ENEL445: Engineering Optimization

Source Geo-Location
Final Report

Submitted by:

Daegan Rose-Love (33480267)

Ben Mangin (92699294)

Date: 31/05/2024

Table of Contents

1. Background

1.1. Brief

Radio source geo-location involves locating the coordinates of where a particular signal is broadcast 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 vectors (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.

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

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

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

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

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 Earth's equatorial radius $R_0 = 6378.137\ km$ and the Earth's 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 and the position vector of the point that is being measured, the position vector of the satellite, the velocity vector of the satellite, and additive noise. 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).

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

$$\rightarrow \vec{f} = \begin{bmatrix} f_{21} & f_{31} & f_{41} \end{bmatrix}^T \tag{3.2}$$

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 within 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 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.

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

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 assumptions 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) the following is found:

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

After expanding and removing cross terms we are left with the uncorrelated noise (7.2).

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

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

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

Where $p\left(\mathcal{D}\middle|\vec{\theta}\right)$ 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 (8.1). Taking the log of both sides gives:

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

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 this 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 satellites 1 and 2, 3, and 4 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. 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 for each plot is shown in the Python code in Appendix A. The contour plot is shown in Figure 1 below.

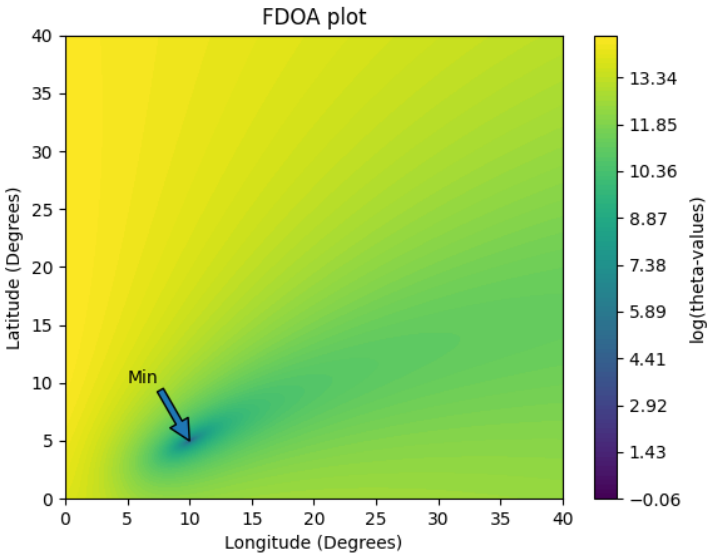

Figure 1: Contour plot of the signal with no noise.

Figure 1 shows that there is a minimum 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. The noise added to the simulation for Figure 2 was 1Hz which resulted in the minimum being found 600 meters away from the true minimum.

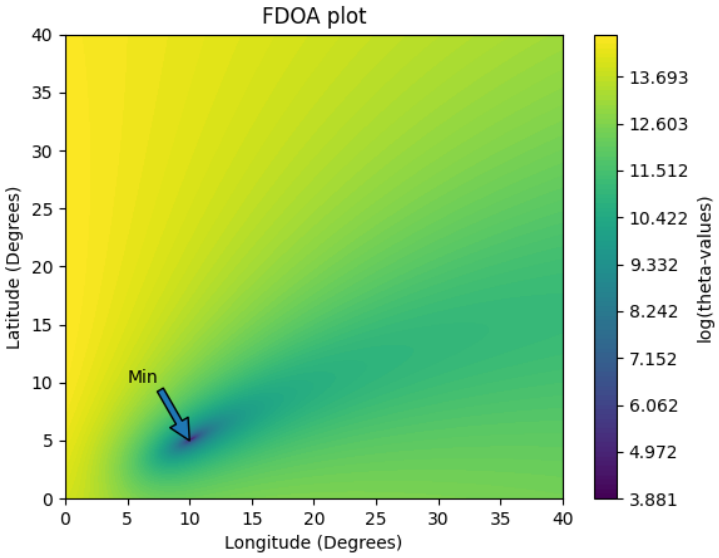

Figure 2: Contour plot of the signal with 1Hz of noise.

To find the minimum within the data a grid search algorithm was implemented. This is done by iterating over each grid point and checking if the new grid point is smaller. Once every grid point is checked, the coordinates at that minimum are returned. This algorithm was implemented in the function for calculating as it already steps through each coordinate with a step size of 0.01. This allows a 0.01-degree accuracy/resolution. The algorithm is shown in the Python code in Appendix B. The grid search algorithm was tested with different noise levels. The results of this test are shown in 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)

4. Find FDOA With Unknown Frequency

In a real-world situation, the original frequency of the transmitter is unknown which means that the wavelength of the signal cannot be found. To get around this an estimation of the original frequency needs to be made to get an estimate of the wavelength. This can be obtained by evaluating the frequency at satellite j using the following equation.

$$f_j = \frac{1}{\lambda}\frac{(u^0 - s_i)^T \dot{s}_i}{\|u^0 - s_i\|} + f_0, \qquad for\ j = 1,2,3,4 \tag{10}$$

