Nikeet Pandit

214118806

Advanced Optimal Estimation Theory and Applications

LE/GS/ESS 5430 3.0

Fall 2021

Professor Wang

Project: Fitting Line, Curve, or Surface by Using Least Squares

Table of Contents

Introductory Remarks

The GRACE-FO (Gravity Recovery and Climate Experiment Follow-On), the predecessor to the GRACE mission, flies in a LEO near-polar orbit in a twin formation. Each satellite is built identically and can be considered to be carbon copies of one another. In actuality, however, GRACE-FO-D (the trailing satellite) had problems with its accelerometer shortly after deployment and the JPL team turned it off. As a result, NASA's JPL uses a *transplant* method to simulate measurements for the satellite using GRACE-FO-C (leading) data.

The purpose of this space mission is used to study dynamic mass transfer events and mass distribution on Earth with the primary objective of providing monthly gravity field maps. Each satellite is separated on average spatially by 220 km or 26 seconds in time.

In my area of research, it is of great interest to find the closest approach the leading and trailing satellites. That is to say, it is imperative to find the closest approach of GRACE-FO-D given a fixed GRACE-FO-C in time and space. Again, GRACE-FO-D is referred to as the trailing satellite, while GRACE-FO-C is referred to as the leading satellite. In other words, it is necessary to find at which epoch GRACE-FO-D will have a minimum distance separation to GRACE-FO-C delta (t) seconds ago. Because GRACE-FO-C and GRACE-FO-D are kept to ~220 km separation, this is a fictitious scenario and is therefore only possible to calculate by post-processing of the data products. Finding the closest approach of the leading and trailing satellites in this fictious scenario is important to determine because it allows for calculation of the gravity gradient functionals from gravity measurements and position data alone. Currently, there is no space mission that is providing gravity gradient information about Earth.

The satellites are fitted with GPS receivers which provide precise position orbit determination. All state vectors (position and velocity information) are fitted with error information in the level 2 data products. All errors are given as formal errors, as a result they are theoretical error estimates based on modelling errors and statistical understanding of the nature of the measurement errors themselves. Errors are not provided with classical variances or standard deviations for each measurement because redundant measurements are not taken for which this statistical quantity can even be calculated. As a consequence, the theoretical estimates must be used. Further, cross correlation of measurement errors is not provided. Although cross correlation for the measurement errors is not given, one could expect that the errors in the x, y, and z would be cross correlated. This could be due to the geometric interpretation of the pseudo-range equation for which positions are estimated using GPS receivers, in addition to correctional terms for the model (for e.g., ionospheric) which should also display positional dependency and then would likely propagate into the positional estimates (hence cross correlation in state vector estimates).

State vectors are given in 1 Hz. Every second the satellites will travel about 7.5 km. Finding the closest approach at this sample rate results in a separation that is too large for my necessary application. As I try to find the closest approach between the two spacecraft, and make this separation as small as possible, the most obvious method was evidently to experiment with different methods of interpolation to increase the sampling rate of the given ephemeris using only the state vector information.

Although, when interpolating every point to increase the sampling rate of the provided state vectors, the line is passed through absolutely each provided measurement point. However, each measurement has error. As a result, perhaps this is not the most optimal and unbiased method to use when estimating the closest approach of the two satellites. As a result, this project looks at using geometric least squares and weighted least squares to fit curves to GRACE-FO position data.

Methodology and Modelling

Code Features

The utilities written for this project are provided as a series of functions that fulfill smaller operations. They are then subsequently called in sequence by higher order functions which than gives the user a function call that is very much encapsulated from any of the innerworkings of the code and with clear function inputs. The code, which is well documented and provides summaries for each of the executables, is provided in the appendix at the end of this report. The GitHub repository can also be found here: https://github.com/NikeetPandit/OptimalEstimators/blob/main/OptimalEstimatorProject1_Routines.py

Since the project is: Fit curve/surface/line by using least squares, it felt intuitive to implement a self-written code package that can fit any of the three geometrical surfaces based on a user selection to the selected dataset. With this in mind, the code can fit a line, curve (polynomial of degree 3), sinusoid (ellipse in 3D), or plane based on the user selection.

The utilities should work for any dataset but is untested and as a result may require some manipulation to the code to work as expected. As a consequence, the code is tailored to work with GRACE-FO level 2 GPS position and velocity data products. Data products may directly be download from the JPL PODAAC (Physical Oceanography Distributed Active Archive Centre). From here, data files may be extracted into Panda data-frames or NumPy arrays to be parsed into functions. Although I have written my own utilities to perform these functions, they are not in my public repository at this time.

The user, in addition to being able to select the geometry to fit the dataset, can also choose to perform least squares fit, or weighted least squares fit. If the user selects to perform weighted least squares fit, there are 6 different options that the user can decide from. Although, it must be noted that the weighted least squares applications are very experimental. For this dataset and associated provided errors, as will be discussed later on in the report, the weighted least squares fit is very unreliable and poor. The methodology for the weighted least squares fit will be discussed in the next section.

In addition to the weighted least squares fit, the code also provides built-in error analysis in the high level (encapsulated) function calls. Specifically, uncertainty in estimated parameters, uncertainty in fit, residuals, chi-square for estimate goodness of fit are provided. All uncertainties are calculated based on data product errors and are also calculated where inputted errors into the formula are taken to be the least-square fit residuals. As will be discussed, the least-squares residuals are too large in this application for the error analysis using the least-squares residuals to be any of use. However, perhaps for another dataset that I (or someone else) will use, it might be helpful.

Finally, the code provides all necessary plotting routines with adequate labelling (titles, axes, legends, color coding). Firstly, a function call can be called which will plot the original (and unfitted) data so the user may inspect the geometry of the data to be fit. I felt this was absolutely necessary to include because as this is a geometrical least squares application, knowledge of the

geometry of the actual dataset which is to be fit in a geometric sense is absolutely essential. After the least squares operation is performed, plots with the original data and the least-squares fit are provided for user inspected. The residuals of the LS fit are also plotted. If extrapolation is required, labelling in the plots with reflect that the LS fit is based on extrapolation. Since the state vectors are still known where the LS is extrapolating to (all post-processing of data), error analysis with the all the above statistics, including plots of the residuals are all provided where the known state vectors that LS extrapolating to is taken to be the "known and true" values.

Code Methodology

LS Geometrical Fit

In this section, the code methodology and math that drives the code features will be explored. To fit a geometry using least-squares, based on the user selection, the normal equations are used. In this application, the independent variable is time while the explanatory variables being position in x, position in y, and position z. Since the coefficients which are estimated for in the least-squares process are solved for simultaneously, this is an application of multiple regression. The normal equation is shown here.

$$A^T Ax = A^T b$$

In the above formula: (1) **A** is the matrix of coefficients; (2) **x** is the parameter vector; (3) **b** is the parameter vector. In least-squares fitting, the process is about finding the coefficients of the equation which minimize the sum of the squares. In other words, the problem can be restated to say that it is to find the coefficients which maximize the probability that estimates actually occur[1]. This strikes similarly to the maximum likelihood estimation criterion. As a result, you can say if the measurements and its errors are normally distributed, there is no practical difference in the optimal estimate when using a least-squares criterion and a maximum likelihood estimation.

Since this is geometric least squares, the coefficients for the equations are simply the coefficients which describe the geometry for a line, surface, polynomial, or ellipsoid in 3D. Since this is a minimization problem, the LS estimate for the coefficients simply becomes an optimization problem which relies heavily on calculus. The whole process can greatly be simplified using linear algebra and more specifically the normal equations.

As an example, the least-squares multiple regression method for fitting an ellipse to the GRACE-FO ephemeris data will be presented, using the normal equations. It must be noted that the methodology for this process is mostly unchanged, regardless of the surface or geometry except evidently the coefficients to be estimated on based on the equation describing the geometrical surface and the resulting matrix of coefficients. As a result, rigorous attention placed in ensuring matrix dimensions and the associated matrix of coefficients are correct. Further, careful attention must be placed to ensure there is not a misinterpretation of a coefficient where one could falsely believe one coefficient to mean another estimated coefficient.

The ellipse is selected in particular, as one of the parameters (the fundamental frequency) is not a parameter which is estimated from the least squares process geometrical fit alone. The parametric equation for an ellipse in 3D is given below:

$$y = A + B * cos(\omega t) + C * cos(\omega t)$$

In the least squares geometrical fit, the coefficients to be estimated using the normal equations are A, B, and C. The omega parameter is a function cyclic frequency which governs the frequency of the cyclic data. This parameter can be estimated using LSSA (least-squares spectral analysis) or Fourier analysis which allow you to spectrally decompose a time series. In my software package, I call on the LSSA written by Professor Ghaderpour to perform the spectral analysis required to estimate the omega parameter necessary to construct the matrix of coefficients.

The LSSA, as an executable function, requires a time series, a bandwidth to investigate, and (if applicable) any trends to be removed from the series. The LSSA can be viewed, when compared to Fourier, as a more general spectral analysis tool as it performs spectral analysis even in the presence of trends in the series (linear, quadratic etc.) and allows for gaps in the data, which Fourier cannot[2]. As a result, LSSA is the preferred method. The LSSA will fit, in the least squares sense, a series of cosine and sinewaves with a series of varying frequencies which is based on the input bandwidth. The sinusoidal base function with the best fit then as a consequence will constitute most of the spectral energy within in the series. From this process, a spectrogram is constructed and given the form of an array based on the inputted bandwidth which governs the range of sinusoidal frequencies to be projected onto the series.

For the bandwidth, I plotted the measurements in the x, y, z and saw there was a clearly dominating periodicity in each of the 3 measurement components. Then simply, I set the input bandwidth in the LSSA call function to include this range for the spectral estimation. Since the parametric equation only takes one omega parameter, I perform individual estimates of the x, y, and z and then averaged them for the parametric equation. However, in all the tests that were run the dominant periodicity was unchanged for x, y, or z position measurements.

After the omega parameter is estimated, the LS fitting for the ellipse can be performed. An example code script written in MATLAB is appended to demonstrate this process. Although the code package is written in Python, MATLAB has a much more simplified process for performing linear algebra operations, in terms of its syntax, and as a consequence is much easier to follow. In the Python code package, all equations and matrix of coefficients are automatedly designed based on user inputs and data products.

```
Pos = [Data(Time,1),Data(Time,2),Data(Time,3)]; %GRACE-FO Position Data
Time = Time-Time(1,1);          %GRACE-FO Time Data (set to 0)
Col1 = repelem(1,length(Time))'; %Column 1 referring to coefficient A
Col2 = cos(Time*2*pi*Omega);     %Column 2 referring to coefficient B
Col3 = sin(Time*2*pi*Omega);     %Column 3 referring to coefficient C
Omega = 0.0001761072878065996;   %Omega Parameter estimated using LSSA
A = [Col1, Col2, Col3];          %Matrix of Coefficients
Fit = inv(transpose(A)*A)*transpose(A)*Pos; %Normal Equations Solving for Coefficients
```

Weighted Least Squares

As indicated in the geometrical least squares section, this problem of fitting the positions in x, y, and z simultaneously is a problem of multiple regression due to the presence of multiple explanatory variables. Since this is a problem of multiple regression, performing the weighted least squares estimate like:

$$x = (A^T W A)^{-1} A^T W b$$

where **W** is the weight matrix is not intuitive. This is because the weight matrix, which is inversely proportional to the measurement errors, is not the same for the positions in x, y, or z. This is because the associated given errors in the data-products are not the same, so then by extension the associated weight matrices would not be the same either. To circumvent this, the observation series for each position measurement in x, y, and z were equalized to be an observation series of unit weight by transferring their individual position errors to the measurements themselves. Following the method, as presented in Prof. Wang Lecture[3].

$$\begin{pmatrix} L_1' \\ L_n' \end{pmatrix} = \begin{pmatrix} \sqrt{p_1} * L_1 \\ \sqrt{p_2} * L_2 \end{pmatrix} \equiv \begin{pmatrix} L_1' \\ L_n' \end{pmatrix} = \begin{pmatrix} p_1' = 1 \\ p_n' = 1 \end{pmatrix}$$

There is a routine that calls in to perform this function and the comments for the code are all provided in the appendix. As a future patch to this code, there is a research paper that performs multiple regression with correlated weights which was written in 2014 by B. Li. The link to the paper can be found here: https://doi.org/10.1179/1752270615Y.0000000006

The weighted least squares results, as will be discussed in the following section, is very unreliable for this dataset and processes. As a result, six different options were experimented with. Specifically, different methods for estimation of the variance factor were looked at. The user then has the option to explore each of the six different options when running their experiment. The estimation of follows, as presented in Prof. Wang Lecture[4]

$$\hat{\sigma}^2 = \frac{[p\Delta\Delta]}{n}$$

where sigma hat is the estimated variance of unit weight, delta are the measurement errors, and p is the associated weight matrix. The measurement weights are always taken to be 1. This was done because when performing the weighted least squares and transferring the weight to the observations, the results were very unreliable. It was thought that this could be attributed to a poor estimation of the variance factor. So, in this estimation, it was hypothesized that this variance factor estimation process would subsequently be better off to not include weights, and then the weighted least squares with this more appropriate variance factor would yield better results. In the following section, it will be defined what "better" is to signify with concrete examples. In practice, unfortunately, this was not the case.

