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

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

Listing 1: Definition of get_random_radian(N)

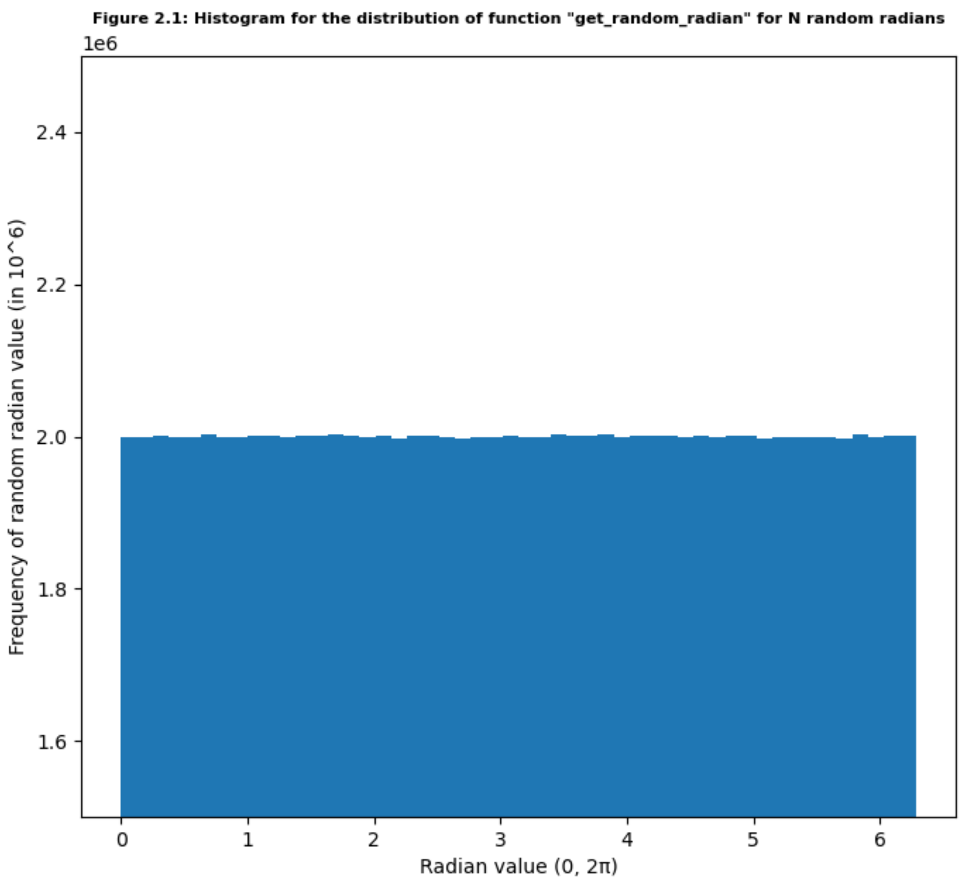

Figure 1: Distribution of the get_random_radian function

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

Listing 2: Definition of new_step(N)

```python
def mean_square_displacement_f(j):
    displacement = np.array([])
    for i in range(len(j[:, 0])):
        displacement = np.append(displacement, np.linalg.norm(pos[i,
            :])**2)
    mean_square_displacement = np.mean(displacement)
    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

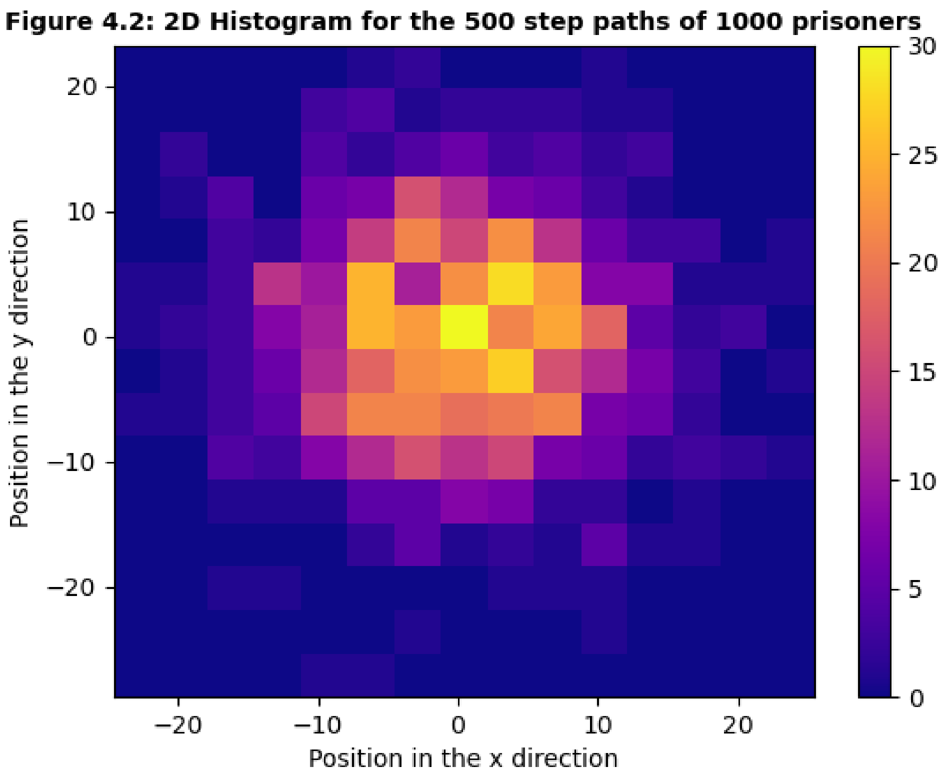

Figure 2: 2D Histogram of the positions of the prisoners after 500 steps

