PHY1112: Assignment 7

> Error Erudition and Vigourous Vectors

Assigned: February 27th, 2024

Due: March 5th, 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 | 16 | 19 | 35 |
| Score |  |  |  |

**Question 1: The Weatherperson is Never Exact Enough.**

1. Like you did in the lab, write a script to read in the lab5 weather data and extract the Ottawa high and low temperatures for March 2022, storing them in NumPy arrays.

Rather than assuming a measurement error that is a constant value, like we did for lab 7, we will instead assume that the measurement error for each data point is 10% of the measured temperature. It will thus be different for each data point.

Generate the errors on the measurements, store them in NumPy array, and record in your solutions document the error values you obtained. Make sure to only output 2 decimal places for each number using f-strings.  
**(3 marks)**

0.93, 0.27, 0.66, 0.30, 0.06, 1.40, 0.32, 0.16, 0.25, 0.26, 0.05, 0.22, 0.33, 0.53, 0.29, 0.60, 1.56, 1.04, 0.45, 0.71, 0.60, 0.51, 0.48, 1.03, 1.03, 0.78, 0.10, 0.76, 0.03, 0.36, 1.41, 1.62, 1.46, 1.99, 2.05, 1.53, 0.28, 0.42, 0.66, 0.53, 0.44, 0.45, 0.99, 1.29, 0.41, 0.36, 0.28, 0.04, 0.22, 0.13, 0.03, 0.29, 0.53, 0.58, 0.02, 0.11, 0.06, 1.19, 1.33, 1.06, 1.01, 0.11

1. Use the `errorbar` plotting function in matplotlib to generate a plot of the March 2022 Ottawa highs and lows with markers (rather than as lines as we did in lab 7). Show the error as error bars (rather than shaded areas as we did for lab 7). Continue to format your plots properly and submit them with figure captions. Take a snapshot of your figure and include it in your solutions document.  
   **(3 marks)**

A graph with red and blue dots

Description automatically generated

Figure 1. A graph of the high and low temperatures in the month of March 2022. High temperatures are shown in red and low temperatures are shown in cyan. Error bars show a 10% margin of error.

1. Write out a formula to determine the daily temperature range. Given that we know the error for the high and low temperatures, write out a formula for the error on the daily temperature range using propagation of errors.

Daily temperature range =

Daily temperature range error =

Implement your formulas into your script to obtain an array containing the daily temperature range, and another containing the errors on the daily temperature range.  
**(3 marks)**

The above formula for error either does not work, or is wrong. I don’t know but the equation under the radical becomes negative for most values (which we definitely don’t want). Taking the absolute value is a possible solution.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Use the `errorbar` plotting function in matplotlib to generate a plot of the Ottawa March 2022 daily temperature range with markers and error bars. Continue to format your plots properly and submit them with figure captions. Take a snapshot of your figure and include it in your solutions document.  
   **(3 marks)**

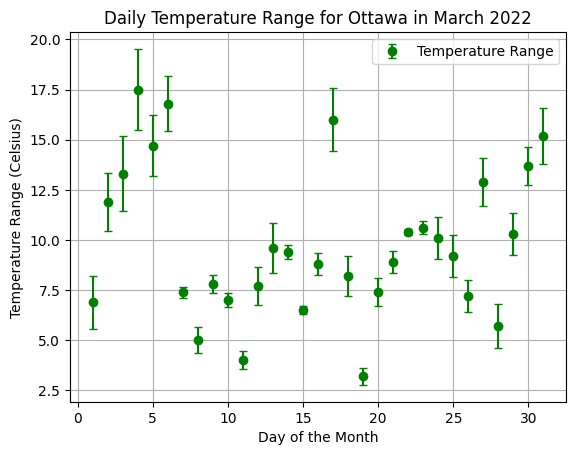


Figure 2. A graph of the daily temperature for the month of March 2022 in Ottawa. Temperature ranges are represented in green with a propagated error margin.

1. Write out a formula to determine the average temperature range. Then write out a formula for the error on this average using propagation of errors. Implement these two formulas into your script and record the values you obtained in your solutions document. Again, remember to only output 2 decimal places for each value using f-strings.

**(3 marks)**

Average daily temperature = (high + low)/2

Average daily temperature error =

(I assume this is what you mean by average temperature range?)

