THIS IS A NICE WALKTHROUGH OF SPC DATA, BUT IT NEEDS UPDATING TO PUBLIC FILES

PSP/SPC Data and Software Familiarization Worksheet

2019 REU Program at Smithsonian Astrophysical Observatory

# Introduction

The purpose of this document is to provide a step-by-step introduction to using SPC data within a python environment. This worksheet assumes that the user has a fully-functioning python environment set up on their computer already, but does not assume any prior knowledge of the python programming language.

The data being to be used in this project are from the Solar Probe Cup (SPC), an instrument on the Parker Solar Probe (PSP) spacecraft, which is currently in a heliocentric orbit with a period of 3-4 months and a perihelion between 35 and 9.86 solar radii (Rs). The SPC instrument is a Faraday cup that measures the plasma that makes up the solar corona and solar wind (primarily protons and alpha particles with energy/charge between ~150 V and 6000 V).

SPC produces data nearly continuously whenever the spacecraft is closer than 0.25 AU from the Sun, and sporadically outside of that distance. The instrument sends its data to the spacecraft (via the suite processor, the “SWEM”), where it is saved in binary files (so-called “Level-0” or “L0” files). The spacecraft sends those L0 files to the ground via the deep-space network, which forwards them to SAO. SAO processes these data into higher-level data products that are usable by the science community. The table below shows those data products:

|  |  |  |
| --- | --- | --- |
| **Data Processing Level** | **Description** | **Publicly Available** |
| L0 | Binary files with all types of data combined, and not necessarily sorted by date/time | No |
| L1 | Files separated by “application identifier” (APID), time-ordered, and sorted into daily files, and converted into CDF format | No |
| L2 | Data units are converted from digital numbers into physically-meaningful units. In general, the level-2 files contain “spectra”. That is, they contain the current or flux of particles measured as a function of energy/charge.  Files are separated based on content:   * l2: “master” file: all science data combined * l2i: data only from “ion mode” * l2e: data only from “electron mode” * l2ifa: data only from “flux-angle mode” * l2ical: data from calibration periods while in ion mode * l2hsk: data from the housekeeping packets | Yes |
| L3 | The spectra from the L2 files are analyzed to produce an estimate of the bulk fluid parameters of the plasma (i.e., velocity, density, temperature). This can be done by taking moments of the spectrum (or some portion of the spectrum) or by fitting the spectrum with a functional form whose free parameters are the fluid parameters of interest (e.g., a Maxwellian). The resulting values from those multiple methods are all contained in these files. | Yes |

For the purposes of this project, you will probably end up concentrating on the L3 files, but this worksheet will also familiarize you with the L2 files.

# Data Access and Filenames

**The data are available on the web at:**

[**http://w3sweap.cfa.harvard.edu/data/sci/spc/**](http://w3sweap.cfa.harvard.edu/data/sci/spc/)

You will need a username and login to the website. Contact Kelly Korreck to attain those credentials if you have not already received them.

Additionally, the data are available on a server at the address: laz.cfa.harvard.edu. You can access the files there through an FTP program (e.g., WinSCP), or perform analysis on the laz server so that your programs will have local access to the data directory at /psp/data/sci/sweap/spc/

The data are organized into separate directories for the L2 and L3 files, and then organized by date, so that the path to a file for the date 2018/11/01 might look like:

<http://w3sweap.cfa.harvard.edu/data/sci/spc/L2/2018/11/spp_swp_spc_l2i_20181101_v16.cdf>

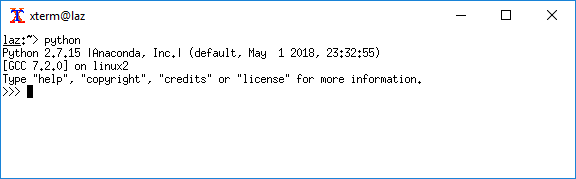
The final number in the filename (following the “\_v”) is the version number of the file. **You should always use the file with the highest version number available.**

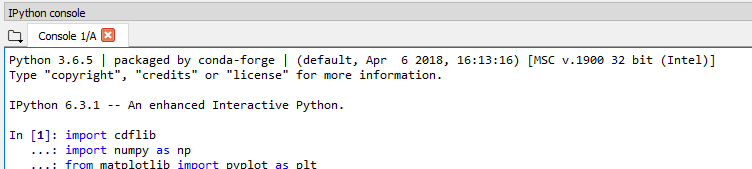
The data are stored in files in the “Common Data Format” (CDF). Some information about that format can be found at NASA’s CDF website ( <https://cdf.gsfc.nasa.gov/> ). Briefly, it stores data and meta-data (extra information that tells users about the data) in a way that can be easily accessed using a variety of programming languages.