The frequencies at each satellite can then be averaged to get an estimate of the frequency as shown in equation 11.

$$f_e = \frac{f_1 + f_2 + f_3 + f_4}{4} \tag{11}$$

Using this estimate of the frequency the wavelength can be found as shown below where c is the speed of light and λ is the wavelength.

$$\frac{1}{\lambda} = \frac{c}{f_e} \tag{12}$$

This can be rearranged to give lambda as shown in equation 12.1.

$$\lambda = \frac{f_e}{c} \tag{12.1}$$

This can then be substituted back into equation 3.1 to give the frequency difference between satellites as shown in equation 13.

$$f_{j1} = \frac{f_e}{c}\frac{\left(\overrightarrow{u^0} - \overrightarrow{s_j}\right)^T \dot{s}_j}{\left\|\overrightarrow{u^0} - \overrightarrow{s_j}\right\|} - \frac{f_e}{c}\frac{\left(\overrightarrow{u^0} - \overrightarrow{s_1}\right)^T \dot{s}_1}{\left\|\overrightarrow{u^0} - \overrightarrow{s_1}\right\|} + (n_j - n_1), \qquad for\ j = 2,3,4 \tag{13}$$

This now allows for the frequency difference between satellites to be found so the objective function can be formulated. Results for this frequency estimation can be seen in Figure 3. This was also verified using the grid search algorithm to check the minimum was at (5,10) which it was.

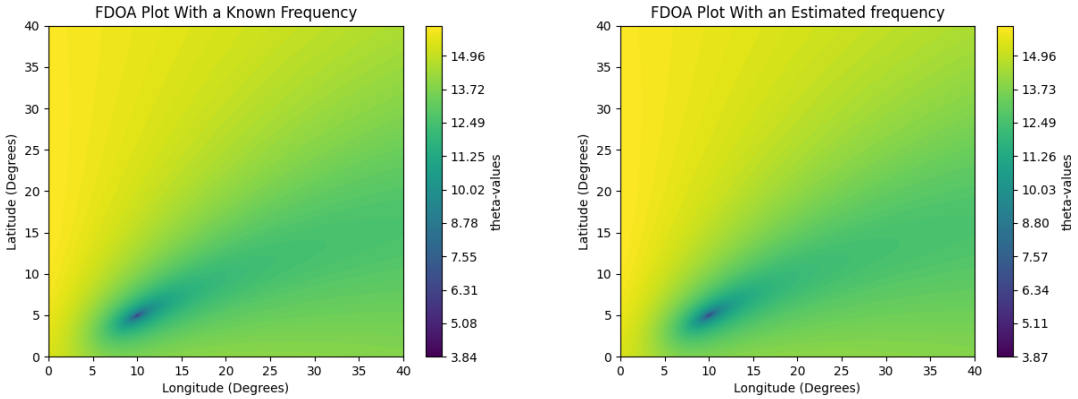

Figure 3: Two plots showing the difference between plots of the know frequency and the estimated frequency.

Note: Using more FDOA measurements such as f_{23}, f_{13}, f_{43} does not result in any performance improvements. The additional measurements are the result of values being shifted but the data in the matrix retains the same information with a different reference satellite, this can be shown mathematically:

$$\begin{pmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} f_{21} \\ f_{31} \\ f_{41} \end{pmatrix} = \begin{pmatrix} 0 & 1 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} f_{23} \\ f_{13} \\ f_{43} \end{pmatrix} \tag{13.1}$$

5. Gradient Derivation

To apply gradient-based optimization methods, the gradient must be computed or approximated. From Equation 3.1 we have:

$$f_{j1} = \frac{1}{\lambda}\frac{\left(\overrightarrow{u^0} - \overrightarrow{s_j}\right)^T \dot{s_j}}{\left\|\overrightarrow{u^0} - \overrightarrow{s_j}\right\|} - \frac{1}{\lambda}\frac{\left(\overrightarrow{u^0} - \overrightarrow{s_1}\right)^T \dot{s_1}}{\left\|\overrightarrow{u^0} - \overrightarrow{s_1}\right\|}, \frac{\partial f_{j1}}{\partial \vec{u}^0} = 0 \tag{14}$$

Let

$$\vec{a} = \vec{u}^0 - \overrightarrow{s_j} \tag{14.1}$$

$$\overrightarrow{a_1} = \vec{u}^0 - \overrightarrow{s_1} \tag{14.2}$$

$$f_{j1} = \frac{1}{\lambda}\frac{\vec{a}^T \dot{s_j}}{\|\vec{a}\|} - \frac{1}{\lambda}\frac{\overrightarrow{a_1}^T \dot{s_1}}{\|\overrightarrow{a_1}\|}, \frac{\partial f_{j1}}{\partial \vec{u}^0}, \quad \frac{\partial f_{j1}}{\partial \vec{u}^0} = \frac{\partial f_{j1}}{\partial \vec{a}}\frac{\partial \vec{a}}{\partial \vec{u}^0} - \frac{\partial f_{j1}}{\partial \vec{a}}\frac{\partial \vec{a}}{\partial \vec{u}^0} \tag{14.3}$$

