

***Wind/3DP TPLLOT Tutorial:***  
**Documentation for the Modified SSL Software**

Lynn B. Wilson III

October 29, 2014

# Table of Contents

<b>1</b>	<b>The <i>Wind</i> Spacecraft</b>	<b>1</b>
<b>2</b>	<b>Motivation for New Software</b>	<b>2</b>
<b>3</b>	<b><i>Wind</i> 3DP Particle Detector</b>	<b>2</b>
<b>4</b>	<b>IDL Setup</b>	<b>3</b>
4.1	Directory Path Setup (LZ Data): UMN3DP $\leq$ Version 2.2.1 . . . . .	3
4.1.1	Directory Path Setup: MFI CDF Files . . . . .	3
4.1.2	Directory Path Setup: Orbit ASCII Files . . . . .	4
4.1.3	Directory Path Setup: 3DP IDL Save Files . . . . .	4
4.1.4	Bash Profile for <i>Wind</i> /3DP TPLOT . . . . .	4
4.2	Directory Path Setup (LZ Data): UMN3DP $>$ Version 2.2.1 . . . . .	5
<b>5</b>	<b>Getting Started with <i>Wind</i>/3DP TPLOT</b>	<b>6</b>
5.1	TPLOT Basics . . . . .	6
5.2	Starting: Load LZ Data . . . . .	8
5.3	Starting: Load Solar Wind Data . . . . .	8
5.4	Starting: Load Orbit Data . . . . .	10
5.5	IDL Implementation: Distribution Function Calculations . . . . .	11
5.6	IDL Implementation: Heat Flux Calculation . . . . .	13
<b>6</b>	<b>Plotting the Results</b>	<b>14</b>
6.1	Plotting: Velocity Distribution Function . . . . .	14
6.2	Plotting: Particle Spectra Data . . . . .	18
<b>A</b>	<b>IDL Implementation: Tensor Rotations and Manipulations</b>	<b>21</b>
<b>B</b>	<b>The Pesa High “Glitch”</b>	<b>25</b>
B.1	Finding the Pesa High “Glitch” in IDL . . . . .	25
<b>C</b>	<b>Appendix: Conversion Factors and Constants</b>	<b>28</b>

List of Tables

I    Pesa High Bad Data Bins . . . . . 25

List of Figures

A	<i>Wind</i> 3DP DF EL Example . . . . .	14
B	<i>Wind</i> 3DP PAD EL Example . . . . .	15
C	<i>Wind</i> 3DP DF PHB Example . . . . .	17

## 1 The *Wind* Spacecraft

The *Wind* spacecraft was launched on November 1st, 1994 by a Delta II rocket from Cape Canaveral Air Force Station in Merritt Island, FL. For the first two years of the mission WIND was in a highly elliptical orbit on the sunward side of the Earth with an apogee of 250 Earth radii ( $R_E$ ) and a perigee of at least 5  $R_E$ . *Wind* is the first of NASA's Global Geospace Science (GGS) program, which is part of the International Solar-Terrestrial Physics (ISTP) Science Initiative, a collaboration between several countries in Europe, Asia, and North America. The aim of ISTP is to understand the behavior of the solar-terrestrial plasma environment in order to predict how the Earth's atmosphere will respond to changes in solar wind conditions. WIND's objective is to measure the properties of the solar wind before it reaches the Earth [Desch, 2005].

The *Wind* spacecraft has an array of instruments including: Konus [Aptekar et al., 1995], the *Wind* Magnetic Field Investigation (MFI) [Lepping et al., 1995], the Solar *Wind* and Suprathermal Ion Composition Experiment (SMS) [Gloeckler et al., 1995], the Solar *Wind* Experiment (SWE) [Ogilvie et al., 1995], a Three-Dimensional Plasma and Energetic Particle Investigation (3DP) [Lin et al., 1995], the Transient Gamma-Ray Spectrometer (TGRS) [Owens et al., 1995], and the Radio and Plasma Wave Investigation (WAVES) [Bougeret et al., 1995]. The Konus and TGRS instruments are primarily for gamma-ray and high energy photon observations of solar flares or gamma-ray bursts. The SMS experiment measures the mass and mass-to-charge ratios of heavy ions. The SWE and 3DP experiments are meant to measure/analyze the lower energy (below 10 MeV) solar wind protons and electrons. The WAVES and MFI experiments were designed to measure the electric and magnetic fields observed in the solar wind. All together, the *Wind* spacecrafts suite of instruments allows for a complete description of plasma phenomena in the solar wind plane of the ecliptic [Desch, 2005].

More information on *Wind* can be found at:

[Wind Home Page](#)

[Wind Wikipedia Page](#)

If the links are broken, then copy-and-paste the following into your browser:

<http://wind.nasa.gov>

[http://en.wikipedia.org/wiki/WIND\\_\(spacecraft\)](http://en.wikipedia.org/wiki/WIND_(spacecraft))

## 2 Motivation for New Software

The main goal of *cleaning up* the *Wind/3DP* SSL libraries was to improve portability, robustness, and remove any scientific *bottleneck* in the software. This software, UMN3DP, is a standalone distribution originally meant to be used independent of any other distribution. Since then, the THEMIS group has produced the TDAS<sup>1</sup> software, which can be found at:

<http://themis.ssl.berkeley.edu/software.shtml>

The UMN3DP software still works as a standalone package used to analyze data from the *Wind/3DP* spacecraft.

The original software had some error handling and few comments, which made it difficult to use as a novice programmer. The tar ball of software I was originally given had more than five levels of directories, each with a copy or different version of a routine found in the first directory. So much of the original work involved organizing the routines and collecting the most relevant versions. The lack of error handling in some of the routines that called the shared object libraries caused segmentation faults on Sun Machine computers if used incorrectly. Thus, in an attempt to automate everything and increase the capacity to view large amounts of data with relatively few command line prompts, I soon realized that a large amount of the software needed to be altered.

Most of the original TPLOT routines have been left untouched because they work and there's no reason to fix something if it isn't broken. I have, however, *cleaned up* a number of programs by either adding a **man page**<sup>2</sup> or just organizing the code and commenting it where it previously had no comments. The hope is that the comments, man pages, and added software will allow the 3DP data to be accessed with more ease. This way the data can be used by scientists and they won't be held up by a *bottleneck* in software.

An updated list of software is kept in the `~/wind_3dp_pros/LYNN_PRO/` directory under the name **list\_of\_3DP-TPLOT\_pros-changed.txt**. This is a list of programs I've altered in the 3DP SSL libraries and programs I've added. Each program I've added has a short description of their purpose along with the date of last modification and version number.

In addition to making the 3DP data more accessible, much of the software should work for other satellite particle detectors with similar IDL data structure formats for their particle distributions. For instance, the particle data structures produced for the STEREO satellites have almost exactly the same format as those for *Wind*. Thus, a great majority of the data can be analyzed using the same software, since it only depends on the structure tag names. Some of the software will not care about which instrument the data came from if the data is examined in TPLOT. The default structure formats used in TPLOT are taken advantage of in most of my software. Though this isn't completely generalized, it is generalized in the context of TPLOT. Other code is generalized enough that it won't care whether you use TPLOT or any of the other 3DP software. Regardless, the code is easily adapted to examine particle data structures from other spacecraft making it a useful software package in many ways.

## 3 Wind 3DP Particle Detector

Detailed detector notes can be found in the PDF file named **wind\_notes\_3dp.pdf** or the following references: *McFadden et al.* [2007], *Wüest, M., Evans, D. S., & von Steiger, R.* [2007], or *Lin et al.* [1995]. The UMN3DP software can be found at:

<http://tetra.space.umn.edu/wiki/doku.php/umn-wind3dp>

<sup>1</sup>The *UMN Modified Wind/3DP IDL Libraries* or UMN3DP software should be run independent of the TDAS software, but routines from the **LYNN\_PRO** and **THEMIS\_PRO** directories can be used with TDAS.

<sup>2</sup>(man page  $\equiv$  Manual Page) I'll refer to these periodically throughout this tutorial. It's a header in each program that documents the code and explains to the user how to use the code and other miscellaneous things.

## 4 IDL Setup

**Warning! There is currently no shared object libraries for Windows OS.**

Most of the software can still be used, but you will not be able to load level zero 3DP data if you run IDL on a Windows machine. One would need Berkeley to compile a new shared object library for this to work.

