Helio-physics or solar physics is the branch of astrophysics that deals with the study of the closest star to our rocky planet, the Sun. it began with Galileo’s observations of sunspots on the surface of the sun with his telescope back in the 17th century. The variation of the number of sunspots is a consequence of the dynamic nature of the sun. later in the 19th century, with the development of spectroscopy, the study of what makes the Sun up, boomed. Helium and hydrogen were declared to be the main components of the sun. towards the 20th century, we had space-based observatories and telescopes to get detailed and undisturbed data about the solar magnetic field, and the spots and flares corresponding to the field’s activity. Ground based telescopes and satellites are currently being extensively used to study a lot more about the sun.

The solar dynamo model is the wide-acknowledged explanation to the creation, development, and evolution of the sun’s magnetic field. It makes use of the interactions between the sun’s axial rotation and the convective motions caused by it. the solar magnetic field can be divided into two main components – the toroidal field and the poloidal field. The toroidal field is aligned along the equator. The poloidal field, on the other hand, is aligned along the north-south or “meridional” direction. The toroidal field is maintained by the differential rotation of the sun, which causes the filed lines to stretch and twist. The poloidal field is taken care of, by the convective motion in the sun’s convective zone. Differential rotation is a feature of the sun that is characterized by the equator rotating around the axis faster (taking about 25 days) than the poles (that tale about 35 days), shearing the magnetic field lines that emerge, turning the poloidal field into toroidal fields. This cyclical process helps the sun to maintain and drive the solar cycle. The meridional circulation is the motion of the plasma inside the sun from the equatorial region to the polar region and back, transporting the flux from the equator to the poles. This convective motion contributes to the regeneration of the poloidal field from the toroidal field.

Convective zone is that part of the sun that extends from about 0.3 solar radius to the beginning of the photosphere. Energy is transmitted through convection in this zone, hence the name. hot plasma rises up towards the surface and then cools and sinks back down. The solar dynamo model not only maintains but amplifies the solar magnetic field. The diffusion of the magnetic field is isotropic, and the weaker parts of poloidal field are converted into stronger toroidal fields.

The solar magnetic activity cycle is mirrored by the sunspot cycle, both of which are 11-year cycles marked by the variation in the number of sunspots on the Sun’s surface. The sun reaches maximum activity – the solar maxima, characterised by a vast number of sunspots overall and closer to the equator; and reaches minimum activity – the solar minima, characterised by smaller number of sunspot emergences, that too being far away from the equator (and closer to the tropical latitudes). Cycles have been numbered consecutively since one of the earliest observed cycles in the mid-18th century, putting the current solar cycle to be numbered – 25.

Sunspots are darker (hence cooler) spots on the hot surface of the sun, which are associated with very intense magnetic activity compared to most of the plain and featureless solar surface. They served the discovery of the solar magnetic cycle as they were the only visual effects of the sun’s dynamic magnetism. These sunspots appear in pairs, owing to the fact that magnetic monopoles do not exist unlike electric poles. The pair of sunspots are associated with opposite magnetic polarities and are sits of strong magnetic fields. They are also linked to solar flares and coronal mass ejections (CMES) – that cause solar winds – massive outflows of charged plasmic particles outward from the sun. these affect the magnetospheres of all planets and could damage anything that lies exposed and unprotected from these high speed and high energy particles. They are believed to have in fact, blown the atmosphere of mars away, a few billion years back.

The solar corona is the outermost part of the sun’s atmosphere. CMEs and solar winds propagate outwards from the corona. The hale cycle is a 22 year long solar cycle during the course of which the sun’s polarity flips twice and returns back to its initial state (once every 11 years – which is the period of the Schwabe cycle, which marks the sunspot cycle).

The sun starts with an initial state of activity and initial change in the state of activity. For example, it starts at “solar minima” and then has increasing activity until it reaches the “solar maxima” and then has decreasing amounts of activity, reaching the “solar minima” again, repeating forever. Now, when the sun is at a solar minima, the over magnetic flux of the sun is also minimal, and the majority of the flux emerges from the poles (owing to the poloidal field being the strongest at this point of the cycle). It also features really less sunspots and emergences and if any they are closer to the middle of each of the hemisphere when viewed from the front. As the cycle progresses, the activity of the sun increases. The magnetic field gets amplified, and the polar fields contribute lesser and lesser to the overall magnetic flux. The sunspots start appearing at lower latitudes and reach the equator (but never across it). two pairs of sunspots can emerge close to the equator and one of the polarities of one of the sunspots that’s closer to the equator can cancel the opposite polarity of the other sunspot (if the opposite polarity spot is closer than the same polarity spot, to the equator), then we will have the left over spots “migrating” in some sense to their respective poles and cancelling a small part of the polar field out. Hence, the polar field reaches minimum magnitude at the solar maxima and then as the cycle continues, its polarity flips.

