ENEL445: Engineering Optimization

Source Geo-Location Progress Report

Submitted by:

Daegan Rose-Love (33480267)

Ben Mangin (92699294)

Date: 26/04/2024

Table of Contents

1. Background

1.1. Brief

Radio source geo-location involves locating the coordinates of where a particular signal is broadcast from by using arbitrary measurements. This project uses four satellites to locate a ground-based signal using the frequency-difference of arrival (FDOA) measurements. Source geo-location has a range of applications, from military to civil, including surveillance, navigation, and search and rescue.

1.2. Math Background

When calculating the doppler shift of the signal the coordinates need to be in the Earth Centred Earth Fixed (ECEF) coordinate system as shown in vector (1) as x, y, and z. The position and velocity of the satellites are already in the ECEF system which is shown in vector (1.2) and (1.3) but the grid points in the search area (grid points) are geodetic positions as shown in vector (1.1) where B is latitude, L is longitude, and H is altitude.

$$\vec{u}^0 = [x^0 \quad y^0 \quad z^0]^T \tag{1}$$

$$\vec{p}^0 = [B^0 \quad L^0 \quad H^0]^T \tag{1.1}$$

$$\vec{s}_i = [s_{x,i} \quad s_{y,i} \quad s_{z,i}]^T \tag{1.2}$$

$$\vec{\dot{s}}_i = [\dot{s}_{x,i} \quad \dot{s}_{y,i} \quad \dot{s}_{z,i}]^T \tag{1.3}$$

To convert the coordinates the geodetic coordinates, need to be converted into radians. Once the coordinates were converted to radians the equatorial radius is found as shown in equation (2).

$$R_E(B^0) = \frac{R_0}{\sqrt{(1 - e^2 \sin^2(B^0)}} \tag{2}$$

This uses the Earths equatorial radius $R_0 = 6378.137\ km$, the earth eccentricity $e = 0.081819198425$. The resultant can then be used in the following equations which will give the converted x, y, and z coordinates.

$$x^0 = (R_E(B^0) + H)\cos(B^0)\cos(L^0) \tag{2.1}$$

$$y^0 = (R_E(B^0) + H)\cos(B^0)\sin(L^0) \tag{2.2}$$

$$z^0 = [(1 + e^2)R_E(B^0) + H]\sin(B^0) \tag{2.3}$$

The equation shown below in (3) calculates the frequency of the signal received by the satellite. This is not the same frequency as the transmitted frequency. The transmitted signal will be subjected to doppler shift and channel noise.

$$f_i = f_0 + \frac{1}{\lambda}\frac{(\vec{u}^0 - \vec{s}_i)\dot{\vec{s}}_i}{\|\vec{u}^0 - \vec{s}_i\|} + n_i \tag{3}$$

The equation uses the frequency of the source f_o the position vector of the point that is being measured $\vec{u}^0$, the position vector of the satellite $\vec{s}_i$, the velocity vector of the satellite $\dot{\vec{s}}_i$, and additive noise, n_i. As the frequency of the source is unknown this equation can't be used. To get around this the frequency of the received signal can be compared to the frequency of the first satellite as shown in the following equation (3.1). This will result in a vector of frequencies as shown below in vector (3.2).

$$\vec{f} = f_j - f_1 = \frac{1}{\lambda}\frac{(\vec{u}^0 - \vec{s}_j)\dot{\vec{s}}_j}{\|\vec{u}^0 - \vec{s}_j\|} - \frac{1}{\lambda}\frac{(\vec{u}^0 - \vec{s}_1)\dot{\vec{s}}_1}{\|\vec{u}^0 - \vec{s}_1\|} + (n_j - n_1)\,, for\ j = 2,3,4 \tag{3.1}$$

$$\rightarrow \vec{f} = [f_{21} \quad f_{31} \quad f_{41}]^T \tag{3.2}$$

The following vector (3.3) is the noise vector used to add noise to each frequency vector at each coordinate. Each n value is a random value chosen withing a gaussian function.