### 4.1 Directory Path Setup (LZ Data): UMN3DP $\leq$ Version 2.2.1

**If you have a later version of the UMN3DP software, skip to Section 4.2.**

Prior to downloading the software, one needs to set up some directory trees. The level zero (lz) data for the 3DP instrument should be put in a specific directory. If the version of your UMN3DP software is  $\leq$  2.2.1, then continue on otherwise skip to Section 4.2. Many of the directory path issues were resolved with the implementation of the IDL system variable defined in *wind\_3dp\_umn\_init.pro*.

The lz data should go in following directory:

`/data1/wind/3dp/lz/????/`

where `????` is the year associated with a given set of data files. Since *Wind* has been in operation from 1995 through 2013, I would suggest creating a directory for each year. In this same directory, one needs a look up file called **wi\_lz\_3dp\_files** which contains the locations of all the level zero files. The format for each file should be, using April 1, 1995, as an example:

`1995-04-01/00:00:00 1995-04-02/00:00:00 /data1/wind/3dp/lz/1995/wi_lz_3dp.19950401_v01.dat.`

The part of the file name shown as `*_v0?.dat*` is referencing the version number of the binary file. The value for `?` changes from file to file, so one must be careful when writing the look up file **wi\_lz\_3dp\_files** to change these accordingly. This file can be produced by running the routine *wind\_mflist\_print.pro*, found in the `~/wind_3dp_pros` directory.

After downloading and uncompressing the tarball of programs which make up the modified *Wind/3DP* TPLOT libraries, one should immediately alter the directory locations of the following programs:

1. *setfileenv.pro*
2. *load\_3dp\_data.pro*
3. *init\_wind\_lib.pro*
4. *idl\_3dp\_init* .

#### 4.1.1 Directory Path Setup: MFI CDF Files

The file for the MFI data is of a similar format but the file names and file path change accordingly. Remember, the directory location of these two files must be sourced in *setfileenv.pro*. To do this, one must use the following commands:

```
mfidir1 = '[MFI Dir. Path]/MFI_CDF'
SETENV,'WI_H0_MFI_FILES='+mfidir1+'/wi_h0_mfi*.cdf'
SETENV,'WIND_DATA_DIR=[3DP Dir. Path]'
```

where the file path locations will change according to the choice of the user as to where they put the data<sup>3</sup>. The easiest thing for the user would be to use the directory path shown above, which will require the fewest modifications to the IDL path specifications.

For the MFI data, there is a specific place for the CDF files downloaded from:  
<http://cdaweb.gsfc.nasa.gov/>

<sup>3</sup>This is no longer necessary and now obsolete.

located in the `~/wind.3dp_pros/wind_data_dir/MFI.CDF/` directory. If you put the CDAWeb CDF files in this directory, then the routine `read_wind_mfi.pro` will not need to be changed because the routine `wind.3dp_umn_init.pro` should have defined this location already.

#### 4.1.2 Directory Path Setup: Orbit ASCII Files

Wind orbit information can be obtained from the following website:

<http://sscweb.gsfc.nasa.gov/cgi-bin/sscweb/Locator.cgi>

The data should be placed in the following directory:

`~/wind.3dp_pros/wind_data_dir/Wind_Orbit_Data/`

which is already defined in the software download. The format used by the routine `read_wind_orbit.pro` is obtained through the following steps:

1. Select the *Wind* spacecraft from the list
2. Enter a start and end time [*e.g.* Start Time: 1996/01/01 00:00:00, End Time: 1996/01/02 00:00:00] and make sure it encompasses only one day, nothing less and nothing more.
3. Click on **Output Options** button
4. Select the following: XYZ-GSE, XYZ-GSM, LAT/LON-GSE, LAT/LON-GSM, Dipole L Value (under **Additional Options**), and Dipole Inv Lat (under **Additional Options**)
5. Click on **Output Units/Formatting** button
6. Select the following: yy/mm/dd (under **Date**), hh:mm:ss (under **Time**), and km-Kilometers with 3 decimal places (under **Distance**)
7. Click on **Submit query and wait for output** button .

The header of the file should have 14 lines of information before any lines of data. If this basic format is kept, then the routine `read_wind_orbit.pro` will be able to read these ASCII files given either a date of interest or a time range (which may encompass several dates). The typical header file format can be found in one of the example files provided in the distribution.

#### 4.1.3 Directory Path Setup: 3DP IDL Save Files

If the user wishes to load data from the level zero files, this must be done when IDL is in 32-bit mode, which limits the user to ~2 GB of memory. If the user would like to load a large amount of data (*e.g.* multiple days of SST data), there is a way around the 2 GB limitation. The user can load the data into IDL (methods discussed below) and create IDL Save files. If these files are put in the following location:

`~/wind.3dp_pros/wind_data_dir/Wind.3DP_DATA/IDL_Save_Files/MMDDYY/`

then one can locate the files using the IDL system variable initialized by `wind.3dp_umn_init.pro` located in the main `~/wind.3dp_pros/` directory. To find the files, one need only do the following:

```
UMN> default_extension = '/wind.3dp_pros/wind_data_dir/Wind.3DP_DATA/IDL_Save_Files/'
UMN> default_location = default_extension+date+'/'
UMN> DEFSYSV,'!wind3dp_umn',EXISTS=exists
UMN> IF NOT KEYWORD_SET(exists) THEN mdir = FILE_EXPAND_PATH('')+default_location[0]
UMN> IF KEYWORD_SET(exists) THEN mdir = !wind3dp_umn.WIND_3DP_SAVE_FILE_DIR+date+'/'
UMN> IF (mdir EQ '') THEN mdir = default_location[0]
UMN> mfiles = FILE_SEARCH(mdir,'*.sav')
```

#### 4.1.4 Bash Profile for Wind/3DP TPLOT

The user should have their unix/linux terminals source some sort of .bash\_profile or .cshrc file. If the user is not using a unix-based machine like Windows, then I cannot help. I have provided an example bash profile, which can be put into your start directory (*i.e.* directory after using `cd` command with no specifications/options from unix prompt) and called `.bash_profile`.



## 4.2 Directory Path Setup (LZ Data): UMN3DP > Version 2.2.1

After some useful feedback, I made a concerted effort to remove the variability of the IDL path specifications. In these later versions of the software, the lz data can be found in the following directories:

`~/wind_3dp_pros/wind_data_dir/data1/wind/3dp/lz/????`

The location of the MFI and other data types has not changed. I have changed the following routines to be platform and machine independent<sup>4</sup>: *setfileenv.pro*, *load\_3dp\_data.pro*, *init\_wind\_lib.pro*, and *idl\_3dp.init*<sup>5</sup>.

To start IDL and use this software, the user can do one of the following:

1. utilize the example bash functions provided at: [UMN 3DP Documentation](#); or
2. source the *setup\_wind3dp\_bash* file prior to starting IDL; or
3. type the following after starting IDL:  
IDL> `@./wind_3dp_pros/start_umn_3dp.pro`
4. Once the paths are correctly secured, run the following:  
UMN> `wind_mflist_print, INSTR='3dp', DTYPE='lz'`  
and take the output file named **wi\_lz\_3dp\_files** and put it in the directory  
`~/wind_3dp_pros/wind_data_dir/data1/wind/3dp/lz/` replacing the default file. This will make sure that *load\_3dp\_data.pro* can find the level zero 3DP data.

If the user has done everything correctly, a series of compile statements should appear and typing the following should produce no errors:

```
UMN> .compile get_3dp_structs.pro
```

If errors occur, or the routine cannot be found, then you did not start IDL from the same directory where `~/wind_3dp_pros` is located or there is an issue with your IDL paths. Make sure you followed these instructions and try again. If an error still persists, feel free to e-mail me (see footer for my e-mail address).

---

<sup>4</sup>This is assuming, of course, that the user has correctly put the `~/wind_3dp_pros` directory in the correct place.

<sup>5</sup>Note that *idl\_3dp.init* is now obsolete.

## 5 Getting Started with *Wind*/3DP TPLLOT

### 5.1 TPLLOT Basics

Let's start with some simple background on TPLLOT. TPLLOT is a conglomeration of routines started at Berkeley that were an attempt to make plotting and changing plots faster in IDL. It has since grown in a massive set of software and branched out into the libraries now known as TDAS and SPEDAS, distributed by Berkeley and UCLA<sup>6</sup>. Anyways, let's start with the basics.

