PHY1112: Assignment 6

> Pretty Pictures and Concise Captions

Assigned: February 13th, 2024

Due: February 27th, 2024

Learning Objectives

1. Learn more about available plotting functionality in matplotlib
2. Learn more about formatting and finalizing your python plots
3. Practice applying mathematical formulas to your NumPy arrays

Grade Breakdown

|  |  |  |  |
| --- | --- | --- | --- |
| Part | 1 | 2 | Total |
| Points | 19 | 15 | 33 |
| Score |  |  |  |

**Question 0: Quality Captions are Key**

In this assignment, you will be generating plots and saving them for submission. The plots should be formatted well, with proper titles, labelled axes, and **figure captions**.

A quality plot is easy for a reader to understand. A big part of this is including a good figure caption that:

* Is **clear and concise**. Your figure captions are not your analysis, they should not be too wordy.
* It should **accurately describe the content** and answer the question “what am I looking at?”. It may point out key elements of the figure.
* It should **include a Figure number** for reference in any analysis or discussion in the main text of the document.
* When relevant, it should **mention the methodology or source** of the data, for example whether this is from theory, experiment, simulation, etc.

Here is an example:

A graph of a graph showing a blue line

Description automatically generated

**Figure 1:** Real (blue) and imaginary (green) parts of the relative permittivity of an optical material described by a Lorentz oscillator model with a resonance at 380 nm.

**Question 1: Dicey Situation**

Referring back to Assignment 4, Question 2:

1. Write a function that takes in a number `number\_of\_dice` and returns all possible combinations of dice rolls[[1]](#footnote-2) as a 2 dimensional NumPy array, using `itertools.product`.

Make it general to any number of sides by adding a `number\_of\_sides` argument to your function.

The number of columns of your output array will equal the number of dice, and the number of rows will equal the number of possible combinations.

**(4 marks)**

1. Now consider 2 dice, each with 6 sides.
   * Write out all the possible combinations of dice rolls by hand.

**[(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(2,1),(2,2),(2,3),(2,4),(2,5),(2,6),(3,1),(3,2),(3,3),(3,4),(3,5),(3,6),(4,1),(4,2),(4,3),(4,4),(4,5),(4,6),(5,1),(5,2),(5,3),(5,4),(5,6),(6,1),(6,2),(6,3),(6,4),(6,5),(6,6)]**

* + Sum the rolls by hand, and

**[2,3,4,5,6,7,3,4,5,6,7,8,4,5,6,7,8,9,5,6,7,8,9,10,6,7,8,9,10,11,7,8,9,10,11,12]**

* + Count (again by hand) how many times each sum value appears.

**[(2:1),(3:2),(4:3),(5:4),(6:5),(7:6),(8:5),(9:4),(10:3),(11:2),(12:1)]**

* + Then, make a table that contains the possible sum value on the left, with how many times it appears on the right.

|  |  |
| --- | --- |
| **Sums** | **Occurences** |
| **2** | **1** |
| **3** | **2** |
| **4** | **3** |
| **5** | **4** |
| **6** | **5** |
| **7** | **6** |
| **8** | **5** |
| **9** | **4** |
| **10** | **3** |
| **11** | **2** |
| **12** | **1** |

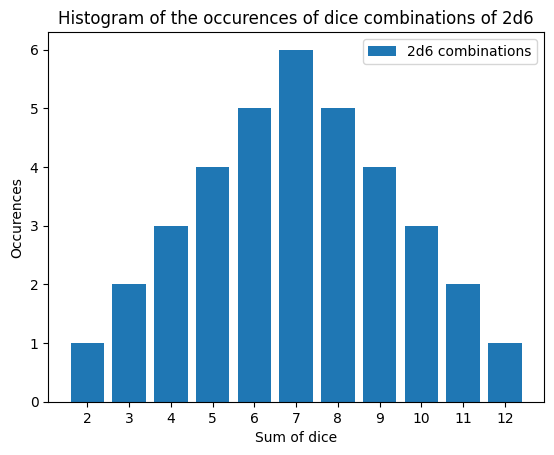
Provide the answers to each bullet point in your document.