```python
for i in range(Number_of_Steps):
    pos = np.add(pos, new_step(Number_of_Prisoners))
    if i == 99:
        pos_after_100_steps = pos
    if i == 199:
        pos_after_200_steps = pos
    if i == 299:
        pos_after_300_steps = pos
    if i == 399:
        pos_after_400_steps = pos
    current_d = mean_square_displacement_f(pos) / (2*
       Number_Of_Dimensions * (i+1))
    dcoeff_vs_time[i, :] = [i + 1, current_d]
    line41.set_data(pos[:, 0], pos[:, 1])
    fig41.canvas.draw()
    fig41.canvas.flush_events()
    plt.pause(0.001)
```

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

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

```python
for n in range(len(pos[:,0])):
    if np.linalg.norm(pos[n,:]) >= radius_of_bounds: # Check if they hit the boundry
        if pos[n,0] > boundry_condition - 0.7:  # Can be removed but its much slower without
            slope_for_testing = (pos[n,1]-pos_ini[n,1])/(pos[n,0]-pos_ini[n,0])
            y_intercept_for_testing = pos[n,1] - (slope_for_testing)*pos[n,0]
            f2 = slope_for_testing * x_for_checking + y_intercept_for_testing  ## Draw a straight line between new point and initial point
            solution_to_be_tested = sympy.solve(f1 - f2, x_for_checking)
            if len(solution_to_be_tested)==1 and solution_to_be_tested[0] >= boundry_condition:
                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
            elif len(solution_to_be_tested) == 2 and solution_to_be_tested[1] >= boundry_condition:
                pos[n,:] = [13,2]
            else:
                pos[n,:] = pos_ini[n,:]
                new_maybe_correct_step = new_step(1)
                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
                while (np.linalg.norm(np.add(pos[n, :], new_maybe_correct_step)) >= radius_of_bounds ):
                    new_maybe_correct_step = new_step(1)
                    number_of_tries = number_of_tries + 1
                    if number_of_tries == 5:
                        break
                if number_of_tries != 5:
                    pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
                elif number_of_tries != 5:
                    pos[n,:] = pos_ini[n,:]
        else:
            pos[n,:] = pos_ini[n,:]
            new_maybe_correct_step = new_step(1)
            while (np.linalg.norm(np.add(pos[n, :], new_maybe_correct_step)) >= 12 ):
                new_maybe_correct_step = new_step(1)
            pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
```
Listing 6: Loop to check if the current move was a success

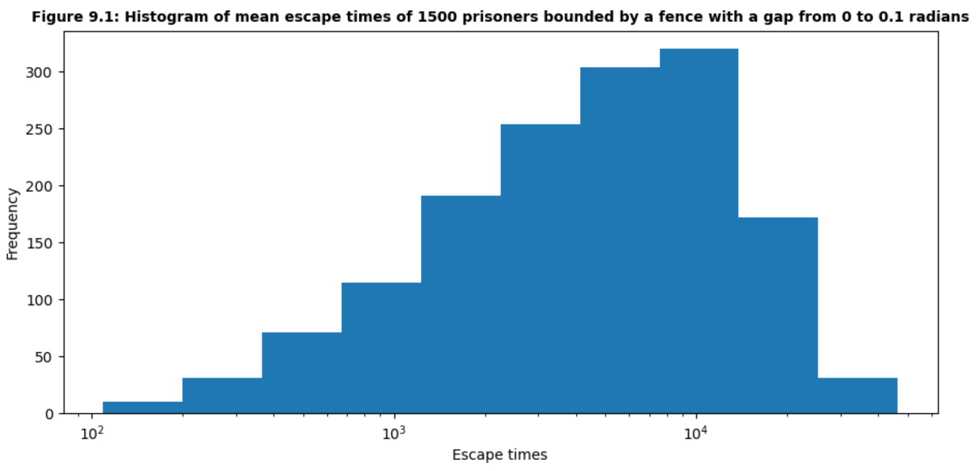

Figure 3: Histogram of 1500 prisoner escape times

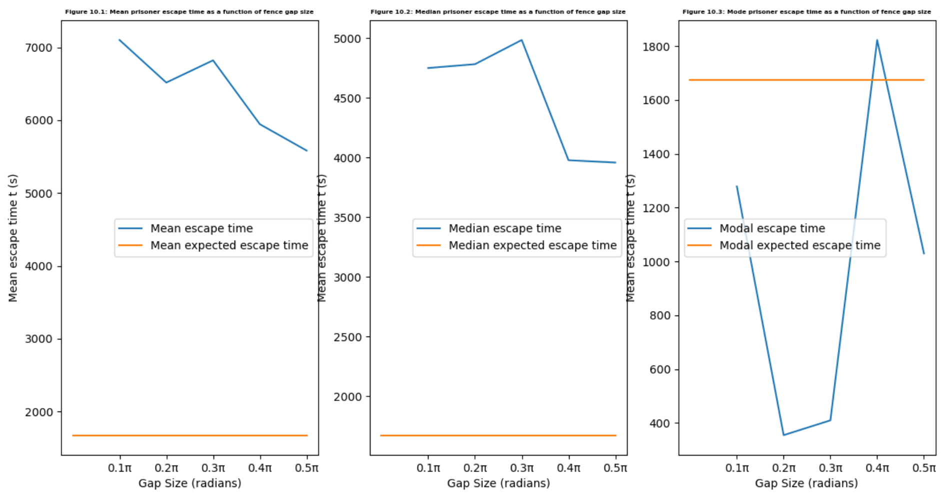

Figure 4: Mean, median, and mode, escape times vs expected escape time.

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

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always
    give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing
```

codes/Task 1.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give error
def get_random_radian(N):
    '''Input must be an integer and output is an array of N columns with random radians'''
    radian_array =  2 * np.pi * np.random.rand(N)
    return radian_array


Random_radian_big_number = 10**8 #Bigger exponent = more time

fig21, ax21 = plt.subplots(1, 1, figsize=(8,7))
ax21.hist(get_random_radian(Random_radian_big_number), bins=50)

#To make graph nicer
ax21.set_title('Figure 2.1: Histogram for the distribution of function "get_random_radian" for N random radians', size=8, weight='bold')
ax21.set_xlabel('Radian value (0, 2  )')
ax21.set_ylabel('Frequency of random radian value (in 10^6)')
ax21.set_ylim([1.5*10**6,2.5*10**6])

plt.show()
plt.savefig("plot2.png")
#Mean
Random_Radian_f_mean = np.mean(get_random_radian(Random_radian_big_number))

#Standard Deviation
Random_Radian_f_stdev = np.std(get_random_radian(Random_radian_big_number))
```

```python

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

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always
   give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.

