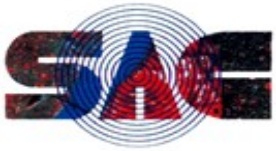


ANALYSIS OF CROSSOVERS AND CORRECTIONS ON SATELLITE ALTIMETER DATA

Report submitted for completion of training

(Scientific Research and Training)
Programme of Space Applications Centre

Submitted by
JAINISH PATEL
(Registration No. 23BCE0503)
B.Tech. (Computer Science)

Under the guidance of

SRI. DARBHA VAMSI PHANI KRISHNA
SCI/ENGR – SD
EPSA/PGSG/GSD
Space Applications Centre, (ISRO) Ahmedabad

Institution
Vellore Institute of Technology
Vellore – 387001
Tamil Nadu

SRTD-RTMG-MISA
Space **A**pplications **C**entre (ISRO)
Ahmedabad, Guajrat

13 May 2025 to 23 July 2025

1. Introduction

Nadir-pointing laser altimeters, such as those onboard satellite missions (e.g., ICESat, GEDI, or upcoming missions), play a crucial role in mapping Earth's topography, monitoring ice sheets, sea surface heights, vegetation, and land elevation. A common technique to assess the accuracy and consistency of laser altimeter data is *crossover analysis (A crossover point is the location of intersection of any two ground tracks charted by multiple platforms (ships, satellite radar and laser altimeters etc.))*, in which we compares elevation readings at locations where the orbital tracks intersect.

This project focused on detecting such crossovers and analyzing the corrections needed to reconcile altitude differences, improving the overall data quality and utility for geophysical applications. To detect these crossovers we are using PyReX: A Recursion Based Crossover Detection Algorithm in Python. The project focusing on both terrestrial (Indian Ocean) data and lunar (Chandrayaan-1) data.

1.1 PyReX Algorithm

In PyReX, the crossover detection is made efficient by excluding parts of tracks which do not intersect by gradually splitting of tracks into smaller sub-tracks, first based on trend changes in latitude or longitude and later recursively. This means that the time taken for detection is proportional to log N, where N is the number of segements in a track. This means it is more effecient than traditional methods that take time proportional to N^2 for detection. This project does not delve into specifics of the detection algorithm. However, to continue the work on this, one must know how it works and how to execute it.

The configuration script (config_cross.py) is the starting point for the user working with PyReX. It is divided into multiple sections each denoted by a calc_id, and a set of initial parameters within each section to perform a specific operatin. This config_cross.py should be updated by adding new sections (with new calc_id) or editing the already existing sections based on the user's requirements. The essence of PyReX is in the detection script (detection.py) which takes the list of track files containing along-track measurements. The user

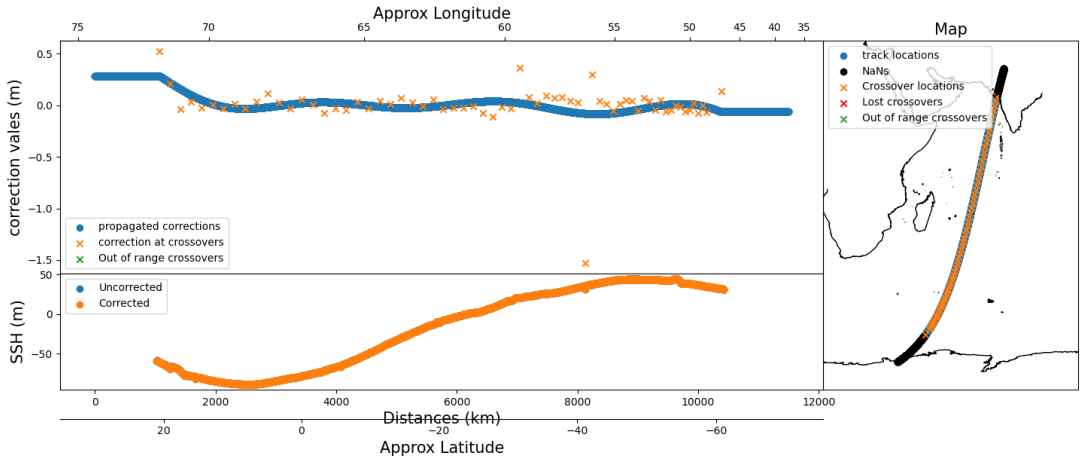
may specify the list of input tracks, a preferred intersection method (line or arc intersection), type of crossover (intersections within a track are “internal” while intersections between two tracks are “external”) as initial parameters within the calc_id section corresponding to detection in config_cross.py. Execution of detection.py with its corresponding calc_id reports the list of track pairs with intersection locations along with interpolated values as an output detection_file.

To execute it keep all the codes in the same directory. Open terminal, change the working directory to the one with the codes. To run the detection script for example, enter python detection.py {calc_id}. Enter whatever calc_id from-config_cross.py that you want to execute, correction.py can be executed similarly. correction.py generates and saves corrected tracks in a folder and along with that generates a correction file, that contains the nature of the correction. The code base also contains a file called inspect_fitting.py which you can use to check corrections for individual tracks, enter: python inspect_fitting.py {calc_id} {track_name}.

2. Methodology

To correct the altitude differences we find at the crossover points we plot the z differences along the track at these points, and try to fit a function on the values we are getting. The function is a sinousoid with a fit order. This fit order correspondes to the distance scale at which the corrections are happening. For example given below is a track which has fundamental distance of 10,000 kms (fundamental distance corresponds to the track lenght) and a fit order of 3, therefore here the correction vales have a scale of 10,000/3, or approximately 3,333 kms.

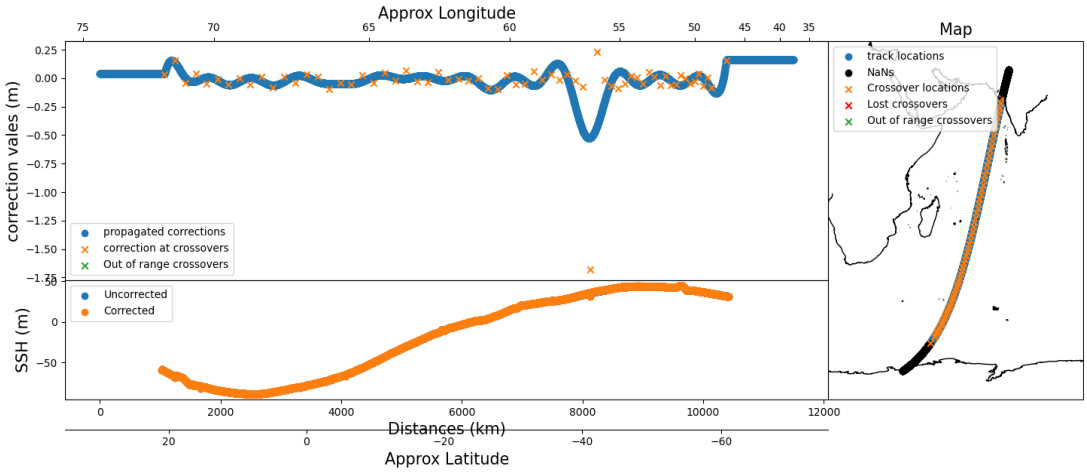
sat_correction_3: Fitting fil_T0009.dat with order 3 and 8 parameters



(These plots are generated via inspect_fitting.py)

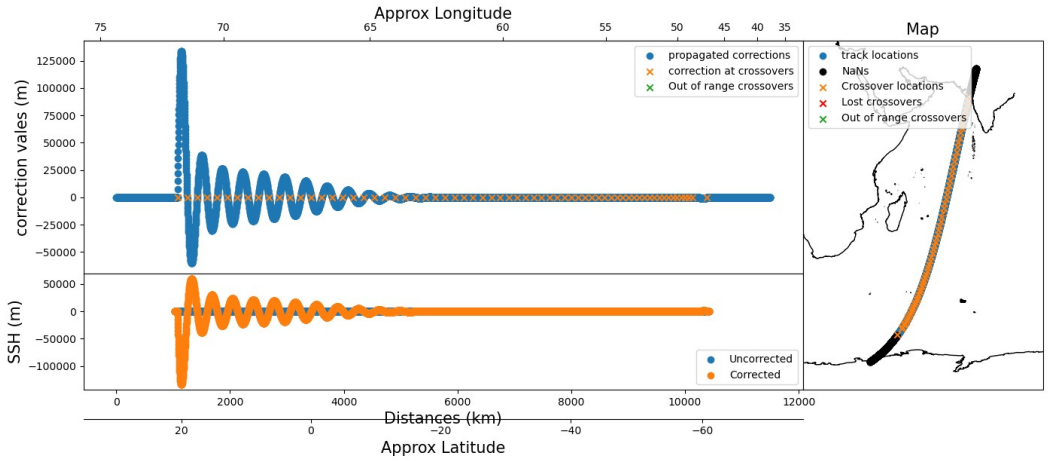
Increasing the fit order reduces the distance scale and gives the function more room to manuever and match the crossover differences as shown below.

sat_correction_13: Fitting fil_T0009_bdn.dat with order 13 and 28 parameters



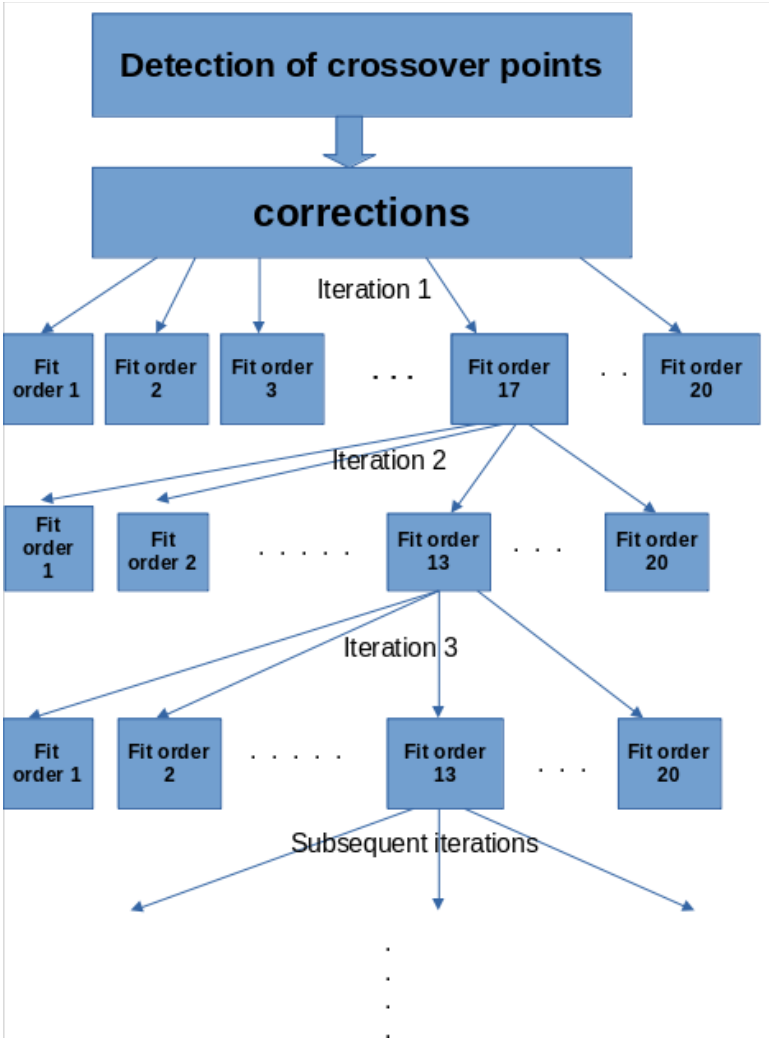
However as we keep on increasing the fit order, at some point when the scales at which corrections are happening become less than the distances between the crossover points, the function overcompensates and the corrections end up making the original track more innacurate (This is also called dancing of the correction vales). This is shown below.