$$\rightarrow \vec{n} = \begin{bmatrix} n_2 - n_1 \\ n_3 - n_1 \\ n_4 - n_1 \end{bmatrix} \tag{3.3}$$

When creating a contour plot of the objective function the frequency at each grid point needs to be evaluated. This will be done as shown in the vector below (4). Where each g value is the doppler shift a satellite receives minus the doppler shift of satellite one.

$$\vec{g}(B^0, L^0) = \begin{bmatrix} g(B^0, L^0, s_2, \dot{s}_2) - g(B^0, L^0, s_1, \dot{s}_1) \\ g(B^0, L^0, s_3, \dot{s}_3) - g(B^0, L^0, s_1, \dot{s}_1) \\ g(B^0, L^0, s_4, \dot{s}_4) - g(B^0, L^0, s_1, \dot{s}_1) \end{bmatrix} \tag{4}$$

The following equation (5) allows us to find the Q matrix for our weighting function used to find theta.

$$Cov(x_i, x_j) = E[(x_i - \mu_i)(x_j - \mu_j)] = E[\Delta\vec{f}\Delta\vec{f}^T] \tag{5}$$

The function for a multivariate gaussian probability function (PDF) is shown below.

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{\sqrt{2\pi\Sigma}} \exp\left(-\frac{(x-\mu)^T\Sigma^{-1}(x-\mu)}{2}\right) \tag{6}$$

X is the known measurement vector, μ is the mean vector, and the Σ is the covariance matrix composed of $\sigma^2 Q$.

1.3 Assumptions

The assumption that were made for this simulation are as follows:

- Altitude is zero and does not change.
- Measurements do not account for special relativity.
- The source frequency is known.
- The location of the source is known for the grid-search and Monte-Carlo simulation.

2. FDOA-Only Objective Function Formulation

For convenience, set the noise vector found in (3.3) to be:

$$\Delta\vec{f} = \begin{bmatrix} n_2 - n_1 \\ n_3 - n_1 \\ n_4 - n_1 \end{bmatrix} \tag{7}$$

Then by using (5) we get:

$$E\left[\begin{bmatrix} n_2 - n_1 \\ n_3 - n_1 \\ n_4 - n_1 \end{bmatrix}\begin{bmatrix} n_2 - n_1 & n_3 - n_1 & n_4 - n_1 \end{bmatrix}\right] \tag{7.1}$$

After expanding and removing cross terms we are left with the uncorrelated noise (6.2).

$$\rightarrow E\begin{bmatrix} (n_2^2 + n_1^2) & n_1^2 & n_1^2 \\ n_1^2 & (n_3^2 + n_1^2) & n_1^2 \\ n_1^2 & n_1^2 & (n_4^2 + n_1^2) \end{bmatrix} \tag{7.2}$$

Notice that each noise term is $\propto \sigma_f^2$ resulting in the following:

$$\Sigma = \begin{pmatrix} 2\sigma_f^2 & \sigma_f^2 & \sigma_f^2 \\ \sigma_f^2 & 2\sigma_f^2 & \sigma_f^2 \\ \sigma_f^2 & \sigma_f^2 & 2\sigma_f^2 \end{pmatrix} = \sigma_f^2 \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} = \sigma_f^2 Q \tag{7.3}$$

$$Q = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \tag{7.4}$$

To solve, we are interested in finding unknown parameters $\vec{\theta}$ which our measurements depend on. By using maximum likelihood estimation (MLE) we locate the values that assign the highest probability to the measurements. The MLE is defined as follows:

$$\hat{\vec{\theta}} = \max_{\theta} p(\mathcal{D}|\vec{\theta}) \tag{8}$$

Where $p(\mathcal{D}|\vec{\theta})$ is the probability distribution function of $\mathcal{D}$ given $\vec{\theta}$. When the measurements in $\mathcal{D} = [\overrightarrow{y_1}, \overrightarrow{y_2}, \dots, \overrightarrow{y_n}]$ are identically distributed and independent, the objective function becomes (7.1). Taking the log of both sides gives:

$$\log(p(\mathcal{D}|\vec{\theta})) = \log\left(\Sigma_{i=1}^{m} p(\overrightarrow{y_i}|\vec{\theta})\right) \tag{8.1}$$