**(4 marks)**

1. The chart you made in part ‘b’ is the data you need for creating a histogram that tells how probable each sum is when you roll two dice. To get the probability, you need to normalize your count data by dividing each number in the right hand column by the total number of possible rolls.

Implement the sums vs probability data as two 1D NumPy arrays. Use the ‘bar’ function to plot a probability histogram. Include appropriate title and axis labels. Take a screenshot of your plot and put it into your report.

**(2 marks)**

****

1. We are going to repeat part ‘b’ numerically through the following steps:

* Use your function from part ‘a’ to make an array with all the possible combinations of dice rolls. Print out your array and include a screenshot in your answers.

A screenshot of a computer

Description automatically generated

* Next, make a new array that contains, for each possible combination (*i.e*, each row), the sum of both dice. It should be a 1D NumPy array, with a length equal to the number of rows in the “all possible combinations of dice rolls” array. Print out your array and include a screenshot in your answers.

A number on a black background

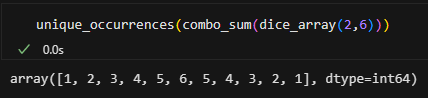
Description automatically generated

* Make another new 1D NumPy array that holds a sorted list of all the unique values of the sums (equivalent to the left side of your chart of part ‘b’). Print out your array and include a screenshot in your answers.

A screenshot of a computer

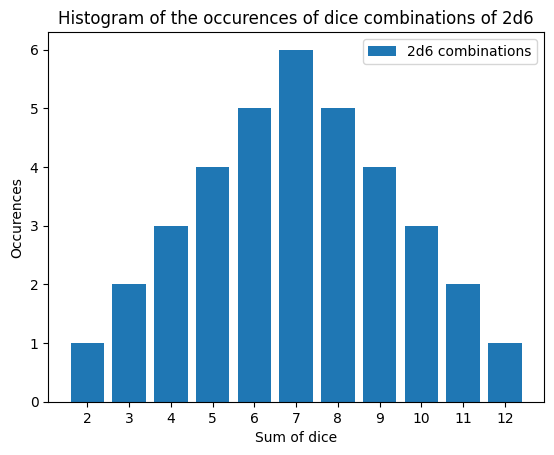
Description automatically generated

* Make another new array that holds how many times each value occurs in the “sums of both dice” array (equivalent to the right side of your chart of part ‘b’). Print out your array and include a screenshot in your answers.  
  **(4 marks)**

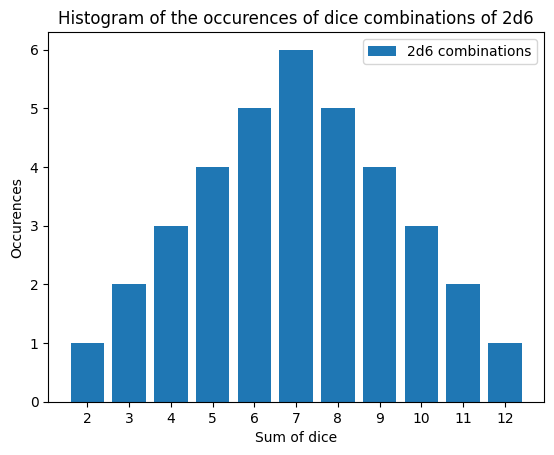
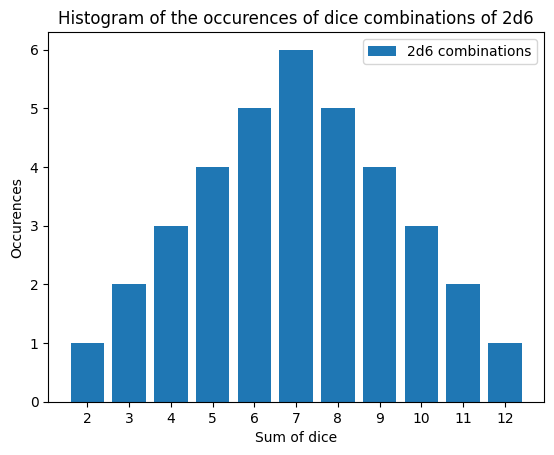


1. Use the ‘bar’ function to plot the probability histogram from the data you obtained in part ‘d’, properly normalized as before. Include appropriate title and axis labels. Take a screenshot of your plot and put it into your report. Do your results align with what you obtained by hand?

