Department of Commerce | National Oceanic & Atmospheric Administration | National Weather Service

Space Weather Prediction Center Controlled Document

Editors: Jeff Johnson, Mark Miesch, George Millward, Nathan Miles

Approver: Steven Hill

Revision: 1.4

01/09/2024

SWFO-L1 Temporal Spectral Binning ATBD

Table of Contents:

1 EXECUTIVE SUMMARY

The NOAA Space Weather Follow On satellite (SWFO-L1) is an operational NOAA satellite planned to launch in April 2025 on a SpaceX Falcon 9 rocket and to begin full operations later that year after it reaches its destination at the L1 point, roughly 1 million miles upstream of Earth in the direction of the Sun. The primary mission of SWFO-L1 is to act as a 'Sentinel', continuously measuring the incoming Solar wind and relaying this data back to Earth in real-time. Its location at the L1 point provides a lead time of fifteen minutes to an hour, allowing short-term forecasting of Space Weather conditions at Earth via a variety of Space Weather computer models, such as the SWPCs Geospace and WAM-IPE models. Output from such models is then used to predict how this upcoming Space Weather could affect a variety of critical technological systems, such as the integrity of the electrical power grid, long-distance radio communications, GPS positioning and the atmospheric drag on satellites operating in Low Earth Orbit (LEO).

There are three instruments on SWFO-L1 measuring in-situ Solar wind parameters: The Magnetometer (MAG), the Solar Wind Plasma Sensor (SWiPS) and the SupraThermal Ion Sensor (STIS). The purpose of this document is to describe the averaging and binning algorithms to be used to produce operational time series products at Level 3 and their theoretical basis. This document provides the theoretical basis for the algorithms chosen and is intended to be a reference for implementing the algorithms into software.

The Level 2 data products for the MAG, SWiPS and STIS instruments on SWFO-L1 all have the same nature in that they are scalar time series, each with their own intrinsic cadence. The algorithms described here then resample each of these time series to produce cadences of 1 min that are of greatest relevance for SWPC forecasters and stakeholders. A time series with a cadence of 1 second is also produced for the MAG instrument. For the STIS instrument, a related algorithm is provided that resamples the spectral energy channels into a desired sequence of spectral bins. In all cases, the resampled data is then referred to as Level 3 data. Robust statistics are used to produce reliable results even in the presence of bad data points, which manifest as outliers. Mathematical and pseudo code descriptions are given for the algorithms.

For the MAG data, the averaging algorithm converts 64 Hz resolution L2 data for all components into 1-second and 1-minute averaged data using the HL and iterative Huber M-Estimator techniques respectively (section 3.2.1). Both algorithms also support data at 8Hz resolution, an optional mode on the instrument, though this is not expected to be used routinely in operations.

For the SWiPS instrument, the algorithm resamples Level 2 data with a potentially irregular 1-minute cadence to a regular 1-min cadence that is aligned with 1-min timestamps in UTC. The technique employed here is a straight linear interpolation (section 3.2.2).

For the STIS instrument, the time averaging algorithm converts 0.5 Hz (2 second) resolution Level 1b and L2 data into 1 min averaged data utilizing an iterative Huber M-Estimator technique (section 3.2.3)

Example test runs using proxy data are shown and compared to results from currently implemented algorithms, with comparisons showing excellent agreement (section 4.2). Pre- and post-launch calibration/validation plans are presented (section 5.3) along with current algorithm limitations (section 6).

Source code for the chosen algorithms (notably the HL and Huber M-estimators) are given in IDL, Python and Java in section 8.

2 AVERAGING OVERVIEW

2.1 Products Generated

The algorithms described here use robust statistical methods to estimate time averages for MAG, SWiPS, and STIS data. In the technical literature, the average or mean value of a sample of data points is often referred to as the **location** of the sample. Furthermore, the standard deviation of the sample is often referred to as the **scale** of the sample.

Thus, this section surveys robust averaging methods that have been designed to reliably estimate the location of a data sample in the presence of outliers. In order to estimate the location, some of the methods also require an estimate of the scale.

The output is averaged time series data (1 minute for MAG, SWiPS, and STIS, and 1 second for MAG). Furthermore, in addition to time averaging, the Level 3 STIS data will include two distinct time series binned over different energy channels. The first L3 STIS data product will coincide with ACE/EPAM energy channels while the second will approximate the higher-resolution √2 logarithmic binning used by NASA's IMAP mission (see Sections 3.1.4 and 3.2.3 for details). The location parameters (classically, time averaged data) will be used extensively by the space physics community and space weather forecasters.

There are many reasons to average data. Averaging data minimizes the effect of random measurement errors and also allows quality flags to be assigned to a given time period. There are also many different ways in which data can be averaged. For previous GOES instrument averages, a sliding boxcar average with zero overlap has been used. This is a simple, well-understood method for calculating an average; however, as is the case with many of the classical statistical methods based on linear models or normality assumptions, such averaging methods can be vulnerable to gross errors. Thus for the SWFO-L1 time series data averages, we employ more robust statistical methods to compute an estimate for a location parameter. This document discusses several robust statistical methods and outlines the methods suggested for use with the SWFO-L1 MAG, SWiPS, and STIS data.

2.2 Performance Requirements

Robust statistical methods are statistics with good performance for data drawn from a wide range of different probability distributions, especially distributions that are not normally distributed. Robust statistical methods have been developed for many common problems, such as estimating location, scale, and regression parameters. One motivation is to produce statistical methods that are not unduly affected by outliers, defined as data points with extreme values that are likely spurious. Another motivation is to provide methods with good performance when there are small departures from parametric distributions. For example, robust methods work well for mixtures of two normal distributions with different standard-deviations.

SWFO-L1 Averaging Algorithm Requirements:

- Needs to be able to deal with outliers
- Needs to handle small data sets in the averages as low as 8 data points
- Needs to run in real-time and not require excessive computational time
- Needs to be straightforward yet robust

The above requirements support the use of robust statistical methods in computing the time series averages for SWFO-L1.

3 ALGORITHM DESCRIPTION

3.1 Algorithm Overview

3.1.1 Physics of the Problem

Data averaging should be a simple process. However, the type of data to be averaged and the intended use of the averaged data can complicate the justifications for using one particular averaging technique over another. For example, as a result of the averaging technique, frequency analysis of averaged data can be adversely affected by aliasing and frequency overlapping. Generally, space physics data, such as GOES MAG data, has utilized a boxcar averaging technique with the time set to the beginning of the minute. However, potential improvements to the averaging algorithm include 1) robust location estimators to account for measurement error, and 2) weighted averages to minimize aliasing.

3.1.2 Mathematical Description: Robust Averaging Methods

Let $x_1, \cdots, x_n$ be independent random variables from a common distribution; such variables are commonly called iid or i.i.d., for *independent and identically distributed*. To begin, let us look at the origin of the theory of location estimation, as described by Gauss. Gauss used the method of least squares to find a location parameter; that is, Gauss minimized the sum of the squares of the differences between observed and computed values. In other words, Gauss minimized the expression

$$\varepsilon^2 = \sum_{i=1}^{n} \left(x_i - T\right)^2, \tag{1}$$

or, equivalently, after differentiating with respect to T, he solved the expression

$$\sum_{i=1}^{n} \left(x_i - T\right) = 0, \tag{2}$$

which immediately yields the sample mean

$$T \equiv \mu = \frac{1}{n}\sum_{i=1}^{n} x_i. \quad (3)$$

But what happens to the sample mean, μ, if there are outliers or if the base distribution is contaminated? An outlier, or an observation that is numerically distant from the rest of the data, can occur in any distribution. They are often indicative of measurement error or signal that the population has a heavy-tailed distribution. Statistical estimators that are capable of dealing with outliers are said to be robust.

One of the basic tools used to describe and measure robustness is the breakdown point. The breakdown point of an estimator is the proportion of incorrect, or arbitrarily large, observations an estimator can handle before giving an arbitrarily large result. The sample mean has a breakdown point of zero, since, as can be seen from Eq. 3, if even one point is moved to ∞, then $\mu \rightarrow \infty$. The higher the breakdown point of an estimator, the more robust it is. Intuitively, we can understand that a breakdown point cannot exceed 50% because if more than half of the observations are contaminated, it is not possible to distinguish between the underlying distribution and the contaminating distribution. Therefore, the maximum breakdown point is 50%. An example of such a robust location estimate is the median.

