Homework 1
AA275: Navigation for Autonomous Systems

Alexandros Tzikas
alextzik@stanford.edu

October 22, 2021

1 Problem 1

The steps for analyzing GNSS measurements were repeated on the GNSS Analysis App and the nine generated plots are included in Fig. 1.

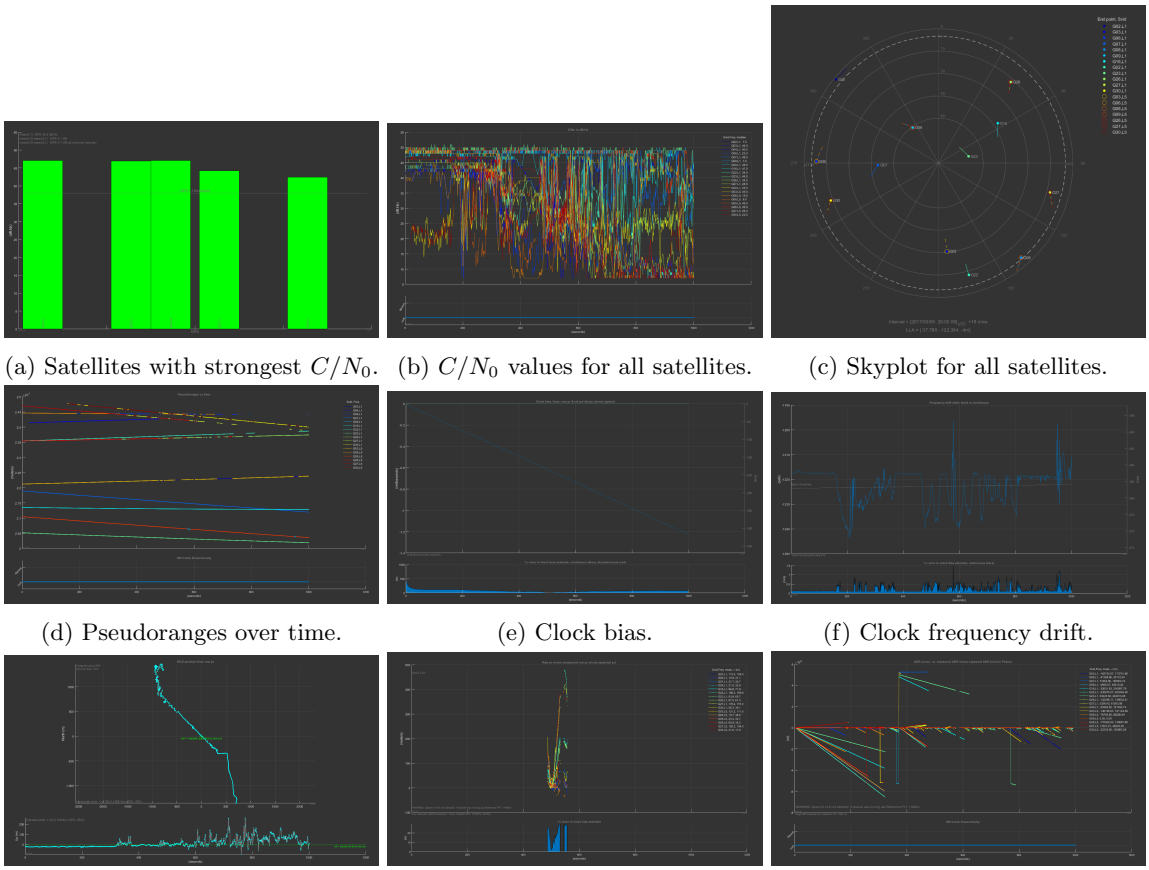
From Fig. 1a, we observe that the five (5) satellite signals with the greatest signal to noise ratio C/N_0 at the receiver are all above the GPS L1 threshold. Generally, the signal strength and thus the C/N_0 can be greatly degraded by NLOS and multi-path. We also observe that the satellites transmitting these signals are in high elevation angles (above the receiver) in Fig. 1c. This geometry enables the signals to avoid undesired influences, especially NLOS. Therefore, the signal strength will be higher. The results of this figure can vary depending on the receiver's position. Being in a different region will result in a different environment, with signal strengths varying due to factors, such as the ones mentioned above.

In Fig. 1b, we can observe how the value of C/N_0 evolves for each satellite signal. Again, we can observe that the satellite signals with the highest C/N_0 in Fig. 1a, which have higher elevation angles, are able to maintain a high C/N_0 value throughout most of the observed time period with little variation. On the other hand, satellites with lower elevation have higher variations, since they are affected more by multi-path and NLOS. The distance the signal must travel is also greater then and hence the losses are greater. We thus observe that satellites with low elevation have greater variation in their C/N_0 value. Multi-path and NLOS in these cases can lead to a greater fluctuation. Multi-path, NLOS, distance from satellite (since different elevations exist) and different frequencies lead to different losses. A lower signal power can occur as a result of these factors and it will be observed as a lower C/N_0. In addition, a challenging environment (multi-path, NLOS) can induce biases that lead to great variations in the C/N_0 value over time.

In Fig. 1c, we can observe the relative position in the sky of each visible satellite. Satellite G23 is right above the receiver. Therefore, due to LOS, it will have a high C/N_0 in Fig. 1b. The best performing satellites are above the receiver, in high elevation angles. Depending on the receiver's position, the sky plot can be very different. In addition, the visible satellites at different times are different, since the satellites are moving, orbiting the rotating Earth.

In Fig. 1d, the pseudoranges from each signal are shown. Slopes in the pseudorange measurements show that the receiver is moving. Satellite motion also influences this plot, but in a larger time scale. Satellite positions change over time, giving different pseudoranges. Pseudoranges are also affected by atmospheric effects, including tropospheric and ionospheric delays, although they have been cleaned out in the software. The time of day influences the ionospheric delay that depends on the total electron content, which in turn varies throughout the day. Satellites in different orbital planes, also measure different pseudoranges. As expected, satellites with lower elevation produce greater pseudoranges, due to their greater distance from the receiver, and, due to the lower received signal strength, have lower C/N_0 with greater variations, as seen in Fig. 1b. This is validated by satellite G30, G27 and G06 for example in the figure. Some lines have a smaller slope indicating that the pseudorange is not affected as much by the change of position of the receiver and satellite.