# Opening and Examining a CDF Level-2 File

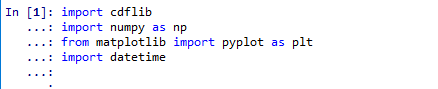
(Note: I’ll be showing these steps on a computer running Linux, but generally the steps are the same if you’re using a Mac)

First, open an interactive terminal window (or a development environment such as “spyder” so that you can open and examine the CDF file. You will see a prompt at which you can type commands. The first image here shows what it looks like to run directly in an xterm window, and the second image shows what it looks like in spyder.

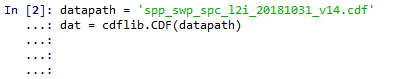




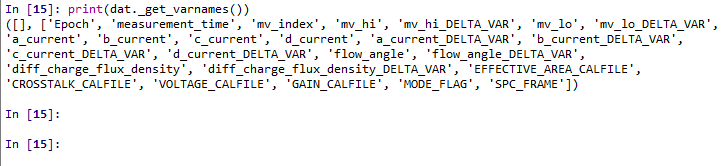
Now you need to import the packages that we’ll be using. Of particular importance is the pysatCDF package, which interfaces with CDF software to read in CDF files. Additionally, the “numpy” package provides a lot of useful features like performing mathematical functions on arrays. The following image shows the commands needed to import those packages.



Now we need to define the path to the file we’re going to try and read. In this case, I’m using the laz server, and so I can specify the path to the data directory on that server. If you’re working on your local machine, then you’ll need to use the local path to the data file. Then you can read in the file with the cdflib package.



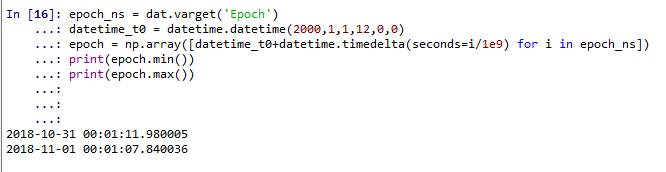
Now you have a variable called “dat” that contains all of the data from the CDF file. Now we can inspect what variables are within the file by using the “keys” method.



Now we can see the data variables that are contained within the CDF file that we read in. In this case (a Level-2 file), we have:

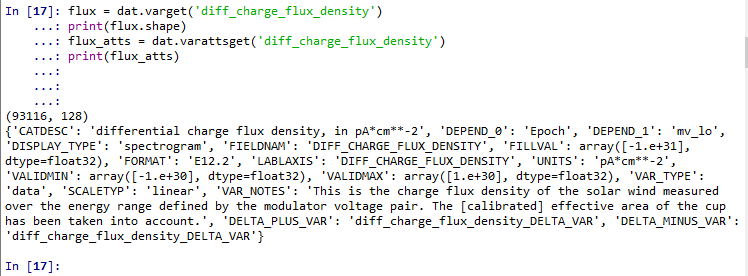
* Epoch: the time of the beginning of each spectrum
* measurement\_time: the time that each individual energy/charge window was measured.
* a\_current, b\_current, c\_current, d\_current: this is the current that was measured on each collector plate
* diff\_charge\_flux\_density: the summed current of all 4 collector plates
* flow\_angle: the incidence angle of the plasma into the instrument at the peak of each spectrum.
* mv\_lo and mv\_hi: the low and high value of the modulator voltage during each energy/charge window
* \*\_DELTA\_VAR: the uncertainty associated with each variable

The “Epoch” variable is the dependent variable, upon which all other variables depend. So let’s look at that one a bit more. Again, since we’re going to be using it regularly, let’s make it easier to type by making a new reference to it, and converting from its native units (nanoseconds since 2000,1,1,12,0,0) to a datetime object. Then we can look at the extent of that variable by using the “min” and “max” functions.