In the case where G is a measurement matrix and $\epsilon \sim \mathcal{N}(\epsilon; 0, \Sigma)$, measurements $\vec{y}$ can be written as:

$$\vec{y} = [y_1, y_2, \dots, y_m]^T = G\vec{\theta} + \vec{\epsilon} \tag{8.2}$$

The measurements thus follow a Gaussian distribution $\mathcal{N}(\vec{y}; G\vec{\theta}, \Sigma)$ as shown in (6).

$$p(\vec{y}|\vec{\theta}) = \mathcal{N}(\vec{y}; G\vec{\theta}, \Sigma) \tag{8.3}$$

The log of the multivariate gaussian distribution removes the exponential from the equation giving the following equation.

$$\log\left(p(\vec{y}|\vec{\theta})\right) \propto -(\vec{y} - G\vec{\theta})^T \Sigma^{-1} (\vec{y} - G\vec{\theta}) \tag{8.4}$$

To find the signal source the maximum needs to be found. The equation can be further simplified by removing the negative sign and finding the minimum. This is shown in the equation below.

$$\hat{\theta} = \max_{\vec{\theta}}(-\log\left(p(\vec{y}|\vec{\theta})\right)) = \min_{\vec{\theta}} \left(\vec{y} - \vec{g}(\vec{\theta})\right)^T \Sigma^{-1} \left(\vec{y} - \vec{g}(\vec{\theta})\right) \tag{8.5}$$

The measurement vector in our case is $\vec{f}$, which gives.

$$\rightarrow \hat{\vec{\theta}} = \min_{\vec{\theta}} \left(\vec{f} - \vec{g}(\vec{\theta})\right)^T \Sigma^{-1} \left(\vec{f} - \vec{g}(\vec{\theta})\right) \tag{9}$$

The objective function in this form is in fact a non-linear least squares (NLS) problem.

3. Grid Search

To develop a map of the objective function (the nonlinear least squares problem) the doppler shift of the signal received from each satellite needed to be found. This was done using Equation (3). As the frequency of the source is not known the difference in doppler shift between satellite 1 and 2, 3, and 4 this uses Equation 3.1, this also allows for the generation of the source vector at (5, 10, 0) as shown in Vector 3.2. To get the measured frequency vector for each grid point the coordinates of the 40 by 40 grid were iterated through and solved as shown in Vector 4. At each coordinate the expected frequency vector and the measured frequency vector were substituted into Equation (6) to give $\vec{\theta}$. This would then be recorded into a matrix that was then plotted as a contour map (using Python) to get a visual representation of the objective function. The code that calculates $\vec{\theta}$ for each plot is shown in the python code in Appendix A. The contour plot is shown in Figure 1 below.

