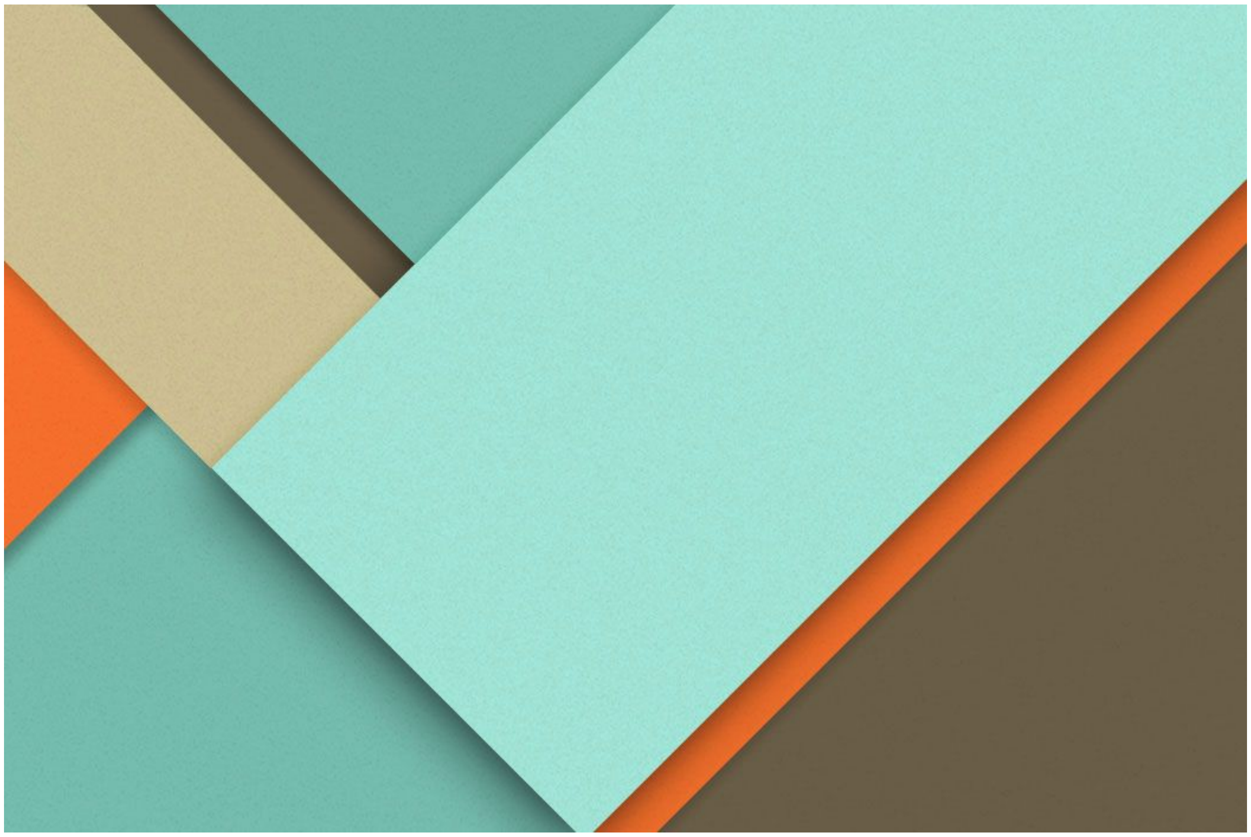


Sunkit Image

Organisation : OpenAstronomy

SubOrganisation : SunPy

Google Summer of Code, 2017

About me

Contact Information

- Name: Nitin Choudhary
- Time Zone: IST (UTC+5:30)
- Chat handle: nitinkgp23
- Github id: nitinkgp23
- Email: nitin.iitkgp23@gmail.com
- Blog: medium.com/@nitinkgp23
- RSS Feed: Feed
- Website: nitinchoudhary.in

Personal Background

Hello, I am Nitin Choudhary, a second year undergraduate student at IIT Kharagpur, India. I'm pursuing a degree in Mathematics and Computing. I work on Ubuntu 16.04 LTS with vim as my primary text editor. I love vim for its power and flexibility. I'm proficient in C, C++ and Python. I like Python because it easily lets me convert my ideas into code.