Let

$$\|\vec{a}\| = b^{\frac{1}{2}} = \sqrt{\sum a_i^2} \tag{14.4}$$

$$f_{j1} = \frac{1}{\lambda}\frac{\left(\vec{a}^T \dot{s}_j\right)}{b^{\frac{1}{2}}} - \frac{1}{\lambda}\frac{\left(\overrightarrow{a_1}^T \dot{s}_1\right)}{b^{\frac{1}{2}}} \tag{14.5}$$

$$\frac{\partial f_{j1}}{\partial \vec{a}} = \frac{\partial}{\partial \vec{a}}\left(\frac{1\vec{a}_1{}^T \dot{s}_j b^{-\frac{1}{2}}}{\lambda}\right) \tag{14.6}$$

Using product rule and chain rule gives:

$$\frac{\partial f_{j1}}{\partial \vec{u}^0} = \left(\frac{1}{\lambda}\left(\dot{s}_j b^{-\frac{1}{2}} - \frac{a^T \dot{s}_j}{2b^{\frac{3}{2}}}\frac{\partial b}{\partial \vec{a}}\right)\right)\frac{\partial \vec{a}}{\partial \vec{u}^0} \tag{14.7}$$

$$\left(b^{\frac{1}{2}}\right)^2 = \left(\sqrt{\sum a_i^2}\right)^2 = \sum a_i^2 = b, \qquad \frac{\partial b}{\partial a} = \sum 2a_i \tag{14.8}$$

$$\frac{\partial \vec{a}}{\partial \vec{u}^0} = \frac{\partial}{\partial \vec{u}^0}\left(\vec{u}^0 - \vec{s}_j\right) = \begin{bmatrix}1\\1\\1\end{bmatrix} \tag{14.9}$$

$$\frac{\partial f_{j1}}{\partial \vec{u}^0} = \left(\frac{1}{\lambda}\left(\frac{\dot{s}_j}{b^{\frac{1}{2}}} - \frac{a^T \dot{s}_j}{2b^{\frac{3}{2}}}(2\vec{a})\right)\right)\begin{bmatrix}1\\1\\1\end{bmatrix} - \left(\frac{1}{\lambda}\left(\frac{\dot{s_1}}{b^{\frac{1}{2}}} - \frac{a^T \dot{s_1}}{2b^{\frac{3}{2}}}(2\vec{a}_1)\right)\right)\begin{bmatrix}1\\1\\1\end{bmatrix} \tag{14.10}$$

$$\frac{\partial f_{j1}}{\partial \vec{u}^0} = \left(\frac{1}{\lambda}\left(\frac{\dot{s_j}}{\|\vec{a}\|} - \frac{a^T \dot{s_j}}{2\|\vec{a}\|^3}(2\vec{a})\right)\right) - \left(\frac{1}{\lambda}\left(\frac{\dot{s_1}}{\|\overrightarrow{a_1}\|} - \frac{a_1^T \dot{s_1}}{2\|\overrightarrow{a_1}\|^3}(2\overrightarrow{a_1})\right)\right) \tag{14.11}$$

Substituting $\vec{a}$ back in gives the derivative:

$$\frac{\partial f_{j1}}{\partial \vec{u}^0} = \left(\frac{1}{\lambda}\left(\frac{\dot{s_j}}{\|\vec{u}^0 - \overrightarrow{s_j}\|} - \frac{(\vec{u}^0 - \overrightarrow{s_j})^T \dot{s_j}}{\|\vec{u}^0 - \overrightarrow{s_j}\|^3}(\vec{u}^0 - \overrightarrow{s_j})\right)\right) - \left(\frac{1}{\lambda}\left(\frac{\dot{s_1}}{\|\vec{u}^0 - \overrightarrow{s_1}\|} - \frac{(\vec{u}^0 - \overrightarrow{s_1})^T \dot{s_1}}{\|\vec{u}^0 - \overrightarrow{s_1}\|^3}(\vec{u}^0 - \overrightarrow{s_1})\right)\right) \tag{14.12}$$

The resulting Jacobian can be written in the following form:

$$J_{\vec{f}}(\vec{u}) = \begin{pmatrix} \frac{\partial f_{21}}{\partial \vec{x}} & \frac{\partial f_{21}}{\partial \vec{y}} & \frac{\partial f_{21}}{\partial \vec{z}} \\ \frac{\partial f_{31}}{\partial \vec{x}} & \frac{\partial f_{31}}{\partial \vec{y}} & \frac{\partial f_{31}}{\partial \vec{z}} \\ \frac{\partial f_{41}}{\partial \vec{x}} & \frac{\partial f_{41}}{\partial \vec{y}} & \frac{\partial f_{41}}{\partial \vec{z}} \end{pmatrix} = \begin{pmatrix} \nabla f_{21}(\vec{u})^T \\ \nabla f_{31}(\vec{u})^T \\ \nabla f_{41}(\vec{u})^T \end{pmatrix} \tag{14.13}$$