A screenshot of a computer program

Description automatically generated

A graph with yellow dots

Description automatically generated

Figure 3. A graph of the average daily temperature for the month of March 2022. Average daily temperatures are represented in yellow with a propagated error margin.

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

**Question 2: Racing to Shift Your Perspective**

You are an observer (and the referee) in a race between a car and a truck, who go in different directions. Whoever gets the farthest away from you in 1 minute wins!

At the start ( seconds), both the car and the truck are located at the origin, km, where you are also located. You stay at that position throughout the entire race.

When the timer starts, both vehicles begin moving immediately at a constant velocity given by (in your frame of reference)

,

and when the 1 minute time ends they will instantly stop.

At any time , the positions of the vehicles (in your reference frame) will be given by:

,

.

We would like to plot the positions of the car and truck as trajectories in time, according to your (the observer’s) frame of reference.

We start by defining a 1D NumPy array for the time variable using np.linspace(). We then calculate and for each time in our time array by applying an outer product to and via np.outer().

Time will be a 1D with a length according to the number of time points we desire. The constant velocities are both vectors with two components. Thus and will be 2D arrays of size .

Putting this all together, we plot the 2D positions of the car and truck for all the time points, as well as determine the final distances and who is the winner, via the following script. There are some lines of code missing that you will need to fill in as part of this question.

import numpy as np

import matplotlib.pyplot as plt

def position(velocity, time):

    '''uses an outer product between velocity and time to generate

    a 2d position array for all times where the columns are the vector

    components and the rows represent different time points'''

    #### INSERT YOUR CODE HERE ####

    ###############################

def magnitude(vector):

    '''returns the square root of the inner product of vector with itself'''

    #### INSERT YOUR CODE HERE ####

    ###############################

def determine\_distances\_and\_winner(r1, s1, r2, s2):

    '''returns the final distances traveled by s1 (string) and s2 (string)

    with position vectors r1 and r2, whose columns are the components

    of the vector and whose rows represent different time points.

    It also returns a string regarding who won the race, s1 or s2

    if they tied, or if a winner cannot be determined'''

    #### INSERT YOUR CODE HERE ####

    ###############################

    return final\_distance\_1, final\_distance\_2, string\_with\_winner

# the velocites are in the observer's frame of reference (road frame)

v\_observer= np.array([0,0])           # km/s

v\_car     = np.array([-45,70])/3600   # converted to km/s

v\_truck   = np.array([50,60])/3600    # converted to km/s

t = np.linspace(0, 60, num=13)  # 0 to 60 seconds, in steps of 5 sec

# calculate position vectors as a function of time

r\_observer = position(v\_observer, t)

r\_car      = position(v\_car, t)

r\_truck    = position(v\_truck, t)

# finding the final distances and the winner

observer\_dist = 0

car\_dist, truck\_dist, winner\_string = determine\_distances\_and\_winner(r\_car, "car", r\_truck, "truck")

# plot the positions of the observer, car and truck (rx versus ry for all the time points) with markers

#### INSERT YOUR CODE HERE ####

###############################

plt.title(f"Trajectories during the race. {winner\_string}")

plt.xlabel("x axis (km)")

plt.ylabel("y axis (km)")

plt.axis("equal")

plt.legend()

plt.show()

Here is the plot that the script should generate:

A graph of a car

Description automatically generated

Figure 1: Trajectory in steps of 5 seconds of observer (red dots), car (blue crosses) and truck (green x’s) during a 1 minute race. We are taking the frame of reference to be the observer frame (which is also the road’s frame of reference)

Now consider a second race between the car and truck where

Use the script provided above, and add all missing functions and any other required details in order to:

1. Determine and print to the terminal the and positions of the car and truck at 5 second intervals. Take a snapshot of your results.

**(3 marks)**

A screenshot of a computer program

Description automatically generated

1. Determine and print to the terminal the speeds of the car and truck. Also do this by hand as a check.

**(3 marks)**

****

1. Determine and print to the terminal how far away they are from the starting point after 1 minute. Also do this by hand as a check. Who won?

**(3 marks)**

****

**Thus the truck won.**

1. Plot the car and truck trajectories, as we did in Figure 1, and include a snapshot here. Don’t forget to format your plots properly. Take a snapshot of your figure and include it in your solutions document with a figure caption.