You should notice a couple things. 1) When copying the ‘Epoch’ variable to a new variable name, we used the “copy” method, which actually reads the data from the CDF file on disc and stores it in RAM. This makes it much faster to access, and actually stores it as a numpy array-ish object that allows for easier manipulation. 2) The pycdf package has automatically converted the ‘Epoch’ variable into a “datetime” object (python’s way of storing date and time in a single variable).

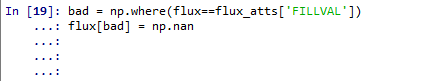
Let’s also grab the data called “diff\_charge\_flux\_density”, which is the total current on all 4 collector plates, and represents the number of particles that were measured during each energy/charge window. Then let’s look at the shape of the variable, and the attributes for that variable.



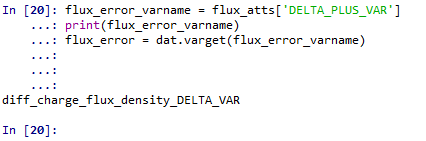
We can see that it is a 2-dimensional variable from our call to “flux.shape”. We can get the attributes for each variable using the “varattsget” function. These attributes tell us lots of useful information about the variable, like what exactly it is (“VAR\_NOTES”), and what it depends on (“DEPEND\_0” and “DEPEND\_1”). In this case, since the variable is 2-dimensional, it depends on two things: “Epoch” (which will be the same for all of our variables), and also “mv\_lo”. If we look at the shape of the “epoch” variable, we will see that it is the same length as the 0th dimension of the flux variable:



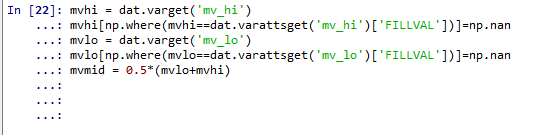
Another key thing that we see in the attributes is the “FILLVAL” attribute; in this case it is “-1e+31”. That means that wherever this variable equals that value, we should disregard that data point. We can use numpy’s “where” function to locate positions in the array where we have fill values, and replace those with “Not a Number”, so that it will be more easily plottable:



Also, we can use the attributes of the flux variable to figure out the name of the variable that contains the uncertainties associated with our data.



We already know that the flux variable depends on time (“Epoch”) and “mvlo” (recall from looking at the attributes). Additionally, we have more information about the voltage window in the “mvhi” variable. Let’s also get that variable and construct a new variable called “mvmid” that contains the center voltage of each voltage window.



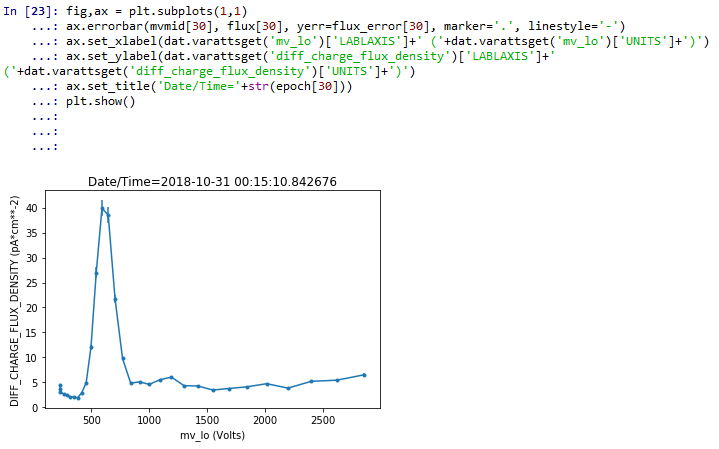
Now we can plot one of the spectra. This data file contained 51,543 separate spectra (the same as the length of the “epoch” variable). Just for kicks, let’s plot the first one.

