**PSPT Sunspots**

CU Boulder CSCI 7000 Semester Project

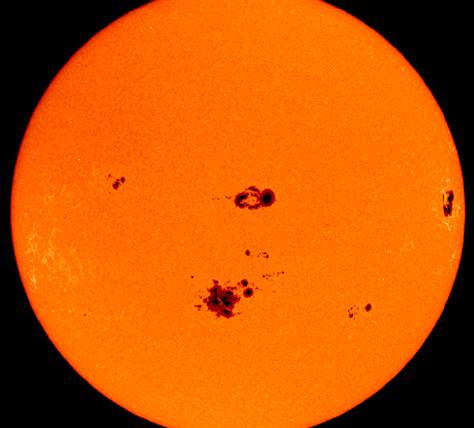
Jamie Mothersbaugh

This document is a ReadMe to accompany the pspt\_sunspots GitHub repository. It describes the project, the method of analysis and a discussion of results. Additionally, it details the code, explains how to run the code and describes the outputs of each Python script.

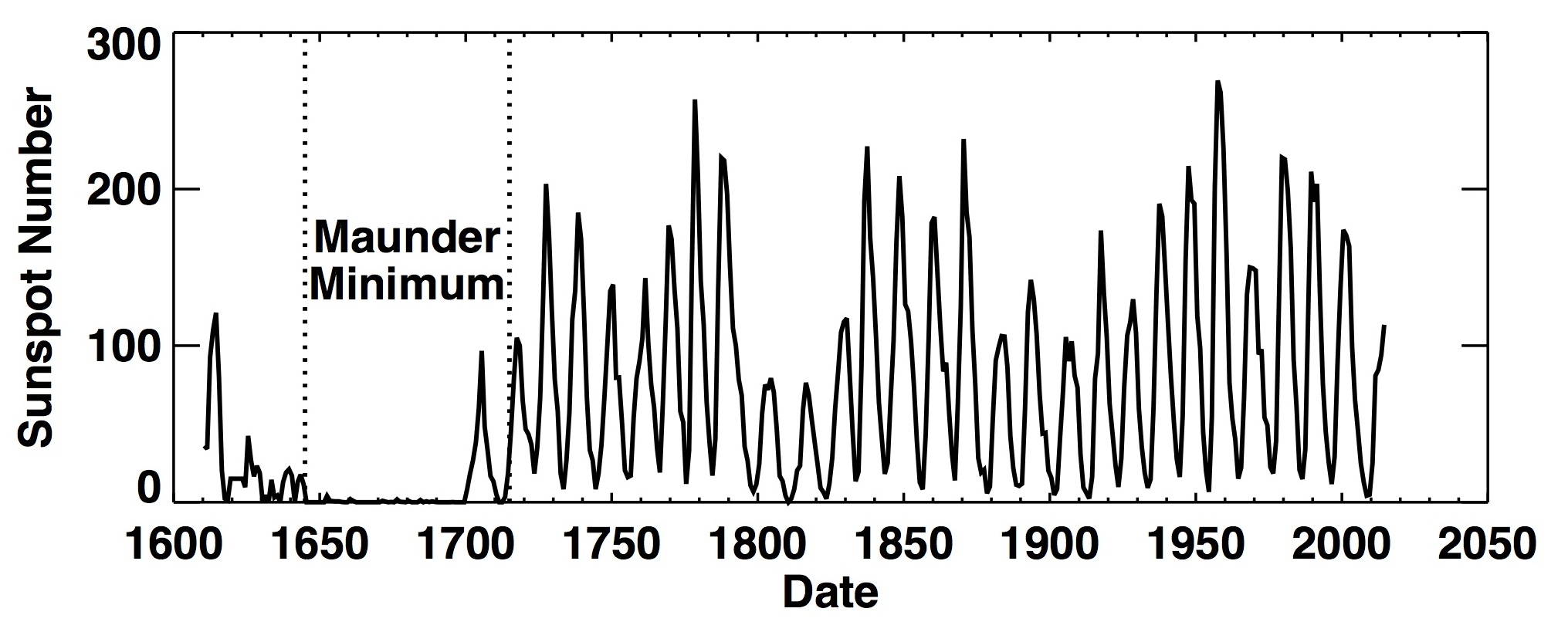
**Project Overview**

The PSPT (Precision Solar Photometric Telescope) was a visible-light telescope on Mauna Loa in Hawaii. It took images of the Sun with 3 different filters at 393nm (calcium), 409nm (blue light) and 607nm (red light). It took ~daily data from March 1998-June 2015. Because the images are taken in visible light, sunspots are apparent on the image of the solar disk.

