0.5
28
29 pos = np.array([[0, 0]])
30
31 def get_xy_velocities(N):
32     random_rad_function = get_random_radian(N)[0]
33     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
   .sin(random_rad_function) * step_size])
34     return random_x_y
35
36 #Check if magnitude = 0.5
37 #print(np.linalg.norm(get_xy_velocities(0))) #Norm does pythagoras
38
39 #Here range is the number of steps
40 #I found vstack which adds the new step to the position array as a new
   row
41 for N in range(1, 1001):
42     Current_Step = get_xy_velocities(N)
43     pos = np.vstack((pos, pos[-1, :] + Current_Step)) # New row = old
   row plus new step
44
45 #Extracting x and y coordinates
46

```

```

47 Task_3_x = pos[:,0]
48 Task_3_y = pos[:,1]
49
50 #Making the figure itself
51 fig31 = plt.figure(figsize=(8,7))
52 ax31 = plt.subplot(1, 1, 1)
53 line, = ax31.plot(Task_3_x, Task_3_y)
54
55 #Making the figure pretty
56 ax31.set_title('Figure 3.1: Animation for a random 1000 step path of a
    single prisoner', size=10, weight='bold')
57 ax31.set_xlabel('Prisoner position in x direction')
58 ax31.set_ylabel('Prisoner position in y direction')
59
60 plt.show(block=False)
61 for i in range(1000):
62     line.set_data(Task_3_x[i-20:i+1], Task_3_y[i-20:i+1])
63     fig31.canvas.draw()
64     fig31.canvas.flush_events()
65     plt.pause(0.0001)

```

codes/Task 3.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing
14
15 #Task 2
16 #From numpy.org Parameters:
17
18 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
19
20
21 #Add documentation and maybe add case where if not an integer give
   error
22 def get_random_radian(N):
23     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
24     return radian_array
25
26
27
28 step_size = 0.5
29
30 pos = np.array([[0, 0]])
31
32 def get_xy_velocities(N):
33     random_rad_function = get_random_radian(N)[0]
34     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
   .sin(random_rad_function) * step_size])
35     return random_x_y
36
37
38 # Prereq from previous tasks
39
40 #Task 4
41
42 def new_step(N):
43     pos = np.zeros([N, 2])
44     rand_rad = get_random_radian(N) # To use the same direction for a
   pair of xy coordinates
45     x_values = np.cos(rand_rad) * step_size
46     y_values = np.sin(rand_rad) * step_size
47     rand_array = np.column_stack([x_values, y_values]) # Used to

```

```

47         combine the x and y into seperate columns
48     pos = np.add(pos, rand_array)
49     return rand_array
50
51
52 Number_of_Steps = 500
53 Number_of_Prisoners = 1000
54
55 pos=np.zeros([Number_of_Prisoners, 2])
56
57
58 #Task 4 graph 1
59 fig41, ax41 = plt.subplots(figsize=(8,7))
60 line41, = ax41.plot([], [], 'o')
61
62 #Making it pretty
63
64 plt.show(block=False)
65 ax41.set_xlim(-50, 50)
66 ax41.set_ylim(-50,50)
67 ax41.set_title('Figure 4.1: Animation for the random 500 step paths of
68               1000 prisoners', size=10, weight='bold')
69 ax41.set_xlabel('Prisoners position in x direction')
70 ax41.set_ylabel('Prisoners position in y direction')
71
72 for i in range(Number_of_Steps):
73     pos = np.add(pos, new_step(Number_of_Prisoners))
74     line41.set_data(pos[:, 0], pos[:, 1])
75     fig41.canvas.draw()
76     fig41.canvas.flush_events()
77     plt.pause(0.001)
78
79
80
81 #I assume that a 2d histogram is just a heatmap
82
83 #For plotting
84
85 fig42, ax42 = plt.subplots()
86
87
88 hist2d42 = ax42.hist2d(pos[:, 0], pos[:, 1], bins=15, cmap=cm.plasma)
89 plt.colorbar(hist2d42[3], ax=ax42)
90
91 ax42.set_xlabel('Position in the x direction')
92 ax42.set_ylabel('Position in the y direction')
93 ax42.set_title('Figure 4.2: 2D Histogram for the 500 step paths of
94               1000 prisoners', size=10, weight='bold')
95
96 plt.show()

```

codes/Task 4.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing
14
15 #Task 2
16 #From numpy.org Parameters:
17
18 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
19
20
21 #Add documentation and maybe add case where if not an integer give
   error
22 def get_random_radian(N):
23     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
24     return radian_array
25
26
27
28 step_size = 0.5
29
30 pos = np.array([[0, 0]])
31
32 def get_xy_velocities(N):
33     random_rad_function = get_random_radian(N)[0]
34     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
   .sin(random_rad_function) * step_size])
35     return random_x_y
36
37
38
39
40 #Task 4
41
42 def new_step(N):
43     pos = np.zeros([N, 2])
44     rand_rad = get_random_radian(N) # To use the same direction for a
   pair of xy coordinates
45     x_values = np.cos(rand_rad) * step_size
46     y_values = np.sin(rand_rad) * step_size
47     rand_array = np.column_stack([x_values, y_values]) # Used to

```



