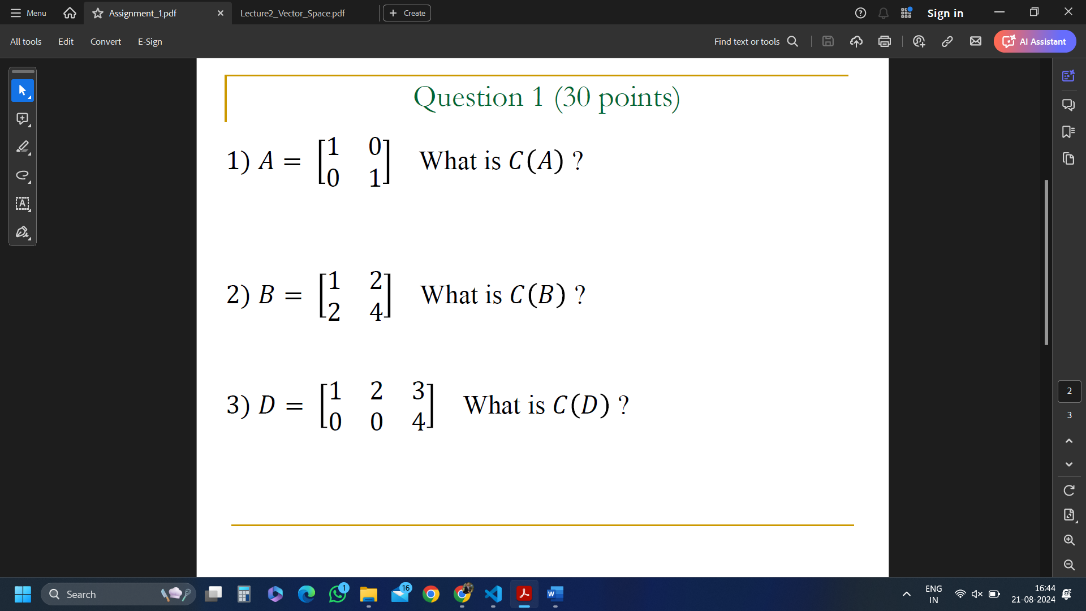
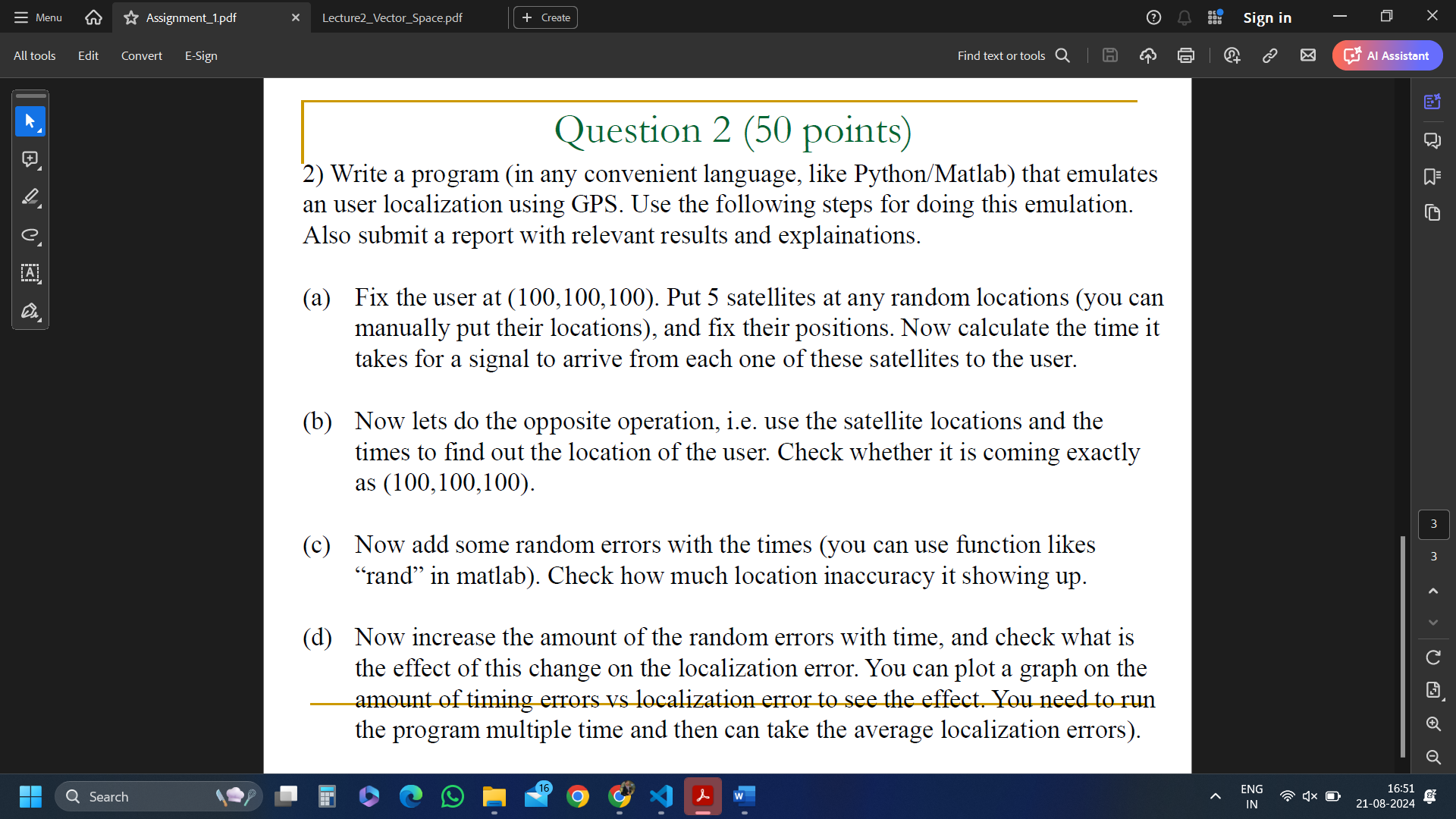
**Assignment-1**



