Here is the summary of my work on NagaSurya 2.0 in the 6th semester.

4th of February 2025:

visualise_fits.py was the first code that I wrote which was an attempt to refresh the way a fits file gets plotted by default. Displayed the metadata of the fits file to see if I had previously missed observing anything related to the fits file's nature.

alm_sunspot_analysis.py made use of the sunspot_numbers.txt and the sunspot_numbers.csv that I had downloaded from the internet. This dealt with twin plotting the magnitudes of the precomputed alm values (spherical harmonic coefficients) for each map, along with the number of sunspots appearing in that map – over all the years. there was an issue that the sunspot data predates the alm values by an exceptionally large duration (1700s vs 2010s). Note that I was not plotting all the alm values, was plotting the average of the top 5%, top 1% and the top 0.5% of all the alm values against the month-year/Carrington map number. Also used the gaussian filter to smoothen the alm values data. Other than visually plotting and trying to find the relation between the number of sunspots and the average of the magnitudes of the alms that cross the three thresholds, I also went ahead to use to Pearson correlation as a metric to find the relation between the smoothed alm values and the sunspot data. Tried performing a continuous wavelet analysis on the alm values, in hopes of finding patterns – performed on the three different cutoff percentiles. I do not fully understand how wavelet transforms and wavelet power plots work. the output plots are stored in the plots folder, and from there it is probably visible that there has been high amounts of high frequency activity in the first few years of the study (2010-2016) which is followed by a period of high frequency fluctuation of way lower intensities and which eventually turns into high frequency, high intensity towards the end of the study period (towards 2023-2024). And extremely low frequency fluctuations in general has lower intensity than the other higher frequency fluctuations.

12th February 2025:

pw.py is a simple code that processes a given set of CR maps and has the beep feature, plots the map, saves error metrics after reconstructing the precomputed alm values. Just refreshing old code knowledge.

maggify_alm_values.py creates new alm value files for each of the CR maps but now includes a new column that has the magnitude of the alm value instead of being separated into imaginary and real parts.

13th February 2025:

I do not know why but there's **split_alm_csv.py** that converts the magnitude of alms back into their real and imaginary parts and stores them as new files of alm values. I am not sure how it works, or why it is useful.

recons_save.py is **pw.py** but with the improvised feature of saving the reconstructed values continuously, just like how the alm values were initially being calculated and stored – to ease restarting of the computation from wherever it was left off at.

1. alm_mag_pred.py which was a very bold first attempt at predicting the alm value magnitudes using linear regression. I used to CR maps to interpolate and to find out the middle CR map's alm values i.e., deriving the solar magnetic features of Carrington Map 2097 by using CR maps 2096 and 2098 – trying to intrapolate I guess and estimate the midpoint of all the values.

2. alm_real_imag_predict.py is next, which does the same as **1. alm_mag_pred.py** but now instead of estimating or predicting the midpoints of just one feature (the magnitude of the alm value) – now it is doing it separately for the two components of the alm values column – which is the real and the imaginary parts. Both the values are predicted/derived in the same way although – averaging out from the two adjacent CR map's alm values. Just that alm magnitude got replaced with the real and imaginary parts of the alm value.

3. reconslocal.py is used to create a plot with the following subplots – the original CR map, the map reconstructed with the original precomputed alm values and then another map reconstructed with the predicted alm values. So this lacks the data for the first map (2096) and the last one (2285) as they do not have map-data on both sides of the number scale. This code runs on the local system and the directories for it to refer to are locally existing. It also prints the b_avg values for all the three maps in the corner of the main plot – hence there are multiple maps generated, one for each CR.

4. reconskaggle.py is the same as **3. reconslocal.py** but now the same code is made to be able to run on Kaggle, making use of the GPU accelerator and the storage facilities of Kaggle.

17th February 2025:

5. reconskaggle-tqdminsteadofrich.py is the same as **4. reconskaggle.py** but I have another function called reconstqdm() along with the original recons() function, where ive replaced the colourful rich progress bar with the simpler tqdm progress bar, so that it shows up on the output logs when I am running it on Kaggle.

18th February 2025:

6. reconslocal-errormetrics.py has the same features of the **3. reconslocal.py** with an additional feature of displaying the MAE, SSIM, and Cosine Similarity values between the original map, the map reconstructed with the originally calculated alm values, and the map reconstructed with the predicted alm values.

7. reconskaggle-errormetrics.py is the same as **6. reconslocal-errormetrics.py** but it is tailored to run on Kaggle and use the dataset inputs from there and store the results and plots there, instead of locally on my system.