#Add documentation and maybe add case where if not an integer give
   error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array

# Prereq from previous tasks

step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np
       .sin(random_rad_function) * step_size])
    return random_x_y

#Check if magnitude = 0.5
#print(np.linalg.norm(get_xy_velocities(0))) #Norm does pythagoras

#Here range is the number of steps
#I found vstack which adds the new step to the position array as a new
   row
for N in range(1, 1001):
    Current_Step = get_xy_velocities(N)
    pos = np.vstack((pos, pos[-1, :] + Current_Step)) # New row = old
       row plus new step

#Extracting x and y coordinates

```

```python
Task_3_x = pos[:,0]
Task_3_y = pos[:,1]

#Making the figure itself
fig31 = plt.figure(figsize=(8,7))
ax31 = plt.subplot(1, 1, 1)
line, = ax31.plot(Task_3_x, Task_3_y)

#Making the figure pretty
ax31.set_title('Figure 3.1: Animation for a random 1000 step path of a
    single prisoner', size=10, weight='bold')
ax31.set_xlabel('Prisoner position in x direction')
ax31.set_ylabel('Prisoner position in y direction')

plt.show(block=False)
for i in range(1000):
    line.set_data(Task_3_x[i-20:i+1], Task_3_y[i-20:i+1])
    fig31.canvas.draw()
    fig31.canvas.flush_events()
    plt.pause(0.0001)
```

codes/Task 3.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always
    give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give
    error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array



step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np
        .sin(random_rad_function) * step_size])
    return random_x_y


# Prereq from previous tasks

#Task 4

def new_step(N):
    pos = np.zeros([N, 2])
    rand_rad = get_random_radian(N) # To use the same direction for a
        pair of xy coordinates
    x_values = np.cos(rand_rad) * step_size
    y_values = np.sin(rand_rad) * step_size
    rand_array = np.column_stack([x_values, y_values]) # Used to
```

```
        combine the x and y into seperate columns
    pos = np.add(pos, rand_array)
    return rand_array


Number_of_Steps = 500
Number_of_Prisoners = 1000

pos=np.zeros([Number_of_Prisoners, 2])


#Task 4 graph 1
fig41, ax41 = plt.subplots(figsize=(8,7))
line41, = ax41.plot([], [], 'o')

#Making it pretty

plt.show(block=False)
ax41.set_xlim(-50, 50)
ax41.set_ylim(-50,50)
ax41.set_title('Figure 4.1: Animation for the random 500 step paths of
     1000 prisoners', size=10, weight='bold')
ax41.set_xlabel('Prisoners position in x direction')
ax41.set_ylabel('Prisoners position in y direction')


for i in range(Number_of_Steps):
    pos = np.add(pos, new_step(Number_of_Prisoners))
    line41.set_data(pos[:, 0], pos[:, 1])
    fig41.canvas.draw()
    fig41.canvas.flush_events()
    plt.pause(0.001)



#I assume that a 2d histogram is just a heatmeap

#For plotting

fig42, ax42 = plt.subplots()


hist2d42 = ax42.hist2d(pos[:, 0], pos[:, 1], bins=15, cmap=cm.plasma)
plt.colorbar(hist2d42[3], ax=ax42)

ax42.set_xlabel('Position in the x direction')
ax42.set_ylabel('Position in the y direction')
ax42.set_title('Figure 4.2: 2D Histogram for the 500 step paths of
    1000 prisoners', size=10, weight='bold')

plt.show()
```

codes/Task 4.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array



step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np.sin(random_rad_function) * step_size])
    return random_x_y




#Task 4

def new_step(N):
    pos = np.zeros([N, 2])
    rand_rad = get_random_radian(N) # To use the same direction for a pair of xy coordinates
    x_values = np.cos(rand_rad) * step_size
    y_values = np.sin(rand_rad) * step_size
    rand_array = np.column_stack([x_values, y_values]) # Used to
```

```
       combine the x and y into seperate columns
    pos = np.add(pos, rand_array)
    return rand_array


Number_of_Steps = 500
Number_of_Prisoners = 1000

pos=np.zeros([Number_of_Prisoners, 2])


#Task 4 graph 1

#Comment out the lines below to hide the graph
fig41, ax41 = plt.subplots()
line41, = ax41.plot([], [], 'o')
plt.show(block=False)
ax41.set_xlim(-50, 50)
ax41.set_ylim(-50,50)
ax41.set_title('Path of 1000 prisoners after 500 steps')
ax41.set_xlabel('x position')
ax41.set_ylabel('y position')








#Prereq from previous tasks

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

#For task 5
dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
Number_Of_Dimensions = 2


for i in range(Number_of_Steps):
    pos = np.add(pos, new_step(Number_of_Prisoners))
    current_d = mean_square_displacement_f(pos) / (2*
       Number_Of_Dimensions * (i+1))
    dcoeff_vs_time[i, :] = [i + 1, current_d]
    line41.set_data(pos[:, 0], pos[:, 1])
    fig41.canvas.draw()
    fig41.canvas.flush_events()
    plt.pause(0.001)
```

```python
#Also comment the lines above to hide the graph


x_values_for_task_5 = np.linspace(0, 500, 100)
expected_diffusion_coeff = (step_size**2)/(2*Number_Of_Dimensions* 1)
    + 0*x_values_for_task_5

fig51, ax51 = plt.subplots()
ax51.scatter(dcoeff_vs_time[:,0], dcoeff_vs_time[:,1])
ax51.plot(x_values_for_task_5, expected_diffusion_coeff, label='
    Expected Diffusion Coefficient')
#Make the label for the straight line show