sat_correction_30: Fitting fil_T0009_bdn.dat with order 30 and 62 parameters

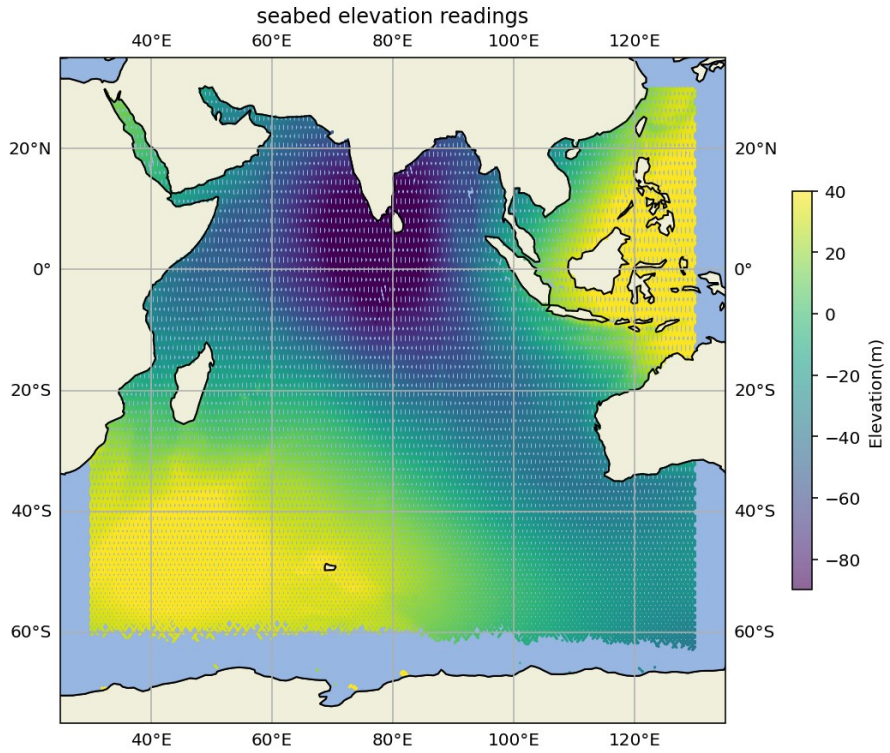


Therefore there exists a sweet spot which gives the best results. The main objective of this project is to find that optimal fit order. To find that optimal fit order we take a region with a dataset that contains n number of tracks (for example the indian ocean region, given below).

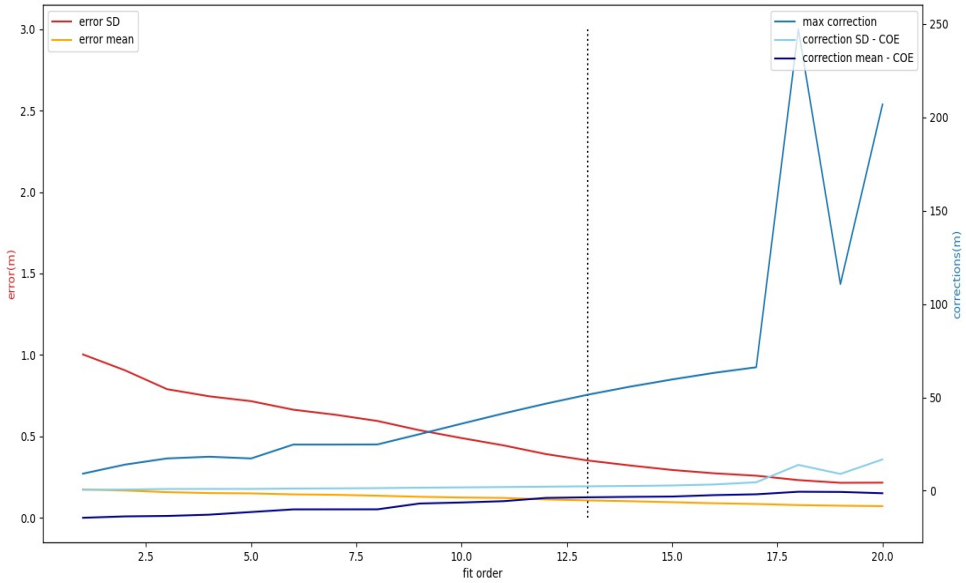
This process of repeated corrections is depicted in the flowchart given below.



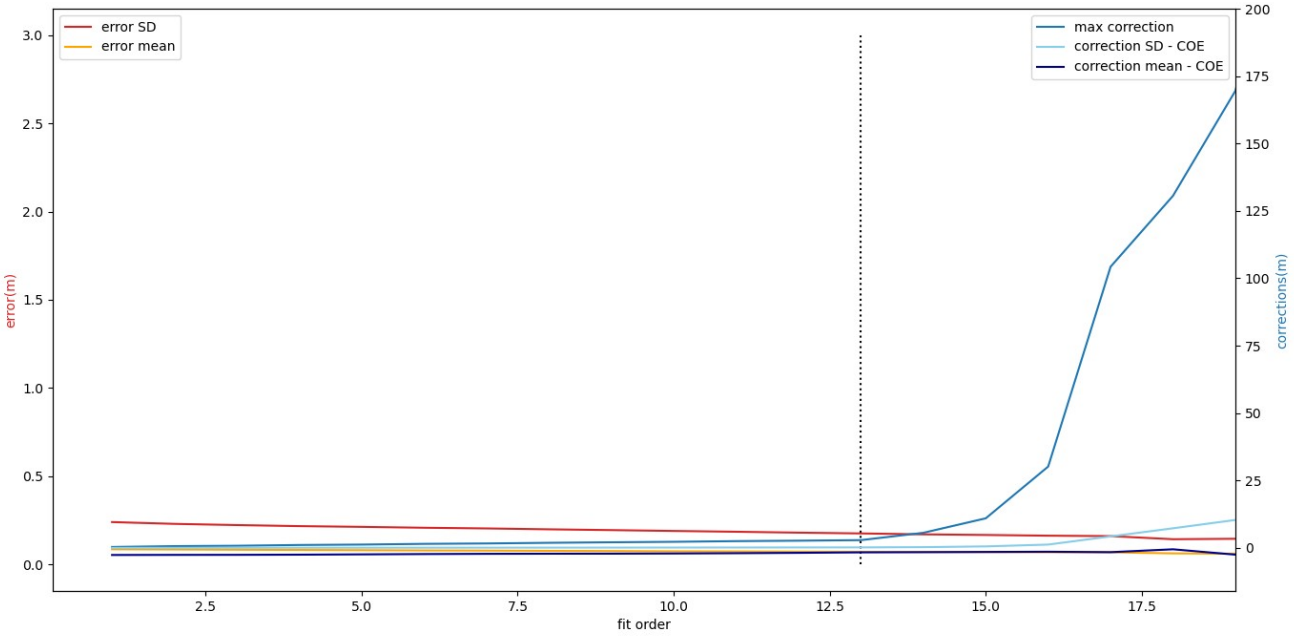
2.1 Terrestrial Data (Indian Ocean)



We then detect all the crossover points between the tracks. To correct the tracks efficiently, we break them if there is a significant gap between the data ponts, here we taken 50 kms as the threshold for breaking. This is then followed by corrections, we plot the mean and standard deviation of the errors, along with the correction statistics that give us an idea of just how the correction vales- are behaving, we adjust the correction statistics to acount for crossover errors.

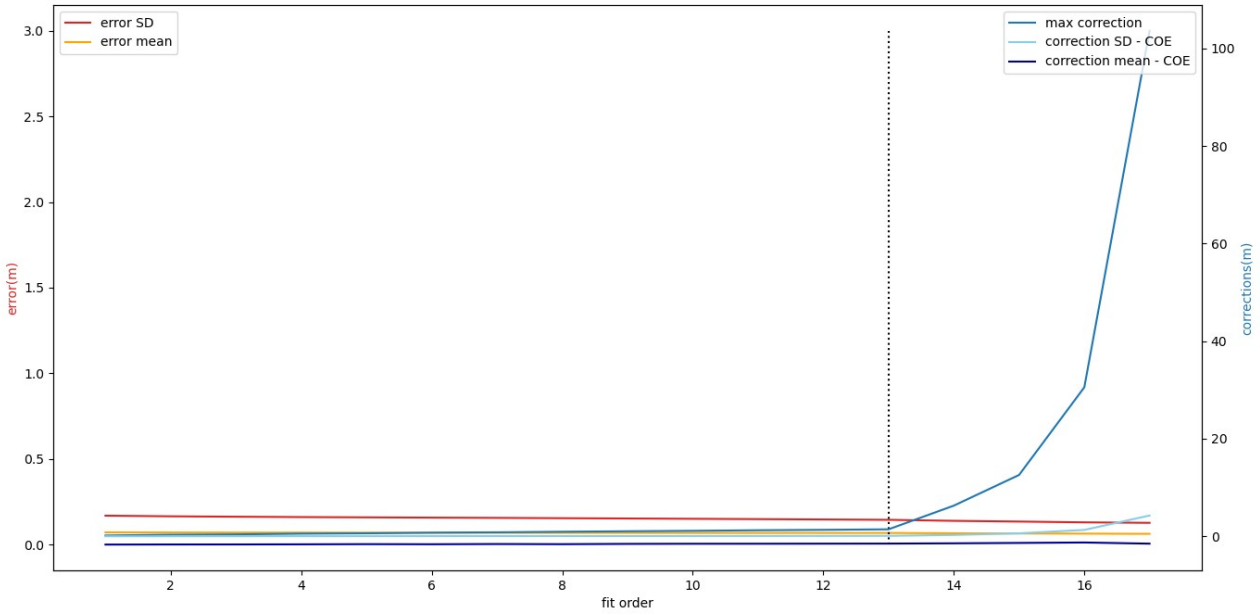


We plot these vs the fit order. It is observed that for this data, there is massive increase in the maximum correction at fit order 18. This indicates overcompensation (dancing of correction vales). Therefore it is suggested to stop at fit order 17 to acheive the best result. Now to acheive better results, we take the corrected tracks and perform corrections on them again. We once again plot the errors and corrections vs the fit order to select the optimal value.



(These plots are generated via correction_order_helper.py)

This time we see a significant increase in maximum corrections from fit order 14. Therefore we stop at 13. We can repeat this iterative process of recorrections, however it will have diminishing effectiveness after each iteration. The results of the third iteration of corrections are shown below.



