Part II Computational Project

Surveying using Stars

Zixiao Hu

Word count: 2827

April 2022

Abstract

Delay data from the CHARA interferometer was analysed to solve for the positions of its telescopes and quantify any possible motion between 2012 and 2019. Singular value decomposition was used to find the least-squares solution, with uncertainties estimated experimentally by finding the day-to-day variation. The telescopes displayed collective motion in the south-east direction relative to telescope W1 in the seven-year period; the W2-W1, S1-W1, S2-W1 and E1-W1 baselines were found to have increased by 3.4 ± 0.4mm, 3.8 ± 0.6mm, 5.5 ± 0.7mm, and 7.9 ± 0.1mm respectively. Seasonal variation was observed in 2019 for the W2-W1, S2-W1, and E1-W1 baselines, with respective magnitudes 6.6 ± 0.8mm, 2.0 ± 0.6mm, 0.9 ± 0.3mm (latter in North only) and no clear directional correlation.

Contents

1. Introduction

The aim of the project was to determine locations of the 6-telescope CHARA interferometer and quantify any possible motion over long timescales, using available data on the optical path delays at each telescope. The interferometer combines light from pairs of telescopes to produce interference fringes, which requires the optical path delays from the star to the point of interference to be matched to a few microns. At each telescope, there are three primary sources of external delays:

1. Geometrical, due to the relative positions of the telescopes
2. Pipes-of-pan (POP) delays due to the discrete settings of the telescope, which can be changed
3. 'Constant term' delays due to the optical path from the telescope to the combination point

There are also random contributions due to the atmosphere, which vary on timescales of less than a second with a maximum amplitude of order 10–100 microns. Importantly, telescope positions are not the only changing variable: the POP and constant delays are known to drift up to hundreds of microns each night, and up to several millimeters over longer timescales.

In order to match the total delays at the combination point, an optical path length equaliser (OPLE) varies continuously to introduces an additional delay for each telescope. These delays are measured to sub-micron accuracy, and are available as data which can be analysed to determine the telescope positions, as well as potential estimates for the other unknown delays. The following section provides a mathematical formulation for the problem.

1.1. Mathematical formulation

Suppose that a pair of telescopes have coordinates $\boldsymbol{x_1}$, $\boldsymbol{x_2}$ relative to some origin (here taken to be one of the telescopes), some POP settings with delays P_1, P_2, constant term delays C_1, C_2, and OPLE delays E_1, E_2. Let $\boldsymbol{B_{12}} = \boldsymbol{x_1} - \boldsymbol{x_2}$ be the vector from telescope 2 to telescope 1, and $\boldsymbol{S}$ the unit vector to the star. Matching delays requires that

$$P_1 + C_1 + E_1 = \boldsymbol{B_{12}} \cdot \boldsymbol{S} + P_2 + C_2 + E_2 \tag{1}$$

$$\boldsymbol{B_{12}} \cdot \boldsymbol{S} - P_1 + P_2 - C_1 + C_2 = E_1 - E_2 \tag{2}$$

$$S_x x_1 + S_y y_1 + S_z z_1 - S_x x_2 - S_y y_2 - S_z z_2 - P_1 + P_2 - C_1 + C_2 = E_1 - E_2 \tag{3}$$

after rearranging the known terms to the right and expanding $\boldsymbol{B_{12}} \cdot \boldsymbol{S}$. Each data point from the interferometer consists of the two telescopes used, their POP settings and OPLE delays, and the direction towards the star, leading to a system of linear equations of form (3). The unknown parameters are the 3D coordinates of the five telescopes relative to the sixth, the delays associated with the N_P POP settings, and the six constant term delays. The constant terms are in fact degenerate parameters, since they can be absorbed into the POP delays for each telescope, leaving us with 15 + N_P total unknown parameters - see section 4.1 for further discussion.

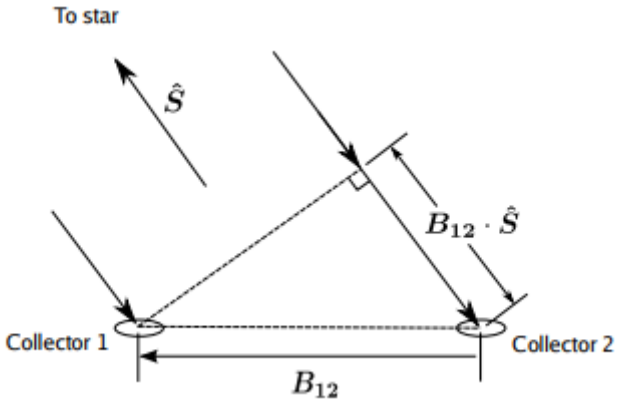

Fig. 1: *Diagram showing the path delay incurred at one of the telescopes due to geometry*

Let us cast the system of equations in matrix form before attempting a solution. We desire an equation of the form

$$A\boldsymbol{a} = \boldsymbol{b} \tag{4}$$

where A is the known *design matrix*, $\boldsymbol{a}$ is the vector of the unknown model parameters, and $\boldsymbol{b}$ the known *data vector*. From above, $\boldsymbol{a}$ takes the form

$$\begin{pmatrix} \boldsymbol{x_1} & \cdots & \boldsymbol{x_5} & P_1 & \cdots & P_{N_P} \end{pmatrix}^T \tag{5}$$

and the rows of the design matrix are just the coefficients of the parameters in $\boldsymbol{a}$. The row corresponding to (3), for example, would be

$$\begin{pmatrix} \{S_x & S_y & S_z & -S_x & -S_y & -S_z & 0 \cdots 0\,\} & \{-1 & 1 & 0 \cdots 0\,\} \end{pmatrix} \tag{6}$$

with the braces delineating the position and POP coefficients. Note that within each row all coefficients are zero except for six (or possibly three, if one of the two telescopes is the reference telescope) position coefficients somewhere in the first 15 columns, and two ± 1 coefficients in the POP columns. The full matrix equation therefore looks like

$$\begin{pmatrix} \cdots \pm \boldsymbol{S}^{(1)} \cdots \mp \boldsymbol{S}^{(1)} \cdots \pm 1 \cdots \mp 1 \cdots \\ \cdots \pm \boldsymbol{S}^{(2)} \cdots \mp \boldsymbol{S}^{(2)} \cdots \pm 1 \cdots \mp 1 \cdots \\ \vdots \\ \vdots \\ \vdots \end{pmatrix} \begin{pmatrix} \boldsymbol{x_1} \\ \vdots \\ \boldsymbol{x_5} \\ P_1 \\ \vdots \\ P_{N_P} \end{pmatrix} = \begin{pmatrix} E1^{(1)} - E2^{(1)} \\ E1^{(2)} - E2^{(2)} \\ \vdots \\ \vdots \\ \vdots \end{pmatrix} \tag{7}$$

For M model parameters and N data points, A is an $N \times M$ matrix. Technically, A and $\boldsymbol{b}$ are usually of the form

$$A_{ij} = \frac{\phi_j(x_i)}{\sigma_i} \tag{8}$$

$$b_i = \frac{y_i}{\sigma_i} \tag{9}$$

where the σ_i are the measurement errors on the data vector. The primary errors in the data vector of the OPLE delays are due to the RMS fluctuations from the atmosphere, which are unknown. The correct approach is to simply set the σ_i to 1 and later estimate them by assuming a chi-squared distribution for the residuals - see section 2.3.

Finding the positions of the telescopes therefore reduces to solving the matrix equation (7). Together with an estimate of the computational errors in the solution for $\boldsymbol{a}$, one can determine whether there is evidence for motion between two times.