The derivatives of the coordinate conversion functions (Eq. 2, 2.1, 2.2, 2.3, 2.4) are used to find the derivative of f_{j1} with respect to $\vec{p}$ (Eq. 1.1). The resulting Jacobian $J_{\vec{f}}(\vec{p})$ is shown below, with derived terms.

$$J_f(\vec{p}) = \begin{pmatrix} \frac{\partial x}{\partial B} & \frac{\partial x}{\partial L} & \frac{\partial x}{\partial H} \\ \frac{\partial y}{\partial B} & \frac{\partial y}{\partial L} & \frac{\partial y}{\partial H} \\ \frac{\partial z}{\partial B} & \frac{\partial z}{\partial L} & \frac{\partial z}{\partial H} \end{pmatrix} \tag{15}$$

The desired Jacobian for $\frac{\partial \vec{f}}{\partial \vec{p}}$ is:

$$J_f(\vec{u}, \vec{p}) = J_{\vec{f}}(\vec{u}) J_{\vec{f}}(\vec{p}) \tag{15.1}$$

The derivation results follow:

$$\frac{\partial R_e}{\partial B} = \frac{R_0 e^2 \cos(B)\sin(B)}{(1 - e^2 \sin^2(B))^{\frac{2}{3}}} \tag{15.2}$$

$$\frac{\partial x^0}{\partial B^0} = \cos(L^0)\left[\frac{\partial R_e}{\partial B}\cos(B^0) - R_e(B^0)\sin(B^0)\right] \tag{15.3}$$

$$\frac{\partial x^0}{\partial L^0} = R_e(B^0)\cos(B^0)\sin(L^0) \tag{15.4}$$

$$\frac{\partial y^0}{\partial B^0} = \sin(L^0)\left[\frac{\partial R_e}{\partial B}\cos(B^0) - R_e(B^0)\sin(B^0)\right] \tag{15.5}$$

$$\frac{\partial y^0}{\partial L^0} = R_e(B^0)\cos(B^0)\cos(L^0) \tag{15.6}$$

$$\frac{\partial z^0}{\partial B^0} = (1-e^2)\left(\frac{\partial R_e}{\partial B}\sin(B^0) + R_e(B^0)\cos(B)\right) \tag{15.7}$$

$$\frac{\partial z^0}{\partial L^0} = 0 \tag{15.8}$$

$$J_f(\vec{p}) = \begin{bmatrix} \frac{\partial x^0}{\partial B^0} & \frac{\partial x^0}{\partial L^0} \\ \frac{\partial y^0}{\partial B^0} & \frac{\partial y^0}{\partial L^0} \\ \frac{\partial z^0}{\partial B^0} & \frac{\partial z^0}{\partial L^0} \end{bmatrix} \tag{15.9}$$

Note that the H column is omitted as the altitude in degrees (H) is zero. Using Equation 14.13 and Equation 15.9 in Equation 15.1 results in the gradient used for the gradient-based optimization method in section 7.

6. Derivative Verification

This section provides a verification of the $f_{21}(B, L)$ derivative using the finite-difference and complex step methods.

The finite-difference method formula is:

$$\frac{\partial f}{\partial x_j} = \frac{f\left(x + h\hat{e}_j\right) - f(x)}{h} + \mathcal{O}(h) \tag{16}$$

From Equation 3.1 and Equation 16 we obtain:

$$\frac{\partial f_{21}}{\partial B} = \frac{\left(\frac{1}{\lambda}\left(\frac{(\vec{u}+h\hat{e}_B-\vec{s_2})^T\dot{s_2}}{\left||\vec{u}+h\hat{e}_B-\vec{s_2}|\right|} - \frac{(\vec{u}+h\hat{e}_B-\vec{s_1})^T\dot{s_1}}{\left||\vec{u}+h\hat{e}_B-\vec{s_1}|\right|}\right) - \frac{1}{\lambda}\left(\frac{(\vec{u}-\vec{s_2})^T\dot{s_2}}{\left||\vec{u}-\vec{s_2}|\right|} - \frac{(\vec{u}-\vec{s_1})^T\dot{s_1}}{\left||\vec{u}-\vec{s_1}|\right|}\right)\right)}{h} \tag{16.1}$$

$$\frac{\partial f_{21}}{\partial L} = \frac{\left(\frac{1}{\lambda}\left(\frac{(\vec{u}+h\hat{e}_L-\vec{s_2})^T\dot{s_2}}{\left||\vec{u}+h\hat{e}_L-\vec{s_2}|\right|} - \frac{(\vec{u}+h\hat{e}_L-\vec{s_1})^T\dot{s_1}}{\left||\vec{u}+h\hat{e}_L-\vec{s_1}|\right|}\right) - \frac{1}{\lambda}\left(\frac{(\vec{u}-\vec{s_2})^T\dot{s_2}}{\left||\vec{u}-\vec{s_2}|\right|} - \frac{(\vec{u}-\vec{s_1})^T\dot{s_1}}{\left||\vec{u}-\vec{s_1}|\right|}\right)\right)}{h} \tag{16.2}$$