In retrospect, however, it would have been interesting to try performing recursive weighted least squares where the variance estimation was also performed using weighted least squares. At this

moment, however, this has not been experimented with, nor been built into the functionality. However, it could be extended very quickly for future work.

The 6 different options for the weighted least-squares are as follows:

1. Inverse of formal error is taken as weight matrix and variance factor is given as 1

2. Same as option 1, except z measurements are not weighted. This is because formal errors in the z direction do not follow a normal distribution.

3. Variance factor is estimated using formal errors and weights are taken to be P = 1 using equation for variance factor estimation.

4. Same as option 3, except z measurements are not weighted.

5. Least squares fit is performed and the residuals from this fit are then taken as the errors for the variance factor estimation where, subsequently, a weighted least squares fit is performed with these variance factor estimates.

6. An attempt to introduce artificial correlation to the measurement errors, which is not provided as discussed in the Introductory Remarks section, was performed. In this method, variance factor estimation was performed by taking weighted averages of each of the individual position measurements.

Extrapolation

The least squares process estimates parameters. Since parameters do not change with time, they are static and thus the extrapolation process becomes the task of using the same equation with the same coefficients that were estimated at a specific epoch t0, and propagating it to epoch t (where t>t0) by extending the time vector to t. The error analysis for the extrapolation functions is given by extrapolating the data from t0 to t and then comparing with the real data from t to t0. This can be performed because there is true data from the database until t provided in the database.

Implementation and Solution

High-Level Function Calls

A quick description of the higher order user function calls should be presented to show the functionality of the implementation and solution. It must be noted that all the software to perform the least-squares regression and error analysis is written mathematically and referenced using literature. In other words, no other software (open-sourced or otherwise) is used to perform the regression or error analysis.

```
Hours = 0.5                                          #Hours of data to fit
Hours_Extrap = 24                                    #Hours of data to extrapolate
[Ind,Exp] = OE.SelectDataProd(DataProd,ID,Hours)     #Formating data based on
selection
OE.PlotData(Ind,Exp)                                 #Plotting so user can select
geometry to fit
```

```
OptionToSet = 'Ellipse'                                #User selection of geometry
(Line / Ellipse/ Polynomial)
P = [0,3] #The first entry is to use weight and the second entry is the weight
type (0 for no and 1 for yes)
[Fit,Type,Omega,Equations,Residuals,Variances,ChiSqr,Fit_Uncertainty] =
OE.LSFittoData(Ind,Exp,P,ID,OptionToSet,Hours) #Fits data, Plots, Error Analysis
[Residuals1,Variances,ChiSqr1,Fit_Uncertainty1] =
OE.ExtrapolateAndPlot(DataProd,Ind,ID,Fit,Type,Omega,Hours,Hours_Extrap)#Extrapol
ates data, Plots, Error Analysis
```

In this code block, which shows all the higher-level functionality that will allow the user to interact with the software, there are a few things to note. Specifically, the user must denote the hours of the data to fit and the hours to extrapolate to (if applicable). The OE.SelectDataProduct() function which outputs the independent and explanatory variables in a specific matrix format calls on functions which are not open source at this time, as they format GRACE-FO JPL data products from the PODAAC database. Upon completion of the research thesis, they will be released. After the independent and explanatory variables are given in the proper formatting (this is why it has been mentioned in this report that this software package is optimized for GRACE-FO dataset but could be used with any dataset given a little manipulation), the OE.PlotData() function is called to plot the data with applicable labelling.

After this function is called, the user may enter in the OptionToSet to denote the selection of the geometry to fit using the LS fitting routines. In P = [X, Y], this deals with the weighted least squares fitting options. X denotes 1 for yes and 0 for no (regular least squares) and Y denotes a number between 0 and 5 for 6 different options weighted least squares methods.

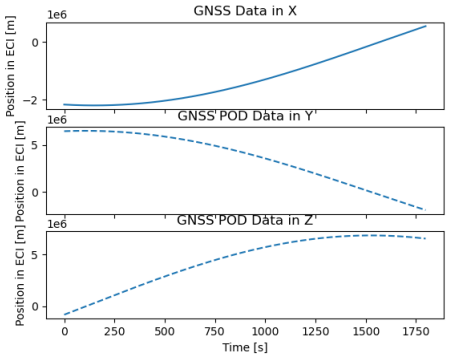
In the OE.LSFitToData(), it fits the data, plots, and provides the error analysis. Similarly, the OE.ExtrapolateAndPlot(), will instead extrapolate the data, but it will also plot and provide error analysis.

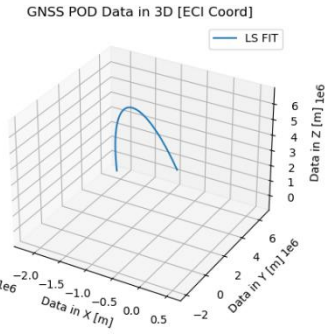
LS Fitting Examples and Extrapolation

In all the following examples, data is fit to GRACE-FO-C (the leading satellite). Data for the state vectors is given in the ECI (Earth-centered inertial) frame. Data is taken from January 17, 2021, and onwards where needed.

Test 1: 0.5 hours of data fitting with polynomial (quadratic)

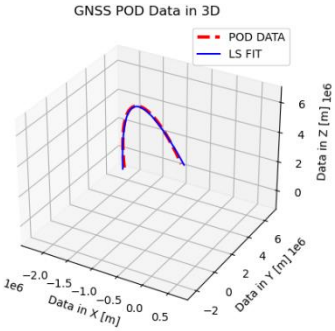
Original Data

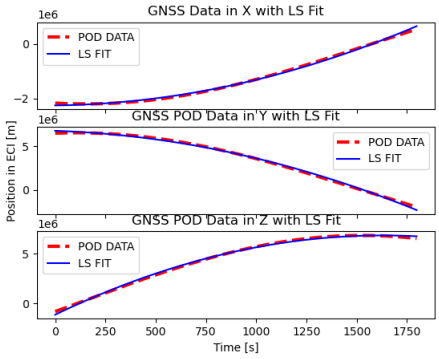




Based on the data, it looks like polynomials of degree 3 would fit quite well. The geometric least-squares (non-weighted) is then performed.

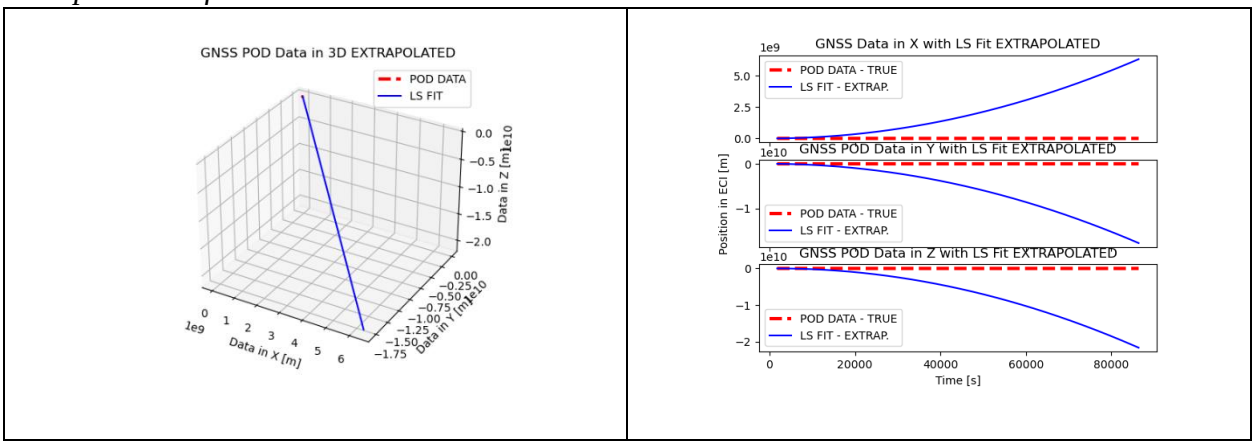
LS Fit to Original Data





As can be seen in Test 1 plots, the LS fit and plotting functions seem to be fitting. However, when the GNSS Data in X, Y, Z with LS fit plots are zoomed in, it is visually identifiable to see their errors in the LS fit are quite large. Discussion about the errors will be given in the next section.

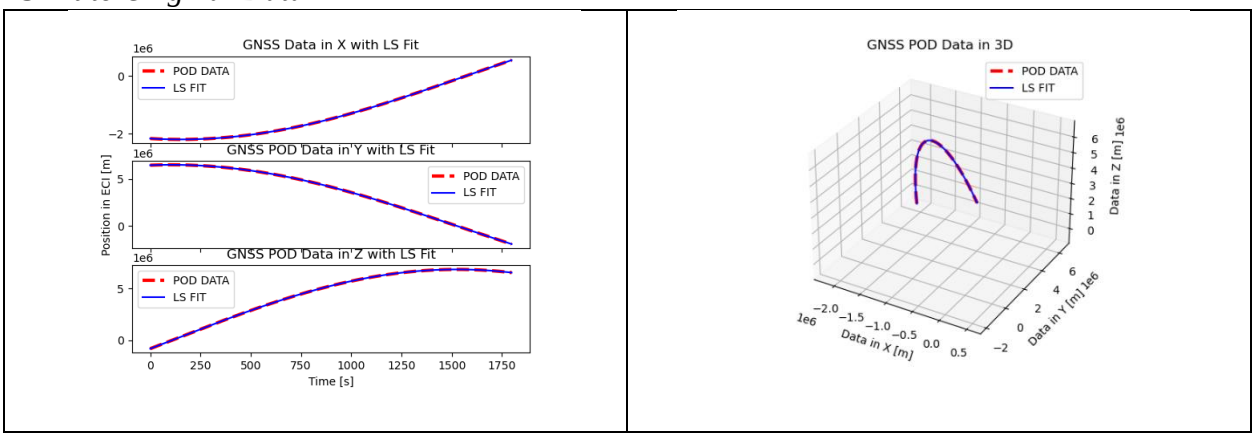
Extrapolation of Data 24 hours later



Clearly, from the extrapolation of data 23.5 hours later plot, the extrapolation in neither x, y, nor z is not even remotely tracking the true GNSS data as given in the database. As a consequence, the same test will be performed where the user will select an ellipse to see the difference in the LS fit to the original data and the extrapolation of data 23.5 hours later.