Unfortunately, while the median is a highly resistant statistic, it is not an efficient statistic. A statistic is said to have a high asymptotic efficiency relative to another estimator if its variance is smaller, which typically means that its mean squared error is smaller and it produces a smaller confidence interval. The median's asymptotic efficiency relative to the normal distribution is 64% (*Kenney and Keeping*, 1962). The efficiency of an estimator may change significantly if the underlying distribution changes. This is one of the motivations of robust statistics. For example, the sample mean is an efficient estimator of the population mean of a normal distribution, but is an inefficient estimator of a contaminated distribution or a non-normal distribution. Hence, while efficiency is a desirable quality of an estimator, it must be weighed against other considerations. In robust statistics, more importance is placed on robustness and applicability to a wide variety of distributions, rather than efficiency on a single distribution.

Since 1960, many theoretical efforts have been devoted to developing robust location estimators. It is generally accepted that screening the data to remove outliers and then to apply classical procedures is not a simple and good way to proceed. First of all, it can be difficult to single out the outliers. Second, in place of rejecting an observation, it would be better to down-weight uncertain observations, although we may wish to reject completely wrong observations. Moreover, rejecting outliers reduces the sample size, which could affect the distribution, and variances could be underestimated from the cleaned data. Finally, empirical evidence shows that robust procedures behave better than techniques based on the rejection of outliers.

M-Estimators
Huber (1963) was a seminal paper in robust statistics. In it, Huber suggested that instead of minimizing Eq. 1, the sum of the square of the residuals, $r_i \equiv x_i - T$, one could obtain a more robust solution by minimizing a function of the normalized residuals,

$$\varepsilon = \sum_i \rho\left(\hat{r}_i\right), \quad (4)$$

where $\hat{r}_i \equiv \frac{\left(x_i - T\right)}{S}$ and S is an unknown scaling parameter. Once again, this expression is differentiated with respect to T, leading to the following equation,

$$\sum_i \psi\left(\hat{r}_i\right)\frac{d\hat{r}_i}{dT} = 0, \quad (5)$$

where

$$\psi\left(\hat{r}_i\right) \equiv \frac{\partial\rho}{\partial\hat{r}_i} \quad (6)$$

is called the influence function. Solutions of Eq. 5 form a broad class of estimators called **maximum likelihood type estimators**, or **M-estimators** for short. Some important properties of a robust influence function, ψ, are that it be bounded for large residuals and linear near zero, such that the M-estimator is asymptotically efficient relative to a Gaussian distribution. Note that Gauss' approach to location estimation introduced at the beginning of Sect. 3.1.2 can easily be recast using the language of Huber; inspection of Eq. 2 shows that $\psi = r$, which is an unbounded influence function. Thus, Gauss' approach forms a non-robust subset of the M-estimators, known as a least squares estimator.

Although there are several common influence functions, many people consider the original *Huber* function so satisfactory that it can be recommended for almost all situations; very rarely has it been found to be inferior to other influence functions (*Zhang*, 1997).

- *Huber Function:*

$$\psi = \begin{cases} r & |r| \leq k \\ k\, sgn(r) & |r| \geq k \end{cases} \tag{7}$$

A 95% asymptotic efficiency relative to the Gaussian distribution is obtained with the tuning constant $k = 1.345$. Here *sgn(r)* is the *sign* function or *signum function*, defined as follows:

$$sgn(x) = \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0. \end{cases} \tag{8}$$

One can use the influence function to define a weight function,

$$w\left(\hat{r}_i\right) \equiv \frac{\psi\left(\hat{r}_i\right)}{\hat{r}_i}, \tag{9}$$

in which case, Eq. 5 becomes

$$\sum_i w\left(\hat{r}_i\right)\, \hat{r}_i = 0, \tag{10}$$

or

$$T = \frac{\sum_i w\left(\hat{r}_i\right) x_i}{\sum_i w\left(\hat{r}_i\right)} \tag{11}$$

Note that all of these methods require a robust estimate of the scale S. A common choice is the median of absolute deviations (MAD), defined by

$$MAD(x) = c\, median(|x - median(x)|) \tag{12}$$

where c is a constant that can be optimized to maximize efficiency and minimize bias. A value of 1.4826 is optimal for a Gaussian distribution (Croux & Dehon 2013). Using a robust estimate of scale, Eq. 10 can be solved using an iterative re-weighting algorithm such as Newton-Raphson.

For a Normal distribution, the standard deviation $\sigma = 1.4826 \times MAD$. So, for all points within one standard deviation of the mean, the Huber M-Estimator gives identical results to a simple boxcar average. But it de-emphasizes rarer, more extreme points in the tails of the distribution that can artificially skew the average when the sample size is limited (even if there are no invalid outliers).

Huber (1981) described an alternative approach in which the location and scale can be estimated simultaneously, which is often referred to in the literature as "Huber's Proposal 2" and which is sometimes found in statistics packages such as python *statsmodels*. However, this approach is less robust, with a lower breakdown value, than minimization with respect to location using the MAD estimate for scale (Rousseeuw & Verboven 2002). When subject to the test data sets described in Section 4, we found that Huber's proposal 2 algorithm for joint estimation of location and scale often failed to converge. This is in contrast to the location estimator with a MAD scale estimate, which never failed to converge for all windows considered.

The Hodges-Lehmann (HL) Estimator
This is a robust and non-parametric estimator of a population's location based on rank tests (*Hodges and Lehmann*, 1963). It has an asymptotic efficiency relative to a Gaussian distribution of 0.95 (*Hodges and Lehmann*, 1963) and a breakdown point of 29%.

Given a set of n iid variables, $x_1, \cdots, x_n$,

- form $n\frac{(n+1)}{2}$ independent pairs $\left(x_i, x_j\right)$, $i \leq j$, including the pair of each item taken twice
- compute the average of each pair
- find the median of all the averages.

This median value is defined to be the Hodges-Lehmann (HL) estimator of location. According to *Bickel* (1965), the HL estimator is the preferred estimator when the degree of contamination or underlying distribution is not known with great precision.

An advantage of the HL estimator relative to M-estimators is that it does not require any parameters or any independent scale estimate. Furthermore, it does not require the convergence of an iterative procedure in order to give a reliable result.

Disadvantages include its lower breakdown point of 29% and its greater computational expense at large sample sizes n, scaling as n^2 (see Section 4).

3.1.3 Mathematical Description: STIS Spectral Binning

By nature, the STIS data are two dimensional. The first dimension is time and the second dimension is the particle energy. In other words, each data point represents the flux of particles in a particular time interval that fall within a particular energy range. So, in addition to averaging over time, we must also consider the distribution of energy channels. This is referred to as spectral binning. In order to properly capture the decreasing particle flux with increasing energy, each spectral bin is chosen to span approximately the same logarithmic range in energy. The number of energy bins is referred to as the spectral resolution. The STIS spectral resolution is programmable but is not expected to change after initial operational capability (IOC).

As described in Section 3.2.4, the full resolution of the L1b STIS data spans 37 logarithmic bins in ion energy and 21 logarithmic bins in electron energy (Table 2). In addition, the L2 STIS data will be produced with ACE/EPAM spectral binning (Table 1), which consists of 8 logarithmic bins in ion energy and 4 logarithmic bins in electron energy.

Though the ACE/EPAM spectral binning is desirable for data continuity, it is beneficial to provide higher spectral resolution at L3 for potential forecast applications (Section 3.2.4). This enhances the scientific information in the L3 product while mitigating the measurement and detector noise in the full resolution L2 product. In this sense, it is a middle ground between the full-resolution L2 data product and the low-resolution L2 data product.

A straightforward way to achieve this intermediate L3 resolution is to create larger bins by summing two adjacent full-resolution bins as illustrated in Fig. 5. The red curve on the right shows a fiducial power-law spectrum of particle counts with an exponent of -2 at the full ion spectral resolution listed in Table 2. Composite logarithmic bins are then created by combining two adjacent bins from the red curve. Gray vertical lines mark the edges of these composite bins and blue points lie at their center. The count values in each composite bin are obtained by summing the counts in the two full-resolution bins (red points) that lie between each pair of vertical lines.