C(A) = R2

C(B) = R1

C(D) = R2



import numpy as np

from sympy import \*

import matplotlib.pyplot as plt

# speed of light in m/s

speed\_of\_light = 3\*(10^8)

# Fixing user at (100, 100, 100)

user\_position = np.array([100, 100, 100])

# Fixing satellites at random location

satellite\_positions = np.array([

    [5000, 1000, 2000],

    [1000, 6000, 7000],

    [3000, 7000, 9000],

    [8000, 2000, 1000],

    [6000, 9000, 3000]

])

print("\nPart (a)")

# Calculating euclidean distance between user and each satellite

distances = np.sqrt(np.sum((satellite\_positions - user\_position)\*\*2, axis=1))

# Time taken by each satellite to send signal to user

times = distances/speed\_of\_light

#enumerate adds a counter to each iterable and returns enumerating object

for i, t in enumerate(times):

    print(f"Signal from satellite {i+1} reaches user in {t} seconds")

print("\nPart (b)")

# Creating matrix 'A' to hold coeeficients of x, y, z as in ppt

A = np.zeros((len(satellite\_positions)-1, 3))

# Creating vector 'b' to hold contant terms of RHS as in ppt

b = np.zeros(len(satellite\_positions)-1)

for i in range(1, len(satellite\_positions)):

    A[i-1] = 2\*(satellite\_positions[i] - satellite\_positions[0])

    b[i-1] = ( (speed\_of\_light\*\*2) \* (times[0]\*\*2 - times[i]\*\*2) ) - np.sum(satellite\_positions[i]\*\*2 - satellite\_positions[0]\*\*2)

estimated\_user\_position = np.linalg.lstsq(A, b, rcond=None)[0]

# User Position without errors

print("\nEstimated User Position (without errors):", estimated\_user\_position)

print("\nPart (c)")

def addRandomError(times, error\_scale):

    return (times + np.random.normal(0, error\_scale, len(times)))

# Error = 1ns

error\_scale = 1e-9

times\_with\_error = addRandomError(times, error\_scale)

# Estimating position with error

b\_with\_error = np.zeros(len(satellite\_positions) - 1)

for i in range(1, len(satellite\_positions)):

    b\_with\_error[i-1] = ( (speed\_of\_light\*\*2) \* (times\_with\_error[0]\*\*2 - times\_with\_error[i]\*\*2) ) - np.sum(satellite\_positions[i]\*\*2 - satellite\_positions[0]\*\*2)

estimated\_user\_position\_with\_error = np.linalg.lstsq(A, b\_with\_error, rcond=None)[0]