**(3 marks)**

A graph of a truck

Description automatically generated

Figure 1. Trajectory in steps of 5 seconds of observer (red dots), car (blue crosses) and truck (green x’s) during a 1 minute race. We are taking the frame of reference to be the observer frame (which is also the road’s frame of reference)

We have now shown the system from the stationary reference frame of the observer (you!). However, if you’ve ever ridden in a car, you know that the world appears to move around the car. That is because the moving car is in a different reference frame than someone standing still on the side of the road.

The plotted reference frame in Figure 1 above is from the perspective of an observer that is standing still on the road. Let’s instead consider the same race, but from the perspective of the ***car***. We consider the car to be at rest in its own reference frame, and the car will be the referee of the race between the truck and the observer.

To proceed, we transform the velocities of the observer (which appears to move according to the car) and the truck. We do this by subtracting the velocity of the car from the observer and truck velocities.

This can be done by adding the following code snippets in the main part of the above script:

# transform the velocities to the car reference frame

v\_observer= v\_observer - v\_car

v\_truck   = v\_truck - v\_car

v\_car     = np.array([0,0]) # change this only after v\_observer and v\_truck

and

# finding the final distances and the winner

car\_dist = 0

observer\_dist, truck\_dist, winner\_string = determine\_distances\_and\_winner(r\_observer, "observer", r\_truck, "truck")

Here is the plot that is generated for the first race (using the same velocities for the car and truck as was used to generate Figure 1), but now in the reference frame in the car. Note that the same script was used as before, except that we added the code snippets above.

A graph of a truck

Description automatically generated

Figure 2: Trajectory in steps of 5 seconds of observer (red dots), car (blue crosses) and truck (green x’s) during a 1 minute race. We are taking the frame of reference to be one where the car is stationary.

This plot now well reflects what would be seen from the car’s reference frame during the first race. The truck won the race according to the car because it moved farther away from the car than the observer did.

1. Repeat parts ‘a’ to ‘d’ above, but from the reference frame of the truck. Like in parts ‘a’ to ‘d’, do this for the for the second race, where, recall,

Do not change any of the code that you wrote for parts ‘a’ to ‘d’, that is, the parts that were indicated by #### INSERT YOUR CODE HERE ####

You may change parts in the main script, similar to what was done in the snippets above.

**(6 marks)**

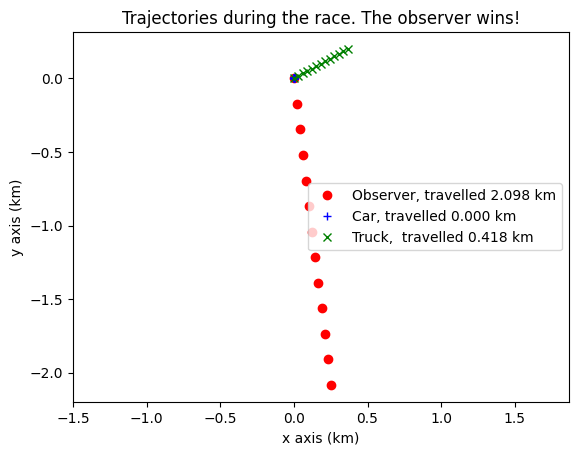
****

Figure 2: Trajectory in steps of 5 seconds of observer (red dots), car (blue crosses) and truck (green x’s) during a 1 minute race. We are taking the frame of reference to be one where the car is stationary.

**A computer screen with white text

Description automatically generated**

**A black screen with white text

Description automatically generated**

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

CODE:

'''

Filename:       a7\_part1.py

Author:         Patrick Geraghty

Date Created:   2024-03-04

Date ModifiedL  2024-03-05

Description:    Assignment 7: Question 1. The Weatherperson is Never Exact Enough

'''

import numpy as np

import matplotlib.pyplot as plt

# Question 1: The Weatherperson is Never Exact Enough

*def* load\_March\_data():

    '''

    () -> np.array

    This function loads the high and low temperatures for Ottawa in March 2022.

    Preconditions: None

    '''

    return np.genfromtxt('weather\_data\_lab5.csv', *delimiter*=',', *usecols*=(9,11), *skip\_header*=1, *dtype*=*float*)[59:90]

*def* error\_array(*error*=0.1):

    '''

    (float) -> np.array

    This function returns an array of errors for the high and low temperatures in March 2022.

    Preconditions: If error is defined, it is a real number.