2. Analysis

2.1. Available data

The available datasets were stored in `.csv` files. Four nights of data from 2019 were available: 7th and 8th April, and 5th and 6th November, each containing around 2000 to 5000 data points. A large dataset from 2012 was also available, containing around 22,000 points spread throughout the year. The project therefore aims to find the seasonal variation between the two months in 2019, and the variation over the seven year time-period.

Each row of the data contains the elevation and azimuthal angle of the star observed in horizontal coordinates, which can be converted to the unit vector $\boldsymbol{S}$

$$S_x = \cos\theta \sin\phi$$
$$S_y = \cos\theta \cos\phi$$
$$S_z = \sin\theta$$

where θ is the elevation, and ϕ is the azimuth. The names of the two telescopes 'tel_1' and 'tel_2', their POP settings 'POP_1' and 'POP_2', and the OPLE delays 'cart_1' and 'cart_2' for each are also given. The difference between the OPLE delays forms the data vector, while the other information exactly determines the design matrix (up to arbitrary permutations of the columns/model parameters).

Since the positions of the telescopes enter the system of equations only as the differences between them i.e. the baseline vectors (see (2)), it is only possible to find the *relative* movement between the telescopes, rather than their absolute positions. Of course, no single telescope will remain fixed in position to be a reference, but within the interferometer it is the baselines that are important, so to find their motion is sufficient.

2.2. Solving the matrix equation

The method chosen to solve the matrix equation is *singular value decomposition* (SVD), based on the following theorem (quoted from *Numerical Recipes*):

Any M × N matrix A whose number of rows M is greater than or equal to its number of columns N, can be written as the product of an M × N column-orthogonal matrix U, an N × N diagonal matrix W with positive or zero elements (the singular values), and the transpose of an N × N orthogonal matrix V [1].

We can therefore decompose the design matrix A as

$$\begin{pmatrix} \\ \boldsymbol{A} \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \boldsymbol{U} \\ \\ \end{pmatrix} \cdot \begin{pmatrix} w_1 & & \\ & \ddots & \\ & & w_N \end{pmatrix} \cdot \begin{pmatrix} \\ \boldsymbol{V^T} \\ \\ \end{pmatrix} \tag{10}$$

which immediately gives a solution to equation (4)

$$\boldsymbol{a} = A^{+}\boldsymbol{b} \tag{11}$$

where

$$A^+ = V \cdot \text{diag}(1/w_j) \cdot U^T \tag{12}$$

is the Moore-Penrose pseudo-inverse of the matrix A. In the case that the w_j are all nonzero, this solution is the least-squares solution i.e. the best possible fit to the data. We can consider the fitted parameters $\boldsymbol{x}$ to be a linear combination of the columns of V:

$$\boldsymbol{a} = \sum_{i=1}^{M} \left(\frac{\boldsymbol{U}_{(i)} \cdot \boldsymbol{b}}{w_i} \right) \boldsymbol{V}_{(i)} \tag{13}$$

Each w_j corresponds to a column of V, with zero or near-zero values of w_j indicating degeneracies in the system of equations. The corresponding columns can be added in any linear combination to the fitted parameters $\boldsymbol{a}$ while still being a solution, representing unconstrained directions in parameter space. These columns are irrelevant to the solution, and so can be ignored when assembling $\boldsymbol{a}$, which is achieved by replacing the corresponding $1/w_j$ with zero[1]. Near-zero is usually defined as around $N\epsilon$ times the maximum singular value, where N is the number of data points and ϵ is the machine precision.

Singular value decomposition is therefore very robust - it allows us to diagnose where data is potentially malformed by throwing up zero singular values where there is a degeneracy or singularity.

2.3. Errors

Differences in the estimated positions may indicate relative motion, but can also arise from experimental uncertainty. This has contributions from the unknown RMS error on the delay measurements from the atmosphere, as well as the computational error itself. Quantification of any likely motion will therefore require an estimate of the errors involved in the SVD method. Two approaches were taken and compared for consistency:

Variation between consecutive days: Since we are looking for seasonal or year-on-year variation in position, the positions can be assumed to be approximately constant across data taken from two consecutive or nearby days. Differences in the two estimates are therefore assumed to be due to experimental error only, which gives the overall uncertainty.

Theoretical SVD error: SVD also provides a theoretical estimate of the uncertainty on the model parameters $\boldsymbol{a}$ using the singular values and the corresponding values of the v matrix:

$$\sigma^2(a_i) = \sum_{i=0}^{N-1} \left(\frac{V_{ji}}{w_i} \right)^2 \sigma_d^2 \tag{14}$$

where the σ_d is the RMS error, assumed to be the same for all measurements. σ_d is estimated as follows: we assume that χ^2 of the best-fit model is approximately equal to the number of degrees of freedom ν (number of equations - number of unknown parameters), which is asymptotically true for linear models with large ν [1]. By definition:

$$\chi^2 = \sum_{i \text{ data points}} \left(\frac{y_i - f(\{\theta_i\}; x_i)}{\sigma_i} \right)^2 = \frac{1}{\sigma_d^2} \sum (\text{ith residual})^2 \tag{15}$$

where the *residual* of a data point is defined as the difference between the predicted data vector using the calculated model parameters, and the actual data vector i.e. $A\boldsymbol{a} - \boldsymbol{b}$. Hence

$$\nu = \frac{1}{\sigma_d^2} \sum (\text{ith residual})^2 \implies \sigma_d^2 = \frac{\text{sum of squares of residuals}}{\nu} \tag{16}$$

Substituting this expression into (14) gives the estimate for the model parameter uncertainties.

3. Implementation

The program was implemented in Python, due to the availability of numerical libraries such as `numpy` and `pandas` to achieve vast speed-up of vectorised operations, as well as easy to use and fast plotting capabilities with `matplotlib.pyplot`. Object-orientation also improved readability and ease of maintenance.

3.1. Structure of the program

The overall program was implemented under the Data class, initialised using a table of data from a given night. The class has three primary attributes:

1. An instance of the custom Telescopes class, containing information about the telescopes and POP settings used in that night.
2. An instance of the Design class, which constructs the design matrix and data vector for the table of data upon initialisation.
3. An instance of the SVD class, which implements the SVD algorithm and contains methods to calculate results such as positions and variances.

Other attributes store information about the data itself, such as the number of data points and the number of model parameters. The Telescopes class contains a dictionary of the POP settings used by each telescope, which is used to construct a reference list for the order of the coefficients of the model parameters (see (5)), as well as a method that grabs the indices of the $\boldsymbol{S}$ and POP coefficients for a given telescope and setting.

The Design class instance, upon construction, iterates through the table of data, taking each data point and calculating the corresponding row in the design matrix. The azimuth and elevation angles for the data point are used to calculate to the unit vector $\boldsymbol{S}$. The name and POP setting of the first and second telescopes are recorded, and coefficients are placed in the correct columns with help from the method in the Telescopes class. The data vector is also constructed as an attribute of the Design class instance, by finding the difference between the OPLE delay measurements for each data point. If fewer than six telescopes were used for a given set of data, the design matrix is modified to include correspondingly fewer position parameters and constant terms.

The SVD class instance took the design matrix and data vector and used the well-tested and documented `numpy.linalg.svd` routine to calculate the U, $1/w$, and V^T matrices, including replacement of degenerate singular values below the threshold discussed previously. Methods of the class allowed the use to return calculated positions, POP and constant delays, as well as residuals and variances.

The telescope taken to be the reference for the other positions was chosen to be W1, since it was used for all nights in 2019, and comprised at least 20% of the points in each 2000 data point interval of the 2012 dataset. Results were plotted using `matplotlib.pyplot` to show the relative changes in position, with error ellipses to show likely overlap due to uncertainties.