8. gecko.py is a simple web scraping code that visits the http://hmi.stanford.edu/data/synoptic.html website and downloads all the fits files of the CR maps in the range mentioned in the code. This was needed because the last time I downloaded fits files (back in July 2024), we had CR maps upto 2285 and the latest ones as of back when I used this program was CR map number 2293 or 2294, I am not sure.

9. plot_bavg_vs_time.py just plots the variation of the b_avg (average magnetic field, in Gauss) of the Sun over the years, as directly computed from the values in the fits files. Had also tried plotting the variation of the northern and southern magnetic fluxes separately but then ended up commenting it out in the code. The logic for that part was not right, the values were not being computed properly – hence, need to look into that the next time I try analysing the magnetic field/flux of the two hemispheres of the Sun separately.

10. try plotting breconsog, breconspred, and b - all avg.py is an attempt to slightly improve the **9. plot_bavg_vs_time.py** and to try and plot the three b_avg values – the original, the reconstructed from original alm values, and the reconstructed from predicted alm values – and to check the variation of all these three values over time. But then the code ends up plotting just the b_avg original over time – without the separation of the north and south fields.

11. 10 but better.py was meant to be what the name says but it ends up plotting the simple graph of how the b_flux (which I assumed to be the same as the average of the absolute values of b) over the years. I spent so many versions of the same code on the same functionality because I wanted the simple B plot vs time to be plotted in perfection, with the right ticks and format.

19th February 2025:

12. KAGGLE selected CR maps lmax60 - reconsdata calc.py is a code tailored to run on Kaggle, such that it computes the recons values for the given set of CR maps, upto a given limit of lmax, and stores the outputs in the reconsdata folder. I used this code to just precompute and have all the recons values and keep them ready for future usage purposes.

13. LOCAL selected CR maps lmax60 - reconsdata calc.py does the same as **12. KAGGLE selected CR maps lmax60 - reconsdata calc.py** but this version is tailored to run locally on my system instead of Kaggle.

20th February 2025:

14. 3 types of b_avg.py finally accomplishes what I wanted to accomplish back in **10. try plotting breconsog, breconspred, and b - all avg.py**. This plots the three values of the b_avg varying over time for all the maps in the given range.

15. 14 but animated - this isnt good, try again.py is the first attempt at trying to make an animation out of the way the three b_avg values vary over time. Did not turn out to be in the way I wanted it to be. It creates an output GIF file but then the animation was not smooth.

16. 14's animation maker (helper).py is a code that attempts to be helpful in the eventual generation of the animation of the three b_avg values varying over the years. It attempts to do this by creating multiple graph plots, each one having one more data point to be plotted per type of b_avg value and attempts to finally make a compilation by stacking all those .png plots into one GIF animation.

21st February 2025:

17. calcalm.py is quite confusing, hmm. For what had I used this code? Going by the looks of it, this was just to check if the alm values upto a given l value varied depending on the lmax value or not. And to check how alm values vary based on other parameters. I might have created a mini subproject folder to understand the factors that affect the alm values, separately. Will write about those codes aswell. YES, got it. Appending them below.

1. calcalm.py in the **"understanding alms better"** folder is the exact same as **17. calcalm.py**.

1. calcalm-gauss.py in the **"understanding alms better"** folder is the same as **1.calcalm.py** but this was used to run along with the prementioned code to compare how the values of alm change based on the application of gaussian filter on the input B data that is derived from the fits files. This also raises a particularly good question I revisited multiple times throughout this journey – which is this – is it really necessary to apply the gaussian filter on the input fits file data? If yes, why? And what should be the sigma value for that filter? How to judge if applying the filter is needed or not, because I did notice that a larger fraction of the B gets reconstructed if the original B average value is chopped down or smoothed out to a lower value thanks to the gaussian filter. Any other filters? I have not read a single research paper on fits files or on solar magnetic field data processing. I should do that, or I could approach my solar physics professor again.

2. predalmsplit.py predicts the missing/middle alm values (real and imaginary parts separately) from the original set of alm values. Does not make use of the gaussian filter on the original solar data.

2. predalmsplit-gauss.py is the same as **2. predalmsplit.py** but as the name clearly indicates, it computes the middle/missing alm values based off of the alm values that were initially computed by the **1. calcalm-gauss.py** instead of

1.calcalm.py which was used by the **2. predalmsplit.py** code. This marked the end of the short detour into trying to understand if applying the gaussian filter was really needed, on the input data. No absolute conclusion, have to ask a mentor or a professor, or research on research papers. Anyways, I think I do proceed with the project from this stage onwards, without the application of the gaussian filter.