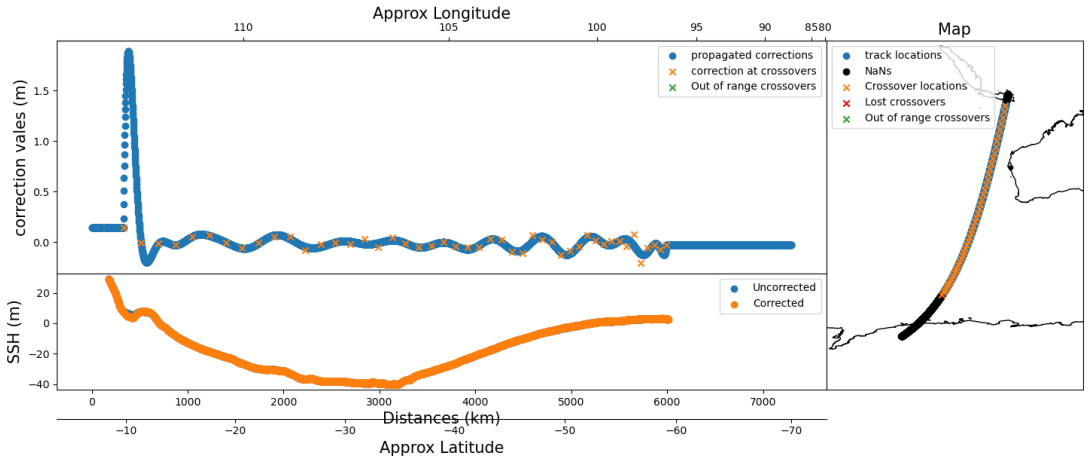
- We can again see that 13 again is the optimal fit order. As we go for further iterations we can see that 13 is a recurring value. Below are the results of the fourth iteration.

This is not a coincidence, 13 is the value we get when we apply the Niquest Theorem to determine the fit order for this data. This is done by taking the fundamental distance (track length of the longest track, 10,000 kms in this case) and dividing it by twice of the maximum crossover distance among all tracks (372 kms for this case). Therefore we get

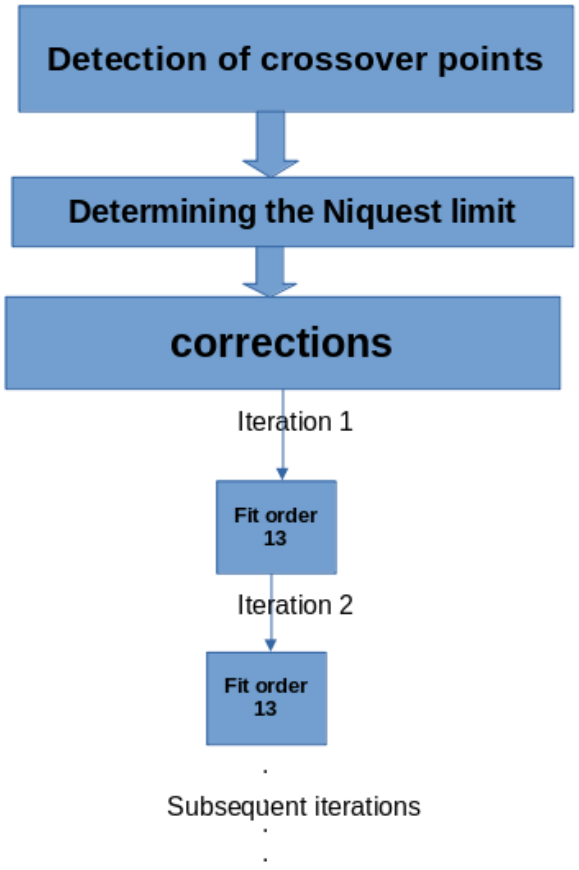
*10000/(2*372) = 13.44 = 13 (taking the lower limit)*

now we can see that in the first iteration of corrections we can go beyond this limit(the dotted line in each graph is the Niquest limit). However when you actually look deeper into the data, going track by track you can find some improper corrections that get hidden due to there smaller magnitude. One such track is shown below, at fit order 17.

sat_correction_17: Fitting fil_T0177_bd3.dat with order 17 and 36 parameters



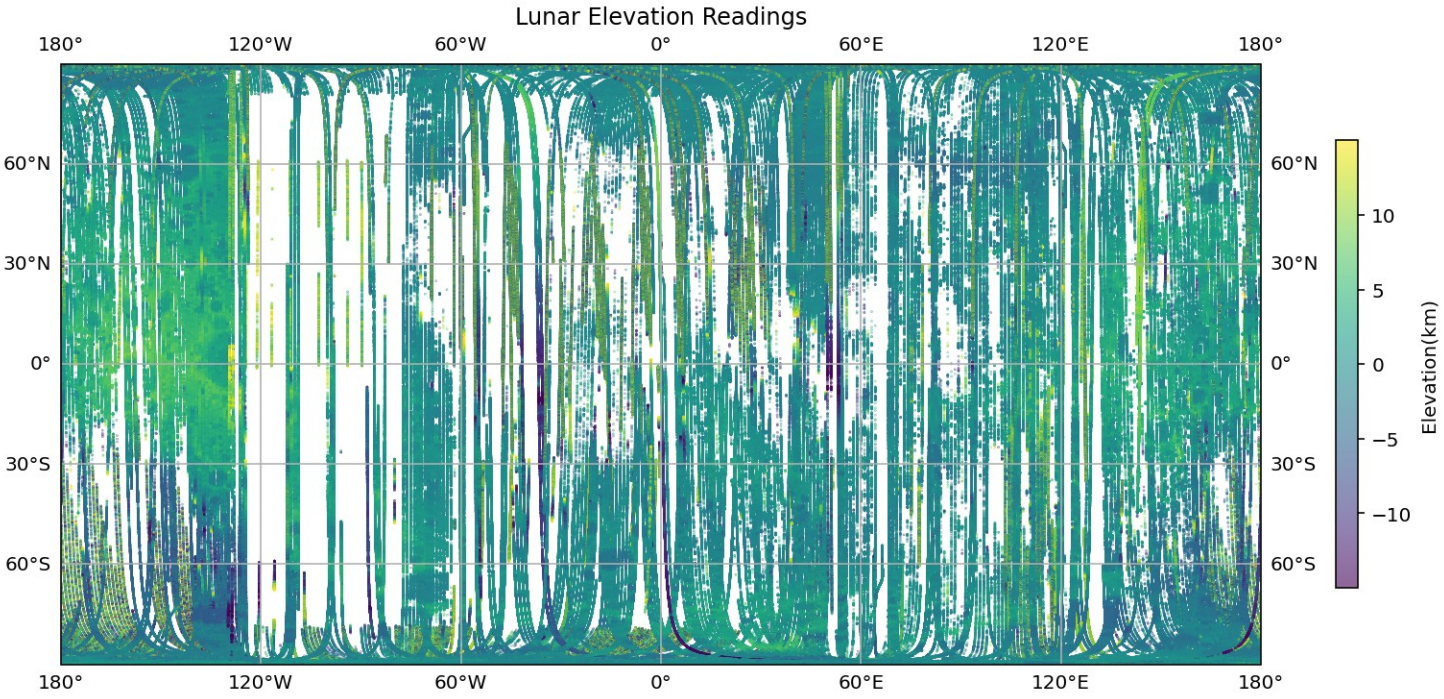
however, these issues do not occur if we stick to the niquest limit of 13 for the given data. Hence it is best to follow niquest limit. It is also noted that in the subsequent iterations we anyway get 13 as the optimal value, both mathematically (from the Niquest theorem) and statistically (from the plots).



However, the value of 13 that we caclulated from the Niquest theorem is based on the worst crossover distance. However that only affects a single track. Each track will have its own Niquest limit based on the maximum crossover distance for that track and the length of the track (fundamental distance). Therefore all tracks do not have the same constraints, and we can calculate the individual Niquest limit for every track and implement the fit order based on that. Doing corrections based on individiual track Niquest limit gives us an error mean of 0.072 meters and standard deviation of 0.19 meters, this is better than the values we get at fit order 17, while the max correction of 45 meters stays within the exceptable limit (this max correction value is due to a likely erronous track near Sri Lanka which has a crossover difference of 50 meters). In this method fundamental distance is also changed for every track based on length of the track where all the crossovers occur. Therefore the best method for corrections seems to be to set the parameters for each track seperately.

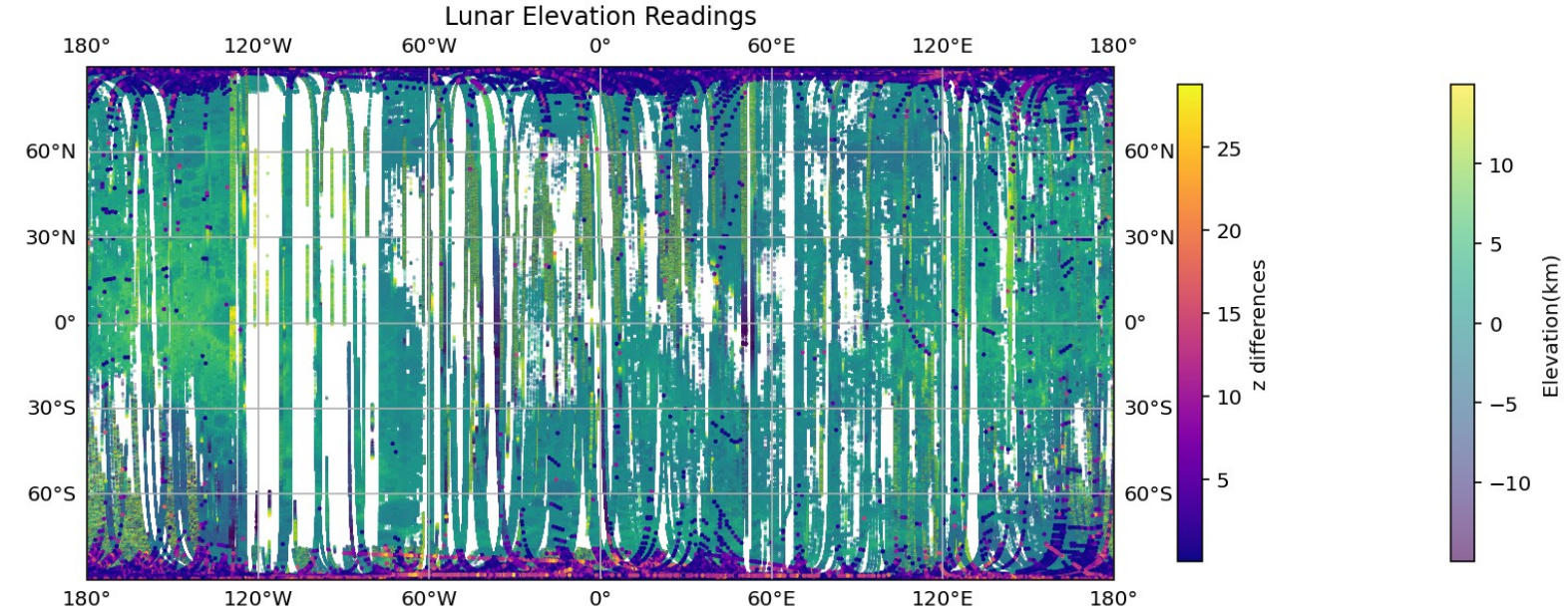
2.2 Lunar Data (Chandrayaan-1)

We have tried to apply the same corrections to the altimeter data from the Lunar Laser Ranging Instrument (LLRI) onboard Chandrayaan-1. Firstly we map out an elevation map from the data.



