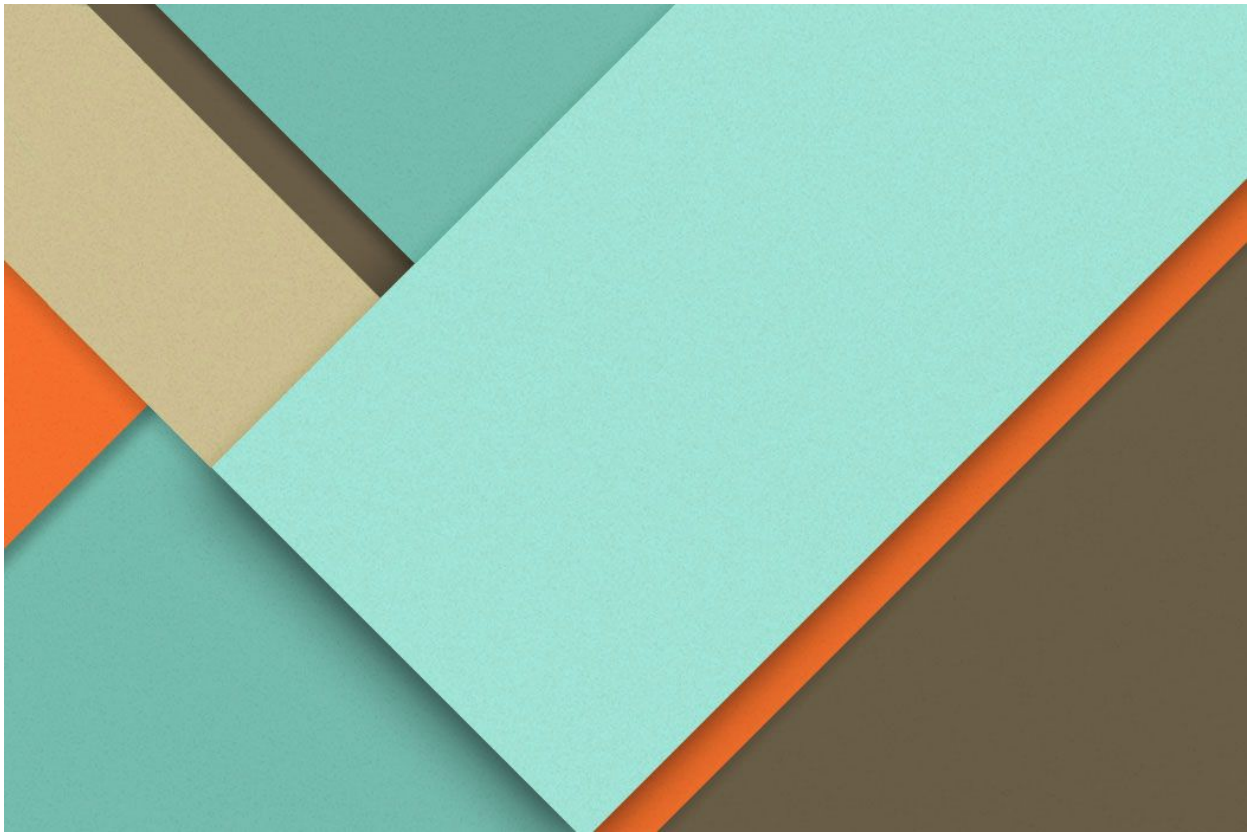

Drms module for JSOC Downloads

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. This is still in progress and I am planning to finish this if it is not taken up as a GSoC project.

- (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

The SunPy JSOC Client is a very rudimentary client to JSOC's export data CGI. The drms module provides a more complete implementation of the DRMS protocol. The drms module needs testing, and some other packaging improvements to automatically run the tests. This might involve adoption of some of the Astropy packaging code such as astropy_helpers, if necessary.

