ENGO 531 – Advanced Photogrammetric and Ranging Techniques

Lab Report #1

Course Instructor: Dr Derek Lichti

Teaching Assistant: Sandra Simenova

Due Date: 10/12/21

Date of Submission: 10/12/21

Student Name: Jan Erik Naess (30068695)

Table of Contents

Table of Figures

Introduction

This lab was constructed to begin developing a full-stack bundle adjustment. Python was chosen because of its modularity capabilities, ability to read in data and store with DataFrames, and for its general ease of use. Software was developed in an Object Oriented Manner, currently housing 8+ classes and dozens of functions. This high level of modularity allowed for efficient debugging and upgrading of code. Data was validated with DataFrames that allow the user to call the desired information with one line of code. Data was verified with post adjustment tests specially formatted for a Bundle Adjustment. All code was sufficiently documented both inline and via descriptive function headers. An input file was built in to read and write the components that were specific to this dataset. Several issues with the data were found and addressed at the end of this lab report.

Documented Source Code

Main Classes

Software was developed within Python using common libraries such as math, numpy and pandas. Data was visualized using matplotlib and was processed with four in-house classes. The "Network" class conducted the overarching LSA and generates final output values for the assistive classes. The one functional model class, "Design_o" which generates all colinear matrices before the least-square adjustment and updates the design matrices for the LSA. The "LSA" super class was inherited by all classes and contained metrics such as x^0 and helped with sorting of columns to correctly input and update important matrices.

Assistive Classes

The "PostAdjustmentTester" class was made to autogenerate the results displayed in the verification report and leverages the "Net" class and "Tools" class to generate and output/save results. The "Tools" and "Tables" classes were made to break down functionality that was needed within other classes and allow then to be inherited for functionality such as outputting files or splitting DataFrames in unique ways.

Function Documentation

All functions were documented with a standard Description, Input and Output portion. The description consisted of the use-case for that function and included any notes about potential changes or scope constraints to the function's use. The input contained all variables that needed to be either read in or initialized before calling the function. The output contained all files that were either returned, updated, or initialized by calling this function. See below for a sample function documentation that was used within the LeastSquares class.

```
def read_2D(self):
    """
    Desc:
        reads in the 2D set of points and assigns values
        expects format of [name easting northing known/unknown]
        more specifically: [Point X[m] Y[m] Known[n]/Unknown[u]]
    Input:
        self.file_name
    Output:
        self.u_list (string list of unknown)
        self.x_0 (initial guesses of unknowns)
        self.c (constant values of knowns)
        self.datums (string list of knowns)
        self.u # of unknowns
    """
```

Figure 1: Sample Function Documentation

General Documentation

The rough goal was to have a comment for nearly every line of code that was written. In general, comments were recorded for each minor piece of functionality within a function. This assisted greatly in debugging and building upon old code.

Formatted Bundle Adjustment Output File (Validation)

The software package developed leverages pandas DataFrames to output whichever values are desired. A function to output the DataFrame with a file name is located within the Tools class and autonomously creates a folder called “Files” and outputs the DataFrame as a csv. Overall, the data seems to have converged to a reasonable standard deviation. It can be seen that most of the angles have a low angular standard deviation and that points locations for the images are within .5mm of any of the three axises. Control points largely kept their values and were not greatly influenced by the adjustment which was a desired result. Image points had standard deviations that stayed under 2mm which is a reasonable result given some of the conditions such as only one tie point for one of the images. Below is a clipping of the outputted file which contains the standard deviation for the final points.

	Unknown	Final Value (mm or rad)	Value Standard Deviation (mm or rad)
0	1Xcj	-746.954882	0.472346
1	1Ycj	-268.594535	0.546328
2	1Zcj	113.702213	0.785786
3	1w	1.617254	0.000295
4	1o	-0.345005	0.000193
5	1k	-0.052867	0.000157
6	5Xcj	2614.137910	0.583023
7	5Ycj	89.358567	0.568314
8	5Zcj	110.034597	0.726797
9	5w	1.564828	0.000307
10	5o	0.664369	0.000238
11	5k	-0.031253	0.000172
12	8Xcj	-122.243956	0.591658
13	8Ycj	-665.847857	0.365527
14	8Zcj	61.103533	0.787534
15	8w	1.484602	0.000270
16	8o	-0.204263	0.000214
17	8k	1.548610	0.000148
18	9Xcj	537.074443	1.309577
19	9Ycj	-633.998912	0.650201
20	9Zcj	57.647061	1.809267
21	9w	1.481513	0.000544
22	9o	-0.093095	0.000397
23	9k	1.576830	0.000200
24	24Xi	-0.091663	0.009999
25	24Yi	1700.002253	0.010000
26	24Zi	100.128517	0.009999
27	71Xi	2200.091747	0.009999
28	71Yi	2499.980564	0.010000
29	71Zi	-1200.026727	0.009999
...	...	...	...

Figure 2: Final Output File (sample)

DataFrames or matrices available for autonomous outputting include but are not limited to: all input files (con, ext, int, out, pho, tie), combined input files (obj), all final matrices (Cl, Cr, x_hat, w_0, S_hat, etc.), all initial matrices (x_0, obs, l_0, etc.) and any of the PostAdjustmentTester dataframes.

Verification Report

The PostAdjustmentTester class was generated to verify the results of the least square adjustment. Overall, the potential issue with the models used was that the additional control points as fake observations were not input due to time constraints. This resulted in an a-posteriori factor that was slightly above the example which generated a value of 7, while this generated a value of 10.5 as seen below.

```
n.a_post

10.529394797808639
```

Figure 31: A-Posteriori Variance Factor (unitless)

This was compared to an a-priori of 1 in the global test. Because of the high level of redundancy, the test required a better set of functional models in order to make full use of the large amount of redundancy. The test and respective analysis may be seen below.

```
15743.277982752987 tested with chi_square boundries of 170.809 and 115.463
Global A-Posteriori Variance Factor Test **failed** at a 95.0% confidence level
There is indication that errors exsist within residual or the math models
```

Figure 42: Global A-Posteriori Variance Factor Test

The a-posteriori variance factor test failed indicating that there may have been errors or poorly chosen math models. The failure of this test is a strong enough indication that other components should be tested but is not a guarantee that there are any major issues.

	Unknown	Final Value	Value Standard Deviation	Test Value	Indicated Significance	Alpha Tested	Confidence Level	Test Bounds
0	1Xcj	-746.954882	0.472346	-1581.370994	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
1	1Ycj	-268.594535	0.546328	-491.635741	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
2	1Zcj	113.702213	0.785786	144.698742	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
3	1w	1.617254	0.000295	5490.585688	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
4	1o	-0.345005	0.000193	-1788.787445	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
5	1k	-0.052867	0.000157	-336.304107	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
6	5Xcj	2614.137910	0.583023	4483.766963	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
7	5Ycj	89.358567	0.568314	157.234479	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
8	5Zcj	110.034597	0.726797	151.396703	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
9	5w	1.564828	0.000307	5102.631309	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
10	5o	0.664369	0.000238	2789.649118	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
11	5k	-0.031253	0.000172	-181.895015	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
12	8Xcj	-122.243956	0.591658	-206.612409	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
13	8Ycj	-665.847857	0.365527	-1821.611528	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
14	8Zcj	61.103533	0.787534	77.588394	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
15	8w	1.484602	0.000270	5491.519742	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
16	8o	-0.204263	0.000214	-955.572536	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
17	8k	1.548610	0.000148	10457.224116	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
18	9Xcj	537.074443	1.309577	410.112903	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
19	9Ycj	-633.998912	0.650201	-975.081758	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
20	9Zcj	57.647061	1.809267	31.862103	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
21	9w	1.481513	0.000544	2722.833148	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
22	9o	-0.093095	0.000397	-234.525702	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
23	9k	1.576830	0.000200	7867.905054	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
24	24Xi	-0.091663	0.009999	-9.167071	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
25	24Yi	1700.002253	0.010000	170004.568524	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
26	24Zi	100.128517	0.009999	10013.894370	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
27	71Xi	2200.091747	0.009999	220023.966745	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
28	71Yi	2499.980564	0.010000	250003.765396	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]
29	71Zi	-1200.026727	0.009999	-120009.435316	Yes	0.05	95.0	[-1.6556551725774078, 1.6556551725774071]

Figure 5: Significance of Estimated Parameters (sample)

The significance of all parameters was then assessed. This checked to see if there was evidence that the parameter had statistical significance to be used within the model and for future applications. The test resulted in an overwhelming yes, indicating that the parameters had reached desirable levels of precision given the functional models and quality of the dataset.

```
644.6222814353696 tested with chi_square of 27.58711163827534
The Semi-Global, goodness-of-fit test on the residuals **Failled**
There is a sign that either there are outliers or the functional model was not appropriate for the data set
```

Figure 6: Goodness-of-fit test

Next the goodness-of-fit test was conducted. The failure of this test indicated that there may have either been outliers or portions of the data that were of relatively low quality. One of these points may be been because there was only one tie point in image four which greatly reduced its quality.

	Observation	Outlier	Confidence Level	Test Value	Test Bounds
0	0	Yes	99.0	3.154959	[-2.575829303548901, 2.5758293035489004]
1	1	Yes	99.0	-3.300945	[-2.575829303548901, 2.5758293035489004]
2	2	No	99.0	0.213372	[-2.575829303548901, 2.5758293035489004]
3	3	No	99.0	0.101073	[-2.575829303548901, 2.5758293035489004]
4	4	No	99.0	0.122585	[-2.575829303548901, 2.5758293035489004]
5	5	No	99.0	0.381360	[-2.575829303548901, 2.5758293035489004]
6	6	No	99.0	-1.224935	[-2.575829303548901, 2.5758293035489004]
7	7	No	99.0	-2.573561	[-2.575829303548901, 2.5758293035489004]
8	8	No	99.0	-0.207838	[-2.575829303548901, 2.5758293035489004]
9	9	No	99.0	0.395735	[-2.575829303548901, 2.5758293035489004]
10	10	No	99.0	-0.144330	[-2.575829303548901, 2.5758293035489004]
11	11	No	99.0	1.205178	[-2.575829303548901, 2.5758293035489004]
12	12	No	99.0	0.106986	[-2.575829303548901, 2.5758293035489004]
13	13	No	99.0	-0.087814	[-2.575829303548901, 2.5758293035489004]
14	14	No	99.0	-0.111076	[-2.575829303548901, 2.5758293035489004]
15	15	No	99.0	0.239778	[-2.575829303548901, 2.5758293035489004]
16	16	No	99.0	-0.320612	[-2.575829303548901, 2.5758293035489004]
17	17	No	99.0	0.498687	[-2.575829303548901, 2.5758293035489004]
18	18	No	99.0	0.052187	[-2.575829303548901, 2.5758293035489004]
19	19	No	99.0	0.006072	[-2.575829303548901, 2.5758293035489004]
20	20	No	99.0	-0.259822	[-2.575829303548901, 2.5758293035489004]
21	21	No	99.0	0.280123	[-2.575829303548901, 2.5758293035489004]
22	22	No	99.0	0.367264	[-2.575829303548901, 2.5758293035489004]
23	23	No	99.0	0.003553	[-2.575829303548901, 2.5758293035489004]
24	24	No	99.0	0.135782	[-2.575829303548901, 2.5758293035489004]
25	25	No	99.0	-0.124609	[-2.575829303548901, 2.5758293035489004]
26	26	No	99.0	0.211705	[-2.575829303548901, 2.5758293035489004]
27	27	No	99.0	-0.029356	[-2.575829303548901, 2.5758293035489004]
28	28	No	99.0	0.393526	[-2.575829303548901, 2.5758293035489004]
29	29	No	99.0	-0.054203	[-2.575829303548901, 2.5758293035489004]
...	...	...	...	...	...