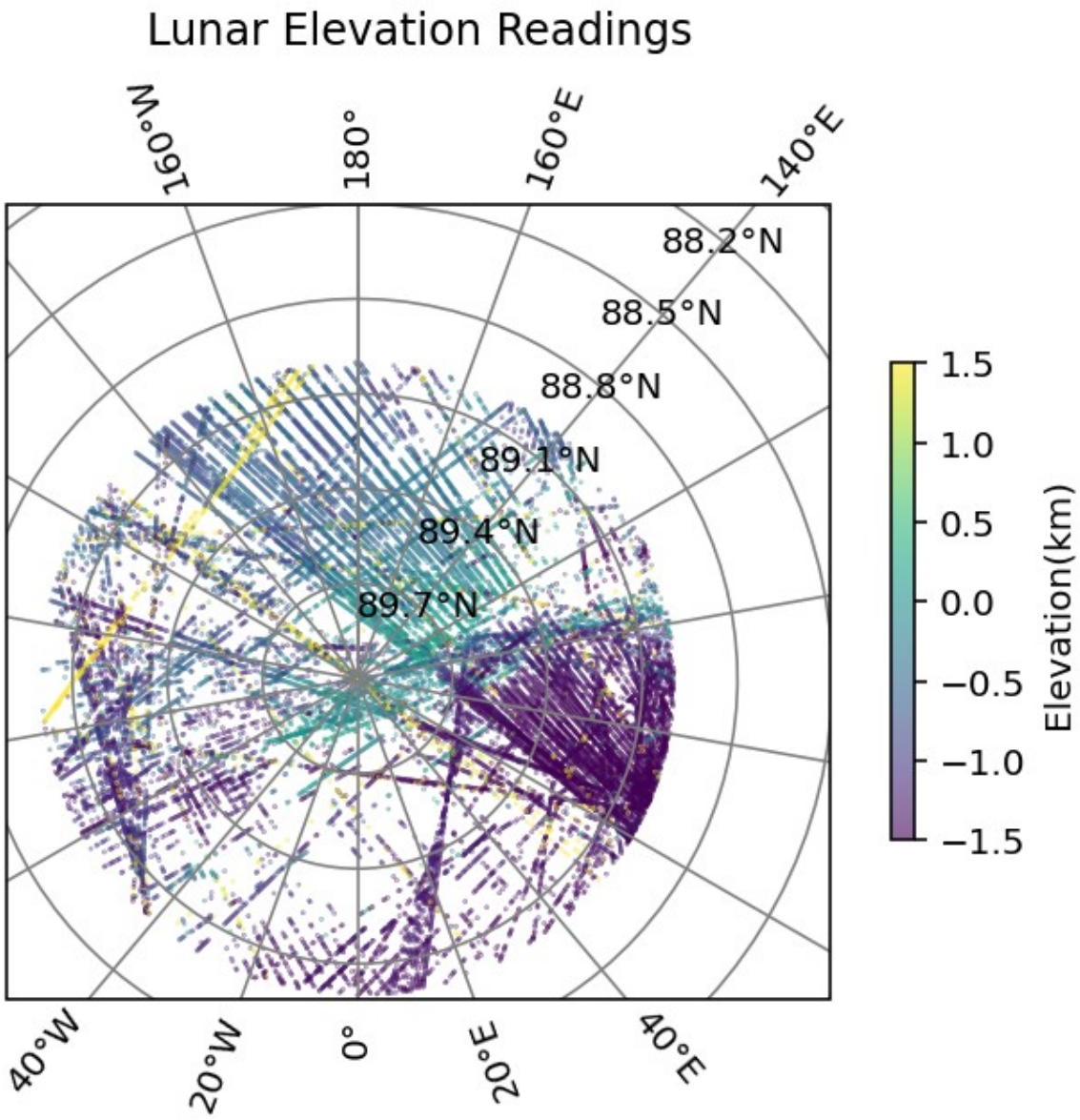
Note that LLRI data has four z (altitude) values (z1, z2, z3, z4). To make this map, and to go forward with the rest of the project we have only taken the ‘z1’ values.

Detecting and plotting the crossovers over this, we can that most of the crossover occur near the south pole and the north pole.

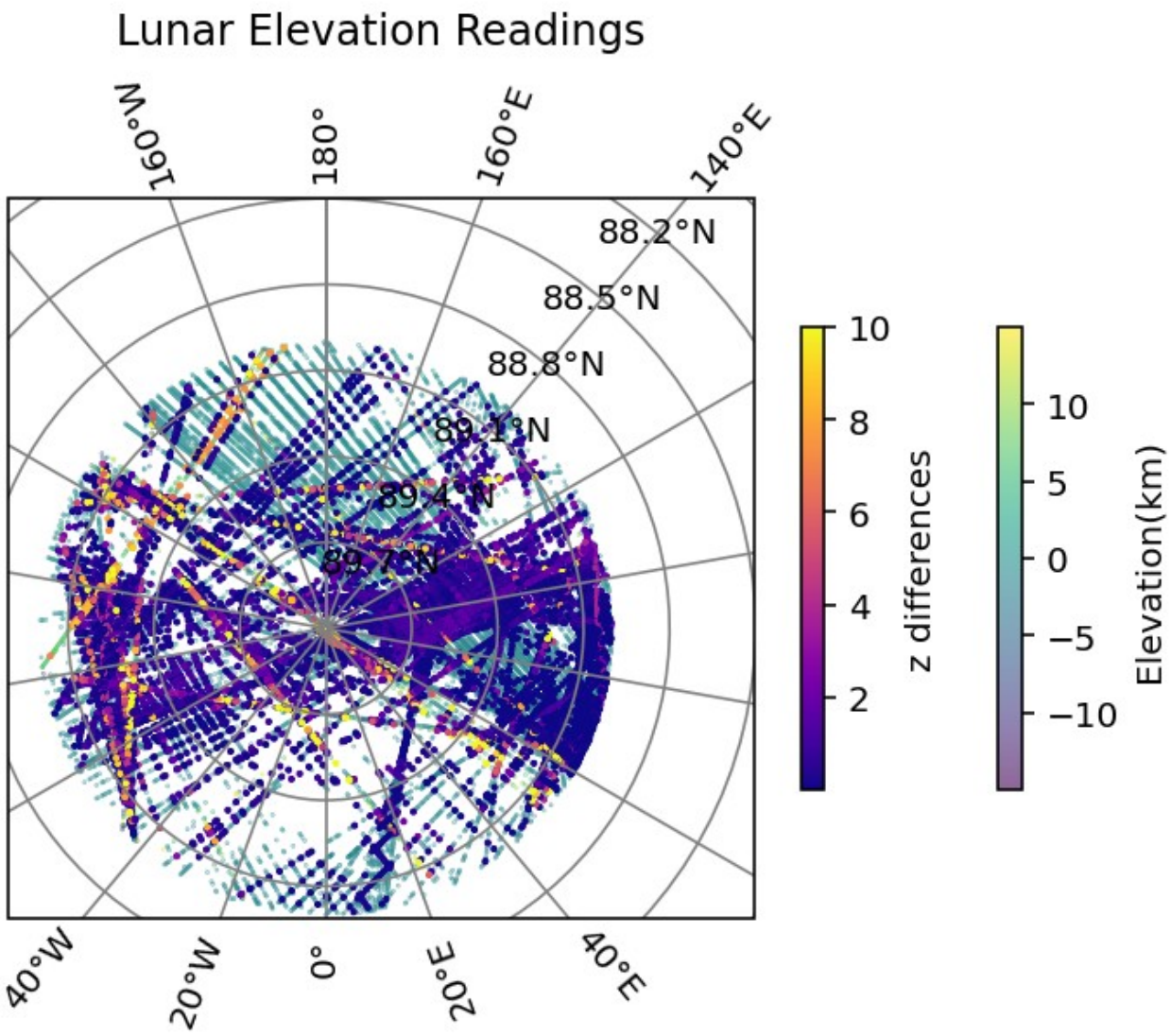


Furthermore we can see that the density of the tracks is greater at the poles, with less breaks and inconsistencies within the data. Further considering that its impossible to do corrections without crossovers, it was decided to limit the scope of this project to polar regions only.

To move foward with the corrections all data poins within the 89^{o} N latitude circle was chosen. The tracks are shown below (range of the map is set from -1.5 to 1.5 km to better showcase the differences, actual readings go up to 10 kms).



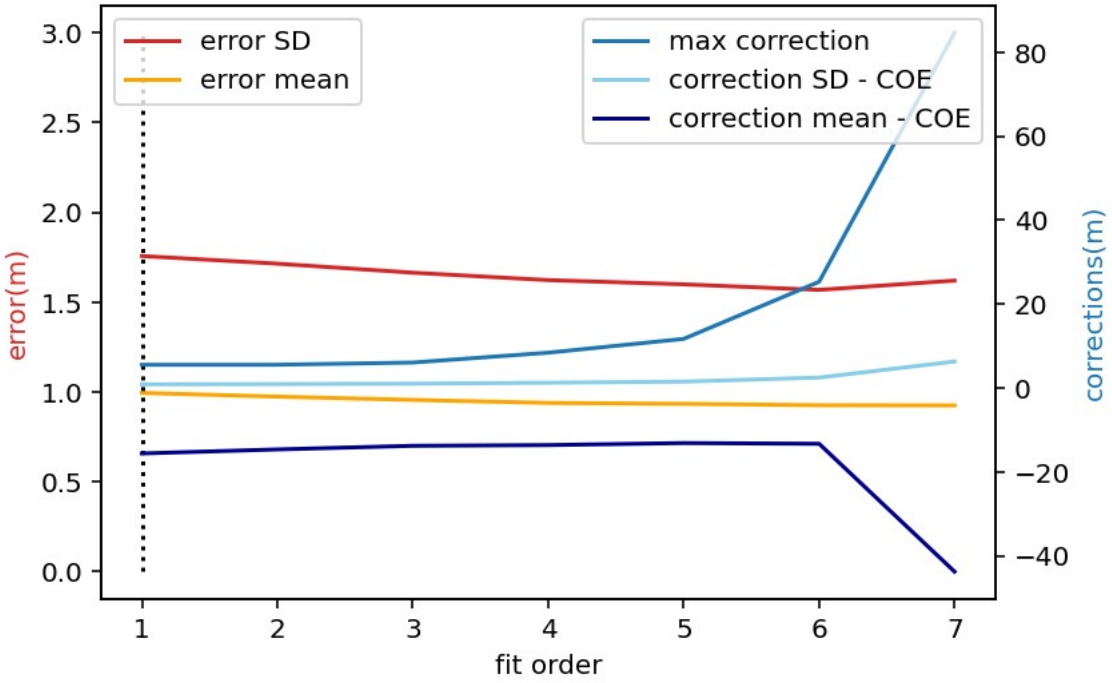
mapping the crossovers along with the crossover differences gives us the below plot.