Education

- University: Indian Institute of Technology, Kharagpur
- Major: Integrated MS Course in Mathematics and Computing
- Current Year: 2nd year (4th semester ongoing)
- Expected Graduation cum Post-graduation: 2020

Brief Bio

I am hooked to open-source software development, an ambitious outgrowth of my personal interests to create free (as in freedom!) and open-source content.

- I like to write and read.
- I love to code.
- I enjoy sharing my experiences.

I am the Executive Head at Kharagpur Open Source Society, which is a student club to foster and develop open source culture in and around the university. I have been involved in several open-source projects, in a variety of fields, including web and app development. My personal projects can be found on my website.

Links to Pull Requests

Here are some of my contributions to SunPy :

- `(Merged)`Fixed minor typos in documentation : While going through the codebase, I found a lot of documentation bugs and typo errors. This PR fixed them.

- `(Open)`Instantiate Map with an array and a WCS object : A feature-request that allows instantiation of a map object with an image data array and a wcs object. This PR is not merged yet.

- `(Merged)`Data directory should not be created on import : This was a bug present in the code, due to which the download directory was created on importing sunpy. This PR fixes the bug, to ensure that the directory is created just prior to downloading of data.

- `(Open)`Add source tests in map_factory test: This PR increases the test coverage by adding source tests for HMI, SWAP, XRT and SXT map.

- `(WIP)`Differential Rotation to use sunpy.coordinates : One of the parts of the project Sunkit-image, this PR attempts to convert sunpy.physics module to use sunpy.coordinates instead of the deprecated sunpy.wcs module.

- `(WIP)`Added module that fetches data using drms : This was the very initial PR that attempted to modify JSOC client to use drms. This later turned to be the 3 month project idea for GSoC,on which I am currently working.

  I will be generating a few more PRs, after submitting my proposal.

The Project

Abstract

This project deals with creating an entirely new package named Sunkit-image, which will contain various image-processing algorithms to be used for solar analysis. With the implementation of 3 algorithms, MGN, NAFE, and OCCULT-2, a set of full-fledged image processing tools will be developed to help scientists analyze solar image data. The differential rotation, a poorly understood concept in solar physics, will get a boost upon implementation of image warping, to easily help scientists visualize solar rotation. It will be made to use the coordinates module, so that differential rotation can be easily calculated in all possible coordinate frames. All further image-processing algorithms, related to SunPy will reside in this package.

The Problem and Motivation

1. **MGN** and **NAFE**

   A lot of other packages and libraries, already have certain image processing routines, but this package will have functions exclusive to solar-image data. One of the important routine is Multi-Gaussian Normalization that is to be implemented. This project implements MGN , porting it from the work already done by Nabobilis and Cadair in #1899 and #964.

   The current MGN code from #1899 has a few issues:

   - The code breaks when a cut-out image of the sun is given as input.
   - The code breaks when a single value of sigma is given as input instead of a list.

- NANs are not handled properly
- Multiprocessing is not supported
- No good documentation exists for the code, nor a proper use case.
- There is no meaningful test case to cover the code.

NAFE has already been implemented here by Eric. This code needs to be rigorously tested and ported to sunkit-image.

2. **Solar Differential Rotation to use sunpy.coordinates**

Differential Rotation is an important, yet slightly less understood concept in solar physics. A lot of its research is based on observational data, hence it is extremely important which coordinate system we are using to express the data. The Howard, Allen and Snodgrass rotation types calculate values based on heliographic coordinates.The most common coordinate system used in solar analysis is Helioprojective coordinates, hence, we have a direct function `rot_hpc()` that takes heliprojective `latitude` and `longitude` as parameters and returns the new `latitude` and `longitude`. The problems with the current code are:

- Astropy quantities are taken as input and given as output.
- Only helioprojective frame is supported. There is no easy conversion between different sunpy frames.

The modification of the above module to use `sunpy.coordinates` is highly recommended because it will make it easier to reproject coordinates from different points of view. Storing the coordinate values in a SkyCoord object will make it very easy to interchange between different coordinate systems without effort.