# Calculating localization error

localization\_error = np.linalg.norm(user\_position-estimated\_user\_position\_with\_error)

print("\nEstimated User Position (with small random errors):", estimated\_user\_position\_with\_error)

print("Localization Error (meters):", localization\_error)

print("\nPart (d)")

# Error scale from 1ns to 100ns

error\_scales = np.linspace(1e-9, 1e-7, 50)

localization\_errors = []

for scale in error\_scales:

    times\_with\_error = addRandomError(times, scale)

    b\_with\_error = np.zeros(len(satellite\_positions) - 1)

    for i in range(1, len(satellite\_positions)):

        b\_with\_error[i-1] = ( (speed\_of\_light\*\*2) \* (times\_with\_error[0]\*\*2 - times\_with\_error[i]\*\*2) - np.sum(satellite\_positions[i]\*\*2 - satellite\_positions[0]\*\*2) )

    estimated\_user\_position\_with\_error = np.linalg.lstsq(A, b\_with\_error, rcond=None)[0]

    localization\_error = np.linalg.norm(user\_position - estimated\_user\_position\_with\_error)

    localization\_errors.append(localization\_error)

# Plotting the results

plt.plot(error\_scales, localization\_errors)

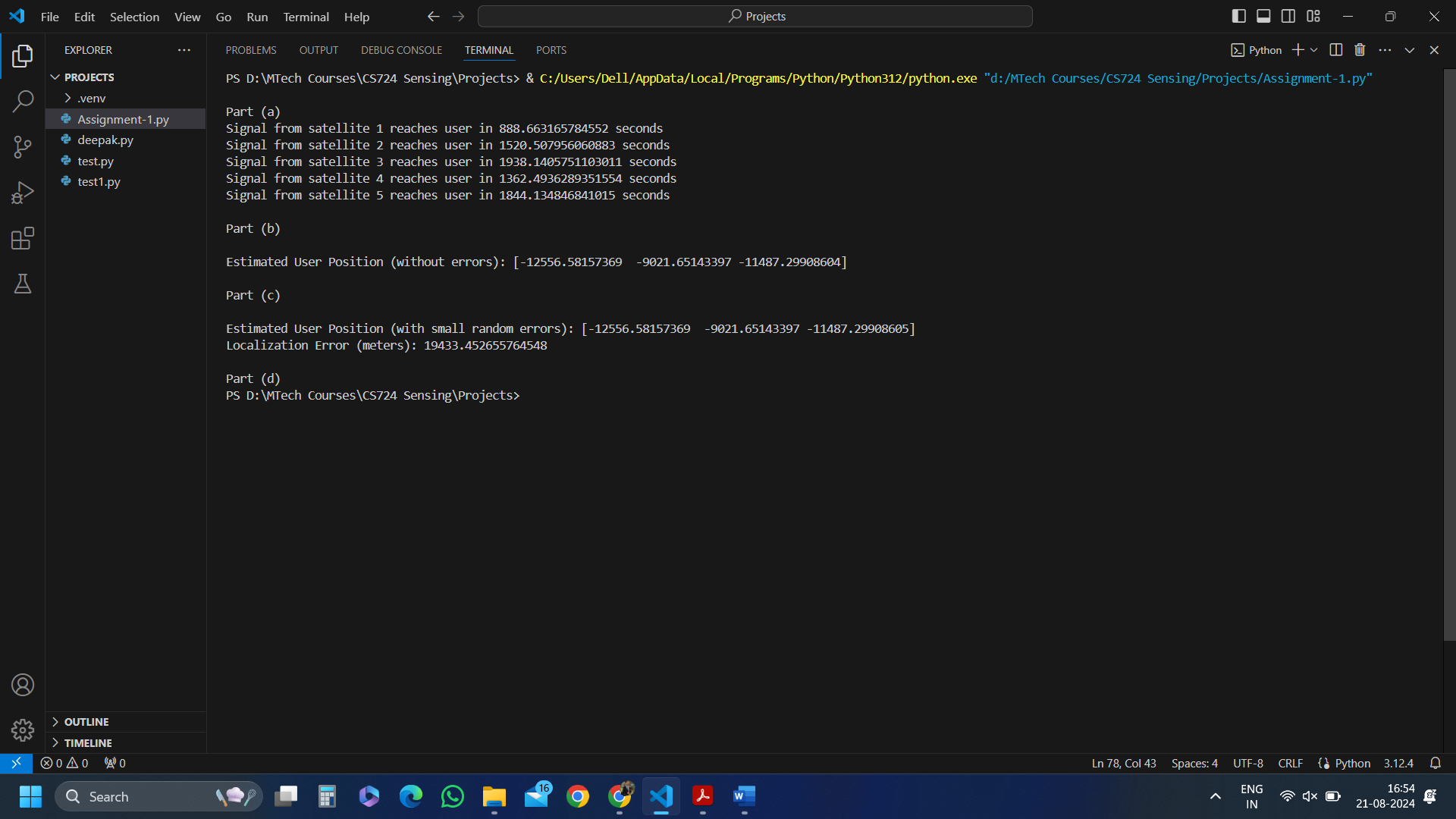
plt.xlabel('Timing Error (seconds)')