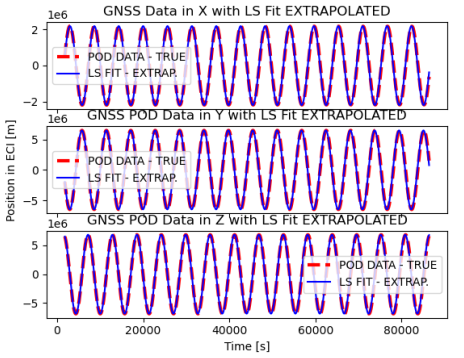
Test 2: 0.5 hours of data fitting with sinusoid (ellipse in 3D)

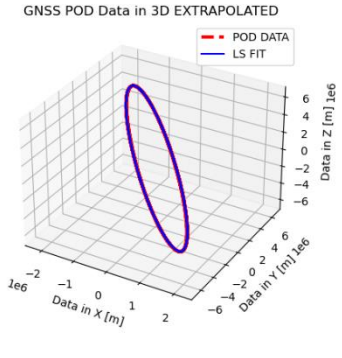
LS Fit to Original Data



When compared to Test 1, the ellipse LS fit data tracks more closely than the polynomial. It is intuitive that the ellipse will extrapolate much more accurately than the polynomial due to the estimated omega parameter which will be able to extrapolate the periodic position data of GRACE-FO which the polynomial does not consider.

Extrapolation of Data 24 hours later

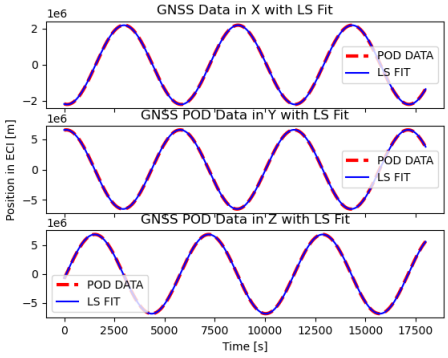


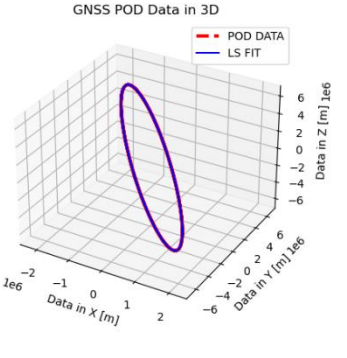


As expected, the extrapolation when using the ellipse parametric equation tracks the true POD data visually very well relative to the polynomial. As indicated, this is due to the estimated omega parameter and the periodic nature of the positional measurements for the GRACE-FO satellite. It is then interesting to test to how the results of the LS fit, and error analysis of the extrapolation will be changed when the more data is fit using the LS process, while the hours to extrapolate to remains unchanged.

Test 3: 5 hours of data fitting with sinusoid (ellipse in 3D)

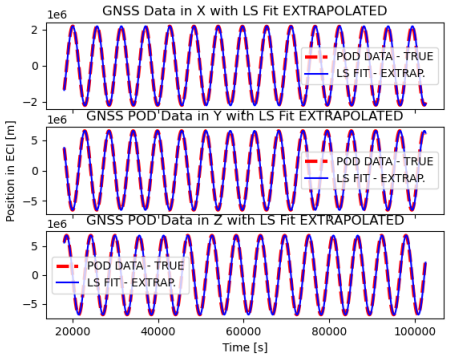
LS Fit to Original Data

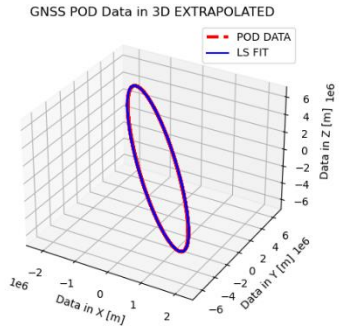




Visually, even when inspecting by zooming in, the LS fit with an ellipse tracks quite well for 5 hours of data fitting. More insight about the fit can only be offered in the error analysis.

Extrapolation of Data 24 hours later



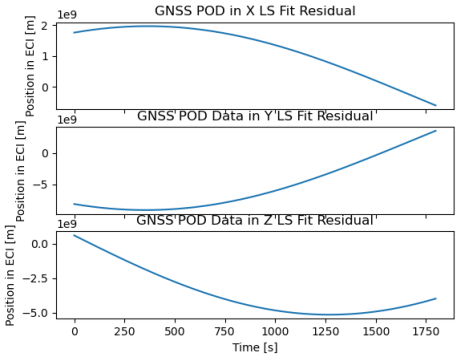


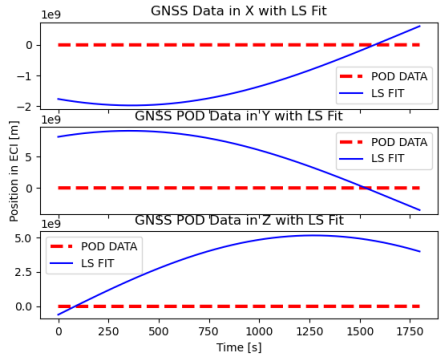
Extrapolation still visually tracks well (even when zooming in) when extrapolating from 5 hours of data to 24 hours later. More insight about the fit can only be offered in the error analysis.

WLS Examples

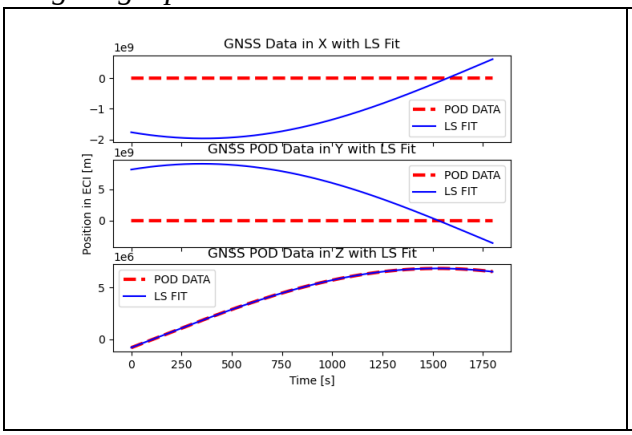
In all the following examples, data is fit to GRACE-FO-C (the leading satellite). Data for the state vectors is given in the ECI (Earth-centered inertial) frame. Data is taken from January 17, 2021, and onwards where needed. For the WLS examples, only 0.5 hours of data is used, and an ellipse is selected as the geometry. The weighting options correspond to the six different options, as presented in the Weighted Least Squares of this report. The six different weighting options with their plots are provided below, followed by a brief discussion on the results.

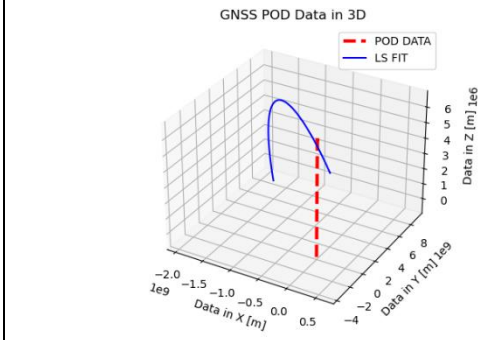
Weighting Option 1



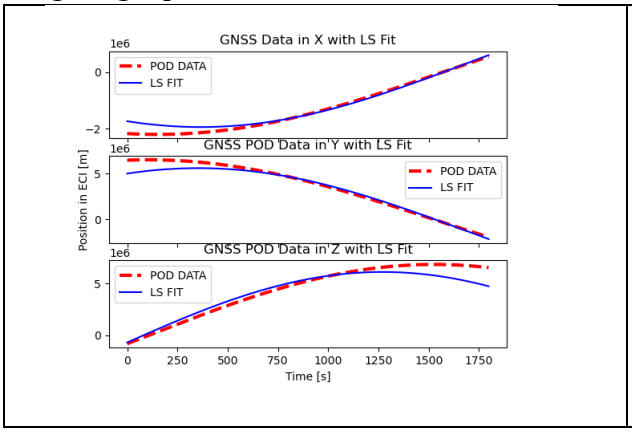


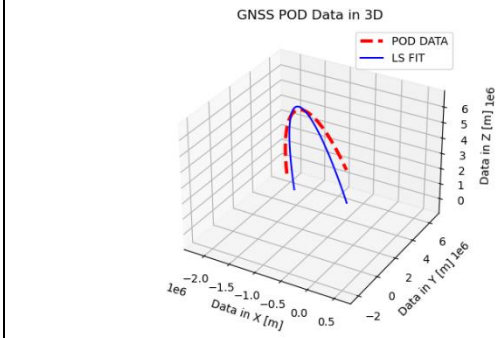
Weighting Option 2



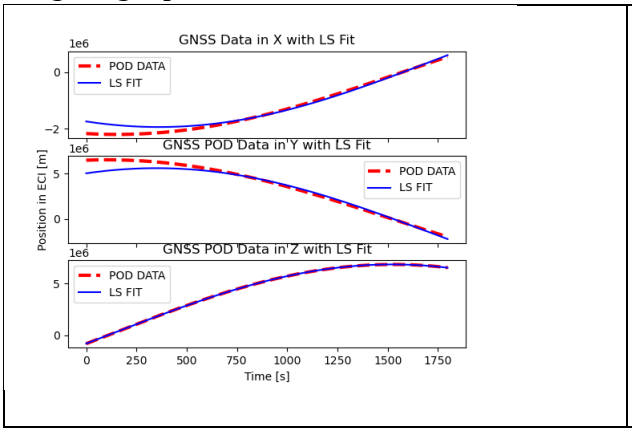


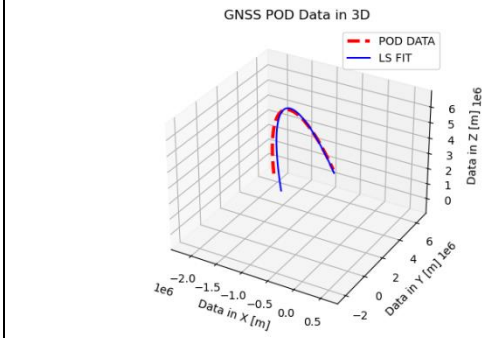
Weighting Option 3



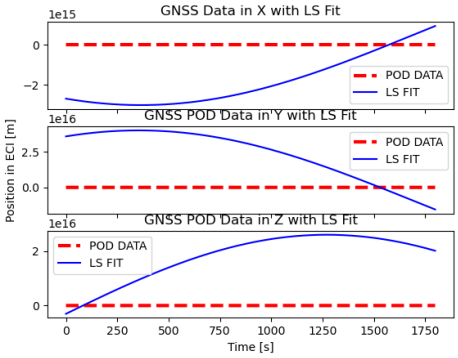


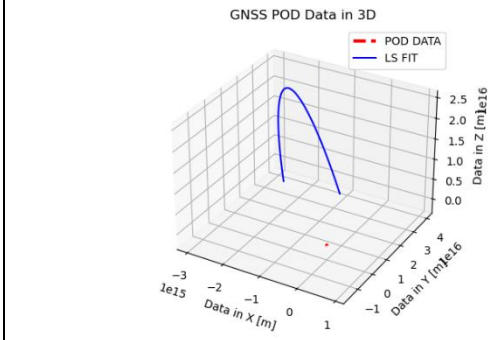
Weighting Option 4



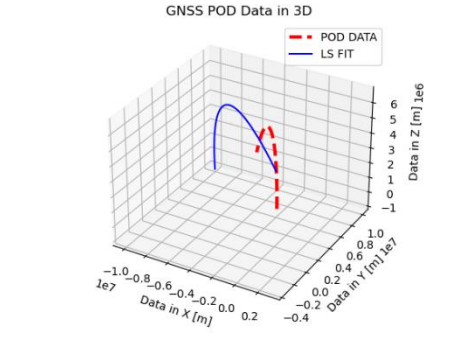


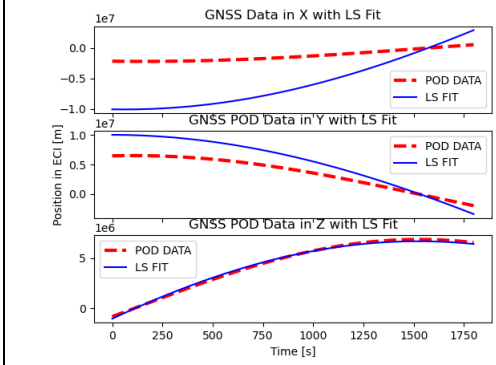
Weighting Option 5





Weighting Option 6





Visually from these plots for all the different weighting options, it is clearly indicated that something is not working as expected. This is demonstrated by the fact that the measurement errors are incredibly small with respect to the values (in magnitude) of the measurements themselves. That is to say, the measurements in the ECI coordinate system are 10^6 meters while the measurement errors are less than 1cm. This means the measurements are incredibly accurate with respect to the coordinate system they are provided in.