#Making ax51 pretty
ax51.set_xlabel('Number of Steps')
ax51.set_ylabel('Diffusion Coefficient')
ax51.set_title('Expected Diffusion Coefficient vs number of steps')



plt.show()
```

codes/Task 5.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array

step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np.sin(random_rad_function) * step_size])
    return random_x_y




#Task 4

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


Number_of_Steps = 500
Number_of_Prisoners = 1000

pos=np.zeros([Number_of_Prisoners, 2])




#Comment out the lines below to hide the graph
fig41, ax41 = plt.subplots()
line41, = ax41.plot([], [], 'o')
plt.show(block=False)
ax41.set_xlim(-50, 50)
ax41.set_ylim(-50,50)
ax41.set_title('Path of 1000 prisoners after 500 steps')
ax41.set_xlabel('x position')
ax41.set_ylabel('y position')






#Prereq from previous tasks

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

#For task 5
dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
Number_Of_Dimensions = 2


for i in range(Number_of_Steps):
    pos = np.add(pos, new_step(Number_of_Prisoners))
    if i == 99:
        pos_after_100_steps = pos
    if i == 199:
        pos_after_200_steps = pos
    if i == 299:
        pos_after_300_steps = pos
    if i == 399:
        pos_after_400_steps = pos
    current_d = mean_square_displacement_f(pos) / (2*
        Number_Of_Dimensions * (i+1))
    dcoeff_vs_time[i, :] = [i + 1, current_d]
    line41.set_data(pos[:, 0], pos[:, 1])
    fig41.canvas.draw()
    fig41.canvas.flush_events()
    plt.pause(0.001)
#Also comment the lines above to hide the graph




#Prereq

#Task 6
#Again the naming, first number is the task number and the second is
    the number  of the graph

fig61 = plt.figure(figsize=(18,8))

ax61 = plt.subplot(2, 4, 1)
hist2d61 = ax61.hist2d(pos_after_100_steps[:, 0], pos_after_100_steps
    [:, 1], bins=15, cmap=cm.plasma)
ax61.set_title('Figure 6.1: Histogram for prisoner positions after 100
     steps', size=7, weight='bold')
plt.colorbar(hist2d61[3], ax=ax61)


ax62 = plt.subplot(2, 4, 2)
hist2d62 = ax62.hist2d(pos_after_200_steps[:, 0], pos_after_200_steps
    [:, 1], bins=15, cmap=cm.plasma)
ax62.set_title('Figure 6.2: Histogram for prisoner positions after 200
     steps', size=7, weight='bold')
plt.colorbar(hist2d62[3], ax=ax62)


ax63 = plt.subplot(2, 4, 5)
hist2d63 = ax63.hist2d(pos_after_300_steps[:, 0], pos_after_300_steps
    [:, 1], bins=15, cmap=cm.plasma)
ax63.set_title('Figure 6.3: Histogram for prisoner positions after 300
     steps', size=7, weight='bold')
plt.colorbar(hist2d63[3], ax=ax63)


ax64 = plt.subplot(2, 4, 6)
hist2d64 = ax64.hist2d(pos_after_400_steps[:, 0], pos_after_400_steps
    [:, 1], bins=15, cmap=cm.plasma)
ax64.set_title('Figure 6.4: Histogram for prisoner positions after 400
     steps', size=7, weight='bold')
plt.colorbar(hist2d64[3], ax=ax64)


ax65 = plt.subplot(1, 2, 2)
hist2d65 = ax65.hist2d(pos[:, 0], pos[:, 1], bins=15, cmap=cm.plasma)
ax65.set_title('Figure 6.5: Histogram for prisoner positions after the
     last step', size=12, weight='bold')
plt.colorbar(hist2d65[3], ax=ax65)



plt.show()
```

codes/Task 6.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array



step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np.sin(random_rad_function) * step_size])
    return random_x_y


# Prereq from previous tasks

#Task 4

def new_step(N):
    pos = np.zeros([N, 2])
    rand_rad = get_random_radian(N) # To use the same direction for a pair of xy coordinates
    x_values = np.cos(rand_rad) * step_size
    y_values = np.sin(rand_rad) * step_size
    rand_array = np.column_stack([x_values, y_values]) # Used to
```

```
        combine the x and y into seperate columns
    pos = np.add(pos, rand_array)
    return rand_array


Number_of_Steps = 500
Number_of_Prisoners = 1000

pos=np.zeros([Number_of_Prisoners, 2])


#Task 4 graph 1

#Uncomment the bottom lines if you want to see the graph of prisoners
#fig41, ax41 = plt.subplots()
#line41, = ax41.plot([], [], 'o')
#plt.show(block=False)
#ax41.set_xlim(-50, 50)
#ax41.set_ylim(-50,50)
#ax41.set_title('Path of 1000 prisoners after 500 steps')
#ax41.set_xlabel('x position')
#ax41.set_ylabel('y position')








#Prereq from previous tasks

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

#For task 5
dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
Number_Of_Dimensions = 2


for i in range(Number_of_Steps):
    pos = np.add(pos, new_step(Number_of_Prisoners))
    if i == 99:
        pos_after_100_steps = pos       #Kinda ugly but it works
    if i == 199:
        pos_after_200_steps = pos
    if i == 299:
        pos_after_300_steps = pos
    if i == 399:
        pos_after_400_steps = pos
    current_d = mean_square_displacement_f(pos) / (2*
       Number_Of_Dimensions * (i+1))
    dcoeff_vs_time[i, :] = [i + 1, current_d]
#    line41.set_data(pos[:, 0], pos[:, 1])
#    fig41.canvas.draw()
#    fig41.canvas.flush_events()
#    plt.pause(0.001)
#Also uncomment the lines above to see