This project aims to recreate the sunspot record by determining the number of sunspots as a function of time visible on the solar disk. The number of sunspots on the solar disk varies in an 11-year cycle called the solar cycle. Sunspots form when local density of magnetic flux increases at a location on the photosphere (surface of the Sun). These fluxes prevent photons from reaching the photosphere, so the sunspots appear darker than the surrounding photosphere.



The sunspot record is one of the oldest scientific data sets, existing since the 1600s.



**Data Acquisition & Organization**

The PSPT data is available at <http://lasp.colorado.edu/pspt_access/>. Instructions for downloading the data are on the ‘Downloads’ tab at the top of the page. The LaTiS method is the most efficient for downloading the data.

The entire data set is ~2 TB. It is important to download the data in reasonably-sized increments- for example, downloading data in all filters in 1 month or downloading all data in 1 filter for 1 year.

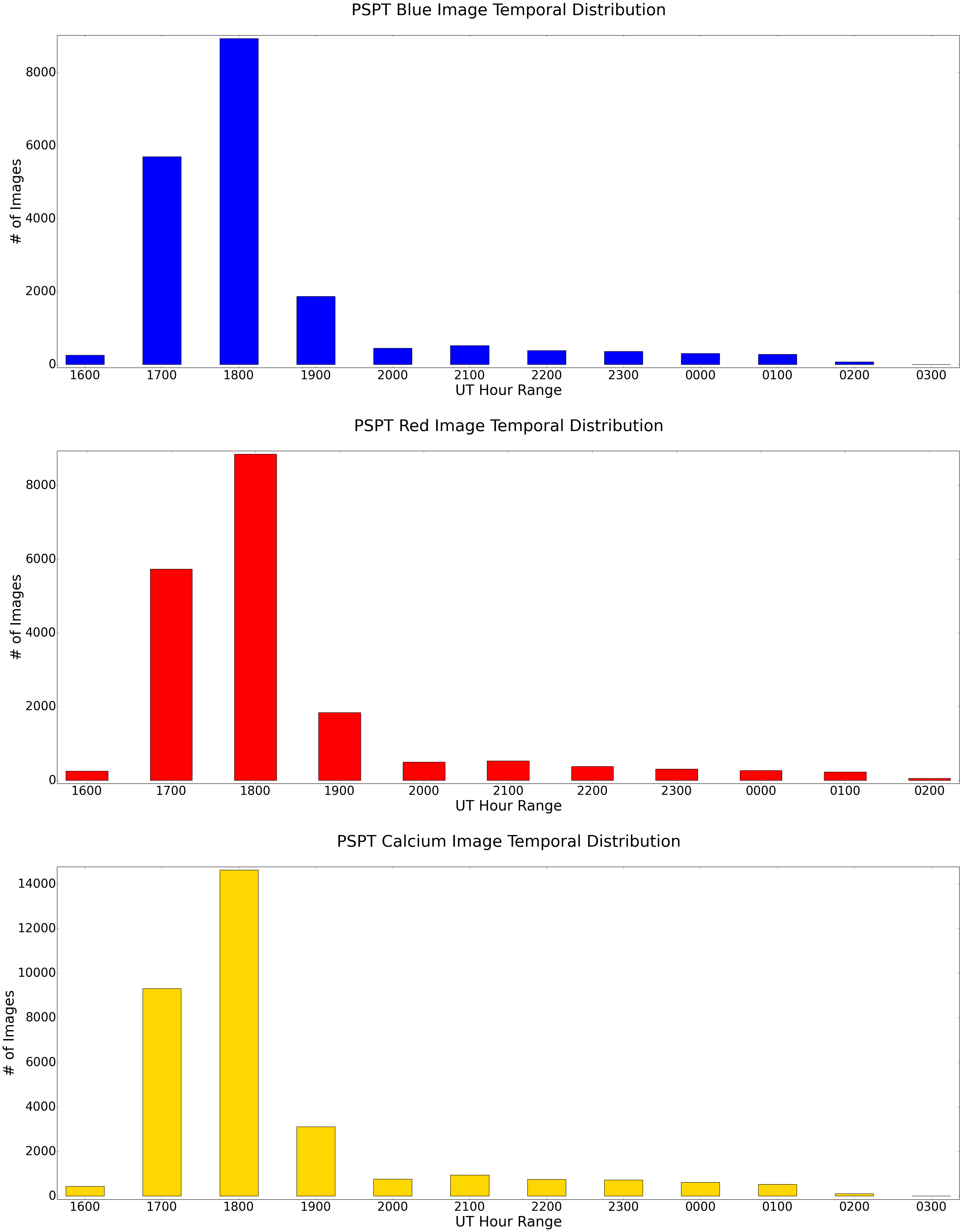
Once I obtained all the images, I looked at the release notes and the information on the PSPT website. These indicate the best images were taken between 16:00 and 19:00 UT. The next step was to trim down the data to include only the highest-quality images.