Once the drms package has been well tested, CI added and a conda package created, the SunPy JSOC client could be improved and extended to enable use of the majority of the drms functionality through the unified search API of SunPy. While the SunPy implementation would not need to provide all the features of the drms library, it would strive to provide a simple API for most queries supported by drms.

Finally, documentation in SunPy should be improved to detail much more of the JSOC functionality. This should include API documentation, narrative documentation in the guide and examples in the gallery.

Motivation

Why drms?

Here is a detailed study of the differences between the present JSOC Client and the drms module and why drms should be preferred over the present client. It also describes simultaneously the extra features that will be implemented upon integrating with the drms module.

- There is only a very little integration of the web API in the present JSOC Client. If one is unsure of the series name to query for, there is no way one can query for the data by providing similar or a subset of the series name.

  Whereas in drms, one can easily acquire all the series names available by `Client.series()` and all the series containing `hmi` in their names by doing `Client.series('hmi')`. Moreover, one can get all the primekeys and segments information interactively, while querying for information. Prior knowledge about the keys and segments is not needed.

- The present JSOC Client supports **very limited attributes**, as segments, while querying for data and filtering out information based on them. The keys that are supported presently are `Instrument`, `Wavelength` and `Level`, which are simple VSO attributes predefined.

  The drms module will allow us to filter out data on the basis of all the keys that are available for a certain series.

- The drms module **deals with 'segments' in a much better way**. While in the present JSOC Client, the segment is a VSO Simple Attr like `Wavelength` and `Instrument`, drms makes a clear distinction between keys and segments. As opposed to the present Client, drms **supports more than 1 segment name** in a single query. This will reduce export time drastically when querying for more than 1 segment simultaneously.

  For e.g., the query for 2 data segments continuum and magnetogram of hmi.sharp_720s data series at a certain given time will generate two different queries, denoted by the query strings :

```
ds = hmi.sharp_720s[2014.11.30_00:00_TAI]{magnetogram}

ds = hmi.sharp_720s[2014.11.30_00:00_TAI]{continuum}
```

This will result in generation of two different Export requests, hence taking a longer time to process the query. Using drms, the query can be made using a single query string.

The drms query will have the query string as :

```
ds = hmi.sharp_720s[2014.11.30_00:00_TAI]{continuum, magnetogram}
```

- The `protocol` is an attribute that comes in use only while making an export request, and has no use while querying for data. But, the present client takes protocol as an argument while querying for data, and not while making a real export.

  Integrating with drms will solve this problem, as **querying can be performed without specifying any protocol** at the beginning.

- The present client **supports only `fits` protocol** and has no option of downloading the files directly from the server by doing an `as-is` request. On one hand, where drms provides a good interface for making a direct `as-is` request, in cases where metadata is not important for a study, drms also provides a way to make an export request for other protocol.

Features of JSOC Client that are to be retained:

- Upon doing `Client.query()`, drms returns a `Pandas` data frame while the present client returns a `JSOCResonse` object. This contains an `astropy.table.Table` and also has attributes that stores the query parameters and the length of the table. Hence, `JSOCResponse` object will be preferred over a `Pandas` dataframe.

- The drms module provides only a very basic download routine that sequentially downloads the requested files. SunPy's downloader, instead, is a threaded-download manager which parallely downloads the requested files, hence greatly reducing the download time.

Sunpy's downloader also gives the user custom options to modify the downloading as required, for e.g. by passing the path as parameter, or an option to whether overwrite or ignore the files in case the names of files match. Path, if not passed, can be taken up from sunpy's config file and be downloaded in the default download directory.

Therefore, we will stick to using SunPy's downloader to get the files downloaded, after fetching the download URLs from the drms client.

The Problem and Approach

The project can be divided into 3 main parts:

I. Improving the drms module

The drms module needs a few improvements in its codebase. The current codebase has almost no test coverage. An important part in improving the drms package is to add tests for all the functions present in the drms module. The code coverage is expected to reach 100% after the midterms of the coding period.

After adding all the tests, and maybe simultaneously, we will be adding the Travis CI testing to check the build of the module whenever an event is triggered. Adding Coveralls will allow to keep a check on the percentage of codebase covered by tests.