#Prereqs

t=100

x = np.arange(-15, 15, 0.005)
y = np.arange(-15, 15, 0.005)

x,y = np.meshgrid(x, y)



expected_diffusion_coeff_numerically = 0.0625

z = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) *  np.exp
   (-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t))
z1 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) *  np.
   exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*2)
   )
z2 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) *  np.
   exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*3)
   )
z3 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) *  np.
   exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*4)
   )
z4 = 1/(4 * np.pi * expected_diffusion_coeff_numerically * t) *  np.
   exp(-(x**2 + y**2)/(4 * expected_diffusion_coeff_numerically * t*5)
   )


fig71 = plt.figure(figsize=(18,8))
ax71 = fig71.add_subplot(2, 3, 1, projection='3d')
ax71.set_title('Figure 7.1: Probability density function 3D projection
    at 100 steps', size=6, weight='bold')
ax71.set_xlabel('x')
ax71.set_ylabel('y')
ax71.set_zlabel('pdf')
ax72 = fig71.add_subplot(2, 3, 2, projection='3d')
ax72.set_title('Figure 7.2: Probability density function 3D projection
    at 200 steps', size=6, weight='bold')
ax72.set_xlabel('x')
ax72.set_ylabel('y')
ax72.set_zlabel('pdf')
```

```python
ax73 = fig71.add_subplot(2, 3, 3, projection='3d')
ax73.set_title('Figure 7.3: Probability density function 3D projection
    at 300 steps', size=6, weight='bold')
ax73.set_xlabel('x')
ax73.set_ylabel('y')
ax73.set_zlabel('pdf')
ax74 = fig71.add_subplot(2, 3, 4, projection='3d')
ax74.set_title('Figure 7.4: Probability density function 3D projection
    at 400 steps', size=6, weight='bold')
ax74.set_xlabel('x')
ax74.set_ylabel('y')
ax74.set_zlabel('pdf')
ax75 = fig71.add_subplot(2, 3, 5, projection='3d')
ax75.set_title('Figure 7.5: Probability density function 3D projection
    at 500 steps', size=6, weight='bold')
ax75.set_xlabel('x')
ax75.set_ylabel('y')
ax75.set_zlabel('pdf')


ax76 = plt.subplot(2, 3, 6)
hist2d65 = ax76.hist2d(pos[:, 0], pos[:, 1], bins=15, cmap=cm.plasma)
ax76.set_title('Figure 7.6: Histogram for prisoner positions after the
    last step', size=6, weight='bold')
ax76.set_xlabel('x')
ax76.set_ylabel('y')
ax76.set_xlim(-15, 15)
ax76.set_ylim(-15,15)

plt.colorbar(hist2d65[3], ax=ax76)





surf1 = ax71.plot_surface(x,y,z, cmap=cm.magma, linewidth=0,
   antialiased=0)
surf2 = ax72.plot_surface(x,y,z1, cmap=cm.magma, linewidth=0,
   antialiased=0)
surf3 = ax73.plot_surface(x,y,z2, cmap=cm.magma, linewidth=0,
   antialiased=0)
surf4 = ax74.plot_surface(x,y,z3, cmap=cm.magma, linewidth=0,
   antialiased=0)
surf5 = ax75.plot_surface(x,y,z4, cmap=cm.magma, linewidth=0,
   antialiased=0)


plt.show()
```

codes/Task 7.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
# Task 1
#You cannot change the range using numpy.random.rand() will always give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array



step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np.sin(random_rad_function) * step_size])
    return random_x_y



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

```python


Number_of_Steps = 500
Number_of_Prisoners = 1000

pos=np.zeros([Number_of_Prisoners, 2])


#Task 4 graph 1

#Uncomment the bottom lines if you want to see the graph of prisoners
#fig41, ax41 = plt.subplots()
#line41, = ax41.plot([], [], 'o')
#plt.show(block=False)
#ax41.set_xlim(-50, 50)
#ax41.set_ylim(-50,50)
#ax41.set_title('Path of 1000 prisoners after 500 steps')
#ax41.set_xlabel('x position')
#ax41.set_ylabel('y position')









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

#For task 5
dcoeff_vs_time = np.zeros((Number_of_Steps, 2))
Number_Of_Dimensions = 2

#Prereqs

#Copied but modified from task 4
fig81, ax81 = plt.subplots()
line81, = ax81.plot([], [], 'o')

#Making it pretty
plt.show(block=False)
ax81.set_xlim(-20, 20)
ax81.set_ylim(-20,20)
ax81.set_title('Figure 8.1: Animation for the random 500 step paths of
    1000 prisoners with bounds')
ax81.set_xlabel('Prisoners position in x direction')
ax81.set_ylabel('Prisoners position in y direction')
```

```python


radius_of_bounds = 12


x1_for_8 = np.linspace(-radius_of_bounds,radius_of_bounds,10**4)
y1_for_8 = np.sqrt(radius_of_bounds**2 - (x1_for_8**2))
y2_for_8 = -1* np.sqrt(radius_of_bounds**2 - (x1_for_8**2))
ax81.plot(x1_for_8, y1_for_8, "r-")
ax81.plot(x1_for_8, y2_for_8, "r-")


plt.show(block=False)
pos=np.zeros([Number_of_Prisoners, 2])
pos = pos + 0.0

for i in range(Number_of_Steps):
    pos_ini = pos.copy()
    pos = np.add(pos, new_step(Number_of_Prisoners))
    for n in range(len(pos[:,0])):
        if np.linalg.norm(pos[n,:]) >= radius_of_bounds: # First check to see if the new position is outside the bounds
            pos[n,:] = pos_ini[n,:]     # Go back to initial position
            new_maybe_correct_step = new_step(1)
            while np.linalg.norm(np.add(pos[n,:], 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)
    line81.set_data(pos[:, 0], pos[:, 1])
    fig81.canvas.draw()
    fig81.canvas.flush_events()
    plt.pause(0.001)