3.2. Testing

A test dataset was created manually using a set of telescopes given arbitrary positions, with arbitrary POP delays and constant terms. Random combinations of the telescopes were chosen, with random azimuth/elevation angles, then used to calculate the necessary data vector for the

delays to equalise. The program was run on this dataset of around 100 values and was able to exactly recover the telescope positions.

After the succeeding on the test dataset, the program was tested on the April 2019 dataset, and the positions compared to a global baseline solution from 2004 (see ten Brummelaar et al. [2]). The relative positions to W1 were correct to one decimal place, which indicates the program runs as expected - any differences are likely due to relative motion and uncertainties in both sets of results.

3.3. Performance

Run times for the program were generally insignificant. The primary bottlenecks were the construction of the design matrix, which involved iterating through several thousand data points, and the running of the SVD algorithm, both of which took on the order of several hundred milliseconds. Vast speed-up was achieved through the use of `pandas` to clean and edit the dataset, which took on the order of several seconds if a loop-based approach was taken instead. `numpy` broadcasting to perform numerical operations on arrays was also several orders of magnitude faster than looping through the arrays. `numpy.linalg.svd` was used since the routine is implemented in FORTRAN, hence fast, and well-tested hence robust.

4. Results and discussion

4.1. Data cleaning and interpretation of degeneracies

Outliers were taken to be the data points with residuals greater than two standard deviations from zero, and were removed before analysis of results. As shown in figure 2, this retained the more reliable data.

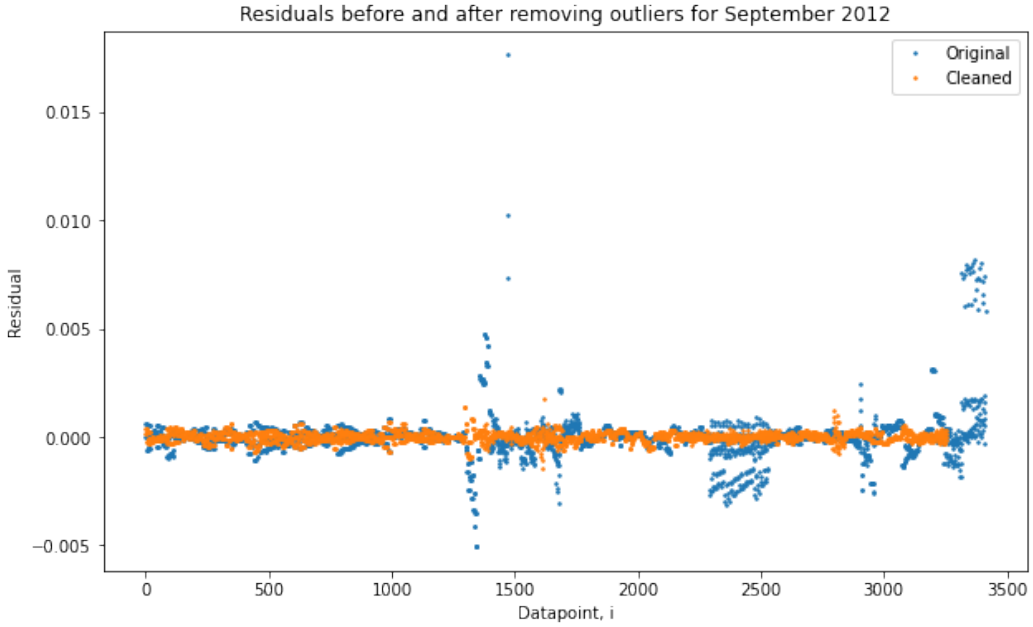

Fig. 2: *Plots showing the residuals before (blue) and after (orange) removing outliers further than 2σ from zero. The cleaned data clearly better conforms to the predicted results.*

As discussed previously, degeneracies from the constant terms were absorbed into the POP settings, and only telescopes included in the data were made parameters of the design matrix.

Nevertheless, a number of zero or near-zero singular values were obtained for each dataset, suggesting previously unaccounted for degrees of freedom. These were explained by the following:

Differences: since the POP delays only enter the equations via their differences, they can all shift by a constant without affecting the solution. Furthermore, if the POPs pair with each other in mutually exclusive groups (shown below), each cluster can independently shift by a constant, contributing additional degeneracies. These only affected the POP delays, and not the telescope positions.

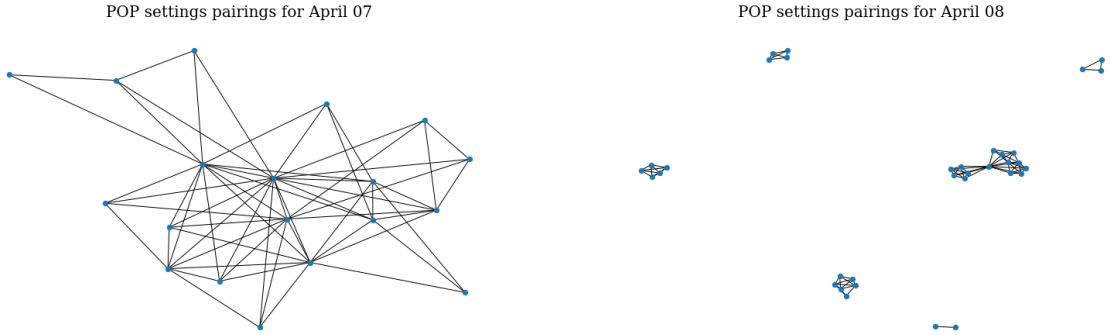

Fig. 3: *Graph showing groups of pairings of the POP settings on 08/04/2019. Each node represents a setting, each edge a pair. There are six distinct clusters, which contributes six degeneracies.*

Small datasets: if the design matrix contains only a few data points, observing perhaps only one or two stars, then we may lack sufficient information to determine the baselines. Some data points may contribute the same information, leading to fewer equations than unknowns and hence no unique solution for the telescope positions. Any additional redundancies not explained by the previous factors can therefore be attributed to the size of the dataset.

4.2. Seasonal variation in 2019

We now use the 2019 data to obtain the positions of the telescopes and their uncertainties, using the two methods outlined in section 2.3.

4.2.1. Using variation between consecutive days as errors

Data in April 2019 was analysed to produce the following positions and absolute deviations between the two nights:

Telescopes	East (m)	North (m)	Height (m)	East (m)	North (m)	Height (m)
W2	105.9760	-16.9891	11.2557	105.9778	-16.9899	11.2568
S1	175.0717	-216.3228	10.7885	175.0734	-216.3234	10.7960
S2	169.3248	-182.7440	11.4282	169.3248	-182.7440	11.4282
E1	300.4086	89.6098	4.8805	300.4086	89.6098	4.8805
E2	245.4646	53.3999	7.9874	245.4646	53.3999	7.9874

Table 1: *Calculated positions for April 2019, 7th shown on the left and 8th on the right*

Taking the uncertainties to be half the deviation (around the mean), and repeating the process for the 5th and 6th of November gives the following results:

It can be seen that in April, the difference in position between the two nights was several hundred microns for most measurements, and several millimeters for a few, with E2 a notable ex-

Telescopes	East (m)	North (m)	Height (m)	East (m)	North (m)	Height (m)
W2	105.9769	-16.9895	11.2563	0.0009	0.0004	0.0005
S1	175.0725	-216.3231	10.792	0.0008	0.0003	0.004
S2	169.3256	-182.74417	11.4294	0.0007	0.00017	0.0011
E1	300.4090	89.6094	4.884	0.0004	0.0005	0.004
E2	245.468	53.395	7.993	0.003	0.005	0.006

Table 2: *Mean positions and uncertainties for April 2019*

Telescopes	East (m)	North (m)	Height (m)	East (m)	North (m)	Height (m)
W2	105.98291	-16.98700	11.25709	0.00003	0.00004	0.00008
S1	175.07312	-216.3227	10.7920	0.00003	0.0003	0.0008
S2	169.32595	-182.7427	11.4309	0.00014	0.0004	0.0005
E1	300.4086	89.6079	4.8812	0.0005	0.0003	0.0012

Table 3: *Mean positions and uncertainties for November 2019*

ception due to a lack of data compared to the others. The same was true for most positions in November, with some exceptions drifting a few tens of microns.