```

47         combine the x and y into seperate columns
48     pos = np.add(pos, rand_array)
49     return rand_array
50
51
52 Number_of_Steps = 500
53 Number_of_Prisoners = 1000
54
55 pos=np.zeros([Number_of_Prisoners, 2])
56
57
58 #Task 4 graph 1
59
60 #Comment out the lines below to hide the graph
61 fig41, ax41 = plt.subplots()
62 line41, = ax41.plot([], [], 'o')
63 plt.show(block=False)
64 ax41.set_xlim(-50, 50)
65 ax41.set_ylim(-50,50)
66 ax41.set_title('Path of 1000 prisoners after 500 steps')
67 ax41.set_xlabel('x position')
68 ax41.set_ylabel('y position')
69
70
71
72
73
74
75
76
77 #Prereq from previous tasks
78
79 def mean_square_displacement_f(j):
80     displacement = np.array([])
81     for i in range(len(j[:, 0])):
82         displacement = np.append(displacement, np.linalg.norm(pos[i,
83         :])*2)
84     mean_square_displacement = np.mean(displacement)
85     return mean_square_displacement
86
87 #For task 5
88 dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
89 Number_Of_Dimensions = 2
90
91 for i in range(Number_of_Steps):
92     pos = np.add(pos, new_step(Number_of_Prisoners))
93     current_d = mean_square_displacement_f(pos) / (2*
94         Number_Of_Dimensions * (i+1))
95     dcoeff_vs_time[i, :] = [i + 1, current_d]
96     line41.set_data(pos[:, 0], pos[:, 1])
97     fig41.canvas.draw()
98     fig41.canvas.flush_events()
99     plt.pause(0.001)

```

```

99 #Also comment the lines above to hide the graph
100
101
102 x_values_for_task_5 = np.linspace(0, 500, 100)
103 expected_diffusion_coeff = (step_size**2)/(2*Number_Of_Dimensions* 1)
    + 0*x_values_for_task_5
104
105 fig51, ax51 = plt.subplots()
106 ax51.scatter(dcoeff_vs_time[:,0], dcoeff_vs_time[:,1])
107 ax51.plot(x_values_for_task_5, expected_diffusion_coeff, label='
    Expected Diffusion Coefficient')
108 #Make the label for the straight line show
109
110
111 #Making ax51 pretty
112 ax51.set_xlabel('Number of Steps')
113 ax51.set_ylabel('Diffusion Coefficient')
114 ax51.set_title('Expected Diffusion Coefficient vs number of steps')
115
116
117
118 plt.show()

```

codes/Task 5.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing
14
15 #Task 2
16 #From numpy.org Parameters:
17
18 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
19
20
21 #Add documentation and maybe add case where if not an integer give
   error
22 def get_random_radian(N):
23     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
24     return radian_array
25
26 step_size = 0.5
27
28 pos = np.array([[0, 0]])
29
30 def get_xy_velocities(N):
31     random_rad_function = get_random_radian(N)[0]
32     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
   .sin(random_rad_function) * step_size])
33     return random_x_y
34
35
36
37
38 #Task 4
39
40 def new_step(N):
41     pos = np.zeros([N, 2])
42     rand_rad = get_random_radian(N) # To use the same direction for a
   pair of xy coordinates
43     x_values = np.cos(rand_rad) * step_size
44     y_values = np.sin(rand_rad) * step_size
45     rand_array = np.column_stack([x_values, y_values]) # Used to
   combine the x and y into seperate columns
46     pos = np.add(pos, rand_array)

```

```

47     return rand_array
48
49
50 Number_of_Steps = 500
51 Number_of_Prisoners = 1000
52
53 pos=np.zeros([Number_of_Prisoners, 2])
54
55
56
57
58 #Comment out the lines below to hide the graph
59 fig41, ax41 = plt.subplots()
60 line41, = ax41.plot([], [], 'o')
61 plt.show(block=False)
62 ax41.set_xlim(-50, 50)
63 ax41.set_ylim(-50,50)
64 ax41.set_title('Path of 1000 prisoners after 500 steps')
65 ax41.set_xlabel('x position')
66 ax41.set_ylabel('y position')
67
68
69
70
71
72
73 #Prereq from previous tasks
74
75 def mean_square_displacement_f(j):
76     displacement = np.array([])
77     for i in range(len(j[:, 0])):
78         displacement = np.append(displacement, np.linalg.norm(pos[i,
79 :])*2)
80     mean_square_displacement = np.mean(displacement)
81     return mean_square_displacement
82
83 #For task 5
84 dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
85 Number_Of_Dimensions = 2
86
87 for i in range(Number_of_Steps):
88     pos = np.add(pos, new_step(Number_of_Prisoners))
89     if i == 99:
90         pos_after_100_steps = pos
91     if i == 199:
92         pos_after_200_steps = pos
93     if i == 299:
94         pos_after_300_steps = pos
95     if i == 399:
96         pos_after_400_steps = pos
97     current_d = mean_square_displacement_f(pos) / (2*
98         Number_Of_Dimensions * (i+1))
99     dcoeff_vs_time[i, :] = [i + 1, current_d]

```

```

99     line41.set_data(pos[:, 0], pos[:, 1])
100     fig41.canvas.draw()
101     fig41.canvas.flush_events()
102     plt.pause(0.001)
103     #Also comment the lines above to hide the graph
104
105
106
107
108     #Prereq
109
110     #Task 6
111     #Again the naming, first number is the task number and the second is
        the number of the graph
112
113     fig61 = plt.figure(figsize=(18,8))
114
115     ax61 = plt.subplot(2, 4, 1)
116     hist2d61 = ax61.hist2d(pos_after_100_steps[:, 0], pos_after_100_steps
       [:, 1], bins=15, cmap=cm.plasma)
117     ax61.set_title('Figure 6.1: Histogram for prisoner positions after 100
        steps', size=7, weight='bold')
118     plt.colorbar(hist2d61[3], ax=ax61)
119
120
121     ax62 = plt.subplot(2, 4, 2)
122     hist2d62 = ax62.hist2d(pos_after_200_steps[:, 0], pos_after_200_steps
       [:, 1], bins=15, cmap=cm.plasma)
123     ax62.set_title('Figure 6.2: Histogram for prisoner positions after 200
        steps', size=7, weight='bold')
124     plt.colorbar(hist2d62[3], ax=ax62)
125
126
127     ax63 = plt.subplot(2, 4, 5)
128     hist2d63 = ax63.hist2d(pos_after_300_steps[:, 0], pos_after_300_steps
       [:, 1], bins=15, cmap=cm.plasma)
129     ax63.set_title('Figure 6.3: Histogram for prisoner positions after 300
        steps', size=7, weight='bold')
130     plt.colorbar(hist2d63[3], ax=ax63)
131
132
133     ax64 = plt.subplot(2, 4, 6)
134     hist2d64 = ax64.hist2d(pos_after_400_steps[:, 0], pos_after_400_steps
       [:, 1], bins=15, cmap=cm.plasma)
135     ax64.set_title('Figure 6.4: Histogram for prisoner positions after 400
        steps', size=7, weight='bold')
136     plt.colorbar(hist2d64[3], ax=ax64)
137
138
139     ax65 = plt.subplot(1, 2, 2)
140     hist2d65 = ax65.hist2d(pos[:, 0], pos[:, 1], bins=15, cmap=cm.plasma)
141     ax65.set_title('Figure 6.5: Histogram for prisoner positions after the
        last step', size=12, weight='bold')
142     plt.colorbar(hist2d65[3], ax=ax65)