If needed, we can make drms use astropy_helpers which includes many build, installation and documentation related tools, used by the astropy project. This won't have any user-end impact, but will provide very nice helpers such as `python setup.py develop` to install the current version, `python setup.py test` to run all the tests automatically, and `python setup.py build_docs` for building the docs locally. Moreover, it will also work nicely with astropy's CI-helpers for running tests on Travis etc.

After achieving a 100% coverage, and the build passing, we can create a conda package for the drms library.

II. Integrating SunPy to use drms

SunPy is currently rather restricted when it comes to accessing HMI and AIA data from JSOC, and the drms module provides a more general interface to access the data series from JSOC servers. Drms module can be used as a backend for the present JSOC Client, to improve communication with JSOC servers.

The current JSOC Client has limited attributes for generating the query string, and can't filter out the contents of data based on other keywords such as `QUALITY`. There is no option of downloading the metadata of the data-series exclusively.

The drms module will allow one to:

- Generally query any set of keyword from any data series, as well as any set of segments from any data series.
- Use the export data interface directly to export not only FITS files but also images and movies.

Though, SunPy will only need to use `as-is` and `fits` protocol. Moreover, the JSOC export system has a few issues with exporting images and movies, hence it is not a very good idea to integrate it into SunPy in the current state.

III. Thorough documentation

Drms:

Documentation and code go hand in hand. Presently, the drms package has a brief but sufficient documentation showcasing the basic query and export requests. Apart from the main documentation, there are a lot of examples present in the main repository for querying other types of data, with some variations. These examples can be well explained on the tutorial page itself, after creating a new section.

For e.g., the documentation of the function `Client.export_from_id()` is missing from the main tutorial. A usage that is present in the example script can be well documented and explained on the tutorial page. A lot of theoretical concepts on how JSOC interface works can be added, explaining well about the three-stage process of exporting data from JSOC, and the significance of the parameters that are being passed to the function, for e.g., `segments` and `keys`.

The overview in the documentation can be enhanced a bit, by adding some more in-depth information about the DRMS and the JSOC export system. The tutorial can be expanded well to include all the other features of drms, that are not shown in the main tutorial page, and which lies in the main repository. Also, examples can be added to the main tutorial from jupyter notebooks of drms-workshop.

Sunpy:

At present, SunPy has a good documentation about the API reference of the JSOC Client, but no documentation exists for the different use cases for querying data from JSOC. Since the project involves a complete overhaul of the JSOC Client, the API reference will have some changes as well. Apart from that, a lot of examples from drms, which are specific and relevant to SunPy, will be adopted to give an overall structure of how JSOC system works.

Examples will be added in the gallery, about all the different ways of querying data, either by inputting sunpy attributes, or directly entering the query string. All the variations of querying data will be well explained. A set of python notebooks will also be added, which will explain the use of the Client in fetching data from JSOC. The first two python notebooks will explain in detail the exporting of data in both `as-is` and `fits` protocol. The third notebook will explain how to fetch metadata and image data separately, and combine it client side (if this functionality has been achieved).

Since, SunPy's client won't strive to have all the functionality provided by drms, it will be made clear in the documentation about which functionalities will be supported by SunPy. Advanced functionalities will be mentioned in SunPy's documentation too, but it will be properly linked to drms' documentation, and made clear to the users to directly use drms for such advanced queries.

Implementation

Improving drms module

The code needs to be covered by tests. The module consists of 3 main classes `SeriesInfo`, `ExportRequest` and `Client`. The tests will, first be written for the main API, and then consecutively, internal functions will be covered. `query` and `export` will require use of `unittest.mock`. A `MagicMock()` object will be used, which will create all related objects and methods, so that we don't need to interact with JSOC Servers, every time the test is run.

`@patch` decorator available in `unittest.mock` allows us to monkey-patch an object, within the scope of the decorator. This will allow us to replace the JSOC database with a custom object, and fetch data from the custom object itself, thus avoiding interaction with JSOC servers.