    '''

    data = load\_March\_data()    # load the data

    return np.array(np.abs([data[:,0]\*error, data[:,1]\*error])) # return the array of errors

*def* print\_error():

    '''

    () -> None

    This function prints the error array for the high and low temperatures in March 2022 as f-strings.

    Preconditions: None

    '''

    data = error\_array()        # load the data

    for i in range(len(data)):  # print the error array for each element

        for j in (data[i]):

            print(*f*'{j*:.2f*}', *end*=', ')

*def* plot\_temperature\_data():

    '''

    () -> None

    This function plots the high and low temperatures for Ottawa in March 2022 with error bars.

    Preconditions: high\_temperatures and low\_temperatures data exist and are both np.arrays of the same length

    '''

    plt.figure(1)

    data = load\_March\_data()        # load the data

    day = np.arange(1,32)           # specify the days of the month

    high\_temperatures = data[:,0]   # specify the high temperatures

    low\_temperatures = data[:,1]    # specify the low temperatures

    errors = error\_array(data)      # specify the errors

    # plot the high and low temperatures with error bars

    plt.errorbar(day, high\_temperatures, *yerr*=np.abs(errors[0]), *fmt*='ro', *capsize*=3, *label*='Highs')

    plt.errorbar(day, low\_temperatures, *yerr*=np.abs(errors[1]), *fmt*='co', *capsize*=3, *label*='Lows')

    # label the plot and show it

    plt.xlabel('Day of the Month')

    plt.ylabel('Temperature (Celsius)')

    plt.title('Monthly High and Low Temperatures for Ottawa in March 2022')

    plt.grid()

    plt.legend()

    plt.show()

*def* daily\_temperature\_range():

    '''

    () -> np.array

    This function returns the daily temperature range for Ottawa in March 2022.

    Preconditions: None

    '''

    data = load\_March\_data()                    # load the data

    high\_temperatures = data[:,0]               # specify the high temperatures

    low\_temperatures = data[:,1]                # specify the low temperatures

    return high\_temperatures - low\_temperatures # return the daily temperature range

*def* range\_error():

    '''

    () -> np.array

    This function returns the error for the daily temperature range for Ottawa in March 2022.

    Preconditions: None

    '''

    high\_error = error\_array()[0]                           # specify the high temperature error

    low\_error = error\_array()[1]                            # specify the low temperature error

    return np.sqrt(np.abs(high\_error\*\*2 - low\_error\*\*2))    # return the daily temperature range error

*def* plot\_temperature\_range():

    '''

    () -> None

    This function plots the daily temperature range for Ottawa in March 2022 with error bars.

    Preconditions: daily\_temperature\_range and range\_error functions exist

    '''

    plt.figure(2)

    day = np.arange(1,32)                   # specify the days of the month

    range\_data = daily\_temperature\_range()  # specify the daily temperature range

    range\_errors = range\_error()            # specify the daily temperature range error

    # plot the daily temperature range with error bars

    plt.errorbar(day, range\_data, *yerr*=range\_errors, *fmt*='go', *capsize*=3, *label*='Temperature Range')

    # label the plot and show it

    plt.xlabel('Day of the Month')

    plt.ylabel('Temperature Range (Celsius)')

    plt.title('Daily Temperature Range for Ottawa in March 2022')

    plt.grid()

    plt.legend()

    plt.show()

*def* average\_daily\_temperature():

    '''

    () -> np.array

    This function returns the average daily temperature for Ottawa in March 2022.

    Preconditions: None

    '''

    data = load\_March\_data()                            # load the data

    high\_temperatures = data[:,0]                       # specify the high temperatures

    low\_temperatures = data[:,1]                        # specify the low temperatures

    return (high\_temperatures + low\_temperatures) / 2   # return the average daily temperature

*def* average\_daily\_error():

    '''

    () -> np.array

    This function returns the error for the average daily temperature for Ottawa in March 2022.

    Preconditions: None

    '''

    high\_error = error\_array()[0]                                       # specify the high temperature error

    low\_error = error\_array()[1]                                        # specify the low temperature error

    return np.sqrt(np.abs(0.25 \* high\_error\*\*2 + 0.25 \* low\_error\*\*2))  # return the average daily temperature error

*def* plot\_average\_temperature():

    '''

    () -> None

    This function plots the average daily temperature for Ottawa in March 2022 with error bars.