Cosmic rays are high energy particles that originate from outside and stream into our solar system. During the sun’s periods of higher activities, the increased strength of the magnetic field of the sun deflects more of these cosmic rays out of our system and in one sense protects all the planets from it. during solar minima, the solar magnetic field is not as strong and hence could allow larger numbers of cosmic rays to penetrate into the system.

The total flux of the sun refers to the total amount of magnetic field lines passing over its surface whereas the average flux is the flux of the sun per unit area. Both are used in different areas to study various parts of the sun’s magnetic activity. Carrington (CR) maps are visual maps that represent the sun’s magnetic field taken over one Carrington rotation of the sun (which is about 27.3 days long). The sun’s whole spherical surface is observed to have rotated about its axis in a period of about 27.3 days and hence these CR map numbers are used to refer to specific images of the magnetic field of the sun.

During solar minima, the magnetic field is dipolar and during solar maxima the magnetic field is more complex and tangled leading to all the observable features and phenomena of the sun. however, sometimes, it is important to note that although the field gets more complicated, there appears to be some sort of large-scale features that represent the sun’s magnetic activity accurate enough to be able to predict the activity in the sun’s atmosphere. There seems to be a sweet range of sunspots below which there are only large-scale features, and above which are multiple small-scale features that combine to give a fairly large-scale feature.

One of the most fundamental equations here is the induction equation for the magnetic field B.

A diagram of a slide show

Description automatically generated

Where v is the velocity of the plasma. This equation describes the evolution of the magnetic field over time due to plasmic flow and diffusion.

Sunspots are primarily created due to the magnetic buoyancy of the magnetic flux tubes that are created in the interiors of the sun and rise through the plasma around it due to buoyancy. The flux tubes themselves are created due to the twisting of the magnetic field lines and their concentration into tubelike structures. These tubes have higher magnetic pressure than the plasma around them and hence their gas pressure is lower than the surrounding plasma. Even the plasma inside these tubes is of lower densities, which created the buoyant force that overcomes the gravitational pull and helps it rise from the convection zone to the surface of the sun where it has the visual sunspots corresponding to the two ends of the flux tube on the surface of the sun. these sunspots give a visual idea of the active regions present on the surface of the sun. sometimes, these magnetic flux tubes can break and reconnect, creating and ejecting loops of magnetic fields that later accumulate particles from the sun’s corona and propagate outwards, popularly known as coronal mass ejections.

Fits file is a digital file format that is extensively used in astronomical photographs. They are easier to store, transmit and analyse copious amounts of data. They allow for complex multi-dimensional data arrays to be stored in them and a number of metadata aswell. They contain atleast one header and data unit (HDU). I have used radial field 720\*360 size fits data files for this project. Retrieved them from the HMI magnetic field synoptic charts website.

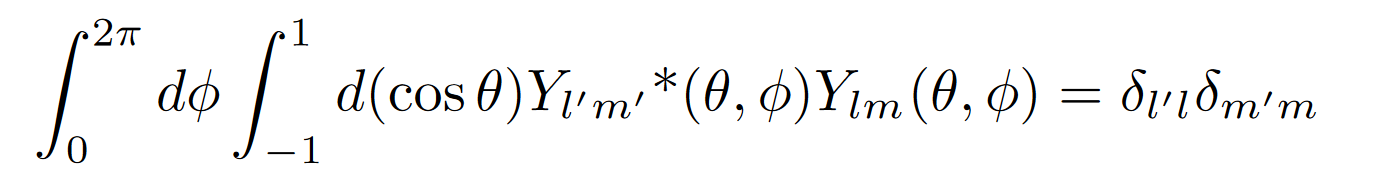
Spherical harmonics are a mathematical tool used to break down functions onto a spherical surface. Primarily used in fields of physics, and geophysics where there exists a spherical symmetry in the system. Ylm (θ, φ)s are spherical harmonics defined on a sphere where θ is commonly referred to as the colatitude (that ranges from zero to π radians) and φ is referred commonly to as the longitude (that ranges from 0 to 2π radians). The variables l and m are called the degree and order of the harmonic/mode, respectively. L is a whole number and m can vary as an integer for each value of l from -l to +l including 0, -l and +l. A combination of l and m represent a particular harmonic/mode. Hence, analysing with the help of spherical harmonics is often also referred to as mode analysis or modal analysis.