As a result, I expected the WLS to not have much (if any) effect on the LS fit of the dataset. Clearly, this was not the case. Perhaps, this can be attributed to the fact that the measurement errors vary up to a factor of ~2 which then results in a great change to the weighting scheme when the weight is transferred to the observations to make the associated weights of P = 1.

If you look at weighting option 1, versus weighting option 2, you may notice that measurements in the Z track very well for weighting option 2. This is only because the Z measurements are not weighted for this scheme to see the difference. You see this effect also when comparing option 3 and 4. Interestingly, in weighting option 6 which attempts to introduce artificial correlation, the data in Z tracks visually quite well. Although, no explanation can be given as to why this occurs as this was purely based on tria; and error to see if there was a set of weights that would track in

x, y, and z positions simultaneously. If further experiments were to have been tried, perhaps transforming all measurements to the satellite reference frame and then performing the WLS would have yielded results with more intuitive sense.

Error Analysis

Methodology

As mentioned, several statistics are given to provide an error analysis of the least squares fit.

1. Firstly, the residuals of the least squares fit are given. The equation for the residuals is:

$$Residuals = [y_i - y(x_i, a_o, b_o, c_o \dots)^2]$$

where the square brackets denote a sum. Evidently, the smaller the residuals of the least squares fit of a geometry to a set of data points denotes a better approximation of the geometry to the dataset.

2. The uncertainty in the fit is provided by an equation similar to that of the residual equation.

$$\sigma_y^2 = \frac{1}{N-n} * [y_i - y(x_i, a_o, b_o, c_o \dots)^2]$$

where N is the number of measurements and n is the number of coefficients to be estimated. The previous two equations are given in reference [6].

3. The uncertainty in the estimated coefficients is given by the propagation of errors as discussed in reference [2]. If there is no cross correlation (recall measurements are not given cross correlation estimates and as a result are assumed to be zero, although in practice this is likely not the case), the equation can be given by:

$$\sigma_{\hat{xyz}}^2 = A * A * \sigma_{xyz}^2$$

In this equation, A is the matrix of coefficients and σ_{xyz}^2 is the vector corresponding to the variances of the errors of the measurements which is calculated given the measurement errors. In another method of calculating the coefficient uncertainty, measurement errors are taken to be the variances of the least squares fit in the x, y, and z. The $\sigma_{\hat{xyz}}^2$ correspond to the uncertainty of the estimated parameters in the least squares process. Both are provided as outputs in the code in a list.

It must be noted that, except for equation 3, equation 1 and equation 2 are not given for multiple regression. Simply, the notation must be denoted to extend the dimensions and careful attention must be placed to ensure matrix dimensions are as desired to compute the intended results.

4. The Chi testing for "Goodness of Fit", as given by reference [6] is given by

$$\gamma^2 = \left[\frac{1}{\sigma_i} * y\left(x_{i,} a, b, c\right)\right]^2 \approx N - n$$

Therefore, a good fit will have

$$\gamma^2{}_{good} = \frac{\gamma^2}{N-n} = 1$$

Errors in this statistic are taken from the position errors as provided in the data products.

5. Finally, all the residuals for all the plots are given to give the user a visual understanding of how the residuals may change with time.

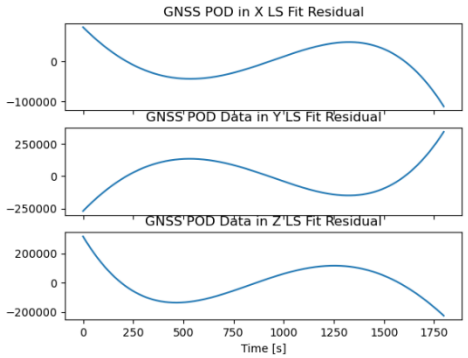
Error Modelling

As concluded in the WLS analysis, there was significant problems in the results. Therefore, error modelling will not be provided for this section. However, for three tests presented in the LS fitting examples and extrapolation section, error analysis is performed. At the end, of a brief conclusion will also be provided.

Test 1: 0.5 hours of data fitting with polynomial (quadratic)

LS Fit to Original Data

Residuals



Visually, we can see that the residuals display a periodic trend and have a high magnitude.

Error Statistics

Uncertainties in x, y and z for LS fit		
σ_x	σ_y	σ_z
38000	119000	105000

Chi-Square Goodness of Fit Test in x, y and z for LS fit		
For X	For Y	For Z
1.64e15	4.06e16	9.2e15

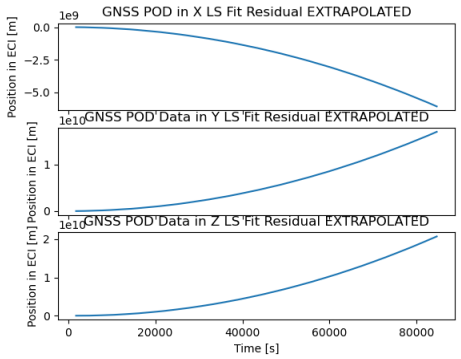
Uncertainties in Estimated Parameters: A, B, C (Errors taken as measurement errors)		
σ_A	σ_B	σ_C
7e-2	3.8e-4	1.2e-7

Uncertainties in Estimated Parameters: A, B, C (Errors taken to be residuals)		
σ_A	σ_B	σ_C
4.1e17	5.6e14	2.2e11

Clearly, the Chi-Square goodness of fit test indicates the fit is not good when fitting a polynomial to the dataset. The variances are high, and upon inspection of the residuals when compared to the associated measurements further shows the fit is very poor. This idea is further demonstrated that the uncertainty in the coefficients, when propagating the measurement errors, is very low (and very high when propagating measurement errors taken as the variance of the LS residuals).

Extrapolation of Data 24 hours later

Residuals



Error Statistics

Uncertainties in x, y and z for LS fit		
σ_x	σ_y	σ_z
2.9e9	8.0e9	9.7e9

Chi-Square Goodness of Fit Test in x, y and z for LS fit		
For X	For Y	For Z
1.4e25	1.15e26	1.0e26

Uncertainties in Estimated Parameters: A, B, C (Errors taken as measurement errors)		
σ_A	σ_B	σ_C
0	0	0

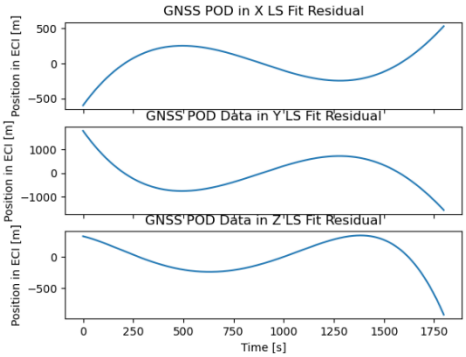
Uncertainties in Estimated Parameters: A, B, C (Errors taken to be residuals)		
σ_A	σ_B	σ_C
4.45e17	6.2e14	2.4e11

Clearly, as indicated by the uncertainties in the fit, and the uncertainties in the estimated parameters where errors are taken to be the measurement residuals, extrapolation when using a polynomial while the data follows a cyclic trend will not yield sensical estimates. Clearly, knowledge of the geometry and understanding of the system at hand (that is to say knowledge that position data of GRACE-FO in x, y, and z follows a cyclic and sinusoidal pattern) allows you to perform better estimates in terms of filtering and prediction.

Test 2: 0.5 hours of data fitting with sinusoid (ellipse in 3D)

LS Fit to Original Data

Residuals



Error Statistics

Uncertainties in x, y and z for LS fit		
σ_x	σ_y	σ_z
200	640	250

Chi-Square Goodness of Fit Test in x, y and z for LS fit		
For X	For Y	For Z
4.8e10	1.1e12	3.9e10

Uncertainties in Estimated Parameters: A, B, C (Errors taken as measurement errors)		
σ_A	σ_B	σ_C
0	0	0

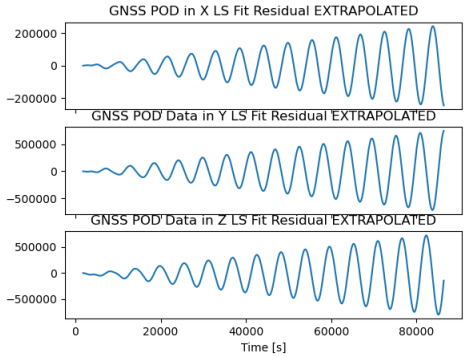
Uncertainties in Estimated Parameters: A, B, C (Errors taken to be residuals)		
σ_A	σ_B	σ_C
1.5e10	6.6e12	6.5e12

When fitting an ellipse to the dataset, the uncertainties are much lower than when compared to the polynomial. The residuals continue to follow cyclic behavior. Again, the Chi-Square for goodness of fit denotes this a poor fit. This is further augmented by the high uncertainty in the variances of

the estimated parameters relative to the uncertainties when propagating the measurement errors themselves. As this point applies for all the following tests too, it will not be repeated.

Extrapolation of Data 24 hours later

Residuals



Error Statistics

Uncertainties in x, y and z for LS fit		
σ_x	σ_y	σ_z
102700	304000	319600

Chi-Square Goodness of Fit Test in x, y and z for LS fit		
For X	For Y	For Z
2e16	1.9e17	1.4e17

Uncertainties in Estimated Parameters: A, B, C (Errors taken as measurement errors)		
σ_A	σ_B	σ_C
0	0	0

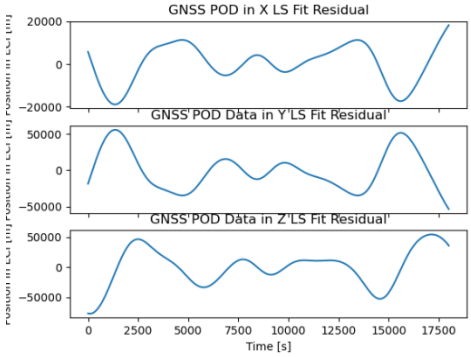
Uncertainties in Estimated Parameters: A, B, C (Errors taken to be residuals)		
σ_A	σ_B	σ_C
1.2e10	5.4e12	5.4e12

When extrapolating the data 24 hours in the future, when the data segment itself is only 0.5 hours, we see a growing sinusoidal trend in the residuals which seems to be continually get excited every orbit (one orbit takes about 1.6 hours). Evidently, the error statistics show that when extrapolating the position measurements using geometry alone and comparing it to the true POD data, the errors are large, and the fit is poor.