# Make it so that the graph prints as a square so that its clear that the boundry is a circle and not an oval
```

codes/Task 8.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import sympy
# Task 1
#You cannot change the range using numpy.random.rand() will always give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array



step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np.sin(random_rad_function) * step_size])
    return random_x_y


# Prereq from previous tasks

#Task 4

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


#Number_of_Steps = 500
#Number_of_Prisoners = 1000
#pos=np.zeros([Number_of_Prisoners, 2])
#Relics from previous tasks

#Task 4 graph 1

#Uncomment the bottom lines if you want to see the graph of prisoners
#fig41, ax41 = plt.subplots()
#line41, = ax41.plot([], [], 'o')
#plt.show(block=False)
#ax41.set_xlim(-50, 50)
#ax41.set_ylim(-50,50)
#ax41.set_title('Path of 1000 prisoners after 500 steps')
#ax41.set_xlabel('x position')
#ax41.set_ylabel('y position')

#For task 5

Number_Of_Dimensions = 2




# Prereqs



radius_of_bounds = 12
boundry_condition = radius_of_bounds*np.cos(0.1*np.pi)

#Comment out the lines below to print the historgram faster
fig91, ax91 = plt.subplots()
line91, = ax91.plot([], [], 'o')
ax91.set_xlim(-20, 20)
ax91.set_ylim(-20,20)
x1_for_9 = np.linspace(-radius_of_bounds,radius_of_bounds,10**4)
x2_for_9 = np.linspace(-radius_of_bounds, boundry_condition, 10**4)
y1_for_9 = np.sqrt(radius_of_bounds**2 - (x2_for_9**2))
y2_for_9 = -1* np.sqrt(radius_of_bounds**2 - (x1_for_9**2))
ax91.plot(x2_for_9, y1_for_9, "r-")
ax91.plot(x1_for_9, y2_for_9, "r-")
ax91.set_ylabel('Position in y')
ax91.set_xlabel('Position in x')



escape_times = []
```

```python
to_be_removed = []
Number_of_Prisoners = 10 ## Set to ten now but you can change it to a
    different number, bigger number = much slower
#Number_of_Steps = 1 # Since were running it until they leave we dont
    know how many steps it will take

## Also comment out this line
ax91.set_title(f'Movement of {Number_of_Prisoners} prisoners in
    circular domain of radius {radius_of_bounds} with a small gap')


pos=np.zeros([Number_of_Prisoners, 2])
step_number = 0

# This one is quicker but its vulnerable to the edge case as described
     in the assignment document figure 1

#Uncomment it to see it
#while len(pos[:,0]) > 0:
#    step_number = step_number + 1
#    for p in range(len(pos[:, 0])):
#        if pos [p, 0] == 13 and pos[p,1] == 2:
#            to_be_removed.append(p)
#            escape_times.append(step_number-1)
#    pos = np.delete(pos, to_be_removed, axis=0)
#    to_be_removed = []
#    pos_ini = pos.copy()
#    pos = np.add(pos, new_step(len(pos[:,0])))
#    for n in range(len(pos[:,0])):
#        if np.linalg.norm(pos[n,:]) >= radius_of_bounds:
#            if pos[n, 1] >= 0 and boundry_condition <= pos[n, 0]:
#                pos[n,:] = [13,2]
#            else:
#               pos[n,:] = pos_ini[n,:]
#               new_maybe_correct_step = new_step(1)
#               while (np.linalg.norm(np.add(pos[n, :],
    new_maybe_correct_step)) >= radius_of_bounds ):
#                   new_maybe_correct_step = new_step(1)
#               pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
#    line91.set_data(pos[:, 0], pos[:, 1])
#    fig91.canvas.draw()
#    fig91.canvas.flush_events()
#    plt.pause(0.0001)


# The one below solves the edge case but its a bit slower


#Check if correctly crosses the border
x_for_checking = sympy.symbols('x', positive=True)

boundry_condition_for_checking = radius_of_bounds*sympy.cos(0.1*sympy.
    pi)

```

```
f1 = sympy.Piecewise((sympy.sqrt(radius_of_bounds**2 - x_for_checking**2), x_for_checking >= boundry_condition_for_checking), (0, True))

# To change number of prisoners use the code that a couple of lines above
# Its quite slow with # of prisoners > 10000

while len(pos[:,0]) > 0: # This makes it so that the code keeps running as long as there is a prisoner insisde the bounds
    step_number = step_number + 1
    for p in range(len(pos[:, 0])):  ## Check if any prisoner is in 13, 2 and remove it
        if pos [p, 0] == 13 and pos[p,1] == 2:
            to_be_removed.append(p)
            escape_times.append(step_number-1)
    pos = np.delete(pos, to_be_removed, axis=0)
    to_be_removed = []
    pos_ini = pos.copy()
    pos = np.add(pos, new_step(len(pos[:,0])))
    for n in range(len(pos[:,0])):
        if np.linalg.norm(pos[n,:]) >= radius_of_bounds: # Check if they hit the boundry
            if pos[n,0] > boundry_condition - 0.7:  # Can be removed but its much slower without
                slope_for_testing = (pos[n,1]-pos_ini[n,1])/(pos[n,0]-pos_ini[n,0])
                y_intercept_for_testing = pos[n,1] - (slope_for_testing)*pos[n,0]
                f2 = slope_for_testing * x_for_checking + y_intercept_for_testing  ## Draw a straight line between new point and initial point
                solution_to_be_tested = sympy.solve(f1 - f2, x_for_checking)
                if len(solution_to_be_tested)==1 and solution_to_be_tested[0] >= boundry_condition:
                   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
                elif len(solution_to_be_tested) == 2 and solution_to_be_tested[1] >= boundry_condition:
                   pos[n,:] = [13,2]
                else:
                   pos[n,:] = pos_ini[n,:]
                   new_maybe_correct_step = new_step(1)
                   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
                   while (np.linalg.norm(np.add(pos[n, :], new_maybe_correct_step)) >= radius_of_bounds ):
                       new_maybe_correct_step = new_step(1)
                       number_of_tries = number_of_tries + 1
                       if number_of_tries == 5:
```

```python
                           break
                   if number_of_tries != 5:
                       pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
                   elif number_of_tries != 5:
                       pos[n,:] = pos_ini[n,:]
            else:
               pos[n,:] = pos_ini[n,:]
               new_maybe_correct_step = new_step(1)
               while (np.linalg.norm(np.add(pos[n, :], new_maybe_correct_step)) >= 12 ):
                   new_maybe_correct_step = new_step(1)
               pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
    line91.set_data(pos[:, 0], pos[:, 1])
    fig91.canvas.draw()
    fig91.canvas.flush_events()
    plt.pause(0.0001)

## To make the histogram print faster, comment out the 4 lines above

fig91, ax91 = plt.subplots(1,1)

number_of_bins = 10

logbins = np.logspace(np.log10(np.min(escape_times)),np.log10(np.max(escape_times)),number_of_bins+1)
ax91.set_title(f'Figure 9.1: Histogram of mean escape times of {Number_of_Prisoners} prisoners bounded by a fence with a gap from 0 to 0.1 radians', size=10, weight='bold')
ax91.set_xlabel('Prisoners position in x direction')
ax91.set_ylabel('Prisoners position in y direction')
ax91.hist(escape_times, bins=logbins)
plt.xscale('log')

plt.show()
```