```

143
144
145
146

```
plt.show()
```

codes/Task 6.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing
14
15 #Task 2
16 #From numpy.org Parameters:
17
18 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
19
20
21 #Add documentation and maybe add case where if not an integer give
   error
22 def get_random_radian(N):
23     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
24     return radian_array
25
26
27
28 step_size = 0.5
29
30 pos = np.array([[0, 0]])
31
32 def get_xy_velocities(N):
33     random_rad_function = get_random_radian(N)[0]
34     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
   .sin(random_rad_function) * step_size])
35     return random_x_y
36
37
38 # Prereq from previous tasks
39
40 #Task 4
41
42 def new_step(N):
43     pos = np.zeros([N, 2])
44     rand_rad = get_random_radian(N) # To use the same direction for a
   pair of xy coordinates
45     x_values = np.cos(rand_rad) * step_size
46     y_values = np.sin(rand_rad) * step_size
47     rand_array = np.column_stack([x_values, y_values]) # Used to

```

```

48         combine the x and y into seperate columns
49         pos = np.add(pos, rand_array)
50         return rand_array
51
52 Number_of_Steps = 500
53 Number_of_Prisoners = 1000
54
55 pos=np.zeros([Number_of_Prisoners, 2])
56
57
58 #Task 4 graph 1
59
60 #Uncomment the bottom lines if you want to see the graph of prisoners
61 #fig41, ax41 = plt.subplots()
62 #line41, = ax41.plot([], [], 'o')
63 #plt.show(block=False)
64 #ax41.set_xlim(-50, 50)
65 #ax41.set_ylim(-50,50)
66 #ax41.set_title('Path of 1000 prisoners after 500 steps')
67 #ax41.set_xlabel('x position')
68 #ax41.set_ylabel('y position')
69
70
71
72
73
74
75
76
77 #Prereq from previous tasks
78
79 def mean_square_displacement_f(j):
80     displacement = np.array([])
81     for i in range(len(j[:, 0])):
82         displacement = np.append(displacement, np.linalg.norm(pos[i,
83             :])*2)
84     mean_square_displacement = np.mean(displacement)
85     return mean_square_displacement
86
87 #For task 5
88 dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
89 Number_Of_Dimensions = 2
90
91 for i in range(Number_of_Steps):
92     pos = np.add(pos, new_step(Number_of_Prisoners))
93     if i == 99:
94         pos_after_100_steps = pos #Kinda ugly but it works
95     if i == 199:
96         pos_after_200_steps = pos
97     if i == 299:
98         pos_after_300_steps = pos
99     if i == 399:

```



```

100     pos_after_400_steps = pos
101     current_d = mean_square_displacement_f(pos) / (2*
        Number_Of_Dimensions * (i+1))
102     dcoeff_vs_time[i, :] = [i + 1, current_d]
103     # line41.set_data(pos[:, 0], pos[:, 1])
104     # fig41.canvas.draw()
105     # fig41.canvas.flush_events()
106     # plt.pause(0.001)
107     #Also uncomment the lines above to see
108
109
110
111     #Prereqs
112
113     t=100
114
115     x = np.arange(-15, 15, 0.005)
116     y = np.arange(-15, 15, 0.005)
117
118     x,y = np.meshgrid(x, y)
119
120
121
122     expected_diffusion_coeff_numerically = 0.0625
123
124     z = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) * np.exp
        (-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t))
125     z1 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) * np.
        exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*2)
        )
126     z2 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) * np.
        exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*3)
        )
127     z3 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) * np.
        exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*4)
        )
128     z4 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) * np.
        exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*5)
        )
129
130
131     fig71 = plt.figure(figsize=(18,8))
132     ax71 = fig71.add_subplot(2, 3, 1, projection='3d')
133     ax71.set_title('Figure 7.1: Probability density function 3D projection
        at 100 steps', size=6, weight='bold')
134     ax71.set_xlabel('x')
135     ax71.set_ylabel('y')
136     ax71.set_zlabel('pdf')
137     ax72 = fig71.add_subplot(2, 3, 2, projection='3d')
138     ax72.set_title('Figure 7.2: Probability density function 3D projection
        at 200 steps', size=6, weight='bold')
139     ax72.set_xlabel('x')
140     ax72.set_ylabel('y')
141     ax72.set_zlabel('pdf')

```