$$\frac{\partial f_{21}}{\partial H} = \frac{\left(\frac{1}{\lambda}\left(\frac{(\vec{u}+h\hat{e}_H-\overrightarrow{s_2})^T\dot{s_2}}{\left||\vec{u}+h\hat{e}_H-\overrightarrow{s_2}|\right|}-\frac{(\vec{u}+h\hat{e}_H-\overrightarrow{s_1})^T\dot{s_1}}{\left|\vec{u}+h\hat{e}_H-\overrightarrow{s_1}\right|}\right)-\frac{1}{\lambda}\left(\frac{(\vec{u}-\overrightarrow{s_2})^T\dot{s_2}}{\left|\vec{u}-\overrightarrow{s_2}\right|}-\frac{(\vec{u}-\overrightarrow{s_1})^T\dot{s_1}}{\left|\vec{u}-\overrightarrow{s_1}\right|}\right)\right)}{h} \quad (16.3)$$

The complex step formula is:

$$\frac{\partial f}{\partial x_j} = \frac{\Im\left(f\left(x+ih\hat{e}_j\right)\right)}{h}+\mathcal{O}(h^2) \quad (17)$$

$$\frac{\partial f_{21}}{\partial B} = \frac{\Im\left(\frac{1}{\lambda}\left(\frac{(\vec{u}+ih\hat{e}_B-\overrightarrow{s_2})^T\dot{s_2}}{\left||\vec{u}+ih\hat{e}_B-\overrightarrow{s_2}|\right|}-\frac{(\vec{u}+ih\hat{e}_B-\overrightarrow{s_1})^T\dot{s_1}}{\left|\vec{u}+ih\hat{e}_B-\overrightarrow{s_1}\right|}\right)\right)}{h} \quad (17.1)$$

$$\frac{\partial f_{21}}{\partial L} = \frac{\Im\left(\frac{1}{\lambda}\left(\frac{(\vec{u}+ih\hat{e}_L-\overrightarrow{s_2})^T\dot{s_2}}{\left||\vec{u}+ih\hat{e}_L-\overrightarrow{s_2}|\right|}-\frac{(\vec{u}+ih\hat{e}_L-\overrightarrow{s_1})^T\dot{s_1}}{\left|\vec{u}+ih\hat{e}_L-\overrightarrow{s_1}\right|}\right)\right)}{h} \quad (17.2)$$

$$\frac{\partial f_{21}}{\partial H} = \frac{\Im\left(\frac{1}{\lambda}\left(\frac{(\vec{u}+ih\hat{e}_H-\overrightarrow{s_2})^T\dot{s_2}}{\left||\vec{u}+ih\hat{e}_H-\overrightarrow{s_2}|\right|}-\frac{(\vec{u}+ih\hat{e}_H-\overrightarrow{s_1})^T\dot{s_1}}{\left|\vec{u}+ih\hat{e}_H-\overrightarrow{s_1}\right|}\right)\right)}{h} \quad (17.3)$$

Where $\vec{e}_j$ is a unit vector in the j_{th} direction (B, L, H for longitude, latitude, and altitude respectively), h is a step size, and $\vec{u}$ is the point about which to differentiate (in B, L, H). The approximation of the derivatives (Eq. 16.1, 16.2, 16.3, 17.1, 17.2, 17.3) and the analytical derivative in Equation 13.12 is compared, with results shown in Table 2:

Table 2: Comparison of derivative approximations and analytical solutions.

Method	Derivative	Step size	Point (B, L, H)	Result
Analytical	$\frac{\partial f_{21}}{\partial B}$	10^{-8}	$(5, 10, 0)$	-0.57761319
Finite Difference	$\frac{\partial f_{21}}{\partial B}$	10^{-8}	$(5, 10, 0)$	-0.5775973477284424
Complex Step	$\frac{\partial f_{21}}{\partial B}$	10^{-200}	$(5, 10, 0)$	-0.5834981372799554
Analytical	$\frac{\partial f_{21}}{\partial L}$	10^{-8}	$(5, 10, 0)$	3.90670032
Finite Difference	$\frac{\partial f_{21}}{\partial L}$	10^{-8}	$(5, 10, 0)$	3.9067515444912715
Complex Step	$\frac{\partial f_{21}}{\partial L}$	10^{-200}	$(5, 10, 0)$	3.905921531938494
Analytical	$\frac{\partial f_{21}}{\partial H}$	10^{-8}	$(5, 10, 0)$	-3.58224626
Finite Difference	$\frac{\partial f_{21}}{\partial H}$	10^{-8}	$(5, 10, 0)$	-3.5822438348986907
Complex Step	$\frac{\partial f_{21}}{\partial H}$	10^{-200}	$(5, 10, 0)$	-3.5822462633136434

It is interesting that the complex step method is very accurate on the altitude derivative but less accurate on the latitude and longitude than the finite difference method. The error resulting from both approximations also appears to be larger than the $\mathcal{O}(h)$ and $\mathcal{O}(h^2)$ remainder terms implied by Equations 16 & 17. This could be due to the computer's finite arithmetic or how functions are being handled "behind the scenes" in the Python code. Still, the accuracy of the approximations is enough to verify the $\frac{\partial f_{21}}{\partial \vec{u}}$ derivatives. It would follow that the other FDOA function derivatives are correct due to the implementation being the same.