Since, I haven't used `unittest.mock` before, I will be working on it in next 2 months, and get familiar with it before the coding period starts.

Integrating drms into SunPy

- **Depreciation of compression property**
  JSOC no longer support exports of uncompressed FITS files containing integer types. Most data held in the JSOC are compressed. Moreover, upon doing an export request, the files are downloaded in the same format in which the files are stored online, implicitly.

- **Downloading metadata without downloading associated image data**

  Drms will allow the metadata to be downloaded exclusively without any image data associated.

  ```
  c = drms.Client()
  r = c.query('hmi.v_45s[2016.04.01_TAI/1m]', key='**ALL**')
  ```

Upon doing the above, a Pandas Data Frame is returned which contains all the metadata of the given series. This will be converted into sunpy.util.metadata.MetaDict, which currently holds the metadata of a Map object. About the interface, a boolean parameter will be added in the function get() and get_request() [or, some similar function, in case a change is made in the API], which will take input from the user whether to exclusively download the metadata for a series, or download the image data associated too. A function get_metadata() will be added which will take T_REC and other keys as input, and will return a sunpy.util.metadata.MetaDict object as output.

- **Add other primekeys as attributes for query**

  Common prime keys like HARPNUM and CAMERA should be added as attributes. This will allow downloading of data based on these primekeys too. Currently, only T_REC is supported. A detailed study and discussion is yet to be made regarding which are the most common primekeys across all the data series. The other uncommon primekeys can be interactively passed after doing a LookData and the query_string be generated thereafter.

  A dynamic attribute creation can be implemented in this case, depending upon the keys and primekeys of the queried series. This will also allow auto-complete of the primekeys.

- The query function should have another attribute query_str which, if provided, won't require any other attribute as input. This query_str will then directly be passed to drms.query() to fetch data. Adding this functionality is necessary because a query string can be modified in a number of ways to filter out data, and a function to generate query string is viable only if the number of arguments are small. In case of large number of arguments or keys, a manually generated query string is more suitable.

- **Allow downloading of metadata and image data separately**
  Solar data, are stored in the form of metadata and image data at two different systems. Data are merged together, upon export, in the form of FITS files at JSOC. This merging of data on the server side takes up some time. If instead, the metadata and image data can be downloaded separately, and be merged together on the client side, it will greatly reduce the server load and the export time.

  There are still discussions going on about how to fetch both the data separately. The problem with the metadata is that the metadata obtained by using `client.query()` is not same as the metadata found in the FITS files. Some kind of a mapping exists, between the keywords. There exists very little documentation on mapping keywords from metadata to FITS files. Hence, it will be better if JSOC directly provides with the modified metadata, that is found in the FITS files.

  Discussion about this is in progress, and I will work upon it once a clear solution is found, after discussing it with mentors.

Timeline and Deliverables

Community Bonding period (May 4 - May 29)

The community bonding period will mainly be for getting to know more about the codebase and learning to write tests. I am already pretty familiar with the JSOC client, but I have to look into Fido client since it uses JSOC's api. I have spent much time playing with export requests on the JSOC interface, and am familiar with the drms module. Yet, more of time is to be given to understand the inner functionality of drms package. Since I have had little prior experience in writing tests, I will devote a larger part of this month in learning to write sample tests in Pytest and using mock. I will constantly be in touch with mentors, and getting help wherever needed. By the end of this period, my aims are to:

- Get familiar with FIDO Client
- Get familiar with all the JSOC functionalities and compare in a greater detail, JSOC online interface, drms, and the present client.
- Learn about Pytest and mock, and have started writing tests for increasing the coverage of both drms and sunpy.

- Discuss with mentors about how to combine header and image data on the client side, once a way out has been found for downloading them separately.