```

142 ax73 = fig71.add_subplot(2, 3, 3, projection='3d')
143 ax73.set_title('Figure 7.3: Probability density function 3D projection
    at 300 steps', size=6, weight='bold')
144 ax73.set_xlabel('x')
145 ax73.set_ylabel('y')
146 ax73.set_zlabel('pdf')
147 ax74 = fig71.add_subplot(2, 3, 4, projection='3d')
148 ax74.set_title('Figure 7.4: Probability density function 3D projection
    at 400 steps', size=6, weight='bold')
149 ax74.set_xlabel('x')
150 ax74.set_ylabel('y')
151 ax74.set_zlabel('pdf')
152 ax75 = fig71.add_subplot(2, 3, 5, projection='3d')
153 ax75.set_title('Figure 7.5: Probability density function 3D projection
    at 500 steps', size=6, weight='bold')
154 ax75.set_xlabel('x')
155 ax75.set_ylabel('y')
156 ax75.set_zlabel('pdf')
157
158
159 ax76 = plt.subplot(2, 3, 6)
160 hist2d65 = ax76.hist2d(pos[:, 0], pos[:, 1], bins=15, cmap=cm.plasma)
161 ax76.set_title('Figure 7.6: Histogram for prisoner positions after the
    last step', size=6, weight='bold')
162 ax76.set_xlabel('x')
163 ax76.set_ylabel('y')
164 ax76.set_xlim(-15, 15)
165 ax76.set_ylim(-15,15)
166
167 plt.colorbar(hist2d65[3], ax=ax76)
168
169
170
171
172
173 surf1 = ax71.plot_surface(x,y,z, cmap=cm.magma, linewidth=0,
    antialiased=0)
174 surf2 = ax72.plot_surface(x,y,z1, cmap=cm.magma, linewidth=0,
    antialiased=0)
175 surf3 = ax73.plot_surface(x,y,z2, cmap=cm.magma, linewidth=0,
    antialiased=0)
176 surf4 = ax74.plot_surface(x,y,z3, cmap=cm.magma, linewidth=0,
    antialiased=0)
177 surf5 = ax75.plot_surface(x,y,z4, cmap=cm.magma, linewidth=0,
    antialiased=0)
178
179
180 plt.show()

```

codes/Task 7.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 # Task 1
8 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
9 #We can just multiply it by 2pi
10
11 Random_N_2pi = 2 * np.pi * np.random.rand()
12
13 # print(Random_N_2pi) # <- For testing
14
15 #Task 2
16 #From numpy.org Parameters:
17
18 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
19
20
21 #Add documentation and maybe add case where if not an integer give
   error
22 def get_random_radian(N):
23     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
24     return radian_array
25
26
27
28 step_size = 0.5
29
30 pos = np.array([[0, 0]])
31
32 def get_xy_velocities(N):
33     random_rad_function = get_random_radian(N)[0]
34     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
   .sin(random_rad_function) * step_size])
35     return random_x_y
36
37
38
39 def new_step(N):
40     pos = np.zeros([N, 2])
41     rand_rad = get_random_radian(N) # To use the same direction for a
   pair of xy coordinates
42     x_values = np.cos(rand_rad) * step_size
43     y_values = np.sin(rand_rad) * step_size
44     rand_array = np.column_stack([x_values, y_values]) # Used to
   combine the x and y into seperate columns
45     pos = np.add(pos, rand_array)
46     return rand_array

```

```

47
48
49 Number_of_Steps = 500
50 Number_of_Prisoners = 1000
51
52 pos=np.zeros([Number_of_Prisoners, 2])
53
54
55 #Task 4 graph 1
56
57 #Uncomment the bottom lines if you want to see the graph of prisoners
58 #fig41, ax41 = plt.subplots()
59 #line41, = ax41.plot([], [], 'o')
60 #plt.show(block=False)
61 #ax41.set_xlim(-50, 50)
62 #ax41.set_ylim(-50,50)
63 #ax41.set_title('Path of 1000 prisoners after 500 steps')
64 #ax41.set_xlabel('x position')
65 #ax41.set_ylabel('y position')
66
67
68
69
70
71
72
73
74
75 def mean_square_displacement_f(j):
76     displacement = np.array([])
77     for i in range(len(j[:, 0])):
78         displacement = np.append(displacement, np.linalg.norm(pos[i,
79 :])*2)
80     mean_square_displacement = np.mean(displacement)
81     return mean_square_displacement
82
83 #For task 5
84 dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
85 Number_Of_Dimensions = 2
86
87 #Prereqs
88
89 #Copied but modified from task 4
90 fig81, ax81 = plt.subplots()
91 line81, = ax81.plot([], [], 'o')
92
93 #Making it pretty
94 plt.show(block=False)
95 ax81.set_xlim(-20, 20)
96 ax81.set_ylim(-20,20)
97 ax81.set_title('Figure 8.1: Animation for the random 500 step paths of
1000 prisoners with bounds')
98 ax81.set_xlabel('Prisoners position in x direction')
99 ax81.set_ylabel('Prisoners position in y direction')

```

```

99
100
101 radius_of_bounds = 12
102
103
104 x1_for_8 = np.linspace(-radius_of_bounds, radius_of_bounds, 10**4)
105 y1_for_8 = np.sqrt(radius_of_bounds**2 - (x1_for_8**2))
106 y2_for_8 = -1* np.sqrt(radius_of_bounds**2 - (x1_for_8**2))
107 ax81.plot(x1_for_8, y1_for_8, "r-")
108 ax81.plot(x1_for_8, y2_for_8, "r-")
109
110
111 plt.show(block=False)
112 pos=np.zeros([Number_of_Prisoners, 2])
113 pos = pos + 0.0
114
115 for i in range(Number_of_Steps):
116     pos_ini = pos.copy()
117     pos = np.add(pos, new_step(Number_of_Prisoners))
118     for n in range(len(pos[:,0])):
119         if np.linalg.norm(pos[n,:]) >= radius_of_bounds: # First check
120             # to see if the new position is outside the bounds
121             pos[n,:] = pos_ini[n,:] # Go back to initial position
122             new_maybe_correct_step = new_step(1)
123             while np.linalg.norm(np.add(pos[n:],
124                 new_maybe_correct_step)) >= radius_of_bounds: # Check
125                 # to see if new step is outside the bounds
126                 new_maybe_correct_step = new_step(1)
127             pos[n,:] = np.add(pos[n:], new_maybe_correct_step)
128     line81.set_data(pos[:, 0], pos[:, 1])
129     fig81.canvas.draw()
130     fig81.canvas.flush_events()
131     plt.pause(0.001)
132
133 # Make it so that the graph prints as a square so that its clear that
134 the boundary is a circle and not an oval