To load data into TPLLOT, one creates a generalized data structure with the following tags:

1. **X** : (Required) [N]-Element array [double] of time stamps in units of Unix time<sup>7</sup> to act as the independent variable
2. **Y** : (Required) [N]- or [N,M]-Element array [float or double] of dependent variable data, where each  $N_j$  in **Y** corresponds to each  $N_j$  in **X**. For vectors (e.g., magnetic fields), then  $M = 3$ .
3. **V** : (Optional) [M]-Element array [float or double] of a second dependent variable. Typically the **V** tag is used for spectral data, such as wavelet transform data, where each element of **V** corresponds to a frequency<sup>8</sup>.

Once this structure is created, say in the following way:

```
UMN> struc = {X:unix,Y:data,V:freqs}
```

then the user can send this data to TPLLOT using the following:

```
UMN> store_data,'TPLLOT_Handle_1',DATA=struc
```

where '[TPLLOT\\_Handle\\_1](#)' is a unique string that will identify the data stored in the *struc* structure. Once in TPLLOT, the user can refer to this data either by its string handle or an integer number defined by the order in which the data was loaded into TPLLOT relative to other data. The user can then plot the data, add options to the format of the output plots, manipulate the data, etc. all from the command line. Before moving on, assume we sent in two other sets of data and called them '[TPLLOT\\_Handle\\_2](#)' and '[TPLLOT\\_Handle\\_3](#)'.

To see whether data was loaded into TPLLOT, type the following:

```
UMN> tplot_names
```

where you should at least see the following output:

```
1 TPLLOT_Handle_1
2 TPLLOT_Handle_2
3 TPLLOT_Handle_3
```

Note that the string you choose (hopefully not '[TPLLOT\\_Handle\\_1](#)') can be changed and is not restricted to explicitly follow any of the names seen in this tutorial. You are free to define the names as you please. If you see names that are not exactly the same as those shown herein, do not worry. The TPLLOT handle is just a unique name one uses to distinguish one set of data from another. So long as you are consistent, you are free to name things as you wish<sup>9</sup>.

To plot data in TPLLOT is very easy, just type one of the following:

<sup>6</sup>Originally, TDAS was just for THEMIS but has since generalized to incorporate other missions. The latest improvement, called SPEDAS, is supposed to act as a generalized platform for current and future missions.

<sup>7</sup>seconds elapsed since 00:00:00 UTC on January 1, 1970.

<sup>8</sup>If the **V** tag is used, one can also use a **SPEC** tag set to 1 (show color-scaled spectra) or 0 (show a stacked line plot). In some cases, **Y** has a third dimension (e.g., for pitch-angle distributions) then one would use **V1** and **V2** instead of just **V**.

<sup>9</sup>I repeat, do NOT blindly trust commands in this tutorial. Some are meant as general references and others are just specific examples.

```
UMN> tplot,[1]
```

or...

```
UMN> tplot,'TPlot_Handle_1'
```

Both commands should have the same effect (assuming 'TPlot\_Handle\_1' is associated with the integer 1). If you want to plot multiple things in one window, then just combine into an array as follows:

```
UMN> tplot,[1,2,3]
```

or...

```
UMN> tplot,['TPlot_Handle_1','TPlot_Handle_2','TPlot_Handle_3']
```

The user can also pass specific time ranges to **tplot.pro** using the *TRANGE* keyword, which accepts a [2]-element array of Unix times corresponding to the start and end time of the time range you wish to view. The user can alter plotting options for each TPlot handle using **options.pro**. The syntax is as follows:

```
UMN> options,[TPlot handle to be altered],[Plot keyword],[value],DEFAULT=[0|1]
```

where *[TPlot handle to be altered]* is the string or integer TPlot handle of the variable for which you wish to alter plotting options, *[Plot keyword]* is a string of nearly any keyword accepted by the IDL built-in routine **PLOT.PRO**<sup>10</sup>, *[value]* is the input appropriate for the associated keyword, and the *DEF* setting determines whether the plotting option is a default(1) or not(0) option for this variable<sup>11</sup>.

If you wish to retrieve the data from TPlot, you need only use **get\_data.pro**. The syntax is as follows:

```
UMN> get_data,[TPlot handle to get],DATA=struc,DLIMITS=dlim,LIMITS=lim
```

where *[TPlot handle to get]* is the string or integer TPlot handle of the variable to return, *struc* will be an IDL structure of similar format to the one described above when creating a TPlot variable, and *dlim* and *lim* will be IDL structures containing default and regular plotting options, respectively, set by the user<sup>12</sup>.

The advantage to using TPlot is that you can add plotting options, plot the data, and then change the time range very quickly (e.g., using **tlimit.pro**) without re-entering a great deal of commands. TPlot has a great deal of versatility and options, which makes it both highly information dense and highly useful. Below I will go into some specific details about different 3DP data loading techniques/options.

<sup>10</sup>Typical strings used include 'YTITLE', 'YLOG', 'YRANGE', 'YSTYLE', 'YMINOR', 'YTICKNAME', 'YTICKV', 'YTICKS', etc. Usually one only sets the Y-Axis labels/titles since the X-axis is defaulted to times (shown as UT times when plotted).

<sup>11</sup>Note that the user could have set these options when originally defining the TPlot handle by using the keywords *DLIMITS* and *LIMITS* for default and regular plotting options, respectively. The inputs should be structures with tag names matching the keywords accepted by routines like **PLOT.PRO**.

<sup>12</sup>If no options were set, then both *dlim* and *lim* will be equal to 0.

## 5.2 Load LZ Data

Assuming you've managed to source all of your environment variables correctly, you have level zero (LZ) data, and you have *Wind*/MFI CDF files, you can get started.

Now that 3DP can locate the data, let's load some into IDL. The following lines will illustrate how to do this using the updated versions of the code:

```
UMN> date      = '040196' ; => i.e. 1996-04-01
UMN> duration = 46 ; => 46 hours of data to load
UMN> tra      = ['1996-04-01/00:00:00','1996-04-02/22:00:00']
UMN> trange   = time_double(tra)
UMN> memsize  = 150.
UMN> load_3dp_data,'96-04-01/00:00:00',duration,QUALITY=2,MEMSIZE=150.
```

where **load\_3dp\_data.pro** will load both the level zero binary file data and the magnetic field data for the dates of interest into TPLLOT. Now that we have loaded the data (assuming the program didn't break or didn't find any data), let's get some particle data. In the original version of the 3DP software library, this next step could be a rather arduous and painful task. Depending on whether you were curious about the ES analyzers or the SST data, one might end up typing tens to dozens of lines on the command prompt to get all the particle structures desired within a given time range. I've reduced that mess, because I'm lazy, to only one line of code:

```
UMN> dat = get_3dp_structs('el',TRANGE=trange)
```

where *dat* is a data structure with an [n,2]-element time array [Unix Times] and an array of 3DP data structures, consistent with what you would get from **get\_??\_pro**<sup>13</sup>.

The program, **get\_3dp\_structs.pro**, eliminates any non-valid structure (*i.e.* *dat.VALID=0*) before returning them to the user, thus all the structures should be good. The *DATA* structure tag will contain all structures in your defined time range, or all the structures available from the amount of data you originally loaded. Originally, the **get\_??\_pro** occasionally returned structures or non-structures which would result in a *Conflicting Data Structures* error in IDL when trying to concatenate structure arrays. I even managed to get around the PESA High mapcodes which cause **get\_ph.pro** and **get\_phb.pro** to return structures with different values for the structure tag *NBINS*, depending on the data mode. Regardless, now we have data, so let's see what we can do with it.

To start with, let's add the magnetic field to our data structures. This will allow us to create pitch-angle distributions later. Now assuming you have loaded the MFI data correctly, then do the following:

```
UMN> ael = dat.DATA
UMN> add_magf2,ael,'wi_B3(GSE)'
```

where **add\_magf2.pro** is an adapted version of **add\_magf.pro** but vectorized (thus much faster when the number of structures in *ael* becomes large). The magnetic field data is now in every structure that has its time range within the time range of loaded MFI data.

## 5.3 Load Solar Wind Data

Now let's look at some solar wind parameters like the velocity and density. These can be found from the PL detector in most cases<sup>14</sup>, so let's get some PL data. The method to load the solar wind parameters into TPLLOT can be done in two different ways depending on whether you want to use the PL structures later. First, let's assume you want the PL<sup>15</sup> structures for later use, thus do:

<sup>13</sup>?? = 'el','elb','ehb','sf','ph','plb', etc.