In Fig. 1e, we can see the estimated clock bias, which is induced by the oscillator of the receiver. In Fig. 1f, we can see the offset frequency of the receiver. This is mostly impacted by sources of error related with the receiver (on-board hardware). The variations of the plot in Fig. 1f indicate a noisy receiver.



(a) Satellites with strongest C/N_0. (b) C/N_0 values for all satellites. (c) Skyplot for all satellites.

(d) Pseudoranges over time. (e) Clock bias. (f) Clock frequency drift.

(g) WLS position solution. (h) Pseudorange errors. (i) Accumulated delta range errors.

Figure 1: GNSS Analysis App Output Figures.

In Fig. 1g, we observe the position solution using Weighted Least-Squares (WLS). The solution variation is higher in the upper region of the diagram. This must correspond to the more challenging urban environment, which exhibits NLOS and multi-path for many signals. The pseudoranges in Fig. 1d do not show great variation. Therefore, the WLS position estimate is quite stable and it does not drift. From Fig. 1g, we can also deduce that the receiver is moving. This validates our conclusion from Fig. 1d. In Fig. 1g, it is also evident that vertical error is greater when the receiver is in the challenging urban environment.

In Fig. 1h we can observe that the error (measured pseudorange - expected pseudorange) is decreasing initially and it then increases again as we enter the challenging urban environment.

Fig. 1i includes the pseudorange-rate residuals. Pseudorange-rate measurements are affected by the satellite clock drift, the receiver clock drift, the delay drift of the error due to the presence of the ionosphere, the delay drift of the error due to the presence of the troposphere and the rate of change of the error due to the presence of the multi-path, among others. We can observe that for satellite signals in higher elevation, the receiver motion creates a greater change in the pseudorange. This occurs because these satellites have a smaller distance to the receiver, and hence the receiver's motion changes the distance from the satellite more. A greater change in pseudorange results in a greater ADR. The slope of the pseudorange plot is thus indicative of the size of the ADR. The behavior is generally different for different satellites, with some having higher ADR.

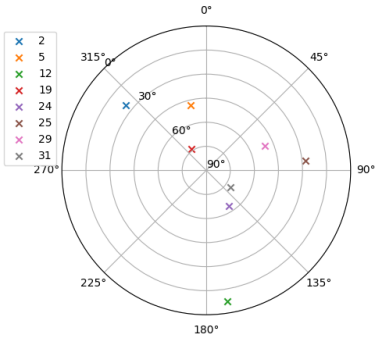


Figure 2: Skyplot for first measurement epoch.

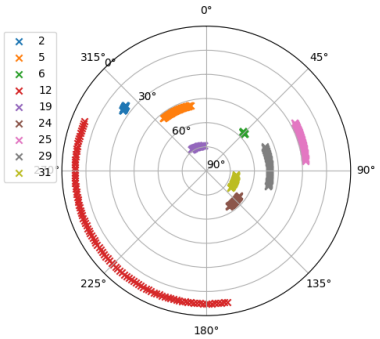


Figure 3: Skyplot for the entire dataset.

2 Problem 2

2.1

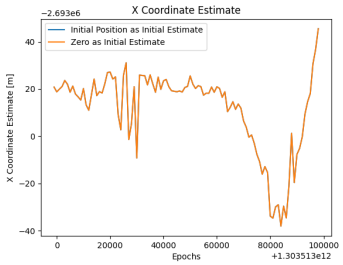
The skyplot for the first measurement epoch is shown in Fig. 2. As observed, only eight (8) satellites are visible.

2.2

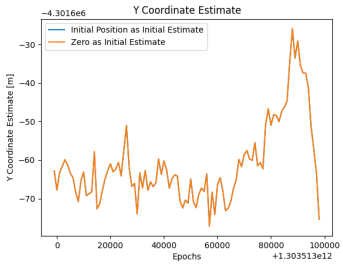
The skyplot for the entire dataset is shown in Fig. 3. One more satellite becomes visible as time progresses. It was obstructed initially. The satellites move in circular paths with respect to the receiver, due to their circular orbits. Some paths are longer than others. Satellites in orbital planes further away appear to move less, due to the increased orbital radius. However, all satellites are in the same orbital altitude. It is also true that for satellites at higher elevation, the relative geometry (relative orientation) with the receiver does not change dramatically if the receiver moves. On the other hand, the geometry orientation between the satellite and the receiver changes drastically for a satellite in a low elevation angle if the receiver moves. The same occurs due to the Earth's rotation; low elevation satelites move more due to the Earth's rotation. Hence, satellites in lower elevation angles appear to move more.

2.3

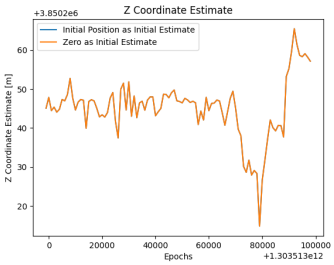
One more satellite (ID 6) becomes visible in later epochs. This means that initially the satellite must have been in an obstructed position. Observing the space around the initial position we can detect a tree, northeast from our position, which can act as an obstacle to the satellite signal. The satellite moves away from the obstacle later (moves at a higher elevation), becoming visible.



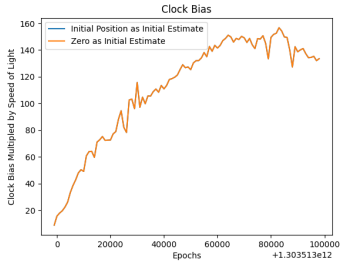
(a) X-Coordinate Estimate.



(b) Y-Coordinate Estimate.



(c) Z-Coordinate Estimate.



(d) Clock Bias Estimate.