Differences between the mean positions of the two months were obtained by subtraction. The error on each difference is given by standard error propagation, where $\sigma^2_{diff} = \sigma^2_{Apr} + \sigma^2_{Nov}$. The error on the baselines was calculated using:

$$r^2 = x^2 + y^2 + z^2 \tag{17}$$

$$\sigma_r^2 = \frac{1}{r^2}\left[(x\sigma_x)^2 + (y\sigma_y)^2 + (z\sigma_z)^2\right] \tag{18}$$

Some illustrative plots of the positional changes are shown below.

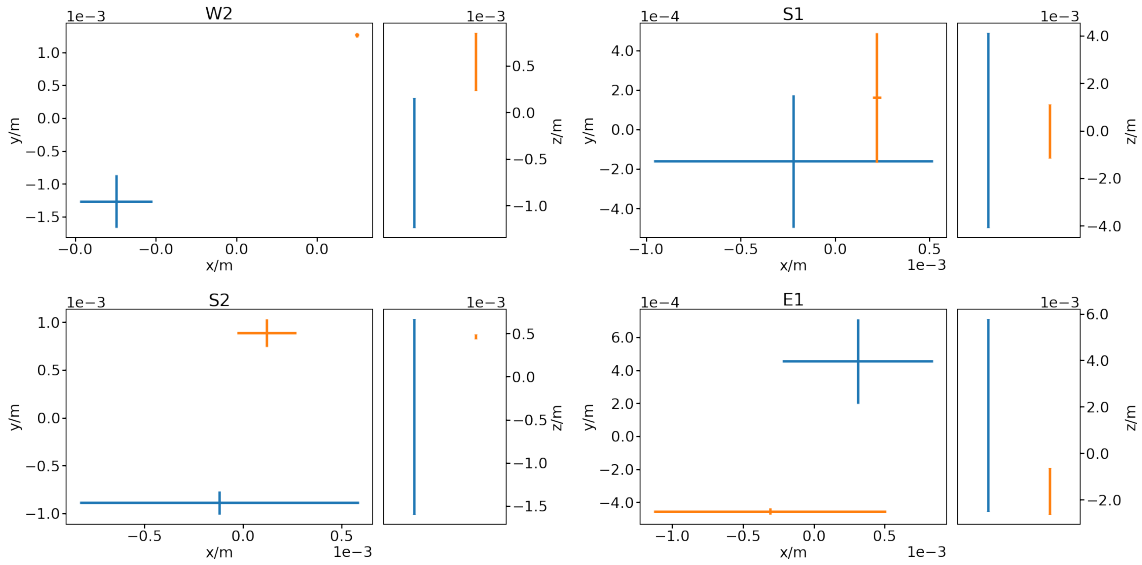

Fig. 4: *Plots of the the telescope positions relative to W1 in April and November 2019, with x-y variation on the left and, z variation on the right. In each plot the April positions are blue, and the November positions orange. Error bars were estimated from the variation between consecutive days.*

Likely motion was judged to have occurred where the magnitude of the difference was greater than the error, with values shown in table 4.

Telescopes	x (m)	y (m)	z (m)	Baseline (m)
W2	$-6.0 \pm 0.9 \times 10^{-3}$	$-2.5 \pm 0.4 \times 10^{-3}$	$-1.1 \pm 0.8 \times 10^{-3}$	$6.6 \pm 0.8 \times 10^{-3}$
S1	$-4.4 \pm 7.4 \times 10^{-4}$	$-3.2 \pm 4.7 \times 10^{-4}$	$0.0 \pm 4 \times 10^{-3}$	$5.5 \pm 7.3 \times 10^{-4}$
S2	$-2.4 \pm 7.2 \times 10^{-4}$	$-1.8 \pm 0.2 \times 10^{-3}$	$-0.9 \pm 1.1 \times 10^{-3}$	$2.0 \pm 0.6 \times 10^{-3}$
E1	$6.2 \pm 9.8 \times 10^{-4}$	$9.1 \pm 2.6 \times 10^{-4}$	$3.3 \pm 4.3 \times 10^{-3}$	$3.5 \pm 4.0 \times 10^{-3}$

Table 4: *Vector changes in positions between April and November 2019*

There is clear evidence of changes in the baselines for telescopes W2 and S2, centred around 6.6mm and 2mm respectively. E1 appears to be on the cusp, but we find that this is primarily due to the x and z contributions; its y component has clearly changed by about a millimetre. There is no straightforward directional correlation between the motion of the telescopes; W2 and S2 move roughly North-East, and S1 possibly also (just within error), but E1 moves in the opposite direction.

We can also observe drifts in the POP delays. As discussed, only differences in POP settings are physically meaningful, so we observe POP pairings which were used in both months. We also need to be careful that the pairs of POP settings chosen are in the same group, as per fig. 3, so that they cannot independently vary by different constants. For April and November, the settings W1-P2B3, W2-P5B2, S1-P1B5, S2-P2B4 and E1-P1B1 meet both requirements. Choosing W1-P2B3 to be a reference, we observe the following changes:

POP delays/mm

Telescope	08 Apr	05 Nov	06 Nov
W2-P5B2	1.2	-3.0	-2.6
S1-P1B5	6.9	4.4	6.1
S2-P2B4	2.5	2.8	3.5
E1-P1B1	6.3	-5.9	-3.4

Table 5: *POP delays compared to the delays on 07 Apr for the settings sampled on all four nights in 2019. The drift in these delays contributes to a minimum uncertainty of order 1mm. W1-P2B3 was chosen to be the reference for the other delays, as only relative POP delays are significant.*

The magnitude of variation seems quite high for the 8th April, far more than the expected few hundred microns over one day. This may be due to a shortage of data for certain POP settings. We can also see that the long-term drift of the delays do not accumulate; if the drifts were linear or random, we would expect by comparing to a random walk model across the 200 days a drift of around $\sqrt{200} \times 1\text{mm} = 4.5\text{cm}$. That the actual drift is only a few millimeters suggests it tends to even itself out over longer time periods.

4.2.2. SVD errors

We can obtain an alternate error estimate by using formula (14), shown in table 6. These are about an order of magnitude smaller than those previously obtained, which seems to suggest the previous variations are not experimental uncertainties, but an actual night to night variation known to a few tens of microns. This is impossible - the time window for data collection within a night is from around 3am to midday, which is about the same length of time that elapses between two 'nights'. Unless the telescopes magically refuse to move while in use, we would expect around the same variation during the night as between the nights. The SVD errors therefore cannot be physically meaningful.

It seems odd that the SVD algorithm can be so certain about the telescope positions, and yet still

find a variations of order 1mm - nearly hundred times the SVD error - between two consecutive datasets. One possible explanation is that the telescopes do remain fixed over this short time period, but their positions are still uncertain due to POP drift. This has a known magnitude of a couple hundred microns (or up to several millimeters as found in the previous section), which is just in the right ballpark for the variation of the telescope positions in consecutive days. Because the POP and positional delays are linearly related, the uncertainty in the estimate of the positions is limited by the POP variation through the night. Once again, we have found that any SVD errors far below about 1mm are spurious.