<sup>14</sup>except for the turbulent region downstream of a shock which I'll discuss later

<sup>15</sup>You should always check the results returned by **pesa\_low\_moment\_calibrate.pro** against results returned by **get\_pmom2.pro** and the SWE Faraday cups. The reason is that unless PESA Low is in burst mode, then the data structures only have a 5x5 array of anodes. In burst mode (and the onboard moments), the moments are calculated using the entire 8x8 array of anodes providing a higher resolution field of view.

```

UMN> pldt = get_3dp_structs('pl',TRANGE=trange)
UMN> plbdt = get_3dp_structs('plb',TRANGE=trange)
UMN> apl = pldt.DATA
UMN> aplb = plbdt.DATA
UMN> pesa_low_moment_calibrate,DATE=date,TRANGE=trange,/NOLOAD,PLM=apl,PLBM=aplb

```

which will result in TPLOT showing the following variables available for plotting (assuming PL data exists and you have MFI data loaded):

```

UMN> tplot_names
  1 wi_B3_MAG(GSE)
  2 wi_B3(GSE)
  3 pl
  4 plb
  5 N_i2
  6 sc_pot_2
  7 T_i2
  8 V_Ti2
  9 V_sw2
 10 V_mag2
 11 PL_MAGT3
 12 pl_flux
 13 Ti_perp
 14 Ti_para
 15 Ti_anisotropy

```

and I'll get back to what each of these represents later, but right now let's look at the other way to get the same result. Let's say we are lazy and don't want to load the PL structures first, then we'd just do:

```

UMN> pesa_low_moment_calibrate,DATE=date,TRANGE=trange

```

which should give the same exact result as the previous method. Now I mentioned previously that downstream of a shock, PL becomes *inaccurate*. The reason for this is multifaceted, but there is an *easy fix*. Using either the *Wind*/WAVES TNR receiver or J.C. Kasper's IP shock data base [Kasper, 2007], estimate the shock compression ratio and determine the center of the ramp in Unix time. Then use the PL calibration program in the following manner:

```

UMN> pesa_low_moment_calibrate,DATE=date,TRANGE=trange,COMPRESS=compr,MIDRA=mid

```

where *compr* is the shock compression ratio (typically 1.5 to 2.5 for IP shocks) and *mid* is the Unix time of the center of the ramp. If you use *tplot.names.pro* now, you'll notice two new TPLOT variables: [N\\_i3](#) and [sc\\_pot.3](#). These are the calibrated ion density and spacecraft potential due to the turbulent downstream region of the shock.

The list of TPLOT variables are defined as follows:

<i>wi_B3_MAG(GSE)</i> : 3-Second magnitude of the MFI vector data	(1a)
<i>wi_B3(GSE)</i> : 3-Second GSE MFI vector data	(1b)
<i>pl</i> : PL moment structures (for now, ignore)	(1c)
<i>plb</i> : PLB moment structures (for now, ignore)	(1d)
<i>N_i2</i> : 1st Ion Density Estimate ( $cm^{-3}$ )	(1e)
<i>sc_pot_2</i> : 1st SC Potential Estimate ( $eV$ )	(1f)
<i>T_i2</i> : PL Avg. Ion Temperature Estimate ( $eV$ )	(1g)
<i>V_Ti2</i> : PL Thermal Speed Estimate ( $km/s$ )	(1h)
<i>V_sw2</i> : GSE Solar Wind Velocity ( $km/s$ )	(1i)
<i>V_mag2</i> : Magnitude of <i>V_sw2</i>	(1j)
<i>PL_MAGT3</i> : Vector Temp. [ <i>Perp</i> <sub>1</sub> , <i>Perp</i> <sub>2</sub> , <i>Parallel</i> ]( $eV$ )	(1k)
<i>pl_flux</i> : Ion Flux ( $s^{-1}cm^{-2}$ )	(1l)
<i>Ti_perp</i> : Avg. Perp. Ion Temp ( $eV$ )	(1m)
<i>Ti_para</i> : Avg. Para. Ion Temp ( $eV$ )	(1n)
<i>Ti_anisotropy</i> : $T_{i,\perp}/T_{i,\parallel}$	(1o)
<i>N_i3</i> : Adjusted Ion Density ( $cm^{-3}$ )	(1p)
<i>sc_pot_3</i> : Adjusted SC Potential ( $eV$ )	(1q)

## 5.4 Load Orbit Data

The following is NOT necessary, but may be desireable for some applications. So if you have issues here, you can move on and just ignore orbit plots/references later.

To load the orbit data for *Wind*, we can type the following into IDL:

```
UMN> wind_orbit_to_tplot,BNAME='wi.B3(GSE)',TRANGE=trange
```

which, you will notice, requires that we have loaded MFI data prior to calling. This is so the routine can interpolate the orbit data to the MFI timestamps.

We can change the Y-axis titles using the following:

```
UMN> options, 'Wind_Radial_Distance', 'YTITLE', 'Radial Dist. [R!DE!N'+']'
```

We can use TPLOT handles as tick mark labels using the following:

```
UMN> gnames = tnames('Wind_Radial_Distance')
UMN> tplot_options,VAR_LABEL=gnames[0]
```

So now the next time you plot the data, you will see an extra set of X-axis tick marks that correspond to the *Wind* location at each time tick mark.

## 5.5 IDL Implementation: Distribution Function Calculations

To calculate the distribution function (DF), use the following programs (after you've done the above steps):

```
UMN> dat = get_3dp_structs('el',TRANGE=trange)
UMN> ael = dat.DATA
UMN> add_magf2,ael,'wi_B3(GSE)'
UMN> add_vsw2,ael,'V_sw2'
UMN> add_scpot,ael,'sc_pot_3'
```

where we have added the GSE magnetic field, GSE solar wind velocity, and spacecraft potential to ALL the data structures. Now to calculate an individual PAD and DF, perform the following:

```
UMN> el = ael[10]
UMN> del = convert_vframe(el,/INTERP) ; Convert into SW Frame
UMN> pd = pad(del,NUM_PA=17L) ; PAD calculation
; Calculate the DF now
UMN> df = distfunc(pd.ENERGY,pd.ANGLES,MASS=pd.MASS,DF=pd.DATA)
; Get the structure tags from df and put them into del
UMN> extract_tags,del,df
```

where *pd* and *df* have the structure formats given by:

```
UMN> HELP,pd,/STRUCT,OUTPUT=hout
UMN> PRINT,hout,FORMAT='(;"",a)'
;** Structure <2122610>, 21 tags, length=14028, data length=14024, refs=1:
;  PROJECT_NAME      STRING      'Wind 3D Plasma'
;  DATA_NAME        STRING      'Eesa Low PAD'
;  VALID             INT          1
;  UNITS_NAME        STRING      'df'
;  TIME              DOUBLE       9.5532272e+08
;  END_TIME          DOUBLE       9.5532272e+08
;  INTEG_T           DOUBLE       3.1001112
;  NBINS             INT          17
;  NENERGY           LONG          15
;  DATA             FLOAT        Array[15, 17]
;  ENERGY           FLOAT        Array[15, 17]
;  ANGLES            FLOAT        Array[15, 17]
;  DENERGY           FLOAT        Array[15, 88]
;  BTH               FLOAT        27.2478
;  BPH               FLOAT        -37.0210
;  GF                FLOAT        Array[15, 17]
;  DT                FLOAT        Array[15, 17]
;  GEOMFACTOR        DOUBLE       0.00039375000
;  MASS              DOUBLE       5.6856591e-06
;  UNITS_PROCEDURE    STRING      'convert_esa_units'
;  DEADTIME          FLOAT        Array[15, 17]

UMN> HELP,df,/STRUCT,OUTPUT=hout
UMN> PRINT,hout,FORMAT='(;"",a)'
;** Structure <c3b4db0>, 3 tags, length=6000, data length=6000, refs=1:
;  VX0               FLOAT        Array[500]
;  VY0               FLOAT        Array[500]
;  DFC               FLOAT        Array[500]
```

where the tags  $VX0$  and  $VY0$  represent the parallel (with respect to the magnetic field) and perpendicular velocities, respectively. The tag  $DFC$  is the actual estimate of the DF in units of DF ( $\text{s}^3\text{km}^{-3}\text{cm}^{-3}$ ).

One should also note that though I used 17 pitch-angles to make these structures, which is good if there are sufficient flux levels. If the fluxes are very high, then one can get away with 24 pitch-angles for EL and ELB, but I'd be wary with the other detectors.