Test 3: 5 hours of data fitting with sinusoid (ellipse in 3D)

LS Fit to Original Data

Residuals



Error Statistics

Uncertainties in x, y and z for LS fit		
σ_x	σ_y	σ_z
8750	26080	30500

Chi-Square Goodness of Fit Test in x, y and z for LS fit		
For X	For Y	For Z
1.1e14	1.5e15	1.0e15

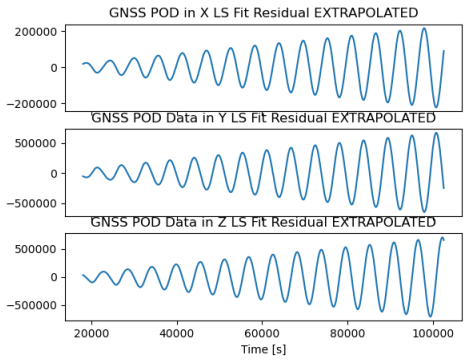
Uncertainties in Estimated Parameters: A, B, C (Errors taken as measurement errors)		
σ_A	σ_B	σ_C
0	0	0

Uncertainties in Estimated Parameters: A, B, C (Errors taken to be residuals)		
σ_A	σ_B	σ_C
3.3e13	8.2e15	8.2e15

The uncertainty in the fit, when the ellipse is fit to a 5-hour data window, is larger by a factor of 40 when compared to the 0.5-hour window. This is likely (partially) attributed to the fact that one complete orbit is performed in 1.6 hours. When you zoom in the graph, there are separate lines denoting each orbit because each ellipse is not dynamically fixed in space and is constantly being perturbed. Even as an orbit progresses before completion of a full orbit, it is constantly being perturbed and deviated from a pure Keplerian orbit. As the geometrical fit is not considering these perturbations, it is therefore intuitive for the errors in the fit to be large, and the errors to be even larger when there is a larger data window to be fit to (more time for orbit to deviate from pure Keplerian orbit, or ellipse). This idea is further augmented by the fact that GPS receivers aboard GRACE-FO are incredibly precise.

Extrapolation of Data 24 hours later

Residuals



Error Statistics

Uncertainties in x, y and z for LS fit		
σ_x	σ_y	σ_z
122300	361300	382000

Chi-Square Goodness of Fit Test in x, y and z for LS fit		
For X	For Y	For Z
2.8e16	2.8e18	1.9e17

Uncertainties in Estimated Parameters: A, B, C (Errors taken as measurement errors)		
σ_A	σ_B	σ_C
0	0	0

Uncertainties in Estimated Parameters: A, B, C (Errors taken to be residuals)		
σ_A	σ_B	σ_C
1.2e10	5.4e12	5.4e12

Interestingly, the error statistics when extrapolating 24 hours in the future from a 5-hour LS ellipse fit compared to 24 hours in the future from 0.5 hours fit is largely unchanged. However, the 0.5 hours LS fit based extrapolation shows slightly better extrapolation results, as seen in a slight reduction in the uncertainties and the Chi-Square goodness of fit. This may partially be attributed to the aforementioned point of using a pure geometrical fit for a perturbed and deviating from Keplerian orbit coupled with very high precision POD.

Conclusions

In this report, all the features of the software package were discussed in addition to the mathematical equations and logic which was used to build the package. The least squares geometrical fit, alongside an introduction to the ellipse fit in 3D using the parametric equation and normal equations were given. Further, a very brief introduction to the least-squares spectral analysis was also provided, in addition to a general review of how it was used to find the max fundamental frequency of the positional data in the context of estimating the omega parameter needed for the parametric ellipse equation. Further, the 6 different methods of the weighted least squares and its methodology were described.

In the implementation and solution sections, the high-level function calls were discussed which allows the user to interact will the original software package in an encapsulated way. A series of tests were provided which showed the LS fitting process and extrapolation. For these same tests, error analysis was provided. In addition to the error analysis, which encompassed providing error statistics, residual plots, and discussions, the methodology and references for the error statistics were also given.

In summary, it was demonstrated the WLS is extremely unreliable and causes a large change to the goodness of fit relative to the normal least-squares geometrical fit. This is because the measurement errors are incredibly small relative to the magnitude of the measurements themselves. As a result, the WLS was expected to not be overwhelmingly different than the LS geometrical fit. As seen, however, this was not the case. This was suspected that it could be to the coordinate system used (ECI). Also, it could be attributed to the fact that the measurement errors themselves do vary by up to a factor of 2, which then results in a great change to when these errors are transferred to the observation series to equalize the weights to be $P = 1$. In essence, every weighting scheme attempted resulted in residuals that were much too big relative to the magnitude to the measurement errors themselves when compared to the normal LS fit. Estimating the variance factor based on the measurement errors seemed to be the best option as it yielded the smallest residuals. However, the residuals and error statistics showed such a poor fit and as a result is rejected as well for the aforementioned reasons in this application.

Extrapolation is poor for the geometrical fit alone, especially when considering a how a dynamic physical model would improve the prediction. Even a basic SPG3 propagation model would be much more accurate to the true POD data than a geometric extrapolation alone, however, the geometrical fit of an ellipse provides a very rough estimate of the state vector at a future epoch without using any physical model. Perhaps, a Kalman filter which would include a physical model of the system would yield a great improvement to the project, as opposed to using geometry alone.

References

[1] John R. Taylor. An Introduction to Error Analysis 2ed. University Science Books. 1997.

[2] Ebrahim Ghaderpour. Least Squares Wavelet Analysis and Its Applications in Geodesy and Geophysics. York Space Institutional Repository. 2018.

[3] Jianguo Wang. Lecture 5: Error Analysis in Least Squares Method. Fall 2021.

[4] Jianguo Wang. Lecture 2: Random Error Theory and Propagation. Fall 2021.

[5] Error Analysis 2: Least Squares Fitting. URL: https://physicscourses.colorado.edu/phys4430/phys4430_fa18/Activities/ErrorAnalysis2.pdf