Figure 4: Newton-Raphson position estimate using all satellites and different initializations.

2.4

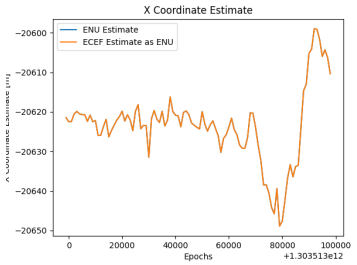
The results for this section are included in Fig. 4. It is evident from the plots that the estimate of the Newton-Raphson algorithm does not depend on the initialization. In the end, it must converge to the solution tuple that satisfies the set of equations. However, faster convergence can be achieved by using a good estimate of the solution.

2.5

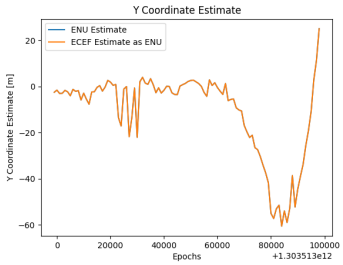
The results for this section are included in Fig. 5. It is evident from the plots that the Newton-Raphson algorithm is invariant towards the used frame of reference. In other words, the algorithm provides the same position-solution if it is applied to measurements with respect to the ECEF or ENU frame. Given that the satellite positions are transformed correctly from one frame to the other, the position solution satisfying the Newton-Raphson equations in the ECEF frame, must be the same position as the one satisfying the Newton-Raphson equations in the ENU frame. The Newton-Raphson algorithm returns the position that satisfies the measurements gathered. If it returned a different position depending on the reference frame, then the position satisfying the equations would depend on the reference frame used, which is unreasonable. In the end, the solution position must be frame independent.

2.6

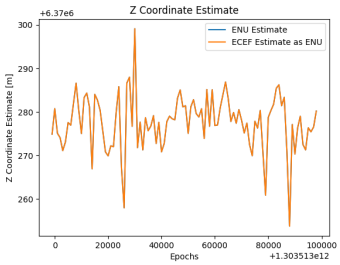
The results for this section are included in Fig. 6. The group of satellites used is varied randomly within each epoch. We observe that the solution using all the measurements is more stable and accurate, since an abundance of measurements is available to remove errors and uncertainties among them and arrive at an accurate solution. As the number of satellites used to estimate the position decreases, the information available becomes limited and errors cannot be exactly cancelled out. Therefore, we observe that as the number of used satellites decreases, the solution has greater variance and more transitions. The solution should agree with the solution with all satellites used as the number of satellites increases. This occurs in Fig. 6, where the solution with four (4) satellites exhibits the greater variance and the solution with (6) the lowest one, almost matching the solution with all satellites. The difference in accuracy between using four (4) and five (5) satellites is greater than the difference in accuracy between using five (5) and six (6) satellites.



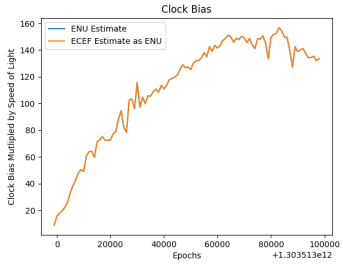
(a) X-Coordinate Estimate.



(b) Y-Coordinate Estimate.



(c) Z-Coordinate Estimate.



(d) Clock Bias Estimate.

Figure 5: Newton-Raphson position estimate using ENU coordinates and Newton-Raphson position estimate using ECEF coordinates transformed to ENU.

This happens, since five (5) satellites provide an extra level of robustness to faulty satellites, which can affect the solution of the four (4) satellites more (less resiliency in the estimation with four (4) satellites). Four (4) satellites is actually the lowest acceptable number to be able to calculate all variables in the problem. In addition, due to the limiting number of satellites (some of which may be faulty), the initialization of the algorithm does matter.

2.7

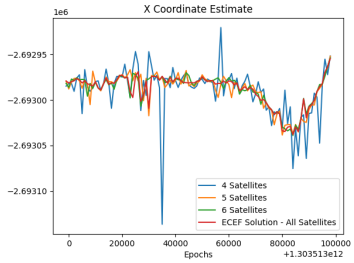
The results for this section are included in Fig. 7. The group of four (4) satellites used is varied randomly within each epoch. Therefore, we expect similar results with high variance for each of the three runs. Whenever the group of four (4) satellites used does not include a faulty one, the solution should be closer to the true one. This happens mostly when the majority of the experiments agree, since the satellites are chosen at random and the faulty satellites are less than the non-faulty. Whenever a faulty satellite is included, the solution should be further away from the correct position. If two faulty measurements are used, the solution must exhibit high variance at that time (spike).

2.8

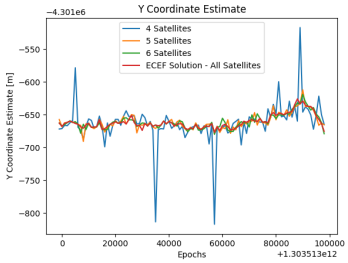
The results for this section are included in Fig. 8. The residuals for each satellite, for each epoch are plotted. The spikes correspond to the cases where the satellite is faulty, providing a pseudorange measurement quite different than the estimated position using the fusion of observations from all satellites. The residuals are quite low, indicating that the data might be cleaned before use.

2.9

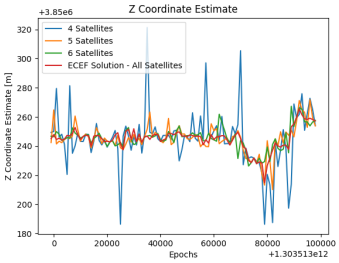
The WLS algorithm was implemented and two variations for the W matrix were used. The results are shown in Fig. 9. In the first case, each diagonal element was replaced with the corresponding satellite signal's ionospheric delay. In the second case, each diagonal element was replaced with the corresponding satellite signal's tropospheric delay. The results are similar, but as expected (given the subjective selection



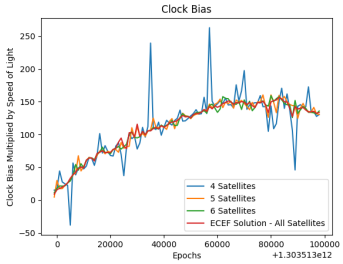
(a) X-Coordinate Estimate.