## 5.6 IDL Implementation: Heat Flux Calculation

The commands necessary to properly calculate the electron heat flux for a given electron distribution are as follows:<sup>16</sup>

```
UMN> el = get_el([unix time of interest])
UMN> add_magf2,el,['TPLLOT name of magnetic field']
UMN> add_vsw2,el,['TPLLOT name of solar wind velocity']
UMN> add_scpot,el,['TPLLOT name of spacecraft potential']
UMN> del = convert_vframe(el,/INTERP)
UMN> sum = mom_sum(del,SC_POT=del.SC_POT)
UMN> sumt = mom_translate(sum)
UMN> charge = -1
UMN> mass = sumt.MASS
UMN> nnorm = SQRT(ABS(2*charge/mass))
;; calculate heat flux tensor
UMN> qtens = (mass*nnorm^2)*sumt.NVVV
UMN> i3 = [[0,4,8],[9,13,17],[18,22,26]]
;; Get only specific elements by assuming symmetries
UMN> qqqs = (sumt.NVVV[sumt.MAP_R3])[i3]
;; Define heat flux vector [eV km/s cm-3, GSE]
UMN> qvec = TOTAL(qqqs,1L,/NAN)
;; Rotate into field-aligned coordinates
UMN> qrot = rot_mat(el[0].MAGF,qvec) ## qvec
```

Another, equally valid and typically more meaningful, approach is to calculate the heat flux vector in field-aligned coordinates which can be done with far less effort by doing:

```
UMN> el = get_el([unix time of interest])
UMN> add_magf,el,['TPLLOT name of magnetic field']
UMN> add_vsw,el,['TPLLOT name of solar wind velocity']
UMN> str_element,el,'SC_POT',[value],/ADD_REPLACE
UMN> del = convert_vframe(el,/INTERP)
UMN> mom = mom3d(del,SC_POT=del.SC_POT)
UMN> qvec = mom.QVEC
```

where *[value]* is the estimate of the spacecraft potential (eV) for this distribution and *qvec* is the heat flux vector in field-aligned coordinates (eV cm<sup>(-3)</sup> km/s) with the elements being [Perp.<sub>1</sub>,Perp.<sub>2</sub>,Para.].

<sup>16</sup> Assume you have already loaded particle, magnetic field, and solar wind velocity data. **Note here is an example where NOT to blindly follow the tutorial.** I have used a generalized naming system for these variables where '[']' should be replaced by the TPLLOT handle defined in your session of IDL.

## 6 Plotting the Results

### 6.1 Velocity Distribution Function

It's nice to have the data readily available, but what do we do with it? Well there are couple of ways one can look at the data. One can either plot the DF or one can plot the PAD for each particle structure. Note that all the plots shown in this tutorial have been cleaned up using Adobe Illustrator<sup>TM</sup>. To plot the DF as a contour plot with parallel and perpendicular cuts of the DF, do the following:

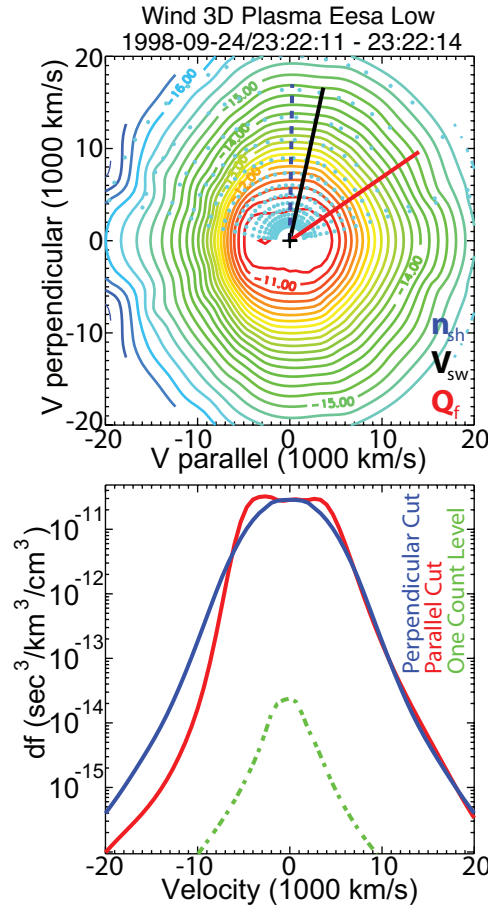


Figure A: This is an example image produced by **cont2d.pro** of an EESA Low data sample. The horizontal axis of the contour plot is parallel to the magnetic field and the vertical axis is perpendicular to the magnetic field in the plane created by the solar wind velocity and magnetic field direction. Also shown are the projections of the shock normal direction (dashed blue line), electron heat flux vector (solid red line), and solar wind vector (solid black line). The bottom panels show the cuts of the distribution function parallel (red line) and perpendicular (blue line) to the magnetic field. The one-count level (green line) is shown for reference. The small blue dots in the contour plot represent the locations of the actual data.

```
UMN> dat = el
UMN> dfra = [1e-16,5e-11]
UMN> cont2d,del,VLIM=2d4,NGRID=30L,GNORM=gnorm,/HEAT_F,MYONEC=dat,DFRA=dfra
```

which should produce a plot like the one seen in Figure A<sup>17</sup>. The plot description is listed in the figure caption. This figure is consistent with a flattop electron distribution [Feldman *et al.*, 1983] and is observed downstream of a strong interplanetary shock on 1998-09-24. The small blue dots show where the data is actually taken in the contour plot. It is important to show these since the manner in which the IDL built-in

<sup>17</sup>Note that the version of **cont2d.pro** in the UMN3DP software is significantly different than the one distributed in the TDAS distribution at: <http://themis.ssl.berkeley.edu/software.shtml>.

function, `CONTOUR.PRO`, attempts to close the contours can result in artificial distribution signatures. For instance, one can see that the data is not sampled perfectly along the magnetic field (horizontal axis of contour plot) but at a small angle ( $\gtrsim 10^\circ$ ). This angle can often be much higher, yet IDL will still make the contours appear as though there is a beam-like signature parallel or anti-parallel to the magnetic field. Also, often times low energy beam-like signatures appear in the data, but they can occasionally occur at a velocity which is inside the lowest velocity available (due to SC potential and reference frame transformation effects). These signals should NOT be trusted as they are artifacts of IDL, NOT the data!