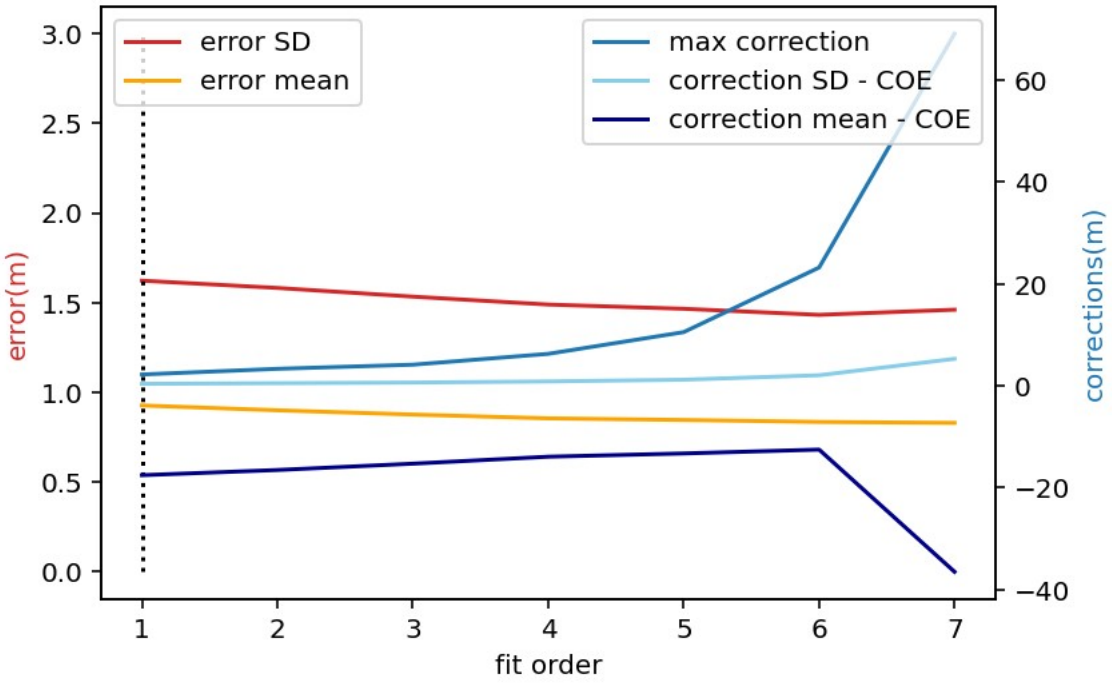
We find that the maximum crossover difference for a track is 28 kms. Considering the fundamental distance of 100 kms, we get the Niquest limit:

*100/(2*28) = 1.78 = 1 (taking the lower limit)*

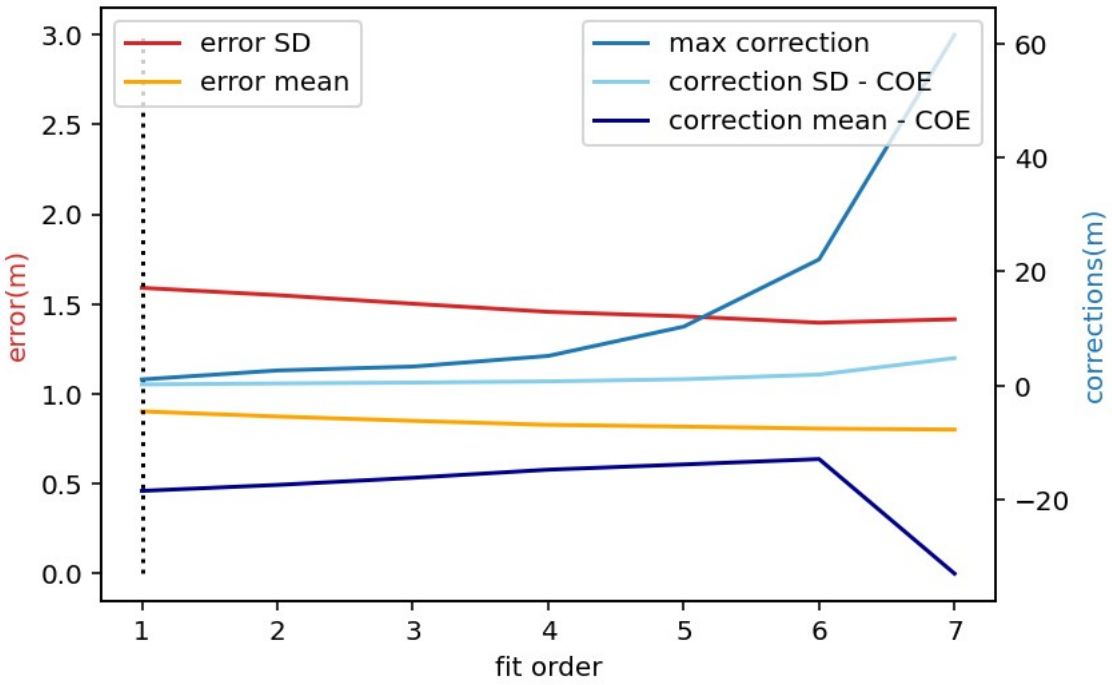
Hence we do iteration wise corrections with fit order 1. To do the corrections we have broken the tracks based on the time difference between the readings of successive data points. A larger than normal time difference suggests a gap in the data. We have taken 4 minutes as the threshold for breaking. The results for other fit orders are also shown below.



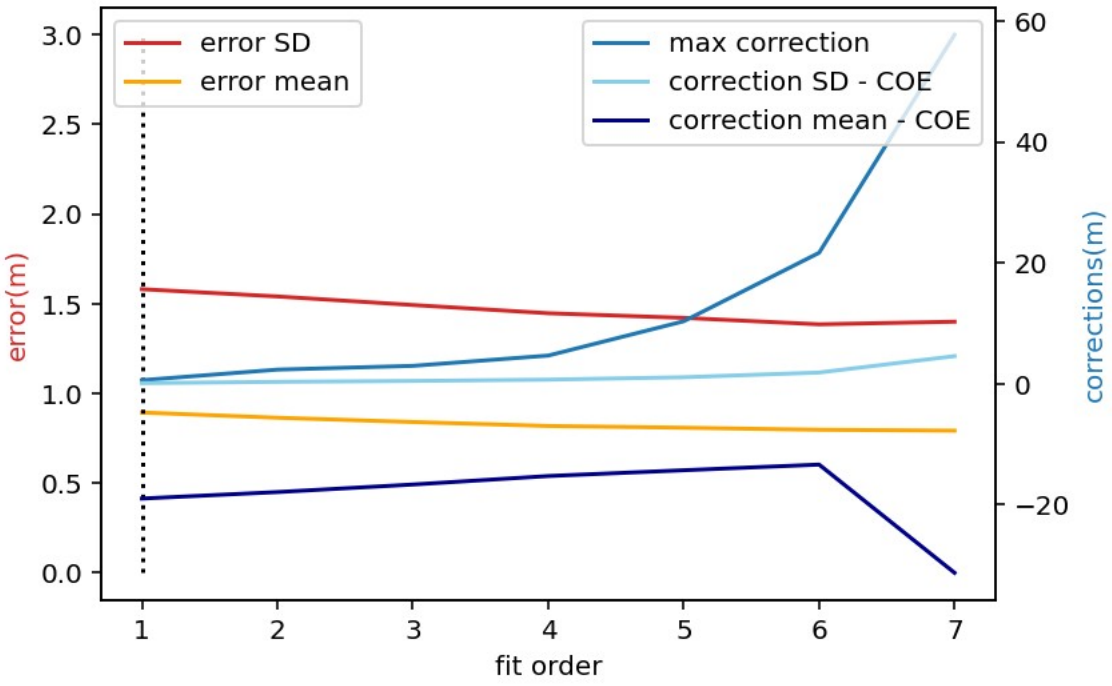
Iteration-1



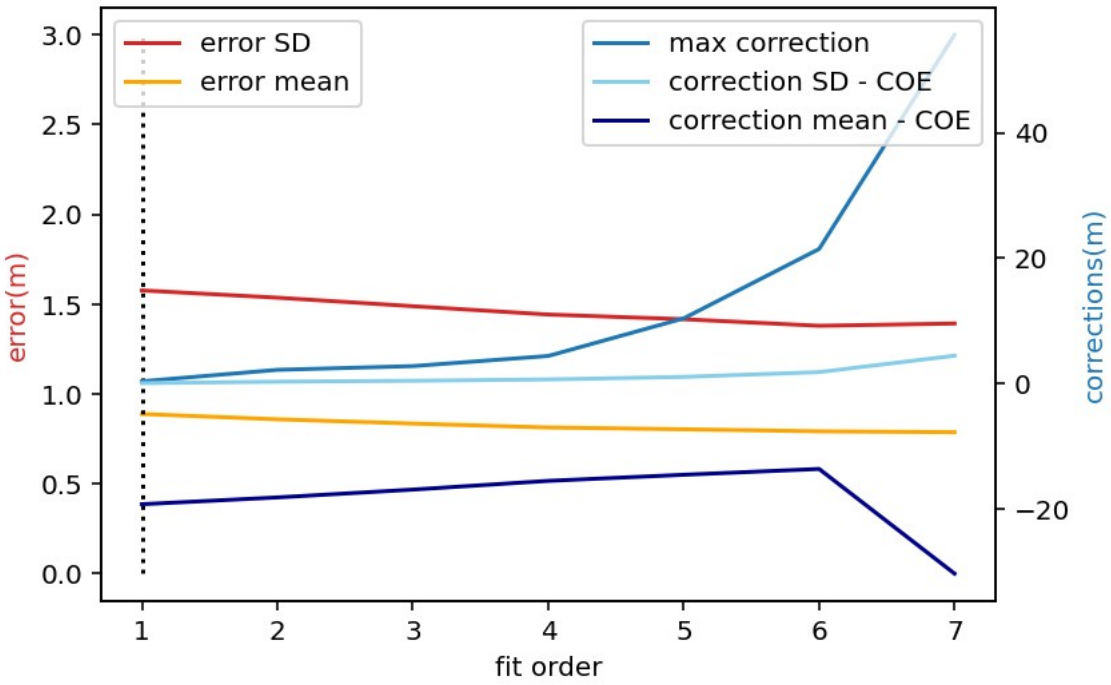
Iteration-2 (after taking fit order 1 for iteration 1)



Iteration-3 (after taking fit order 1 for iteration 1&2)



Iteration-4(after taking fit order 1 for iteration 1,2&3)

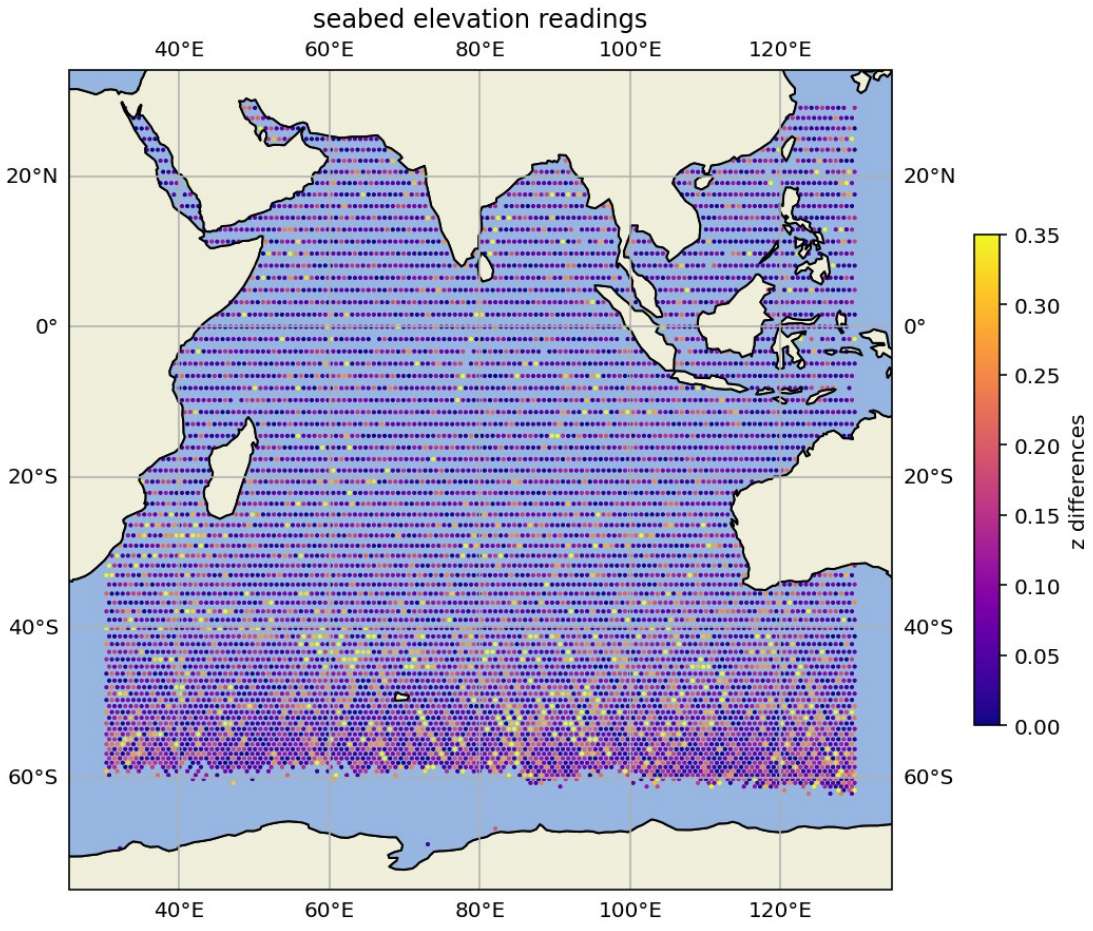


Iteration-5(after taking fit order 1 for iteration 1,2,3&4)