The general form of a spherical harmonic is

A math equations with numbers

Description automatically generated with medium confidence

where Plm are the associated Legendre polynomials. The Ylm functions are solutions to the angular part of Laplace’s equation converted to the spherical coordinate system. The spherical harmonics are orthogonal for different l and m, and they are normalized so that their integrated square over the sphere is unity:



A function of θ and φ can be represented using spherical harmonics as summation from l = 0 to infinity (summation from m = -l to +l (alm \* Ylm (θ, φ))). Here, alms are called the coefficients which can be computed from the same equation above and also making use of the orthogonality of the Ylm, as the double integral of f(θ, φ)\*(( Ylm \*)(θ, φ))\*sin(θ)\*dθ\*dφ where θ varies from 0 to π and φ varies from 0 to 2\*π and (Ylm \*) is the complex conjugate of Ylm. Every combination of and m represents a node, and every node has its own distinct way of cutting the sphere up into parts, such that there are m vertical slices and l-m horizontal slices that the sphere has been broken into. Hence, usually for the right amount of resolution, the values of l and m are close by on the number line. Too many vertical cross-sections and lesser number of horizontal cross-sections are more likely than equal number of horizontal and vertical cross sections due to hemispherical anti-symmetry of the sun. since there exists no east-west hemispherical symmetry/anti-symmetry, therefore the number of vertical divisions is high. Orthogonality of the Ylm function means to say that the integral over θ and φ of Ylm conjugate \* Ylm \* sin θ \* dθ \* dφ is equal to a constant. Legendre’s polynomials are the solutions to Legendre’s differential equations. Plm(x) is called the associated Legendre polynomial to Pl(x) which is equal to:

A black and white math equation

Description automatically generated with medium confidence

Python supports a variety of spherical harmonic calculations, methods of integration like the Simpson’s 1/3 rule, adaptive dblquad method of integration, and multiple other sunpy and astropy libraries and packages to help deal with fits type of files and the calculation of the date-time variable from the Carrington map number. sunpy is widely used to manipulate, analyse, retrieve, and visualise the solar data. Astropy has tools to manage large astronomical data, perform coordinate transformations and much more. The sph\_harm function from the SciPy library is one of the multitude of scientific and technical computational tools available for python. It includes tools for integration (as mentioned before), optimisation and some special functions like the spherical harmonics. The sph\_harm function in specific can be used to compute the spherical harmonics of a functions and reconstruct that same function. Here, a function can also be the sampled data from a fits file.

The first step to break down and reconstruct any function using spherical harmonics is to define a θ φ mesh grid. It is to define the number of points to be considered for slicing up the sphere along the θ and φ (colatitude and latitude respectively). Then comes creating a mesh with these num\_points\_θ and num\_points\_φ variables. Usually, num\_points\_θ is half of num\_points\_φ owing to the fact that θ varies from 0 to π and φ varies from 0 to 1π for a spherical surface. So, to get a square shaped grid, we have to take the number of points that we take along the φ direction to be twice as that of the number of points that we take along the θ direction. Then we use scipy.special.sph\_harm to compute the spherical harmonics for each pair of θ and φ. Then we expand the function that we want to break down (in this case, the CR map as a 2d NumPy array) in terms of those spherical harmonics, i.e., to every mode of the whole set of spherical harmonics equations, we assign/compute magnitudes that tell us how significant that mode is in the broken down set of Ylm equations. Then using those magnitudes of the different coefficients of the spherical harmonics, the original function is reconstructed. This is remarkably similar to using Fourier transform to break an analog signal into the multitude of sine and cosine waves of different sets of frequences and amplitudes that build them. The magnitude represents the contribution of each of all those modes.