    Preconditions: average\_daily\_temperature and average\_daily\_error functions exist

    '''

    plt.figure(3)

    day = np.arange(1,32)                       # specify the days of the month

    average\_data = average\_daily\_temperature()  # specify the average daily temperature

    average\_errors = average\_daily\_error()      # specify the average daily temperature error

    # plot the average daily temperature with error bars

    plt.errorbar(day, average\_data, *yerr*=average\_errors, *fmt*='yo', *capsize*=3, *label*='Average Temperature')

    # label the plot and show it

    plt.xlabel('Day of the Month')

    plt.ylabel('Temperature (Celsius)')

    plt.title('Average Daily Temperature for Ottawa in March 2022')

    plt.grid()

    plt.legend()

    plt.show()

'''

Filename:       a7\_part2.py

Author:         Patrick Geraghty

Date Created:   2024-03-04

Date Modified:  2024-03-05

Description:    Assignment 7: Question 2. Racing to Shift Your Perspective

'''

import numpy as np

import matplotlib.pyplot as plt

# Question 2: Racing to Shift Your Perspective

import numpy as np

import matplotlib.pyplot as plt

*def* position(*velocity*, *time*):

    '''uses an outer product between velocity and time to generate

    a 2d position array for all times where the columns are the vector

    components and the rows represent different time points'''

    #### INSERT YOUR CODE HERE ####

    return np.outer(velocity, time)

    ###############################

*def* magnitude(*vector*):

    '''returns the square root of the inner product of vector with itself'''

    #### INSERT YOUR CODE HERE ####

    return np.sqrt(np.inner(vector, vector))

    ###############################

*def* determine\_distances\_and\_winner(*r1*, *s1*, *r2*, *s2*):

    '''returns the final distances traveled by s1 (string) and s2 (string)

    with position vectors r1 and r2, whose columns are the components

    of the vector and whose rows represent different time points.

    It also returns a string regarding who won the race, s1 or s2

    if they tied, or if a winner cannot be determined'''

    #### INSERT YOUR CODE HERE ####

    final\_distance\_1 = *str*(magnitude(r1[:,-1]))     # final distance of s1

    final\_distance\_2 = *str*(magnitude(r2[:,-1]))     # final distance of s2

    # determine the winner given the final distances (conditional statements)

    if final\_distance\_1 < final\_distance\_2:

        string\_with\_winner = *f*'The {s2} wins!'

    elif final\_distance\_1 > final\_distance\_2:

        string\_with\_winner = *f*'The {s1} wins!'

    elif final\_distance\_1 == final\_distance\_2:

        string\_with\_winner = 'The vehicles tied!'

    else:

        string\_with\_winner = 'A winner cannot be determined'

    ###############################

    return final\_distance\_1, final\_distance\_2, string\_with\_winner

*def* road\_reference\_race():

    '''

    () -> None

    Plots the positions of the observer, car and truck over a 60 second race in increments of 5 seconds from the observer (road perspective). Prints positions over 5 second intervals, vehicles speeds, and distance from the origin after 60 seconds.

    Preconditions: None

    '''

    # the velocites are in the observer's frame of reference (road frame)

    v\_observer= np.array([0,0])           # km/s

    v\_car     = np.array([-15,125])/3600   # converted to km/s

    v\_truck   = np.array([7,137])/3600    # converted to km/s

    t = np.linspace(0, 60, *num*=13)  # 0 to 60 seconds, in steps of 5 sec

    # calculate position vectors as a function of time

    r\_observer = position(v\_observer, t)

    r\_car      = position(v\_car, t)

    r\_truck    = position(v\_truck, t)

    # finding the final distances and the winner

    observer\_dist = 0

    car\_dist, truck\_dist, winner\_string = determine\_distances\_and\_winner(r\_car, "car", r\_truck, "truck")

    # plot the positions of the observer, car and truck (rx versus ry for all the time points) with markers

    #### INSERT YOUR CODE HERE ####

    plt.figure(1)

    # plot the positions of the observer, car and truck

    plt.plot(r\_observer[0], r\_observer[1], 'ro', *label*=(*f*'Observer, travelled {*float*(observer\_dist)*:.3f*} km'))

    plt.plot(r\_car[0], r\_car[1], 'b+', *label*=(*f*'Car, travelled {*float*(car\_dist)*:.3f*} km'))

    plt.plot(r\_truck[0], r\_truck[1], 'gx', *label*=(*f*'Truck,  travelled {*float*(truck\_dist)*:.3f*} km'))