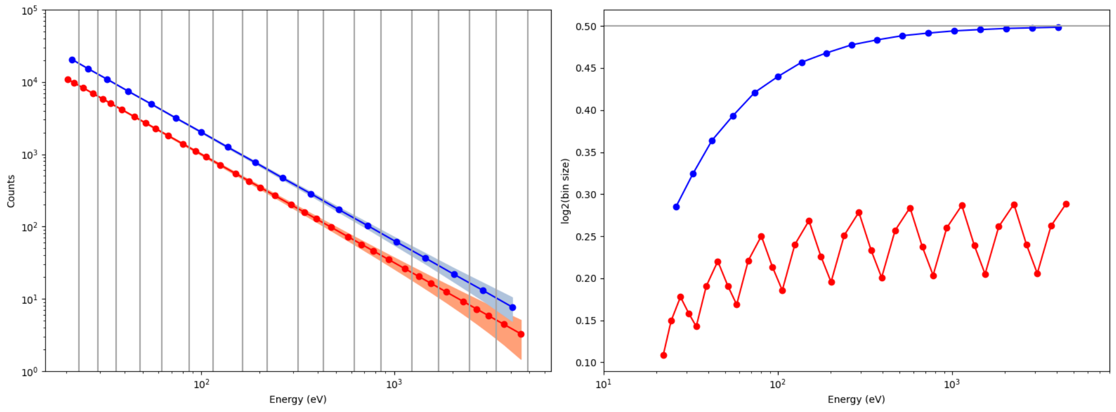

Figure 4. Spectral rebinning by summing adjacent bins. (left) A fiducial power-law count spectrum is shown in red using the full ion resolution represented in Table 2, excluding the last bin. The blue curve is generated by summing the values in adjacent bins of the red curve, creating the larger logarithmic bins delineated by the vertical gray lines. Points indicate logarithmic bin centers. Shaded areas indicate √N Poisson error bars. (right) base-two logarithmic bin sizes for the full spectral resolution (red) and the rebinned resolution (blue). A √2 bin size is shown for reference by the horizontal gray line.

The boundary between two adjacent logarithmic bins i and i+1 is given by the geometric mean of the center values, $\sqrt{x_i x_{i+1}}$. Similarly, the center of a bin is given by the geometric mean of the two edge values. So, when combining two bins centered on x_i and x_{i+1}, the center of the composite bin, x' is given by:

$$x' = \sqrt{\sqrt{x_{i-1}x_i}\,\sqrt{x_{i+1}x_{i+2}}} = (x_{i-1}x_i x_{i+1}x_{i+2})^{1/4} \qquad (15)$$

This is the relationship between the composite bin centers (blue points) and original bin centers (red points) in Figure 4. In this way, the 36 original bins are resampled into 18 composite bins.

Note that this simple summation of bins neglects leakage in the detector that may give rise to double-counting of particle events. But this is expected to be negligible for STIS. Note also that it preserves the power-law form in Fig. 5 and reduces the noise somewhat, as illustrated here with √N Poisson error bars. Furthermore, the integrated flux, obtained by summing each of the two curves in Figure 4 (left) over bins, is unchanged by the rebinning. Note that if the curves in Figure 4 (left) were normalized by the bin spacing then they would coincide. This is how particle fluxes are often portrayed.

A further benefit of the rebinning is that, for the STIS L2 spectral bins listed in Table 2, the composite (blue) bins smoothly approach √2 logarithmic spacing toward the upper range of energies. This resolution is comparable to the real-time data stream that will be provided by NASA's Interstellar Mapping and Acceleration Probe (IMAP) mission to be launched to L1 in 2025 (McComas et al 2018). The CODICE-Hi instrument on IMAP measures suprathermal ion fluxes in the range 0.03 - 5 MeV/nuc and electron fluxes in the range 20-600 keV, overlapping with STIS. This will provide redundancy in the event that the STIS data stream becomes unavailable, as well as an opportunity for post-launch calibration.

3.2 Data Processing Algorithm

3.2.1 Magnetometer (MAG)

For MAG we will use non-overlapping windows for the 1 second and 1 minute averages. For consistency with DSCOVR data, we will define the timestamp to be the beginning of each time window. The cadence of the Level 2 magnetometer data is 64 Hz. Thus, for the 1-second Level 3 averages we expect 64 data points in each averaging window. Furthermore, the 1-minute Level 3 averages will be computed directly from the 64 Hz Level 2 data. So, the 1-minute averages will contain 3840 data points in each (1-minute) averaging window.

The MAG instrument is also capable of delivering data at an 8 Hz cadence as opposed to an 64 Hz cadence. This is not expected to be used for SWFO-L1 operations but the L3 processing algorithm must be prepared to handle this data stream if it is enabled.

For the 1-second MAG averages, we will use the HL estimator. This provides a robust, efficient location estimate that has a proven heritage; this is the algorithm that has been used to produce Level 3 data products for the MAG instrument on the DSCOVR spacecraft from 2015 to the present. It does not require iterative convergence to produce a reliable result and its breakdown point of 29% admits up to 2 bad data points in each 64-point averaging window. The HL estimator will also be used for the 1-second averages if the 8 Hz MAG data stream is enabled.

For the 1-minute MAG average, we will use an M-Estimator with a Huber influence function (e.q. 7), since it exhibits a higher breakdown point of 50% with no loss of efficiency relative to HL. Based on experience with DSCOVR operations, the computation time is expected to be easily fast enough. The HL method will be used as a backup in case the M-Estimator fails to converge (see Section 4). But, this is expected to be very rare.

The use of two different estimators for the 1-second and 1-minute time averages maximizes computational efficiency and leverages the strengths of the two methods. Between them, they are expected to give reliable results regardless of the underlying distribution of the data points. And, they can be used as a cross-check on one another both in pre-launch testing and in post-launch real-time operations.

Magnetometer Processing Outline

1. Read in 64 Hz Level 2 MAG data.
2. Use HL estimator with a non-overlapping averaging window to compute the 1-second average from 64 Hz Level 2 MAG data. Define timestamps at the beginning of each 1-second time window.
3. Use iterative Huber M-Estimator with a non-overlapping averaging window to compute the 1-minute average from 64 Hz Level 2 MAG data. Define timestamps at the beginning of each 1-minute time window.
4. Output 1-sec and 1-min averages at each time along with the number of data points in each window.

3.2.2 Solar Wind Plasma Sensor (SWiPS)

For the SWiPS instrument, the cadence of the Level 2 data is expected to be approximately 1 minute, which is the time it takes to complete one sweep of the field of view and the voltage steps. Though this nominally meets the desired 1 minute cadence, the intervals between sweeps may not be exactly one minute. But many data products, applications, and models require a regular time series.

So, the main goal of the Level 3 processing of SWiPS Level 2 data is to ensure that the time series is sampled on a regular 1-min time cadence that is aligned with UTC. We will use linear interpolation to resample the Level 2 SWiPS data onto a regular 1-minute temporal cadence at Level 3.

SWiPS Processing Outline

1. Read in 1-min Level 2 SWiPS data.
2. Interpolate the Level 2 data onto a regular 1-min grid, aligned with UTC, using linear interpolation.
3. Output interpolated L3 time series with time grid.

3.2.3 Suprathermal Ion Sensor (STIS)

The cadence of the Level 1b and Level 2 STIS data is programmable to be between 1 and 300 seconds; the expected cadence based on available data rate is 2 seconds. So, with a non-overlapping averaging window (Section 3.1.3) we expect 30 data points in each of the 1-minute averaging windows that will be used for the 1-minute Level 3 averages.

As described in Sections 3.1.2 and 3.2.1, the iterative Huber M-Estimator is the preferred method when the number of data points in a window, n, is relatively large. So, we will use the Huber M-Estimator for computing the 1-min Level 3 STIS data, as for the 1-min Level 3 MAG data.

Level 1b data represent the full temporal and spectral resolution of the STIS instrument. At Level 2, the energy bins are resampled to approximately match the spectral bins used for the EPAM instrument on the ACE spacecraft (Gold et al. 1998), which is used now for SWPC forecast products. Thus, the ACE/EPAM spectral binning provides data continuity and facilitates integration with existing SWPC forecasting applications and models that are calibrated to this distribution of energy channels. The ACE/EPAM energy binning is summarized in Table 1. The nominal L2 channel midpoint energies are summarized in Table 2, for those channels whose response functions are well-behaved. (These energies will change when the flight article is calibrated.)