(b) Y-Coordinate Estimate.

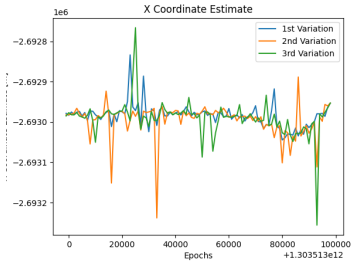


(c) Z-Coordinate Estimate.

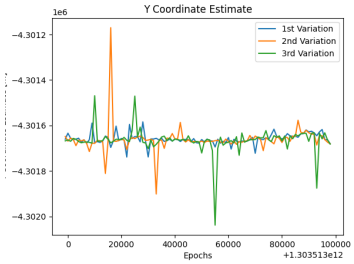


(d) Clock Bias Estimate.

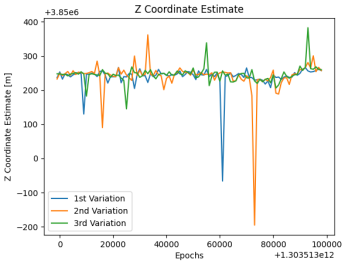
Figure 6: Newton-Raphson position estimate using ECEF coordinates and varying number of satellites.



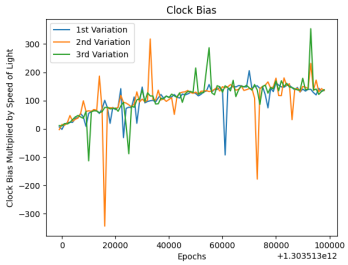
(a) X-Coordinate Estimate.



(b) Y-Coordinate Estimate.



(c) Z-Coordinate Estimate.



(d) Clock Bias Estimate.

Figure 7: Newton-Raphson position estimate using ECEF coordinates and 4 satellites for different roll-outs.

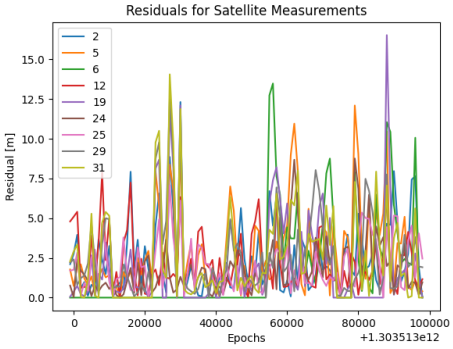
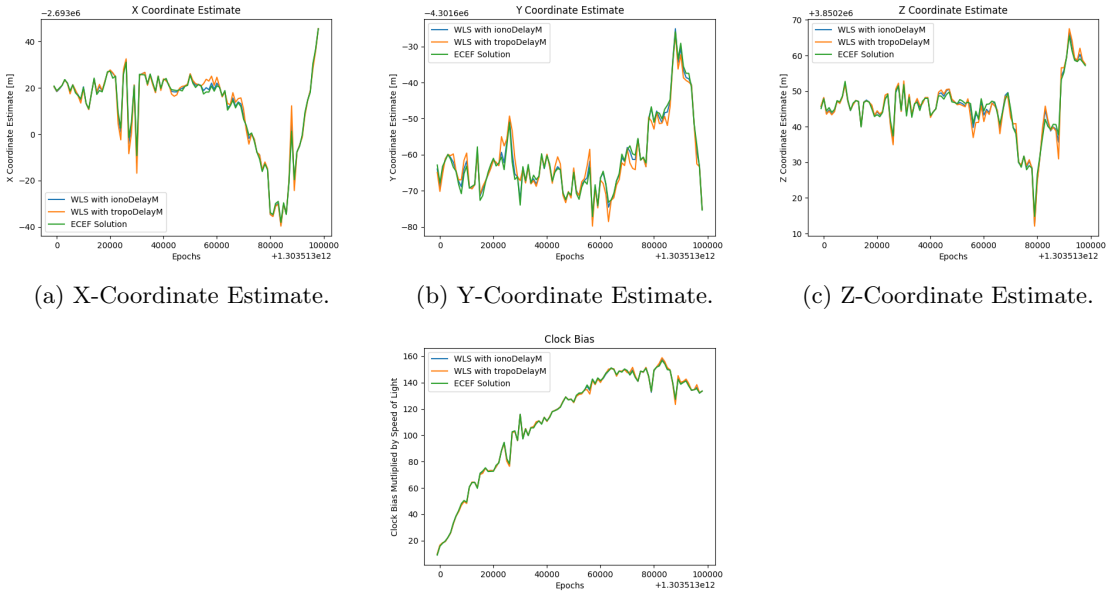


Figure 8: Residuals for all satellites.



(a) X-Coordinate Estimate.

(b) Y-Coordinate Estimate.

(c) Z-Coordinate Estimate.

(d) Clock Bias Estimate.

Figure 9: Comparison of Weighted Least-Squares and Newton-Raphson methods.

of W), slightly inferior to those of the Newton-Raphson method. This however means that WLS is robust to the choice of the matrix W.

3 Problem 3

Self-Care is critical for any human being. It is vital in order to create a sustainable lifestyle, where one works hard and strives to achieve goals.

During this week, I mostly practiced self-care by devoting time to go for coffee with some of my friends during the afternoons. We enjoyed a nice cup of coffee talking, relaxing and getting to know each other better. In addition, I played beach volley on the weekend and watched a movie on Sunday night.