Appendix

```
''''
Optimal Estimator Project 1 - Fitting line/curve/surface by using Least Squares
by Nikeet Pandit
******************** ALL UTILITIES ARE GIVEN SUMMARIES DESCRIBING FUNCTIONALITY
********************

This module comprises of a series of executables that allows you to fit a
geometry to a dataset using LS and WLS
(1) line, curve (polynomial-quadractic), surface (plane), and sinusoid (or
ellipse in 3D)
(2) user must select this my setting OptionToSet variable which is -> Linear,
Polynomial, Ellipse
(3) all descriptions of routines are provided so that can easily be manipulated
(4) functions will work (may require some manipulation) for any dataset, but it
is optimized to work with GRACE-FO Ephemeris files
(5) module calls LSSA to estimate fundamental frequency of sine waves written by
Professor Ebrahim Ghaderpour
(6) module calls DataFormCallFunctions and DataFormat2 which read in the GRACE-FO
data files provided by JPL which I have previously written
(7) the plane fitting option is not included as an OptionToSet because it is not
necessary for this project, but can easily be if needed
(7) for a further application. It was added to show that it could easily be done
for my own learning.
(8) user must specifiy the DataProd using the DataFormatCallFunctions, the ID
(GRACE-FO identifier), the Hours of the dataset to fit
(8) and the weighted scheme which is a [1,2] list. The first element denotes if
weighted is used (1) or not (0). The second element
(9) denotes the type of weighting scheme to be used. For example to not weight
you would specify P as P = [0,0]. To weight
(9) and specify weighting scheme 3... you would specify P as P = [1,3]. More info
is given in SUMMARY OF WeightedObsTransfer
```

```
'''

# ------- IMPORTING LIBRARIES ------- #
import numpy as np
import DataFormatCallFunctions as DFF
import DataFormat2 as DF2
import matplotlib.pyplot as plt
import LSSA as LS
# ------- IMPORTING LIBRARIES ------- #


#----SUMMARY OF SelectDataProd(DataProd,ID,Hours)----
#SELECTS DATA PRODUCTS TO FIT
#SELECTS ID OF GRACE SATELLITE (IDENTIFIER) TO CONSTRUCT DATAFRAME PROVIDED A
GIVEN DATE-RANGE IN INPUTS.PY SUBMODULE
#SELECTS HOURS FOR FURTHER MANIPULATION TO GIVE USER CAPABABILITY TO DICTATE
DURATION TO WHICH DATA IS TO BE FITTED TO [IN HOURS]... CAN BE MANIPULATED EASILY
#RETURNS IND/EXP VARIABLES WHICH ARE INPUTS TO MANY OF THESE FUNCTIONS
def SelectDataProd(DataProd,ID,Hours):
    Index = int(Hours*3600-1)
    DataBase = list(map(DF2.CreateDataFrame,DataProd,ID))
    Pos = DataBase[0].iloc[0:Index,1:4].to_numpy()
    Time = DataBase[0].iloc[:Index,0].to_numpy()
    Time = Time[:]-Time[0]
    return Time,Pos


#----SUMMARY of PlotData(DataProd,ID,Hours)----
#THIS FUNCTION SERVES AS A PLOTTING ROUTINE TO PLOT X,Y,Z POSITION DATA IN A
SUBPLOT IN RESPECTIVE DIMENSIONS
#ALSO IT WILL PROVIDE ANOTHER PLOT IN 3-DIMENSIONS
#SINCE LS CURVE FITTING REQUIRES KNOWLEDGE OF GEOMETRY (OR GENERAL SURFACE) WHICH
IS TO FIT TO DATA...
#THIS IS AN INCREDIBLY HELPFUL UTILITY TO DECIDE WHICH OF THE
LINE/PLANE/ELLIPSE/POLYNOMIAL IS SUFFICIENT TO FIT DATA
#OR IF MORE GEOMETRY IS NEEDED TO FIT
#THIS GIVES USER ABILITY TO EASILY MANIPULATE FUNCTIONS IN THIS CODE TO FIT
CURVE/LINE/SURFACE TO ANY DATASET BY MANIPULATING THESE BLOCKS OF CODE
def PlotData(Ind,Exp):
    fig, axs = plt.subplots(3)
    axs[0].plot(Ind,Exp[:,0])
    axs[0].set_title('GNSS Data in X')
    axs[1].plot(Ind,Exp[:,1],'--')
    axs[1].set_title('GNSS POD Data in Y')
```

```
    axs[2].plot(Ind,Exp[:,2],'--')
    axs[2].set_title('GNSS POD Data in Z')
    for ax in axs.flat:
          ax.set(xlabel='Time [s]', ylabel = 'Position in ECI [m]')
    for ax in axs.flat:
        ax.label_outer()
    fig.show()
    fig1 = plt.figure()
    ax1 = plt.axes(projection='3d')
    ax1.plot(Exp[:,0],Exp[:,1],Exp[:,2],label = 'LS FIT')
    ax1.set_title('GNSS POD Data in 3D [ECI Coord]')
    ax1.legend()
    ax1.set(xlabel='Data in X [m]',ylabel='Data in Y [m]',zlabel='Data in Z [m]')
    fig1.show()
    return print("Examine Geometry and Select Curve to Fit...\n\n")


#----SUMMARY OF LSFitToData(Ind,Exp,P,ID,OptiontoSet)----
#Ind -> Independent Variable, Exp->Explanatory Variable, P-> Weighted LS Method
Identifier, ID->Exp. Above, OptiontoSet->Geometry Selection for LS Fit
#IND/EXP ARE OUTPUTTED OUT OF SelectDataProd(DataProd,ID,Hours) FUNCTION
#P VAR (IDENTIFIER FOR METHOD OF WLS REGRESSION) ARE DISCUSSED IN
SelectCurveTypeAndFit(Ind,Exp,P,ID,OptiontoSet)
#OptiontoSet ALLOWS FOR: (1) ELLIPSE [USES SINUSOIDAL BASE FUNCTIONS [COS AND
SINE] + CONSTANT]... FUNDAMENTAL FREQUENCY PARAMETER IS EST. USING LSSA IN NESTED
FUNC
#(2) LINEAR (3) POLYNOMIAL - USES QUADRACTIC BUT CAN EXTEND EASILY TO GENERAL
CASE (4) PLANE FUNCTION IS FitPlaneType(Ind,Exp) AND WOULD NEED TO BE ADDED TO BE
USED AS AN OPTION
#IT IS NOT USED AN OPTION CURRENTLY BECAUSE IT IS NOT NEEDED FOR THIS APPLICATION
OF FITTING DATA TO POD DATA FOR GRACE-FO
#ANY OTHER FUNCTION (LINEAR) CAN EASILY BE ADDED FOLLOWING THE METHOD OF USING
THE NORMAL EQUATIONS AS NEEDED
#-----
#IN FUNCTION -> CALLS NESTED FUNCTION
SelectCurveTypeAndFit(Ind,Exp,P,ID,OptiontoSet) WHICH DETERMINES COEFFICIENTS
ESTIMATED FROM LS PROCESS
#IT ALSO WILL RETURN OMEGA IF APPLICABLE (FOR ELLIPSE) AND WILL RETURN 999
OTHERWISE
#IT ALSO RETURNS THE TYPE (ELLIPSE/LINE/POLYNOMIAL) TO CONDITIONALLY CONSTRUCT
EQUATIONS REQURIED FOR PLOTTING
#ALSO RESIDUALS ARE CALCULATED AND PLOTTED. DATA SEPERATED INTO X,Y,Z COMPONENTS
AND PLOTTED IN 3D ARE GIVEN WITH RESIDUALS AND LS BEST FITS
#ALL PLOTS ARE LABELLED AS NECESSARY FOR THE APPLICATION FOR FITTING
LINE/CURVE/SURFACE TO GRACE-FO DATA IN ECI COORD.
```

```
def LSFittoData(Ind,Exp,P,ID,OptiontoSet,Hours):
    #[Ind,Exp] = SelectDataProd(DataProd,ID,Hours)
    [Fit,Type,Omega] = SelectCurveTypeAndFit(Ind,Exp,P,ID,OptiontoSet)
    if Type == 'Ellipse':
        Eq0 = Fit[0,0] + Fit[1,0]*np.cos(2*np.pi*Omega*Ind) +
Fit[2,0]*np.sin(2*np.pi*Omega*Ind)
        Eq1 = Fit[0,1] + Fit[1,1]*np.cos(2*np.pi*Omega*Ind) +
Fit[2,1]*np.sin(2*np.pi*Omega*Ind)
        Eq2 = Fit[0,2] + Fit[1,2]*np.cos(2*np.pi*Omega*Ind) +
Fit[2,2]*np.sin(2*np.pi*Omega*Ind)

        Eq0_Residual = Exp[:,0]-Eq0
        Eq1_Residaul = Exp[:,1]-Eq1
        Eq2_Residual = Exp[:,2]-Eq2

    elif Type == 'Linear':
        Eq0 = Fit[0,0]+Fit[1,0]*Ind
        Eq1 = Fit[0,1]+Fit[1,1]*Ind
        Eq2 = Fit[0,2]+Fit[1,2]*Ind

        Eq0_Residual = Exp[:,0]-Eq0
        Eq1_Residaul = Exp[:,1]-Eq1
        Eq2_Residual = Exp[:,2]-Eq2

    elif Type == 'Polynomial':
        Eq0 = Fit[0,0]+Fit[1,0]*Ind+Fit[2,0]*Ind**2
        Eq1 = Fit[0,1]+Fit[1,1]*Ind+Fit[2,1]*Ind**2
        Eq2 = Fit[0,2]+Fit[1,2]*Ind+Fit[2,2]*Ind**2

        Eq0_Residual = Exp[:,0]-Eq0
        Eq1_Residaul = Exp[:,1]-Eq1
        Eq2_Residual = Exp[:,2]-Eq2

    fig, axs = plt.subplots(3)
    axs[0].plot(Ind,Exp[:,0],'r--',linewidth = 3,label='POD DATA')
    axs[0].plot(Ind,Eq0,'b',label='LS FIT')
    axs[0].set_title('GNSS Data in X with LS Fit')
    axs[1].plot(Ind,Exp[:,1],'r--',linewidth = 3,label='POD DATA')
    axs[1].plot(Ind,Eq1,'b',label='LS FIT')
    axs[1].set_title('GNSS POD Data in Y with LS Fit')
    axs[2].plot(Ind,Exp[:,2],'r--',linewidth = 3,label='POD DATA')
    axs[2].plot(Ind,Eq2,'b',label='LS FIT')
    axs[2].set_title('GNSS POD Data in Z with LS Fit')
    axs[1].set(ylabel = 'Position in ECI [m]')
```

```
    axs[2].set(xlabel = 'Time [s]')
    for ax in axs.flat:
        ax.label_outer()
    axs[0].legend()
    axs[1].legend()
    axs[2].legend()
    fig.show()
    fig1 = plt.figure()
    ax1 = plt.axes(projection='3d')
    ax1.plot(Exp[:,0],Exp[:,1],Exp[:,2],'r--',linewidth = 3,label='POD DATA')
    ax1.plot(Eq0,Eq1,Eq2,'b',label='LS FIT')
    ax1.set_title('GNSS POD Data in 3D')
    ax1.set(xlabel='Data in X [m]',ylabel='Data in Y [m]',zlabel='Data in Z [m]')
    ax1.legend()
    fig1.show()

    fig2, axs2 = plt.subplots(3)
    axs2[0].plot(Ind,Eq0_Residual)
    axs2[0].set_title('GNSS POD in X LS Fit Residual')
    axs2[1].plot(Ind,Eq1_Residaul)
    axs2[1].set_title('GNSS POD Data in Y LS Fit Residual')
    axs2[2].plot(Ind,Eq2_Residual)
    axs2[2].set_title('GNSS POD Data in Z LS Fit Residual')
    for ax in axs2.flat:
          ax.set(xlabel='Time [s]', ylabel = 'Position in ECI [m]')
    for ax in axs2.flat:
        ax.label_outer()
    fig2.show()

    Equations = [Eq0,Eq1,Eq2]
    Residuals = [Eq0_Residual,Eq1_Residaul,Eq2_Residual]

    Variances =
[SigmaExp(Eq0_Residual,Type),SigmaExp(Eq1_Residaul,Type),SigmaExp(Eq2_Residual,Ty
pe)] #calculting variance
    Index,DataBase = int(Hours*3600-1),
list(map(DF2.CreateDataFrame,[DFF.GNV1B_all()],ID))
    Pos_ERR = ((DataBase[0].iloc[:Index,4:7].to_numpy()))
    ChiSqr = ChiSqrFun(Residuals,Pos_ERR,Type) #calculating Chisqr
    Fit_Uncertainty =  PropErrorToFit(Fit,Pos_ERR,Residuals) #calculating
uncertainty in coefficients of fit

    return Fit,Type,Omega,Equations,Residuals,Variances,ChiSqr,Fit_Uncertainty
```

```
#----SUMMMARY OF SelectCurveTypeAndFit(Ind,Exp,P,ID,OptiontoSet)----
#P VAR (IDENTIFIER FOR METHOD OF WLS REGRESSION) REQUIRES TO ELEMENTS IN A LIST