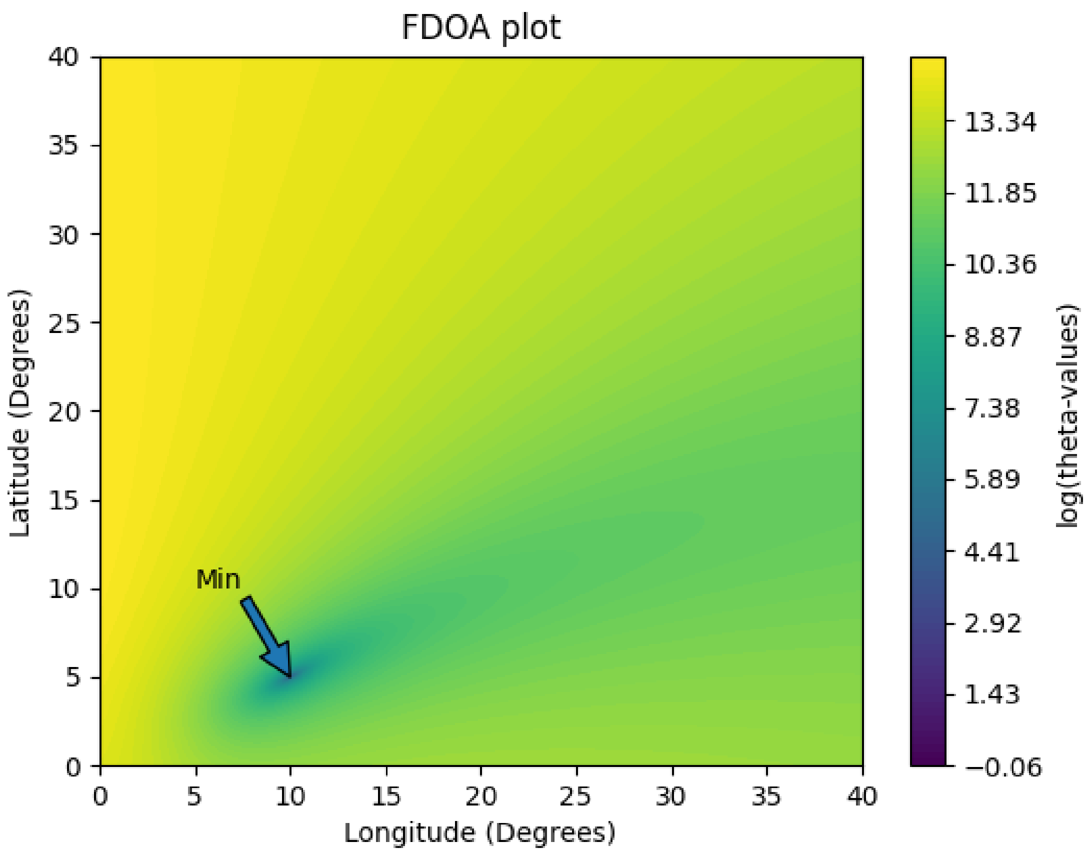

Figure 1: Contour plot of the signal with no noise.

Figure 1 shows that there is a minima at g(5, 10) which is the location of the source. Although this contour graph does not have noise, the following contour map in Figure 2 does. There is a slight error at the minimum with the source of 0.1-degree longitude and latitude.