7. Gradient Based algorithm

The Optimization method used to find the source location was the Levenberg-Marquardt algorithm (LMA). This is a surrogate-based optimization method that estimates the hessian. LMA was developed to solve non-linear least-square equations. The LMA interpolates between the Gauss-Newton algorithm (GNA) and gradient-based optimization. GNA is very fast at converging to the minimum but is not very accurate once it gets close. This is where the gradient-based algorithm works best. LMA will get close to the solution with GNA and then switch to the steepest decent method to get a more precise solution. A derivation of the update formula (Eq. 18) can be found in Appendix A.

$$s = (J^T J - \mu D)^{-1} J^T \big(y - f(x)\big) \tag{18}$$

Where J is the Jacobian of the frequency derivatives for each satellite at the current position, y is the evaluation of the frequency at the source location using equation 4, $f(x)$ is the evaluation of

the frequency at the estimated position that the satellites are measuring, μ is the control of the direction ranging between GNA and steepest decent. We can improve the scaling of the function by multiplying by D. Where D is:

$$D = diag(J^T J) \tag{18.1}$$

The algorithm works by initializing the code with an initial position, ρ, μ, and initial position x. The Jacobian and the residual will then be evaluated. The first step is calculated and added to the current position, and the error is then found using Equation 18.2. This is the error for the next step (e_s).

$$es = \|e_s\|^2 \tag{18.2}$$

The difference in the current error e and the previous error e_s is then found using Equation 18.3.

$$\Delta = es - e \tag{18.3}$$

If the difference is negative then the new position will be accepted and e, the Jacobian and the residual will be updated. For every fifth iteration, the μ value will be divided by ρ (the damping factor) making it act more like GNA. If the difference in error is not negative, μ will be multiplied by ρ to make the algorithm act more like a steepest decent. This code will then loop and check if the difference in errors is less than a tolerance of 10^{-3}. If the error is less than the tolerance it will return the position, if not the code will loop again. This process is shown in the flow diagram in Figure 4.

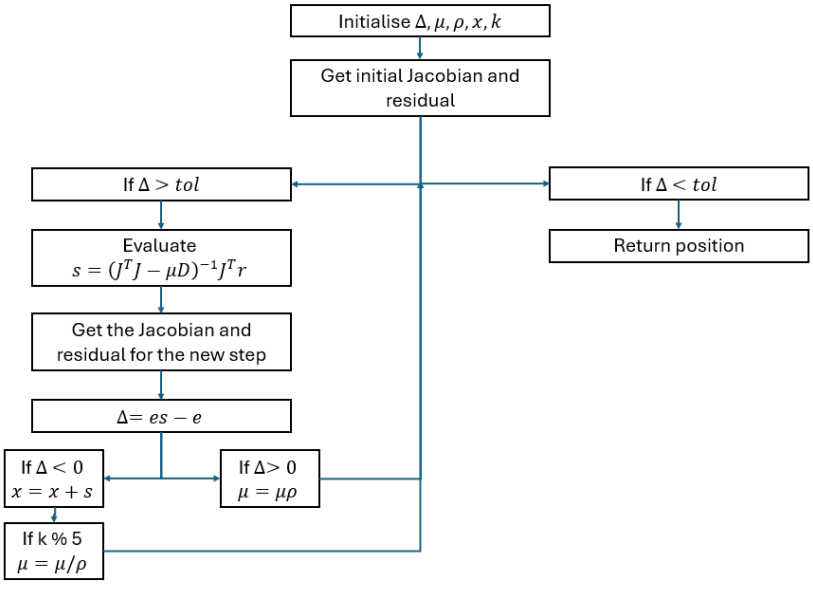

Figure 4: Flow chart of the algorithim.

Figure 5 shows the convergence of the LMA algorithm from multiple starting positions. This shows that the algorithm converges to the same position from many different start positions.