$ ls \*.HourHour\*jpg | wc –l

This command lists the number of filenames that start with the chosen .HourHour timestamp value.

I determined the number of images in each hourly bin from 1600-0300 hours UT time. For each filter, I made a bar graph showing the image distribution as a function of hourly bin. This graph is made by running pspt\_data.py:

$ python pspt\_data.py

The script produces the plot below.

For all 3 filters, 1800-1900 hours contains the most images. However, looking at the data, there are often multiple images in this hour range on any given day, taken minutes apart. Because the goal of the project is to track the sunspot frequency over long (decades) timescales, I decided to trim the data down to contain only 1 image per filter per day on the days for which there is data. This can be done using the commands below.

$ ls \*jpg > filter\_jpg\_all.txt

This command writes all filenames into a text file. Not all images can be listed at once because there are too many files. List by subgroup instead- e.g., 199\*, 200\*, 201\*. To list the images into a file that already exists:

$ ls \*jpg >> filter\_jpg\_all.txt

$ sort -u -k1,1.8 jpg.txt > filter\_jpg\_unique\_days.txt

This command compares first 8 characters (YearMonthDay substring0 of every line and list lines with unique YearMonthDay strings into a separate text file. Only the first unique occurrence is listed into the file.

$ find . -name "\*" | grep -vFf filter\_jpg\_unique\_days.txt | xargs rm -rf

This command deletes all files whose names are NOT in the file of the filenames with unique YearMonthDay strings.

The commands above are repeated for the FITS images for the same filter, then both the JPG and FITS sorting is done on the remaining filters.

After organizing by unique year, month and day, the following data remains:

3277 unique days of blue images (1 image per day)

3267 unique days of red images (1 image per day)

3362 unique days of calcium images (1 image per day)

The final data set is now comprised of images in the 18:00-19:00 UT range, consistent with the highest-quality images.