3. **Implement image warping in Solar differential rotation**

Some work has already been done by wafels in his branch regarding implementation of image warping. This part of the project will take the work ahead. A few points that are specifically to be added are:

- Add unit test for the function `_warp_sun()`

- Add the rotation time to the new map This may require addition of a new keyword in Map or `peek()` method has to be updated to show the rotation time.
- Work out how to update the coordinates mostly in the case of diff-rotating a not full-disk image
- Handling of NaNs and infs in the map
- Integrate with sunpy.coordinates and sunpy.map to access observer location.

This part of the project requires further research and careful look into previous work done by gbear, dpshelio and wafels.

4. **Implement Occult-2 algorithm to perform coronal loop tracing**

Coronal loops are bright, curving structures that appear as arcs above the sun's surface. Study of the coronal loops is important, because it is one of the keys to solve and understand the sun's atmosphere in a greater detail. The Oriented Coronal Curved Loop Tracing (OCCULT) code is a pattern recognition code, that extracts magnetised loops from the images of solar corona, with the aim of optimum completeness. We propose to implement the OCCULT-2 algorithm (a modified version of OCCULT), that will extract the coronal loops from the solar images and trace them on a Map object.

Oriental Coronal Curved Loop Tracing (OCCULT) code was developed initially, to automatically trace curved loop structures from the obtained solar images. This algorithm had a number of control parameters, which was an issue because this allowed less of automation. A modified version of this code was then developed (OCCULT-2), which has only 2 control parameters - the low pass filter and the minimum curvature radius (r-min). Another control parameter , q-med, can handle the noise treatment in the image. Thus, this new version of the algorithm offers a simpler choice of control parameters and provides more automation in tracing coronal loop structures.

For the implementation of OCCULT algorithm, we have to assume that the width of the structure of interest is much smaller than its length, which is pretty much the case with coronal loops. This allows them to be expressed in a 1D path.

With the input image being a `numpy.ndarray` , consisting of N_x pixels and N_y pixels respectively in both the dimensions, we aim to get a list of array of tuples as output. Let $[X(s_k),Y(s_k)]$ be an array of coordinates (tuple, here), where $k = 0,1,2,...,N_{s-1}$. The loop length, or the number of pixels present in the coronal loop, will be given by N_s. We get a number of such array of tuples, each depicting a unique coronal loop.

The goal of the algorithm is to depict as many loops as possible, keeping in mind not to pick up false signals, which would otherwise be a mere noise in the image. Therefore, the challenges in the implementation of the algorithm are:

- An optimum threshold level should be evaluated so as to separate noise levels from existing loops.
- Retrieve coronal loops for the maximum length possible, and avoid dividing it into smaller segments.

A two-dimensional search algorithm will have a computation time that will grow with the square of the image-size, hence our strategy will be to extract the loops in a one-dimensional search algorithm. Apart from the one-dimensional parameter S_k , we use two other parameters A_l (Local direction angle) and R_m (Curvature radius). After selecting the brightest point from the image (X_0,Y_0) and using these 3 parameters, we continue tracing the loop following the principle of Orientation-guided tracing.

5. **Implement running and base difference functionality and the persistence transform**

Coronal loops oscillation event was observed for the first time on October 16, 2010. The oscillating loop is discernible as a faint semi- circular structure in the logarithmically scaled intensity image. Since the background is time variable, it poses a serious challenge for the exact measurements of oscillation parameters. To enhance the best contrast, a number of time difference schemes can be applied. This part of the project aims at implementing four different running schemes :

- Baseline difference
- One-sided running difference

- Symmetric running difference
- Minimum running difference

All these enhancement methods have their own pros and cons, hence it is important to implement all of them. This part requires a detailed study of the paper, and related online journals, and a thorough discussion with the mentors.