Telescopes	x (m)	y (m)	z (m)
W2	6.7×10^{-5}	4.1×10^{-5}	1.9×10^{-4}
S1	8.5×10^{-5}	1.0×10^{-4}	2.7×10^{-4}
S2	7.5×10^{-5}	8.5×10^{-5}	2.0×10^{-4}
E1	8.6×10^{-5}	8.2×10^{-5}	2.3×10^{-4}
E2	2.2×10^{-3}	4.3×10^{-3}	3.1×10^{-3}

Table 6: *SVD errors for the 7th April 2019. Some of the errors are a order of magnitude smaller than those obtained using the variation between consecutive days (see table 2), which cannot have physical meaning*

4.3. Evidence for movement of telescopes between 2012 and 2019

The unreliability of the SVD errors means we need some other way of quantifying errors in the 2012 data. While there is nothing wrong with using the consecutive day deviations, this is difficult to achieve with the 2012 data because few nights have enough data to make accurate predictions alone.

It is therefore necessary to use timespans of longer than one night to make predictions. A possible concern might be that over a long period of time, the POP drifts may snowball, causing overwhelming uncertainties in the positions. However, as seen in the 2019 analysis, the drifts do not accumulate well over time, preferring to hover around some mean - this is shown to also apply in 2012 in the next section. Given that this is the case, it is then safe to use deviations between consecutive weeks or months of data as uncertainties, which is approach chosen here.

4.3.1. Structure of 2012 data

The majority of the 2012 data lay between the months of June to November, with a peak from August to October. The analysis will focus on these months. POP drifts were observed for the settings common to August, September and October 2012, shown in table 7; these did not exceed a few millimetres in magnitude, which justified the assertion that the drifts do not significantly impact the positional error when longer timespans are used.

4.3.2. Using variation between nearby time periods as errors

The months of August, September and October were each split into their first and second halves, which were used as the adjacent time periods to quantify uncertainty. The positions and errors were plotted and compared with the 2019 solutions, shown in fig. 6.

Table 8 gives values for the differences between September 2012 and November 2019. Both the table and the plots make clear a southward motion of around 3-6mm for all the telescopes. Furthermore, W2, S1 and E1 all showed clear signs of an eastward shift of around 2-4mm. If there is any motion in the vertical direction, it is no more than the experimental uncertainties

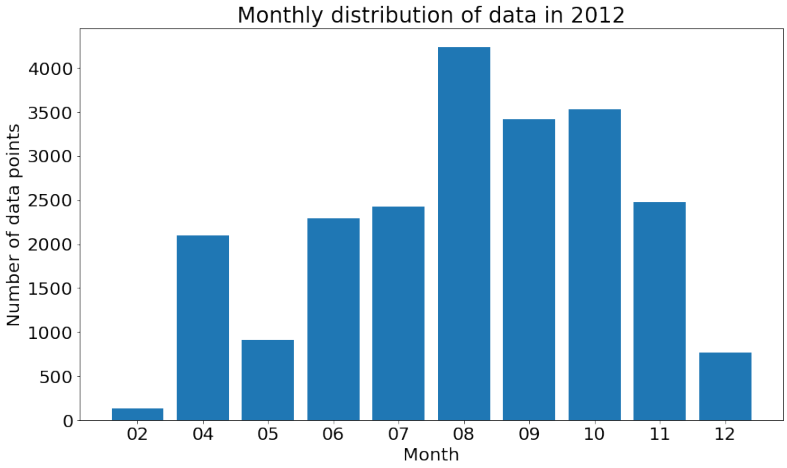

Fig. 5: *Distribution of data for 2012. The bulk of the data is within the months June-November*

POP setting	Sep 12	Oct 12
W1-P1B2	1.5×10^{-4}	-8.9×10^{-4}
W1-P3B2	1.0×10^{-5}	-1.1×10^{-3}
W1-P5B2	-6.5×10^{-5}	-6.8×10^{-4}
W2-P5B2	-1.1×10^{-3}	-1.3×10^{-3}
S1-P1B1	3.0×10^{-3}	-3.5×10^{-5}
S1-P4B1	3.7×10^{-3}	-6.7×10^{-6}
E1-P1B3	8.0×10^{-4}	8.0×10^{-4}

Table 7: *POP delays for September/October compared to August 2012, using the W1-P1B1 setting as reference. The maximum drift is less than 4mm, suggesting that the drift will not significantly contribute to positional uncertainty when analysing datasets with multiple nights included.*

present. Values for the positional changes for E2 were not shown due to the large errors involved in the 2019 solution, which can be seen in fig. 6.

It is suspected that there may be some systematic error involved in the April 2019 measurements for W2; all other telescopes are clustered closely within the same year, and the April W2 measurement is the only one that bucks the trend of an eastward shift.

Telescopes	x (m)	y (m)	z (m)	Baseline (m)
W2	$1.7 \pm 0.7 \times 10^{-3}$	$-2.84 \pm 0.03 \times 10^{-3}$	$-6.4 \pm 8.5 \times 10^{-4}$	$3.4 \pm 0.4 \times 10^{-3}$
S1	$1.6 \pm 0.7 \times 10^{-3}$	$-3.1 \pm 0.4 \times 10^{-3}$	$-1.4 \pm 1.1 \times 10^{-3}$	$3.8 \pm 0.6 \times 10^{-3}$
S2	$3.0 \pm 4.0 \times 10^{-4}$	$-5.4 \pm 0.7 \times 10^{-3}$	$0.8 \pm 1.5 \times 10^{-3}$	$5.5 \pm 0.7 \times 10^{-3}$
E1	$3.8 \pm 0.8 \times 10^{-3}$	$-6.2 \pm 0.6 \times 10^{-3}$	$-3.0 \pm 2.0 \times 10^{-3}$	$7.9 \pm 0.1 \times 10^{-3}$

Table 8: *Vector changes in positions between September 2012 and November 2019*

5. Conclusion

The system of equations satisfied by the positions of the telescopes and their POP setting delays was successfully and efficiently solved using singular value decomposition. The theoretical