4 Code Implementation

The code can also be found at: https://github.com/alextzik/navigation_autonomous_systems/blob/main/GNSS%20Data%20Analysis/script.py.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
import math
import pandas as pd
from mpl_toolkits import mplot3d


# Read csv data into pandas Dataframe
def read_data(file):
  data = pd.read_csv(file)
  return data

# Convert ECEF (m) coordinates to ENU (m) about reference latitude (N) and longitude (E)
def ecef2enu(x,y,z, lat_ref, lon_ref):
  lat_ref = np.deg2rad(lat_ref)
  lon_ref = np.deg2rad(lon_ref + 360)
  C = np.zeros((3,3))
  C[0,0]=-np.sin(lat_ref)*np.cos(lon_ref)
  C[0,1]=-np.sin(lat_ref)*np.sin(lon_ref)
  C[0,2]= np.cos(lat_ref)

  C[1,0]=-np.sin(lon_ref)
  C[1,1]= np.cos(lon_ref)
  C[1,2]= 0

  C[2,0]=np.cos(lat_ref)*np.cos(lon_ref)
  C[2,1]=np.cos(lat_ref)*np.sin(lon_ref)
  C[2,2]=np.sin(lat_ref)

  x, y, z = np.dot(C,np.array([x, y, z]))

  return x, y, z


# Convert zero-centered Cartesian coordinates to Elevation and Azimuth
def cart2elaz(x,y,z):
    R = x**2 + y**2
    elev = np.rad2deg(math.atan2(z,math.sqrt(R)))
    az = np.rad2deg(math.atan2(y,x))
    return elev, az

# Create an empty skyplot with a closure to add satellites
def create_skyplot():
  ax = plt.gca(projection='polar')
  ax.set_theta_zero_location('N')
  ax.set_theta_direction(-1)
  ax.set_rlim(1, 91)
  ax.grid(True, which='major')
  degree_sign = u'\N{DEGREE SIGN}'
```

```
  r_labels = [
      '90' + degree_sign,
      '',
      '60' + degree_sign,
      '',
      '30' + degree_sign,
      '',
      '0' + degree_sign,
  ]
  ax.set_rgrids(range(1, 106, 15), r_labels, angle=-45)

# Closure to add satellites to the skyplot from list of Elevation and Azimuth
  def add_satellite(trajectory, label=None):
    trajectory = np.array(trajectory)
    ax.scatter(trajectory[:, 1], trajectory[:, 0], marker='x', label=label)

  return add_satellite, ax

# Utility function to compute expected pseudorange from (x, y, z, cdt)
# to satellite (X, Y, Z, B)
def expected_prange(x, y, z, cdt, X, Y, Z, B):
  return np.sqrt((x - X)**2 + (y - Y)**2 + (z - Z)**2) + cdt - B

# Pseudoinverse of matrix A using diagonal weight matrix W
def weighted_pinv(A, W=None):
    if W is None:
        W = np.diag(np.ones(len(A)))
    _A = np.linalg.inv(A.T @ W @ A)
    return _A @ A.T @ W

######################################################################
##                                  Main Code                       ##
data = read_data('hw1_data.csv')


######################################################################
# 2.1
def plot_sats(data):
  # Find all satellites. Different satellites in each time epoch.
  # Some never visible due to NLOS
  satIDs = np.unique((data['svid']).to_numpy())
  nOfSats = len(satIDs)
  add_satellite_All, skyplotAll = create_skyplot() # New plot
  for i in range(0, nOfSats):
    # iterate for every satellite: find the satellite's trajectory
    satData = data.loc[data['svid']==satIDs[i]]
    satTraj = satData[['xSatPosM', 'ySatPosM', 'zSatPosM']].to_numpy()
    for k in range(0, satTraj.shape[0]):
      x, y, z = ecef2enu(satTraj[k,0], satTraj[k,1], satTraj[k,2], 37.3715, -122.047861)
      satTraj[k, 0] = x
      satTraj[k, 1] = y
      satTraj[k, 2] = z

    satTrajAlEl = np.zeros((satTraj.shape[0], 2))
```

```
    for j in range(0, satTraj.shape[0]):
      satTrajAlEl[j,:] = cart2elaz(satTraj[j, 0], satTraj[j, 1], satTraj[j,2])

    l = "{}".format(satIDs[i])
    add_satellite_All(satTrajAlEl, satIDs[i])
  plt.legend(bbox_to_anchor=(0, 1), loc='upper right', ncol=1)
  plt.show()

epochs = np.sort(np.unique(data['millisSinceGpsEpoch'].to_numpy())) # Find all epochs
firstEpoch = np.min(epochs) # Find first epoch
# Get the data of the first epoch
firstEpochData = data.loc[data['millisSinceGpsEpoch']==firstEpoch]

plot_sats(firstEpochData)

##############################################################################
# 2.2
plot_sats(data)

##############################################################################
# 2.3
# No code needed. Compare the 2 skyplots

##############################################################################
# 2.4
def newtonRaphson_ECEF(data, initialEstimate, kmax, epoch, nSats=0):
  newEstimate = np.zeros((4,1))
  newEstimate[0,0] = initialEstimate[0,0]
  newEstimate[1,0] = initialEstimate[1,0]
  newEstimate[2,0] = initialEstimate[2,0]
  newEstimate[3,0] = initialEstimate[3,0]

  epochData = data.loc[data['millisSinceGpsEpoch']==epoch]
  s = np.unique((epochData['svid']).to_numpy())

  if nSats !=0:
    s = np.random.choice(s, nSats, replace=False)

  for k in range(0, kmax):
    F = np.zeros((len(s),1))
    A = np.zeros((len(s),4))
    j = 0
    for i in s:
      satData = epochData.loc[data['svid']==i]

      r_i = satData['PrM']
      b_i = satData['satClkBiasM']
      x_sat = satData['xSatPosM']
      y_sat = satData['ySatPosM']
      z_sat = satData['zSatPosM']

      F[j,0] = r_i - (math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2)) +
      newEstimate[3] - b_i)
```