codes/Task 9.py

```python
#Libraries
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import sympy
from statistics import mode
# Task 1
#You cannot change the range using numpy.random.rand() will always give you a number from 0 to 1.
#We can just multiply it by 2pi

Random_N_2pi = 2 * np.pi * np.random.rand()

# print(Random_N_2pi) # <- For testing

#Task 2
#From numpy.org Parameters:

#    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.


#Add documentation and maybe add case where if not an integer give error
def get_random_radian(N):
    radian_array = Random_N_2pi = 2 * np.pi * np.random.rand(N)
    return radian_array



step_size = 0.5

pos = np.array([[0, 0]])

def get_xy_velocities(N):
    random_rad_function = get_random_radian(N)[0]
    random_x_y = np.array([np.cos(random_rad_function) * step_size, np.sin(random_rad_function) * step_size])
    return random_x_y


# Prereq from previous tasks

#Task 4

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


Number_of_Steps = 500
Number_of_Prisoners = 1000

pos=np.zeros([Number_of_Prisoners, 2])


#Task 4 graph 1

#Uncomment the bottom lines if you want to see the graph of prisoners
#fig41, ax41 = plt.subplots()
#line41, = ax41.plot([], [], 'o')
#plt.show(block=False)
#ax41.set_xlim(-50, 50)
#ax41.set_ylim(-50,50)
#ax41.set_title('Path of 1000 prisoners after 500 steps')
#ax41.set_xlabel('x position')
#ax41.set_ylabel('y position')

#For task 5

Number_Of_Dimensions = 2




# Prereqs

#Uncomment to see the movement of the prisoners

radius_of_bounds = 12
boundry_condition = radius_of_bounds*np.cos(0.1*np.pi)

#Comment out the lines below to print the historgram faster
#This makes the simulation much slower

fig100, ax100 = plt.subplots()
line100, = ax100.plot([], [], 'o')
ax100.set_xlim(-20, 20)
ax100.set_ylim(-20,20)
x1_for_10 = np.linspace(-radius_of_bounds,radius_of_bounds,10**4)
x2_for_10 = np.linspace(-radius_of_bounds, boundry_condition, 10**4)
y1_for_10 = np.sqrt(radius_of_bounds**2 - (x2_for_10**2))
y2_for_10 = -1* np.sqrt(radius_of_bounds**2 - (x1_for_10**2))
ax100.plot(x2_for_10, y1_for_10, "r-")
ax100.plot(x1_for_10, y2_for_10, "r-")
ax100.set_ylabel('Position in y')
ax100.set_xlabel('Position in x')
```

```python



escape_times = []
to_be_removed = []
step_number = 0
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
#Number_of_Steps = 1 # Since were running it until they leave we dont know how many steps it will take
pos=np.zeros([Number_of_Prisoners, 2])

ax100.set_title(f'Movement of {Number_of_Prisoners} prisoners in circular domain of radius {radius_of_bounds} with a small gap')

# This one is quicker but its vulnerable to the edge case as described in the assignment document figure 1
#Uncomment it to see it
# For task 10, it may be quicker to use this one and sacrifice the edge cases

#while len(pos[:,0]) > 0:
#    step_number = step_number + 1
#    for p in range(len(pos[:, 0])):
#        if pos [p, 0] == 13 and pos[p,1] == 2:
#            to_be_removed.append(p)
#            escape_times.append(step_number-1)
#    pos = np.delete(pos, to_be_removed, axis=0)
#    to_be_removed = []
#    pos_ini = pos.copy()
#    pos = np.add(pos, new_step(len(pos[:,0])))
#    for n in range(len(pos[:,0])):
#        if np.linalg.norm(pos[n,:]) >= 12:
#            if pos[n, 1] >= 0 and boundry_condition <= pos[n, 0]:
#                pos[n,:] = [13,2]
#            else:
#               pos[n,:] = pos_ini[n,:]
#               new_maybe_correct_step = new_step(1)
#               while (np.linalg.norm(np.add(pos[n, :], new_maybe_correct_step)) >= 12 ):
#                   new_maybe_correct_step = new_step(1)
#               pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
#    line91.set_data(pos[:, 0], pos[:, 1])
#    fig91.canvas.draw()
#    fig91.canvas.flush_events()
#    plt.pause(0.0001)


# The one below solves the edge case but its a bit slower

# Prereqs