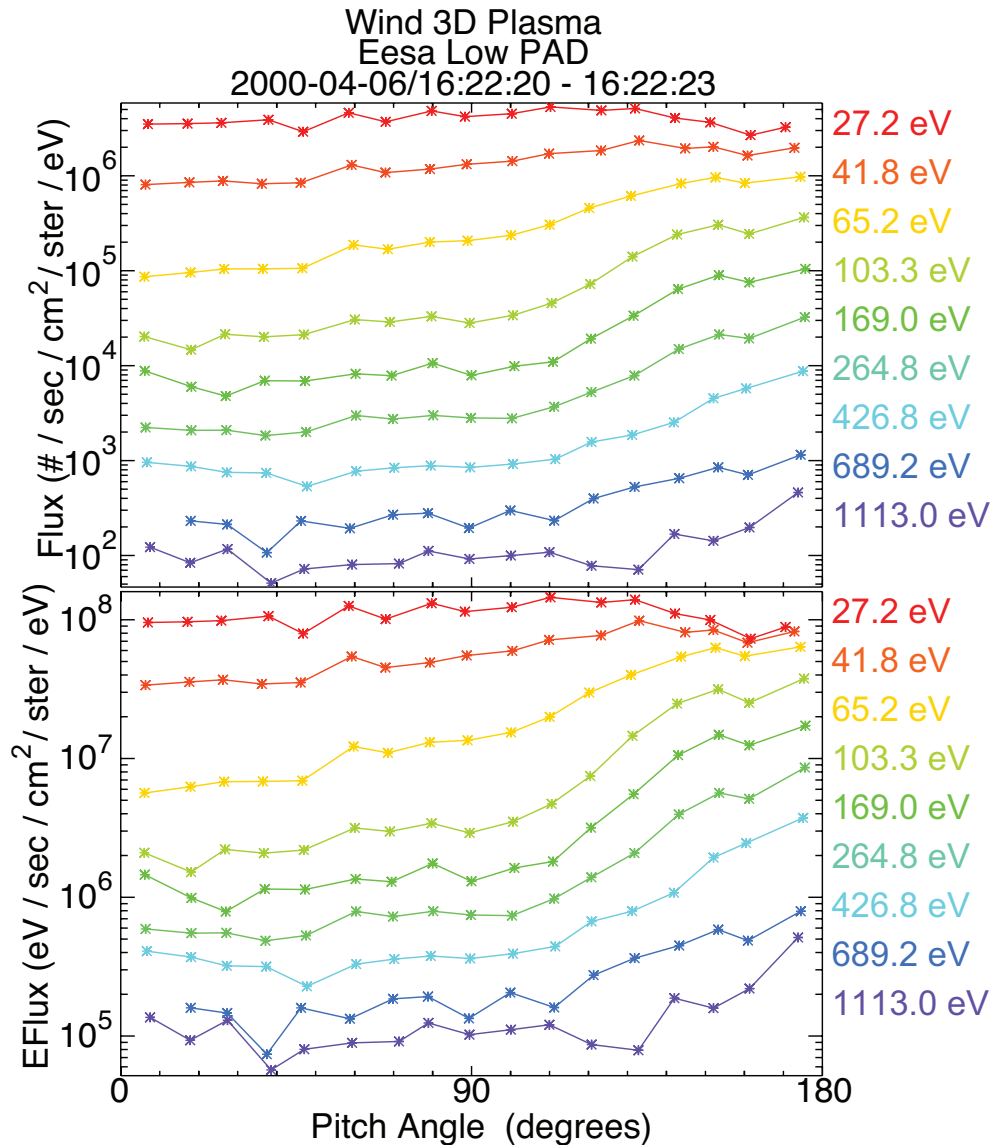


Figure B: This is an example image produced by `my_padplot_both.pro` using `UNITS='flux'` for an EESA Low data sample. The top panel is a stacked line plot of the pitch-angle distributions at the 9 highest energies in units of number flux. The bottom panel is the same thing except in units of energy flux. The vertical axes are logarithmically scaled while the horizontal axes range from  $0^\circ$  to  $180^\circ$ .

To plot the PADs in a useful manner, one can use either `padplot.pro` or `my_padplot_both.pro`. The former will produce a single plot of the PAD with in one set of units while the latter will produce two plots as seen in Figure B. To produce the PAD plot, do the following:

```
UMN> my_padplot_both,pd,UNITS='flux'EBINS=[0L,8L]
```

The other detector in which one might wish to view the distribution functions of is PH. The manner in which the DF is calculated for PH is slightly different and somewhat more complicated due to a number of sources of noise etc. Regardless, assume you have already retrieved particle structures for PH in a similar manner to the one used to get the EL structures above<sup>18</sup>, then do the following:

```
;; Get all the PHB data in time range
UMN> phbdat = get_3dp_structs('phb',TRANGE=trange)
UMN> aphb   = phbdat.DATA
;; Select the 10th structure [arbitrary choice here]
UMN> dat    = aphb[10]
;; Define number of grids to use in contour
UMN> ngrid  = 20L
;; Define the velocity range of the plots
UMN> vlim   = 25e2
UMN> sunv   = [1.,0.,0.]
UMN> sunn   = 'Sun Dir.'
;; Define a circle of constant energy at 780 km/s
UMN> vcirc  = 780.
;; Define vectors that will define coordinate basis for plots
UMN> vec1   = dat.MAGF
UMN> vec2   = dat.VSW
;; Define names to use for those vectors
UMN> xname  = 'B!Do!N'
UMN> yname  = 'V!Dsw!N'
;; Define plane of projection to use
UMN> plane  = 'xy';; [B x (V x B)] vs. B
;; Non-smoothed DF cuts
UMN> contour_3d_1plane,dat,vec1,vec2,VLIM=vlim,NGRID=ngrid,XNAME=xname, $
      YNAME=yname,PLANE=plane[0],/NO_REDF,VCIRC=vcirc
;; Add one-count level DF parallel cut and smooth
UMN> ns     = 4L
UMN> smc    = 1
UMN> smct   = 1
UMN> contour_3d_1plane,dat,vec1,vec2,VLIM=vlim,NGRID=ngrid,XNAME=xname, $
      YNAME=yname,SM_CUTS=smc,NSMOOTH=ns,PLANE=plane[0], $
      SM_CONT=smct,/NO_REDF,/ONE_C,VCIRC=vcirc
```

These should produce a plot like the one seen in Figure C, with slight variations for different keywords<sup>19</sup>, of course. The format of Figure C is similar to that of Figure A. For more examples, see the crib sheets found in the `~/wind.3dp_pros/wind.3dp_cribs` directory. For examples used in the literature, see *Wilson III et al.* [2009, 2010, 2012, 2013a,b].

<sup>18</sup>Use PHB and short time ranges here to avoid multiple mapcodes

<sup>19</sup>For instance, the axis labels should be different from that shown in Figure C because this is just an old example that was modified using Adobe Illustrator. The figures will not keep up with the software updates, just as this tutorial may be slightly behind the software. Please bear with me as significant changes to documents like this one take much more time than quick bug fixes in my software.

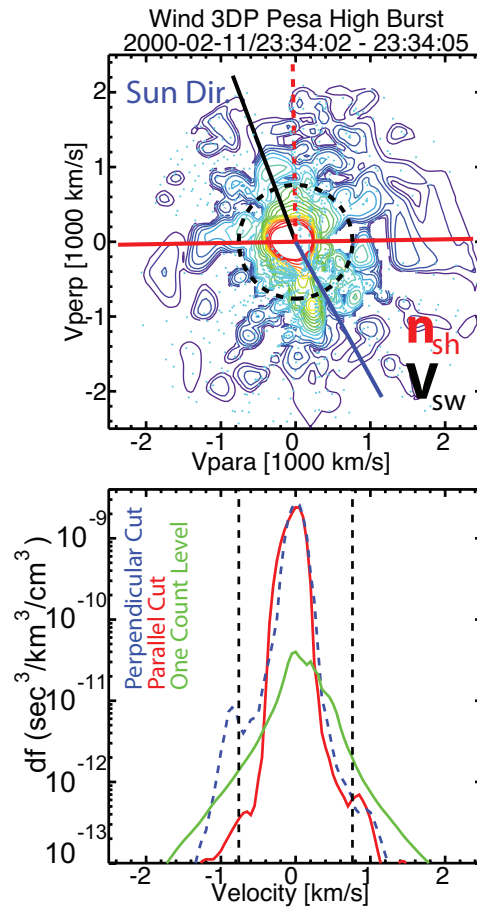


Figure C: This is an example image produced by `eh_cont3d.pro`<sup>20</sup> of an PESA High Burst data sample. The format is similar to that of Figure A except that this does not assume gyrotropy. Also shown are the projections of the shock normal direction (dashed red line), shock surface (solid red line), sun direction (solid blue line), solar wind vector (solid black line), and a circle of constant speed at  $\sim 780$  km/s (dashed circle). The bottom panels show the cuts of the distribution function parallel (red line) and perpendicular (blue line) to the magnetic field. The one-count level (green line) is shown for reference. The small blue dots in the contour plot represent the locations of the actual data.

## 6.2 Particle Spectra Data

Often times one would like to examine the particle moments in a different manner than the two examples shown above. For instance, the SST detectors are used to detect higher energy particles than the ES analyzers and thus one would expect lower fluxes, among other things, from SST data. An examination of the particle data from the SST detectors in the contour plot formats seen above can be difficult and impossible at times (*i.e.* IDL contours won't close if there aren't enough data points resulting in artificial data spikes in the DF plots). A useful method of analysis is the examination of the particle spectra plots. One can produce spectra plots in the following manner:

```
UMN> sfspec = calc_padspecs('sf',TRANGE=tr,NUM_PA=num_pa)
```

after one has loaded 3DP data for the time range, *tr*, of course. This will produce a number of TPLOT variables which can be viewed in the typical fashion. The TPLOT names with `'*-i-j:2*'` in their names are stacked spectra plots split up by the pitch-angles defined by the indices *i* and *j*. The angles, with respect to the magnetic field, associated with these indices are defined by the program ***pangle.pro*** called in ***pad.pro***, called by ***get\_padspecs.pro***<sup>21</sup> which was called inside of ***calc\_padspecs.pro***. The number of different pitch-angle bins is controlled by the keyword *NUM\_PA*. The data can also be *cleaned*, *shifted*, and/or *normalized*. Each of these options is allowed in the program ***calc\_padspecs.pro*** or one can do this after spectra is produced.