*(1st line with >>>: sets up a figure and axes to which you can plot*

*2nd line: plots the 30th spectrum with error bars*

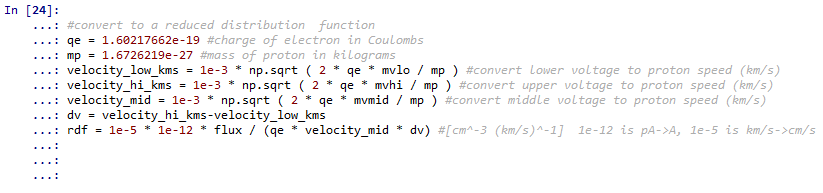
*3rd-5th lines: uses attributes to set axes labels and titles*

*6th line: shows the plot to the screen)*

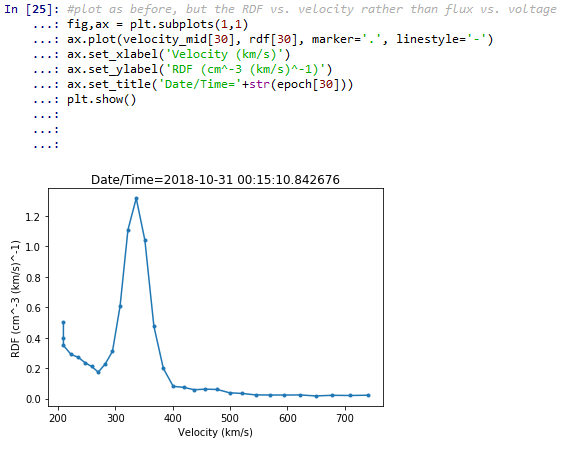


The peak of the plot (at about 700 V) is the kinetic energy at which the most protons are moving and therefore the energy at which we measured the highest current.

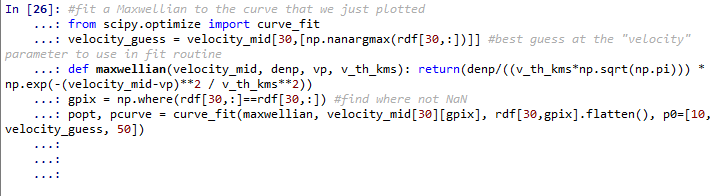
The fluid parameters that we will look at in the level-3 files are derived directly from these data that we just plotted. One of the ways to calculate those parameters is to fit the curve with a relevant functional form. Of particular interest to us is the Maxwellian ( <https://en.wikipedia.org/wiki/Maxwell%E2%80%93Boltzmann_distribution> ). But first we need to convert to the “reduced distribution function”. This is a way of displaying our data that removes any specific features that are due to how the instrument functions, and instead uses purely physical units that refer only to the plasma itself.



Let’s plot to see what that looks like:



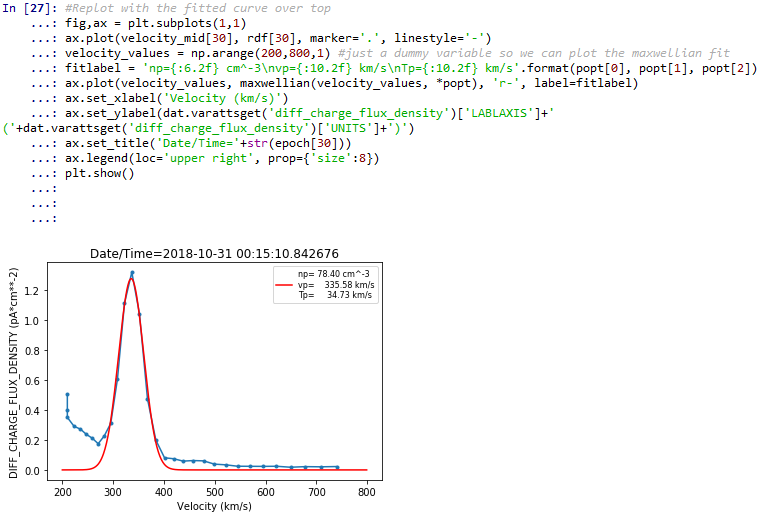
We can now fit the curve we just plotted with a Maxwellian, to determine the best-fit fluid parameters we can use to describe this distribution of plasma. First we need to import the “curve\_fit” function from the right package. Then we define the functional form that we’re going to use to fit. The function “maxwellian” is defined with four input parameters. The first (“velocity\_mid”) is the dependent variable, and the following 3 are the fluid parameters that we’re trying to fit. The “curve\_fit” function will vary those 3 parameters until it finds the ideal parameters to match the data that we have provided.



If you print “popt”, you can see what best-fit parameters were found.

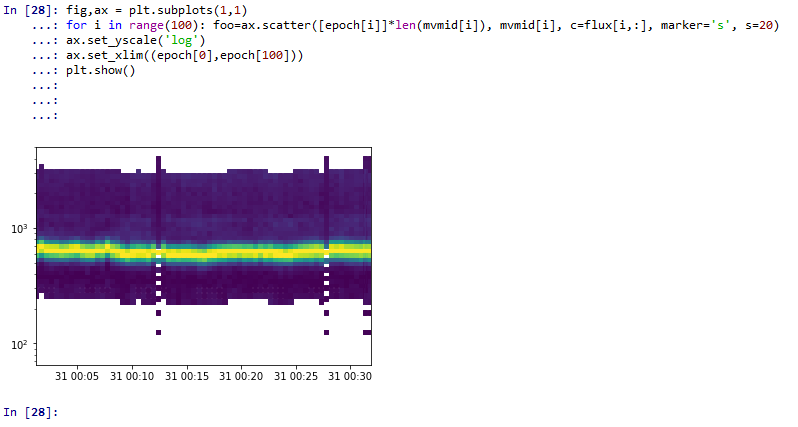


The “popt” variable now contains the ideal parameters for a Maxwellian fit. Density=78 cm^-3, velocity=336 km/s, and thermal speed = 35 km/s. Now we can re-plot the data we have along with the best Maxwellian fit.



In fact, the process we just did (fitting a Maxwellian to the L2 spectrum) is almost exactly what happens in the pipeline processing software that takes L2 spectra and creates the L3 files containing velocity, density, and temperature (of course that software pays much more attention to uncertainties, and is much more robust than what we just did).

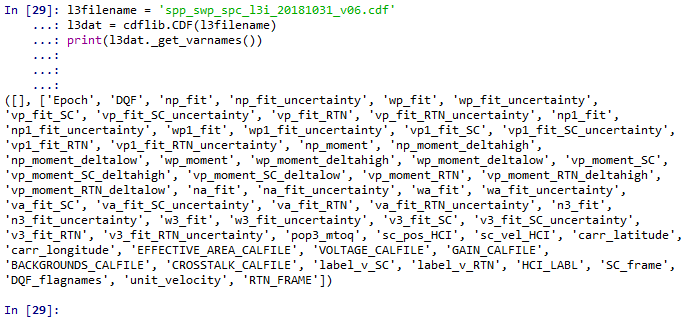
Before we move on to the L3 files, let’s make a figure that instead of showing a single spectrum can display many spectra at once, with color representing the flux, and time on the x-axis. We call this a “spectrogram” or a “dynamic spectrum”. There are better ways to do it than that shown below, but this is a quick and dirty way. Of course I’ve neglected to properly label the axes and to put a colorbar on there, which should most certainly be done for any plots that you make.



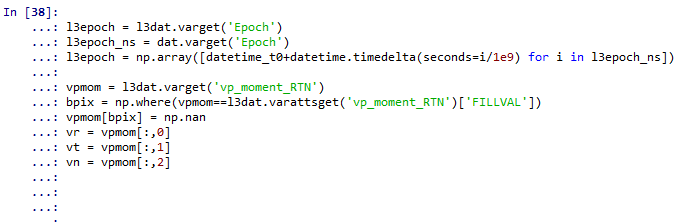
From this plot we can see how the distribution of the particles is changing over time. The highest currents (in yellow) are around 700 V (or so), and is pretty stable over the 27 seconds shown here.

# Level-3 Files

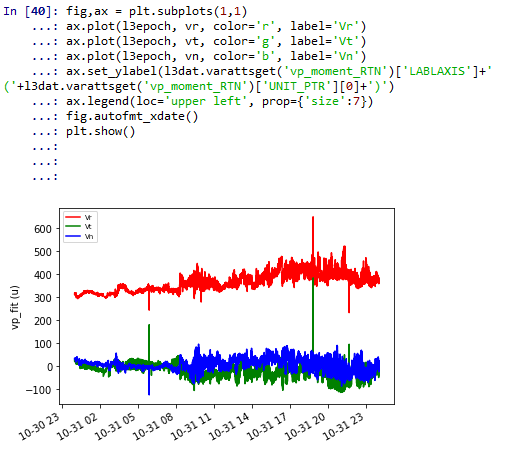
Now let’s move on to the level-3 file. Again, this is a CDF file, and we read it in the same way that we did before.



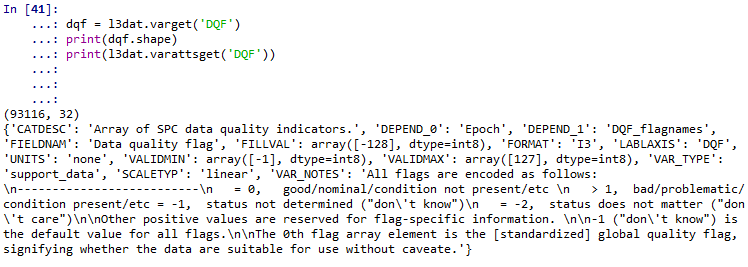
From the variable names, we can see that we are looking at the fluid parameters that have been calculated from the level-2 files. The “\*\_moment\_\*” and the “\*\_fit\_\*” parameters are from two different ways of calculating the bulk fluid parameters from the distributions. Note that the “RTN” and the “SC” suffixes refer to the coordinate system that the data are in. For now, let’s concentrate on the RTN variables. Let’s look at the vp\_moment\_RTN. First, grab all the variables we’ll want to work with.



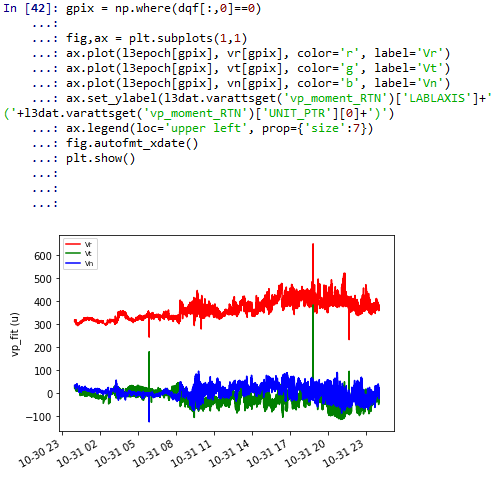
Now plot them up:



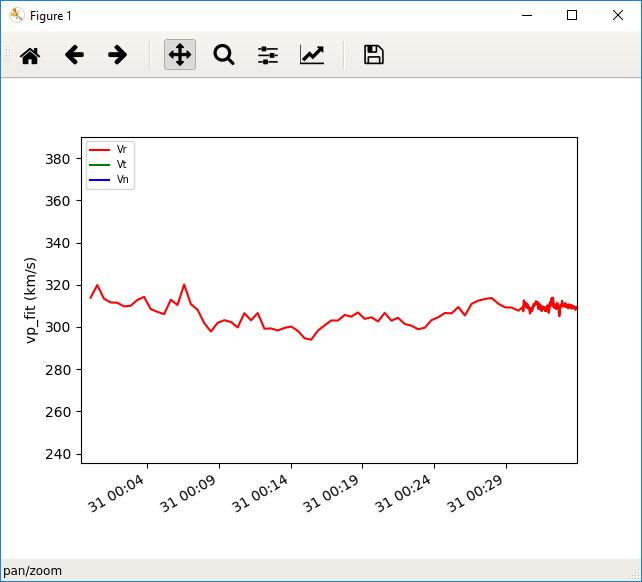
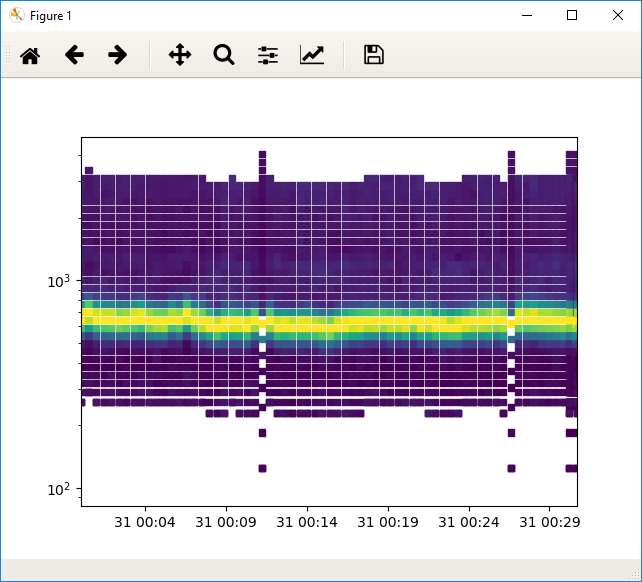
We should also use the data quality flag to make sure we’re only looking at data where we think we had relatively good calculations of the velocity. If you look at the attributes for the “DQF” variable, you can see a little bit about how it works.



So we can see that there are 32 flags for each data point, and when it is set to zero, that means the data are of high quality. Specifically, the 0th flag is the general flag. So we should make sure we are only using data points where the 0th flag is zero. We do that by again using the “where” function and using the resulting indices as a filter when we plot the variables.



And if we zoom into the very earliest data points that correspond to the dynamic spectrum we plotted earlier, we should be able to see the correspondence between the L2 data and the fluid parameters that are in the L3 data.

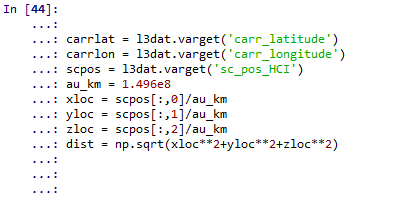
 

‘

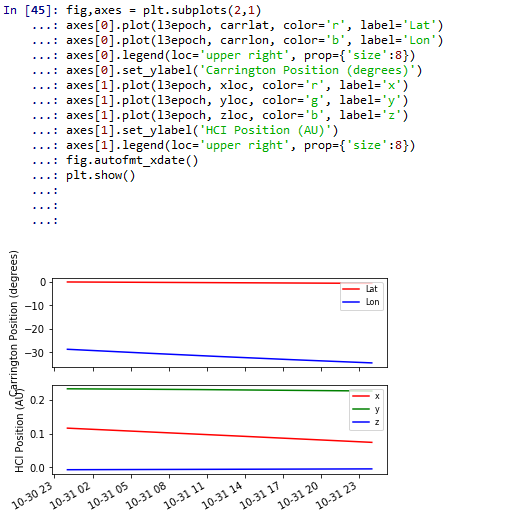
# Incorporating Ephemeris Data

Up until now we’ve plotted all of our data as a function of time (the “Epoch” variable). But an interesting facet of the Parker Solar Probe mission (and of particular importance for your project) is that the spacecraft is not just stationary, but rather rapidly visiting drastically different regions of interplanetary space as it orbits the Sun every 3-4 months.

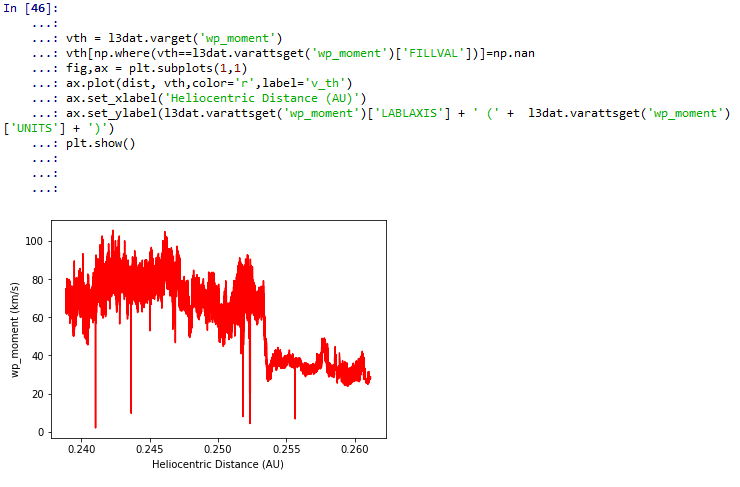
The information we need to determine where the spacecraft is, is included in the L3 files. We can pull them out of the CDF and make them into more useful variables



And then, for example, we can plot the location of the spacecraft for the day of data that we have already read in.:



And now that we have those positional variables to use, we can plot the SPC data as a function of these variables rather than of time. For example:



Of course, we can get fancy and maybe plot the location around the Sun, with color showing the speed, density, or temperature of the plasma at that location. That plot is essentially what was shown in the project outline document and copied here:

# 

As an exercise for the student, you should try and recreate these types of plots.