Though the ACE/EPAM binning at a 1-min cadence is an essential Level 3 product, it is also of interest to the forecasting and scientific community to have access to a product with higher spectral resolution. This offers enhanced insight that could be exploited in space weather forecasting. For example, coronal mass ejections (CMEs) are among the most important space weather events in terms of socioeconomic impact. As a CME travels through interplanetary space from the Sun to the Earth, it may accelerate energetic particles that can be observed with STIS. These particles arrive before the shock wave of the interplanetary CME itself. Stronger shock waves (arising from powerful CMEs) are expected to produce relatively more particles at higher energies (a so-called hard spectrum). Thus, detailed information about the spectrum (high spectral resolution) could potentially be exploited to develop new forecast products or enhance existing products that warn us when a high-impact space weather event is imminent.

As demonstrated in Section 3.1.4, re-binning the (nominal) full resolution L2 data for STIS by a factor of two gives logarithmic bin spacings that smoothly approach a value of √2 at the higher end of the energy range. And this is comparable to the backup real-time data stream we expect to receive from NASA's IMAP mission, which provides an opportunity for redundancy and calibration.

So, Level 3 data products will be provided for both the ACE/EPAM spectral binning (derived from Level 2 products) and for the energy bins listed in Table 3 (derived from Level 1b products).

STIS Processing Outline

1. Read in 0.5 Hz Level 1b and 0.5 Hz Level 2 STIS data.
2. Rebin L1b data to the spectral bins listed in Table 3 by combining bins as described in Section 3.1.4.
3. Use iterative Huber M-Estimator with non-overlapping averaging windows to compute the 1-minute average of rebinned L1b data from Step 2 (Table 3 spectral binning). Define timestamps at the beginning of each 1-minute time window.
4. Use iterative Huber M-Estimator with non-overlapping averaging windows to compute the 1-minute average of Level 2 data (ACE/EPAM spectral binning). Define timestamps at the beginning of each 1-minute time window.
5. Output 1-min averages at Table 3 binning and 1-min averages at ACE/EPAM binning (Table 1) along with the number of data points in each averaging window.

Table 1: ACE/EPAM Spectral Binning (Gold et al. 1998). The ion channels are from the LEMS120 telescope. The highest-energy electron channel is DE4 from the "deflected electron" detector. The other electron channels are from the LEFS60 telescope.

Ion Channels (keV)	47-68	68-115	115-195	195-321	310-580	587-1060	1060-1900	1900-4800
Electron Channels (keV)	45-62	62-102	102-175	175-315	–	–	–	–

Table 2: Nominal STIS channel midpoint energies, from 'swfo_stis_sci_level_1b.pro', 11 August 2022 version. These are the built-in (non-LUT) energies.

Ion Channel Energies (keV)	20.4	22.0	24.4	27.6	30.8	34.0	38.8	45.2
	51.6	58.0	67.6	80.4	93.2	106.0	125.2	150.8
	176.4	202.0	240.4	291.6	342.8	394.0	470.8	573.2
	675.6	778.0	931.6	1136.4	1341.2	1546.0	1853.2	2262.8
	2672.4	3082.0	3696.4	4515.6	5334.8	–	–	–

Electron Channel Energies (keV)	20.4	22.0	24.4	27.6	30.8	34.0	38.8	45.2
	51.6	58.0	67.6	80.4	93.2	106.0	125.2	150.8
	176.4	202.0	240.4	291.6	342.8	–	–	–

Table 3: Nominal midpoint energies for L3 data products, obtained by rebinning the energy channels in Table 2 as described in Section 3.1.4.

Ion Channel Energies (keV)	21.3	26.0	32.5	41.9	55.0	73.6	99.9	137.1
	189.6	264.0	369.0	517.7	727.9	1025.2	1445.6	2040.2
	2881.0	5334,8	–	–	–	–	–	–
Electron Channel Energies (keV)	21.3	26.0	32.5	41.9	55.0	73.6	99.9	137.1
	189.6	264.0	342.8	–	–	–	–	–

The availability of this conventional option is necessary during the post-launch commissioning period. The two averaging methods will be compared in order to compare means and fluctuations in the data.

For particle counters like STIS, Poisson statistics are significant at low flux levels, high energy resolution and rapid cadence. These conditions will hold for much of the L1b data. Time-averaging of Poisson statistics-dominated data is perhaps better seen as accumulation of more counts, thereby reducing the statistical error bars. Preserving more extreme values, as well as zeros, that occur naturally as part of a Poisson process is necessary for full benefits of count accumulation to be realized.

The Huber M-estimator assumes a Gaussian distribution in determining how data are weighted in the calculation of the averages. A Poisson distribution departs from a Gaussian distribution at lower means of the distribution. An estimator based on a Gaussian assumption may not be appropriate for asymmetric distributions like a Poisson distribution (Elsaied & Fried 2016). Because the Huber M-estimator deemphasizes more extreme though valid points in the tail of the Poisson distribution (section 3.1.2), it may bias low the average of the STIS fluxes. The degree to which this happens with the actual STIS data will have to be determined during commissioning, through the comparison of the two averaging methods.

4 TEST DATA SETS

4.1 Synthetic and Proxy Data Sets for Testing

For testing and validation purposes, we will use both synthesized data sets where we can control the sample size, amount of noise and number of outliers, as well as real solar wind and energetic particle data obtained from the DSCOVR and ACE spacecraft.

As a proxy for the 64Hz MAG data on SWFO-L1, we will use the 50Hz Level 1 magnetometer data from the MAG instrument on DSCOVR. The data for the Faraday Cup (FC) instrument on DSCOVR has a variable time cadence, ranging from 5 to 15 seconds (4 to 12 data points per 1-min average). So this provides an even greater test of the robustness of the averaging algorithms to data gaps, outliers, and sample size, the latter of which can range from 1 to 25 data points. This data will be interpolated to an irregular 1-min cadence to provide proxy data for SWiPS.

Level 1 MAG and FC data from DSCOVR can be obtained from the NCEI DSCOVR data portal at https://www.ngdc.noaa.gov/dscovr/portal/. Each file from the archive is provided in NetCDF3 format and contains 1 day's worth of data. After resampling this data to SWFO-L1 cadences, we will generate NetCDF4 test files for MAG and SWiPS that each span 6 hours.

For the STIS proxy data, we will use Level 2 data from the ACE/EPAM instrument which is provided at a 12-second cadence from the ACE Science Center hosted by CalTech: http://www.srl.caltech.edu/ACE/ASC/level2/lvl2DATA_EPAM.html. These are provided as HDF files, each spanning 27 days. We will resample these to a 15 second cadence appropriate for STIS and write 6 hours of output to each test file.

Artificial (simulated) data will also be provided for MAG, SWiPS, and STIS with the appropriate cadences and random noise. Simulated and real test data sets will include data gaps and outliers, as well as valid data. Test cases with more than 50% contaminated values can be included to illustrate the breakdown behavior that occurs with high levels of data contamination.

Format of MAG, SWiPS, and STIS Test Data Sets

- Test data will be at the same time resolution as the expected SWiPS and STIS real-time data (irregular 1-min intervals for SWiPS data, 0.5 Hz for STIS)
- Test data for MAG at 50Hz will be at roughly the same time resolution as the expected SWFO-L1 data rate of 64Hz.
- Test data files will include metadata headers that are consistent with the metadata headers expected from the Level 2 data for each instrument.
- Test data will include data gaps and outliers as well as valid data.
- Test data files will be provided as NetCDF files.
- For each test data file, a file containing the expected output will also be provided.

Like the data itself, the expected results will be provided as NetCDF4 files. So the results can be compared by reading them into a python script (or equivalent) or through command-line tools such as nccmp. (https://gitlab.com/remikz/nccmp).

4.2 Assessment of Algorithms using Test Data Sets

In order to assess the accuracy, robustness, and efficiency of the algorithms discussed in Sections 3.1 and 3.2 we have implemented preliminary versions of the recommended algorithms in python. And, we have subjected these algorithms to proxy data sets. In this section we describe the results of this assessment, which form the basis of the algorithms that have been recommended in Section 3.2. Furthermore, we provide code samples to illustrate how these algorithms can be implemented. All code used in this assessment is publicly available in the following github repository (Robust Averaging of Time Series):

https://github.com/mmiesch/RATS

4.2.1 Test Data and Algorithm Implementation

Two types of proxy data have been considered. The first is 50 Hz Level 1 data from the MAG instrument on DSCOVR, obtained through the National Centers for Environmental Information (NCEI) DSCOVR data portal as described in Section 4.1. One archive file contains data from 1 day of measurements at 50 Hz. The file we use here is from 5 Feb, 2022 and contains 4317541 data points.