This was my first time ever working on a project with python. Sure I had heard about the language and knew the general syntax and structures. But everything that I came across regarding writing the code in python in this project was very new and I had to refer to geeksforgeeks, stackoverflow and the libraries’ official python documentations for every feature of my code here.

-

Write about all the versions of your code here:

Started with a program that calculates alm values and stores them in text files and plotting pre downloaded CR maps’ fits files. Went on to define simple functions of b but did not know how to integrate the function into the alm calculation part. However in the later versions of the code I was able to integrate the function B into calculating the alm values for that function B.

2 days later, I wrote a code to reconstruct and plot the function that was broken down, using the calculated alm values and the Ylm conjugates. Another code took a particular fits file, calculated its alm values and stored them in a text file, and plotted the original Carrington map.

The next day, I developed a code that plots the original B function, calculates the alm values for the given lmax and reconstructs the function, B and plots the reconstructed function as well. I was plotting both, the original and the reconstructed plots separately, one after the other. That is when I came across the subplots feature of python that allows you to plot multiple graphs/diagrams in the same plot. So I implemented that in the next version of my code. Had to play around with the positioning and sizes of the subplots for better viewing of the graphs.

4 days later I had these following new codes. The first one was a very neat one that calculates alm values upto the given lmax value, reconstructs the original function using those alm values and then plots both, the original function, and the reconstructed function in two subplots. This was when I was playing around with different lmax values and different num\_points values to see how accurate the reconstruction gets. Note that I was using a single num\_point variable instead of the earlier mentioned “num\_points\_θ” and “num\_points\_φ,” and this was because I was using simple functions of y like sin(y) and cos(y) as B, which change very smoothly over the x and y coordinates. But later somewhere I do change it to two variables, one num\_points variable for θ and another for φ. This does provide better reconstruction features especially when the function is not a smooth or periodic one. the code that I had just developed at this point had only one lmax value and I had to manually change the lmax value in the code every time to see how the results vary. So the next step of improvement was obviously to automate the variation of lmax through a set of values corresponding to another set of num\_points values that varies too.

For the next day, there was no major improvement. I just ran the same code for multiple values of lmax and realised that after lmax crosses 85, the moment l=86, some of the alm magnitudes turn into nan values. And I realised that its mostly because the magnitudes of alm are shooting ridiculously high up or ridiculously low. The modes that had these nan values were modes where the values of l and m are the same. Just off by less than 10. So if l was eighty-seven then the values of m for which alm magnitude was being stored as nan were like 84, 85, 86, and 87. Hence, I started limiting the upper end of lmax to be eighty-five for most of the codes further down the lane.

The next day I introduced the third subplot of delta/difference that plots the pixel-by-pixel difference in magnitude of the reconstructed function with respect to the original function. This caused the upper and lower limits of the colourbar to be dependent on the values being plotted in the diagram. I had initially just set the limits to 0 and 1 respectively because I was dealing with a sine function and the reconstructed function had some parts that shot up higher than 1, hence the colour grading of the whole reconstructed plot seemed to be way off the original, but it was the case with only really small and really negligible areas, so I set the limits to be 0 and 1 so that the same colour scheme is maintained between the original plot and the reconstructed plot. But however, as soon as I introduced the delta subplot, I could not set the limits of that subplot to be 0 and 1 aswell because they varied with values smaller than 1. Hence, I used two different plotting functions, one with the limits set to 0 and 1 and the other where the limits are set based on the values in the 2d array that it is to plot. This way, I could have the right colour scheme for the original and the reconstructed maps and still have an accurate way to represent the delta plot.