6. **Wrapping FLCT code**

FLCT (Fourier Local Correlation Tracking) is a technique to find a 2-D velocity field, from which an initial image is evolved into a second image after a time delta t. This helps in giving a proper mathematical expression to the change or the shift in different pixels all over the image. In other words, it is useful for tracking certain features on a map, over a time series. This technique described here, has already been implemented in SSL, in both IDL and C. This proposal will implement a python wrapper around the already built C code, so that the already implemented C code can be used from SunPy, which is built in python.

The C code for FLCT is over 2500 lines, and is much faster than its IDL counterpart, and will be performing much faster than its Python version (if the code is plainly translated). A better solution to this problem is developing a Python wrapper around the already built C code, using `Cython`. This will offer several advantages, other than maintaining the running speed of the algorithm. `Cython` can act as a perfect bridge between C and Python, and allow modifying the C code accordingly, to extract the best of both the languages.

Implementation

The two main parts of the project Differential Rotation and Occult-2 algorithm's implementation has been described below. Other parts of the project like MGN have already been implemented to a great extent, hence I am not describing them here.

1. **Differential Rotation**

A new function will be defined with the name `solar-rotate()` that will take input as a `sunpy coordinate` and give back the new `sunpy coordinate` after an interval of time.

```
def solar_rotate():

Parameters
----------
coord: `astropy.coordinates.SkyCoord`
        Sunpy-coordinate of a point in any frame (can be an array)

tstart: `sunpy.time.time`
          date/time to which `coord` referred to

tend: `sunpy.time.time`
        date/time to which `coord` will be rotated to

rot_type: {'howard' | 'snodgrass' | 'allen'}

frame_time: {'sidereal' | 'synodic'}

Returns
-------
new_coord: `astropy.coordinates.SkyCoord`
            New Sunpy-coordinate of the point in the same frame as the
input (can be an array)
```

Since the main calculation is performed in heliographic coordinates, the above function will make use `sunpy.coordinates.transformations` module to convert in between the input frame and heliographic frame. SkyCoord has the function `SkyCoord.transform_to()` that can convert the given input frame to any other frame taking the shortest path.

2. **Coronal Loops**

   The main function that will take the input image and give the output `detect_loops()`, which will have an implementation as follows:

```
def detect_loops():

Parameters
----------
image: `numpy.ndarray` or `sunpy.map.Map`
       The data array of the input image, or a Map object.

lpfilter: `int`
          The low pass filter with a boxcar smoothing constant to
smooth out the data noise

hpfilter: `int` (Optional)
          The high pass filter with a boxcar smoothing constant to
enhance the fine structures
          If not given, hpfilter = lpfilter+2

rmin: `astrophys.pix` / units of `pix`
      The minimum curvature radius for loop detection

qmed: `float`
      The control factor that suppresses the data noise in the
background. Ranges from 0 to 1.
      qmed=0 makes no change to the image, while qmed=1 makes image
flat in the fainter half.

Returns
-------
A list of `CoronalLoop` objects
```

This function will call a number of internal functions such as:

`suppress_bg` suppresses the unwanted structures present in the image. `qmed` can be adjusted depending on the estimated fraction of the image that is covered with structures of interest.

```
def suppress_bg():

Parameters
----------
image : `numpy.ndarray`
qmed :  `float`

Returns
-------
image : `numpy.ndarray`
```

`bp_filter` applies a band-pass filter on the given image. This helps in filtering broad structures and smooths out fine structures.

```
def bp_filter():

Parameters
----------
image : `numpy.ndarray`
lpfilter :  `int`
hpfilter : `int` (Optional)

Returns
-------
image : `numpy.ndarray`
```