The second type of test data consists of artificial (simulated) trigonometric functions (sinusoids and arctangents) superposed with random noise. In order to emphasize data outliers, a Cauchy distribution is chosen for the noise. A sample Python code segment to produce such artificial data is given in Section 8.2.1

4.2.2 Algorithm Performance

Figure 5 shows the recommended estimators, namely the Hodges-Lehmann (HL) and Huber M-estimator, applied to a subsample of the 50Hz DSCOVR MAG data (green and blue lines respectively). A boxcar average is also shown for comparison (red line). For this data set the three estimators agree very closely; The red, green and blue lines essentially lie on top of each other, and are offset slightly on the plot, for clarity. This demonstrates that the robust estimators give nearly the same answer as the sample mean for data that is not contaminated by many outliers.
For this 50Hz data, the HL estimator computes roughly 6000 times faster than real-time while the Huber M-estimator is 300 times faster than real-time.

The full code to create Figure 5 in IDL, including IDL implementations of both the HL and Huber estimators, is given in section 8.1

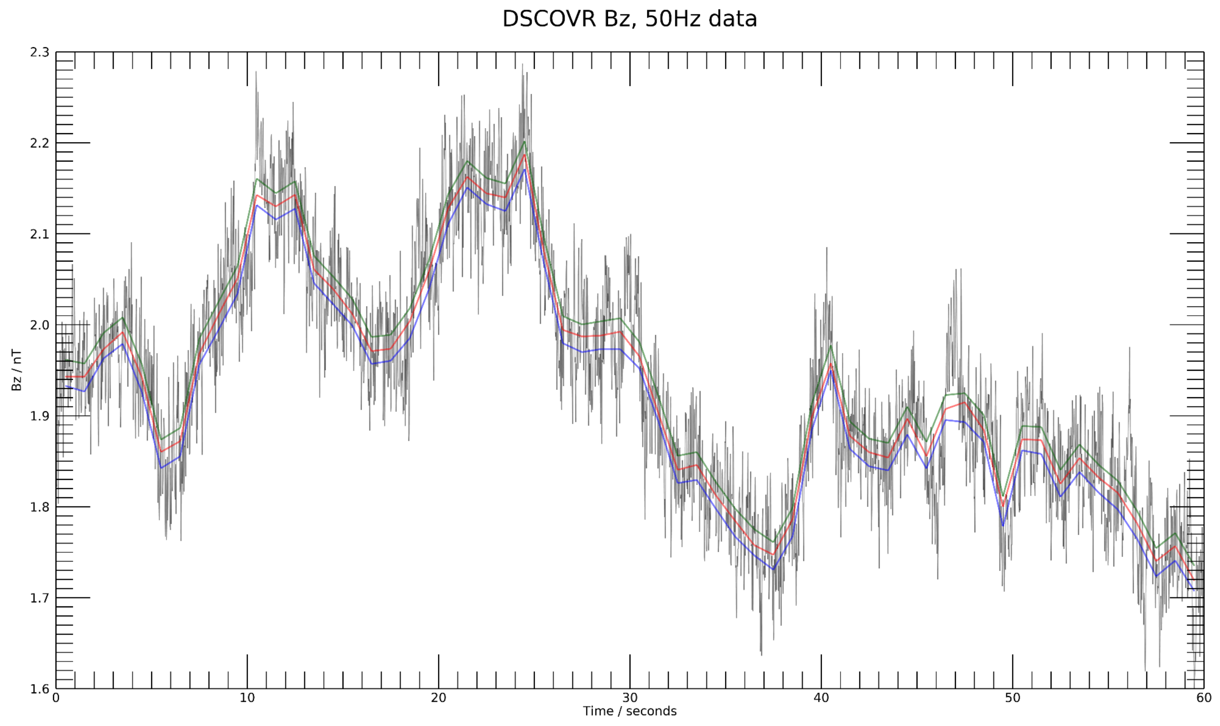

Figure 5: Location estimators applied to a typical subset of DSCOVR 50Hz MAG data (Bz parameter). The black line shows 60 seconds of the original 50Hz data, a boxcar average (red), the Hodges-Lehmann estimator (green, offset +0.015nT for clarity), and the Huber M-estimator (blue, offset -0.015nT for clarity).

Figure 6 shows a longer, 1 hour, section of 50Hz DSCOVR data. The black line shows the original 50Hz data. The blue line shows 1 second averages, using the Huber M-estimator technique, offset by +6nT for clarity. The green line shows 1 minute averages, again using the Huber M-estimator (offset by -6nT). Of note here is the sudden jump, a shock in the Solar wind, around the 27th minute mark. The 1 second averaging clearly reproduces the shock, whereas it can be seen that 1 minute averaging smooths the shock, producing a ramp-up over 3 to 4 minutes. This shows the importance, for Space weather, of 1 second averaging, to give forecasters a clear indication of the existence of the shock itself.

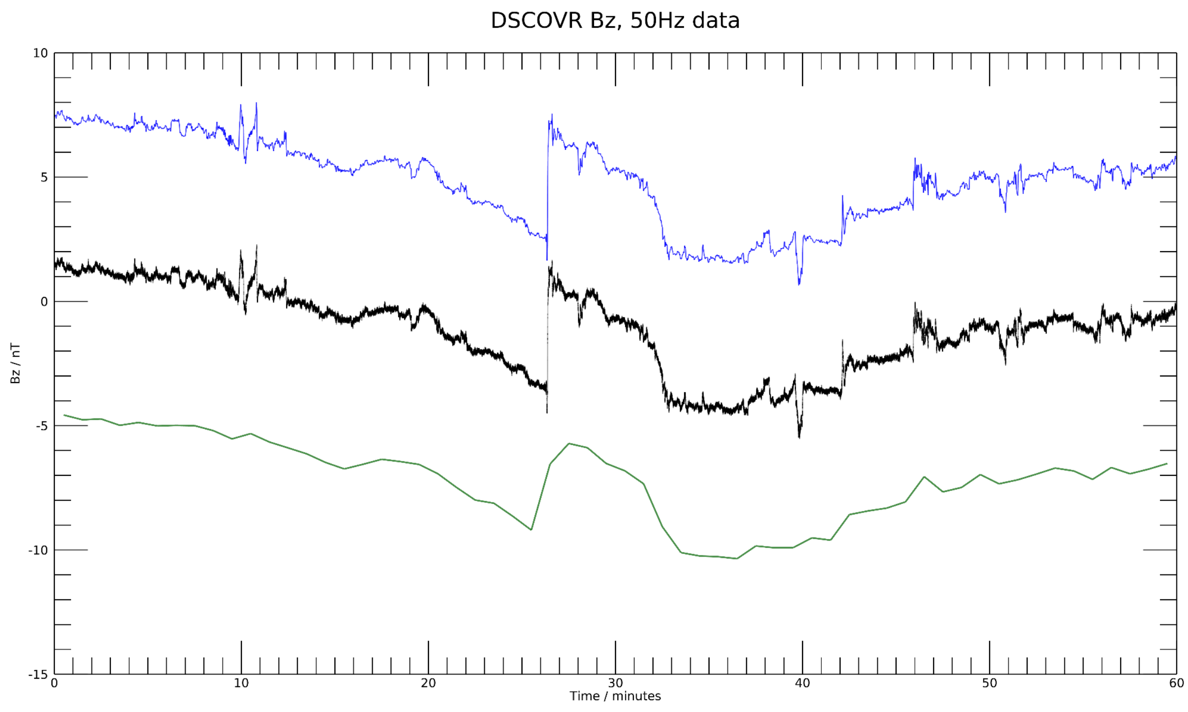

Figure 6: A 1 hour section of DSCOVR Bz data. The black line shows the original 50Hz data. The blue line shows 1 second averages, using the Huber M-estimator technique whilst the green line shows 1 minute averages, again using the Huber M-estimator. Both blue and green lines are offset by 6nT for clarity.

Note that the input time series is on a regular grid so expressing the time and averaging windows in terms of data points is equivalent to expressing them in terms of a time interval. From the discussion in section 3.2, there are several time windows that are relevant for SWFO-L1, namely 64 data points for MAG 1-second averages and 30 data points for STIS 1-minute averages.