    ###############################

    plt.title(*f*"Trajectories during the race. {winner\_string}")

    plt.xlabel("x axis (km)")

    plt.ylabel("y axis (km)")

    plt.axis("equal")

    plt.legend()

    plt.show()

    # print the positions of the observer, car and truck over 5 second intervals

    for i in range(0, len(t)):

        print(*f*"Time: {t[i]} seconds")

        print(*f*"Car position: x = {r\_car[0,i]}, y = {r\_car[1,i]}")

        print(*f*"Truck position: x = {r\_truck[0,i]}, y = {r\_truck[1,i]}")

        print()

    # print the speeds of the car and truck

    print(*f*'The car is travelling at {magnitude(v\_car)*:.3f*} km/s')

    print(*f*'The truck is travelling at {magnitude(v\_truck)*:.3f*} km/s', *end*='\n\n')

    # print the final distances of the car and truck

    print(*f*'The car is {*float*(car\_dist)*:.3f*} km from the origin after 60 seconds')

    print(*f*'The truck is {*float*(truck\_dist)*:.3f*} km from the origin after 60 seconds')

*def* transformed\_race():

    '''

    () -> None

    Plots the positions of the observer, car and truck over a 60 second race in increments of 5 seconds from the car (car perspective). Prints positions over 5 second intervals, truck and car speeds, and distance from the origin after 60 seconds.'''

    # the velocites are in the observer's frame of reference (road frame)

    v\_observer= np.array([0,0])           # km/s

    v\_car     = np.array([-15,125])/3600   # converted to km/s

    v\_truck   = np.array([7,137])/3600    # converted to km/s

    v\_observer= v\_observer - v\_car

    v\_truck   = v\_truck - v\_car

    v\_car     = np.array([0,0]) # change this only after v\_observer and v\_truck

    t = np.linspace(0, 60, *num*=13)  # 0 to 60 seconds, in steps of 5 sec

    # calculate position vectors as a function of time

    r\_observer = position(v\_observer, t)

    r\_car      = position(v\_car, t)

    r\_truck    = position(v\_truck, t)

    # finding the final distances and the winner

    car\_dist = 0

    observer\_dist, truck\_dist, winner\_string = determine\_distances\_and\_winner(r\_observer, "observer", r\_truck, "truck")

    # plot the positions of the observer, car and truck (rx versus ry for all the time points) with markers

    #### INSERT YOUR CODE HERE ####

    plt.figure(2)

    # plot the positions of the observer, car and truck

    plt.plot(r\_observer[0], r\_observer[1], 'ro', *label*=(*f*'Observer, travelled {*float*(observer\_dist)*:.3f*} km'))

    plt.plot(r\_car[0], r\_car[1], 'b+', *label*=(*f*'Car, travelled {*float*(car\_dist)*:.3f*} km'))

    plt.plot(r\_truck[0], r\_truck[1], 'gx', *label*=(*f*'Truck,  travelled {*float*(truck\_dist)*:.3f*} km'))

    ###############################

    plt.title(*f*"Trajectories during the race. {winner\_string}")

    plt.xlabel("x axis (km)")

    plt.ylabel("y axis (km)")

    plt.axis("equal")

    plt.legend()

    plt.show()

    # print the positions of the observer and truck over 5 second intervals

    for i in range(0, len(t)):

        print(*f*"Time: {t[i]} seconds")

        print(*f*"Observer position: x = {r\_observer[0,i]}, y = {r\_observer[1,i]}")

        print(*f*"Truck position: x = {r\_truck[0,i]}, y = {r\_truck[1,i]}")

        print()

    # print the speeds of the observer and truck

    print(*f*'The observer is travelling at {magnitude(v\_observer)*:.3f*} km/s')

    print(*f*'The truck is travelling at {magnitude(v\_truck)*:.3f*} km/s', *end*='\n\n')

    # print the final distances of the observer and truck

    print(*f*'The observer is {*float*(observer\_dist)*:.3f*} km from the origin after 60 seconds')

    print(*f*'The truck is {*float*(truck\_dist)*:.3f*} km from the origin after 60 seconds')