Figure 7: Blunder Test (sample)

A blunder test was conducted on all observations in order to check whether they may be outliers. This test was conducted to the 99% confidence interval and resulted in several observations being considered outliers. However, the data proved largely free of outliers, and where there was one indicated, there was enough redundancy to compensate for it.

Adjustment Configuration Input File

Key values for the Network Adjustment were input into the Input.txt file. This file consisted of only two row, one for the header names and the second for the respective values that would be read in. These values were used by both the “Bundle” class and “Design_o” class to initialize their variables. Below is the input DataFrame that is read in and accessed.

	std_u,	std_n,	pix_to_m,	n_p_d(mm),	Np,	Mp,
0	.00345,	.01,	3.45e-6,	7,	3000,	4000

Figure 8: Input.txt DataFrame

In order to add new values all that the user needs to do is ass the variable name and the assign its to the class once it is read in. Below is the format of the file, both white spaces and commas may be used to separate variables and numbers.

```
std_u, std_n, pix_to_m, n_p_d(mm), Np, Mp,
.00345, .01, 3.45e-6, 7, 3000, 4000
```

Figure 3: Input.txt composition

Short Answer to Question

The quality of this dataset is such that the bundle adjustment will converge within a few iterations, depending on the tolerances used. However, the dataset contains at least three “problems”. Identify and briefly explain these problems

Input File Spacing

A plethora of issues were found in the consistency of the input file separation techniques. Separation between values varied both between files and within each file themselves. Some examples included extra tabs after completion of a line, using whitespaces instead of tabs and using varying lengths of white spaces.

Pandas’ read_csv() function was leveraged to read in these files regardless of their poor formatting. By setting the delim_whitespace parameter to be True, all files were read in based on commas, tabs, and all variations of spacing. Below is the function and parameter in that was called to read in the .pho observation file.

```
#uses mixed spacing to read in files... nbd ;-)
df = pd.read_csv(self.pho_file, header = None, delim_whitespace =True)
```

Figure 10: pd.read_csv function

Poor Tie Point Count

Image four included only one tie point. In order to better stich together the images that were taken, it is common practice to have a minimum of four or five tie points. However, negative effects of this were mitigated by using control points between images. That being said, the issue with using control points is that they offer very little room to move because of their heavy weights. This results in the images have a low level of willingness to readjust to possibly better positions.

High Correlation

Correlations between the EOP’s were relatively high which indicated that the final positions may have systematic errors. Correlation may be reduced if a different model is used or if there is a larger amount of redundancy. Therefore, the easiest fix would be to collect more data or to use a second camera to collect datapoints of tie points and control points from a different set of parameters so that there would be less reliance on the IOP’s of only one camera. As always, more redundancy is better.

Blunders

Both the example output, and the software’s output detected various blunders at the 99% confidence level. Because of the large amount of redundancy, it would be wise to redo the LSA, taking the largest blunder out each time until no blunders were detected.

Discussion

Unit Conversion Importance

It is common practice in surveying to provide angular precision in arc seconds and distance precision in whole numbers (2mm instead of .002m). There are two important adjustments that must made to errors in order to integrate them into a programable least-squares-adjustment. The first adjustment is to the angular error, which must be converted into decimal degrees, and is often then converted from degrees to radians. Radians is preferred because most code libraries by default read and return angle measurements in radians. The second adjustment that should be made is in the conversion of non-meter errors into decimal millimeters. This is because measurements and positioning are normally provided in meters. If they are provided in a different unit, then all distance and x, y, z coordinates and errors should at the bare minimum be uniform. Otherwise, disproportional adjustments will occur.

The importance of conversion from LHC to RHC should also be noted. In the future it would be wise to write down which coordinate system functional models output as so that respective conversions may be made without unnecessary debugging.

Conversion Criteria

It is a commonly accepted practice to have the conversion criteria of an LSA be based off of the parameter's standard deviations. This means that Cx is computed and then the diagonal elements are taken out and square rooted to see their standard deviation. Once all standard deviations are below one-half of the observation's standard deviations then it is often acceptable to end convergence. Because the functional models used were well suited for this application, a simpler convergence criterion was used. This LSA was programmed to meet convergence at .0001 mm: of which was converged to after the fourth iteration.

Automated output of results and matrices

Least-squares-adjustments often use repeatable statistical tests and desire similar formats of outputted results. Additional time was invested in this lab to create several classes for performing fully autonomous analysis and matrix figure generation. The "Tools" class was created to automate visualization of importance matrices. The "PostAdjustmentTester" class was created to leverage final matrices outputs and conduct statistical tests on them. Lastly, the "Tables" class was to consolidate all statistic table values in an easily accessible and formattable location so that statistical tests could be easily conducted and automated.

References

Gao, Y. (2021). *Lab1 - Instructions.* Retrieved from D2L: https://d2l.ucalgary.ca/d2l/le/content/399854/viewContent/4877097/View

El-Sheimy, N. (2021). *Review of least squares (parametric).* Retrieved from D2L: https://d2l.ucalgary.ca/d2l/le/content/399854/viewContent/4843398/View

Detchev, I. (2020). *Examples for Post-Adjustment Tests.* [PDF]

Appendices

Cl Values (mm^2 and $radians^2$)

```
matrix([[ 6.74823301e-07,  3.34208092e-08,  1.38005978e-07, ...,
         -2.75125218e-09, -4.51771739e-09, -1.20103519e-08],
        [ 3.34208092e-08,  9.11841435e-07,  6.22579638e-08, ...,
          1.26568697e-09,  3.91013344e-09, -2.14700648e-09],
        [ 1.38005978e-07,  6.22579638e-08,  5.44082447e-07, ...,
         -8.20836564e-09, -8.80541832e-09, -1.39891662e-08],
        ...,
        [-2.75125218e-09,  1.26568697e-09, -8.20836564e-09, ...,
          1.23082211e-06,  3.26215153e-07,  2.95061082e-07],
        [-4.51771739e-09,  3.91013344e-09, -8.80541832e-09, ...,
          3.26215153e-07,  1.43532329e-06,  3.61296839e-07],
        [-1.20103519e-08, -2.14700648e-09, -1.39891662e-08, ...,
          2.95061082e-07,  3.61296839e-07,  8.71448720e-07]])
```

Cr values (mm^2 and $radians^2$)

```
matrix([[ 1.31893339e-03, -3.34208092e-08, -1.38005978e-07, ...,
          2.75125218e-09,  4.51771739e-09,  1.20103519e-08],
        [-3.34208092e-08,  1.31869637e-03, -6.22579638e-08, ...,
         -1.26568697e-09, -3.91013344e-09,  2.14700648e-09],
        [-1.38005978e-07, -6.22579638e-08,  1.31906413e-03, ...,
          8.20836564e-09,  8.80541832e-09,  1.39891662e-08],
        ...,
        [ 2.75125218e-09, -1.26568697e-09,  8.20836564e-09, ...,
          1.31837739e-03, -3.26215153e-07, -2.95061082e-07],
        [ 4.51771739e-09, -3.91013344e-09,  8.80541832e-09, ...,
         -3.26215153e-07,  1.31817289e-03, -3.61296839e-07],
        [ 1.20103519e-08,  2.14700648e-09,  1.39891662e-08, ...,
         -2.95061082e-07, -3.61296839e-07,  1.31873676e-03]])
```

Final Output File

,Unknown,Final Value (mm or rad),Value Standard Deviation (mm or rad)