Figure 7 highlights the performance of the estimators on a simulated data set that, by construction, has more data outliers. Here the superior performance of the robust HL and Huber M-estimators over the boxcar average is evident; they give an accurate and well-behaved estimate of the average despite the presence of large deviations.

One advantage of synthetic data such as this is that we know the true signal. As demonstrated in Section 4.1, the synthetic data was created by adding random noise to a known base signal, in this case a cosine. So we can check the accuracy of the various algorithms by comparing the average to the known base signal, computed on the target grid. For the example shown in Figure 7, the average deviation from the base signal for the boxcar average is 1.38. This is of course unsatisfactory given that the amplitude of the base signal is 1. Not surprisingly, given the visual impression from Figure 7, the robust estimators do much better. The HL estimator gives a mean deviation from the true signal of 0.0389. The Huber M-estimator is nearly identical, at 0.0383.

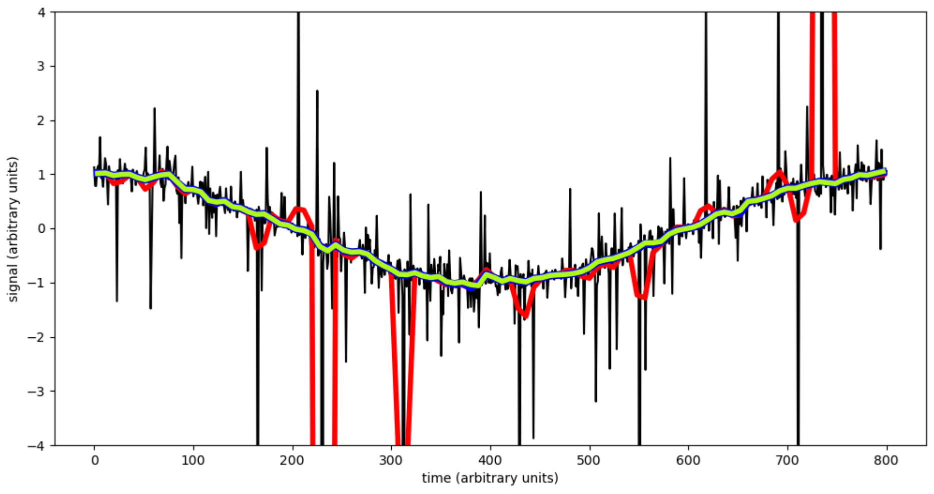

Figure 7: Location estimators applied to a simulated data set with many outliers. Line colors are as in Figure 5.

Figure 8 shows the performance of the estimators on a simulated data set with a sharp gradient, intended to mimic the behavior of an interplanetary shock. As can be seen, all three techniques, boxcar, HL and M-estimator, essentially produce the same, and satisfactory, result when faced with a sharp discontinuity of this kind.

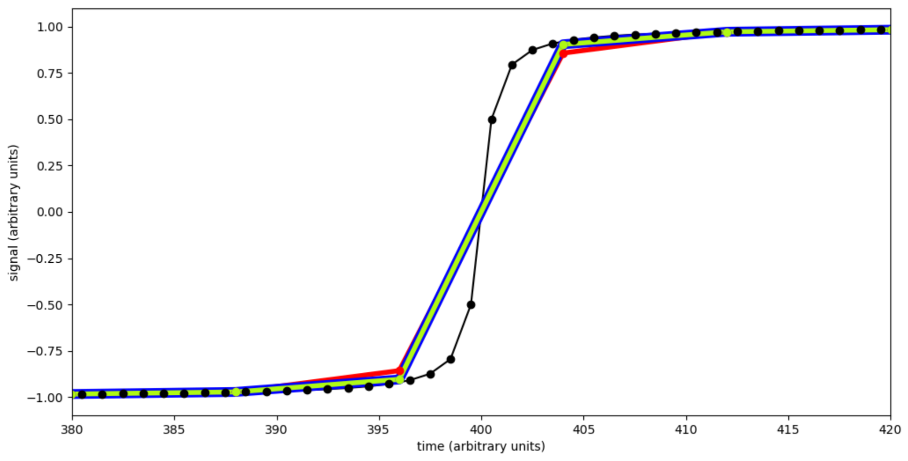

Figure 8: Location estimators applied to simulated data with a sharp gradient intended to mimic an interplanetary shock wave. Line colors represent the different estimators as in Figure 5.

Computational Performance

As emphasized in Section 2.2, another operational requirement for averaging algorithms is that they need to run in real-time and not require excessive computational time. In tests, all of the algorithms proposed have computation times which are several orders of magnitude faster than real time. Therefore, both HL and Huber M-estimator techniques are satisfactory for real-time operations in this regard.

In summary, the accuracy of the HL and Huber M-estimators is found to be comparable. The Huber M-estimator is also more robust in the sense that it has a higher breakdown point of 50% rather than 29% (Section 3.1.2) and it is substantially more computationally efficient for a time window of 3840 data points (needed for the 1-min MAG averages). But, it does require iterative convergence. Though we did not encounter any occasions where the Huber M-estimator failed to converge, we cannot guarantee this will never happen in operations. So, the HL estimator provides a robust backup if and when the Huber M-estimator fails to converge.

5 PRACTICAL CONSIDERATIONS - ALGORITHM VALIDATION

Pre-Launch Validation:

The algorithm will be validated as described in Section 4 using both simulated data as well as real DSCOVR and ACE spacecraft data. Averaged data will be compared to both high-resolution data sets. Furthermore, for the DSCOVR MAG and FC time series, new software implementations of the algorithms described here will be compared to averages computed using the current DSCOVR operational software. Similarly, averages computed for the ACE/EPAM proxy data will be compared with 1-min averages computed using existing algorithms.

Post-Launch Validation:

As noted in Section 3.2.1, the use of the HL estimator for MAG 1-second averages and the Huber M-estimator for 1-minute averages provides a robust quality check on the performance of the algorithms. The results presented in Section 4.2 demonstrate that the two methods should give similar answers. Similarly, the results for the STIS averaging with the Huber M-estimator can be validated at any time by computing an HL estimate for comparison. Comparison of averaging results with the original Level 2 or Level 1b data also provides a validation check (cf. Figs. 6-7).

The SWiPS instrument only requires a linear interpolation so comparison of these results with the original Level 2 data provides a sufficient validation.

6 ASSUMPTIONS AND LIMITATIONS

6.1 Performance

The averaging schemes described here for use with SWFO-L1 MAG, SWiPS, and STIS data rely on robust statistical methods to ensure high quality averages. These averaging methods are the same as those that are currently in use for the MAG and FC instruments on the DSCOVR mission and have demonstrated sufficient accuracy, robustness, and efficiency in an operational setting.

6.2 Assumed Sensor Performance

The algorithm assumes sensors will meet specifications and perform within normal operational parameters. When invalid sensor data occurs, it is assumed that these errors are replaced by error fill-in values and/or tagged by quality control (QC) flags in the Level 2 (MAG, SWiPS) or Level 1b (STIS) metadata. If the Level 2 and Level 1b input files are in NetCDF4, error fill-in values can be a designated value, such as – 9999, or they can be set to the IEEE 754 standard NaN (not a Number) value which is recognized and handled appropriately by python packages such as numpy, scipy, matplotlib, and pandas. Alternatively, or in addition, NetCDF4 files may also include a **valid_range** attribute that specifies the expected range of valid data and treats data outside of this range as invalid. During Level 3 processing, data outside of the valid range as specified in the instrument GPAs will be treated as NaN.

7 REFERENCES

Bickel, P. J. (1965). On some robust estimates of location, *Ann. Math. Statist.,* **36**, 847 – 858.

Croux, C. and Dehon, C. (2013). Robust estimation of location and scale. *Encyclopedia of Environmetrics*.

Elsaied, H., and Fried, R. (2016). Tukey's M-estimator of the Poisson parameter with a special focus on small means. *Statistical Methods & Applications, 25,* 191-209.

Erceg-Hurn, D. M. and Mirosevich, V. M. (2008). Modern robust statistical methods: an easy way to maximize the accuracy and power of your research, *Amer. Psych*., **63(7)**, 591 – 601.