**(2 marks)**

~~~~

1. Be sure to submit your two plots with appropriate figure captions.  
   **(2 marks)**

**Figure 1. Hand array of 2d6 Figure 2. Function-defined array of 2d6**

**(19 marks total, 1 for docstrings/file header/variable naming/comments)**

**Question 2: Scatterbrained**

This question is a continuation of the Rigid Body problem from Assignment 5, Question 3.

For this assignment, a new data set is being provided that contains the masses of particles, “PHY1112\_A6\_Q2\_Masses.csv”.

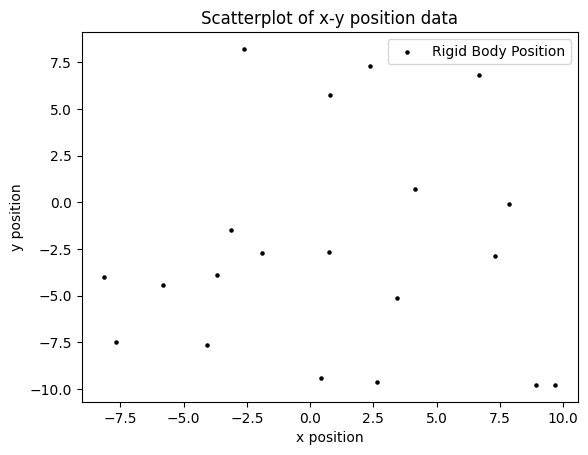
Data files containing the particles’ corresponding positions and velocities were given for Assignment 5 (“PHY1112\_A5\_Q3\_Positions.csv” and “PHY1112\_A5\_Q3\_Velocities”)

1. Recycling your previous code, read in the all the positions and velocities for the Rigid Bodies and store them into NumPy arrays.

**(1 mark)**

1. Using the `scatter` plotting function from matplotlib.pyplot, plot the (x,y) positions of the Rigid Bodies as black filled circles on an x-y graph. Don’t forget to include a title, axis labels and a legend. Include a snapshot of your figure.

**(3 marks)**



1. Read in all the masses for the Rigid Bodies and store them in a NumPy array.   
   **(1 mark)**

A screenshot of a computer

Description automatically generated

1. Determine the average speed and average mass of your Rigid Bodies, as well as the standard deviations of speed and mass. Report the values you obtained.

**(4 marks)**

**A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated**

1. Determine the x and y coordinates of the Center of Mass of this particle distribution, using the formulas:

Report your values.

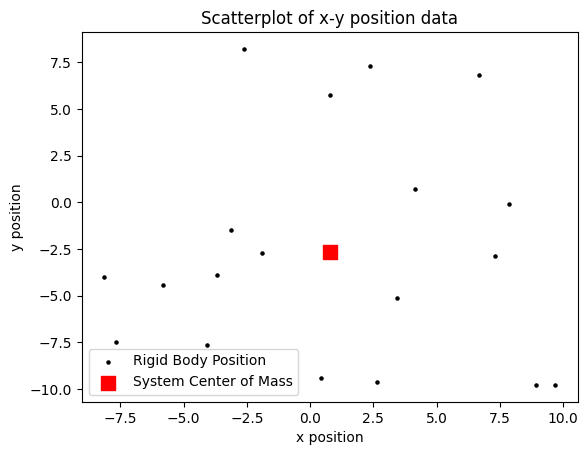
A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generated

Then, add this point to the figure you created in part ‘b’ as a red square, and ensure it is included in the legend. Include a snapshot of your figure.  
**(3 marks)**

**Figure 3.**

****

1. Be sure to submit your two plots with appropriate captions  
   **(2 marks)**

**(15 marks total, 1 for docstrings/file header/variable naming/comments)**

**CODE:**

'''

Filename:       a6\_300349204.py

Author:         Patrick Geraghty

Date Created:   2024-02-20

Date Modified:  2024-02-26

Description:    Conatins functions for Assignment 6. Functions include: 'dice\_array', 'hand\_histogram\_2d6', 'combo\_sum', 'unique\_combinations', 'unique\_occurrences', 'dice\_histogram', 'velocity', 'position', 'rigid\_position\_xy\_scatterplot', 'mass', average\_speed', 'average\_mass', 'average\_speed\_std\_dev', 'average\_mass\_std\_dev', 'x\_center\_of\_mass', 'y\_center\_of\_mass', 'system\_center\_of\_mass', and 'system\_scatterplot'. See docstrings for details.

'''

import matplotlib.pyplot as plt

import numpy as np

import itertools as it

# Question 1

# define a function to calculate the sum of two n-sided dice

*def* dice\_array(*number\_of\_dice*, *number\_of\_sides*):

    '''

    (int, int) -> np.array

    Returns an array of combinations of a number of n-sided dice.

    Preconditions: number\_of\_dice > 0, number\_of\_sides > 0, number\_of\_dice and number\_of\_sides are integers.

    '''

    # create a list to store all possible combinations of the dice

    combinations = []

    # iterate through the product of the range of the number of sides and the number of dice

    for i in it.product(range(1, number\_of\_sides+1), *repeat*=number\_of\_dice):

        # append the sum of the combination to the list

        combinations.append(i)

        # return the list as an array

    return np.array(combinations)