Coding period begins

Week 1 - Week 3 (May 30 - Jun 19)

The first 3 weeks of the coding period will solely be given for writing tests. I will go about doing this in 3 steps :

Step 1 : Write tests for the **main API**. This will include tests for the three main classes `SeriesInfo`, `ExportRequest`, and `Client`. The main functions such as `query()`, `export()`and `series()` will be covered in this week. Mock objects will be created to do the test with. The code coverage is expected to reach **upto 40-50%** at the end of this week.

Step 2 : This week will require writing tests for the **lesser used functions** such as `info()` and `keys()`. The code will be well covered after this week, and the coverage is expected to **reach to about 80%**. Still, there will be a lot of internal functions which will have been already covered, but need to have separate tests. Thus, separate tests will be written for the **internal functions** and will be covered independently.

Step 3 : This week's target will be to achieve **the coverage to 100%**. All the internal functions will have been independently tested, and the missing out code (if any), that are not covered will be dealt with. Finally, the **tests will be well documented,** so that they can be well understood by developers trying to improve the tests.

Week 4 (Jun 20 - Jun 26)

This week will act as a **buffer period** for Part-1 of the project.

- **Refining the work** done so far on tests.
- 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**.
- Get the **Travis CI** running, **Coveralls** installed
- **Conda package** for drms built and released

Phase 1 Evaluation

Week 5 - Week 7 (Jun 27 - Jul 17)

All works and **modifications on the drms package will have been done** before Phase 1 deadline. Second phase of the GSoC period will be given to Part 2 of the project, on integrating drms with Sunpy.

Step 1

- Have integrated drms package in SunPy, with **basic functionalities**, and changed API
- Remove all related bugs that appear during porting

Step 2

- Improve the JSOC Client so as to **keep the API as similar** to before as possible
- More improvements in the client, to bring in **additional features** that drms has to offer.
- Allow querying of metadata and image-data **separately**.

By the end of this week, the SunPy's JSOC Client will have all the basic functionalities for querying data, and all the old supported features will come back.

Step 3

- Add the function for directly taking `query_str` as input
- Make JSOC Client **interactive** while querying based on keys and segments
- Add the functionality for **dynamic attribute creation**
- Support **tab auto-completion** of attributes

By the end of this 3 week period, **80% of the task** will have been achieved. A few bugs, here and there, and writing test cases will be done in the coming weeks.

Week 8 (Jul 18 - Jul 24)

This week will act as a **buffer period** for Part-2 of the project.

- Complete any **left-over task** from the previous period.

- After such an overhaul, there are chances of bugs getting introduced in the code. This period will consist of **rigorous testing** of the code, and solving any of the bugs that arise in the process.
- Keep a track of all the **corner-cases** that can be added to the tests later (next week).
- In case, the week goes a little empty, I will start over the tasks of **adding tests for the JSOC Client.**

Phase 2 Evaluation

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

After Phase 2 Evaluation, the major work remaining will be that of documentation and a few tests for covering the code developed during the past weeks. This 2 week period will comprise of:

- Covering the jsoc module with **tests**
- Finalize the code for `sunpy.net.jsoc`, i.e. **rigorous final testing**, and **code cleanup**
- **Documenting** the drms package

Week 11 (Aug 8 - Aug 14)

Possibly, this will be my **final week** for GSoC. This week will solely be given for documentation of Sunpy's jsoc module.

- Make changes in **API reference** (if any)
- Add a **narrative documentation** in Sunpy's user guide
- Add a variety of working **examples in gallery**

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 `drms` or `sunpy.net.jsoc`, 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:

Language: Python

Modules : drms, pytest, mock

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:

The API of drms module is quite stable, yet tests are a necessary part of any package. Improvements in the drms module will benefit the community as it is the main package for interacting with the JSOC servers and downloading AIA, HMI and MDI data. Integrating the drms package with SunPy will improve the JSOC client to a large extent. This will enable a smoother way to access the data from JSOC. Documentation and gallery examples will help the scientific community understand JSOC easily, and will provide a better explanation to how to download data from JSOC.

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 **Sunkit-image**, 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.

****