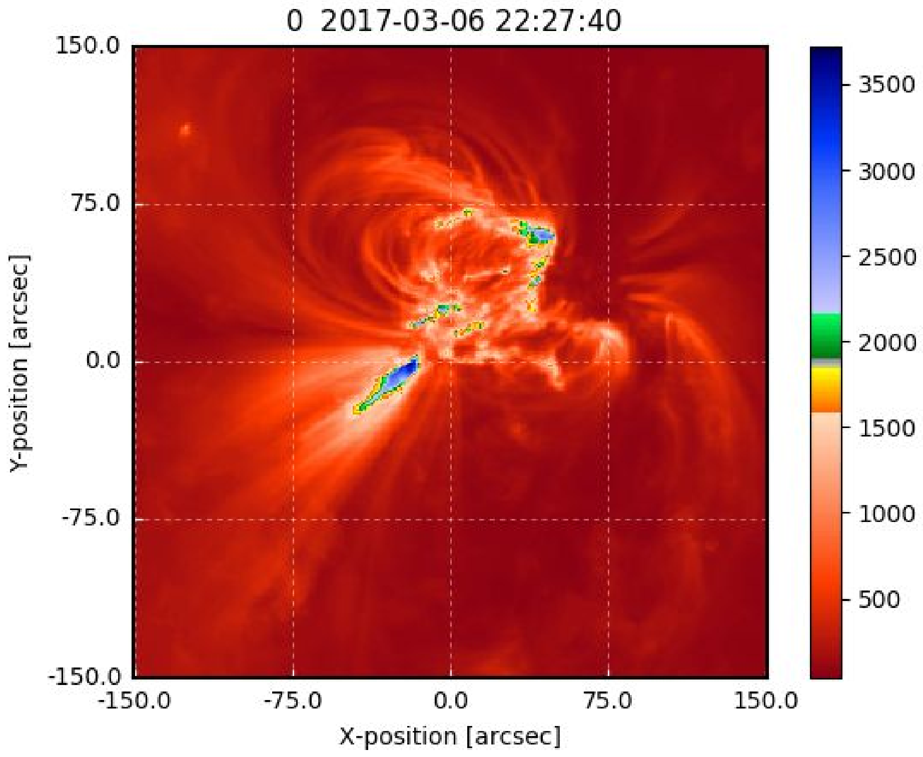
`noise_threshold` finds the threshold for the noise in the given image. This threshold will be used as a stop criterion for the loop tracing.

```
def noise_threshold():

Parameters
----------
image : `numpy.ndarray`

Returns
-------
threshold : `float`
```

We propose to create a new object `CoronalLoop`, that will hold information about a coronal loop obtained from the solar image. Expected attributes of the object are:

- `CoronalLoop.data` : `numpy.ndarray`
  This is an array of `sunpy coordinates`.
- `CoronalLoop.length` : `int`
  This gives the number of pixels present in the loop.

This is a zoomed-in image of Sun, depicting the presence of coronal loops. This is a sample input to the function.