# define a function to display the histogram of the sum of 2d6 using pre-existing data

*def* hand\_histogram\_2d6():

    '''

    () -> None

    Plots a histogram of the sum of two six-sided dice and their occurence using data collected by observation.

    Preconditions: None

    '''

    # define the x and y values for the histogram

    x = np.array([2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

    y = np.array([1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1])

    # initialize a new figure for the plot

    plt.figure(1)

    # plot the histogram as a bar graph

    plt.bar(x, y, *label* = '2d6 combinations')

    # set the title and labels for the plot

    plt.title('Histogram of the occurences of dice combinations of 2d6')

    plt.xlabel('Sum of dice')

    plt.ylabel('Occurences')

    # set the x-axis ticks to the values of the dice combinations (matplot lib shows only even values by default, so we need to set the ticks manually)

    plt.xticks(x)

    # display the legend

    plt.legend()

    # display the plot

    plt.show()

*def* combo\_sum(*array*):

    '''

    (np.array) -> np.array

    Returns an array of the sums of the rows of the input array.

    Preconditions: array is a 2D numpy array.

    '''

    # return the sum of the rows of the array

    return np.sum(array, *axis*=1)

*def* unique\_combinations(*array*):

    '''

    (np.array) -> np.array

    Returns an array of the unique combinations of the input array.

    Preconditions: array is a 1D numpy array.

    '''

    # return the unique values of the input array

    return np.unique(array)

*def* unique\_occurrences(*array*):

    '''

    (np.array) -> np.array

    Returns an array of the occurrences of the unique values of the input array.

    Preconditions: array is a 1D numpy array.

    '''

    # define the occurences of the unique values (which are equal to the index number) of the input array

    bin\_occurences =  np.bincount(array)

    # simplify the array to remove the 0 values and return result

    return bin\_occurences[bin\_occurences != 0]

*def* dice\_histogram(*number\_of\_dice*, *number\_of\_sides*):

    '''

    () -> None

    Plots a histogram of the unique sums of m n-sided dice combinations and the frequency of their occurences.

    Preconditions: None

    '''

    # define the x and y values for the histogram

    x = unique\_combinations(combo\_sum(dice\_array(number\_of\_dice, number\_of\_sides)))

    y = unique\_occurrences(combo\_sum(dice\_array(number\_of\_dice, number\_of\_sides)))

    # initialize a new figure for the plot

    plt.figure(2)

    # plot the histogram as a bar graph

    plt.bar(x, y, *label*='{}d{} combinations'.format(number\_of\_dice, number\_of\_sides))

    # set the title and labels for the plot

    plt.title('Histogram of the occurences of dice combinations of {}d{}'.format(number\_of\_dice, number\_of\_sides))

    plt.xlabel('Sum of dice')

    plt.ylabel('Occurences')

    # set the x-axis ticks to the values of the dice combinations (matplot lib shows only even values by default, so we need to set the ticks manually)

    plt.xticks(x)

    # display the legend

    plt.legend()

    # display the plot

    plt.show()

# Question 2

*def* velocity():

    '''

    () -> np.array

    Returns an array of velocities containing the relevant data in 'PHY1112\_A5\_Q3\_Velocities.csv'.

    Preconditions: 'PHY1112\_A5\_Q3\_Velocities.csv' is a valid file.

    '''

    # load the data from the file using np.genfromtxt. Define necessary columns, skip the header, identify the separator, and define the data type as float