The next day I realised that the codes were actually taking too long to run (which was actually because I was using battery saver mode which limited the performance of my laptop to about half of its actual capability, which increased the run time to be twice as much as it should have taken). So to get an idea of how long they actually take to run, I explored the datetime libraries and found out ways to add a stopwatch to the code so that after the code stops running and displays the subplots, I can see how long it took to run that (I printed the run time and the sum(abs(delta)) on the empty 4th subplot in the diagram). In the same code, I varied the plotty function to improve the font, the spacing and the area taken for each subplot. Another code that I developed on this day, had the added functionality of reading alm values that have been previously calculated and stored as csv files. This was because when I used to re-run the code with the same values of num\_points and lmax , my code was recomputing all the alm values. Hence, although I did have the storing alm values feature, if I added the reading from the stored alm values csv file feature then it would be more helpful and reduce the runtime to 10% of the time that it was taking in case it was re-computing all the same alm values again. Now the way I was computing the error was fine but then as an error metric I was calculating the sum of the absolute values of all the pixels/elements in the 2d delta array. This is not a wise way to find out how bad the reconstruction is, in comparison to the original function but it does work okay. Built another code this day that saves all these sum(abs(delta)) in text files and then plots the way these errors change as I change the num\_points variable, keeping lmax constant. Now the result of this was obvious – as the num\_points variable increases, the sum(abs(delta)) increases in more than a linear way, but far from quadratic. This plot also had a stopwatch – whats funny about that is that it plots the runtimes variation graph, and it also has a runtime of it being displayed at the bottom of the plot. Later, I developed a more concise and efficient model of the previous code. Now the problem was that I needed more error metrics to plot a graph and see how the errors vary for each num\_points for a fixed lmax , so I had another code that precomputes and stores alm values for different parameters and stores them in csv files so that errors could be calculated for all these instantly and their graph as the num\_points varies can be plotted without delay. But this code took the user’s input for lmax , num\_points\_θ and num\_points\_φ (yes, I switched from num\_points to num\_points\_θ and num\_points\_φ somewhere between a few codes ago), which seemed like too much work. So then I automated the process and made the alm calculation to loop through a set of values of num\_points\_θ and num\_points\_φ for the same lmax .

The next day, I came across SciPy’s dblquad integrate function. Its called adaptive calculus where it decides how to divide the original function on its own, i.e., the num\_points\_θ and num\_points\_φ does not matter to the code anymore. When its calculating the alm values if I call this DBL quadrature function then it divides the original function based on how smoothly the given function changes. Wonderful, right? Yes, but we shall later see how it takes too much time to run plus sometimes when the function is not as simple as a sine or cosine curve, the approximate grid that it assumes/creates is not good enough for our calculation. Later I start using a better way of storing and re-reading the calculated alm values. This takes shorter periods of time and I also start updating the alm value.csv file with every few alm values calculated so that if the program stops abruptly, then it can continue from that alm value that it had last computed, the next time I run it. there was another issue with my code. I never used to get to know when my program got over i.e., completed its execution. I used to let it run for a while, keep watching, get involved with some other work and when I am back in front of my laptop, I would see that it finished its execution quite a while back. So I introduced the usage of winsound beeps of particular frequences and durations to notify the user (myself) that the program has started, reached distinct parts of the code, and has finished the execution of the code – all these parts/stages. Now, just for better experience of the user (me), I also introduced the feature of displaying all the various parameters in the command terminal before it asks for confirmation from the user (me) to start running the code for those parametric values.

The next day I implemented the same code but now with way better UI, it looks better, more formal and displays more useful data. In another code, I was experimenting with how to change the name of a string variable called func\_name and how to result that change into the changing of the actual B function that it uses for the whole alm calculation and reconstruction process. Stumbled upon the usage of lambda functions that could have helped me with changing the NumPy expression that the B function returns but that is hard coding some values into the code, so I did leave lambda functions right there. Then, I introduced the feature of progress bars into the code. So now I could see what percentage of the total alm values calculation is done, and what percentage of the reconstruction of the original function is done. Then in the next code, I just improvised on the progress bar. TQDM is pretty nice, it is pretty and its nice.

The next day I added an error folder to store different total absolute errors into it as text files and an alm values folder that had all the calculated alm values csv files in it. looks more organised now. Now, wrote a program that computes the total absolute error for a range of num\_points values and the same lmax . Tried to make the code work for functions of x instead of y. after that worked, I had to make it work for other functions of y and x not just sine and cosine. Basically, tried to make the code work no matter what expression I return from the B function.

The next day was filled with same attempts at trying to make the code work for a multitude or atleast a set of known functions of several types.