0,1Xcj,-746.9548816473772,0.47234639090886116

1,1Ycj,-268.59453481058773,0.5463283331279489

2,1Zcj,113.70221338066949,0.7857857785265298

3,1w,1.6172539672138702,0.00029455035566097027

4,1o,-0.34500465628088756,0.00019287068297236745

5,1k,-0.05286714511601137,0.00015720041486024032

6,5Xcj,2614.1379095145135,0.5830226975124676

7,5Ycj,89.3585670723902,0.5683140735063945

8,5Zcj,110.03459675304649,0.7267965198349986

9,5w,1.5648278842545995,0.0003066707722978768

10,5o,0.6643685871286442,0.00023815489294351521

11,5k,-0.03125266606156934,0.00017181705668870259

12,8Xcj,-122.24395610749683,0.5916583459119032

13,8Ycj,-665.8478573599588,0.3655268136354254

14,8Zcj,61.10353279014381,0.7875344443864282

15,8w,1.4846016896456586,0.000270344414528684

16,8o,-0.20426343919457873,0.000213760265605699

17,8k,1.5486101077566996,0.0001480899797660257

18,9Xcj,537.074443237871,1.309577044172131

19,9Ycj,-633.9989119913413,0.6502007723911171

20,9Zcj,57.64706144648279,1.8092673258249081

21,9w,1.4815126212776706,0.0005441070167349298

22,9o,-0.09309534507325597,0.00039695156806900404

23,9k,1.5768299188781774,0.00020041293178162085

24,24Xi,-0.09166306174906819,0.009999165806853212

25,24Yi,1700.0022530724082,0.009999744523501405

26,24Zi,100.12851688797763,0.009998958765750757

27,71Xi,2200.0917465010602,0.009999327705273975

28,71Yi,2499.980564030343,0.009999771643717221

29,71Zi,-1200.0267266151197,0.00999943648980049

30,85Xi,2762.3830142962056,9.967323342003434

31,85Yi,1972.3483630592066,10.437907505927175

32,85Zi,470.6209001182112,1.9562866541856834

33,89Xi,2770.6058521373275,0.583547210332845

34,89Yi,1973.8472563141615,1.3487743807474804

35,89Zi,-228.29113708077492,0.6733761503442887

36,132Xi,1799.9588938710501,0.009999620211336296

37,132Yi,2999.988907736204,0.009999900140227568

38,132Zi,-800.0035974025712,0.009999613130439637

39,133Xi,2300.066148951539,0.009999250512698173

40,133Yi,2400.016969910048,0.009999827776666153

41,133Zi,-599.9254192525424,0.009999361461033926

42,A5Xi,46.99837427920278,0.009999120048456687

43,A5Yi,1787.9902454560681,0.009999290966118436

44,A5Zi,1982.0075490227478,0.009999027692961621

45,B1Xi,1646.0055155105613,0.00999940593587011

46,B1Yi,3051.9992635012086,0.009999832623166916

47,B1Zi,1258.003800031549,0.00999935381676519

48,B2Xi,2225.0011853992123,0.009999023660392013

49,B2Yi,2489.9992496611776,0.009999634025090777

50,B2Zi,1254.002153316236,0.009999055573122606

51,B8Xi,1667.001954568516,0.009999413958788015

52,B8Yi,3038.0010785921386,0.009999880301437688

53,B8Zi,488.9977624099092,0.00999935202246111

54,B9Xi,2214.993602906973,0.009999072309933504

55,B9Yi,2506.003104893984,0.009999756322524233

56,B9Zi,460.9917098422195,0.009999071248749724

57,A32Xi,-759.9860751912686,0.009997868925721095

58,A32Yi,987.9960298743194,0.009999218051006283

59,A32Zi,-547.0244577726111,0.009998415122966349

60,A33Xi,-181.99972579399545,0.009999022253982375

61,A33Yi,1554.9982308198673,0.00999963828361255

62,A33Zi,-541.007779812752,0.009998836220797102

63,A34Xi,391.9962337582829,0.009999164042695146

64,A34Yi,2121.9991246098284,0.009999764524707664

65,A34Zi,-547.0063228444194,0.009998998305791495

66,A35Xi,975.9952435053966,0.009999391391597558

67,A35Yi,2702.99940863817,0.009999851973240867

68,A35Zi,-558.0032366302449,0.009999283826059475

69,BS8Xi,2558.992734660378,0.009998531318041244

70,BS8Yi,2177.006027587869,0.009999563229771004

71,BS8Zi,700.9885663325562,0.009998808188310155

72,AS10Xi,236.0011440434026,0.009998973132279227

73,AS10Yi,1981.9946898772273,0.00999937167035915

74,AS10Zi,1751.0026138316653,0.00999881027459042

75,AS12Xi,-380.9936993129382,0.009998680626491431

76,AS12Yi,1372.997845180642,0.009999098090579273

77,AS12Zi,1463.9961271380248,0.009998677354957453

78,AS31Xi,-867.9814414476119,0.009999002884843221

79,AS31Yi,889.0171677406566,0.00999901307628166

80,AS31Zi,-1221.9955368658143,0.009999232563890126

81,AS32Xi,839.9930657617855,0.009999520319999625

82,AS32Yi,2580.9970830693983,0.009999846684807507

83,AS32Zi,-1196.0049173972925,0.00999947402669792

84,BS11Xi,1585.0021490529202,0.00999944964277421

85,BS11Yi,3128.0013926652487,0.009999898663210416

86,BS11Zi,248.9967482820198,0.009999390341903034

87,BS17Xi,1766.9981257155125,0.009999619982285919

88,BS17Yi,2956.0014345122127,0.009999895773972802

89,BS17Zi,-795.99884511165,0.00999960634427488

90,BS20Xi,2561.985486163508,0.009998799042748295

91,BS20Yi,2178.999164487887,0.009999652551347491

92,BS20Zi,-1033.0072388263068,0.009999216307153521

93,BVS5Xi,2444.0005729879563,0.009999610636187754

94,BVS5Yi,2270.9917172963646,0.009999579394370309

95,BVS5Zi,1610.0129047642085,0.009999414851668697

96,BVS6Xi,2212.0048298027004,0.0099990231287243

97,BVS6Yi,2497.9946101469054,0.009999559227744117

98,BVS6Zi,1613.0104389518888,0.009999080443748741

99,AVS50Xi,63.007814043732544,0.009998809042843268

100,AVS50Yi,1798.999737220244,0.009999263110786276

101,AVS50Zi,1627.9970328220686,0.009998649313436102

102,AVS51Xi,345.00026806027086,0.009999291781789845

103,AVS51Yi,2074.9967768488004,0.009999618850171674

104,AVS51Zi,1623.0038730974168,0.009999136257463292

105,AVS57Xi,-512.9919764967757,0.00999840963347474

106,AVS57Yi,1229.004036019933,0.009999084802026513

107,AVS57Zi,1229.989063478223,0.009998540792827512

108,AVS58Xi,60.00591542750996,0.00999882019331969

109,AVS58Yi,1792.0026751331404,0.009999472846839545

110,AVS58Zi,1235.9924854883006,0.009998650982191343

111,AVS59Xi,629.9975096556891,0.009999411481711865

112,AVS59Yi,2360.997013573541,0.00999979769103441

113,AVS59Zi,1247.9997148928799,0.009999280598017624

114,AVS65Xi,-795.9941081630642,0.009998783143265131

115,AVS65Yi,945.0070449316535,0.009999186683921562

116,AVS65Zi,775.9890528345459,0.009998893654553687

117,AVS69Xi,357.99743878907054,0.009999097447533556

118,AVS69Yi,2092.996963763993,0.009999743773754897

119,AVS69Zi,783.9920618728692,0.009998915700155703

120,AVS70Xi,640.9973395401622,0.009999431330621648

121,AVS70Yi,2370.997766167578,0.009999859314952262

122,AVS70Zi,776.9969183194466,0.00999930724372695

123,AVS71Xi,955.9993903681435,0.009999498125767309

124,AVS71Yi,2681.999363175862,0.009999893252210092

125,AVS71Zi,796.9983078146444,0.009999406470394318

126,AVS75Xi,-793.9940018408455,0.009998832682146196

127,AVS75Yi,946.0046913097259,0.009999378304126495

128,AVS75Zi,401.99034816130734,0.009998943331262528

129,AVS91Xi,1318.0017500806543,0.00999967489124806

130,AVS91Yi,3040.000171076009,0.009999957606806697

131,AVS91Zi,-5.0006544505552055,0.00999964830748661

132,BVS10Xi,1940.9934647893533,0.009999259239719425

133,BVS10Yi,2766.010781954679,0.00999975452047458

134,BVS10Zi,1252.967849463882,0.00999922135304634

135,BVS16Xi,2746.99394066266,0.009997942950225427

136,BVS16Yi,1983.0051442863255,0.009999322973194405

137,BVS16Zi,878.9959044616988,0.009998680845437754

138,BVS35Xi,2239.9933257998878,0.009999332998929834

139,BVS35Yi,2480.000150915669,0.00999986042383599

140,BVS35Zi,-400.0038746445866,0.009999404664370874

141,BVS36Xi,1598.076452835577,0.9998680080650479

142,BVS36Yi,3113.6452739764823,2.3268757051343965

143,BVS36Zi,-359.7148769806604,0.946509849727181

144,BVS37Xi,1738.9965096344895,0.00999939832723616

145,BVS37Yi,2973.0001008211634,0.009999857886862343

146,BVS37Zi,-614.0072670590386,0.009999353101929996

147,BVS38Xi,2765.981755378779,0.009998170782131097

148,BVS38Yi,1967.9967079192475,0.009999560237218746

149,BVS38Zi,-810.0165295450636,0.009999041575945167

150,BVS42Xi,1604.9985524736921,0.009999449272878571

151,BVS42Yi,3101.001021402925,0.009999865473064733

152,BVS42Zi,-868.0044984295388,0.00999941307065333

153,BVS43Xi,2494.9890300737125,0.009998924147661304

154,BVS43Yi,2234.0000251602396,0.00999964356473114

155,BVS43Zi,-1205.0047318143609,0.009999278011223488

156,BVS44Xi,1966.9947804334222,0.009999512549416938

157,BVS44Yi,2747.0024452599255,0.009999836260761397

158,BVS44Zi,-1191.997409056887,0.009999543224801287

159,AVS104Xi,-760.9851755729808,0.009999062886828513

160,AVS104Yi,983.0116957494113,0.009999270571548409

161,AVS104Zi,-897.9943763319072,0.009999140095608235

162,AVS105Xi,-475.4333866146429,0.5973076637059802

163,AVS105Yi,1261.6945245267584,1.2157348495487883

164,AVS105Zi,-876.1208992979716,0.8471598341952145

165,AVS106Xi,-191.9982708081328,0.009998670786603605

166,AVS106Yi,1542.9991170715296,0.009999408908042408

167,AVS106Zi,-895.0056670573017,0.009998574306189978

168,AVS107Xi,100.99825458398954,0.00999931242645576

169,AVS107Yi,1833.998574695271,0.00999972073352705

170,AVS107Zi,-885.0056033625405,0.009999217548181397

171,AVS108Xi,391.10721170433413,0.7778228806010511

172,AVS108Yi,2122.8149919404505,1.6254523007706978

173,AVS108Zi,-912.1110880088877,0.8961776950297774

174,AVS116Xi,392.9882978221536,0.009999430451317508

175,AVS116Yi,2128.9955077359587,0.009999735987558994

176,AVS116Zi,-1352.0012546863063,0.009999371682493405

177,AVS117Xi,952.9937926320332,0.00999953444201709

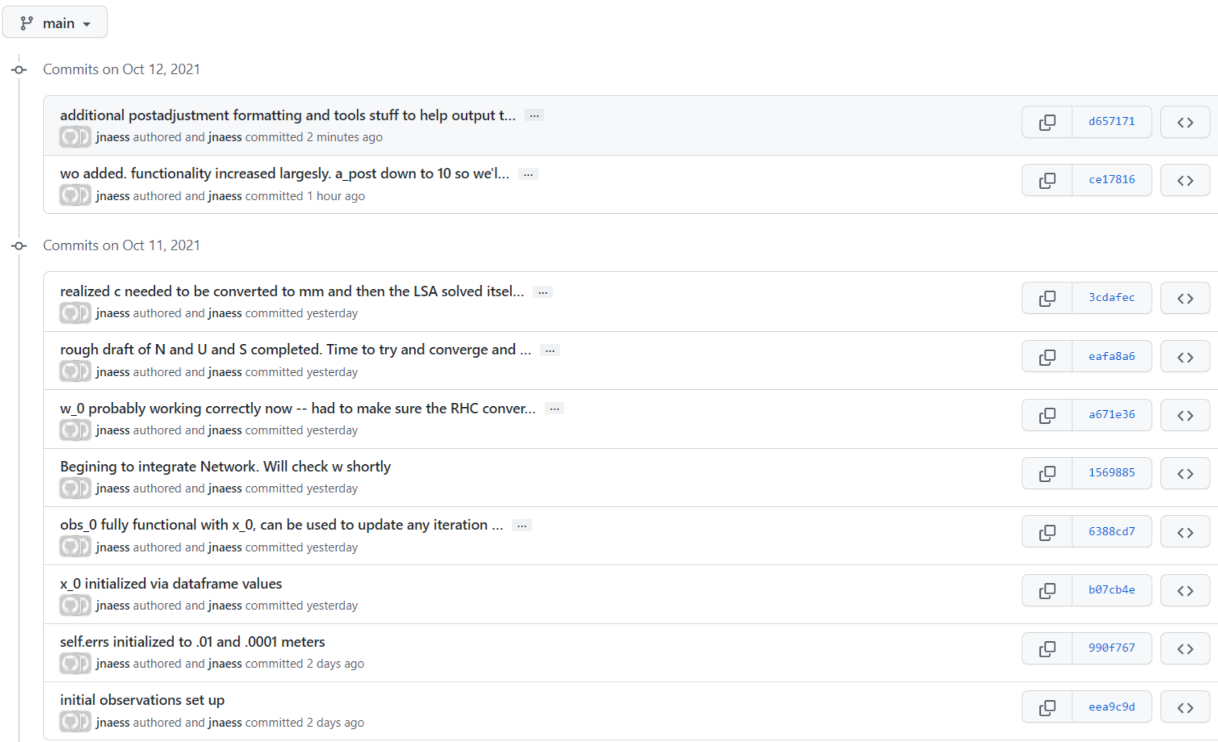
178,AVS117Yi,2682.996924446994,0.009999848758447836

179,AVS117Zi,-1334.0055585488697,0.009999505986941378

Github Repository

https://github.com/jnaess/ENGO531.git

Github Commits



Commits on Oct 10, 2021

Finally got the Ae working, ended up just being a degrees to radians ... …
jnaess authored and jnaess committed 2 days ago — 7c4e2a9

Commits on Sep 21, 2021

U and W value functions generated. Need to integrate into a class for... …
jnaess authored and jnaess committed 21 days ago — 5655a33

forgot to save
jnaess authored and jnaess committed 21 days ago — 6a22908

Conversion function from pixel to RHC made. M matrix function also made
jnaess authored and jnaess committed 21 days ago — e496ab4