#THE FIRST ELEMENT IN THE LIST DENOTES IF WLS IS USED AT ALL
#THE SECOND ELEMENT DENOTES THE TYPE OF WLS TO BE USED (EXPERIMENTAL).
#**THE TYPES AND METHODS/REASONINGS BEHIND THE UTILITIES ARE GIVEN IN:
WeightedObsTransfer(ID,Hours,Exp,P,OptionToSet)**
#THIS UTILITY CALLS THE WeightedObsTransfer FUNCTION TO UPDATE (EQUALIZE) THE
SERIES TO BE AN EQUIVALENT SERIES OF UNIT WEIGHT
#BASED ON THE OptiontoSet TYPE IT CONSTRUCTS THE NORMAL EQUATIONS AND SOLVES FOR
THE COEFFICIENTS TO BE USED TO CONSTRUCT THE EQUATIONS FOR PLOTTING/ RESIDUALS
ETC.
#ALSO IN THIS FUNCTION... THE LSSA IS CALLED TO ESTIMATE THE FUNDAMENTAL
FREQUENCY WHICH IS NOT A PARAMETER THAT IS TO BE ESTIMATED FROM LS ALONE
#BANDWIDTH FOR LSSA IS CHOSEN BASED ON VISUALLY INSPECTING PLOTS IN X,Y,Z AND
WOULD NEED TO BE UPDATED IF A DIFFERENT DATASET IS USED
#LSSA IS NOT ESTIMATED WITH WEIGHTS ALTHOUGH IT WOULD BE INTERESTING TO
EXPERIMENT WITH

def SelectCurveTypeAndFit(Ind,Exp,P,ID,OptiontoSet):
    if P[0] == 1: #If want weight LS... equalize observation series to be
equivalent obs series of unit weight
        Exp1 = WeightedObsTransfer(ID,Hours,Exp,P,OptionToSet)
        Exp = Exp1
    if OptiontoSet == 'Polynomial':
        Col1,Col2,Col3 = np.repeat(1,len(Ind)),Ind,Ind*Ind
        A = np.stack([Col1,Col2,Col3],axis=1)
        c = np.linalg.inv(A.T @ A) @ A.T @ Exp #normal equation
        return c,OptiontoSet,999
    elif OptiontoSet == 'Linear':
        Col1,Col2 = np.repeat(1,len(Ind)),Ind
        A = np.stack([Col1,Col2],axis=1)
        c = np.linalg.inv(A.T @ A) @ A.T @ Exp
        return c,OptiontoSet,999
    elif OptiontoSet == 'Ellipse':
        Omega = 1/(np.linspace(10000,500,500)) #Setting BW for LSSA
        maxSpectrum = []
        for x in range(0,3):
            maxSpectrum.append(np.argmax(LS.LSSA(Ind,Exp[:,x],P=1,Omega=Omega,ind
=[],level=0.01,trend='linear')[0]))
        w = np.mean([x for x in Omega[maxSpectrum]])
        Col1,Col2,Col3 =
np.repeat(1,len(Ind)),np.cos(Ind*2*np.pi*w),np.sin(Ind*2*np.pi*w)
        A = np.stack([Col1,Col2,Col3],axis=1)
        c = np.linalg.inv(A.T @ A) @ A.T @ Exp
        return c,OptiontoSet,w
```

```
    else:
        print("Valid Option Not Selected. Try again.")
        SelectCurveTypeAndFit()


#----SUMMMARY OF FitPlaneType(Ind,Exp)----
#FITS PLANE TO DATASET USING LS. IS NOT USED FOR THIS PROJECT, HOWEVER, WAS USING
IT FOR TESTING
#SPECIFICALLY METHODS ONLINE FOR FITTING A GEOMETRY IN 3D THAT IS IMPLICITLY
DEFINED IN 2D (EXAMPLE A LINE)
#WOULD USE METHOD OF SVD TO FIT A PLANE TO THE DATASET AND THEN PROJECT ALL
POINTS ONTO THIS PLANE
#THEN FIT THE GEOMETRY USING LS THEN PROJECT BACK AFTER THE PARAMETERS HAVE BEEN
ESTIMATED
#WAS INTERESTED TO SEE ABOUT USING LS TO FIT THE PLANE TO THE DATASET AND
ALTHOUGH THE FUNCTION DOES WORK (TESTED)
#AND COMPARE IT TO THE METHOD OF MULTIPLE REGRESSION USED IN THIS CODE
#FIRSTLY FOUND IT DIFFICULT TO DETERMINE HOW TO PROJECT ALL THE POINTS TO THE
PLANE BUT ALSO THIS METHOD SEEMS THAT IT WOULD INTRODUCE ADDITIONAL ERRORS
#THEN THE METHOD PRESENTED HERE - SPECIFICALLY THERE WILL BE ERROR IN FIT OF
PLANE AND THEN ERROR IN THE SUBSEQUENT ESTIMATION OF THE PARAMETERS ON THIS PLANE
#WHICH SEEMS HEURISTICALLY TO BE WORSE OFF THEN FITTING AT ONE TIME USING
MULTIPLE REGRESSION
def FitPlaneType(Ind,Exp): ## LS Solution to Fitting a Plane to Data
    Col1,Col2,Col3 = Exp[:,0],Exp[:,1],np.repeat(1,len(Ind))
    A = np.stack([Col1,Col2,Col3],axis=1)
    B = Exp[:,2].T
    c = np.linalg.inv(A.T @ A) @ A.T @ B
    return c


#----SUMMARY OF WeightedObsTransfer(ID,Hours,Exp,P,OptionToSet)----
#FOR THE SELECTED TIMEFRAME OF POSITION DATA OF GRACE-FO THIS FUNCTION FIRST
CREATES A MATRIX OF THE ASSOCIATED FORMAL ERRORS
#THE FORMAL ERRORS ARE A THEORETICAL ESTIMATE BASED ON MODELLING WHICH ESTIMATES
THE STANDARD DEVIATION OF THE MEASUREMENT
#IN LINEAR REGRESSION WITH ONE EXPLANATORY AND ONE INDEPENDANT VARIABLE... THE
WEIGHT MATRIX SIMPLY DENOTES THE WEIGHTING OF EACH INDIVIDUAL MATRIX
#IN MULTIPLE LINEAR REGRESSION... USING A WEIGHTED MATRIX WHICH ALSO INCLUDES
CROSS CORRELATION BETWEEN THE MEASUREMENT ERRORS FOR EACH OF THE DEPENDENT
VARIABLES
#BECOMES A COMPLEX TASK. A PAPER ON THE SUBJECT IS WRITTEN BY B. LI WHERE THEY
DISCUSS AND COME UP WITH AN ALGORITHM TO: PERFORM MULTIPLE LINEAR REGRESSION
#WITH CORRELATED EXPLANATORY VARIABLES AND RESPONSES. THIS METHOD SEEMED BEYOND
THE SCOPE OF THIS PROJECT AND 6 DIFFERENT METHODS WERE TESTED EXPERIMENTALLY
```

```
#WITH DECISIONS DRIVEN BASED ON LEARNED COURSE KNOWLEDGE.
https://doi.org/10.1179/1752270615Y.0000000006 [PAPER LINK]

#OPTION 0 -> INVERSE OF FORMAL ERROR IS TAKEN AS WEIGHT MATRIX AND THEN
MULTIPLIED BY ASSOCIATED EXPLANATORY VARIABLES TO EQUALIZE OBS TO WEIGHT 1...
    #VARIANCE OF UNIT FACTOR IS GIVEN VALUE OF 1
#OPTION 1 -> SAME AS OPTION 0, EXCEPT FOR Z WHOSE MEASUREMENTS ARE GIVEN
CONSISTENT WEIGHT OF 1 (NOT ADJUSTED). THIS WAS EXPERIMENTED WITH BECAUSE THE WLS
    #WAS NOT WORKING PROPERLY FOR OPTION 0. ALSO, WHEN PLOTTING A HISTOGRAM OF
THE MEASUREMENT ERRORS FOR X,Y,Z... X AND Y FOLLOWED CLOSELY A NORMAL DIST.
    #BUT ERROR Z DID NOT. AS LS ASSUMES MEASUREMENT ERRORS ARE NORMALLY DIST. I
IMAGINED PERHAPS THIS WAS CAUSING PROBLEMS IN THE WLS... SO I SET IT TO 1 FOR Z.
    #THIS WAS ALSO NOT YIELDING A SENSICAL SOLUTION OR APPROXIMATING ANY GEOMETRY
BY FITTING THE GEOMETRY AS A BEST FIT TO THE DATASET
#OPTION 2 -> I ESTIMATED THE UNIT VARIANCE FACTOR K. USING THE FORMAL IN THE
SLIDES. IN THE FORMULA, I ASSUME THE WEIGHTS ARE ALL GIVEN A VALUE OF 1.
    #THIS WAS DONE BECAUSE THIS METHOD TRANSFERS THE WEIGHT TO THE OBSERVATIONS
TO MAKE AN ASSOCIATED WEIGHT MATRIX OF 1
#OPTION 3 -> REPEATED OPTION 2 BUT WITHOUT INCORPERATING MEASUREMENT ERRORS FOR Z
AND USING MEASUREMENTS AS IS WITHOUT ANY WEIGHTING TRANSFER
#OPTION 4 -> INSTEAD OF ASSUMING ERRORS ARE FORMAL ERRORS GIVEN IN THE DATA-
FILES... ERRORS ARE ESTIMATED ARE SAID TO BE EQUAL TO THE RESIDUALS
    #FROM THE LS FIT. WITH THESE ERRORS... THE VARIANCE FACTORS ARE THEN
ESTIMATED FOR THE EXPLANATORY VARIABLES. THEN THE TRANSFERED WEIGHTED
    #LS IS PERFORMED
#OPTION 5 -> I TRY TO INDUCE ARTIFICAL CORRELATION TO THE MEASUREMENT ERRORS TO
SEE THE EFFECT IN THE WLS. AS I MENTIONED ABOVE, MULTIPLE REGRESSION
    #WEIGHTED LS WITH CORRELATED EXPLANATORY VARIABLES, TO MY KNOWLEDGE, SEEMS A
BIT ADVANCED FOR NOW. I REALLY DO NOT BELIEVE THIS METHOD
    #HAS ANY MERIT, BUT IT WAS WORTH THE TRY. ESSENTIALLY I TRIED INTRODUCING THE
CORRELATION OF MEASUREMENT ERRORS BY ESTIMATING THE VARIANCE FACTOR
    #FOR EACH POSITION X, Y, Z AS A WEIGHTED SUM OF EACH OTHER. WHEN LOOKING AT
THE GRAPHS FOR ERRORS, I NOTICED THAT AS Y ERRORS GO UP Z ERRORS GO DOWN.
    #AS X GOES UP, Y GOES DOWN AND X GOES UP Z GOES UP. SO IN THE WEIGHTED
AVERAGE OF THE THREE ERRORS I MULT EITHER 0.5 FACTOR OR -0.5 FACTOR DEP
    #IF IT THE ERROR WOULD GO UP OR DOWN. IN PRACTICE THE WEIGHTED LEAST SQUARES
DID NOT WORK WELL IN THE X OR Y, BUT SOMEHOW THE WEIGHTED LEAST
    #SQUARES IN THE Z WORKED QUITE WELL FOLLOWING THIS METHOD. PERHAPS, AN
ITTERATIVE SOLUTION COULD HAVE BEEN DONE IN THIS MANNER. PERHAPS
    #IT WORKED WELL IN THE Z BECAUSE THE ERRORS IN THE Z DID NOT FOLLOW A NORMAL
DISTRIBUTION VERY WELL AND WEIGHTING THEM WITH X AND Y ERRORS
    #SKEWED THE ERROR TO FOLLOW A MORE NORMAL DISTRIBUTION. CHANGING THE VARIANCE
FACTOR FOR THE X POSITION MEASUREMENTS FOR EXAMPLE WOULD
    #AFFECT THE FIT IN THE Y AND THE Z, WHICH WAS UNEXPECTED BUT MAKES INTUITIVE
SENSE.
```

```
def WeightedObsTransfer(ID,Hours,Exp,P,OptionToSet): #Determining Weight Matrix
    Index,DataBase = int(Hours*3600-1),
list(map(DF2.CreateDataFrame,[DFF.GNV1B_all()],ID))
    Pos_ERR = ((DataBase[0].iloc[:Index,4:7].to_numpy()))
    if P[1] == 0: #Est of Var Unit Weight is Taken as 1 and divided by SD to give
Weight Mat
            P_Posx, P_posy, P_posz = 1/Pos_ERR[:,0],1/Pos_ERR[:,1],1/Pos_ERR[:,2]
#Var Unit Weight == 1
    elif P[1] == 1: #Est of Var Unit Weight is Taken as 1 and divided by SD to
give Weight Mat Except for Z
            P_Posx, P_posy, P_posz =
1/Pos_ERR[:,0],1/Pos_ERR[:,1],np.repeat(1,len(Pos_ERR[:,2])) #Var Unit Weight ==
1 and Non-weighting to Z
    elif P[1] == 2: #Est of Var Unit Weight is estimated by sigma_hat =
sqrt(p*delta**2)/n where the delta (measurement error) is assumed to equal the SD
            EstVarOfUnitWeight =
EstVarUnitWeight(ID,Ind,Exp,P,OptionToSet,Pos_ERR,0)
            P_Posx, P_posy, P_posz =
EstVarOfUnitWeight[0]/Pos_ERR[:,0],EstVarOfUnitWeight[1]/Pos_ERR[:,1],EstVarOfUni
tWeight[2]/Pos_ERR[:,2]
    elif P[1] == 3: #Only weight x, y (follow normal dist with ~0 mean)
            EstVarOfUnitWeight =
EstVarUnitWeight(ID,Ind,Exp,P,OptionToSet,Pos_ERR,0)
            P_Posx, P_posy, P_posz =
EstVarOfUnitWeight[0]/Pos_ERR[:,0],EstVarOfUnitWeight[1]/Pos_ERR[:,1],np.repeat(1
,len(Pos_ERR[:,2]))
    elif P[1] == 4: #Errors are residuals from LS best fit without weights. Then
Estimate Variance Factor
            EstVarOfUnitWeight =
EstVarUnitWeight(ID,Ind,Exp,P,OptionToSet,Pos_ERR,1)
            P_Posx, P_posy, P_posz =
EstVarOfUnitWeight[0]/Pos_ERR[:,0],EstVarOfUnitWeight[1]/Pos_ERR[:,1],EstVarOfUni