Now the next day I introduce the feature where the code reads from a b.txt file where the LHS is the string name of the expression that it returns and the RHS is the NumPy expression. So the code takes in the string that I πck and then reads the text file and picks the NumPy expression and uses it as what the B function returns in the code. So various versions throughout this week were consisting of the following – one that runs only for a particular function, one that runs for all functions in b.txt and then compares the max runtimes (runtime when alm values are not precomputed) and the percentage errors (which is calculated by the max magnitude delta / range of original function’s output) for all of them. Another code applies gaussian filter to the input data from the fits files / 2D B array. This is to smoothen out any irregular spikes in the input which will make the reconstruction difficult for the code. Then one of the codes simply plots fits data instead of function data. Now I wanted to change the CR map number and wanted that change to reflect in the code such that it takes that particular CR map number’s fits file from a fits\_files folder and plots and computes for that fits file. Also, by the way, at this point I had switched back to normal integration from the DBL quadrature integration because the grid that it was assuming to divide the original map was not good enough. Later versions of code had better error metrics like max, min, average(absolute error), percentage error etc. also started displaying IST start and IST end somewhere before this code. Switched to Simpson’s 1/3 rule for integration as it was a bit more accurate than the normal integration. Then there was another code that has the normal integration method instead of Simpson’s 1/3 rule just to see the difference. Another code applies gaussian filter to the input fits file data instead of applying it to the B function data. I also tried in another code, to use reduction in resolution for the input instead of making use of gaussian filter but the gaussian smoothening turned out to be a better way to remove the unnecessary peaks in the data. There is another code that simply tries Simpson’s 1/3 rule of integration out.

A week later, here are the updated versions of the codes. One of the codes computes the Bavg, hemispherical\_avg and polar\_avg flux and stores them all in csv files, one per CR map for future reuse. Another code computes the total magnetic flux for every CR Map and plots the variation of it over the years. I had to make use of all the fits files available on the internet with the 720\*360 dimensions, hence I had to download all of them. But instead of doing that manually, I came across the selenium package and made use of it and the default gecko code with some minor variations to change the website that it visits and the element tags that it clicks on after setting some textbox elements to particular values. The gecko default code is a basic web scraping or web crawling code that is mostly used to extract copious amounts of data where one webpage is interlinked with multiple other web pages through some key element tags. So I made use of the gecko code to download all the 720\*360 fits files from the hmi.standford website and store them all in a single folder. Then I made use of these fits files, wrote another code that takes all these fits files as input, and one by one it computes alm values for all of them, reconstructs and plots them all, saves them in designated folders (yes I did not know that plots could be automatically saved until a few days before I wrote this code and all plots saved until then were done manually), and computes the new error metrics and saves them in a separate folder. It makes use of a mapπng function that maps every Carrington map number to the month-year that it corresponds to. It moves all the processed fits files into a subfolder called done so that even if I stop the execution of the program all of a sudden ,the next time I start running it, it does not re-run for the already processed fits files. This program has the normal progress bars and simple beeps to signal to the user (me) that it has reached various parts of the code. Then I develop this better version of this code and save it as another code where I change the normal TQDM progress bar to a rich progress bar so it simply looks way better and removed unnecessary beeps in the middle of the code and now its only one beep per start and one beep per completion of the processing of a Carrington map’s fits file. The issue with the previous code was that it made use of a mapπng function to find the month-year from the Carrington map number, but now in this better version of the code I make use of sunpy.coordinates.sun - Carrington\_rotation\_time module/function to extract the month-year (datetime) from the Carrington map number. I ran this code with everything except the alm calculation part commented out so that I could first have all the alm values csv files stored in a folder and then I could run the same program repeatedly for the reconstruction parts. I also discovered that if I kept the lmax value to be something as low as ten or even 5 only during the reconstruction of the original map then it plots the large-scale features that are supposed to be on the surface of the sun. so playing all those plots (one per CR map again), we could get an idea of the motion of the large-scale features of the sun’s magnetic field. Then I realised that when I am processing multiple CR maps’ fits files, the code runs fast enough for the first few of them but as the fits files progress, it seemed like as if the runtime were increasing by a lot and was doubling in most cases. So I thought that this was the problem with the program taking too much memory as they keep going on processing increased fits files. So I broke down the best version of the code that I had into a main\_script and a supervisor code such that the supervisor code runs the main code and as soon as the main code is done processing one fits file (1 CR map), the supervisor code aborts the execution of the main code and then waits for a short period of time (to clear up memory) and then calls the same main code again. Although this seemed to be a useful additional feature, I later realised (as previously mentioned) that this wasn’t he effect of too much memory being taken up as a result of multiple CR maps being processed one after the other inside the same code – instead, it was the effect of me using the battery saver mode which limited the performance of my laptop by a lot. Codes started running twice as fast, literally, after I switched the power saver mode off and made my laptop focus on peak performance rather than peak battery life. Also had to switch from Viridis map to linear colour segmented map for accurate representation of the fits data. Another problem was that if there the negative flux peak’s magnitude was greater than the positive flux peak’s magnitude then the white was no longer representing the actual 0 flux line. Had to fix it by setting limits to the colourbar.