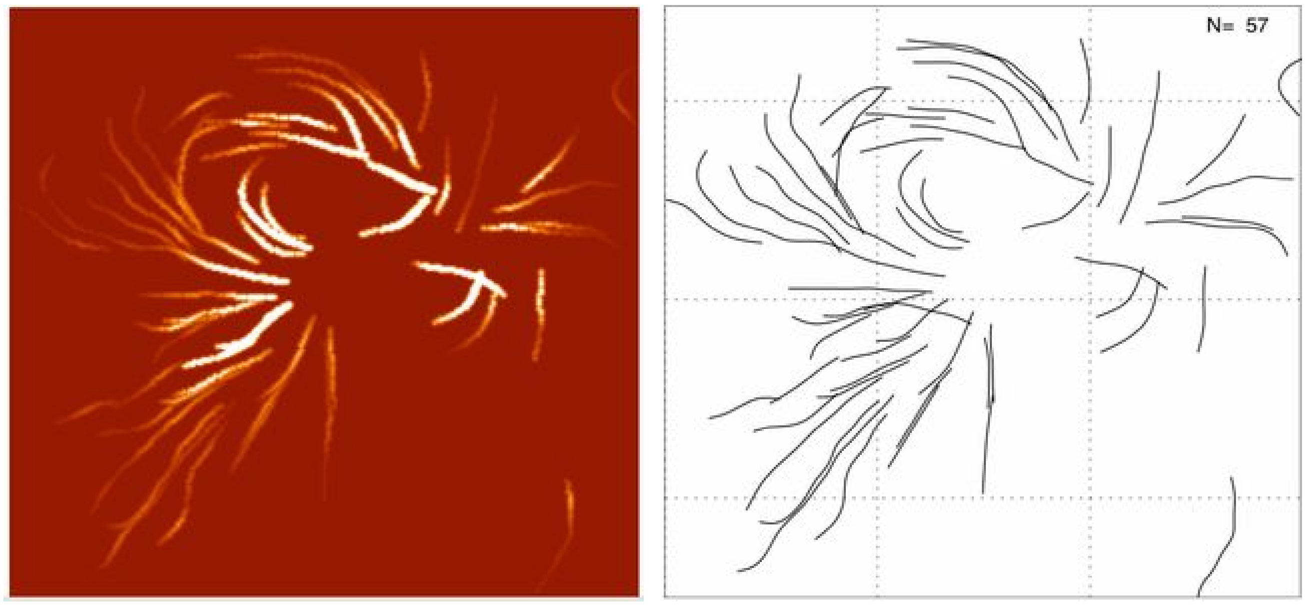


Upon applying `detect_loops()` on the above input image, it will return a list of `CoronalLoop` objects, which will hold information about the detected loops.

and will have the following methods:

- `plot()` : Plots the coronal loops on the map object
- `plot_distribution()` : Plots a graph showing the distribution of loops for each length

`plot()` method will generate a result like this:



A few modifications to the already implemented OCCULT-2 algorithm, will come in handy in a larger parameter space and a variety of images. Though, the algorithm is self sufficient and can work fairly using only 3 control parameters, a few modifications in other parameters can help in better detection of loops.

Timeline and Deliverables

Community Bonding period (May 4 - May 29)

My exams will already have been over by the time community bonding period starts, hence I can start coding right away. Though, I will have a lot of things to look into before I really start to code. Hence, this period will comprise of both getting to know better about the codebase and about successfully implementing MGN and NAFE. I specifically focus on achieving the first task in this period itself, so that I can get a head-start and meet all my other tasks well before deadline. My aims during this period are:

- Have the package repository setup, get CI and documentation running
- Get more familiar with the codebase, especially sunpy.coordinates module
- Start learning about writing tests, having a good hands-on experience before coding period starts
- Have successfully implemented MGN and NAFE

Though, it may look a little ambitious, I am confident of achieving this, since MGN and NAFE have already been implemented upto a great extent, and only a few tasks are remaining to be covered, which consists of a good documentation and few test cases to cover the code well.

Coding period begins

Week 1 (May 30 - Jun 5)

This week will be given to convert sunpy.physics module to use sunpy.coordinates instead of sunpy.wcs. I have already attempted to do this in one of my PRs, and I am pretty familiar with coordinates, which will allow me to achieve this task easily within a week. I will further go about making minor changes in tests, so that it may pass, considering the changed codebase.

Week 2 - Week 3 (Jun 6 - Jun 19)

During this 2 week period, my aims will be to:

- Implement **image warping** for differential rotation
- Add suitable **test cases**

Since I am not very clear about how I will go forward with this, I have given a good 2 week window to implement this functionality.

Week 4 (Jun 20 - Jun 26)

This week will act as a buffer period for the work done so far.

- Refining the work done so far.
- It is not necessary that everything goes as planned out and there might be unavoidable delays due to a nasty bug hidden from eyesight. My target in this week will be to put a firm pencils down on work done so far - updating documentation and code cleanup.
- Adding any missing test cases and bug fixing

Phase 1 Evaluation

Week 5 (Jun 27 - Jul 3)

My aims for this week are to:

- Successfully implement OCCULT-2 algorithm
- Ability to easily extract coronal loops and store them in the object
- Plot the coronal loops back on a map (If time permits)

I have a well thought out plan for implementing this algorithm, and it can be well achieved in a week. If time permits, we can also have the loops successfully plotted back on a map. Otherwise, this will get carry forwarded to next week.

Week 6 - Week 7 (Jul 4 - Jul 17)

This 2 week period will consist of finalizing the OCCULT-2 algorithm, with running and base difference also implemented. My aims for this period are:

- Functionality to plot the loops back on the map

- Implementing running and base difference
- Writing suitable test cases

I have given a good 2 week time for this, since I still have to research on running and base difference and persistence transform.