tWeight[2]/Pos_ERR[:,2]
    elif P[1] == 5: #Do Weighted LS With POSx full error then POSy half error
then POSz half error ->VarOfUnitWeight for X... then for Y... then for Z... then
do weighted LS
                    #With these variance factors
            #Fixing X error and introducing artificial correlation
            P_Posx = (Pos_ERR[:,0]+-0.5*Pos_ERR[:,1]+Pos_ERR[:,2]*0.5)/3
            Pos_ERR = np.stack([P_Posx,P_Posx,P_Posx],axis=1)
            VarEst1 = EstVarUnitWeight(ID,Ind,Exp,P,OptionToSet,Pos_ERR,0)[0]

            #Fixing Y error and introducing artifical correlation
            P_Posx = (Pos_ERR[:,1]+-0.5*Pos_ERR[:,0]+Pos_ERR[:,2]*0.5)/3
```

```
            Pos_ERR = np.stack([P_Posx,P_Posx,P_Posx],axis=1)
            VarEst2 = EstVarUnitWeight(ID,Ind,Pos_ERR,P,OptionToSet,Pos_ERR,0)[1]

            P_Posx = (Pos_ERR[:,2]+0.5*Pos_ERR[:,1]+Pos_ERR[:,0]*0.5)/3
            Pos_ERR = np.stack([P_Posx,P_Posx,P_Posx],axis=1)
            VarEst3 =
EstVarUnitWeight(ID,Ind,Pos_ERR,P,OptionToSet,Pos_ERR,0)[2]

            #Using Individual Variance of Unit Estimate Weight
            P_Posx, P_posy, P_posz =
VarEst1/Pos_ERR[:,0],VarEst2/Pos_ERR[:,1],VarEst3/Pos_ERR[:,2]

    else:
        P_Posx,P_posy,P_posz =
np.repmat(1,len(Ind)),np.repmat(1,len(Ind)),np.repmat(1,len(Ind))

    P_Pos = np.stack([P_Posx, P_posy, P_posz],axis=1)
    Xweighted,Yweighted,Zweighted = Exp[:,0]*P_Pos[:,0], Exp[:,1]*P_Pos[:,1],
Exp[:,2]*P_Pos[:,2] #turns weight matrix to 1
    PosWeighted = np.stack([Xweighted,Yweighted,Zweighted],axis=1)
    return PosWeighted

#----SUMMARY OF EstVarUnitWeight(ID,Ind,Exp,P,OptionToSet,Pos_ERR,Weight_Itt)----
#THIS FUNCTION ESTIMATES THE ESTIMATED VARIANCE OF UNIT WEIGHT
#IF THE WEIGHT_ITT IS EQUAL TO 1... IT ESTIMATES THE VARIANCE FACTOR BY USING
ERRORS TAKEN FROM THE LS RESIDUALS AND EST. THE FACTOR FROM THIS
#IF THE WEIGHT_ITT IS SET TO 0... IT ESTIMATES THE VARIANCE FACTOR BY USING
ERRORS TAKEN FROM THE FORMAL ERRORS PROVIDED IN THE EPHEMERIS DATA FILES
#IN BOTH CASES THE WEIGHT IS SET TO 1 FOR THE MEASUREMENTS BECAUSE IN THIS LS
PROCESS THE WEIGHT IS BEING TRANSFERRED TO THE OBSERVATIONS
#TO MAKE A WEIGHT MATRIX OF 1. AS A RESULT - I BELIEVED IT TO HOLD TO KEEP THE
WEIGHT MATRIX SET TO 1 IN THE VARIANCE FACTOR ESTIMATION
def EstVarUnitWeight(ID,Ind,Exp,P,OptionToSet,Pos_ERR,Weight_Itt): #Estimating
Variance of Unit Weight
    if Weight_Itt == 1:
        P = [0,0]
        [Fit,Type,Omega,Eq0,Eq1,Eq2,Eq0_Residual,Eq1_Residaul,Eq2_Residual] =
LSFittoData(Ind,Exp,P,ID,OptionToSet)
        Eqs = [Eq0_Residual,Eq1_Residaul,Eq2_Residual] #Observation Errors
(derived from LS residaul)
        VarUnitW_hat = []
        Weighted1 = np.repeat(1,len(Ind))
        for x in range(0,3):
            VarUnitW_hat.append(np.sqrt(np.sum(Eqs[x]**2*Weighted1)/len(Ind)))
        return VarUnitW_hat
```

```
    else:
        Weighted1 = np.repeat(1,len(Ind))
        VarUnitW_hat = []
        for x in range(0,3):
            VarUnitW_hat.append(np.sqrt(np.sum(Weighted1*Pos_ERR[:,x]**2)/len(Ind
)))
        return VarUnitW_hat


#----SUMMARY OF PlotError(Hours,DataBase,ID)----
#THIS FUNCTION PLOTS HISTOGRAM OF THE ERRORS GIVEN IN THE EPHEMERIS FILES
#IN A SUBPLOT
def PlotError(Hours,DataBase,ID):
    Index,DataBase = int(Hours*3600-1),
list(map(DF2.CreateDataFrame,[DFF.GNV1B_all()],ID))
    Pos_ERR = ((DataBase[0].iloc[:Index,4:7].to_numpy()))
    fig, axs = plt.subplots(3)
    axs[0].hist(Pos_ERR[:,0],label='POS X Sigma')
    axs[1].hist(Pos_ERR[:,1],label='POS Y Sigma')
    axs[2].hist(Pos_ERR[:,2],label='POS Z Sigma')
    axs[2].set(xlabel='Model Estimate of Sigma')
    axs[1].set(ylabel = 'Occurence of Error')
    axs[0].legend()
    axs[1].legend()
    axs[2].legend()
    fig.show()

#----SUMMARY OF SigmaExp(Residual,Type)----
#THIS FUNCTION RETURNS THE VARIANCE OF THE LEAST SQUARES REGRESSION
def SigmaExp(Residual,Type):
    if Type == 'Linear':
        Sigma = np.sqrt((1/(len(Residual)-2))*(np.sum(Residual**2)))
    else:
        Sigma = np.sqrt((1/(len(Residual)-3))*(np.sum(Residual**2)))
    return Sigma

#----SUMMARY OF ChiSqrFun(Residual_List,ID,Hours,Type)
#THIS FUNCTION CALCULATES THE CHISQR TO SEE IF IT IS ~1 AND INDEED A GOOD FIT
def ChiSqrFun(Residual_List,Pos_ERR,Type):
    ChiSqr = []
    for x in range(0,len(Residual_List)):
        if Type == 'Linear':
            ChiSqr.append(np.sum((1/(Pos_ERR[:,x]**2))*Residual_List[x]**2)/(len(
Residual_List[x]-2))) # Should ~ N-n
        else:
```

```
            ChiSqr.append(np.sum((1/(Pos_ERR[:,x]**2))*Residual_List[x]**2)/(len(
Residual_List[x]-3))) # Should ~ N-n
    return ChiSqr


#----SUMMARY OF PropErrorToFit(Fit,Pos_ERR,Residual_List)
#PROPAGATES ERROR FROM OBSERVABLES INTO COEFFICIENT ESTIMATES USING GENERAL LAW
OF ERROR PROPAGATION
def PropErrorToFit(Fit,Pos_ERR,Residual_List):
    Pos_Err_Var = []
    for x in range(0,3):
        Pos_Err_Var.append(np.var(Pos_ERR[:,x]))
    Pos_Err_Var = np.array(Pos_Err_Var)
    Error_WithError = Fit**2 @ Pos_Err_Var**2

    Residual_array = np.array(Residual_List)
    Residual_Var = []
    for x in range(0,3):
        Residual_Var.append(np.var(Residual_array[:,x]))
    Residual_Var = np.array(Residual_Var)
    Error_WithRes = Fit**2 @ Residual_Var**2
    Fit_Uncertainty = [np.sqrt(Error_WithError), np.sqrt(Error_WithRes)]
    return Fit_Uncertainty

#----SUMMARY OF
ExtrapolateAndPlot(DataProd,ID,Fit,Type,Omega,Hours_Fit,Hours_Extrap)
#EXTRAPOLATES POSITION BASED ON LS FIT
def ExtrapolateAndPlot(DataProd,Ind,ID,Fit,Type,Omega,Hours_Fit,Hours_Extrap):
    IndEnd = len(Ind)
    [Ind,Exp] = SelectDataProd(DataProd,ID,Hours_Extrap)
    Ind = Ind[IndEnd:] #Extracting Data Only After Extrapolation
    Exp = Exp[IndEnd:,:]
    if Hours_Fit >= Hours_Extrap:
        Hours_Extrap = input("Hour to extrapolate to is <= hours of data-set.
Please enter new hour to extrapolate to ")
    else:
        if Type == 'Ellipse':

            Eq0 = Fit[0,0] + Fit[1,0]*np.cos(2*np.pi*Omega*Ind) +
Fit[2,0]*np.sin(2*np.pi*Omega*Ind)
            Eq1 = Fit[0,1] + Fit[1,1]*np.cos(2*np.pi*Omega*Ind) +
Fit[2,1]*np.sin(2*np.pi*Omega*Ind)
            Eq2 = Fit[0,2] + Fit[1,2]*np.cos(2*np.pi*Omega*Ind) +
Fit[2,2]*np.sin(2*np.pi*Omega*Ind)
            print(Exp[:,0])
```

```
        Eq0_Residual = Exp[:,0]-Eq0
        Eq1_Residaul = Exp[:,1]-Eq1
        Eq2_Residual = Exp[:,2]-Eq2

    elif Type == 'Linear':
        Eq0 = Fit[0,0]+Fit[1,0]*Ind
        Eq1 = Fit[0,1]+Fit[1,1]*Ind
        Eq2 = Fit[0,2]+Fit[1,2]*Ind

        Eq0_Residual = Exp[:,0]-Eq0
        Eq1_Residaul = Exp[:,1]-Eq1
        Eq2_Residual = Exp[:,2]-Eq2

    elif Type == 'Polynomial':
        Eq0 = Fit[0,0]+Fit[1,0]*Ind+Fit[2,0]*Ind**2
        Eq1 = Fit[0,1]+Fit[1,1]*Ind+Fit[2,1]*Ind**2
        Eq2 = Fit[0,2]+Fit[1,2]*Ind+Fit[2,2]*Ind**2

        Eq0_Residual = Exp[:,0]-Eq0
        Eq1_Residaul = Exp[:,1]-Eq1
        Eq2_Residual = Exp[:,2]-Eq2


fig, axs = plt.subplots(3)
axs[0].plot(Ind,Exp[:,0],'r--',linewidth = 3,label='POD DATA - TRUE')
axs[0].plot(Ind,Eq0,'b',label='LS FIT - EXTRAP.')
axs[0].set_title('GNSS Data in X with LS Fit EXTRAPOLATED')
axs[1].plot(Ind,Exp[:,1],'r--',linewidth = 3,label='POD DATA - TRUE')
axs[1].plot(Ind,Eq1,'b',label='LS FIT - EXTRAP.')
axs[1].set_title('GNSS POD Data in Y with LS Fit EXTRAPOLATED')
axs[2].plot(Ind,Exp[:,2],'r--',linewidth = 3,label='POD DATA - TRUE')
axs[2].plot(Ind,Eq2,'b',label='LS FIT - EXTRAP.')
axs[2].set_title('GNSS POD Data in Z with LS Fit EXTRAPOLATED')
axs[1].set(ylabel = 'Position in ECI [m]')

axs[2].set(xlabel = 'Time [s]')
for ax in axs.flat:
    ax.label_outer()
axs[0].legend()
axs[1].legend()
axs[2].legend()
fig.show()
fig1 = plt.figure()
ax1 = plt.axes(projection='3d')
```

```
    ax1.plot(Exp[:,0],Exp[:,1],Exp[:,2],'r--',linewidth = 3,label='POD DATA')
    ax1.plot(Eq0,Eq1,Eq2,'b',label='LS FIT')
    ax1.set_title('GNSS POD Data in 3D EXTRAPOLATED')
    ax1.set(xlabel='Data in X [m]',ylabel='Data in Y [m]',zlabel='Data in Z [m]')
    ax1.legend()
    fig1.show()

    fig2, axs2 = plt.subplots(3)
    axs2[0].plot(Ind,Eq0_Residual)
    axs2[0].set_title('GNSS POD in X LS Fit Residual EXTRAPOLATED')
    axs2[1].plot(Ind,Eq1_Residaul)
    axs2[1].set_title('GNSS POD Data in Y LS Fit Residual EXTRAPOLATED')
    axs2[2].plot(Ind,Eq2_Residual)
    axs2[2].set_title('GNSS POD Data in Z LS Fit Residual EXTRAPOLATED')
    for ax in axs2.flat:
          ax.set(xlabel='Time [s]', ylabel = 'Position in ECI [m]')
    for ax in axs2.flat:
        ax.label_outer()
    fig2.show()

    Residuals = [Eq0_Residual,Eq1_Residaul,Eq2_Residual]

    Variances =
[SigmaExp(Eq0_Residual,Type),SigmaExp(Eq1_Residaul,Type),SigmaExp(Eq2_Residual,Ty
pe)] #calculting variance
    Index,DataBase = int(Hours_Extrap*3600-1),
list(map(DF2.CreateDataFrame,[DFF.GNV1B_all()],ID))
    Pos_ERR = ((DataBase[0].iloc[:Index,4:7].to_numpy()))
    Pos_ERR = Pos_ERR[IndEnd:,:]
    ChiSqr = ChiSqrFun(Residuals,Pos_ERR,Type) #calculating Chisqr
    Fit_Uncertainty =  PropErrorToFit(Fit,Pos_ERR,Residuals) #calculating
uncertainty in coefficients of fit

    return Residuals,Variances,ChiSqr,Fit_Uncertainty
```