Commits on Sep 18, 2021

Added LS and FileReader for modulated inputs. Must work on deriving c... …
jnaess authored and jnaess committed 24 days ago — d0f6d1a

int and pho read in functions constructed. Also implemented delim_whi... …
jnaess authored and jnaess committed 24 days ago — 14c5355

Commits on Sep 15, 2021

read_tie, read_ext completed and read.int in construction. Will need ... …
jnaess authored and jnaess committed 27 days ago — 78436d6

D2L Files
jnaess committed 28 days ago — Verified — f3f4f9d

Initial commit
jnaess committed 28 days ago — Verified — c43ea01

jnaess authored and jnaess committed 14 days ago

Update coords.txt
caitlynmaida committed 14 days ago

Update angles.txt
caitlynmaida committed 14 days ago

Update coords.txt
caitlynmaida committed 14 days ago

Update distances.txt
caitlynmaida committed 14 days ago

Update distances.txt
caitlynmaida committed 14 days ago

Update and rename distance.txt to distances.txt
caitlynmaida committed 14 days ago

Update angles.txt
caitlynmaida committed 14 days ago

Add files via upload
caitlynmaida committed 14 days ago

updated from 501 stuff
jnaess authored and jnaess committed 14 days ago

Initial commit
jnaess committed 14 days ago

Newer Older

Tools.py

```
import numpy as np
import matplotlib.pyplot as plt
import os

class Tools():
    """
    Desc:
        This class was made as a toolbox for plotting and converting values
    """
    def __init__(self):
        """
        Just exsists :-)
        """

    def plot_mat(self, matrix, title = "Title", round_to = 6):
        """
        Desc:
            Checks to see if a "Figures" folder has been made. If it is not made then it makes it.
            Then saves the input matrix as a .png to the folder with "Title" as the name
        Input:
            matrix: the numpy array to plot
            title: the title of the array and output image (default "Title")
            round_to: decimals to round to (default 6)
        Output:
        """
        #set up figure with decently sized boxes
        fig, ax = plt.subplots(figsize = (10,15))
        ax.imshow(matrix)

        plt.title(title)

        # Loop over data dimensions and create text annotations.
        for i in range(matrix.shape[0]):
            for j in range(matrix.shape[1]):
                #inputs numerical values
                text = ax.text(j, i, round(matrix[i, j],round_to),
                               ha="center", va="center", color="w")

        #plt.axis('off')

        #folder is just called figures
        folder_path = 'Figures/'
        file_name = title

        #makes folder if not already there
        if not os.path.isdir(folder_path):
            os.makedirs(folder_path)

        #saves to the folder using the title name
        fig.savefig(os.path.join(folder_path,file_name))

        plt.figure().clear()
        plt.close()
        plt.cla()
        plt.clf()

    def save_df(self, df, title):
        """
        Desc:
            saves df to the file
        Input:
        Output:
        """
        #folder is just called files
        folder_path = 'Files/'
        file_name = title
```

```
        #makes folder if not already there
        if not os.path.isdir(folder_path):
            os.makedirs(folder_path)

        #saves to the folder using the title name
        df.to_csv(os.path.join(folder_path,file_name))Tables.py
from scipy import misc
from scipy import stats
import pandas as pd
import numpy as np

class Tables():
    """
    Parent class to PostAdjustmentTester which generates the significant values to increase modularity
    """
    def __init__(self):
        """
        """

    def newtons_method(self, x, tolerance=0.0001):
        while True:
            x1 = x - self.f(x) / misc.derivative(self.f, x)
            t = abs(x1 - x)
            if t < tolerance:
                break
            x = x1
        return x

    def f(self, x):
        return 1 - stats.chi2.cdf(x, self.r) - self.pvalue

    def x_2(self):
        """
        Reference:
            Code reformatted to return a single line of the desired x_2 value based on our DOF (instead of a given value)
            Code refers to functions "newtons_method", "f", "x_2"
            https://moonbooks.org/Articles/How-to-create-a-Chi-square-table-using-python-/
        Desc:
            returns a chi-square dataframe row for the designated DOF
        Input:
            r: defrees of freedom
        Output:
        """
        self.pvalueList = [0.995, 0.99, 0.975, 0.95, 0.90, 0.10, 0.05, 0.025, 0.01, 0.005]
        results = []
        for i in range(self.r,self.r+1):
            self.r = i
            Result = []
            for self.pvalue in self.pvalueList:
                x0 = self.r  # x0 approximation
                x = self.newtons_method(x0)
                Result.append(x)
            for i in range(10):
                Result[i] = round(Result[i],3)
            results.append(Result)
        return pd.DataFrame(results, columns = self.pvalueList)
```

Net.py

```
from numpy import transpose as t
from numpy import matrix as mat, matmul as mm

from numpy import linalg as lin
from numpy.linalg import inv
import math as m
import numpy as np
import pandas as pd
from LeastSquares import LS
from Level import Delta
from PostAdjustmentTester import PostAdjustmentTester
```

```
class Network(LS, PostAdjustmentTester):
    """
    Build to run the least squares adjustment and set up the overall network
    """
    def __init__(self, models, net_type = "Photo"):
        """
        Desc:
        Input:
            models: list of models that have been initialized with
                data. Must contain the same number of columns in their a
                matrix (predefined by LS())
        Output:
        """
        LS.__init__(self)
        PostAdjustmentTester.__init__(self)

        #for picking things
        self.net_type = net_type

        self.models = models

        if self.net_type == "Photo":
            #_________________setup first round of stuff______________________
            self.initialize_variables()

            #_________________begin LSA______________________
            self.photo_LSA()

            #_________________format matrices for outputting statistics______________________
            self.photo_mats()

        #_________________output statistics______________________
        self.final_matrices()

    def initialize_variables(self):
        """
        Desc:
            initializes major variables (combining matrices and stuff)

        Input:
        Output:
           self.u
        """
        self.ue = self.models[0].ue
        self.uo = self.models[0].uo
        self.u = self.models[0].ue + self.models[0].uo

        #set up observation matrix
        temp = []
        for obs in self.models:
            temp.append(obs.obs)
        self.obs = np.vstack(temp)

        #set up errors matrix
        temp = []
        for obs in self.models:
            temp.append(obs.errs)
        self.errs = np.vstack(temp)

        if self.net_type == "Photo":
            #set up control weight errors matrix
            temp = []
            for obs in self.models:
```

```
            temp.append(obs.errs_o)
        self.errs_o = np.vstack(temp)

    #set up number of observations variable
    self.n = len(self.errs)

    #set up design matrix
    self.design()

    #set up covariance (no additional formatting needed)
    self.covariance()

    #set up apriori
    self.apriori = 1

    #set up weight matrix
    self.P = self.apriori**2 * inv(self.Cl)

    if self.net_type == "Photo":
        #then a Po will also need to be made
        self.Po = mat(np.zeros((self.uo, self.uo)))

        for i in range(0,self.uo):
            if self.errs_o[i] != 0:
                self.Po[i,i] = 1/self.errs_o[i]**2


def final_matrices(self):
    """
    Desc:
        Once the LSA is completed then this generates all desired matrices for analysis
    Input:
    Output:
        self.r_hat: residuals
        self.l_hat: adjusted observations
        self.a_post: a-posteriori variance factor
        self.uvf: unit variance factor
        self.Cx (also Cs):
        self.Cl:
        self.Cr:
    """

    self.r_hat = mm(self.A,self.S_hat) + self.w_0
    self.l_hat = self.obs + self.r_hat
    self.a_post = m.sqrt(mm(t(self.r_hat),mm(self.P,self.r_hat)/(self.n-self.u))[0,0])
    self.uvf = self.a_post**2 / self.apriori**2

    self.Cx = self.a_post**2 * inv(mm(t(self.A),mm(self.P,self.A)))
    if self.net_type == "Photo":
        self.Cx = inv(self.N)
    #self.plot_mat(self.Cx, "Covariance Matrix of Unknowns")

    self.Cl = mm(self.A,mm(self.Cx,t(self.A)))
    #self.plot_mat(self.Cl, "Covariance Matrix of Measurements")

    self.Cr = self.a_post**2*inv(self.P)-self.Cl
    #self.plot_mat(self.Cr, "Covariance Matrix of Residuals")

def nonlinear_LSA(self):
    """
    Desc:
        Iterates a nonlinear LSA, checking whether criterea was met. Once it was met then it constructs the final matrices for analysis
    Input:
    Output:

    """
    self.not_met = True
```

```
        i = 0

        self.w_0 = mat(np.zeros((self.n, 1)))
        self.S_hat = mat(np.zeros((self.n, 1)))
        self.x_hat = mat(np.zeros((self.n, 1)))

        while self.not_met:
            i = i + 1
            #print("LSA iteration: " + str(i))
            #print("x_0: ")
            #print(LS.x_0)

            #l_0
            self.obs_0()

            #update l_0 and A
            self.update_values()

            #misclosure

            self.w_0 =  self.l_0 - self.obs

            #S_hat

            self.S_hat = -mm(inv(mm(t(self.A),mm(self.P,self.A))),mm(t(self.A),mm(self.P,self.w_0)))

            #print("l_0: ")
            #print(self.l_0)

            #x_hat
            self.x_hat = LS.x_0 + self.S_hat


            #update x_0
            LS.x_0 = self.x_hat

            #print("S_hat:")
            #print(self.S_hat)
            #print("x_hat: ")
            #print(self.x_hat)
            #print("A: ")
            #print(self.A)





            self.convergence(i)

        #print("LSA passed in: " + str(i) + " iterations")
        #self.final_matrices()

    def photo_LSA(self):
        """
        Desc:
            Iterates a nonlinear LSA, checking whether criterea was met. Once it was met then it constructs the final matrices for analysis
        Input:
        Output:

        """
        self.not_met = True

        i = 0


        self.w_0 = mat(np.zeros((self.n, 1)))
        self.S_hat = mat(np.zeros((self.n, 1)))
```

```
        self.x_hat = mat(np.zeros((self.n, 1)))

        while self.not_met and i < 5:
            i = i + 1

            #l_0
            self.obs_0()

            #update l_0 and A
            self.update_values()

            #misclosure
            self.w_0 =  self.l_0 - self.obs

            self.set_N()
            self.set_U()

            #S_hat
            self.S_hat = -mm(inv(self.N),self.U)

            #x_hat
            self.x_hat = LS.x_0 + self.S_hat

            #update x_0
            LS.x_0 = self.x_hat

            self.convergence(i)

        print("LSA passed in: " + str(i) + " iterations")
        #self.final_matrices()

        #not 100% sure but probably
        self.A = self.N

    def error_ellipses(self):
        """
        Desc:
            generates the error ellipses, minor, major, bearing_major
            **must already have self.Cx generates**
            **assumes Xa, Ya, Xb, Yb, Xc, Yc etc in the Cx diagonal**
        Input:
        Output:
        """
        self.u
        ellipses = []
        for i in range(0,self.Cx.shape[0],2):
            q11 = self.Cx[i,i]
            q12 = self.Cx[i,i+1]
            q21 = self.Cx[i+1,i]
            q22 = self.Cx[i+1,i+1]
            ellipses.append(self.ellipse(q11, q12, q21, q22))

        return ellipses

    def ellipse(self, q11, q12, q21, q22):
        """
        Desc:
            Calculates the error ellipse, returns back a dataframe of the values
        Input:
            q11,
            q12,
            q21,
            q22
        Output:
            {
            "minor": float,
            "major": float,
            "major_orientation": radians
```

```
        }
    """
    minor = m.sqrt(abs((q11 + q22 - m.sqrt((q11-q22)**2+4*(q12**2)))/2))
    major = m.sqrt(abs((q11 + q22 + m.sqrt((q11-q22)**2+4*(q12**2)))/2))

    major_orientation = m.atan(q12/(major**2-q22))

    return {
        "minor": minor,
        "major": major,
        "major_orientation": major_orientation
        }



def convergence(self,i):
    """
    Desc:
        Checks based on this criterea, if convergence is met then sets self.not_met to False
    Input:
        i: number of iterations (for simple # of ter break)
    Output:
        self.not_met --> False if the criterea is met
    """
    #max 10 iterations
    if i > 3:
        self.not_met = False

    #minimum self.S_hat to be under .001m

    not_under = False
    for key in self.S_hat:
        if abs(key[0,0]) > .0001:
            #this means the criterea was not met for atleast one of the unknowns
            not_under = True

    if not not_under:
        #then all things were under .0001m in change and therefore the criterea was met
        self.not_met = False


def covariance(self):
    """
    Desc:
        Initialized covariance matrix based on observation standard deviations
    Input:
    Output:
       self.Cl
    """
    self.Cl = mat(np.zeros((self.n, self.n)))

    for i in range(0,self.n):
        self.Cl[i,i] = self.errs[i]**2


def photo_mats(self):
    """
    Desc:
        Sets up matrices needed for statistics
    Input:
    Output:
       self.A
       self.S
    """
    self.A = np.concatenate((self.Ae,self.Ao), axis = 1)
    self.u_list = []
    for i in self.models[0].u_list_ae:
```

```
        self.u_list.append("{}Xcj".format(i))
        self.u_list.append("{}Ycj".format(i))
        self.u_list.append("{}Zcj".format(i))
        self.u_list.append("{}w".format(i))
        self.u_list.append("{}o".format(i))
        self.u_list.append("{}k".format(i))

    for i in self.models[0].u_list_ao:
        self.u_list.append("{}Xi".format(i))
        self.u_list.append("{}Yi".format(i))
        self.u_list.append("{}Zi".format(i))


def design(self):
    """
    Desc:
        Set up overall design matrix
    Input:
    Output:
       self.A
    """
    #self.A = mat(np.zeros((self.n, self.u)))

    #temp = []
    #for model in self.models:
    #    temp.append(model.A)
    #self.A = np.vstack(temp)
    self.Ae = self.models[0].Ae
    self.Ao = self.models[0].Ao


def set_N(self):
    """
    Desc:
        Sets up the N matrix with the four quadrants
    Input:
        self.Ae
        self.Ao
        self.P
    Output:
        self.Nee
        self.Neo
        self.Noo
        self.N
    """
    self.Nee = mm(t(self.Ae),mm(self.P,self.Ae))
    self.Neo = mm(t(self.Ae),mm(self.P,self.Ao))
    self.Noo = mm(t(self.Ao),mm(self.P,self.Ao))+self.Po

    a = np.concatenate((self.Nee,self.Neo), axis = 1)
    b = np.concatenate((t(self.Neo),self.Noo), axis = 1)

    self.N = np.concatenate((a,b), axis = 0)

def set_U(self):
    """
    Desc:
        Sets up the U matrix with the two halves
    Input:
        self.Ae
        self.Ao
        self.P
        self.w_0
    Output:
        self.Ue
        self.Uo
        self.U
    """
```

```
        #self
        self.w_0_o = LS.x_0_ao - self.x_0[self.ue:,0]

        self.Ue = mm(t(self.Ae),mm(self.P,self.w_0))
        self.Uo = mm(t(self.Ao),mm(self.P,self.w_0))+mm(self.Po,self.w_0_o)
        #print(self.Uo.shape)
        self.U = np.concatenate((self.Ue,self.Uo), axis = 0)


    def n_mat(self):
        """
        """
        self.N = mm(t(self.A),mm(self.P,self.A))

    def cx_mat(self):
        """
        """
        self.Cx = inv(self.N)

    def w_mat(self):
        """

        """
        #adds constants and unknowns together and solves for values
        self.w = mm(self.A,LS.x_0) - self.obs

          #____________for non linear this will need to change______
    def u_mat(self):
        """
        """
        self.v = t(self.A,mm(self.P,self.w))

    def correction(self):
        """
        """
        self.S = -mm(inv(self.N),mm(t(self.A),mm(self.P,self.w)))

    def obs_0(self):
        """
        Desc:
            Assembles l_obs from each matrix
        Input:
        Output:
            self.l_0 constructed
        """

        self.l_0 = mat(np.zeros((self.n, 1)))

        temp = []
        for obs in self.models:
            temp.append(obs.l_0)
        self.l_0 = np.vstack(temp)

    def update_values(self):
        """
        Desc:
            Updates x_0 and design and l_0
        Input:
            Uses most recent x_hat value
        Output:
            none:
        """
        #update models
        for model in self.models:
            #model.x_0 = self.x_0

            model.obs_0()
```

```
            #update design matrix
            model.set_design()

        #update within network
        self.design()
        self.obs_0()
```

LeastSquares.py

```
from numpy import transpose as t
from numpy import matrix as mat, matmul as mm
import math as m
import numpy as np
import pandas as pd
from Tools import Tools

class LS(Tools):
    """
        Holds the universal values needed to integrate the different LS adjustments into one
    """
    x_0 = []
    def __init__(self, file_name = "coords.txt", debugging = False):
        """
        Desc:
            reads in the list of knowns and unknowns and assigns their values. Will construct design matrix, etc. based off of these
        Input:
            file_name where the knowns and unknowns are defined
            debugging, T/F. If true then more printing of stuff happens
        Output:
            sets up u_list (predefined in here)
            sets up number of unknowns (self.u)

        """
        #brings in the tool files for use
        Tools.__init__(self)

        self.debugging = debugging
        self.file_name = file_name
        #self.read_2D()

    def read_2D(self):
        """
        Desc:
            reads in the 2D set of points and assigns values
            expects format of [name easting northing known/unknown]
            more specifically: [Point X[m] Y[m] Known[n]/Unknown[u]]
        Input:
            self.file_name
        Output:
            self.u_list (string list of unknown)
            self.x_0 (initial guesses of unknowns)
            self.c (constant values of knowns)
            self.datums (string list of knowns)
            self.u # of unknowns
        """
        df = pd.read_csv(self.file_name, sep = ' ')
        #currently only formatted for 2D

        self.u_list = []
        LS.x_0 = []
        self.c = []
        #pretty sure datums aren't actually used
        self.datums = []

        #assign values
        for index, row in df.iterrows():
            #check if known or unknown
            if row[3] == "u":
                #unknown name
                self.u_list.append(row[0]+"_E")
```

```
            self.u_list.append(row[0]+"_N")

            #add unknown values in order of x, y
            LS.x_0.append(row[1])
            LS.x_0.append(row[2])
        else: #then they are "n" --> knowns
            #known name
            self.datums.append(row[0]+"_E")
            self.datums.append(row[0]+"_N")

            #add known values in order of x, y
            self.c.append(row[1])
            self.c.append(row[2])

    LS.x_0 = t(mat(LS.x_0))
    self.c = t(mat(self.c))
    self.u = len(self.u_list)

def find_col(self, dimension, point_name, li = "u"):
    """
    Desc:
        returns the column index of the desired points
        expects 'n' for known and 'u' for unknown
        **all values must be in caps**
    Input:
        u_list, list of strings of "pointname_dimension"
        dimension, string either "N", "E", "H"
    Output:
        integer value of the column to place the value
        in the desired design matrix
    """
    if li == "u" :
        li = self.u_list
    else:
        li = self.datums

    index = 0
    for key in li:
        #split the key into point name and dimension
        temp_name = key.split('_')[0]
        temp_dimension = key.split('_')[1]
        if (point_name == temp_name and dimension == temp_dimension):
            return index
        else:
            index = index + 1

    #debugging stuff
    if self.debugging:
        print(point_name + " Could not be found")
    return -1

def set_col_list_ae(self):
    """
    Desc:

        Initializes the order of image_id's for the Ae matrix so that numbers are positioned correctly
    Input:
    Output:
        self.u_list_ae for Ae
    """
    #assumes images already sorted in ascending order
    self.u_list_ae = self.pho['image_id'].unique()

def find_col_ae(self, image_id, li = "u"):
    """
    Desc:
        returns the column index of the desired points
        expects 'n' for known and 'u' for unknown
```

```
            **all values must be in caps**
        Input:
            u_list_ae, list of strings of "pointname_dimension"
            image_id: string of the image id index to return
        Output:
            integer value of the column to place the value in the desired design matrix multiplied by 6
        """

        if li == "u" :
            li = self.u_list_ae
        else:
            li = self.datums

        index = 0
        for key in li:
            if image_id == key:
                return index*6
            else:
                index = index + 1


    def set_col_list_ao(self):
        """
        Desc:

            Initializes the order of point_id's for the Ao matrix so that numbers are positioned correctly from all points observed (unique values for
columns)
        Input:
        Output:
            self.u_list_ao for Ao
        """
        #assumes images already sorted in ascending order
        self.u_list_ao = self.obj['point_id'].unique()

    def find_col_ao(self, point_id, li = "u"):
        """
        Desc:
            returns the column index of the desired points
            expects 'n' for known and 'u' for unknown
            **all values must be in caps**
        Input:
            u_list_ao, list of strings of "pointname_dimension"
            point_id: string of the image id index to return
        Output:
            integer value of the column to place the value in the desired design matrix multiplied by 3 for XYZ
        """

        if li == "u" :
            li = self.u_list_ao
        else:
            li = self.datums

        index = 0
        for key in li:
            if point_id == key:
                return index*3
            else:
                index = index + 1
```