23rd February 2025:

18. vary b_avg.py has the same core functionality as that of **14. 3 types of b_avg.py** except now, its being animated perfectly. There were a couple of attempts earlier (**15. 14 but animated - this isnt good, try again.py** and **16. 14's animation maker (helper).py**) but this one perfects the logic of computing and saving the image for every additional set of data points being added into the plot figure. The loop part in the end where I have to set the tempupperbound, and how every iteration will result in the new set of data points being appended into the previous plot figure was a little confusing to figure out initially. The issue with this method of creating the visuals is that there is a lot of redundant calculation of the same values and a lot of memory being wasted in saving all the plots temporarily. This isnt time or memory efficient.

19. b vary singular.py is realising that having an animation is not that helpful or even needed and that just having one plot with all the three b_avg values varying over all the years, is more useful to study the variation.

11th March 2025:

20. check alm variation.py is a code that was used to check how the individual mode's alm magnitude might/might not vary based on the lmax value. This resulted in the conclusion that the value of the lmax (the extent of calculating alm node values) does not affect the lower mode's amplitudes. Those modes will have those amplitudes/magnitudes irrespective of the degree of division of the sphere in the spherical harmonics. This makes me think that I have not understood the method of working of spherical harmonics perfectly, or that there is no full clarity of the way the alm magnitudes are computed and eventually end up adding with each other such that they result in the original data, which was used to compute those same alm values. How does the magnitude of lower modes (with lower l and m values) not change as I increase the degree of division of the sphere? Does it already know how much those modes will contribute to the overall stacking up of all the alm magnitudes? If that is the case, then is it possible to estimate the uncalculated alm values for a given input/CR map based on some of the lower modes? Is there another relation or a logic that I am missing?

14th March 2025:

22. KAGGLE varying l for recons.py works the same as **21. varying l for recons.py** except that it is tailored to work on Kaggle instead of locally on my system.

15th March 2025:

21. varying l for recons.py is one of the first codes in this series or version of this project where I reach the idea of plotting the fraction of the b_avg that is reconstructed as a function of the value of lmax upto which I allow/permit the reconstruction to take place. I also played around with the gaussian filter for a bit and reached a conclusion to not use the filter anymore as that would hamper the actual data that the code is trying to replicate. So I will not be expecting too high of a fraction of the original b_avg values to be reconstructed, either ways – even if the value of lmax goes all the way till 85. This brings another crucial point. the law of diminishing returns – the higher the value of lmax that I compute the reconstructed B values till, the lower is the growth or the increase in the fractional amounts that they eventually add up to. As the lmax value increases, the longer it takes for it to reconstruct the B values upto that high degree and lower is the increment from the reconstruction of the same data upto the previous lmax value. The returns are diminishing, and the cost of computing is continually increasing. Not a good trade-off I would say, but I still implemented the same idea for research purposes – just to see how much I would be losing out on by reconstructing to a lower lmax value, how the overall fractional variation over different lmax limits itself varies as the input CR map data changes. Also plotted the top five maximum magnitudes of the fractional b_avg values in the map, to highlight how the distribution of the higher fractions of b_avg being reconstructed looked like – and where were these reconstructions arising from - which lmax values. Now an important mini conclusion from this study was that the CR maps where the sun is in closer to having minimal activity i.e., the solar minima, then the overall b_avg from the original map would be lower (close to 1-3 or 4 Gauss) and these maps would almost never be reconstructed upto a fraction higher than 0.5 even with really high values of lmax. However, on the other hand, the CR maps of the sun where the activity is either peaking or reaching or diminishing from the peak (close to the solar maxima), here – the reconstruction fraction of the b_avg turns very high (about more than 7-9 Gauss of overall b_avg), and hits fractions of upto 90-95% of the b_avg. The law of diminishing returns still stands as a constant for all the data either ways. Low solar activity, or high or transitioning between the two: the returns or the increment of the fraction of b_avg being reconstructed increases by more than unity per increment in the lmax value up until a lmax value of 20-30, and then the increment is lesser than unity for the next few increments up till an lmax value of 40-45 and anything beyond that is way too less of an increment of the total fraction. By the way, since it takes really long to compute the b_recons values for lmax values as high as 85 for multiple maps, even with multiple Kaggle accounts – hence I could not extend this study for more than a couple select maps. Just less than a dozen, namely (or numberly) – 2096, 2097, 2098, 2099, 2100, 2226, 2253, 2265. The idea was to consider two maps where the Sun would be in peak solar maxima activity and two where it would be in the lower peak of solar minima, lacking activity and one where the sun is ascending into the solar maxima peak activity and another where the sun is descending from the solar maxima peak