When looking at SST data, one often wants the averaged properties, NOT necessarily the instantaneous individual particle structure results. For instance, the top two panels in Figure 1 of *Ergun et al.* [1998] show examples of the stacked particle plots I keep referring to. The plots correspond to smoothed versions of the first TPLOT variable listed (for each instrument) produced by ***calc\_padspecs.pro***<sup>22</sup>. The top panel shows stacked spectra, separated by energy bin, for the EESA High detector and the bottom shows a similar plot for the SST Foil detector. In TPLOT, these lines will each have their own color for ease of distinguishing between different energy bin values.

To *clean*, *shift*, or *normalize* the data using ***calc\_padspecs.pro***, one need only use the keywords *DAT\_CLN*, *DAT\_SHFT*, or *DAT\_NORM* respectively. For instance, to produce *cleaned* SST Foil spectra, do the following:

```
UMN> sfspec = calc_padspecs('sf',TRANGE=tr,NUM_PA=num_pa,/DAT_CLN)
```

and you'll notice that new TPLOT names exist with the suffix `'*_cln'`. A similar results with the other two keywords but the suffixes change to `'*_sh'` for shifted data and `'*_n'` for normalized data. The programs used to produce the *cleaned*, *shifted*, and/or *normalized* data plots are ***clean\_spec\_spikes.pro*** and ***spec\_vec\_data\_shift.pro***. Each program has a semi-useful man page allowing you to examine what each keyword and input parameter should be to produce the desired results. Also, one can remove energy bins if the energy bin seems to contain only noise or bad data, thus corrupting and/or ruining the plot in general using ***energy\_remove\_split.pro***.

I will briefly show some examples of how one can use these programs to manipulate and alter the spectra data. The first thing to do is clean up the data spikes and empty regions using ***clean\_spec\_spikes.pro***. Let's assume the TPLOT name of the base particle spectra data is `nsf_pads`. The SST Foil detector has seven energy bins and often times I use only eight pitch-angle bins for both SST detectors at most. Notice that if you got the data for `nsf_pads` by doing:

```
UMN> get_data,'nsf_pads',DATA=sfpads
```

and looked at the data structure by typing:

```
UMN> HELP, sfpads, \STRUCT,OUTPUT=hout
UMN> PRINT,hout,FORMAT='(";","a)'
```

<sup>21</sup>this is an adaptation of ***get\_padspec.pro*** but more robust, faster, and more user friendly (*i.e.* many more comments)

<sup>22</sup>more likely the smoothed version of the TPLOT variable produced by the original ***get\_padspec.pro***

```

;** Structure <213cc10>, 8 tags, length=11525052, data length=11525050, refs=1:
;  YTITLE      STRING      'SF Flux!C(# cm!U-2!N s!U-1!N sr!U-1!N eV!U-1!N)'
;  X           DOUBLE      Array[39469]
;  Y           FLOAT       Array[39469, 7, 8]
;  V1          FLOAT       Array[39469, 7]
;  V2          FLOAT       Array[39469, 8]
;  YLOG        INT         1
;  LABELS      STRING      Array[7]
;  PANEL_SIZE  FLOAT       2.00000

```

where the number **39469** refers to the number of different particle structures (*i.e.* time steps) in the data, **7** is the number of energy bins, and **8** is the number of pitch-angles. Note that the TPLOT structure above is a standard format for 3-dimensional data arrays depending on time and two other quantities. The tags *X* and *Y* are the Unix time and data, respectively, as usual and the tags *V1* and *V2* refer to the particle energies and pitch-angles, respectively. The plot resulting from these would be an average over *V2*, thus an effective omni-directional spectra. If we look at a TPLOT variable which as already been split up by the programs *reduce\_pads.pro* and *reduce\_dimen.pro*, we'll find a slightly different structure format. To do so, do the following:

```

UMN> get_data,'nsf_pads-2-0:1',DATA=sfpd01
UMN> HELP, sfpd01, \STRUCT,OUTPUT=hout
UMN> PRINT,hout,FORMAT='(";",a)'

```

```

;** Structure <1dc3500>, 3 tags, length=2526016, data length=2526016, refs=1:
;  X           DOUBLE      Array[39469]
;  Y           FLOAT       Array[39469, 7]
;  V           FLOAT       Array[39469, 7]

```

where only the tag *V* now exists in place of *V1* and *V2* corresponding to the energies for each element of *Y*. So now let's smooth out the data spikes in *nsf\_pads* by smoothing over 7 points in the lower energies and 10 points in the two highest energies by doing:

```

UMN> oldnn = 'nsf_pads'
UMN> newnn = 'nsf_pads_cln'
UMN> nsmth = 7
UMN> esmth = [0L,1L,10L]
UMN> clean_spec_spikes,oldnn,NEW_NAME=newnn,NSMOOTH=nsmth,ESMOOTH=esmth

```

which will produce a new TPLOT variable named *nsf\_pads\_cln*. To shift the four highest energies of the data, do the following:

```

UMN> newnn = 'nsf_pads_sh'
UMN> spec_vec_data_shift,oldnn,NEW_NAME=newnn,WSHIFT=[0L,3L]

```

and to shift AND normalize the data, do the following:

```

UMN> newnn = 'nsf_pads_sh_n'
UMN> spec_vec_data_shift,oldnn,NEW_NAME=newnn,/DATS,/DATN

```

Let's say that the two highest energy bins appear to be only noise and they are skewing your Y-Axis limits making the plot impossible to read. To remove these two energy bins, do the following:

```

UMN> newnn = 'nsf_pads_5-Lowest-E'
UMN> energy_remove_split,oldnn,NEW_NAME1=newnn,ENERGIES=[0,1]

```

Notice that the program redetermines the colors, TPLOT labels, and the vertical axis range for you. If the two highest energies weren't noise, but they appeared to be interesting also, we could split the energy ranges into low and high by doing the following:

```
UMN> newn1 = 'nsf_pads_5-Lowest-E'  
UMN> newnh = 'nsf_pads_2-Highest-E'  
UMN> energy_remove_split,oldnn,NEW_NAME1=newn1,NEW_NAME2=newnh,ENERGIES=[0,1],/SPLIT
```



## A IDL Implementation: Tensor Rotations and Manipulations

To begin with, I will illustrate the tensor and array notation used in IDL to help the user become familiar with the calculations being done in the programs used. If you aren't concerned with these particulars, you can skip to the next section for the IDL commands. The two programs which will calculate the heat flux vector and tensor from a given 3DP particle distribution structure are *mom\_sum.pro* and *mom\_translate.pro*. The programs were initially uncommented and impossibly opaque. I have since gone through and commented and/or explained certain aspects of these programs which should help a well prepared expert use them. However, a beginning graduate student might find them far beyond what they deem themselves capable of understanding. So I will try and help illustrate, in this section, the more complicated and semi-opaque steps one finds in these two routines. Much of *mom\_sum.pro* is commented which makes it pretty straight forward. The operations in *mom\_translate.pro* become remarkably complicated in the sense that comments might require more space than is *ethically* allowed under decent programming standards. So I'll try and illuminate as many of these opaque and *messy* steps as possible.

The first thing one should notice are the structure tags labeled MAP\_\* in the beginning segment of the *mom\_sum.pro*. These are simply indexing arrays used to map a one dimensional array to a  $3 \times 3$ ,  $3 \times 3 \times 3$ , or  $3 \times 3 \times 3 \times 3$  array (or vice versa). Notice many of the numbers in each of these arrays are repeated, which results because of assumed symmetries. To view this in another way, I'll illustrate with some examples. To start, try the following in IDL:

```
UMN> y = STRARR(3,3)
UMN> vec = ['x','y','z']
UMN> FOR i=0L, 2L DO BEGIN $
UMN>   FOR j=0L, 2L DO BEGIN $
UMN>     y[j,i] = vec[j]+vec[i]
UMN> PRINT, y
xx yx zx
xy yy zy
xz yz zz
UMN> vstr = STRARR(3)
UMN> FOR i=0L, 2L DO BEGIN $
UMN>   vstr[i] = 'v'+vec[i]
UMN> PRINT, vstr
vx vy vz
```

As one can see, the string array *y* is a  $3 \times 3$  matrix/array that has the elements shown just below the command PRINT, *y*. Now IDL won't let us do matrix multiplication on string arrays, so do the following:

```
UMN> x = FINDGEN(9) + 1.
UMN> v_test = [0.,10.,100.]
UMN> PRINT, x # v_test,FORMAT='(9f8.1)'
    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
   10.0   20.0   30.0   40.0   50.0   60.0   70.0   80.0   90.0
  100.0  200.0  300.0  400.0  500.0  600.0  700.0  800.0  900.0 .
```