```
      A[j,0] = - (newEstimate[0]-x_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,1] = - (newEstimate[1]-y_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,2] = - (newEstimate[2]-z_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,3] = -1

      j = j+1

    pseudoInv_A = weighted_pinv(A)
    newEstimate = newEstimate + np.matmul(pseudoInv_A,-F)
  return newEstimate

def newtonRaphson_ECEF_AllEpochs(data, epochs, initialEstimate, kmax, nSats=0):
  estimatesAllEpochs_ECEF = np.zeros((4,len(epochs)))
  k = 0
  for i in epochs:
      result = newtonRaphson_ECEF(data, initialEstimate, kmax, i, nSats)
      estimatesAllEpochs_ECEF[0,k] = result[0,0]
      estimatesAllEpochs_ECEF[1,k] = result[1,0]
      estimatesAllEpochs_ECEF[2,k] = result[2,0]
      estimatesAllEpochs_ECEF[3,k] = result[3,0]
      k = k+1
  return estimatesAllEpochs_ECEF

# Calculate Newton Raphson Estimate in ECEF, with All Satellites
# and Initial Position as Initial Estimate
print("ECEF Estimate with All Satellites - Initial Position as Initial Estimate")
estimatesAllEpochs_InitialPosEst_ECEF = newtonRaphson_ECEF_AllEpochs(data, epochs,
np.array([[-2692974], [-4301659], [3850240],[0]]), 5, 0)
print(estimatesAllEpochs_InitialPosEst_ECEF[:,0])

# Calculate Newton Raphson Estimate in ECEF, with All Satellites
# and Initial Position as Initial Estimate
print("ECEF Estimate with All Satellites - Zero as Initial Estimate")
estimatesAllEpochs_InitialZeroEst_ECEF = newtonRaphson_ECEF_AllEpochs(data, epochs,
np.array([[0], [0], [0],[0]]), 5, 0)
print(estimatesAllEpochs_InitialZeroEst_ECEF[:,0])

# Plots
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[0,:],
label = 'Initial Position as Initial Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialZeroEst_ECEF[0,:],
label = 'Zero as Initial Estimate')
plt.xlabel('Epochs')
plt.ylabel('X Coordinate Estimate [m]')
plt.title('X Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[1,:],
label = 'Initial Position as Initial Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialZeroEst_ECEF[1,:],
```

```
label = 'Zero as Initial Estimate')
plt.xlabel('Epochs')
plt.ylabel('Y Coordinate Estimate [m]')
plt.title('Y Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[2,:],
label = 'Initial Position as Initial Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialZeroEst_ECEF[2,:],
label = 'Zero as Initial Estimate')
plt.xlabel('Epochs')
plt.ylabel('Z Coordinate Estimate [m]')
plt.title('Z Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[3,:],
label = 'Initial Position as Initial Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialZeroEst_ECEF[3,:],
label = 'Zero as Initial Estimate')
plt.xlabel('Epochs')
plt.ylabel('Clock Bias Multipled by Speed of Light')
plt.title('Clock Bias')
plt.legend()
plt.show()

#########################################################################
# 2.5
def newtonRaphson_ENU(data, initialEstimate, kmax, epoch):
  newEstimate = np.zeros((4,1))
  newEstimate[0,0] = initialEstimate[0,0]
  newEstimate[1,0] = initialEstimate[1,0]
  newEstimate[2,0] = initialEstimate[2,0]
  newEstimate[3,0] = initialEstimate[3,0]

  epochData = data.loc[data['millisSinceGpsEpoch']==epoch]
  s = np.unique((epochData['svid']).to_numpy())

  for k in range(0, kmax):
    F = np.zeros((len(s),1))
    A = np.zeros((len(s),4))
    j = 0
    for i in s:
      satData = epochData.loc[data['svid']==i]

      r_i = satData['PrM']
      b_i = satData['satClkBiasM']
      x_sat, y_sat, z_sat = ecef2enu(satData['xSatPosM'],satData['ySatPosM'],satData['zSatPosM'],
      37.3715, -122.047861)

      F[j,0] = r_i - (math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2)) + newEstimate[3] -
      b_i)
```

```
      A[j,0] = - (newEstimate[0]-x_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,1] = - (newEstimate[1]-y_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,2] = - (newEstimate[2]-z_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,3] = -1

      j = j+1

    pseudoInv_A = weighted_pinv(A)
    newEstimate = newEstimate + np.matmul(pseudoInv_A,-F)
  return newEstimate

def newtonRaphson_ENU_AllEpochs(data, epochs, initialEstimate, kmax, nSats=0):
  estimatesAllEpochs_ENU = np.zeros((4,len(epochs)))
  k = 0
  for i in epochs:
    result = newtonRaphson_ENU(data, initialEstimate, kmax, i)
    estimatesAllEpochs_ENU[0,k] = result[0,0]
    estimatesAllEpochs_ENU[1,k] = result[1,0]
    estimatesAllEpochs_ENU[2,k] = result[2,0]
    estimatesAllEpochs_ENU[3,k] = result[3,0]
    k = k+1

  return estimatesAllEpochs_ENU

# Determine ENU Solution from Newton-Raphson
print("ENU Estimate with All Satellites")
estimatesAllEpochs_ENU = newtonRaphson_ENU_AllEpochs(data, epochs,
np.array([[-20621.87590879528], [-7.501155948266387], [6370267.020520784], [0.0]]), 5, 0)
print(estimatesAllEpochs_ENU[:,0])
print("ECEF Estimate with All Satellites Transformed to ENU")
print(ecef2enu(estimatesAllEpochs_InitialPosEst_ECEF[0,0],
estimatesAllEpochs_InitialPosEst_ECEF[1,0], estimatesAllEpochs_InitialPosEst_ECEF[2,0], 37.3715,
-122.047861))

# Convert ECEF Solution with All Satellites to ENU
estimatesAllEpochs_InitialPosEst_ECEFtoENU = np.zeros((4, len(epochs)))
k = 0
for i in epochs:
  estimatesAllEpochs_InitialPosEst_ECEFtoENU[0,k], estimatesAllEpochs_InitialPosEst_ECEFtoENU[1,k],
  estimatesAllEpochs_InitialPosEst_ECEFtoENU[2,k] =
  ecef2enu(estimatesAllEpochs_InitialPosEst_ECEF[0,k],
  estimatesAllEpochs_InitialPosEst_ECEF[1,k], estimatesAllEpochs_InitialPosEst_ECEF[2,k],
  37.3715, -122.047861)
  estimatesAllEpochs_InitialPosEst_ECEFtoENU[3,k] = estimatesAllEpochs_InitialPosEst_ECEF[3, k]
  k = k+1