Gold, R.E., Krimigis, S.M., Hawkins, I.I.I., Haggerty, D.K., Lohr, D.A., Fiore, E., Armstrong, T.P., Holland, G. and Lanzerotti, L.J., 1998. Electron, Proton, and Alpha Monitor on the Advanced Composition Explorer spacecraft. **Space Science Reviews, 86(1-4)**, 541-562.

Harvey, C. C. and Schwartz, S.J. (1998). Time Series Resampling Methods, in *Analysis Methods for Multi-Spacecraft Data*, G. Paschmann and P. W. Daly (Eds.), ISSI Scientific Report SR-001 (Electronic edition 1.1)

Hodges, J. L. Jr. and Lehmann, E. L. (1963). Estimate of location based on rank test, *Ann. Math. Statist.,* **34(2)**, 598 – 611.

Huber, P. J. (1964). Robust estimation of location parameters, *Ann. Math. Statist.,* **35**, 75 – 102.

Huber, P.J. (1981). *Robust Statistics*, Wiley, New York

Kenney, J. F., and Keeping, E. S. (1962). "The Median". *Mathematics of Statistics, Pt. 1* (3rd ed.). Princeton, NJ: Van Nostrand. pp. 211–212.

McComas, D. J. et al. (2018). "Interstellar Mapping and Acceleration Probe (IMAP): A New NASA Mission", Space Science Reviews, **214**:116

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing*, 3rd Edition, Cambridge Univ. Press, Cambridge

Rousseeuw, P. J and Verboven, S. (2002). "Robust Estimation in Very Small Samples", *Computational Statistics & Data Analysis*, **40**, 741 – 758.

Zhang, Z. (1997). Parameter estimation techniques: a tutorial with applications to conic fitting, *Image and Vision Computing*, **15(1)**, 59 – 76.

Zwickl, R. D. et al. (1998). The NOAA real-time solar wind (RTSW) system using ACE data, *Space Science Reviews,* **86**, 633 – 648.

8 SOURCE CODES

8.1 IDL code used to create Figure 5

The IDL code used to create Figure 5, with implementation of Boxcar, Hodges-Lehmann (HL) and Huber M-Estimator algorithms on 50Hz DSCOVR MAG data is given here:

```
function compute_median_of_absolute_differences, values, avg0
n_values = n_elements(values)
absDiffs = fltarr(n_values)
for i = 0 , n_values - 1 do begin
  absDiffs[i] = abs(values[i] - avg0);
endfor
return, median(absDiffs)
End


function huber_function, values
n_values = n_elements(values)
MAX_ITERATIONS = 1000
F_TOLERANCE = 1.0E-7
HUBER_TUNING_FACTOR = 1.345
MAX_VALUE = 1.e99

avg = 0.0
avg0 = median(values)
tol = F_TOLERANCE * abs(avg0)
fWeight = fltarr(n_values)
diff = MAX_VALUE
mad = compute_median_of_absolute_differences(values, avg0)
iteration = 0

while iteration lt MAX_ITERATIONS and diff gt tol do begin

residual = fltarr(n_values)
totalValue = 0.0
totalWeight = 0.0
for j = 0 , n_values - 1 do begin

  residual[j] = (avg0 - values[j]) / mad

  if abs(residual[j]) le HUBER_TUNING_FACTOR then begin
    fWeight[j] = 1
  endif else begin
    fWeight[j] = HUBER_TUNING_FACTOR / abs(residual[j])
  endelse
  totalValue = totalvalue  + (values[j] * fWeight[j])
  totalWeight = totalweight + fWeight[j]

endfor

; compute the average
avg = totalValue / totalWeight
; update the median absolute difference
mad = compute_median_of_absolute_differences(values, avg);
; compute the difference of the averages to see if we're converging on a solution
diff = abs(avg - avg0)
; use the new average as the basis for comparison next time
avg0 = avg
iteration = iteration + 1
```

```
endwhile

return, avg
end



function MAD, x
;Median Absolute Deviation of a sample
c = 1.4826
x0 = median(x)
y = abs(x - x0)
return, c * median(y)
end


function HL, x,w,n
;Hodges-Lehmann location estimator
idx = 0
for j = 0 , n - 1 do begin
  for i = 0 , j do begin
    w[idx] = 0.5*(x[i]+x[j])
    idx = idx + 1
  endfor
endfor
return, median(w)
end



function binrange ,i, n_width, n_vals
  i1 = max([i*n_width,0])
  i2 = min([i1+n_width,n_vals])
  return, [i1, i2]
end



pro plot_dscovr_50hz_data_with_resampling_algorithms

compile_opt idl2

dir = "/Users/ghgm/IDLWorkspace/rats/"
;dir = "C:\Users\george.millward\dscovr_proxy_python\data\"
file = "oe_mg1_dscovr_s20220205000000_e20220205235959_p20220206013755_pub.nc"
filename = dir + file
ncdf_get, filename, ['time','bz_gsm'], output
time = output['time']
time_orig = time['value']
bz = output['bz_gsm']
bz_orig = bz['value']
i1 = 71000
i2 = i1 + 3000
```

```
bz = bz_orig[i1:i2 - 1] + 0.5   ; <- arbitrary add 0.5
time = (time_orig[i1:i2 - 1] - time_orig[i1]) *1.e-3

; plot margins
left = 0.04
right = 0.04
bottom = 0.16
top = 0.16

win = window(dimensions=[1200,900], location = [4000,200])
a = plot(time, bz, margin = [left, bottom, right, top], transparency = 70, $
  xtitle = 'Time / seconds', ytitle = 'Bz / nT', title = 'DSCOVR Bz, 50Hz data', /current)
a.title.font_size = 16



; n_vals is the length of the sample
n_vals = n_elements(time)
; n_width is the size of the averaging window
n_width = 50
; na is the length of the averaged variable
if n_vals mod n_width eq 0 then begin
  na = n_vals / n_width
endif else begin
  na = (n_vals / n_width) + 1
endelse
; define time at the center of each window
tbox = fltarr(na)
for i = 0 , na - 1 do begin
  i1 = i*n_width
  i2 = min([i1+n_width,n_vals])
  mw = i2 - i1
  tbox[i] = 0.5*(time[i1] + time[i2-1])
endfor

;-----------------------------------------------------------------------------
; resampling with boxcar average
bzbox = fltarr(na)

print, ''
print, 'BOXCAR '
TIC
for i = 0 , na - 1 do begin

  this = binrange(i, n_width, n_vals)
  i1 = this[0]
  i2 = this[1]
  mw = i2 - i1
  bzbox[i] = total(bz[i1:i2-1]) / mw

endfor
TOC

print, ''
```

```
b = plot(tbox, bzbox, color = 'red', /overplot, thick = 4, transparency = 50)

;#---------------------------------------------------------------------------
;# Hodges-Lehmann estimator
bzhl = fltarr(na)
print, ''
print, 'HODGES-LEHMANN ESTIMATOR'
TIC
; allocate work array
nhl = long((n_width*(n_width+1))/2)
work = fltarr(nhl)
for i = 1 , na - 3 do begin
  this = binrange(i, n_width, n_vals)
  i1 = this[0]
  i2 = this[1]
  bzhl[i] = HL(bz[i1:i2],work,n_width)
endfor
;
; do end bins separately because they may not be the same size
;
i1 = 0
i2 = n_width
mw = i2 - i1
nhl2 = long((mw*(mw+1))/2)
work2 = fltarr(nhl2)
bzhl[0] = HL(bz[i1:i2-1],work2,mw)
;
i = na-2
this = binrange(i, n_width, n_vals)
i1 = this[0]
i2 = this[1]
mw = i2 - i1
nhl2 = long((mw*(mw+1))/2)
work2 = fltarr(nhl2)
bzhl[na-2] = HL(bz[i1:i2],work2,mw)
;
i1 = (na-1)*n_width
i2 = n_vals
mw = i2 - i1
nhl2 = long((mw*(mw+1))/2)
work2 = fltarr(nhl2)
bzhl[na-1] = HL(bz[i1:i2-1],work2,mw)
TOC
print, ''

c = plot(tbox, bzhl + 0.015, /overplot, color = 'dark green', thick = 4, transparency = 50)

;#---------------------------------------------------------------------------
;# M-estimator
bzhuber = fltarr(na)
print, ''
print, 'HUBER M-ESTIMATOR'
TIC
for i = 0 , na - 1 do begin
```

```
  this = binrange(i, n_width, n_vals)
  i1 = this[0]
  i2 = this[1]
  mw = i2 - i1
  bzhuber1 = huber_function(bz[i1:i2-1])
  bzhuber[i] = bzhuber1
endfor
TOC
print, ''

d = plot(tbox, bzhuber - 0.015, /overplot, color = 'blue', thick = 4, transparency = 50)

end
```