Week 8 (Jul 18 - Jul 24)

This week will act as a buffer period for the work done so far.

- Complete any left-over task from the previous period.
- Add relevant documentation and code - cleanup

Phase 2 Evaluation

Week 9 - Week 10 (Jul 25 - Aug 7)

This 2 week period will consist of wrapping up FLCT code. The C code of FLCT is over 2500 lines, and since I haven't done wrapping before, I cant foresee how much time I will require to wrap up the full code. Still, I expect it to be completed in the 2 week period. If it is not achieved successfully, the task will be carry forwarded to next week.

Week 11 (Aug 8 - Aug 14)

My aims for this week are to:

- Wrap up complete FLCT code
- Add suitable documentation

Week 12 - Week 13 (Aug 15 - Aug 28)

If everything goes well, I will have **achieved my target** till now. This 2 weeks and further, I will **pick items from my wishlist** to work upon. If any other improvements are needed which are even remotely connected to `Sunkit-image`, I will be discussing it with mentors and picking it up. If not, I will be picking up a project or two that I have in mind.

GSoC period ends

Apart from the above mentioned schedule, I will be

- **Pushing code to my fork daily** so that my mentors can evaluate and keep track of whatever work I am doing.
- **Blogging every week** about the progress and related experiences in the said week so that mentors and others can get an overall summary of my week's work.
- **Send a PR** to the main master branch, as soon as the code is ready and cleaned-up, preferably **before each evaluation deadline**.

Software packages to be used:

Languages: Python, Cython

Modules : skimage, pytest, matplotlib

How I propose to complete the project:

I am pretty confident of completing this project because I have fairly good experience in python. I have been working on drms over 3 weeks now, and have become familiar with the data export requests and how to communicate with JSOC servers using both drms package and the existing JSOC Client. My one of the very first pull requests (that is still open), involved working on this same issue, which later got drafted into a 3 month project idea.

I have a good prior experience with open source, and have a good command over git and github. I will be pushing my changes to my remote fork daily, so that mentors can evaluate my work whenever they have time. I will also keep updating my mentors over my progress both through chat and blog. Having little experience in writing tests, I have already started learning more about pytest. I will spend a lot of my time to have a good hands-on experience before the coding period starts. This will help me to avoid any problems later.

Even after the GSoC coding period ends, I will be actively contributing to SunPy and be available if anybody has questions regarding my work.

Benefits to the community:

`Sunkit-image` is an external python package under SunPy, that will contain implementation of less used image processing algorithms. It may, at a later stage, be merged in the main SunPy repository. Algorithms, like OCCULT-2, will allow scientists to easily extract the coronal loops out of a map, and study it. Using sunpy.coordinates for differential rotation will allow users to input the coordinates in any system, and get back the output in the same system. More image-processing methods will be implemented in the future, and all will reside in this newly-built package.

GSoC

Have you participated previously in GSoC? When? Under which project?

No, I have **not** participated in GSoC before. This is the first time I am participating in GSoC.

Are you also applying to other projects?

I am also applying for the project **Drms module for JSOC Downloads**, in the same organisation SunPy. I am **not** applying to any other organisation.

Commitments

I may be involved in a short internship of 20 days from 10th May till 31st May. The work hours will be from 5:30 UTC - 11:30 UTC. Since, the internship will be over before the coding period starts, I won't face any problem in managing my tasks. Even in the community bonding period (during my internship), I will be able to give 5-6 hours easily on understanding the codebase and discussing with the mentors, since I will be free for most of the day according to UTC time.

I won’t have any involvement during the first 8 weeks of coding period. I will be able to give 8 - 9 hours daily, including weekends during the first 2 phases. My classes for the new semester will resume on 17th July. Due to very less academic load during the start of the semester, I will be able to dedicate around 6 hours everyday for the last 4 weeks of the coding period. Nevertheless, I have distributed my tasks in such a way that the last 4 weeks will only be given to documentation and will act as a buffer period.

Eligibility

Yes, I am eligible to receive payments from Google. For any queries, clarifications or further explanations, feel free to contact nitin.iitkgp23@gmail.com.