About a week later these were the updates on my codes. Wrote codes to help analyse the alm values that it was storing. One code plots the average value of the alm magnitudes in a given percentile of the total alm magnitudes, against the years. There is an option to enable gaussian smoothing on the datapoints being plotted. The next attempt on this day was to try to plot the alms against l-m axes with a Viridis colourbar to represent their magnitudes, highlighting the max, the second max and the average magnitudes. One plot for each CR map. Initially I was plotting max and min magnitude alm but the min alm is of no significance, hence I switched to the second max alm magnitude. Also, the alm magnitude for the same l, +m and -m was the same hence I decided to plot only for values of m > = 0.

The next day, I decided to combine all the features that I had developed until then. So in this code, I was plotting the original CR map, the map saturated to +/- 200 and then the same map saturated to +/- 700 along with the alm magnitude distribution broken down into 4 subplots. The next code that I developed on this day involved plotting atomic orbital like visual representations of the maximum magnitude alm’s respective Ylm spherical harmonic. This was in subplot number 1 and subplot number 2 had the alm magnitude distribution plot. Now the next code was to obviously combine these two prementioned codes of that day. So the next code had the following being sub plotted – one, the original map. Two, the alm magnitude distribution. Three, the original map but saturated to +/- 200 and then four, was the atomic orbital like visual representation of the maximum magnitude spherical harmonic from the alm magnitude distribution plot. Wrote another code that does the same as the previous code but has better visualisations and looks cleaner and neater. I was plotting the atomic orbital like visual representation because after many trials, I was still not able to plot the Ylm function as how it would have looked on the spherical surface. But I do discover that later on in another code and then make use of that. Anyways, the next code dealt with plotting the alm magnitude distribution, the reconstructed map with the lmax = l of the max magnitude alm only, the atomic orbital like visualisation of that particular spherical harmonic with the max and second max magnitude of alms in that distribution highlighted, and then plots the original map saturated to +/- 200 in the fourth subplot. Developed another code with the same functionality as the previous only but it just looks better overall. Then wrote another code that does what has been the best plotting as of yet. It plots the original map saturated to +/- 200 and reconstructs with lmax = max of (l of the max magnitude alm , l of the second max magnitude alm ), and then plots the alm magnitude distribution along with two atomic orbital like visual representations of both – the max and the second max magnitude’s spherical harmonics, side by side together in the 4th subplot. Theres one simple code that I used to plot just the atomic orbital like visual representation of a particular spherical harmonic. Theres another code that plots the l (corresponding to the max magnitude alm ) as a function of maps/month-years. There is another code that reconstructs the original map with lmax = 1 and plots that along with the original map and the delta diagram, so that it was supposed to help us understand how the dipole magnetic field varies over time. So there is one plot per CR map for that program. That is the case with the 4 subplots (2 atomic orbitals, 1 atomic orbital) code aswell. Anyways, I do not think that represented the dipole magnetic field properly so then I wrote the next program. This next program is plotting only the alm magnitude plots (one per CR map) except here it plots for l = 1 only so there are three modes for l =1 and its plotting those three modes (1,-1), (1,0), (1,1) as three dots per CR map and saving them to get an idea as to how the dipole structure of the sun varies over time. Then the last code for that same day deals with plotting the sum of all the alm magnitudes for that particular value of l. one for each CR map. Dividing the sum by (2l+1) as an attempt to compute the average alm value for each l, for each CR map.