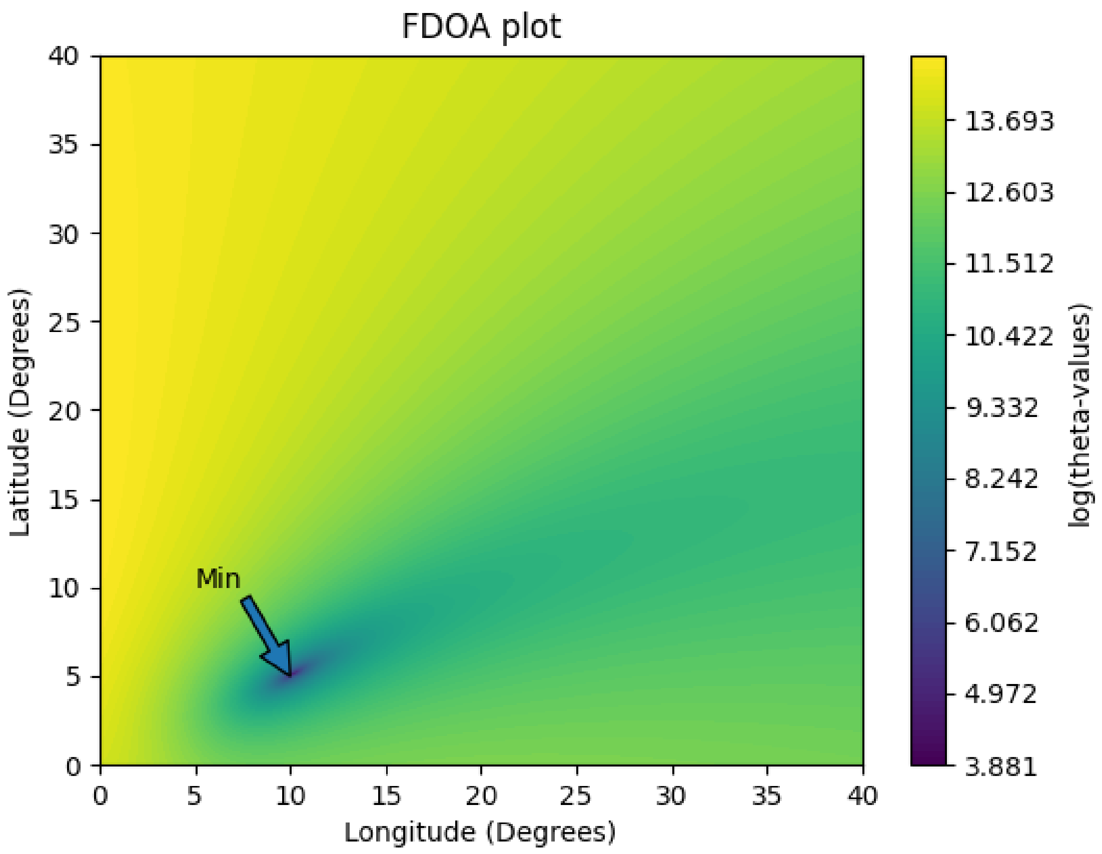

Figure 2: Contour plot of the signal with 1Hz of noise.

To find the minimum within of the data a grid search algorithm was implemented. This is done by iterating over each grid point and checking if the new grid point had a smaller $\vec{\theta}$. Once every grid point was checked, the minimum's coordinates and value is returned. This algorithm was implemented in the function for calculating $\vec{\theta}$ as it already steps through each coordinate with a step size of 0.1. This allows a 0.1-degree accuracy/resolution. The algorithm is shown in the python code in Appendix A. The grid search algorithm was tested with different noise levels. The results for this test are shown in the Table 1.

Table 1: Minima found with different noise levels.

NOSIE DEVIATION (HZ)	**MINIMA (B^0, L^0, H^0)**
0	(5, 10, 0)
1	(5, 10, 0)
2	(5, 10, 0)
4	(5, 10.1, 0)
8	(4.9, 9.9, 0)
16	(5.1, 10.1, 0)

This shows the algorithm is accurate to 0.1-degrees at all frequencies.

4. Determining Accuracy via Monte Carlo Simulation

To determine the accuracy of the grid-search algorithm, a Monte Carlo simulation is run to estimate the root-mean-squared-error. The formula for the calculation is as follows:

$$RMSE = \sqrt{\frac{1}{L}\sum_{l-1}^{L} \|u_l - u^o\|^2} \tag{10}$$

The equation uses the total number of Monte Carlo simulations, L, and the summation of the 2-norm squared difference between the position vector found by the grid search and the actual position vector of the source. The number of Monte Carlo runs used was 30 to test the functionality. More simulations consequently require the program to take too long to run. The way that the algorithm was run is shown in the Monte Carlo function in Appendix A. While testing the grid search, noise levels of $\sigma_f = (1, 2, 4, 8, 16)$ Hz were used. This process gave the results shown in the Table 2 below.

Table 2: Table showing the Deviation used for the noise and the resultant RMSE.

DEVIATION (HZ)	**RMSE**
1	0
2	0
4	0
8	0.14
16	0.2

The data shows that the deviation and the error have a linear relation.

References

[1] L. Yang, "Project 2 FDOA-only Geolocation," 3 2024. [Online]. Available: https://learn.canterbury.ac.nz/course/view.php?id=20007&section=10.