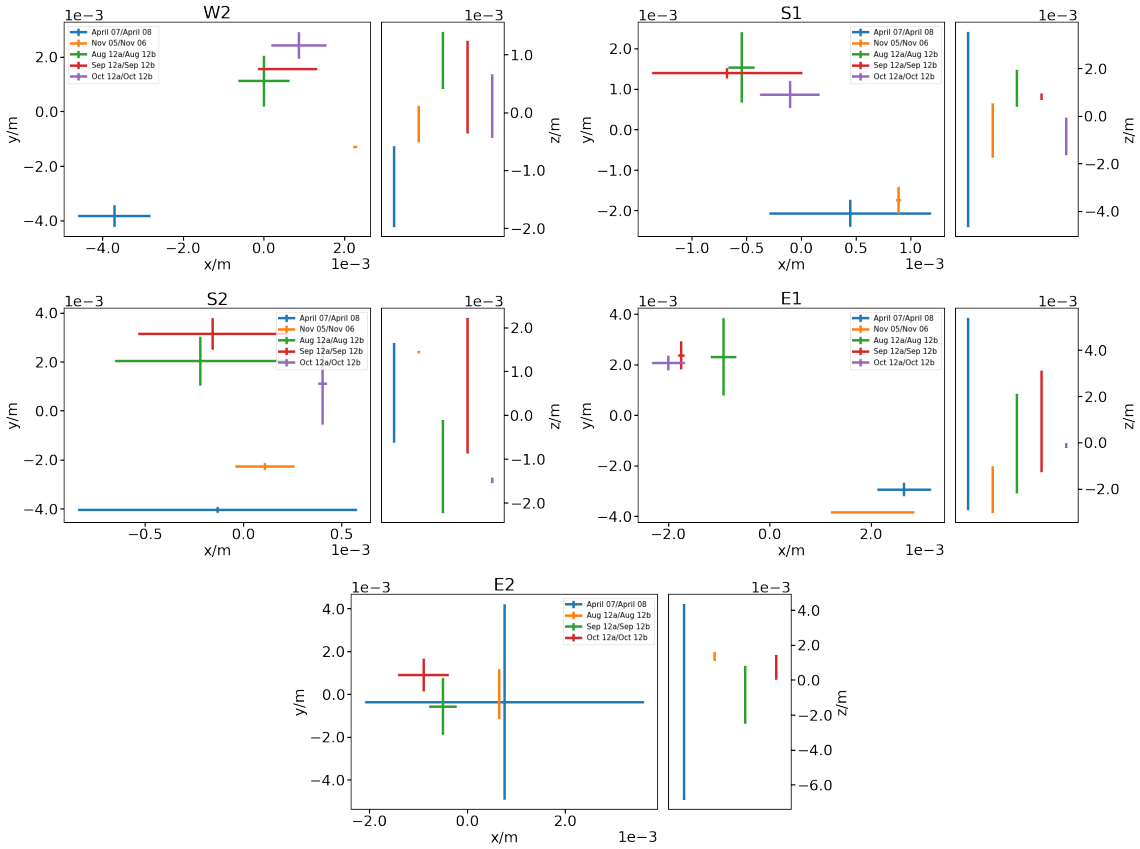

Fig. 6: *Plots of the the telescope positions relative to W1 for August-October 2012, compared to the 2019 data. Error bars were estimated by splitting each month of data in 2012 in two, and taking the half the deviation between the two as the error. A general trend of southward motion with time can be seen.*

SVD errors in the positions were too small to be physically meaningful, so errors were instead quantified experimentally by using the deviation between two datasets adjacent in time. Within 2019, the W2, S2, and E1 telescopes showed seasonal variation relative to W1, with the baselines changing by $6.6 \pm 0.8 \times 10^{-3}$m, $2.0 \pm 0.6 \times 10^{-3}$m, and $9.1 \pm 2.6 \times 10^{-4}$m (the latter only in the y direction) respectively. From 2012 to 2019, the telescopes showed collective motion towards the south and east, with respective magnitudes of 2-6mm and 2-4mm.

Further work could be done to investigate the possibility of systematic errors involved in the measurements, for example by calculating the predicted positions of the stars and comparing to the angular measurements made. One may also benefit from a more detailed analysis of the seasonal variation in 2012 and investigate correlations with known seismic activity.

6. Appendix

6.1. Code listing

classes.py

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
```

```
class Data:
    def __init__(self, data_table, name=None):
        """Initialise all variables unique to this particular
           ↪ dataset"""
        self.table = data_table
        self.telescopes = Telescopes(data_table)
        self.N_datapoints = data_table.shape[0]
        # sum of telescope coordinates/total number of pops/
           ↪ number of constant terms
        self.N_parameters = self.telescopes.N_telescope_positions
           ↪  + self.telescopes.N_pops # + self.telescopes.
           ↪ N_telescopes
        self.design = Design(self)
        self.svd = SVD(self)
        self.name = name


class Telescopes:
    """List of unique telescopes and POP settings for each
       ↪ defines a row in the design matrix uniquely"""
    def __init__(self, data_table):
        self.all_telescopes = ['W1', 'W2', 'S1', 'S2', 'E1', 'E2
           ↪ ']
        self.unique_telescopes = self.all_telescopes.copy()
        self.pops_dictionary = {}
        self.pops_reference = []
        self.N_pops = 0
        self.N_telescopes = 0
        self.N_telescope_positions = 0
        self.G_pop = None

        # Make list of unique telescope names
        tel_1 = data_table['tel_1'].unique()
        tel_2 = data_table['tel_2'].unique()
        unique_telescopes_set = set().union(tel_1, tel_2)
        for telescope in self.all_telescopes:
            if telescope not in unique_telescopes_set:
                self.unique_telescopes.remove(telescope)

        # Construct dictionary showing possible POP settings for
           ↪ each telescope
        for telescope in self.unique_telescopes:
            # For a particular telescope:
            pops1 = set(data_table.loc[data_table['tel_1'] ==
               ↪ telescope]['pop_1'].unique()) # unique pop1
               ↪ settings
            pops2 = set(data_table.loc[data_table['tel_2'] ==
               ↪ telescope]['pop_2'].unique()) # unique pop2
               ↪ settings
```

```
            pops = set().union(pops1, pops2) # combine unique pop
               ↪ settings
            self.pops_dictionary[telescope] = list(pops) # add to
               ↪ dictionary as a list

        # Create count of total number of distinct POP settings
        for pop_list in self.pops_dictionary.values():
            self.N_pops += len(pop_list)

        # Create reference for order of POP settings in a row of
           ↪ the design matrix
        for telescope in self.unique_telescopes:
            for telescope_pop in self.pops_dictionary[telescope]:
                telescope_pop_id = telescope+'-'+telescope_pop #
                   ↪ Create unique ID for a POP setting of a
                   ↪ given telescope
                self.pops_reference.append(telescope_pop_id)

        self.N_telescopes = len(self.unique_telescopes)
        self.N_telescope_positions = 3*(self.N_telescopes-1)

        # Create graph of POP settings pairings
        df = data_table
        df['telpop_1'] = df.tel_1 + '-' + df.pop_1
        df['telpop_2'] = df.tel_2 + '-' + df.pop_2
        df_graph = df.groupby(['telpop_1', 'telpop_2']).size().
           ↪ reset_index().rename(columns={0:'count'})
        unique_telpops = set(df['telpop_1'].unique().tolist() +
           ↪ df['telpop_2'].unique().tolist())
        G = nx.Graph()
        G.add_nodes_from(unique_telpops)
        for index, row in df_graph.iterrows():
            G.add_edge(row['telpop_1'], row['telpop_2'])
        self.G_pop = G


    def indices_of(self, given_telescope, given_pop):
        """For a given telescope and POP setting, return the
           ↪ index of the telescope x coordinate,
        the index of the POP setting, and the index of the
           ↪ constant term within a row of the design matrix."""