The next day, I start off with a code that plots the  variation of the magnitudes of modes (1,0), (2,0), (1,0), (2,0) and (5,0) over the years with the intention of seeing how those modes in particular vary as the total magnetic flux varies in the background for each one of them. Did note that modes 3 and 5 had features of mimicking the actual magnetic flux data. Modes 2 and 4 do not really fit in. got to change the bars on the side into fractional change to get the right perspective. New area for new code. Should make more sense like that. Plot it with the data point’s value divided by the upper limit of that plot. So all values will vary from 0 to 1. Anyways, the next code that I wrote was to check something about the dipole nature of the magnetic field. Hence it was the same as what I had written that day where I explained the supervisor-mainscript code earlier. Except for this one I had fixed lmax to be at 1 instead of the default of 85. Another code was the same as the one of the previous ones. I used it here to check something about it. no real progress with that though. Then another code I wrote today plots the original map saturated to +/- 200 and the reconstructed map with lmax = l that belongs to either the max magnitude alm or the second max magnitude alm , the other two plots were – the alm magnitude distribution and the other one had both the max alm magnitude’s and the second max alm magnitude’s corresponding Ylm function plotted as both – the atomic orbital representation as well as projected onto a spherical surface. 4 diagrams in a subplot. The next code was this same previous code except the visualisations were improved upon. And then there is one code that plots just the spherically projected visual representations, unlike the atomic orbital like visual representation of the Ylm given to it in the code.

The next day I improvised on the visually improved version of the best of whats there and added the feature of what fraction of the Bavg of the original map is the reconstruction upto new lmax reproducing. Then follow the side quests – centre of mass a.k.a COM. the next code was my first attempt at the com. Plots the variation of Gaussian smoothed (lmean, mmean) - one datapoint per map - over the years. Prints the sum of the alm along with the mean (l,m). the next program plots the COM in the alm distribution plot - one per CR map and saves all these plots. Does something regarding the closest index. Doesn't actually get the right COM alm magnitude, does it? now introducing cheatCOM. The next idea was that this would be the same as COM but now considering datapoints above a given threshold instead of all the datapoints. But is this code actually about that? Yes, it does plot those points in the alm magnitude distribution map - one for each CR map. User can modify threshold. The next code was to see the evolution of the value that the above code gives, over time. Yes, it does that. There is an option for Gaussian smoothing of the datapoints before plotting them. Another code that plots the variation of the value of l corresponding to the max magnitude of alm for that CR map, over all the maps/years. After that, there is a separate program, or rather an attempt at bettering the summation\_alm\_vs\_l.py, by taking into consideration only the last 5 alm values for every value of l - produces one plot for each CR map. Makes sense, because as l values go higher and higher, only the last few modes would have higher magnitudes so should average those values out instead of averaging over the full set of modes for that l.

Three days later I started with plotting the l corresponding to the cheat COM (above threshold means cheat) along with the average flux over the years. Can apply Gaussian smoothing or can plot without it too. then there is a code that plots the l corresponding to the COM along with the average flux over the years. Optional Gaussian smoothing available. There is another that plots the magnitude of the cheat COM along with the average flux over the years - later realized that I am plotting the average of all the alms for that CR map bahahahaha. Next code plots the magnitude of the COM along with the average flux over the years - later realized that I am plotting the average of all the alms for that CR map bahahahaha. There is one program which is the same as com\_time\_b\_avg.py except here I am making use of total magnetic flux instead of average magnetic flux. The next code is the same as magcom\_time\_b\_avg.py except here I am making use of total magnetic flux instead of average magnetic flux. The next piece of code is the same as above (magcom\_time\_b\_tot.py) as the name suggests, except now it plots in a different directory the fractional change of both the parameters over time, to be able to compare what fraction of change in Btot caused what fraction of change in the COM's magnitude. Last point to note is that the magnitude of the COM is the average of all the alm values for that l. Please do refer to the readme file if youre getting confused as to which program has what feature. Anyways, having said that I think that I should have used Btot everywhere wherever I used Bavg in all of the previous codes. Kinda got to check and see which a better metric is to compare the other analytical plots against.

The next day’s codes include one where I got a new idea to convert all those multiple plots' pictures into one video, displaying every plot of that code for 0.05 seconds. Just to get way better overall visuals. There’s one code that was made in an attempt to get the green and orange colours into the original CR Map plots, I downloaded this code from here - <https://docs.sunpy.org/en/stable/generated/gallery/plotting/hmi_synoptic_maps.html>. Then there is one last code which is the modification of the above just to provide a series of multiple plots, one per CR map. It is the same as the above code except here it runs for all CR Maps numbered from 2096 till 2285. However, there seems to be an issue with maps numbered 2141 (including) to 2151 (including), a total of 11 maps are not being plotted.