# Plots
plt.plot(epochs, estimatesAllEpochs_ENU[0,:], label = 'ENU Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEFtoENU[0,:], label = 'ECEF Estimate as ENU')
plt.xlabel('Epochs')
plt.ylabel('X Coordinate Estimate [m]')
```

```
plt.title('X Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_ENU[1,:], label = 'ENU Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEFtoENU[1,:], label = 'ECEF Estimate as ENU')
plt.xlabel('Epochs')
plt.ylabel('Y Coordinate Estimate [m]')
plt.title('Y Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_ENU[2,:], label = 'ENU Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEFtoENU[2,:], label = 'ECEF Estimate as ENU')
plt.xlabel('Epochs')
plt.ylabel('Z Coordinate Estimate [m]')
plt.title('Z Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_ENU[3,:], label = 'ENU Estimate')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEFtoENU[3,:], label = 'ECEF Estimate as ENU')
plt.xlabel('Epochs')
plt.ylabel('Clock Bias Mutlipled by Speed of Light')
plt.title('Clock Bias')
plt.legend()
plt.show()

########################################################################
# 2.6

# Calculate Newton Raphson estimate using limited number of satellites.
# The choice of satellites is independent in each epoch.
estimate_ECEF_4 = newtonRaphson_ECEF_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 4)
estimate_ECEF_5 = newtonRaphson_ECEF_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 5)
estimate_ECEF_6 = newtonRaphson_ECEF_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 6)
print("ECEF Estimate with 4 Satellites")
print(estimate_ECEF_4[:,0])
print("ECEF Estimate with 5 Satellites")
print(estimate_ECEF_5[:,0])
print("ECEF Estimate with 6 Satellites")
print(estimate_ECEF_6[:,0])
  # As number of sats increases, the estimate closer to our expected position

plt.plot(epochs, estimate_ECEF_4[0,:], label = '4 Satellites')
plt.plot(epochs, estimate_ECEF_5[0,:], label = '5 Satellites')
plt.plot(epochs, estimate_ECEF_6[0,:], label = '6 Satellites')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[0,:], label = 'ECEF Solution - All Satellites')
plt.xlabel('Epochs')
plt.ylabel('X Coordinate Estimate [m]')
plt.title('X Coordinate Estimate')
```

```
plt.legend()
plt.show()

plt.plot(epochs, estimate_ECEF_4[1,:], label = '4 Satellites')
plt.plot(epochs, estimate_ECEF_5[1,:], label = '5 Satellites')
plt.plot(epochs, estimate_ECEF_6[1,:], label = '6 Satellites')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[1,:], label = 'ECEF Solution - All Satellites')
plt.xlabel('Epochs')
plt.ylabel('Y Coordinate Estimate [m]')
plt.title('Y Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimate_ECEF_4[2,:], label = '4 Satellites')
plt.plot(epochs, estimate_ECEF_5[2,:], label = '5 Satellites')
plt.plot(epochs, estimate_ECEF_6[2,:], label = '6 Satellites')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[2,:], label = 'ECEF Solution - All Satellites')
plt.xlabel('Epochs')
plt.ylabel('Z Coordinate Estimate [m]')
plt.title('Z Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimate_ECEF_4[3,:], label = '4 Satellites')
plt.plot(epochs, estimate_ECEF_5[3,:], label = '5 Satellites')
plt.plot(epochs, estimate_ECEF_6[3,:], label = '6 Satellites')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[3,:], label = 'ECEF Solution - All Satellites')
plt.xlabel('Epochs')
plt.ylabel('Clock Bias Multiplied by Speed of Light')
plt.title('Clock Bias')
plt.legend()
plt.show()

########################################################################
# 2.7

# Calculate Newton Raphson for different groups of 4 satellites
estimate_ECEF_4_1 = newtonRaphson_ECEF_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 4)
estimate_ECEF_4_2 = newtonRaphson_ECEF_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 4)
estimate_ECEF_4_3 = newtonRaphson_ECEF_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 4)

plt.plot(epochs, estimate_ECEF_4_1[0,:], label = '1st Variation')
plt.plot(epochs, estimate_ECEF_4_2[0,:], label = '2nd Variation')
plt.plot(epochs, estimate_ECEF_4_3[0,:], label = '3rd Variation')
plt.xlabel('Epochs')
plt.ylabel('X Coordinate Estimate [m]')
plt.title('X Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimate_ECEF_4_1[1,:], label = '1st Variation')
```

```
plt.plot(epochs, estimate_ECEF_4_2[1,:], label = '2nd Variation')
plt.plot(epochs, estimate_ECEF_4_3[1,:], label = '3rd Variation')
plt.xlabel('Epochs')
plt.ylabel('Y Coordinate Estimate [m]')
plt.title('Y Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimate_ECEF_4_1[2,:], label = '1st Variation')
plt.plot(epochs, estimate_ECEF_4_2[2,:], label = '2nd Variation')
plt.plot(epochs, estimate_ECEF_4_3[2,:], label = '3rd Variation')
plt.xlabel('Epochs')
plt.ylabel('Z Coordinate Estimate [m]')
plt.title('Z Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimate_ECEF_4_1[3,:], label = '1st Variation')
plt.plot(epochs, estimate_ECEF_4_2[3,:], label = '2nd Variation')
plt.plot(epochs, estimate_ECEF_4_3[3,:], label = '3rd Variation')
plt.xlabel('Epochs')
plt.ylabel('Clock Bias Multiplied by Speed of Light')
plt.title('Clock Bias')
plt.legend()
plt.show()