8.2 Python codes

8.2.1 Cauchy Noise Distribution

A Cauchy distribution for noise is generated by means of the *scipy.stats* python package. A sample Python code segment to produce such artificial data is as follows:

```
import numpy as np
from scipy.stats import cauchy

# Artificial data with a Cauchy distribution
ns = 800     # number of points in sample
rseed = 2422 # reproducible random seed

time = np.linspace(0, float(ns), num = ns, endpoint = True, dtype='float')
bz = np.cos(2*np.pi*time/t2)
np.random.seed(rseed)
noise = cauchy.rvs(loc = 0.0, scale = 0.1, size = ns)
bz += noise
```

8.2.1 Hodges-Lehmann estimator

The Hodges-Lehmann estimator is straightforward to program:

```
import numpy as np

def HL(x,w,n):
    """
    Hodges-Lehmann location estimator
    To avoid the computational expense of re-allocating it each time, pass the
    work array and size as arguments

    x = input array of size n
    w = work array of size n(n+1)/2
    n = length of x
    """

    idx = 0
    for j in np.arange(n):
        for i in np.arange(j+1):
            w[idx] = 0.5*(x[i]+x[j])
            idx += 1

    return np.median(w)
```

8.2.3 Huber M-estimator

The Huber M-estimator requires an iterative minimization, which we perform using the Newton-Raphson minimizer *newton* provided by the *scipy.optimize* software package. Convergence is not guaranteed. So, in case the minimizer does not converge, the Hodges-Lehmann estimator can be used as a robust backup:

```
import numpy as np
import scipy.optimize as opt

ns = len(bz)            # length of source time series
nw = 4                  # size of time window
nb = 2*nw               # size of averaging window
nov = int((nb - nw)/2)  # overlap of averaging windows

na = np.int64(ns / nw)

for i in np.arange(na,dtype=np.int64):
    i1, i2 = binrange(i, nw, nb, nov, ns)
    x = bz[i1:i2]

    scale = MAD(x)
    mu0 = np.median(x)

    try:
        loc = opt.newton(Huber_psi, mu0, fprime=Huber_psi_prime, args = (x, scale), tol=1.e-6)
    except:
        print("M-ESTIMATOR FAILED TO CONVERGE: DEFAULTING TO HL")
        mw = i2 - i1
        nhl = int(mw*(mw+1)/2)
        work = np.empty(nhl)
        loc = HL(x,work,mw)

    bzm[i] = loc
```

The custom functions are defined as follows. *Huber_psi* and *Huber_psi_prime* represent the Huber function defined in equation (7) and its derivative with respect to the location, *mu*. These are used by the Newton-Raphson iterative minimizer.

```python
def MAD(x, c = 1.4826):
    """
    Median Absolute Deviation of a sample
    """
    x0 = np.median(x)
    y = np.abs(x - x0)
    return c * np.median(y)

def Huber_psi(mu,x,sigma,k=1.345):
    r = (x - mu)/sigma
    z = np.where(np.abs(r) <= k, r, np.sign(r)*k)
    return np.sum(z)

def Huber_psi_prime(mu,x,sigma,k=1.345):
    r = (x - mu)/sigma
    dr = -1.0/sigma
    z = np.where(np.abs(r) <= k, dr, 0.0)
    return np.sum(z)

def binrange(i, nw, nb, nov, ns):
    i1 = np.max([i*nw-nov,0])
    i2 = np.min([i1+nb,ns])
    return i1, i2
```

8.3 Java code for implementing the Huber M-estimator

The Huber M-estimator java source code, as implemented for DSCOVR time series data, is given here. In the event that the Huber M-estimator fails to converge on a solution (in this case, after 1000 iterations) the algorithm resorts to a simple median average.

The constants required for this code are:

MAX_ITERATIONS = **1000**
F_TOLERANCE = **1.0E-7**
HUBER_TUNING_FACTOR = **1.345**
MAX_VALUE = **1.0E99**

```java
/**

* Compute the average using the Huber M-estimator. (See DSCOVR Time Series Data Averages ATBD v2.4)

*

* @param values the array of values to average

* @param statusSocket the ZMQ socket for the Status Processor

* @return the Huber average

*/

public static Double computeHuberAverage(double[] values, ZMQ.Socket statusSocket) {

// get the median of the array of values to use as the initial basis of comparison

Double avg0 = Utils.median(values);

// fall out if no median

if (avg0 == null) {

warn("No values to average", statusSocket);

return null;

}
```

```
// get the median of the array of absolute differences between the value and the median
Double mad = computeMedianOfAbsoluteDifferences(values, avg0);
if (mad == 0) {
LOGGER.debug("median absolute difference = 0, all values are equal", statusSocket);
return avg0;
}
return computeHuberAverageDetail(values, avg0, mad, statusSocket);
}
/**
* Compute the average using the Huber M-estimator. (See DSCOVR Time Series Data Averages ATBD v2.4)
*
* @param values the array of values to average
* @param avg0 median of the array of values
* @param mad median of the array of absolute differences between the value and the avg0
* @param statusSocket the ZMQ socket for the Status Processor
* @return the Huber average
*/
private static Double computeHuberAverageDetail(double[] values, Double avg0, Double mad, ZMQ.Socket
statusSocket) {
// the final average
Double avg = 0.0;
// difference tolerance for convergence
double tol = F_TOLERANCE * Math.abs(avg0);
// weighting factors
double[] fWeight = new double[values.length];
double diff = Double.MAX_VALUE;
int iteration = 0;
while (iteration < MAX_ITERATIONS && diff > tol) {
// get the array of residuals of the average0 - value[i] / mad
double[] residual = new double[values.length];
double totalValue = 0;
double totalWeight = 0;
for (int j = 0; j < values.length; j++) {
// compute the residual
residual[j] = (avg0 - values[j]) / mad;
// select the weighting factor based on the absolute value of the residual
if (Math.abs(residual[j]) <= HUBER_TUNING_FACTOR) {
fWeight[j] = 1;
} else {
fWeight[j] = HUBER_TUNING_FACTOR / Math.abs(residual[j]);
}
totalValue += values[j] * fWeight[j];
totalWeight += fWeight[j];
}
// compute the average
avg = totalValue / totalWeight;
// update the median absolute difference
mad = computeMedianOfAbsoluteDifferences(values, avg);
// compute the difference of the averages to see if we're converging on a solution
diff = Math.abs(avg - avg0);
// use the new average as the basis for comparison next time
avg0 = avg;
iteration++;
}
// did we fail to converge on a solution?
if (iteration == MAX_ITERATIONS) {
// did not converge
warn("Huber failed to converge " + Arrays.toString(values), statusSocket);
avg = null;
} else {
// did converge
LOGGER.debug("# of Huber iterations {}", iteration);
}
return avg;
}
/**
* Get the median of the array of absolute differences between the value and the basis of comparison.
*
* @param values the array of values being averaged
* @param avg0 the basis of comparison
* @return the median of the absolute differences
*/
private static Double computeMedianOfAbsoluteDifferences(double[] values, double avg0) {
double[] absDiffs = new double[values.length];
for (int i = 0; i < values.length; i++) {
absDiffs[i] = Math.abs(values[i] - avg0);
}
return Utils.median(absDiffs);
}
/**
* Return the median of an array of real values.
*
* @param values the array of values
* @return the median value
```

```
*/
public static Double median(double[] values) {
// create a copy of the input array for sorting
double[] sortedValues = values.clone();
Arrays.sort(sortedValues);
Double median = null;
if (sortedValues.length > 0) {
if ((sortedValues.length % 2) == 0) {
// even, compute the middle left index
int middle = (sortedValues.length / 2) - 1;
LOGGER.debug("even: middle={}", middle);
// median is the simple average of the two middle values
median = (sortedValues[middle] + sortedValues[middle + 1]) / 2.0;
} else {
// odd, compute the middle index
int middle = sortedValues.length / 2;
LOGGER.debug("odd: middle={}", middle);
median = sortedValues[middle];
}
}
return median;
}
```