        # Find index of x-coordinate of the telescope
        telescope_index = self.unique_telescopes.index(
           ↪ given_telescope)
        if given_telescope == 'W1':
            # Return None if telescope is W1, since W1 is used as
               ↪ the reference for the baseline vectors
            x_index = None
        else:
            x_index = 3*(telescope_index-1)
```

```
        # Use the reference list to calculate the index of the
          ↪ given POP setting
        given_pop_id = given_telescope+'-'+given_pop
        pop_index = self.N_telescope_positions + self.
          ↪ pops_reference.index(given_pop_id)

        # const_index = self.N_telescope_positions + self.N_pops
          ↪ + telescope_index

        return x_index, pop_index #, const_index


    def get_index(self, telescope):
        """Returns position index of telescope within
          ↪ unique_telescopes - 1, i.e. W2 is index 0, for
          ↪ referencing purposes"""
        if telescope == "W1":
            raise Exception("W1 is the reference telescope")

        return self.unique_telescopes.index(telescope) - 1


    def draw_pops(self, labels=True, size=(10,10), fontsize=15):
        """Plots graph of POP pairings"""
        plt.figure(figsize=size)
        nx.draw(self.G_pop, with_labels=labels, node_size=40)
        plt.title(f'POP settings pairings for {self.data.name}',
          ↪ font='serif', fontsize=fontsize)


    def pops_linked(self, telpop, querypop=None):
        """Returns set of pops paired with a given telpop,
          ↪ returns True/False if querypop given is linked to
          ↪ telpop"""
        components = nx.node_connected_component(self.G_pop,
          ↪ telpop)
        if querypop != None:
            return (querypop in components)
        else:
            return components


class Design:
    """Data structure that calculates and stores the design
      ↪ matrix and data vector
    from a data object"""
    def __init__(self, data):
        """<data> must be a Data object"""
        self.data = data
        self.A = np.zeros((data.N_datapoints, data.N_parameters))
```

```
            ↪  # initialise design matrix
        self.b = np.array(data.table['cart_1'] - data.table['
            ↪ cart_2']) # create data vector

        # Construct design matrix
        for i in range(data.N_datapoints):
            data_row = data.table.iloc[i]
            design_row = np.zeros(data.N_parameters)

            # Calculate unit vector S to star
            phi = np.deg2rad(data_row['azimuth'])
            theta = np.deg2rad(data_row['elevation'])
            S = [np.cos(theta)*np.sin(phi), np.cos(theta)*np.cos(
                ↪ phi), np.sin(theta)] # unit vector pointing
                ↪ towards star

            # Get location of nonzero coefficients
            telescope_1 = data_row['tel_1']
            telescope_2 = data_row['tel_2']
            pop_1 = data_row['pop_1']
            pop_2 = data_row['pop_2']
            x_index_1, pop_index_1 = data.telescopes.indices_of(
                ↪ telescope_1, pop_1)
            x_index_2, pop_index_2 = data.telescopes.indices_of(
                ↪ telescope_2, pop_2)

            for j in range(3):
                # If one of the telescopes is W1, leave S = 0
                    ↪ since coordinates are zero
                if telescope_1 != 'W1':
                    design_row[x_index_1 + j] = S[j]
                if telescope_2 != 'W1':
                    design_row[x_index_2 + j] = -S[j]

            design_row[pop_index_1] = -1
            design_row[pop_index_2] = 1

            #design_row[const_index_1] = -1
            #design_row[const_index_2] = 1

            self.A[i] = design_row


class SVD:
    """Class containing methods to calculate model parameters,
        ↪ uncertainties etc.
    using SVD"""
    def __init__(self, data):
        self.data = data
        self.A = data.design.A
        self.b = data.design.b
```

```
        self.u, self.w, self.vh = np.linalg.svd(self.A,
            ↪ full_matrices=False) # since design matrix is not
            ↪ square (overdetermined)

        # Calculate w_inv; if any w/w_max < this min ratio,
            ↪ corresponding w_inv is taken to be zero
        machine_precision = np.finfo(np.float64).eps
        min_sv_ratio = self.data.N_datapoints * machine_precision
        w_inv = 1/self.w
        w_inv[self.w < self.w.max() * min_sv_ratio] = 0 # replace
            ↪  singular values under min ratio in 1/w with zero
        self.w_inv = w_inv


    def x(self, variables='telescopes', telescope=None):
        """Return model parameters vector x as calculated by SVD,
            ↪  removing degeneracies (singular values near zero).
        By default returns positions only; variables='pops' if
            ↪ want pops, set telescope='W2' etc if want position
            ↪ of specific telescope"""

        x = self.vh.T @ np.diag(self.w_inv) @ self.u.T @ self.b #
            ↪  vector of all parameters

        if variables == 'telescopes':
            # Just return positions
            N_telescope_positions = self.data.telescopes.
                ↪ N_telescope_positions
            N_telescopes = self.data.telescopes.N_telescopes
            positions = x[:N_telescope_positions].reshape(
                ↪ N_telescopes-1, 3) # since W1 is not included

            if telescope == None:
                return positions
            else:
                telescope_index = self.data.telescopes.get_index(
                    ↪ telescope) # W1 not included as index
                return positions[telescope_index]

        elif variables == 'pops':
            return x[self.data.telescopes.N_telescope_positions:]

        elif variables == 'all':
            return x

        else:
            print('Incorrectly specified parameters')


    def residuals(self):
        """Returns list of residuals (differences between actual
```

```
        ↪ data vector b and the predicted values
    from the model, i.e. A multiplied by x)"""

    x = self.x(variables='all')
    residuals = self.data.design.A @ x - self.data.design.b #
        ↪  theoretical - observed

    return residuals


def variances(self):
    """Return vector of variances for the estimated model
        ↪ parameters. Set std=True if want standard
        ↪ deviations,
    all_parameters=True if want all variances instead of just
        ↪  variances of positions"""
    residuals = self.residuals()

    degrees_of_freedom = self.data.N_datapoints - self.data.
        ↪ N_parameters

    residualsquaresum = (residuals**2).sum()
    rms_variance = residualsquaresum/degrees_of_freedom # RMS
        ↪  error on each measurement

    w_inv = np.reshape(self.w_inv, (-1, self.data.
        ↪ N_parameters)) # reshape w_inv into 2D array
    elements = (self.vh*w_inv.T)**2 * rms_variance # matrix
        ↪ to be summed over to find model parameters
    model_parameters_variance = elements.sum(axis=0)

    return model_parameters_variance


def std(self, variables='telescopes', telescope=None, telpop=
    ↪ None):
    """Returns standard deviation of model parameters with
        ↪ option to return only those for
    positions, pops, or a specific delay from those"""

    all_std = np.sqrt(self.variances())
    telescopes = self.data.telescopes

    if variables == 'telescopes':
        telescope_std = all_std[:telescopes.
            ↪ N_telescope_positions].reshape(telescopes.
            ↪ N_telescopes-1, 3)
        if telescope == None:
            return telescope_std
        else:
            telescope_index = self.data.telescopes.get_index(
```

```
                ↪ telescope) # W1 not included
            return telescope_std[telescope_index]

    elif variables == 'pops':
        pops_std = all_std[telescopes.N_telescope_positions:]
        if telpop == None:
            return pops_std
        else:
            return pops_std[telescopes.pops_reference.index(
                ↪ telpop)]

    else:
        return all_std


def positions_dict(self, telescope_name=None):
    """Returns a dictionary of the values of the telescope
       ↪ positions, with option to return specific telescope
       ↪ """
    positions_dict = {}

    for telescope in self.data.telescopes.unique_telescopes
       ↪ [1:]:
        positions_dict[telescope] = self.x(telescope=
           ↪ telescope)

    if telescope_name != None:
        return positions_dict[telescope_name]
    else:
        return positions_dict


def positions_array(self):
    """Returns array of telescope positions, in order
       ↪ specified by
    all_telescopes, zeroes included if telescope not in
    unique_telescopes"""
    positions_array = np.zeros((5,3))
    all_telescopes = self.data.telescopes.all_telescopes
    unique_telescopes = self.data.telescopes.
       ↪ unique_telescopes
    for i, telescope in enumerate(all_telescopes[1:]):
        if telescope in unique_telescopes:
            positions_array[i] = self.positions_dict(
               ↪ telescope)
    return positions_array


def pops_values(self, pop_name=None):
    """Returns a dictionary of the values of the POP delays,
       ↪ with the option
```