```

codes/Task 8.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 import sympy
8 # Task 1
9 #You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
10 #We can just multiply it by 2pi
11
12 Random_N_2pi = 2 * np.pi * np.random.rand()
13
14 # print(Random_N_2pi) # <- For testing
15
16 #Task 2
17 #From numpy.org Parameters:
18
19 #    d0, d1, ..., dnint, optional The dimensions of the returned array
   , must be non-negative. If no argument is given a single Python
   float is returned.
20
21
22 #Add documentation and maybe add case where if not an integer give
   error
23 def get_random_radian(N):
24     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
25     return radian_array
26
27
28
29 step_size = 0.5
30
31 pos = np.array([[0, 0]])
32
33 def get_xy_velocities(N):
34     random_rad_function = get_random_radian(N)[0]
35     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
   .sin(random_rad_function) * step_size])
36     return random_x_y
37
38
39 # Prereq from previous tasks
40
41 #Task 4
42
43 def new_step(N):
44     pos = np.zeros([N, 2])
45     rand_rad = get_random_radian(N) # To use the same direction for a
   pair of xy coordinates
46     x_values = np.cos(rand_rad) * step_size
47     y_values = np.sin(rand_rad) * step_size

```

```

48     rand_array = np.column_stack([x_values, y_values]) # Used to
        combine the x and y into seperate columns
49     pos = np.add(pos, rand_array)
50     return rand_array
51
52
53 #Number_of_Steps = 500
54 #Number_of_Prisoners = 1000
55 #pos=np.zeros([Number_of_Prisoners, 2])
56 #Relics from previous tasks
57
58 #Task 4 graph 1
59
60 #Uncomment the bottom lines if you want to see the graph of prisoners
61 #fig41, ax41 = plt.subplots()
62 #line41, = ax41.plot([], [], 'o')
63 #plt.show(block=False)
64 #ax41.set_xlim(-50, 50)
65 #ax41.set_ylim(-50,50)
66 #ax41.set_title('Path of 1000 prisoners after 500 steps')
67 #ax41.set_xlabel('x position')
68 #ax41.set_ylabel('y position')
69
70 #For task 5
71
72 Number_Of_Dimensions = 2
73
74
75
76
77 # Prereqs
78
79
80
81 radius_of_bounds = 12
82 boundry_condition = radius_of_bounds*np.cos(0.1*np.pi)
83
84 #Comment out the lines below to print the histogram faster
85 fig91, ax91 = plt.subplots()
86 line91, = ax91.plot([], [], 'o')
87 ax91.set_xlim(-20, 20)
88 ax91.set_ylim(-20,20)
89 x1_for_9 = np.linspace(-radius_of_bounds,radius_of_bounds,10**4)
90 x2_for_9 = np.linspace(-radius_of_bounds, boundry_condition, 10**4)
91 y1_for_9 = np.sqrt(radius_of_bounds**2 - (x2_for_9**2))
92 y2_for_9 = -1* np.sqrt(radius_of_bounds**2 - (x1_for_9**2))
93 ax91.plot(x2_for_9, y1_for_9, "r-")
94 ax91.plot(x1_for_9, y2_for_9, "r-")
95 ax91.set_ylabel('Position in y')
96 ax91.set_xlabel('Position in x')
97
98
99
100 escape_times = []

```

```

101 to_be_removed = []
102 Number_of_Prisoners = 10 ## Set to ten now but you can change it to a
    different number, bigger number = much slower
103 #Number_of_Steps = 1 # Since were running it until they leave we dont
    know how many steps it will take
104
105 ## Also comment out this line
106 ax91.set_title(f'Movement of {Number_of_Prisoners} prisoners in
    circular domain of radius {radius_of_bounds} with a small gap')
107
108
109 pos=np.zeros([Number_of_Prisoners, 2])
110 step_number = 0
111
112 # This one is quicker but its vulnerable to the edge case as described
    in the assignment document figure 1
113
114 #Uncomment it to see it
115 #while len(pos[:,0]) > 0:
116 #     step_number = step_number + 1
117 #     for p in range(len(pos[:, 0])):
118 #         if pos [p, 0] == 13 and pos[p,1] == 2:
119 #             to_be_removed.append(p)
120 #             escape_times.append(step_number-1)
121 #     pos = np.delete(pos, to_be_removed, axis=0)
122 #     to_be_removed = []
123 #     pos_ini = pos.copy()
124 #     pos = np.add(pos, new_step(len(pos[:,0])))
125 #     for n in range(len(pos[:,0])):
126 #         if np.linalg.norm(pos[n,:]) >= radius_of_bounds:
127 #             if pos[n, 1] >= 0 and boundry_condition <= pos[n, 0]:
128 #                 pos[n,:] = [13,2]
129 #             else:
130 #                 pos[n,:] = pos_ini[n,:]
131 #                 new_maybe_correct_step = new_step(1)
132 #                 while (np.linalg.norm(np.add(pos[n, :],
    new_maybe_correct_step)) >= radius_of_bounds ):
133 #                     new_maybe_correct_step = new_step(1)
134 #                     pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
135 #     line91.set_data(pos[:, 0], pos[:, 1])
136 #     fig91.canvas.draw()
137 #     fig91.canvas.flush_events()
138 #     plt.pause(0.0001)
139
140
141 # The one below solves the edge case but its a bit slower
142
143
144 #Check if correctly crosses the border
145 x_for_checking = sympy.symbols('x', positive=True)
146
147 boundry_condition_for_checking = radius_of_bounds*sympy.cos(0.1*sympy.
    pi)
148

```