5. Appendix A

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import ticker
import random

#frequency of transmitter
fs = 1*10**9

#wave length
wl = (3*10**8)/fs

u0 = np.array([[5], [10], [0]])

Q = np.array([[2, 1, 1],
             [1, 2, 1],
             [1, 1, 2]])

Qinv = np.linalg.inv(Q)

n1 = random.gauss(0, 0)
n2 = random.gauss(0, 0)
n3 = random.gauss(0, 0)
n4 = random.gauss(0, 0)

nf = np.array([[n2-n1], [n3-n1], [n4-n1]])

#data matrix
theta = np.empty(shape = (161, 161))

#satalites
s1 = np.array([[7378.1, 0, 0]]).T
s1d = np.array([[0.0001, 4.4995, 5.3623]]).T
s2 = np.array([[7377.5, 100, 0]]).T
s2d = np.array([[-0.0671, 4.9493, 4.9497]]).T
s3 = np.array([[7377.5, -100, 0]]).T
s3d = np.array([[0.0610, 4.4991, 5.3623]]).T
s4 = np.array([[7377.5, 0, 100]]).T
s4d = np.array([[-0.0777, 4.0150, 5.7335]]).T

#Should convert from ECEF to LLA
def ECEF_to_LLA(lat, long):
    R0 = 6378.137
    e = 0.081819198425
    lat = np.radians(lat)
    long = np.radians(long)
    Re = R0/(np.sqrt(1-(e)**2*(np.sin(lat))**2))
    x = (Re*np.cos(lat)*np.cos(long))
    y = (Re*np.cos(lat)*np.sin(long))
    z = ((1-e**2)*Re*np.sin(lat))
    u = np.array([[x], [y], [z]])
    return u

#grid search doppler shift
def dopler_shift(lat, long):
    u = ECEF_to_LLA(lat, long)
    f0 = 1/wl*np.dot((u - s1).T , s1d)/(np.linalg.norm(u-s1))
    f1 = 1/wl*np.dot((u - s2).T , s2d)/(np.linalg.norm(u-s2)) - f0
    f2 = 1/wl*np.dot((u - s3).T , s3d)/(np.linalg.norm(u-s3)) - f0
    f3 = 1/wl*np.dot((u - s4).T , s4d)/(np.linalg.norm(u-s4)) - f0
    f = np.array([f1[0], f2[0], f3[0]])
    return f

#grid search calc
def grid_search():
    long = 0
    lat  = 0
    min = float('inf')
    coord = 0

    f = dopler_shift(5, 10)
    for i in range(0, 161):
        for j in range(0, 161):
            g = dopler_shift(lat, long)
            g += nf
            theta[j][i] = np.log((f - g).T @ Qinv @ (f - g))
            if theta[j][i] < min: #grid search stuff
                min = theta[j][i]
                coord = np.array([[j], [i], [0]])*0.25
            lat += 0.25
        lat = 0
        long += 0.25
    return coord, min

#monte carlo simulation to find the accuracy of the data
def monte_carlo():
    sumation = 0
    for l in range(0, 30):
        coord, min = grid_search()
        sumation += np.linalg.norm(coord - u0)**2

    RMSE = np.sqrt((1/30)*sumation)
    print(RMSE)

#plotting stuff
def graph():
    coord, min = grid_search()
    feature_x = np.arange(0, 40.25, 0.25)
    feature_y = np.arange(0, 40.25, 0.25)

    [X, Y] = np.meshgrid(feature_x, feature_y)

    contour = plt.contourf(X, Y, theta)

    plt.colorbar(contour, label='log(theta-values)')

    plt.xlabel("Longitude (Degrees)")
    plt.ylabel("Latitude (Degrees)")
    plt.title("FDOA plot")
    print("Minmum is found at: ", coord.T)
    plt.show()

def main():
    monte_carlo()
    graph()

main()
import numpy as np
import matplotlib.pyplot as plt
import random


fs = 1*10**9       #frequency of transmitter
wl = (3*10**5)/fs #wave length

#source location
u0 = np.array([[5], [10], [0]])

#Data matrix
theta = np.empty(shape = (401, 401))