    return np.genfromtxt('PHY1112\_A5\_Q3\_Velocities.csv', *skip\_header*=1, *usecols*=(1,2,3), *delimiter*=',', *dtype*=*float*)

# define function 'position' to contain the data from 'PHY1112\_A5\_Q3\_Positions.csv' in an array

*def* position():

    '''

    () -> np.array

    Returns an array of positions containing the relevant data in 'PHY1112\_A5\_Q3\_Positions.csv'.

    Preconditions: 'PHY1112\_A5\_Q3\_Positions.csv' is a valid file.

    '''

    # load the data from the file using np.genfromtxt. Define necessary columns, skip the header, identify the separator

    return np.genfromtxt('PHY1112\_A5\_Q3\_Positions.csv', *skip\_header*=1, *usecols*=(1,2,3), *delimiter*=',', *dtype*=*float*)

*def* rigid\_position\_xy\_scatterplot():

    '''

    () -> None

    Plots a scatterplot of the x-y position data from the 'PHY1112\_A5\_Q3\_Positions.csv' and 'PHY1112\_A5\_Q3\_Velocities.csv' files.

    Preconditions: None

    '''

    # initialize a new figure for the plot

    plt.figure(3)

    # plot the scatterplot of the position and velocity data

    x = position()[:,0]

    y = position()[:,1]

    plt.scatter(x, y, *label*='Rigid Body Position', *color*='black', *marker*='o', *s*=5)

    # set the title and labels for the plot

    plt.title('Scatterplot of x-y position data')

    plt.xlabel('x position')

    plt.ylabel('y position')

    # display the legend

    plt.legend()

    # display the plot

    # plt.show()

# define function to read mass data from 'PHY1112\_A6\_Q2\_Masses.csv'

*def* mass():

    '''

    () -> np.array

    Returns an array of masses containing the relevant data in 'PHY1112\_A6\_Q2\_Masses.csv'.'

    Preconditions: 'PHY1112\_A6\_Q2\_Masses.csv' is a valid file.

    '''

    # load the data from the file using np.genfromtxt. Define necessary columns, skip the header, identify the separator, and define the data type as float

    return np.genfromtxt('PHY1112\_A6\_Q2\_Masses.csv', *skip\_header*=1, *usecols*=(1), *delimiter*=',', *dtype*=*float*)

# import velocity\_magnitude from a5\_300349204.py for later use

*def* velocity\_magnitude():

    '''

    () -> np.array

    Returns an array of the magnitude of the velocity for each body using data from the 'velocity()' function.

    Preconditions: None

    '''

    vel\_x = velocity()[:,0]     # set x velocity to the first column of the velocity array

    vel\_y = velocity()[:,1]     # set y velocity to the second column of the velocity array

    vel\_z = velocity()[:,2]     # set z velocity to the third column of the velocity array

    # calculate the magnitude of the velocity using the Pythagorean theorem and return the array

    return np.array([np.sqrt(vel\_x[i]\*\*2 + vel\_y[i]\*\*2 + vel\_z[i]\*\*2) for i in range(len(vel\_x))])

# define function average\_speed to calculate the average speed of the bodies using the data from velocity\_magnitude

*def* average\_speed():

    '''

    () -> float

    Returns the average speed of the bodies using the data from the 'velocity()' function.

    Preconditions: None

    '''

    # return the average speed of the bodies using the magnitude of the velocity and the number of bodies

    return np.mean(velocity\_magnitude())

# define function average\_mass to calculate the average mass of the bodies using the data from mass

*def* average\_mass():

    '''

    () -> float

    Returns the average mass of the bodies using the data from the 'mass()' function

    Preconditions: None

    '''

    # return the average mass of the bodies using the mass array

    return np.mean(mass())