```

149 f1 = sympy.Piecewise((sympy.sqrt(radius_of_bounds**2 - x_for_checking
150 **2), x_for_checking >= boundry_condition_for_checking), (0, True))
151 # To change number of prisoners use the code that a couple of lines
152 # above
153 # Its quite slow with # of prisoners > 10000
154 while len(pos[:,0]) > 0: # This makes it so that the code keeps
155     running as long as there is a prisoner inside the bounds
156     step_number = step_number + 1
157     for p in range(len(pos[:, 0])): ## Check if any prisoner is in
158         13, 2 and remove it
159         if pos [p, 0] == 13 and pos[p,1] == 2:
160             to_be_removed.append(p)
161             escape_times.append(step_number-1)
162             pos = np.delete(pos, to_be_removed, axis=0)
163             to_be_removed = []
164             pos_ini = pos.copy()
165             pos = np.add(pos, new_step(len(pos[:,0])))
166             for n in range(len(pos[:,0])):
167                 if np.linalg.norm(pos[n,:]) >= radius_of_bounds: # Check if
168                     they hit the boundry
169                     if pos[n,0] > boundry_condition - 0.7: # Can be removed
170                         but its much slower without
171                         slope_for_testing = (pos[n,1]-pos_ini[n,1])/(pos[n,0]-
172                             pos_ini[n,0])
173                         y_intercept_for_testing = pos[n,1] - (
174                             slope_for_testing)*pos[n,0]
175                         f2 = slope_for_testing * x_for_checking +
176                             y_intercept_for_testing ## Draw a straight line
177                             between new point and initial point
178                         solution_to_be_tested = sympy.solve(f1 - f2,
179                             x_for_checking)
180                         if len(solution_to_be_tested)==1 and
181                             solution_to_be_tested[0] >= boundry_condition:
182                             pos[n,:] = [13,2] ## 13, 2 is just an
183                                 arbitrary position outside the domain and its a
184                                 unique position and we know that we can remove a
185                                 prisoner if their position is 13,2
186                         elif len(solution_to_be_tested) == 2 and
187                             solution_to_be_tested[1] >= boundry_condition:
188                             pos[n,:] = [13,2]
189                         else:
190                             pos[n,:] = pos_ini[n,:]
191                             new_maybe_correct_step = new_step(1)
192                             number_of_tries = 0 ## To make it a bit faster we
193                                 only give them 5 tries to make a new move
194                                 otherwise we send them back to their original
195                                 position
196                             while (np.linalg.norm(np.add(pos[n, :],
197                                 new_maybe_correct_step)) >= radius_of_bounds ):
198                                 new_maybe_correct_step = new_step(1)
199                                 number_of_tries = number_of_tries + 1
200                             if number_of_tries == 5:

```

```

183         break
184         if number_of_tries != 5:
185             pos[n,:] = np.add(pos[n,:],
186                               new_maybe_correct_step)
187         elif number_of_tries != 5:
188             pos[n,:] = pos_ini[n,:]
189     else:
190         pos[n,:] = pos_ini[n,:]
191         new_maybe_correct_step = new_step(1)
192         while (np.linalg.norm(np.add(pos[n, :],
193                                       new_maybe_correct_step)) >= 12 ):
194             new_maybe_correct_step = new_step(1)
195             pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
196     line91.set_data(pos[:, 0], pos[:, 1])
197     fig91.canvas.draw()
198     fig91.canvas.flush_events()
199     plt.pause(0.0001)
200
201     ## To make the histogram print faster, comment out the 4 lines above
202
203     fig91, ax91 = plt.subplots(1,1)
204
205     number_of_bins = 10
206
207     logbins = np.logspace(np.log10(np.min(escape_times)),np.log10(np.max(
208         escape_times)),number_of_bins+1)
209     ax91.set_title(f'Figure 9.1: Histogram of mean escape times of {
210         Number_of_Prisoners} prisoners bounded by a fence with a gap from 0
211         to 0.1 radians', size=10, weight='bold')
212     ax91.set_xlabel('Prisoners position in x direction')
213     ax91.set_ylabel('Prisoners position in y direction')
214     ax91.hist(escape_times, bins=logbins)
215     plt.xscale('log')
216
217     plt.show()

```

codes/Task 9.py

```

1 #Libraries
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import time
5 from matplotlib import cm
6 from mpl_toolkits.mplot3d import Axes3D
7 import sympy
8 from statistics import mode
9 # Task 1
10 #You cannot change the range using numpy.random.rand() will always
    give you a number from 0 to 1.
11 #We can just multiply it by 2pi
12
13 Random_N_2pi = 2 * np.pi * np.random.rand()
14
15 # print(Random_N_2pi) # <- For testing
16
17 #Task 2
18 #From numpy.org Parameters:
19
20 #    d0, d1, ..., dnint, optional The dimensions of the returned array
    , must be non-negative. If no argument is given a single Python
    float is returned.
21
22
23 #Add documentation and maybe add case where if not an integer give
    error
24 def get_random_radian(N):
25     radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
26     return radian_array
27
28
29
30 step_size = 0.5
31
32 pos = np.array([[0, 0]])
33
34 def get_xy_velocities(N):
35     random_rad_function = get_random_radian(N)[0]
36     random_x_y = np.array([np.cos(random_rad_function) * step_size, np
        .sin(random_rad_function) * step_size])
37     return random_x_y
38
39
40 # Prereq from previous tasks
41
42 #Task 4
43
44 def new_step(N):
45     pos = np.zeros([N, 2])
46     rand_rad = get_random_radian(N) # To use the same direction for a
        pair of xy coordinates
47     x_values = np.cos(rand_rad) * step_size

```