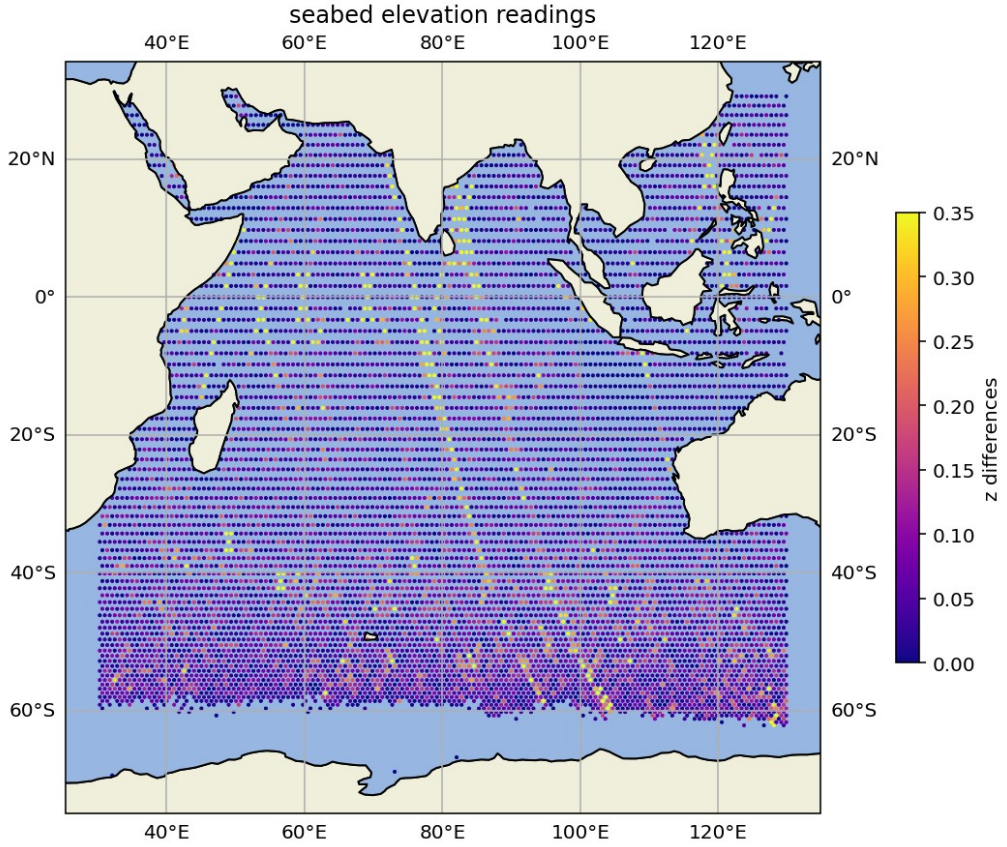
We can see improvements after every iteration, although with decreasing effectiveness. We go till five iterations at fit order 1.

3. Results

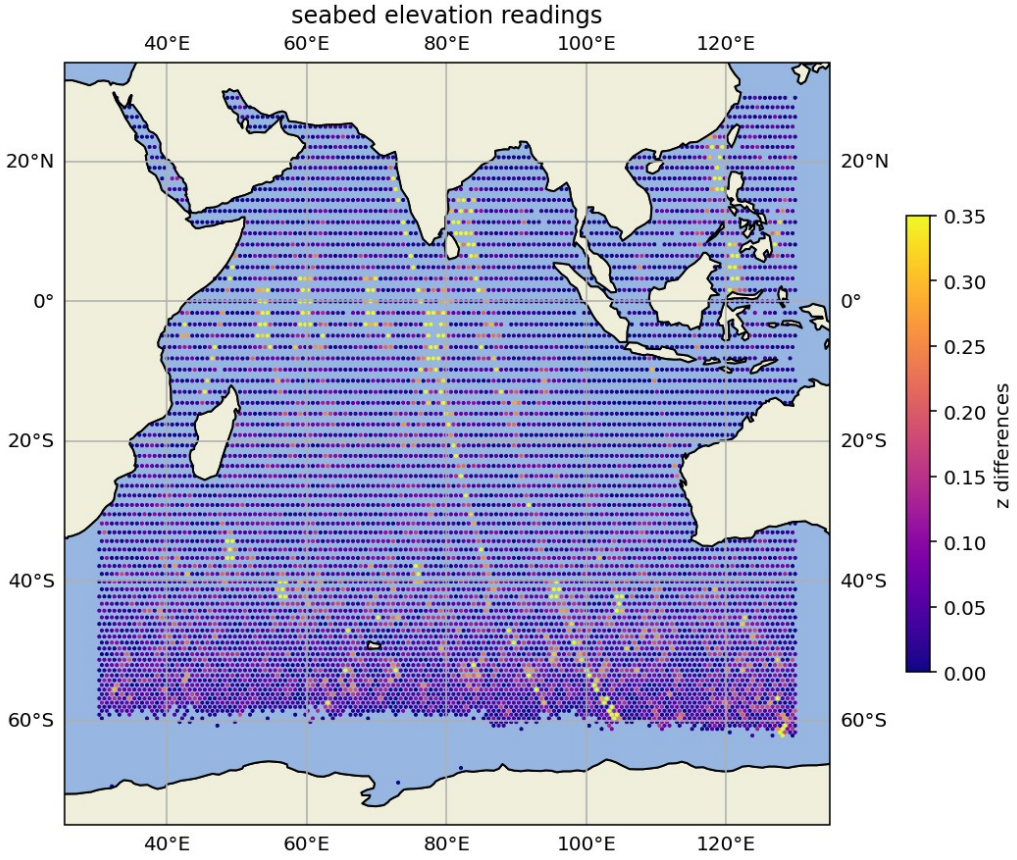
3.3 Terestrial Data(Indian Ocean)



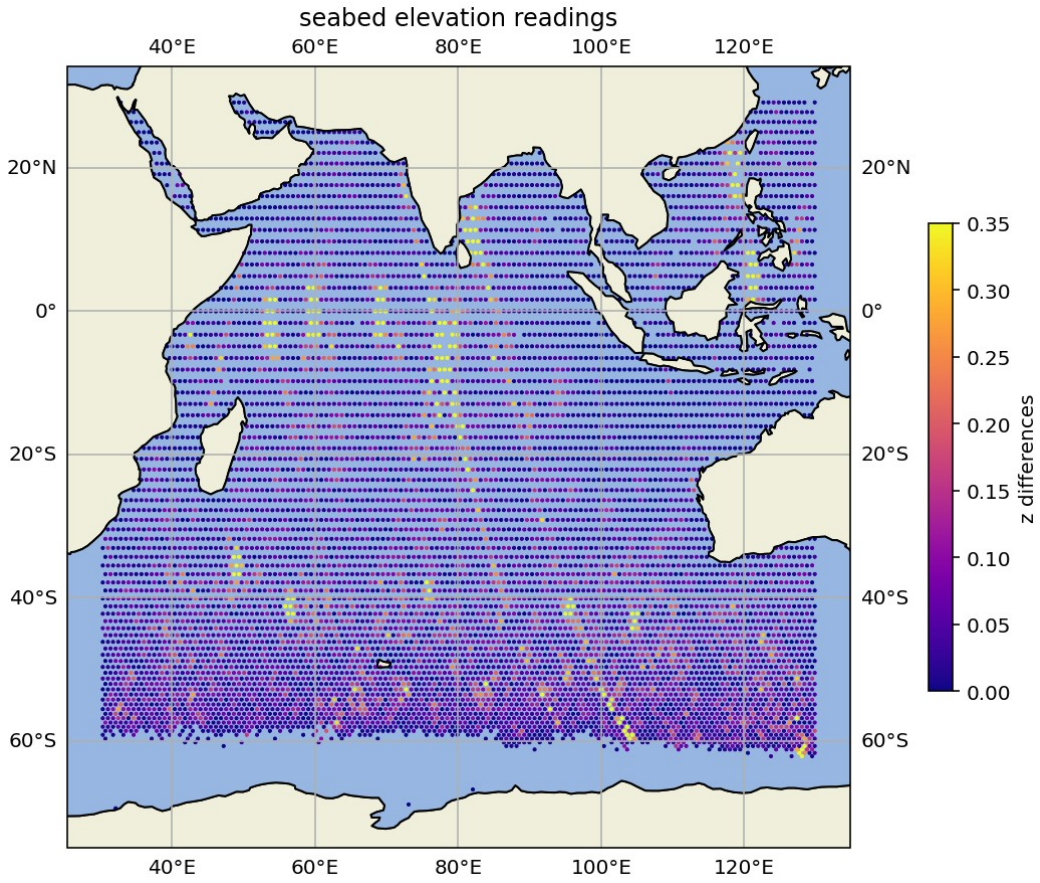
The original crossover differences are shown above (the scale has been set till 0.35 meters to better highlight the differences, actual crossover differences go upto 50 meters). Iteration wise results are shown below:



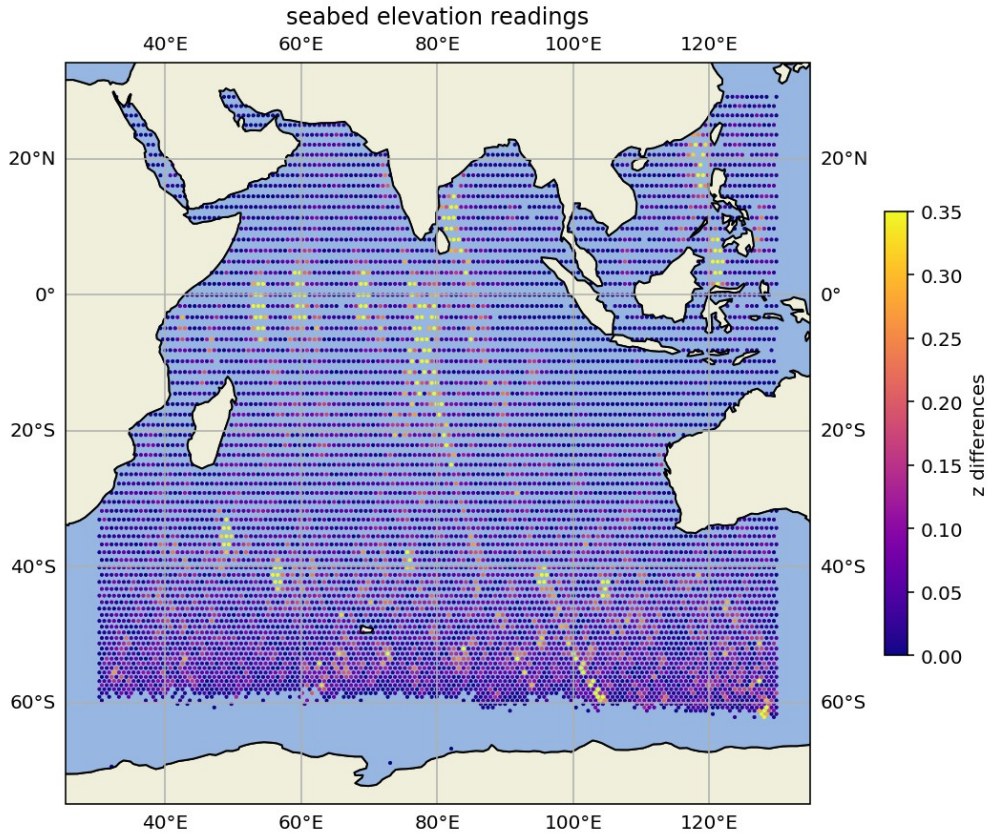
Iteration-1 (at fit order 17)



Iteration-2 (at fit order 13)



Iteration-3 (at fit order 13)

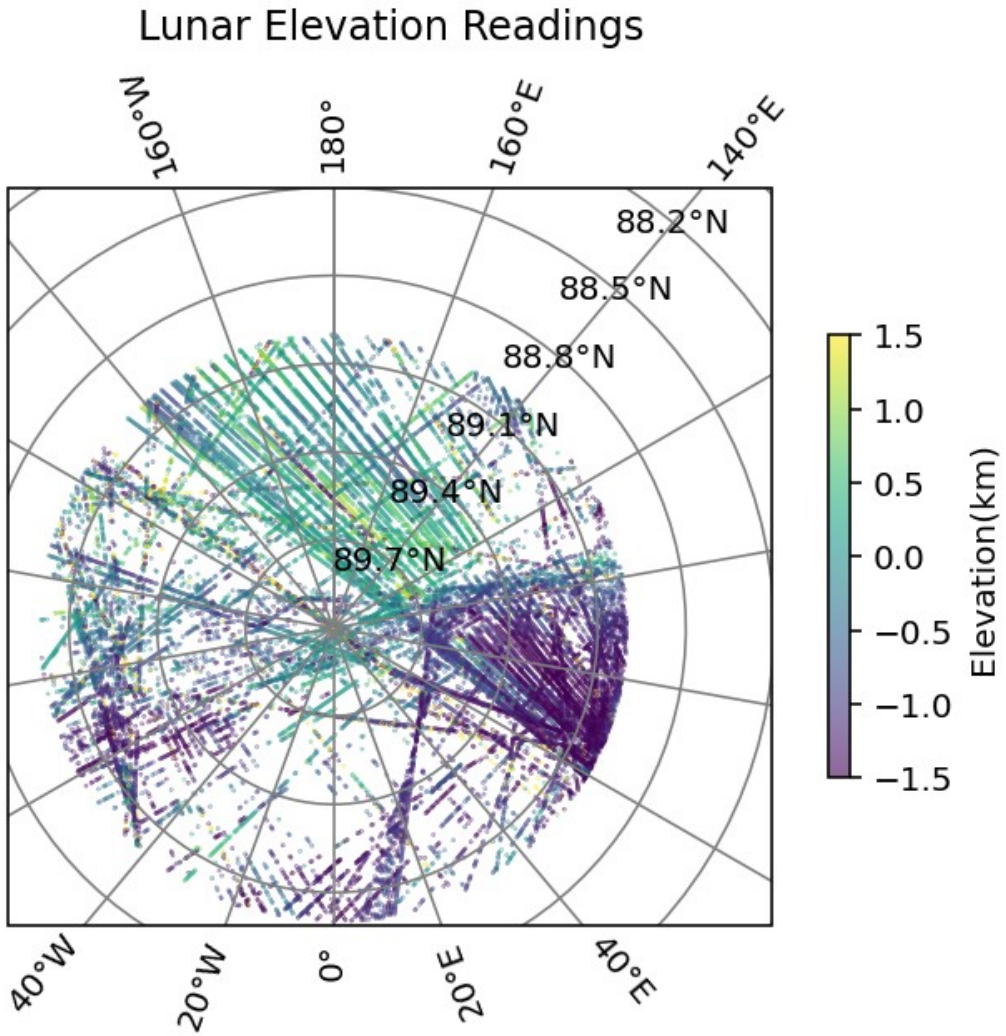


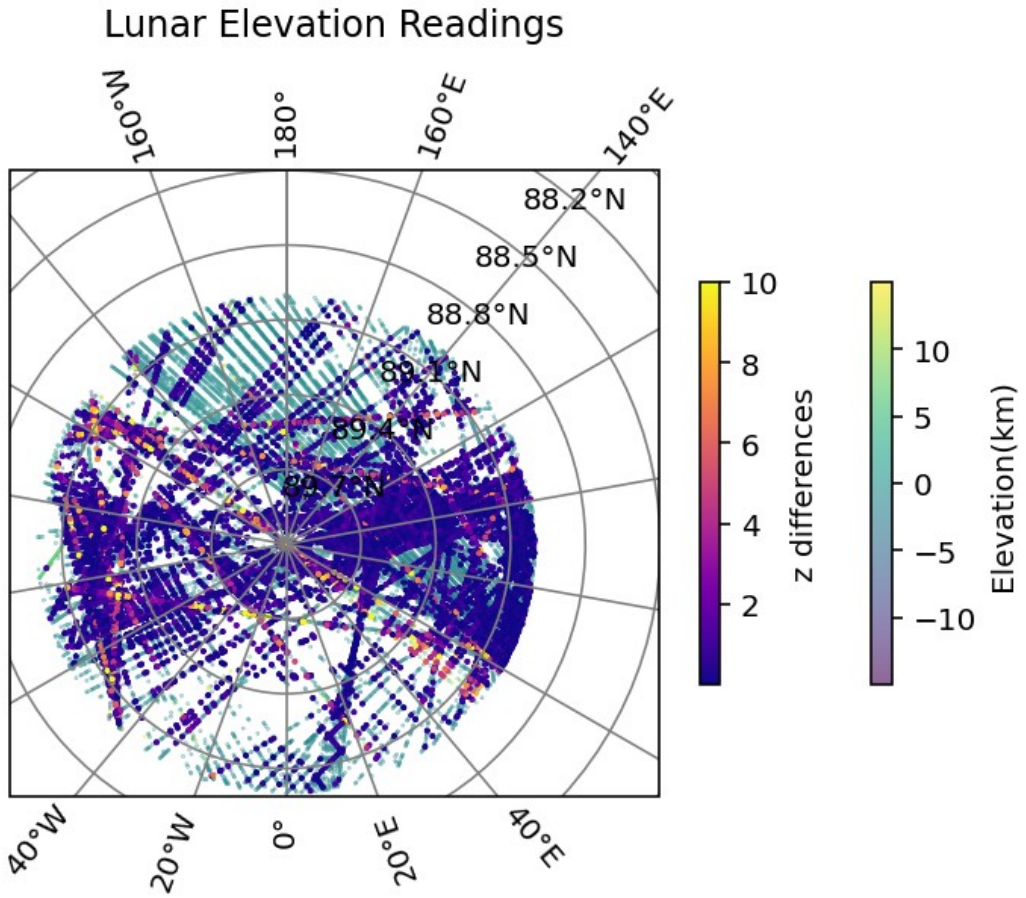
Iteration-4 (at fit order 13)

We can see that overall, altitude differences at most crossover points have improved.

3.2 Lunar Data (Chandrayaan-1)

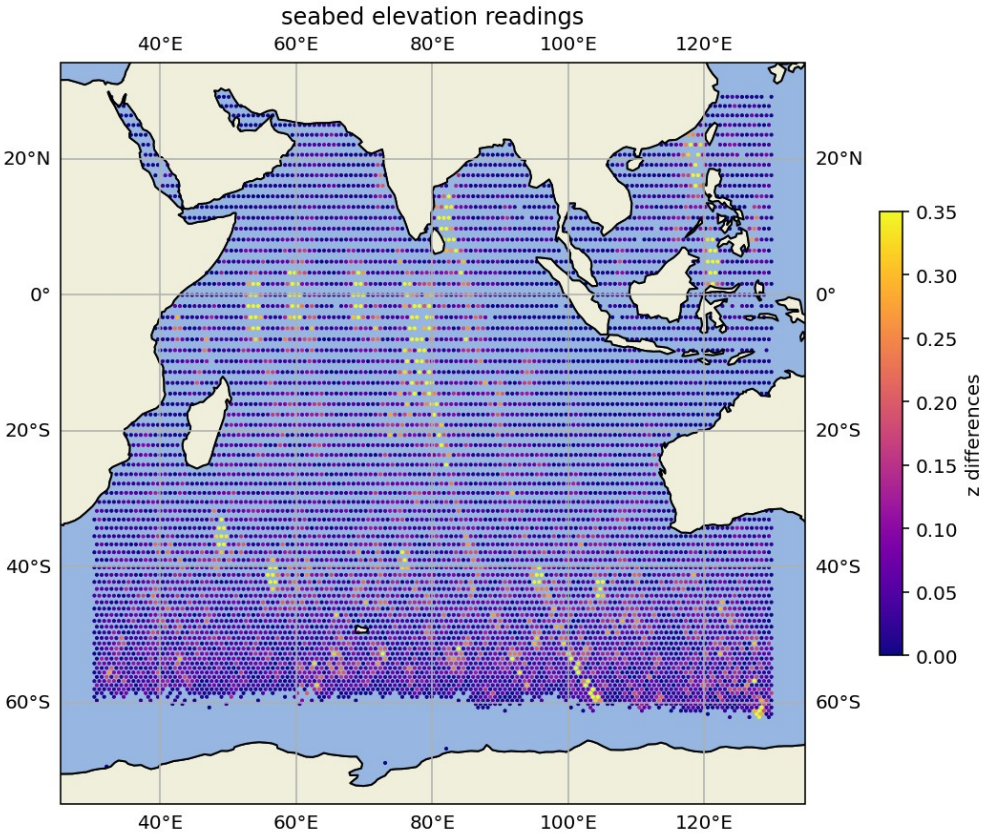
The results after corrections for the north pole data are shown below.