activity. But this project had to come to an unexpected and maybe foreseen end, so abruptly because I felt that I needed help from an advisor/mentee or a professor who knows how to analyse the results and help me decide what to focus on, where to find or look for patterns or what are insights that are meaningful for future study. But yes, there were minor improvements from this particular version of this code, beyond this date's but this was low-key the essence of this project, although this was not something I started out with as an objective or goal. That goal or objective was actually to be able to predict the variation of the solar cycle through multiple visual representations (if possible, else just one simple prediction visualisation) to show how the sun's activity might fluctuate over the years but yes that idea is too far-fetched and I might need to study a lot more research papers in this topic and look at how other people have approached similar if not the same problems and seek guidance for directions and improvements. Another point to note is that why should the lmax value stop at 85? Why can the code not go higher? Is it issues with my hardware/system or the language that I am using (python)? Is there a more efficient way to reconstruct the solar magnetic field data? Is spherical harmonics the only way? I remember my IISc professor telling me that there are other ways of reconstruction yes, but the math behind all those functions was more complex(?) hmm, shall look into those things the next time I revive this project as NagaSurya 3.0. Also, on side note, regarding something I noticed now that I am writing all these summaries based on reading all the codes in one go, one after the other. Well I can have different helper codes that compute the alms and recons values in separate python files, and I can call those functions in the code on which I am working. I can make my own library of functions. This would reduce the visual size of the code that I would be working with at any given moment, any version of code, working on any functionality of the overall project would still require lesser lines of code in the current working python file, as it would be referencing a lot of the customised functions from my libraries. Even for plotting different types of graphs. That would increase modularity, reusablitity and make the whole project appear more formal. Changes will be applied for NagaSurya 3.0 whenever that happens.

24. separate 2096-2100 brecons og and pred.py was created to help separate the b_recons values of a certain set of CR maps (previously mentioned as part of the final study) and only those that are reconstructed from the original alm values computed directly from the fits files data and not from the alm values predicted by using the linear regression – all these were a part of this large pool of "reconsdata" folder, which has grown to a really large size by now.

25. check bunits.py was just to check the units that the fits file stores the data in. I was not sure if the data points in the fits file were in Gauss or not. This was just to cross verify. It returned Maxwell per centimetre square which is the same as Gauss.

17th March 2025:

26. CRmap + vary b_avg.py is roughly the same as **23. varying b_avg recons (2096-2100) & 2226, 2253, 2265.py** except it makes use of the plotty function which was a part of the resultant helper functions from NagaSurya 1.0, but I was not able to integrate the same plotting function properly into the varying fractional b_avg recons plot.

27. overlapplot.py is a simple code that just plots two plots, where I have the freedom to overlap one plot over the other, this is to try and see how the overlapping of plots works and how I could leverage this feature to be able to plot the CR map along with the respective map's data being fractionally reconstructed as a variation of the lmax upto which it is reconstructing. Also made use of this code to learn about the "dpi" parameter, and how detailed the clarity of each of the exported pngs end up being, based on the dots per inch parameter while saving the plot, and to check what is the level of detailing that I will need in this project of mine. Turned out ot be 300-450ish dpi.

18th March 2025:

23. varying b_avg recons (2096-2100) & 2226, 2253, 2265.py is **21. varying l for recons.py** but with an additional feature of plotting the CR map from the fits file, unaltered in the bottom right corner of the main plot that consisted the fraction of b_avg being reconstructed from the various value of lmax upto which I was allowing the reconstruction to happen in each iteration; for all the mentioned CR maps.

28. simpfitsoverlap.py has the same testing of how overlapping plots work, except I am plotting a sine wave and the solar map, in the same frame – again, just to test how to overlap a plot onto another and how to customise and tailor it, to use in **23. varying b_avg recons (2096-2100) & 2226, 2253, 2265.py** eventually.

29. b_recons similarity.py is the last code in this version of the project. Man I am so sad that this is the last program, whose summary I will be typing. I do not want to type the description of this code out. Aaaaaaah, okay I should keep this formal. Okay, here goes: this code was initiated with the intention of comparing the amount of reconstruction of the b_avg value for any two lmax values of my choice. It would compare by computing the cosine similarities of the real, complex and absolute magnitudes of the b_recons values for the given lmax values of the same CR map. This was supposed to give me insights into what is a wise lmax upto which I could reconstruct all of the CR maps, and to see how these error metrics or similarity metrics change as I conduct the study over different CR maps as each CR map captures the sun in a different stage of activity. To summarise this whole study – it has been computationally expensive – memory and time wise – the insights obtained were not too great, and I do not know what to make out of it now. Also trying to focus on core academics and projects that have a definitive output and goal, objective and which I am sure of completing and arriving at a defined output/result. So, until then I guess (: