AC-6 Microburst Scale Size Project

Normalizing Procedure for the LEO Scale Size Distribution

Mykhaylo Shumko

Montana State University

Last Modified:

The coincident microburst catalogs can be easily binned by the AC-6 total separation (“Dist\_Total” variable) to create a histogram of the number of coincident microbursts observed as a function of separation. Due to the measurement apparatus, this distribution is highly correlated to the AC-6 orbit. Their separation, and the availability of good 10 Hz data contribute to this bias.

One simple approach to remove this bias is to normalize the number of coincident microbursts detected, by the exposure time of the two spacecraft as a function of separation. In simple terms, we are looking to find the occurrence rate of microbursts which corresponds to the “number of coincident microbursts observed at a given separation / how long the spacecraft were observing at that separation”. The exposure time was calculated using *norm.py’s* Hist1D and Hist2D classes which create normalization histograms for 1D and 2D distributions.

**Hist1D Description**

Hist1D loops over all days given a start/end date by the user. For each day, the class attempt to load 10 Hz data from both spacecraft. If successfully loaded, the data is filtered by time and data quality flag. The time filtering is done with the following nifty piece of code

*tA = date2num(self.ac6dataA['dateTime'])*

*tB = date2num(self.ac6dataB['dateTime'])*

*#* np.in1d returns a boolean array that correspond to indicies in tB

# that are also in tA. np.where will convert this mask array into

# an index array

*ind = np.where(np.in1d(tB, tA, assume\_unique=True))[0] # Find indicies*

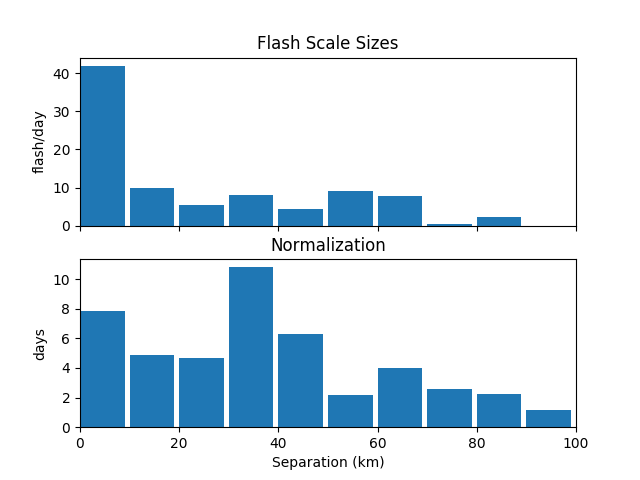
Then the data quality filtering is done by the following code

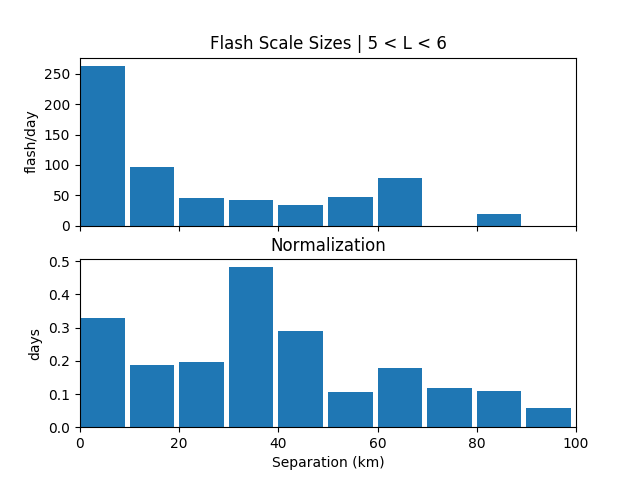
*indf = np.where(self.ac6dataB['flag'] == 0)[0]*

*ind = np.intersect1d(ind, indf)*

Lastly, for each day the self.ac6dataB['Dist\_Total'] data is filtered by the *ind* detections and histrogrammed by specified bins given by self.d. Then the day’s histogram is divided by 10 (to convert from data points to seconds) and added to the self.count His1D attribute. The self.d and self.count arrays are then saved to a csv file using the self.save\_data() method.

Optionally, the a user can supply *filterDict* which contains a *key:value* pairs for which the *key* corresponds to a AC-6 data key, and *value* is a 2d array with bounding values for that data key. For example, filterDict={'Lm\_OPQ':[3, 4]} will make a histogram that is filtered by time and data quality flag as explained above, and by the Lm\_OPQ data between 3 and 4. Figure 1 show examples of the coincident microburst scale size distributions that are normalized by the histogram in the bottom panel.

  
Figure Figure 1: Coincident Microburst Scale size for all L shells

  
Figure 1: : Coincident Microburst Scale size distribution for 5 < Lm\_OPQ < 6.

**Hist2D Description**

Hist2D is similar to the Hist1D class, except that it makes 2D normalizations. It is a child class of Hist1D and utilizes its loop\_data() method to loop over all of the days. The difference is in the histogram method which uses a 2D histrogram with two keys that are given to the Hist2D \_init\_ method to make the histograms. This data is saved to two files: one csv file contains the filter keys and all of the bin edge value combinations. The second file contains the count values in the same format. The way to read this data is to read in both files, and use each row in the bin edges file and the corresponding row in the count data.