```

48     y_values = np.sin(rand_rad) * step_size
49     rand_array = np.column_stack([x_values, y_values]) # Used to
        combine the x and y into seperate columns
50     pos = np.add(pos, rand_array)
51     return rand_array
52
53
54 Number_of_Steps = 500
55 Number_of_Prisoners = 1000
56
57 pos=np.zeros([Number_of_Prisoners, 2])
58
59
60 #Task 4 graph 1
61
62 #Uncomment the bottom lines if you want to see the graph of prisoners
63 #fig41, ax41 = plt.subplots()
64 #line41, = ax41.plot([], [], 'o')
65 #plt.show(block=False)
66 #ax41.set_xlim(-50, 50)
67 #ax41.set_ylim(-50,50)
68 #ax41.set_title('Path of 1000 prisoners after 500 steps')
69 #ax41.set_xlabel('x position')
70 #ax41.set_ylabel('y position')
71
72 #For task 5
73
74 Number_Of_Dimensions = 2
75
76
77
78
79 # Prereqs
80
81 #Uncomment to see the movement of the prisoners
82
83 radius_of_bounds = 12
84 boundry_condition = radius_of_bounds*np.cos(0.1*np.pi)
85
86 #Comment out the lines below to print the histogram faster
87 #This makes the simulation much slower
88
89 fig100, ax100 = plt.subplots()
90 line100, = ax100.plot([], [], 'o')
91 ax100.set_xlim(-20, 20)
92 ax100.set_ylim(-20,20)
93 x1_for_10 = np.linspace(-radius_of_bounds,radius_of_bounds,10**4)
94 x2_for_10 = np.linspace(-radius_of_bounds, boundry_condition, 10**4)
95 y1_for_10 = np.sqrt(radius_of_bounds**2 - (x2_for_10**2))
96 y2_for_10 = -1* np.sqrt(radius_of_bounds**2 - (x1_for_10**2))
97 ax100.plot(x2_for_10, y1_for_10, "r-")
98 ax100.plot(x1_for_10, y2_for_10, "r-")
99 ax100.set_ylabel('Position in y')
100 ax100.set_xlabel('Position in x')

```

```

101
102
103
104 escape_times = []
105 to_be_removed = []
106 step_number = 0
107 Number_of_Prisoners = 1 # As the number of prisoners increase the
    time to print inceases a lot. Make it smaller to get a faster print
108 #Number_of_Steps = 1 # Since were running it until they leave we dont
    know how many steps it will take
109 pos=np.zeros([Number_of_Prisoners, 2])
110
111 ax100.set_title(f'Movement of {Number_of_Prisoners} prisoners in
    circular domain of radius {radius_of_bounds} with a small gap')
112
113 # This one is quicker but its vulnerable to the edge case as described
    in the assignment document figure 1
114 #Uncomment it to see it
115 # For task 10, it may be quicker to use this one and sacrifice the
    edge cases
116
117 #while len(pos[:,0]) > 0:
118 #     step_number = step_number + 1
119 #     for p in range(len(pos[:, 0])):
120 #         if pos [p, 0] == 13 and pos[p,1] == 2:
121 #             to_be_removed.append(p)
122 #             escape_times.append(step_number-1)
123 #     pos = np.delete(pos, to_be_removed, axis=0)
124 #     to_be_removed = []
125 #     pos_ini = pos.copy()
126 #     pos = np.add(pos, new_step(len(pos[:,0])))
127 #     for n in range(len(pos[:,0])):
128 #         if np.linalg.norm(pos[n,:]) >= 12:
129 #             if pos[n, 1] >= 0 and boundry_condition <= pos[n, 0]:
130 #                 pos[n,:] = [13,2]
131 #             else:
132 #                 pos[n,:] = pos_ini[n,:]
133 #                 new_maybe_correct_step = new_step(1)
134 #                 while (np.linalg.norm(np.add(pos[n, :],
    new_maybe_correct_step)) >= 12 ):
135 #                     new_maybe_correct_step = new_step(1)
136 #                     pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
137 #     line91.set_data(pos[:, 0], pos[:, 1])
138 #     fig91.canvas.draw()
139 #     fig91.canvas.flush_events()
140 #     plt.pause(0.0001)
141
142
143 # The one below solves the edge case but its a bit slower
144
145 # Prereqs
146
147
148

```

```

149
150
151 #Check if correctly crosses the border
152
153 x_for_checking = sympy.symbols('x', positive=True)
154
155 Mean_escape_time_list = []
156 Median_escape_time_list = []
157 Mode_escape_time_list = []
158
159 gapsizes = []
160 escape_times = []
161 to_be_removed = []
162 step_number = 0
163
164
165 for a in range(1, 6):
166     pos=np.zeros([Number_of_Prisoners, 2])
167     escape_times = []
168     to_be_removed = []
169     step_number = 0
170     boundry_condition_for_loop = 12 * np.cos(0.1 * a * np.pi) - 0.7
171     boundry_condition_for_checking = 12*sympy.cos(0.1*a*sympy.pi) #
        Pretty much same but using sympy
172     f1 = sympy.Piecewise((sympy.sqrt(12**2 - x_for_checking**2),
        x_for_checking >= boundry_condition_for_checking), (0, True))
173     while len(pos[:,0]) > 0: # This makes it so that
174         step_number = step_number + 1
175         for p in range(len(pos[:, 0])): ## Check if any prisoner is
            in 13, 2 and remove it
176             if pos [p, 0] == 13 and pos[p,1] == 2:
177                 to_be_removed.append(p)
178                 escape_times.append(step_number-1) #only the previous
                    step matters
179             pos = np.delete(pos, to_be_removed, axis=0)
180             to_be_removed = []
181             pos_ini = pos.copy()
182             pos = np.add(pos, new_step(len(pos[:,0])))
183             for n in range(len(pos[:,0])):
184                 if np.linalg.norm(pos[n,:]) >= 12: # Check if they hit the
                    boundry
185                     if pos[n,0] > boundry_condition_for_loop - 0.7 and pos
                        [n,1] > -0.5 and pos[n,1] < 12 * np.sin(0.1 * a *
                            np.pi): # Can be removed but its much slower
                                without
186                         slope_for_testing = (pos[n,1]-pos_ini[n,1])/(pos[n
                            ,0]-pos_ini[n,0])
187                         y_intercept_for_testing = pos[n,1] - (
                            slope_for_testing)*pos[n,0]
188                         f2 = slope_for_testing * x_for_checking +
                            y_intercept_for_testing ## Draw a straight
                                line between new point and initial point
189                         solution_to_be_tested = sympy.solve(f1 - f2,
                            x_for_checking)

```