```
        to return a specific delay"""
        x_pops = self.x(variables='pops')
        pops_values = {}

        for j, pop in enumerate(self.data.telescopes.
            ↪ pops_reference):
            pop_delay = x_pops[j]
            pops_values[pop] = pop_delay

        if pop_name != None:
            return pops_values[pop_name]

        return pops_values
```

functions.py

```
import classes as cl
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


def remove_outliers(old_data):
    """Takes in Data object, removes all datapoints with
        ↪ residuals > 2 standard deviations out, returns cleaned
        ↪ data object"""
    df = old_data.table
    residuals = old_data.svd.residuals()
    df['residuals'] = residuals
    clean_df = df[abs(df.residuals) < 2 * residuals.std()].copy(
        ↪ deep=False)
    return cl.Data(clean_df, old_data.name)


def load_data(filename):
    """Reads data from a file, renames/adds new columns for
        ↪ 2019/2012, cleans data by removing outliers"""
    df = pd.read_csv(filename)
    if '2019' in filename:
        df.rename(columns={"elevation": "azimuth", "azimuth": "
            ↪ elevation"})
    if '2012' in filename:
        df['day'] = df['utc'].str[:10]
        df['month'] = df['utc'].str[5:7]
    original_data = cl.Data(df)
    clean_data = remove_outliers(original_data)
    return clean_data


def df_12month(df, month):
    """Get month of data from 2012. Enter month as e.g. '08'"""
```

```
    return df[df.month == month]


def df_12date(df, date):
    """Returns datapoints for a single night from 2012, enter
       ↪ date as 12-31 etc"""
    dayname = f'2012-{date}'
    return df[df.day == dayname]


def present_uncertainties(value, error):
    """Returns formatted string of value plus/minus error to the
       ↪ correct number of sig figs and in
    scientific notation"""
    def find_exp(number) -> int:
        base10 = np.log10(abs(number))
        exponent = np.floor(base10)
        value = number/(10**exponent)
        if np.isnan(exponent):
            print(f"{number}")
            raise Exception("NaN error")
        return value, int(exponent)

    swapped = False
    if error > abs(value): # Ensures readable formatting if error
       ↪  > value
        swapped = True
        value, error = error, value

    value, value_exponent = find_exp(value)
    error, error_exponent = find_exp(error)
    exponent_difference = value_exponent - error_exponent

    if error > 2:
        value_roundto = exponent_difference
    else:
        value_roundto = exponent_difference + 1
    value = np.around(value, decimals=value_roundto)

    if error > 2:
        error = np.around(error, decimals=0)/10**
           ↪ exponent_difference
    else:
        error = np.around(error, decimals=1)/10**
           ↪ exponent_difference

    if not swapped:
        return f"{value} +/- {error} E{value_exponent}"
    else:
        return f"{error} +/- {value} E{value_exponent}"
```

```
def common_telpops(data_list):
    """Finds the POP settings common to all datasets in the given
       ↪ list of data"""
    common = set(data_list[0].telescopes.pops_reference)
    for data in data_list:
        telpops = set(data.telescopes.pops_reference)
        common = common.intersection(telpops)
    return common


def pops_difference(data_1, data_2, telpop_1, telpop_2):
    """Finds how much the difference between the two given POP
       ↪ delays changes between two nights"""
    diff_1 = data_1.svd.pops_values(telpop_1) - data_2.svd.
       ↪ pops_values(telpop_1)
    diff_2 = data_1.svd.pops_values(telpop_2) - data_2.svd.
       ↪ pops_values(telpop_2)
    return diff_1 - diff_2


def telpop_changes(data_list, refpop='W1-P2B3'):
    """W1-P2B3 is default reference POP, changes relative to
       ↪ first day in list given
    Finds changes in POP delays from night to night (only
       ↪ differences in delays are significant, hence need
       ↪ reference setting"""
    common = common_telpops(data_list)
    if refpop not in common:
        raise Exception("Reference␣POP␣not␣used␣in␣every␣night␣
           ↪ given")
    common.remove(refpop)
    for data in data_list[1:]:
        print(data.name + ":")
        for telpop in common:
            difference = pops_difference(data_list[0], data,
               ↪ refpop, telpop)
            print(f"{telpop}␣=␣{difference:.2e}")


def month_mean(data_1, data_2):
    """Returns mean and deviation uncertainty for two datasets"""
    positions_1 = data_1.svd.x()
    positions_2 = data_2.svd.x()
    mean_positions = 0.5 * (positions_1 + positions_2)
    errors = 0.5 * abs(positions_1 - positions_2)
    return mean_positions, errors


def vector_change(positions_1, positions_2, errors_1, errors_2):
    """Calculates distance moved and its uncertainty from two
```

```
    ↪ (3,1) vector positions and their errors
    Returns positions_1 - positions_2"""
    vector_diff = positions_1 - positions_2
    vector_sigma = np.sqrt(errors_1**2 + errors_2**2)
    return vector_diff, vector_sigma


def vector_changes_all(datalist_1, datalist_2, ref_data):
        """Calculates change+uncertainty between two pairs of
            ↪ datasets as vector and scalar distance, output
            ↪ Latex formatted"""
    mean1, err1 = month_mean(datalist_1[0], datalist_1[1])
    mean2, err2 = month_mean(datalist_2[0], datalist_2[1])
    telescopes = ref_data.telescopes.unique_telescopes[1:]
    for telescope in telescopes:
        i = telescopes.index(telescope)
        r, sigma_r = vector_change(mean1[i], mean2[i], err1[i],
            ↪ err2[i])
        rl, sigma_rl = length_change(mean1[i], mean2[i], err1[i],
            ↪  err2[i])
        x, y, z = r
        sx, sy, sz = sigma_r
        #print(f"{telescope}: {x:.1e} \pm {sx:.1e}, {y:.1e} \pm {
            ↪ sy:.1e}, {z:.1e} \pm {sz:.1e} \\ ")
        print(f"{telescope} & ${present_uncertainties(x, sx)}$ & $
            ↪ {present_uncertainties(y, sy)}$ & ${
            ↪ present_uncertainties(z, sz)}$ & ${
            ↪ present_uncertainties(rl, sigma_rl)}$ \\\ ")


def length_change(positions_1, positions_2, errors_1, errors_2):
    """Calculates distance moved and its uncertainty from two
        ↪ vector positions for (3,1) array"""
    vector_diff, vector_sigma = vector_change(positions_1,
        ↪ positions_2, errors_1, errors_2)
    diff_r = np.sqrt((vector_diff**2).sum())
    sigma_r = 1/diff_r * np.sqrt(np.sum((vector_diff *
        ↪ vector_sigma)**2))
    return diff_r, sigma_r


def length_changes_all(datalist_1, datalist_2, ref_data):
    """Calculates change between two pairs of datasets as scalar
        ↪ distance, prints with uncertainties"""
    mean1, err1 = month_mean(datalist_1[0], datalist_1[1])
    mean2, err2 = month_mean(datalist_2[0], datalist_2[1])
    telescopes = ref_data.telescopes.unique_telescopes[1:]
    for telescope in telescopes:
        i = telescopes.index(telescope)
        r, sigma_r = length_change(mean1[i], mean2[i], err1[i],
            ↪ err2[i])
```

```
print(f"{telescope}: {r:.1e} +/- {sigma_r:.1e} \\\ ")
```

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C*, pages 59–70, 656–706. Cambridge University Press, 2nd edition, 1992.

[2] T. A. ten Brummelaar, H. A. McAlister, S. T. Ridgway, Jr. W. G. Bagnuolo, N. H. Turner, L. Sturmann, J. Sturmann, D. H. Berger, C. E. Ogden, R. Cadman, W. I. Hartkopf, C. H. Hopper, and M. A. Shure. First results from the CHARA array. II. a description of the instrument. *The Astrophysical Journal*, 628(1):453–465, jul 2005.