**Analysis & Results**

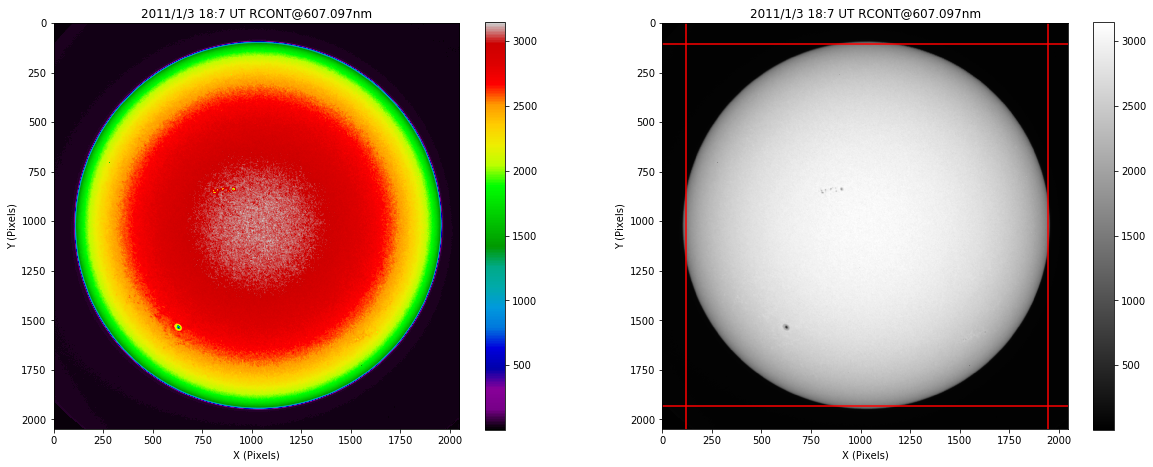
The image data is stored in FITS (Flexible Image Transport System) files, a standard format of astronomical data. The Python astropy library contains a number of modules that enable reading & manipulation of FITS data.

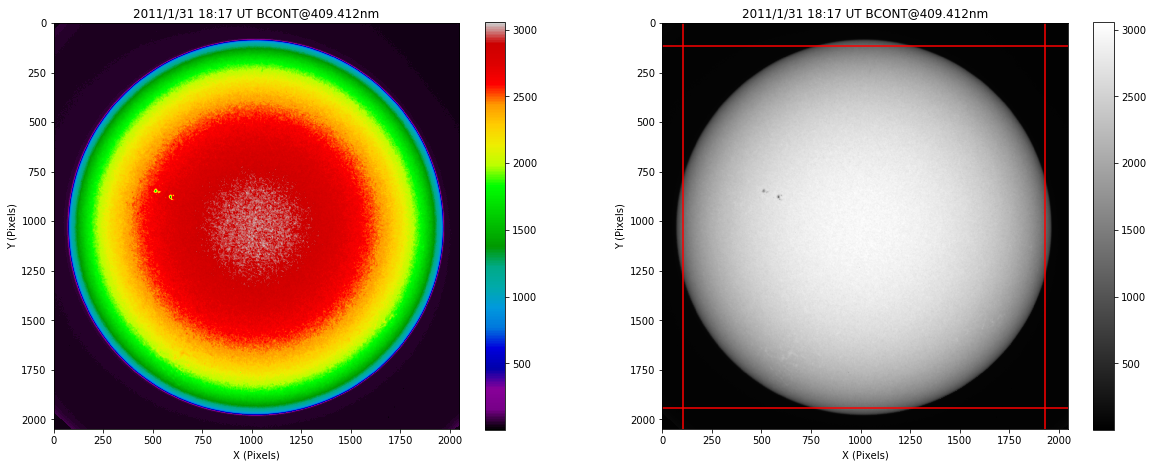
The code analysis for this project used the astropy FITS library to read & parse the headers of the images, and to analyze the actual image data in a 2D numpy array. My original method was to scan the solar disk in each image to locate the dark pixels (corresponding to sunspots), and then determine the number of sunspots in each image. However, this proved to be difficult to do. Basic Python search functions, like numpy.where(), did not return the sunspot pixels relative to the original FITS array. The code would return the total number of sunspot pixels, without any indication as to how many were in each sunspot.

An additional idea was to scan the array linearly- slice across all x-pixels at each y-pixel to determine the sunspot pixels, then cross-correlate the x and y-pixels to determine the locations, and thus the number, of distinct sunspots. This is perhaps a reasonable method for analyzing a few images, but running this code on several thousand images would be a very slow, inefficient method of computation.