PostAdjustmentTester.py

```
from numpy import transpose as t
from numpy import matrix as mat, matmul as mm
import matplotlib as plt

from numpy import linalg as lin
from numpy.linalg import inv
import math as m
import numpy as np
```

```
import pandas as pd
from LeastSquares import LS
from Level import Delta
from Tables import Tables

from scipy import stats as st
from scipy.stats import t as stu
from scipy.stats import chi2



class PostAdjustmentTester(Tables):
    """
    Desc:
        Assumes that the LSA has been conducted and outputs results for post adjustment tests
    """
    def __init__(self):
        """
        Desc:
            Figuring out if we need to take in matrices or if we'll just inherit the class and assume that they're build
        """
        Tables.__init__(self)

    def global_a_posteriori(self, alpha = .05):
        """
        Desc:
            Tests the statistical sifnigicance of the aposteriori to a priori variance factor
        Input:
            alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values,
otherwise they won't be found :-)
            self.u: # of unknowns
            self.n: # of observations
            self.a_post: final computed a posteriori variance factor
            self.apriori: initial apriori variance factor
        Output:
            Prints the output and respective indication
        """
        #set up DOF (r)
        self.r = self.n - self.u

        #retrieves dataframe of chi values for our respective DOF
        ch_df = self.x_2()

        low = ch_df[alpha][0]
        high = ch_df[1-alpha][0]

        y = (self.r * self.a_post**2)/self.apriori**2

        #if fails this check then there is an indication that the residuals or math model may be off


        print("{} tested with chi_square boundries of {} and {}".format(y, low, high))
        if y > low and y < high:
            print("Global A-Posteriori Variance Factor Test passes at a {} confidence level".format((1 - alpha)*100))
            print("There is no indication for errors within residual or the math models")
        else:
            print("Global A-Posteriori Variance Factor Test **failed** at a {}% confidence level".format((1 - alpha)*100))
            print("There is indication that errors exsist within residual or the math models")

    def significance_estimated_param(self, alpha = .05):
        """
        Desc:
            Determines whether there is statistical signifiance to beleive the final estimated value of parameters
        Input:
            alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values,
otherwise they won't be found :-)
            self.n: # of observations
            self.x_hat
```

```
        self.u_list: for labelling
        self.Cx: for extracting std dev values of parameters
    Output:
        retrunds dataframe of values [Unknown	Final Value	Value Standard Deviation	Test Value	Indicated
Significance	Alpha Tested	Confidence Level	Test Bounds]
    """
    #set up DOF (r)
    self.r = self.n - self.u

    high = stu.ppf(1.0 - alpha, self.r)
    low = stu.ppf(alpha, self.r)

    #final paramter values
    xs = []

    #unknown names in string format
    us = []

    #list to store their signifiance as Signifiance or Not Significant
    sig = []

    #list to store the value that was checked
    sig_value = []

    #list of standard deviation values
    std = []

    #test values
    y = []

    #confidence levels
    conf = []

    #confidence levels
    alphs = []

    #test bounds
    bounds = []


    for i in range(0,self.u):
        std.append(m.sqrt(self.Cx[i,i]))

        y.append((self.x_hat[i]/std[i])[0,0])

        if y[i] > low and y[i] < high:
            #if fails then there IS statistical significance
            sig.append("No")
        else:
            sig.append("Yes")

        xs.append(self.x_hat[i][0,0])
        us.append(self.u_list[i])
        conf.append((1-alpha)*100)
        alphs.append(alpha)
        bounds.append(str([low, high]))
    #to store values in a dictionary before conversion to dataframe
    dict_list = {
            "Unknown": us,
            "Final Value": xs,
            "Value Standard Deviation": std,
            "Test Value": y,
            "Indicated Significance": sig,
            "Alpha Tested": alphs,
            "Confidence Level": conf,
            "Test Bounds": bounds
        }
    #return dict_list
```

```
        return pd.DataFrame.from_dict(dict_list)

    def semi_global_residuals(self, alpha = .05):
        """
        Desc:
            Conducts the semi global test on residuals, also known as the gooness-of-fit or normality test on residuals
        Input:
            self.r_hat: residuals
            alpha = .05: to find confidence level
            self.Cr: extracting std of residuals
            self.n: number of observations
        Output:
            prints whether the test passed and the reccomended interpretation
        """
        #normalize residuals
        norm_r = []

        for i in range(0,self.n):
            norm_r.append(self.r_hat[i,0]/m.sqrt(self.Cr[i,i]))

        #number of bins
        M = round(m.sqrt(self.n))

        counts, bins = np.histogram(norm_r, bins = M)

        #compute estimated number of residuals per bin
        e = []
        for i in range(M):
            #get probability of total bin
            p_start = st.norm.cdf(bins[i])
            p_end = st.norm.cdf(bins[i+1])
            p = p_end - p_start

            #append total number of expected residuals
            e.append(p*self.n)

        #compute X_2 for each bin
        chis = []
        for i in range(M):
            chis.append((e[i]-counts[i])**2/e[i])

        #sum all chis for test statistic y
        y = sum(chis)

        #conduct statistical test
        dof = M - 1
        prob = 1 - alpha
        chi = chi2.ppf(prob, dof)

        print("{} tested with chi_square of {} ".format(y, chi))
        if y > chi:
            print("The Semi-Global, goodness-of-fit test on the residuals **Failled**")
            print("There is a sign that either there are outliers or the functional model was not appropriate for the data set")
        else:
            print("The Semi-Global, goodness-of-fit test on the residuals **Passed**")
            print("There is no sign of outliers or functional model errors")
        #plt.hist(norm_r, m)

    def blunder_detection(self, alpha = .01):
        """
        Desc:
            Conducts the local test on the residuals, aka blunder detection
        Input:
            alpha = .01: for 99% confidence of a blunder
            self.Cr: for extracting std of residuals
            self.r_hat: for extracting residuals
        Output:
            Returns a dataframe with columns ["Observation", "Outlier", "Test Value", "Test Bounds"]
```

```
        """
        #statistical test values
        low = st.norm.ppf(alpha/2)
        high = st.norm.ppf(1-alpha/2)

        #normalize residuals (test statistic)
        y = []

        #list of Yes or No outliers
        outlier = []

        #confidence levels
        conf = []

        #observations
        observations = []

        #test bounds
        bounds = []

        for i in range(0,self.n):
            #for DF
            observations.append(i)
            bounds.append(str([low, high]))
            conf.append((1-alpha)*100)

            y.append(self.r_hat[i,0]/m.sqrt(self.Cr[i,i]))

            if y[i] > low and y[i] < high:
                #passes test --> not an outlier
                outlier.append("No")
            else:
                outlier.append("Yes")

        dic = {
            "Observation": observations,
            "Outlier": outlier,
            "Confidence Level": conf,
            "Test Value": y,
            "Test Bounds": bounds
        }
        return pd.DataFrame.from_dict(dic)


    def final_file(self, alpha = .05):
        """
        Desc:
            Final Dataframe File
        Input:
            alpha: to generate the two confidence intervals. Be sure to make sure that these values are generated in the respective dataframe of values,
otherwise they won't be found :-)
            self.n: # of observations
            self.x_hat
            self.u_list: for labelling
            self.Cx: for extracting std dev values of parameters
        Output:
            retrunds dataframe of values [Unknown        Final Value        Value Standard Deviation        Test Value        Indicated
Significance        Alpha Tested        Confidence Level        Test Bounds]
        """
        #set up DOF (r)
        self.r = self.n - self.u

        high = stu.ppf(1.0 - alpha, self.r)
        low = stu.ppf(alpha, self.r)

        #final paramter values
        xs = []
```

```
        #unknown names in string format
        us = []

        #list to store their signifiance as Signifiance or Not Significant
        sig = []

        #list to store the value that was checked
        sig_value = []

        #list of standard deviation values
        std = []

        #test values
        y = []

        #confidence levels
        conf = []

        #confidence levels
        alphs = []

        #test bounds
        bounds = []


        for i in range(0,self.u):
            std.append(m.sqrt(self.Cx[i,i]))

            y.append((self.x_hat[i]/std[i])[0,0])

            if y[i] > low and y[i] < high:
                #if fails then there IS statistical significance
                sig.append("No")
            else:
                sig.append("Yes")

            xs.append(self.x_hat[i][0,0])
            us.append(self.u_list[i])
            conf.append((1-alpha)*100)
            alphs.append(alpha)
            bounds.append(str([low, high]))
        #to store values in a dictionary before conversion to dataframe
        dict_list = {
                "Unknown": us,
                "Final Value (mm or rad)": xs,
                "Value Standard Deviation (mm or rad)": std,
                #"Test Value": y,
                #"Indicated Significance": sig,
                #"Alpha Tested": alphs,
                #"Confidence Level": conf,
                #"Test Bounds": bounds
            }
        #return dict_list
        return pd.DataFrame.from_dict(dict_list)
```

