Assignment-1

Question 1 (30 points)

1) $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ What is $C(A)$?

2) $B = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ What is $C(B)$?

3) $D = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 4 \end{bmatrix}$ What is $C(D)$?

C(A) = R2

C(B) = R1

C(D) = R2

Question 2 (50 points)

2) Write a program (in any convenient language, like Python/Matlab) that emulates an user localization using GPS. Use the following steps for doing this emulation. Also submit a report with relevant results and explainations.

(a) Fix the user at (100,100,100). Put 5 satellites at any random locations (you can manually put their locations), and fix their positions. Now calculate the time it takes for a signal to arrive from each one of these satellites to the user.

(b) Now lets do the opposite operation, i.e. use the satellite locations and the times to find out the location of the user. Check whether it is coming exactly as (100,100,100).

(c) Now add some random errors with the times (you can use function likes "rand" in matlab). Check how much location inaccuracy it showing up.

(d) Now increase the amount of the random errors with time, and check what is the effect of this change on the localization error. You can plot a graph on the amount of timing errors vs localization error to see the effect. You need to run the program multiple time and then can take the average localization errors).

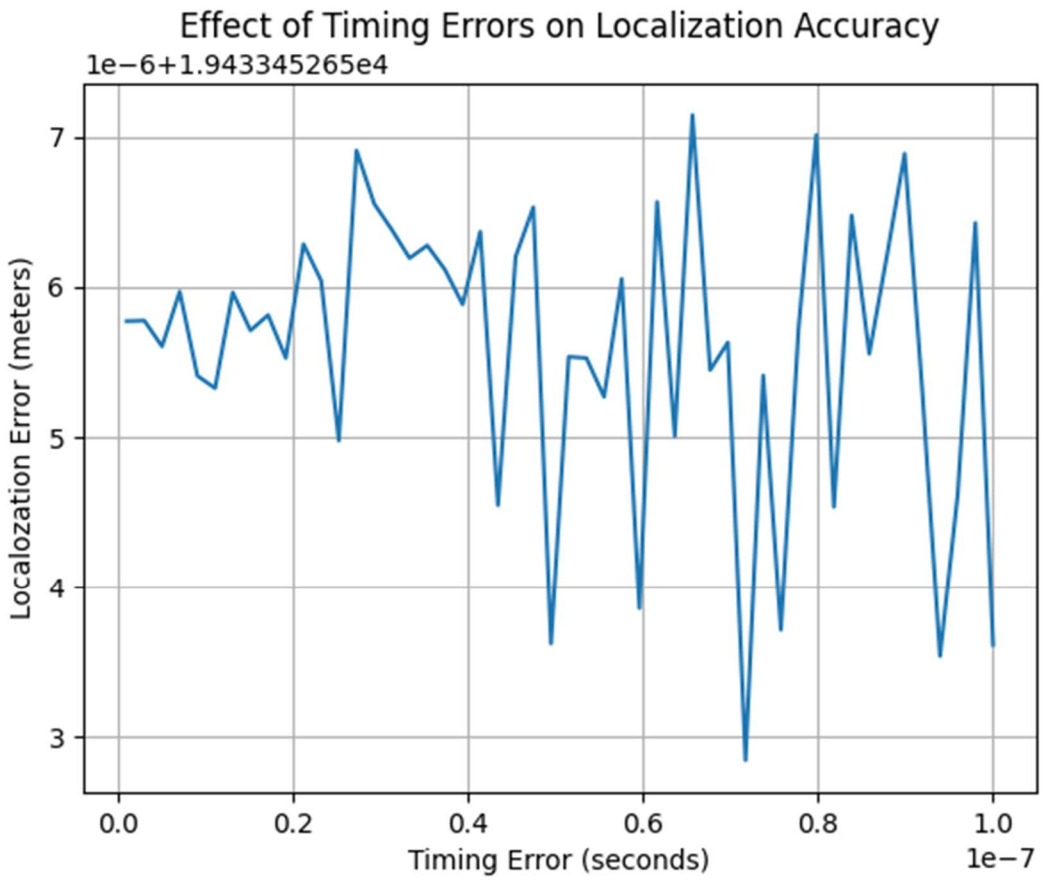
```
import numpy as np
from sympy import *
import matplotlib.pyplot as plt

# 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)
```

```python
# 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()
```

```
PS D:\MTech Courses\CS724 Sensing\Projects> & C:/Users/Dell/AppData/Local/Programs/Python/Python312/python.exe "d:/MTech Courses/CS724 Sensing/Projects/Assignment-1.py"

Part (a)
Signal from satellite 1 reaches user in 888.663165784552 seconds
Signal from satellite 2 reaches user in 1520.507956060883 seconds
Signal from satellite 3 reaches user in 1938.1405751103011 seconds
Signal from satellite 4 reaches user in 1362.4936289351554 seconds
Signal from satellite 5 reaches user in 1844.134846841015 seconds

Part (b)

Estimated User Position (without errors): [-12556.58157369  -9021.65143397 -11487.29908604]

Part (c)

Estimated User Position (with small random errors): [-12556.58157369  -9021.65143397 -11487.29908605]
Localization Error (meters): 19433.452655764548

Part (d)
PS D:\MTech Courses\CS724 Sensing\Projects>
```



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.