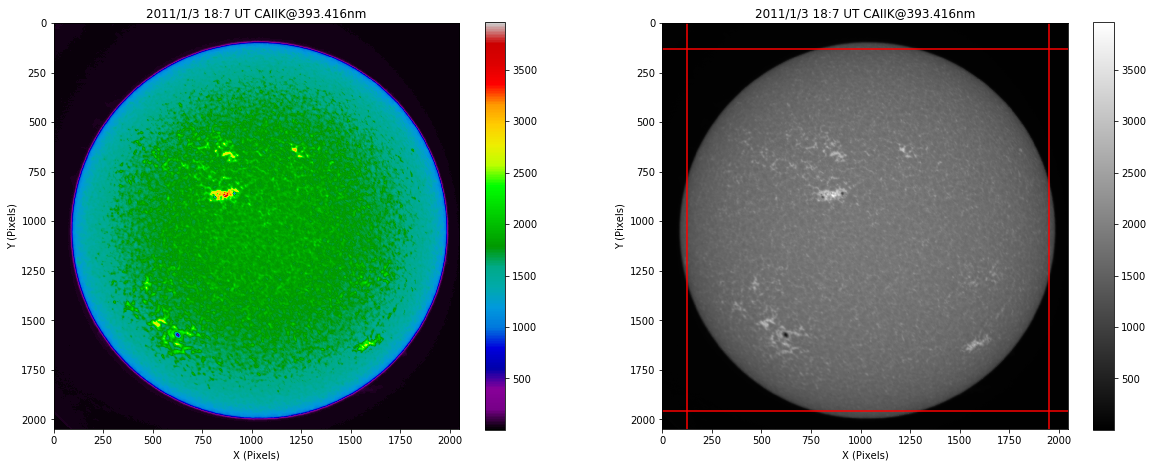
I next tried to mask out the solar disk in the FITS image array, then use ratios of the sunspot and disk areas in pixels2 to determine the total sunspot area relative to the total disk area, then assume each sunspot is of an average size to determine the number of sunspots. I came close to making a solar mask- I was able to make an array whose indices corresponded to a circle with the radius & center (x, y) location of each image, but I couldn’t figure out how to take the FITS array values at these indices and put them in the solar mask.

The plots below show the FITS image array in 2 different color maps: spectral and grayscale. The grayscale image is the same as the .jpg images that accompany each FITS file. I also drew the image ‘window’ that I used to find the sunspots (detailed below) on the grayscale image.

Red image from January 3 2011. Sunspots are visible in both color maps.



Blue image from January 31 2011. Sunspots are visible in both color maps.



Calcium image from January 3 2011. Sunspots are visible in both color maps.

I used the following method to determine the sunspots:

Read in FITS image and parse the header to determine:

Dark values (average value per quadrant)

Image date (average of start & end date)

Image wavelength & filter and image geometry ((x, y) pixel positions of the center of the solar disk, x and y pixel radius of the solar disk, x and y pixel positions of the edge of the solar disk)

Subtract the average dark value from the image to remove the pixel values of the background sky behind the solar disk

Use the image geometry to draw a ‘box’ around the solar disk. This would ideally be a mask of the solar disk, but a box proxy is the best I could do.

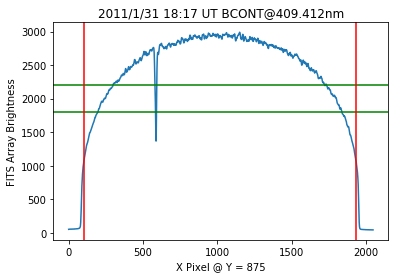
Find all pixels where the FITS array was > 1800 and < 2200 (arbitrary brightness value of the FITS array). I determined this range by slicing across the FITS image in x and y to see the brightness profile of the image.