Design_o.py

```
from numpy import matrix as mat, matmul as mm
from numpy import transpose as t
import math as m
import numpy as np
import pandas as pd
from Bundle import Bundle
from Design_e import Design_e as ae
from LeastSquares import LS

class Design_o(Bundle, LS):
    """
    Desc:
```

```
    Generates and facilitates the manipulation of Ae
"""

def __init__(self):
    """
    Desc:
    Input:
    Output:
    """
    Bundle.__init__(self)
    LS.__init__(self)

    self.initialial_setup()



def initialial_setup(self):
    """
    Desc:
        initializes major variables (combining matrices and stuff)
    Input:
    Output:
       self.u
    """
    self.xp = self.pix_to_m*self.int["xp"][0]
    self.yp = self.pix_to_m*self.int["yp"][0]
    self.c = self.pix_to_m*self.int["c"][0]

    #from LS class to find unknown columns
    self.set_col_list_ao()
    self.set_col_list_ae()

    self.set_X_0()

    self.set_obs()

    self.obs_0()

    self.set_design()


def set_obs(self):
    """
    Desc:
       uses self.pho to take the x and y and set up the observations and converts them to RHC with a bundle functions

       sets control point to .01mm and current tie points to 10mm
    Input:
        self.pho
    Output:
        self.obs: l matrix (never changes)
        self.errs
    """
    self.obs = mat(np.zeros((self.n, 1)))

    #data input as ***mm***
    self.errs = mat(np.zeros((self.n, 1)))



    #get desired numbers in a list
    y = self.pho['y'].to_list()
    x = self.pho['x'].to_list()
    check = self.pho['knowns'].to_list()

    j = 0
    for i in range(0, self.n, 2):
        #set up x_ij and y_ij info
```

```
            self.rhc(x[j],y[j])

            #if j == 0:
                #print("xp: {} | yp: {} | xmm: {} | ymm: {}".format(x[j], y[j], self.x_ij, self.y_ij))
            #x pixel
            self.obs[i,0] = self.x_ij

            #y pixel
            self.obs[i+1,0] = self.y_ij

            self.errs[i,0] = .00345
            self.errs[i+1,0] = .00345

            #assign errors
            j = j+1
        self.set_control_weights()

    def set_control_weights(self):
        """
        Desc:
            Sets control weights for datum definition
        Input:
        Output:
            self.errs_o
        """
        #for Po
        self.errs_o = mat(np.zeros((self.uo, 1)))

        #to skip the Ae ones (only pixel points wanted)
        check = self.pho['knowns'].to_list()
        j = self.ue
        for i in range(0,self.uo,3):

            #    print(str(i)+"     "+str(self.ue)+"        "+str(self.uo))
            if check[j] == "u":
                #then tie point and larger std
                self.errs_o[i,0] = 0
                self.errs_o[i+1,0] = 0
                self.errs_o[i+2,0] = 0
            else:
                #control points given extra weight
                self.errs_o[i,0] = .01
                self.errs_o[i+1,0] = .01
                self.errs_o[i+2,0] = .01

            #increment index in y and x lsits
            j = j+1

    def set_X_0(self):
        """
        Desc:
            Sets up X_0 from the dataframe values
        Input:
        Output:
            self.x_0
            and
            LS.x_0
        """
        #assumes images already sorted in ascending order
        #assumes camera also sorted
        x_0_ae = []
        for index, row in self.ext.iterrows():
            x_0_ae.append(row["Xc"])
            x_0_ae.append(row["Yc"])
            x_0_ae.append(row["Zc"])
            x_0_ae.append(m.radians(row["w"]))
            x_0_ae.append(m.radians(row["o"]))
            x_0_ae.append(m.radians(row["k"]))
```

```
        x_0_ao = []
        for index, row in self.obj.iterrows():
            x_0_ao.append(row["X"])
            x_0_ao.append(row["Y"])
            x_0_ao.append(row["Z"])

        LS.x_0_ao = t(mat(x_0_ao))

        self.x_0 = t(mat(x_0_ae+x_0_ao))
        LS.x_0 = self.x_0

    def obs_0(self):
        """
        desc:
            Sets up self.l_0 (extimated observations)
            Used for finding the current misclosure
            Assumes only one camera for IOP's from self.int
        input:
            self.x_0
        output:
            self.l_0
        """
        self.rhc(self.int["xp"][0], self.int["yp"][0])
        self.xp = self.x_ij
        self.yp = self.y_ij
        self.c = self.pix_to_m*self.int["c"][0]

        #set it up as just zeros
        self.l_0 = mat(np.zeros((self.n, 1)))

        for i in range(0, self.n, 2):
            obs = self.pho.iloc[int(i/2)]

            #row for ae parameters
            j = self.find_col_ae(obs["image_id"])
            #row for ue parameters
            j_2 = self.ue + self.find_col_ao(obs["point_id"])

            self.X_cj = LS.x_0[j]
            self.Y_cj = LS.x_0[j+1]
            self.Z_cj = LS.x_0[j+2]
            self.w = LS.x_0[j+3]
            self.o = LS.x_0[j+4]
            self.k = LS.x_0[j+5]

            #xp, yp, c values should be updated here if multiple cameras were used

            self.X_i = LS.x_0[j_2]
            self.Y_i = LS.x_0[j_2+1]
            self.Z_i = LS.x_0[j_2+2]
            #if i == 0:
                #print("xp: {} | yp: {} | c: {} | X_cj: {} | Y_cj: {} | Z_cj: {} | w: {} | o: {} | k: {} | X_i: {} | Y_i: {} | Z_i: {}".format(self.xp, self.yp,
self.c, self.X_cj, self.Y_cj, self.Z_cj, self.w, self.o, self.k, self.X_i,self.Y_i,self.Z_i))
            v = self.V()
            w = self.W()
            u = self.U()
            m_temp = self.M()

            #if i == 0:
                #print("xp: {} | yp: {} | c: {} | u: {} | w: {} | v: {}".format(self.xp, self.yp, self.c, u,w,v))
            x = self.xp - self.c*u/w
            y = self.yp - self.c*v/w

            #setup xij
            self.l_0[i,0] = x

            #set up yij
```

```
        self.l_0[i+1,0] = y

    def set_design(self):
        """
        Desc:
            Initializes the design matrix
        Output:
        Input:
        """
        self.xp = self.pix_to_m*self.int["xp"][0]
        self.yp = self.pix_to_m*self.int["yp"][0]
        self.c = self.pix_to_m*self.int["c"][0]
        self.update_Ae()

        #set it up as just zeros
        self.Ao = mat(np.zeros((self.n, self.uo)))

        #0, 2, 4, etc. are X pixels
        #1, 3, 5, etc. are Y pixels
        #__print("n: "+str(self.n))
        for i in range(0, self.n, 2):
            #increments every two because one row is for X, one row is for Y

            #each time we should go through one observation
            #indexes every 2
            #this is the observation
            #get image id from photo obs
            obs = self.pho.iloc[int(i/2)]

            #get image row from ext EOP's
            #j = int(obs["image_id"])
            j = self.find_col_ao(obs["point_id"])

            j_2 = self.find_col_ae(obs["image_id"])

            #then evens (X partial)
            self.Ao[i,j] = -self.Ae[i,j_2]
                #Y
            self.Ao[i,j + 1] = -self.Ae[i,j_2+1]
                #Z
            self.Ao[i,j + 2] = -self.Ae[i,j_2+2]

            #then odds (Y partial)
                #X
            self.Ao[i+1,j] = -self.Ae[i+1,j_2]
                #Y
            self.Ao[i+1,j + 1] = -self.Ae[i+1,j_2+1]
                #Z
            self.Ao[i+1,j + 2] = -self.Ae[i+1,j_2+2]

    def update_Ae(self):
        """
        Desc:
            Initializes the design matrix
        Input:
            LS.x_0
        Output:

        """
        self.xp = self.pix_to_m*self.int["xp"][0]
        self.yp = self.pix_to_m*self.int["yp"][0]
        self.c = self.pix_to_m*self.int["c"][0]

        #set it up as just zeros
        self.Ae = mat(np.zeros((self.n, self.ue)))

        #0, 2, 4, etc. are X pixels
        #1, 3, 5, etc. are Y pixels
```

```
        #__print("n: "+str(self.n))
        for i in range(0, self.n, 2):
            obs = self.pho.iloc[int(i/2)]

            #row for ae parameters
            j = self.find_col_ae(obs["image_id"])
            #row for ue parameters
            j_2 = self.ue + self.find_col_ao(obs["point_id"])

            self.X_cj = LS.x_0[j]
            self.Y_cj = LS.x_0[j+1]
            self.Z_cj = LS.x_0[j+2]
            self.w = LS.x_0[j+3]
            self.o = LS.x_0[j+4]
            self.k = LS.x_0[j+5]

            #xp, yp, c values should be updated here if multiple cameras were used

            self.X_i = LS.x_0[j_2]
            self.Y_i = LS.x_0[j_2+1]
            self.Z_i = LS.x_0[j_2+2]

            v = self.V()
            w = self.W()
            u = self.U()
            m_temp = self.M()


                #then evens (X partial)
                #X
            self.Ae[i,j] = -(self.c/w**2)*(m_temp[2,0]*u-m_temp[0,0]*w)
                #Y
            self.Ae[i,j + 1] = -self.c/w**2*(m_temp[2,1]*u-m_temp[0,1]*w)
                #Z
            self.Ae[i,j + 2] = -self.c/w**2*(m_temp[2,2]*u-m_temp[0,2]*w)
                #w
            self.Ae[i,j + 3] = -self.c/w**2*((self.Y_i - self.Y_cj)*(u*m_temp[2,2]-w*m_temp[0,2])
                                             -(self.Z_i - self.Z_cj)*(u*m_temp[2,1]-w*m_temp[0,1]))
                #o
            self.Ae[i,j + 4] = -self.c/w**2*((self.X_i - self.X_cj)*(-w*m.sin(self.o)*m.cos(self.k)-u*m.cos(self.o))
                                   +(self.Y_i - self.Y_cj)*(w*m.sin(self.w)*m.cos(self.o)*m.cos(self.k)-u*m.sin(self.w)*m.sin(self.o))
                                   +(self.Z_i - self.Z_cj)*(-w*m.cos(self.w)*m.cos(self.o)*m.cos(self.k)+u*m.cos(self.w)*m.sin(self.o)))
                #k
            self.Ae[i,j + 5] = -self.c*v/w

                #then odds (Y partial)
                #X
            self.Ae[i+1,j] = -self.c/w**2*(m_temp[2,0]*v-m_temp[1,0]*w)
                #Y
            self.Ae[i+1,j + 1] = -self.c/w**2*(m_temp[2,1]*v-m_temp[1,1]*w)
                #Z
            self.Ae[i+1,j + 2] = -self.c/w**2*(m_temp[2,2]*v-m_temp[1,2]*w)
                #w
            self.Ae[i+1,j + 3] = -self.c/w**2*((self.Y_i - self.Y_cj)*(v*m_temp[2,2]-w*m_temp[1,2])
                                              -(self.Z_i - self.Z_cj)*(v*m_temp[2,1]-w*m_temp[1,1]))
                #o
            self.Ae[i+1,j + 4] = -self.c/w**2*((self.X_i - self.X_cj)*(w*m.sin(self.o)*m.sin(self.k)-v*m.cos(self.o))
                                   +(self.Y_i - self.Y_cj)*(-w*m.sin(self.w)*m.cos(self.o)*m.sin(self.k)-v*m.sin(self.w)*m.sin(self.o))
                                   +(self.Z_i - self.Z_cj)*(w*m.cos(self.w)*m.cos(self.o)*m.sin(self.k)+v*m.cos(self.w)*m.sin(self.o)))
                #k
            self.Ae[i+1,j + 5] = self.c*u/w
```

Tables.py

```
from scipy import misc
from scipy import stats
import pandas as pd
import numpy as np

class Tables():
```

```
    """
    Parent class to PostAdjustmentTester which generates the significant values to increase modularity
    """
    def __init__(self):
        """
        """

    def newtons_method(self, x, tolerance=0.0001):
        while True:
            x1 = x - self.f(x) / misc.derivative(self.f, x)
            t = abs(x1 - x)
            if t < tolerance:
                break
            x = x1
        return x

    def f(self, x):
        return 1 - stats.chi2.cdf(x, self.r) - self.pvalue

    def x_2(self):
        """
        Reference:
            Code reformatted to return a single line of the desired x_2 value based on our DOF (instead of a given value)
            Code refers to functions "newtons_method", "f", "x_2"
            https://moonbooks.org/Articles/How-to-create-a-Chi-square-table-using-python-/
        Desc:
            returns a chi-square dataframe row for the designated DOF
        Input:
            r: defrees of freedom
        Output:
        """
        self.pvalueList = [0.995, 0.99, 0.975, 0.95, 0.90, 0.10, 0.05, 0.025, 0.01, 0.005]
        results = []
        for i in range(self.r,self.r+1):
            self.r = i
            Result = []
            for self.pvalue in self.pvalueList:
                x0 = self.r  # x0 approximation
                x = self.newtons_method(x0)
                Result.append(x)
            for i in range(10):
                Result[i] = round(Result[i],3)
            results.append(Result)
        return pd.DataFrame(results, columns = self.pvalueList)
```