#########################################################################
# 2.8

residuals = np.zeros((len(np.unique(data['svid'].to_numpy())), len(epochs)))
sats = np.unique(data.svid.to_numpy())
dict = {}

t = 0
for j in sats:
  dict[j] = t
  t = t+1

t = 0
for i in epochs:
  epochData = data.loc[data['millisSinceGpsEpoch']==i]
  s = np.unique((epochData['svid']).to_numpy())
  for j in s:
    satData = epochData.loc[data['svid']==j]

    r_i = satData['PrM']
    b_i = satData['satClkBiasM']
    x_sat = satData['xSatPosM']
    y_sat = satData['ySatPosM']
    z_sat = satData['zSatPosM']

    res = abs(r_i - (math.sqrt(math.pow((estimatesAllEpochs_InitialPosEst_ECEF[0,t]-x_sat),2) +
    math.pow((estimatesAllEpochs_InitialPosEst_ECEF[1,t]-y_sat),2) +
    math.pow((estimatesAllEpochs_InitialPosEst_ECEF[2,t]-z_sat),2)) +
```

```
    estimatesAllEpochs_InitialPosEst_ECEF[3,t] - b_i))
    residuals[dict[j], t] = res

  t = t+1

# Plot Residuals
for i in sats:
  plt.plot(epochs, residuals[dict[i],:], label = i)

plt.xlabel('Epochs')
plt.ylabel('Residual [m]')
plt.title('Residuals for Satellite Measurements')
plt.legend()
plt.show()

#############################################################################
# 2.9
def wls(data, initialEstimate, kmax, epoch, nSats=0, choice=0):
  newEstimate = np.zeros((4,1))
  newEstimate[0,0] = initialEstimate[0,0]
  newEstimate[1,0] = initialEstimate[1,0]
  newEstimate[2,0] = initialEstimate[2,0]
  newEstimate[3,0] = initialEstimate[3,0]

  epochData = data.loc[data['millisSinceGpsEpoch']==epoch]
  s = np.unique((epochData['svid']).to_numpy())

  if nSats !=0:
    s = np.random.choice(s, nSats, replace=False)

  for k in range(0, kmax):
    F = np.zeros((len(s),1))
    A = np.zeros((len(s),4))
    W = np.zeros((len(s),len(s)))
    j = 0
    for i in s:
      satData = epochData.loc[data['svid']==i]

      r_i = satData['PrM']
      b_i = satData['satClkBiasM']
      x_sat = satData['xSatPosM']
      y_sat = satData['ySatPosM']
      z_sat = satData['zSatPosM']
      ionoDelay = satData['ionoDelayM']
      tropoDelay = satData['tropoDelayM']

      F[j,0] = r_i - (math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2)) + newEstimate[3] -
      b_i)
      A[j,0] = - (newEstimate[0]-x_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,1] = - (newEstimate[1]-y_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,2] = - (newEstimate[2]-z_sat)/math.sqrt(math.pow((newEstimate[0]-x_sat),2) +
```

```
      math.pow((newEstimate[1]-y_sat),2) + math.pow((newEstimate[2]-z_sat),2))
      A[j,3] = -1

      if choice == 0:
        W[j,j] = ionoDelay
      else:
        W[j,j] = tropoDelay
      j = j+1

    K1 = np.matmul(np.transpose(A), W)
    K1 = np.matmul(K1, A)
    K1 = np.linalg.inv(K1)
    K2 = np.matmul(np.transpose(A), W)
    K2 = np.matmul(K2, -F)
    newEstimate = newEstimate + np.matmul(K1, K2)
  return newEstimate

def wls_AllEpochs(data, epochs, initialEstimate, kmax, nSats=0, choice=0):
  estimatesAllEpochs_wls = np.zeros((4,len(epochs)))
  k = 0
  for i in epochs:
      result = wls(data, initialEstimate, kmax, i, nSats, choice)
      estimatesAllEpochs_wls[0,k] = result[0,0]
      estimatesAllEpochs_wls[1,k] = result[1,0]
      estimatesAllEpochs_wls[2,k] = result[2,0]
      estimatesAllEpochs_wls[3,k] = result[3,0]
      k = k+1
  return estimatesAllEpochs_wls

# WLS Solution using IonoDelay
estimatesAllEpochs_wls_0 = wls_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 0, 0)

# WLS Solution using TropoDelay
estimatesAllEpochs_wls_1 = wls_AllEpochs(data, epochs, np.array([[-2692974], [-4301659],
[3850240],[0]]), 5, 0, 1)

plt.plot(epochs, estimatesAllEpochs_wls_0[0,:], label = 'WLS with ionoDelayM')
plt.plot(epochs, estimatesAllEpochs_wls_1[0,:], label = 'WLS with tropoDelayM')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[0,:], label = 'ECEF Solution')
plt.xlabel('Epochs')
plt.ylabel('X Coordinate Estimate [m]')
plt.title('X Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_wls_0[1,:], label = 'WLS with ionoDelayM')
plt.plot(epochs, estimatesAllEpochs_wls_1[1,:], label = 'WLS with tropoDelayM')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[1,:], label = 'ECEF Solution')
plt.xlabel('Epochs')
plt.ylabel('Y Coordinate Estimate [m]')
plt.title('Y Coordinate Estimate')
plt.legend()
plt.show()
```

```
plt.plot(epochs, estimatesAllEpochs_wls_0[2,:], label = 'WLS with ionoDelayM')
plt.plot(epochs, estimatesAllEpochs_wls_1[2,:], label = 'WLS with tropoDelayM')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[2,:], label = 'ECEF Solution')
plt.xlabel('Epochs')
plt.ylabel('Z Coordinate Estimate [m]')
plt.title('Z Coordinate Estimate')
plt.legend()
plt.show()

plt.plot(epochs, estimatesAllEpochs_wls_0[3,:], label = 'WLS with ionoDelayM')
plt.plot(epochs, estimatesAllEpochs_wls_1[3,:], label = 'WLS with tropoDelayM')
plt.plot(epochs, estimatesAllEpochs_InitialPosEst_ECEF[3,:], label = 'ECEF Solution')
plt.xlabel('Epochs')
plt.ylabel('Clock Bias Mutliplied by Speed of Light')
plt.title('Clock Bias')
plt.legend()
plt.show()
```