Assume each sunspot is the same, average size (circular, radius of 5000 km)

Determine the total number (area) of sunspot pixels and the total number (area) of solar disk pixels, including sunspot pixels

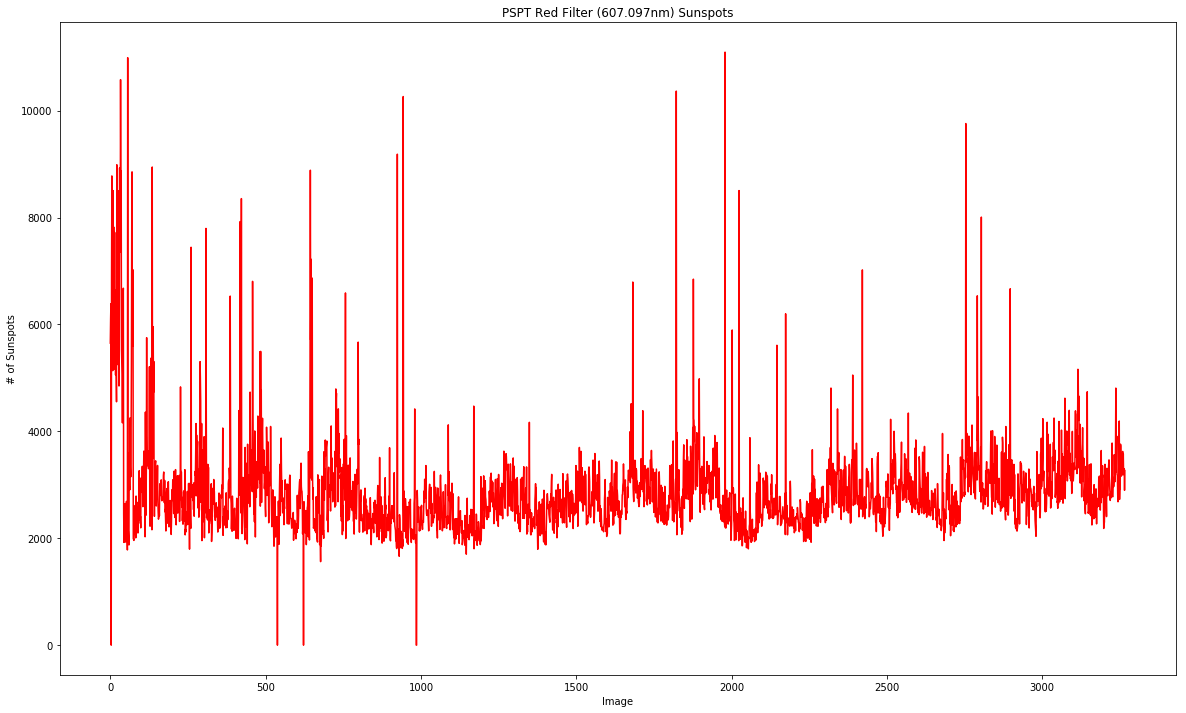
Take ratios to find the total number of sunspots on the solar disk

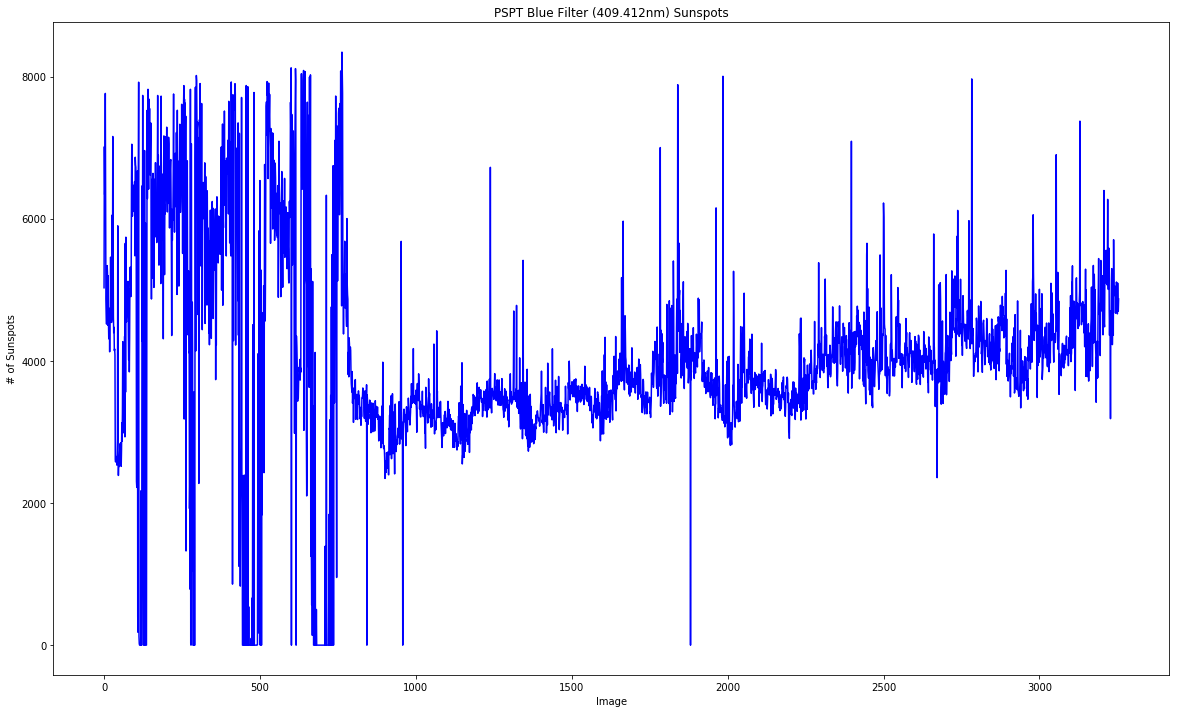
Repeat the analysis above for each of the images in a single filter

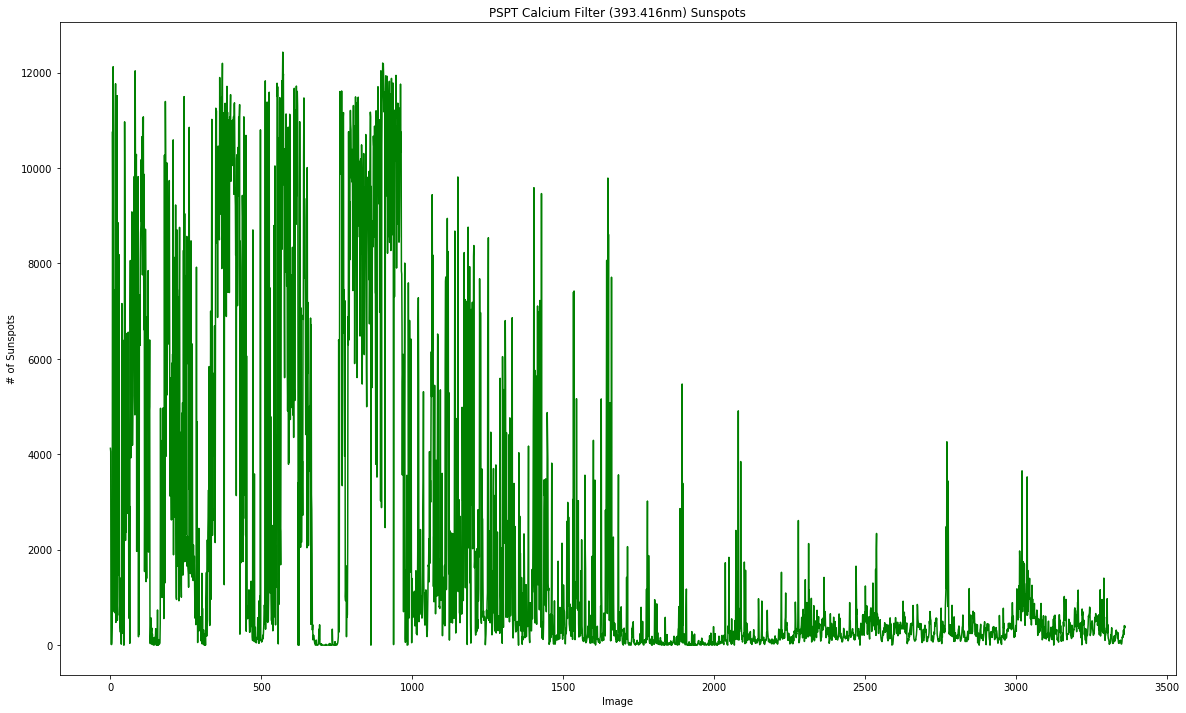


Slice across the blue image plotted above. The blue line shows the brightness across the solar disk- the sunspot is the dip. Green lines show the pixel brightness range used to classify sunspots. Red lines show the x-pixel edges of the solar disk, taken from the image header.

Once the image analysis was complete, I plotted sunspot number vs. time. Below are plots of the red, blue and calcium sunspots. Additionally, in the plots below the x-axis is not by time but by image number. I can’t figure out how to plot the unique year/month date strings as the x ticks.

Red sunspots vs. time

Blue sunspots vs. time

Calcium sunspots vs. time

My method is doing a poor job of identifying sunspots. There is no discernible trend in either plot and the magnitude of identified sunspots is a factor of ~10 too large. This is most likely due to the use of a square mask on the image instead of a circular mask. The code is likely identifying background sky pixels behind the solar disk as sunspots and including these pixels in the sunspot pixels.

Additionally, the pixel value range used in the code is probably incorrect for some images. In the calcium images in particular, the pixels surrounding the sunspots show significant variation from both the solar disk and the sunspot pixels.

**Improvements**

There are several improvements that could be made to the code and analysis methods on this project to derive a more accurate sunspot record from the image data.

The most significant improvement is to use a circular mask on the image to block out the background sky, allowing the code to scan only the circular disk. This would prevent any background sky pixels from being included in the sunspot pixel count and would improve the sunspot identification.

In addition to simple array & index manipulation with numpy to create a circular mask, the Python library OpenCV is an image processing tool that enables masking, drawing and edge detection on images. A sunspot mask could be defined in OpenCV and used to methodically scan each image. In conjunction with a solar image mask, a robust image processing tool would allow for direct location of the sunspots in each image and remove any need for size approximation or pixel brightness ranges.

As seen in the images above, the typical sunspot pixel value varies based on the image filter. Filter-specific sunspot brightness ranges would avoid incorrectly identifying surrounding photosphere pixels as sunspot pixels and would not over or underestimate the pixels determined to be sunspots.

Finally, comparison to the known sunspot record would provide a thorough test of both the code used to identify the sunspots in the PSPT images and the quality of the images themselves. This comparison could be done by overplotting the known sunspot record, or using function fitting tools from Python scipy or other libraries. Comparison to any known data isn’t possible with the results I currently have in this analysis.

**Code Guide**

Below is a guide for running the Python scripts in this repository and understanding the outputs they produce.

plot\_pspt\_blue\_image.py- makes ‘blue\_image.png’ and ‘blue\_slice.png’

plot\_pspt\_red\_image.py- makes ‘red\_image.png’

plot\_pspt\_calcium\_image.py- makes ‘calcium\_image.png’

pspt\_data.py- makes ‘pspt\_data\_distribution.png’

plot\_pspt\_sunspots.py- main script that calls find\_pspt\_fits\_images\_sunspots.py; makes ‘pspt\_red\_sunspots.png’, ‘pspt\_blue\_sunspots.png’ and ‘pspt\_calcium\_sunspots.png’

find\_pspt\_fits\_images\_sunspots.py- function called by plot\_pspt\_sunspots.py; parses & analyzes each FITS image

Additionally, the text files below were made during the data acquisition & organization process described above.

calcium\_fits\_unique\_days.txt- list of calcium FITS images used in analysis

calcium\_jpg\_unique\_days.txt- list of calcium JPG images (each FITS image has an accompanying JPG image)

red\_fits\_unique\_days.txt- list of red FITS images used in analysis

red\_jpg\_unique\_days.txt- list of red JPG images (each FITS image has an accompanying JPG image)

blue\_fits\_unique\_days.txt- list of blue FITS images used in analysis

blue\_jpg\_unique\_days.txt- list of blue JPG images (each FITS image has an accompanying JPG image)

A test of each script is included in the pspt\_sunspots\_test.sh script. However, this script may fail if Python can’t find the astropy library. In this case, Python won’t be able to read any of the images and the code will crash out.