FileReader.py

```
import numpy as np
import pandas as pd

class File_Reader():
    """
    Contains a bunch of file reading functions so that the class may be imported when desired files what to be read in
    """

    def __init__(self, tie_file = 'engo531_lab1.tie',
                     ext_file = 'engo531_lab1.ext',
                      int_file = 'engo531_lab1.int',
                      pho_file = "engo531_lab1.pho",
                 con_file = "engo531_lab1.con"
                ):
        """
        Desc:
            does not have any need to setup anything. More of just a function container
            all id's are in strings
        In:
        Out:
            self.tie: DF of tie points
            self.ext: data frame of exterior orientation parameters
            self.int: DF of interior orientation parameters
```

```
        self.pho: Dataframe of image (photo) point obs
        self.con: Df of control points
        self.obj: control and tie point dataframes
    """
    self.tie_file = tie_file
    self.ext_file = ext_file
    self.int_file = int_file
    self.pho_file = pho_file
    self.con_file = con_file

    self.con = self.read_con()
    self.tie = self.read_tie()
    self.ext = self.read_ext()
    self.int = self.read_int()
    self.pho = self.read_pho()
    self.obs_points()

def read_tie(self):
    """
    Desc:
        Reads in the tie points as returns dataframe of the values
    In:
        filename, default set to lab1 filename
    Out:
        dataframe with columns "X, Y, Z" and index not set to point_id
    """
    df = pd.read_csv(self.tie_file, sep = "\t", header = None)
    df.columns = ["point_id", "X", "Y", "Z"]
    #df = df.set_index("point_id")

    #convert all value columns to flaots
    df[["X", "Y", "Z"]] = df[["X", "Y", "Z"]].astype(float)

    #convert ID's to strings
    df[["point_id"]] = df[["point_id"]].astype(str)

    #std for tie points is 1 pixel

    return df

def read_con(self):
    """
    Desc:
        Reads in the control points as returns dataframe of the values
    In:
        filename, default set to lab1 filename
    Out:
        dataframe with columns "X, Y, Z" and index not set to point_id
    """
    df = pd.read_csv(self.con_file, sep = "\t", header = None)

    #cleaning the data
    df = df.drop(4, axis=1)

    df.columns = ["point_id", "X", "Y", "Z"]
    #df = df.set_index("point_id")

    #convert all value columns to flaots
    df[["X", "Y", "Z"]] = df[["X", "Y", "Z"]].applymap(np.float64)

    #convert ID's to strings
    df[["point_id"]] = df[["point_id"]].astype(str)

    return df

def read_ext(self):
    """
    Desc:
```

```
            Reads in the tie points as returns dataframe of the values
        In:
            filename, default set to lab1 filename
        Out:
            dataframe with columns "image_id","camera_id","Xc", "Yc", "Zc", "w", "o", "k" and index set to natural incrementation
        """
        df = pd.read_csv(self.ext_file, sep = "\t", header = None)

        #cleaning the data
        df = df.drop([8,9,10,11,12,13,14], axis=1)

        df.columns = ["image_id","camera_id","Xc", "Yc", "Zc", "w", "o", "k"]

        #convert all value columns to flaots
        #df = df[["Xc", "Yc", "Zc", "w", "o", "k"]].astype(float)
        df[["Xc", "Yc", "Zc", "w", "o", "k"]] = df[["Xc", "Yc", "Zc", "w", "o", "k"]].applymap(np.float64)

        #convert ID's to strings
        df[["camera_id"]] = df[["camera_id"]].astype(str)
        df[["image_id"]] = df[["image_id"]].astype(str)

        return df

    def read_int(self):
        """
        Desc:
            Reads in the tie points as returns dataframe of the values
            Currently only formatted for a single row. Multiple rows will need reformatting
        In:
            filename, default set to lab1 filename
        Out:
            dataframe with columns "camera_id", 'xp', 'xp', "c" and index set to natural incrementation
        """
        df = pd.read_csv(self.int_file, sep = "\t", header = None)

        #cleaning the data
        df = df.drop([0], axis=1)

        #break column 2 into the proper X, Y, Z string
        l = df.loc[0][2].split(" ")
        l.remove('')
        l = [x for x in l if x!='']
        corrected = [df.loc[0][1]] + l

        #recombine data again
        df = pd.DataFrame([corrected], columns = ["camera_id", 'xp', 'yp', "c"])

        #convert ID's to strings
        df[["camera_id"]] = df[["camera_id"]].astype(str)

        #secure flaot type numbers
        df[["c"]] = df[["c"]].astype(float)
        df[["xp"]] = df[["xp"]].astype(float)
        df[["yp"]] = df[["yp"]].astype(float)

        return df

    def read_pho(self):
        """
        Desc:
            Reads in the pho (observation) points as returns dataframe of the values
            Must have self.tie initialized
        In:
            self.tie
            filename, default set to lab1 filename
        Out:
            dataframe with columns "point_id", "image_id", "x", "y" and index set to natural incrementation
        """
```

```
        #uses mixed spacing to read in files... nbd ;-)
        df = pd.read_csv(self.pho_file, header = None, delim_whitespace =True)

        #assign column values
        df.columns = ["point_id", "image_id", "x", "y"]

        #combines point_id and image_id for a unique identifier
        df["unique_id"] = df["point_id"].to_numpy()+df["image_id"].astype(str).to_numpy()

        #convert ID's to strings
        df[["point_id"]] = df[["point_id"]].astype(str)
        df[["image_id"]] = df[["image_id"]].astype(str)

        #sort values in ascending inage_id's
        df = df.sort_values(by=['image_id'])

        #std for control points is .01mm and temporarily 10mm for tie
        temp = []
        for index, row in df.iterrows():
           # print(row['point_id'])
            if any(self.tie["point_id"] == row['point_id']):
                temp.append("u")
            else:
                temp.append('n')
        df["knowns"] = temp

        return df

    def obs_points(self):
        """
        Desc:
            Initializes the object point dataframe
            ***may bee differentiating between tie points and control points***
        Input:
            self.tie
            self.con
        Output:
            self.obj
        """
        self.obj = pd.concat([self.tie, self.con])

        #convert ID's to strings
        self.obj[["point_id"]] = self.obj[["point_id"]].astype(str)
```

Bundle.py

```
from numpy import matrix as mat, matmul as mm
from numpy import transpose as t
import math as m
import numpy as np
import pandas as pd
from LeastSquares import LS
from FileReader import File_Reader

class Bundle(LS, File_Reader):
    """
    Desc:
        Contains the LS for all LSA info
        Contains the Bundle for all Bundle Adjustment specific specs
    """

    def __init__(self):
        """
        Desc:
        Input:
        Output:
        """
        LS.__init__(self)
        File_Reader.__init__(self)
```

```
        self.initialize_variables()

    def initialize_variables(self):
        """
        Desc:
            initializes import dimensions as taken in from the File_Reader
        Input:
        Output:
            self.ue
            self.uo
            self.n
        """
        #pixel spacing (mm)
        self.pix_to_m = 3.45e-3

        #pixel spacing (mm)
        self.delta_x = 3.45e-6*1000
        self.delta_y = 3.45e-6*1000

        #normal principal distance (mm)
        self.n_p_d = 7

        #number of pixels for total columns
        self.Np = 3000

        #number of rows of pixels
        self.Mp = 4000

        self.set_ue()
        self.set_uo()
        self.set_n()

    def set_ue(self):
        """
        Desc:
            finds m from # of images and then makes ue = 6 * m
        Input:
            self.ext
        Output:
            self.ue
        """
        m = len(self.ext.index)

        self.ue = 6 * m

    def set_uo(self):
        """
        Desc:
            finds p from # of points (currently just tie) and then makes uo = 2 * p
        Input:
            maybe self.con??
            self.tie
        Output:
            self.uo
        """
        #control stuff added
        q = len(self.obj.index)

        self.uo = 3 * q

    def set_n(self):
        """
        Desc:
            finds n from total number of pixel observations
        Input:
            self.pho
        Output:
            self.n
```

```
        """
        p = len(self.pho.index)

        self.n = 2 * p

    def rhc(self, n_ij = 2015.203, m_ij = 1566.904):
        """
        Desc:
            converts from LHC to RHC
            Must be formatted to assign or return the x, y coordinates as desired
        Input:
            n_ij (number of columns for that pixel)
            m_ij (number of rows for that pixel)

        Out:
            self.x_ij
            self.y_ij
        """
        #self.x_ij = (n_ij-((self.Np/2)-.5))*self.delta_x
        #self.y_ij = (((self.Mp/2)-.5)-m_ij)*self.delta_y

        self.x_ij = (n_ij-((self.Np/2)-.5))*self.delta_x
        self.y_ij = (((self.Mp/2)-.5)-m_ij)*self.delta_y

    def M(self):
        """
        Desc:
            Generates the M rotation matrix (3x3)
            converts from LHC to RHC
            Must be formatted to assign or return the x, y coordinates as desired
        Input:
            w, in radians
            k, in radians
            o, in radians
        Out:
            none atm
        """
        o = self.o
        k = self.k
        w = self.w

        temp = mat(np.zeros((3,3)))
        #row zero
        temp[0,0] = m.cos(o)*m.cos(k)
        temp[0,1] = m.cos(w)*m.sin(k)+m.sin(w)*m.sin(o)*m.cos(k)
        temp[0,2] = m.sin(w)*m.sin(k)-m.cos(w)*m.sin(o)*m.cos(k)

        #row one
        temp[1,0] = -m.cos(o)*m.sin(k)
        temp[1,1] = m.cos(w)*m.cos(k)-m.sin(w)*m.sin(o)*m.sin(k)
        temp[1,2] = m.sin(w)*m.cos(k)+m.cos(w)*m.sin(o)*m.sin(k)

        #row two
        temp[2,0] = m.sin(o)
        temp[2,1] = -m.sin(w)*m.cos(o)
        temp[2,2] = m.cos(w)*m.cos(o)

        #testing using matrix multiplication instead
        #temp = mm(self.R3(k), mm(self.R2(o), self.R1(w)))

        #for future reference
        #w = m.atan(-temp[2,1]/temp[2,2])
        #o = m.asin(temp[2,0])
        #k = m.atan(-temp[2,1]/temp[0,0])

        return temp

    def U(self):
```

```
        """
        Desc:
        *******test values are for angles ATM************
            uses the angle values and input XYZ values to output U
        Input:
            w, in radians
            k, in radians
            o, in radians
            X_i,
            self.X_cj,
            Y_i,
            self.Y_cj,
            self.Z_i,
            self.Z_cj
        Out:
            none atm
        """
        U = self.M()[0,0]*(self.X_i-self.X_cj)+self.M()[0,1]*(self.Y_i-self.Y_cj)+self.M()[0,2]*(self.Z_i-self.Z_cj)

        return U

    def W(self):
        """
        Desc:
        *******test values are for angles ATM************
            uses the angle values and input XYZ values to output W
        Input:
            w, in radians
            k, in radians
            o, in radians
            self.X_i,
            self.X_cj,
            self.Y_i,
            self.Y_cj,
            self.Z_i,
            self.Z_cj
        Out:
            none atm
        """
        W = self.M()[2,0]*(self.X_i-self.X_cj)+self.M()[2,1]*(self.Y_i-self.Y_cj)+self.M()[2,2]*(self.Z_i-self.Z_cj)

        return W

    def V(self):
        """
        Desc:
        *******test values are for angles ATM************
            uses the angle values and input XYZ values to output W
        Input:
            w, in radians
            k, in radians
            o, in radians
            self.X_i,
            self.X_cj,
            self.Y_i,
            self.Y_cj,
            self.Z_i,
            self.Z_cj
        Out:
            none atm
        """
        V = self.M()[1,0]*(self.X_i-self.X_cj)+self.M()[1,1]*(self.Y_i-self.Y_cj)+self.M()[1,2]*(self.Z_i-self.Z_cj)

        return V

    def R1(self, o):
        """
        Desc:
```

```
        Returns R1 matrix
    Input:
        radians o
    Output:
        R1 (3x3)
    """
    temp = mat(np.zeros((3,3)))
    #row zero
    temp[0,0] = 1
    temp[0,1] = 0
    temp[0,2] = 0

    #row one
    temp[1,0] = 0
    temp[1,1] = m.cos(o)
    temp[1,2] = m.sin(o)

    #row two
    temp[2,0] = 0
    temp[2,1] = -m.sin(o)
    temp[2,2] = m.cos(o)

    return temp

def R2(self, o):
    """
    Desc:
        Returns R2 matrix
    Input:
        radians o
    Output:
        R2 (3x3)
    """
    temp = mat(np.zeros((3,3)))
    #row zero
    temp[0,0] = m.cos(o)
    temp[0,1] = 0
    temp[0,2] = -m.sin(o)

    #row one
    temp[1,0] = 0
    temp[1,1] = 1
    temp[1,2] = 0

    #row two
    temp[2,0] = m.sin(o)
    temp[2,1] = 0
    temp[2,2] = m.cos(o)

    return temp

def R3(self, o):
    """
    Desc:
        Returns R3 matrix
    Input:
        radians o
    Output:
        R3 (3x3)
    """
    temp = mat(np.zeros((3,3)))
    #row zero
    temp[0,0] = -m.cos(o)
    temp[0,1] = m.sin(o)
    temp[0,2] = 0

    #row one
    temp[1,0] = -m.sin(o)
```

```
temp[1,1] = m.cos(o)
temp[1,2] = 0

#row two
temp[2,0] = 0
temp[2,1] = 0
temp[2,2] = 1

return temp
```