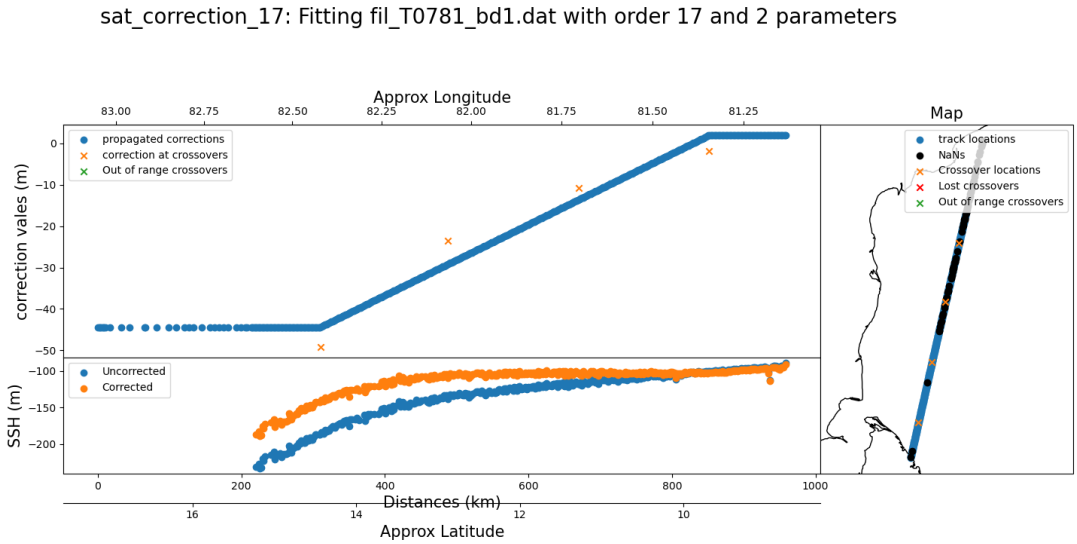


Errors have been minimized, although the erratic nature of the data limits the scope of the project and the magnitude of corrections.

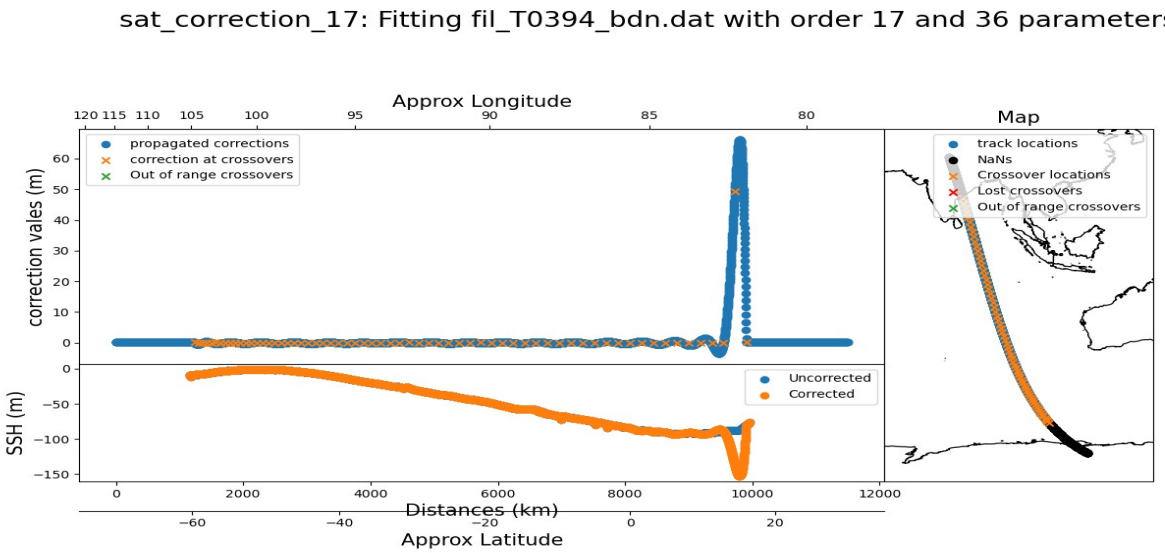
4. Limitations



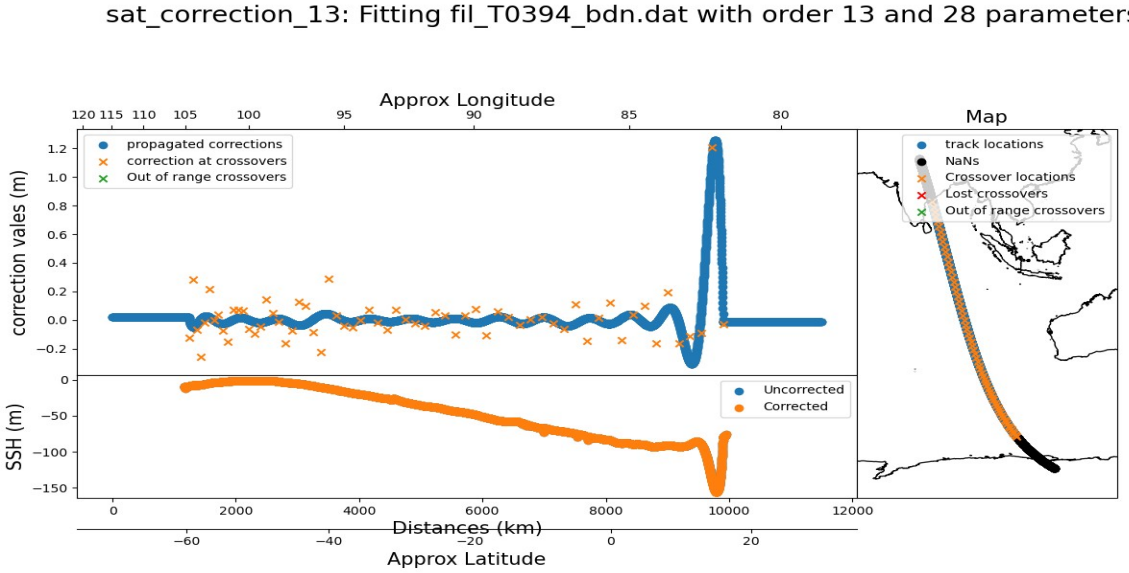
For the Indian ocean region by iteration 4 we can see certain patches developing that have unsually high crossover differences. This happens due to tracks or points having unusually high crossover differences. For example shown below is a track with unsually high crossover differences (essentially a bad track).



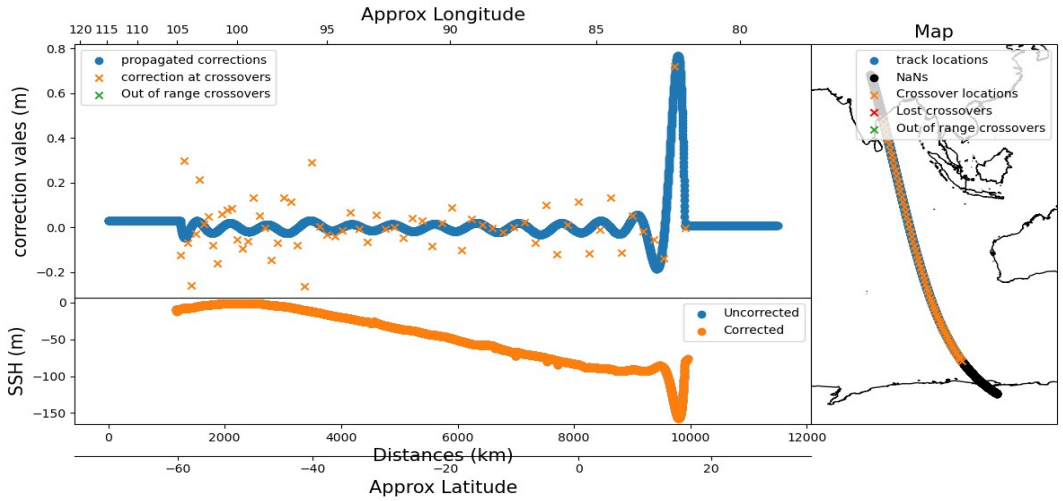
This crosses over with another track that is a relevatively good one.



We can see that except for the crossover point with the bad track, the rest of the crossover points are alright. However in trying to correct the tracks and match the worst point the function oscillates, leading to the error propagating. This can be seen in the subsequent iterations.

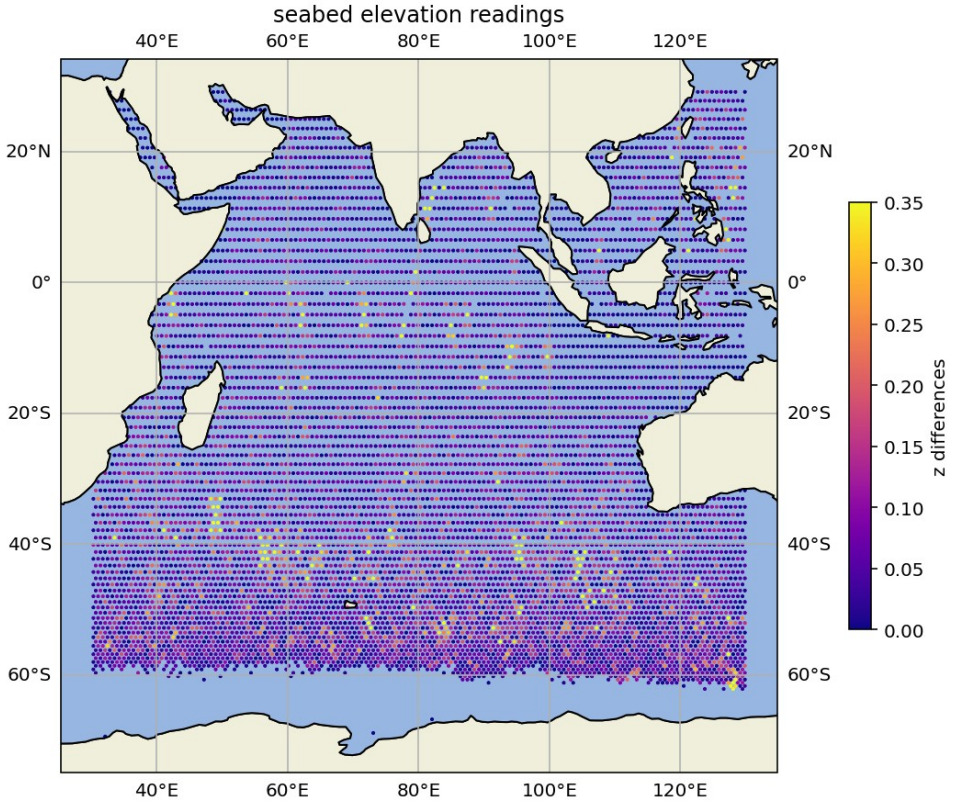


Iteration-2



Iteration-3

we can see while the error at the worst crossover point is reduced significantly, the crossover points nearby are affected (in certain cases, entire tracks are affected). This happens to any track crossing the bad track, and can also happen due to erronous data points in an otherwise good track (that can happen due to islands). Ultimately leading to patches with unusually high crossover differences surrounding that point/track. Hence it becomes important to filter out such points/tracks. This can be done using the xdiff_min & xdiff_max atrributes in the config cross script. Given below is the result we get if we exclude points having crossover differences of more than two meters.



This is the result after three iterations. While errors are contained, the erronous tracks/points are not corrected.

Regarding the lunar data, we can see that even after corrections there are large crossover differences of several kilometers. This is due to the poor quality of the input data. The data point intervals are not constant for every track, even within the same track. For certain tracks the number of crossover points exceeds the number of data points. Even corrections at individual Niquest limit do not yield satisfactory results. Correction at fit order one restricts the function, leading to a lesser reduction of errors. These factors combined limits the scope of the project.

5. Conclusion

This project showcases that there is a definite method to find the optimal function for corrections. It also showcases the various precautions that need to be taken to ensure that the quality of the corrections is acceptable. The project has refined the pre-existing codebase of the PyReX algorithm along with some new additions. It has laid the groundwork for further studies and research into this topic.