# define function average\_speed\_std\_dev to calculate the standard deviation of the average speed of the bodies using the data from velocity\_magnitude

*def* average\_speed\_std\_dev():

    '''

    () -> float

    Returns the standard deviation of the average speed of the bodies using the data from the 'velocity()' function.

    Preconditions: None

    '''

    # return the standard deviation of the average speed of the bodies using the magnitude of the velocity

    return np.std(velocity\_magnitude())

# define function average\_mass\_std\_dev to calculate the standard deviation of the average mass of the bodies using the data from mass

*def* average\_mass\_std\_dev():

    '''

    () -> float

    Returns the standard deviation of the average mass of the bodies using the data from the 'mass()' function

    Preconditions: None

    '''

    # return the standard deviation of the average mass of the bodies using the mass array

    return np.std(mass())

# define function x\_center\_of\_mass to calculate the x-coordinate of the center of mass of the overall system using the data from mass and position

*def* x\_center\_of\_mass():

    '''

    () -> float

    Returns the x-coordinate of the center of mass of the overall system using the data from the 'mass()' and 'position()' functions.

    Preconditions: None

    '''

    # calculate the x-coordinate of the center of mass

    return np.sum(mass() \* position()[:,0]) / np.sum(mass())

# define function y\_center\_of\_mass to calculate the y-coordinate of the center of mass of the overall system using the data from mass and position

*def* y\_center\_of\_mass():

    '''

    () -> float

    Returns the y-coordinate of the center of mass of the overall system using the data from the 'mass()' and 'position()' functions.

    Preconditions: None

    '''

    # calculate the y-coordinate of the center of mass

    return np.sum(mass() \* position()[:,1]) / np.sum(mass())

# define function system\_center\_of\_mass to display the x and y coordinates of the center of mass of the overall system on the plot

*def* system\_center\_of\_mass():

    '''

    () -> None

    Prints the x and y coordinates of the center of mass of the overall system on the plot.

    Preconditions: None

    '''

    x = x\_center\_of\_mass()

    y = y\_center\_of\_mass()

    # # initialize the original figure for the plot

    plt.figure(3)

    # plot the center of mass on the scatterplot

    plt.scatter(x, y, *label*='System Center of Mass', *color*='red', *marker*='s', *s*=100)

    # show legend

    plt.legend()

    # display the plot

    # plt.show()

# define function system\_scatterplot to plot the scatterplot of the x-y position data from the 'PHY1112\_A5\_Q3\_Positions.csv' and 'PHY1112\_A5\_Q3\_Velocities.csv' files, as well as the center of mass of the system

*def* system\_scatterplot():

    '''

    () -> None

    Plots a scatterplot of the x-y position data from the 'PHY1112\_A5\_Q3\_Positions.csv' and 'PHY1112\_A5\_Q3\_Velocities.csv' files, as well as the center of mass of the system.

    Preconditions: None

    '''

    # call the rigid\_position\_xy\_scatterplot function to plot the position of each individual body

    rigid\_position\_xy\_scatterplot()

    # call the system\_center\_of\_mass function to plot the center of mass of the system

    system\_center\_of\_mass()

    # show the plot

    plt.show()

1. Different orderings of the same numbers are considered distinct. For example, in the case of 2 dice, a roll of (1,2) is distinct from a roll of (2,1) and are both considered as possible combinations. [↑](#footnote-ref-2)