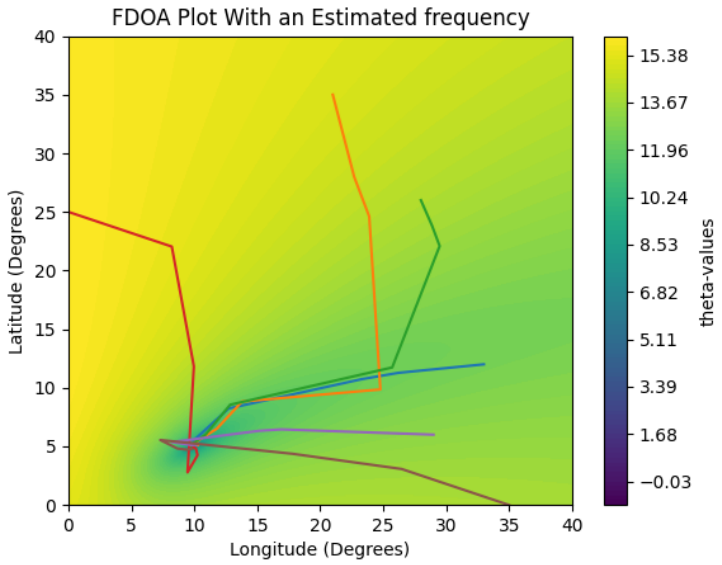

Figure 5: Plot of the contour and the path of the algorithm.

Delta (Eq. 18.3) is used as the stopping criteria for the algorithm. Once delta is less than 10^{-3} the algorithm stops looping and returns the position. The LMA algorithm converges in 15- 30 iterations. It will evaluate the Jacobian and the residual $k + 1$ times where k is the number of iterations it takes to converge. The step calculation will be evaluated once every iteration so will have the same number of evaluations as iterations. The relation between the number of evaluations, the difference in the current error and next step error (delta) can be seen in Figure 6 below.

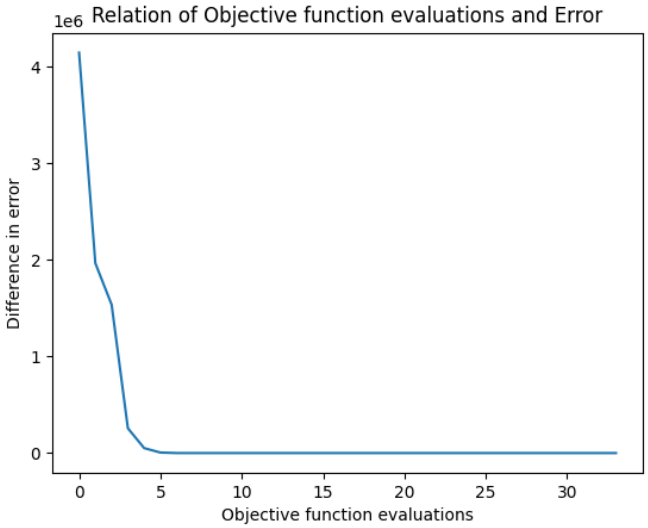

Figure 6: Plot showing the convergence of the algorithm.

8. Monty Carlo Comparison

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

$$RMSE = \sqrt{\frac{1}{L}\sum_{l-1}^{L}\|u_l - u^o\|^2} \tag{19}$$

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 10^4. The way that the algorithm was run is shown in the Monte Carlo function in Appendix B. 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 Table 3 below.

Table 3: Table of the Monty Carlo results.

NOISE LEVEL (HZ)	LMA (KM)	GRID SEARCH (KM)
1	1.0941220938193204	0.6197978901842434
2	2.1879750170699395	1.0849477826620166
4	4.365063017360187	2.2123866587402907
8	8.725234012378314	4.227109425169162
16	17.44703333621689	8.175176490117455

The results from the RMSE were plotted in Figure 7. This shows that the results are linear and that the LMA algorithm follows the same trend as the grid search. The relationship between the grid search and the LMA was that the error doubled what the grid search error was.

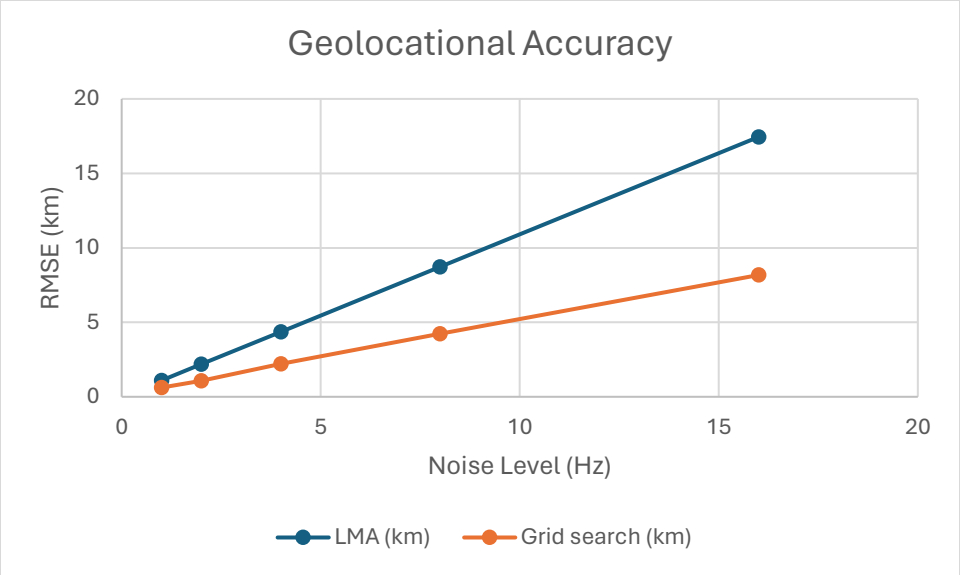

Figure 7 7: Plot showing the RMSE evaluation of the grid search algorithm and the LMA algorithm in km.