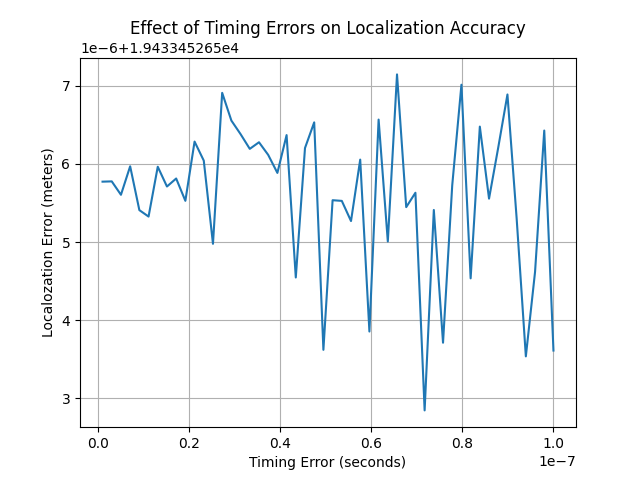
plt.ylabel('Localozation Error (meters)')

plt.title('Effect of Timing Errors on Localization Accuracy')

plt.grid(True)

plt.show()





**Explanation and Report**

**Part (a)**

* The user is fixed at (100, 100, 100).
* The 5 satellites are placed at random locations (manually fixed).
* Speed = Distance/time
* Speed is speed of light as signals travel with speed of light.
* For distance we need to calculate the distance between user and each satellite which can be done by using Euclidean Formula.
* After getting all 5 distances we can easily calculate time taken by each satellite to transmit signal to user.
* The resultant time of every satellite is shown in output terminal.

**Part (b)**

* To find the user location we can use the concept GPS Localization explained during lecture and that location should be exactly same as our fixed location of user (100, 100, 100).
* We can form linear equations to solve for the value user coordinate.
* ‘A’ – is a matrix consisting of all the coefficients of x, y, z.
* ‘b’ – is a vector containing squared difference of distances along with squared difference of satellite coordinates.
* Looping 4 times to get 4 values in A and b to form equations.
* Finally using least square method of linear algebra of numpy to solve the above formed equations.
* After using above method we get 3 values corresponding to coordinates of user.
* The result was exactly the same as originally fixed location of user.
* This means user position can be accurately determined by using time.

**Part (c)**

* Finding the user location after adding errors in time to get the idea of how localization works in real world.
* Error is scaled to 1 nanosecond because even a slightest change in time can give very vast difference in user location.
* We again created a new vector ‘b\_with\_errors’ to store the new values including errors.
* Again applying least square method to get the solution. But this time the location is not the same as initial user location instead the location is slightly changed due to errors introduced.
* Also the difference is printed on output screen in meters.
* We can easily see that even a smallest nanosecond difference in time can create a significant inaccuracies in user location.

**Part (d)**

* We have increased the errors within a range and fixing values to be only 50 just to observe the effect of errors on timing accuracy by plotting it.
* We have created an array ‘times\_with\_errors’ ehich hold time with errors introduced.
* Again vector ‘b\_with\_error’ is calculated and least square method is used.
* We have also created ‘localization\_with\_errors’ array to hold the location inaccuracies of user position.
* Using the above array and errors\_scale we are plotting the results using matplotlib.pyplot.
* The plot specifies that when the timing errors increases the localization error also increases. Showing how sensitive GPS localization system is.