```

190         if len(solution_to_be_tested)==1 and
           solution_to_be_tested[0] >= boundry_condition:
191             pos[n,:] = [13,2]      ## 13, 2 is just an
           arbitrary position outside the domain and
           its a unique position and we know that we
           can remove a prisoner if their position is
           13,2
192         elif len(solution_to_be_tested) == 2 and
           solution_to_be_tested[1] >= boundry_condition:
193             pos[n,:] = [13,2]
194         else:
195             pos[n,:] = pos_ini[n,:]
196             new_maybe_correct_step = new_step(1)
197             number_of_tries = 0  ## To make it a bit faster
           we only give them 5 tries to make a new
           move otherwise we send them back to their
           original position
198             while (np.linalg.norm(np.add(pos[n, :],
           new_maybe_correct_step)) >= 12 ):
199                 new_maybe_correct_step = new_step(1)
200                 number_of_tries = number_of_tries + 1
201                 if number_of_tries == 5:
202                     break
203                 if number_of_tries != 5:
204                     pos[n,:] = np.add(pos[n,:],
           new_maybe_correct_step)
205                 elif number_of_tries != 5:
206                     pos[n,:] = pos_ini[n,:]
207             else:
208                 pos[n,:] = pos_ini[n,:]
209                 new_maybe_correct_step = new_step(1)
210                 while (np.linalg.norm(np.add(pos[n, :],
           new_maybe_correct_step)) >= 12 ):
211                     new_maybe_correct_step = new_step(1)
212                 pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
213             line100.set_data(pos[:, 0], pos[:, 1])
214             fig100.canvas.draw()
215             fig100.canvas.flush_events()  ## IF you want to see it
           uncomment the 4 lines
216             plt.pause(0.0001)
217             Mean_escape_time_list.append(np.mean(escape_times))
218             Median_escape_time_list.append(np.median(escape_times))
219             gapsizes.append(0.1 * a * np.pi)
220             Mode_escape_time_list.append(mode(escape_times))
221             print(f'Cycle {a} complete!')
222
223
224
225
226
227
228
229 Gapsize_mean_escape_time = np.column_stack([gapsizes,
           Mean_escape_time_list])

```

```

230 Gapsize_median_escape_time = np.column_stack([gapsizes,
        Median_escape_time_list])
231 Gapszie_mode_escape_time = np.column_stack([gapsizes,
        Mode_escape_time_list])
232
233
234
235
236 def mean_escape_time(r, s, t , d):
237     '''Takes 4 input parameters, radius, step size (epsilon), delta t,
        and number of dimensions.'''
238     epsilon = s
239     diffusion_coeff_in_function = (epsilon**2)/(2 * d * t)
240     residence_time = (r**2)/(diffusion_coeff_in_function) * (np.log10
        (epsilon**(-1)) + np.log10(2) + 8**(-1))
241     return residence_time
242
243
244 fig101 = plt.figure(figsize=(14,7))
245
246 x_ticks = ['0.1 ', '0.2 ', '0.3 ', '0.4 ', '0.5 ']
247
248 x_values_for_task_10 = np.linspace(0, 0.5*np.pi, 1000)
249 y_values_for_task_10 = mean_escape_time(12, 0.5, 1, 2) + 0*
        x_values_for_task_10
250
251 #Mean
252 ax101 = plt.subplot(1, 3, 1)
253 ax101.plot(Gapsize_mean_escape_time[:,0], Gapsize_mean_escape_time
       [:,1], label='Mean escape time')
254 ax101.set_xticks(Gapsize_mean_escape_time[:,0])
255 ax101.set_xticklabels(x_ticks)
256 ax101.set_title('Figure 10.1: Mean prisoner escape time as a function
        of fence gap size', size=5.5, weight='bold')
257 ax101.set_xlabel('Gap Size (radians)')
258 ax101.set_ylabel('Mean escape time t (s)')
259 ax101.plot(x_values_for_task_10, y_values_for_task_10, label='Mean
        expected escape time')
260 plt.legend()
261
262
263 #Median
264 ax102 = plt.subplot(1,3,2)
265 ax102.plot(Gapsize_median_escape_time[:,0], Gapsize_median_escape_time
       [:,1], label='Median escape time')
266 ax102.set_xticks(Gapsize_mean_escape_time[:,0])
267 ax102.set_xticklabels(x_ticks)
268 ax102.set_title('Figure 10.2: Median prisoner escape time as a
        function of fence gap size', size=5.5, weight='bold')
269 ax102.set_xlabel('Gap Size (radians)')
270 ax102.set_ylabel('Mean escape time t (s)')
271 ax102.plot(x_values_for_task_10, y_values_for_task_10, label='Median
        expected escape time')
272 plt.legend()

```



```

273
274 #Mode
275 ax103 = plt.subplot(1,3,3)
276 ax103.plot(Gapszie_mode_escape_time[:,0], Gapszie_mode_escape_time
277          [:,1], label='Modal escape time')
278 ax103.set_xticks(Gapszie_mean_escape_time[:,0])
279 ax103.set_xticklabels(x_ticks)
280 ax103.set_title('Figure 10.3: Mode prisoner escape time as a function
281               of fence gap size', size=5.5, weight='bold')
282 ax103.set_xlabel('Gap Size (radians)')
283 ax103.set_ylabel('Mean escape time t (s)')
284 ax103.plot(x_values_for_task_10, y_values_for_task_10, label='Modal
285           expected escape time')
286 plt.legend()
287
288
289 plt.show()

```

codes/Task 10.py