Appendix A: Levenberg-Marquardt Update Derivation

Let y be the true function value, $f(B, L)$ the next function estimate and $f(B_0, L_0)$ the current function estimate.

Let $\vec{x}$ denote (B, L) and $\overrightarrow{x_0}$ denote (B_0, L_0).

$$\big(y - f(B,L)\big)^T \big(y - f(B,L)\big) \tag{A1}$$

$$f(B,L) = f(B_0,L_0) + J(B_0,L_0)\begin{pmatrix} B - B_0 \\ L - L_0 \end{pmatrix} \tag{A1.1}$$

$$\left(y - f(B_0,L_0) + J(B_0,L_0)\begin{pmatrix} B - B_0 \\ L - L_0 \end{pmatrix}\right)^T \left(y - f(B_0,L_0) + J(B_0,L_0)\begin{pmatrix} B - B_0 \\ L - L_0 \end{pmatrix}\right) \tag{A1.2}$$

$$g = y - f(\overrightarrow{x_0}) + J(\overrightarrow{x_0})^T \overrightarrow{x_0} \tag{A1.3}$$

$$(g - J(\overrightarrow{x_0})^T \vec{x})^T (g - J(\overrightarrow{x_0})^T \vec{x}) \tag{A1.4}$$

$$Let\ u_i = g - J(\overrightarrow{x_0})^T \vec{x} \tag{A1.5}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u}\ \frac{\partial u}{\partial x} \tag{A1.6}$$

$$\rightarrow \frac{\partial}{\partial x} \Sigma u_i^2 \rightarrow 2u_i \frac{\partial u_i}{\partial x} \tag{A1.7}$$

$$\frac{\partial u_i}{\partial x} = -J(\overrightarrow{x_0}) + \frac{\partial g}{\partial x} \tag{A1.8}$$

$$\frac{\partial g}{\partial x} = 0 \tag{A1.9}$$

$$\rightarrow \frac{\partial f}{\partial x} = 2(g - J(\overrightarrow{x_0})^T \vec{x})\big(-J(\overrightarrow{x_0})\big) \tag{A1.10}$$

$$\rightarrow -2g^T J(\overrightarrow{x_0}) + 2J(\overrightarrow{x_0})^T \vec{x} J(\overrightarrow{x_0}) \tag{A1.11}$$

$$\rightarrow J(\overrightarrow{x_0})^T \vec{x} J(\overrightarrow{x_0}) = g^T J(\overrightarrow{x_0}) \tag{A1.12}$$

$$\vec{x} = \big(J(\overrightarrow{x_0})^T J(\overrightarrow{x_0})\big)^{-1} g^T J(\overrightarrow{x_0}) \tag{A1.13}$$

$$\begin{pmatrix} B \\ L \end{pmatrix} = \big(J(B_0,L_0)^T J(B_0,L_0)\big)^{-1} g^T J(B_0,L_0) \tag{A1.14}$$

For the Levenberg-Marquardt Algorithm a factor of μD is added, where μ is a scaling constant.

$$D = diag\big(J(B_0,L_0)^T J(B_0,L_0)\big) \tag{A1.15}$$

$$\binom{B}{L} = -(J(B_0, L_0)^T J(B_0, L_0) + \mu D)^{-1} g^T J(B_0, L_0) \qquad (A1.16)$$

Appendix B: Algorithm and Grid Search Code.

```
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([[2], [1], [0]])

#Data matrix
theta = np.empty(shape = (501, 501))

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

Qinv = np.linalg.inv(Q)

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

#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

u0 = LLA_to_ECEF(u0[0,0], u0[1,0])[:2]

#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
    noise = nf(2)
    f = dopler_shift(2, 1)
    for i in range(0, 501):
        for j in range(0, 501):
            g = dopler_shift(lat, long)
            g += noise
            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]])*0.01
            lat += 0.01
        lat = 0
        long += 0.01
    xyz = LLA_to_ECEF(coord[0,0], coord[1,0])
    return coord, xyz[:2]
```

```
#monte carlo simulation to find the accuracy of the data
def monte_carlo():
    sumation = 0
    L = 500
    for l in range(0, L):
        coord, xyz = grid_search()

        sumation += np.linalg.norm(xyz - u0)**2
        print(l)
    RMSE = np.sqrt((1/L)*sumation)
    print(RMSE)

#plotting stuff
def graph():
    coord = grid_search()
    tmin = np.min(theta)
    tmax = np.max(theta)
    levels = np.linspace(tmin, tmax, 100)
    feature_x = np.arange(0, 5.01, 0.01)
    feature_y = np.arange(0, 5.01, 0.01)

    [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()
```

This code is also in our Git hub in project.py.

https://github.com/Mangin4/ENEL445_Optimization