#Weighting function
Q = np.array([[2, 1, 1],
            [1, 2, 1],
            [1, 1, 2]])

Qinv = np.linalg.inv(Q)

#making the noise vector
n1 = random.gauss(0, 16)
n2 = random.gauss(0, 16)
n3 = random.gauss(0, 16)
n4 = random.gauss(0, 16)
nf = np.array([[n2-n1], [n3-n1], [n4-n1]])

#satalites
s1 = np.array([[7378.1, 0, 0]]).T
s1d = np.array([[0.0001, 4.4995, 5.3623]]).T
s2 = np.array([[7377.5, 100, 0]]).T
s2d = np.array([[-0.0671, 4.9493, 4.9497]]).T
s3 = np.array([[7377.5, -100, 0]]).T
s3d = np.array([[0.0610, 4.4991, 5.3623]]).T
s4 = np.array([[7377.5, 0, 100]]).T
s4d = np.array([[-0.0777, 4.0150, 5.7335]]).T

#Should convert from ECEF to LLA
def LLA_to_ECEF(lat, long):
    R0 = 6378.137
    e = 0.081819198425
    lat = np.radians(lat)
    long = np.radians(long)
    Re = R0/(np.sqrt(1-(e)**2*(np.sin(lat))**2))
    x = (Re*np.cos(lat)*np.cos(long))
    y = (Re*np.cos(lat)*np.sin(long))
    z = ((1-e**2)*Re*np.sin(lat))
    u = np.array([[x], [y], [z]])
    return u

#grid search doppler shift
def dopler_shift(lat, long):
    u = LLA_to_ECEF(lat, long)
    f0 = 1/wl*np.dot((u - s1).T , s1d)/(np.linalg.norm(u-s1))
    f1 = 1/wl*np.dot((u - s2).T , s2d)/(np.linalg.norm(u-s2)) - f0
    f2 = 1/wl*np.dot((u - s3).T , s3d)/(np.linalg.norm(u-s3)) - f0
    f3 = 1/wl*np.dot((u - s4).T , s4d)/(np.linalg.norm(u-s4)) - f0
    f = np.array([f1[0], f2[0], f3[0]])
    return f

#grid search calc
```

```
def grid_search():
    long = 0
    lat  = 0
    min = float('inf')
    coord = 0

    f = dopler_shift(5, 10)
    for i in range(0, 401):
        for j in range(0, 401):
            g = dopler_shift(lat, long)
            g += nf
            theta[j][i] = np.log((f - g).T @ Qinv @ (f - g))

            if theta[j][i] < min: #grid search stuff
                min = theta[j][i]
                coord = np.array([[j], [i], [0]])*0.1
            lat += 0.1
            lat = round(lat, 1)
        lat = 0
        long += 0.1
        long = round(long, 1)

    return coord, min

#monte carlo simulation to find the accuracy of the data
def monte_carlo():
    sumation = 0
    L = 30
    for l in range(0, L):
        coord, min = grid_search()
        sumation += np.linalg.norm(coord - u0)**2

    RMSE = np.sqrt((1/L)*sumation)
    print(RMSE)

#plotting stuff
def graph():
    coord, min = grid_search()
    tmin = np.min(theta)
    tmax = np.max(theta)
    levels = np.linspace(tmin, tmax, 100)
    feature_x = np.arange(0, 40.1, 0.1)
    feature_y = np.arange(0, 40.1, 0.1)

    [X, Y] = np.meshgrid(feature_x, feature_y)
    contour = plt.contourf(X, Y, theta, levels = levels)

    plt.colorbar(contour, label='log(theta-values)')

    plt.xlabel("Longitude (Degrees)")
    plt.ylabel("Latitude (Degrees)")
    plt.title("FDOA plot")
    print("Minmum is found at: ", coord.T)
    plt.annotate('Min',xy=(10,5),xytext=(5,10),arrowprops={})

    plt.show()

def main():
    #monte_carlo()
    graph()

main()
```