```

```python


#Check if correctly crosses the border

x_for_checking = sympy.symbols('x', positive=True)

Mean_escape_time_list = []
Median_escape_time_list = []
Mode_escape_time_list = []

gapsizes = []
escape_times = []
to_be_removed = []
step_number = 0


for a in range(1, 6):
    pos=np.zeros([Number_of_Prisoners, 2])
    escape_times = []
    to_be_removed = []
    step_number = 0
    boundry_condition_for_loop = 12 * np.cos(0.1 * a * np.pi) - 0.7
    boundry_condition_for_checking = 12*sympy.cos(0.1*a*sympy.pi) # Pretty much same but using sympy
    f1 = sympy.Piecewise((sympy.sqrt(12**2 - x_for_checking**2), x_for_checking >= boundry_condition_for_checking), (0, True))
    while len(pos[:,0]) > 0: # This makes it so that
        step_number = step_number + 1
        for p in range(len(pos[:, 0])):  ## Check if any prisoner is in 13, 2 and remove it
            if pos [p, 0] == 13 and pos[p,1] == 2:
                to_be_removed.append(p)
                escape_times.append(step_number-1) #only the previous step matters
        pos = np.delete(pos, to_be_removed, axis=0)
        to_be_removed = []
        pos_ini = pos.copy()
        pos = np.add(pos, new_step(len(pos[:,0])))
        for n in range(len(pos[:,0])):
            if np.linalg.norm(pos[n,:]) >= 12: # Check if they hit the boundry
                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
                    slope_for_testing = (pos[n,1]-pos_ini[n,1])/(pos[n,0]-pos_ini[n,0])
                    y_intercept_for_testing = pos[n,1] - (slope_for_testing)*pos[n,0]
                    f2 = slope_for_testing * x_for_checking + y_intercept_for_testing  ## Draw a straight line between new point and initial point
                    solution_to_be_tested = sympy.solve(f1 - f2, x_for_checking)
                    if len(solution_to_be_tested)==1 and solution_to_be_tested[0] >= boundry_condition:
                       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
                    elif len(solution_to_be_tested) == 2 and solution_to_be_tested[1] >= boundry_condition:
                       pos[n,:] = [13,2]
                    else:
                       pos[n,:] = pos_ini[n,:]
                       new_maybe_correct_step = new_step(1)
                       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
                       while (np.linalg.norm(np.add(pos[n, :], new_maybe_correct_step)) >= 12 ):
                           new_maybe_correct_step = new_step(1)
                           number_of_tries = number_of_tries + 1
                           if number_of_tries == 5:
                               break
                       if number_of_tries != 5:
                           pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
                       elif number_of_tries != 5:
                           pos[n,:] = pos_ini[n,:]
                else:
                   pos[n,:] = pos_ini[n,:]
                   new_maybe_correct_step = new_step(1)
                   while (np.linalg.norm(np.add(pos[n, :], new_maybe_correct_step)) >= 12 ):
                       new_maybe_correct_step = new_step(1)
                   pos[n,:] = np.add(pos[n,:], new_maybe_correct_step)
        line100.set_data(pos[:, 0], pos[:, 1])
        fig100.canvas.draw()
        fig100.canvas.flush_events()   ## IF you want to see it uncomment the 4 lines
        plt.pause(0.0001)
    Mean_escape_time_list.append(np.mean(escape_times))
    Median_escape_time_list.append(np.median(escape_times))
    gapsizes.append(0.1 * a * np.pi)
    Mode_escape_time_list.append(mode(escape_times))
    print(f'Cycle {a} complete!')







Gapsize_mean_escape_time = np.column_stack([gapsizes, Mean_escape_time_list])
```

```python
Gapsize_median_escape_time = np.column_stack([gapsizes, Median_escape_time_list])
Gapszie_mode_escape_time = np.column_stack([gapsizes, Mode_escape_time_list])




def mean_escape_time(r, s, t , d):
    '''Takes 4 input parameters, radius, step size (epsilon), delta t, and number of dimensions.'''
    epsilon = s
    diffusion_coeff_in_function = (epsilon**2)/(2 * d * t)
    residence_time = (r**2)/(diffusion_coeff_in_function)  * (np.log10(epsilon**(-1)) + np.log10(2) + 8**(-1))
    return residence_time


fig101 = plt.figure(figsize=(14,7))

x_ticks = ['0.1  ', '0.2  ', '0.3  ', '0.4  ', '0.5  ']

x_values_for_task_10 = np.linspace(0, 0.5*np.pi, 1000)
y_values_for_task_10 = mean_escape_time(12, 0.5, 1, 2) + 0*x_values_for_task_10

#Mean
ax101 = plt.subplot(1, 3, 1)
ax101.plot(Gapsize_mean_escape_time[:,0], Gapsize_mean_escape_time[:,1], label='Mean escape time')
ax101.set_xticks(Gapsize_mean_escape_time[:,0])
ax101.set_xticklabels(x_ticks)
ax101.set_title('Figure 10.1: Mean prisoner escape time as a function of fence gap size', size=5.5, weight='bold')
ax101.set_xlabel('Gap Size (radians)')
ax101.set_ylabel('Mean escape time t (s)')
ax101.plot(x_values_for_task_10, y_values_for_task_10, label='Mean expected escape time')
plt.legend()


#Median
ax102 = plt.subplot(1,3,2)
ax102.plot(Gapsize_median_escape_time[:,0], Gapsize_median_escape_time[:,1], label='Median escape time')
ax102.set_xticks(Gapsize_mean_escape_time[:,0])
ax102.set_xticklabels(x_ticks)
ax102.set_title('Figure 10.2: Median prisoner escape time as a function of fence gap size', size=5.5, weight='bold')
ax102.set_xlabel('Gap Size (radians)')
ax102.set_ylabel('Mean escape time t (s)')
ax102.plot(x_values_for_task_10, y_values_for_task_10, label='Median expected escape time')
plt.legend()
```

```
#Mode
ax103 = plt.subplot(1,3,3)
ax103.plot(Gapszie_mode_escape_time[:,0], Gapszie_mode_escape_time
    [:,1], label='Modal escape time')
ax103.set_xticks(Gapsize_mean_escape_time[:,0])
ax103.set_xticklabels(x_ticks)
ax103.set_title('Figure 10.3: Mode prisoner escape time as a function
    of fence gap size', size=5.5, weight='bold')
ax103.set_xlabel('Gap Size (radians)')
ax103.set_ylabel('Mean escape time t (s)')
ax103.plot(x_values_for_task_10, y_values_for_task_10, label='Modal
    expected escape time')
plt.legend()





plt.show()
```

codes/Task 10.py