Now one of the steps in *mom\_translate.pro* takes a result similar to this and changes it into a  $3 \times 3 \times 3$ . One can do this in the following manner:

```
UMN> PRINT, REFORM(y,9)
xx yx zx xy yy zy xz yz zz
```

which simply illustrates how to use the IDL built-in routine, REFORM.PRO. Now to use this on the array, *x*, in the manner proposed, do the following:

```
UMN> PRINT, REFORM(x # v_test,3,3,3)
      0.00000      0.00000      0.00000
      0.00000      0.00000      0.00000
      0.00000      0.00000      0.00000

      10.0000     20.0000     30.0000
      40.0000     50.0000     60.0000
      70.0000     80.0000     90.0000

      100.000     200.000     300.000
      400.000     500.000     600.000
      700.000     800.000     900.000 .
```

Thus one can see that the first row of the resultant array of `x # v_test` is now a  $3 \times 3$  array in a  $3 \times 3 \times 3$  array. It is worth noting that were one to take any 3-element array, call it `z` for now, and then define the following:

```
UMN> outer_product ≡ (z # z)
```

one would simply be defining the outer product of a vector with itself. The IDL operator `#` computes a matrix operation by taking the columns of the first array and multiplying them by the rows of the second array. The next step is somewhat complicated. Let the variable `zz` be defined as:

```
UMN> zz = REFORM(x # v_test,3,3,3)
```

and let `zz1` be defined as:

```
UMN> zz1 = TRANSPOSE(zz,[1,2,0]) .
```

The result of this operation is to shift the rows of the variable `zz` such that the first column of each  $3 \times 3$  array in the  $3 \times 3 \times 3$  array `zz` is the first rows of each  $3 \times 3$  array in the  $3 \times 3 \times 3$  array `zz1`. To illustrate,

```
UMN> PRINT, zz1
      0.00000      0.00000      0.00000
      10.0000     40.0000     70.0000
      100.000     400.000     700.000

      0.00000      0.00000      0.00000
      20.0000     50.0000     80.0000
      200.000     500.000     800.000

      0.00000      0.00000      0.00000
      30.0000     60.0000     90.0000
      300.000     600.000     900.000
```

which is, perhaps, more easily seen if we use our string arrays `y` and `vstr` above to illustrate (recall that IDL won't let you do these operations on string arrays so I am merely doing this by hand as an example<sup>23</sup>):

```
UMN> PRINT, REFORM(y,9) # vstr
(xx vx) (yx vx) (zx vx) (xy vx) (yy vx) (zy vx) (xz vx) (yz vx) (zz vx)
(xx vy) (yx vy) (zx vy) (xy vy) (yy vy) (zy vy) (xz vy) (yz vy) (zz vy)
```

<sup>23</sup>I've artificially inserted the parentheses also for clarity.

```
(xx vz) (yx vz) (zx vz) (xy vz) (yy vz) (zy vz) (xz vz) (yz vz) (zz vz)
```

```
UMN> PRINT, REFORM(REFORM(y,9) # vstr,3,3,3)
```

```
(xx vx) (yx vx) (zx vx)
(xy vx) (yy vx) (zy vx)
(xz vx) (yz vx) (zz vx)
```

```
(xx vy) (yx vy) (zx vy)
(xy vy) (yy vy) (zy vy)
(xz vy) (yz vy) (zz vy)
```

```
(xx vz) (yx vz) (zx vz)
(xy vz) (yy vz) (zy vz)
(xz vz) (yz vz) (zz vz)
```

```
UMN> PRINT, TRANSPOSE(REFORM(REFORM(y,9) # vstr,3,3,3),[1,2,0])
```

```
(xx vx) (xy vx) (xz vx)
(xx vy) (xy vy) (xz vy)
(xx vz) (xy vz) (xz vz)
```

```
(yx vx) (yy vx) (yz vx)
(yx vy) (yy vy) (yz vy)
(yx vz) (yy vz) (yz vz)
```

```
(zx vx) (zy vx) (zz vx)
(zx vy) (zy vy) (zz vy)
(zx vz) (zy vz) (zz vz)
```

The next rotation is done in a similar manner as shown in the following:

```
UMN> PRINT, TRANSPOSE( REFORM( REFORM(y,9) # vstr,3,3,3),[2,0,1])
```

```
(xx vx) (xx vy) (xx vz)
(yx vx) (yx vy) (yx vz)
(zx vx) (zx vy) (zx vz)
```

```
(xy vx) (xy vy) (xy vz)
(yy vx) (yy vy) (yy vz)
(zy vx) (zy vy) (zy vz)
```

```
(xz vx) (xz vy) (xz vz)
(yz vx) (yz vy) (yz vz)
(zz vx) (zz vy) (zz vz)
```

The next two examples are not entirely straight forward, so I'll try to take them in multiple sub-steps to help illustrate what is going on. A new variable is defined in *mom\_translate.pro* which is used to transform the data into a new coordinate system. The command used is `REFORM(REFORM(v#v,9) # v,3,3,3)` where *v* can be represented by our variable *vstr*. As we saw before, the outer product of our vector, *vstr*, would give:

```
(vx vx) (vy vx) (vz vx)
(vx vy) (vy vy) (vz vy)
(vx vz) (vy vz) (vz vz)
```

which can be reformed in the manner outlined above into a 9-element array given by:

(vx vx) (vy vx) (vz vx) (vx vy) (vy vy) (vz vy) (vx vz) (vy vz) (vz vz)

then using the operator # to multiply these values by the original *vstr* (i.e. `REFORM(v#v,9) # v`) gives:

(vx vx vx) (vy vx vx) (vz vx vx) (vx vy vx) (vy vy vx) (vz vy vx) (vx vz vx) (vy vz vx) (vz vz vx)  
 (vx vx vy) (vy vx vy) (vz vx vy) (vx vy vy) (vy vy vy) (vz vy vy) (vx vz vy) (vy vz vy) (vz vz vy)  
 (vx vx vz) (vy vx vz) (vz vx vz) (vx vy vz) (vy vy vz) (vz vy vz) (vx vz vz) (vy vz vz) (vz vz vz)

which can be reformed into a 3×3×3 array by doing:

```
UMN> rstr = REFORM(REFORM(v#v,9) # v,3,3,3)
UMN> PRINT, rstr
(vx vx vx) (vy vx vx) (vz vx vx)
(vx vy vx) (vy vy vx) (vz vy vx)
(vx vz vx) (vy vz vx) (vz vz vx)

(vx vx vy) (vy vx vy) (vz vx vy)
(vx vy vy) (vy vy vy) (vz vy vy)
(vx vz vy) (vy vz vy) (vz vz vy)

(vx vx vz) (vy vx vz) (vz vx vz)
(vx vy vz) (vy vy vz) (vz vy vz)
(vx vz vz) (vy vz vz) (vz vz vz).
```

The values are then subtracted from the original mapped data, re-mapped and returned to the user in the following manner:

```
UMN> v = mom.NV/mom.N ;=> Velocity [eV(1/2)]
UMN> mt = mom.NVV[mom.MAP_R2] ;=> [eV(1/2) km/s cm(-3)]
UMN> pt = mt - mom.N*(v # v)
UMN> mom.NVV = pt[mom.MAP_V2] ;=> [eV(1/2) km/s cm(-3)]
UMN> nvvv = mom.NVVV[mom.MAP_R3]
UMN> up0 = REFORM(REFORM(pt,9) # v,3,3,3)
UMN> up1 = TRANSPOSE(up0,[1,2,0])
UMN> up2 = TRANSPOSE(up0,[2,0,1])
UMN> nuuu = mom.N * REFORM(REFORM(v#v,9) # v,3,3,3)
UMN> qt = nvvv - (up0 + up1 + up2 + nuuu)
UMN> mom.NVVV = qt[mom.MAP_V3]
```

where `mom.NVVV` is the third rank tensor as a 3×3×3 array, which after a few unit conversion factors ( $M_s * n_s^2$ ) results in the heat flux tensor.

## B The Pesa High “Glitch”

On occasion, the PH detector has a *glitch* in the data which always occurs in the same data bins, regardless of data/time. I use the term *glitch*, though the bad bins result from things which are not likely due to an electronic glitch or anything like that. In fact, the consistency of the bin numbers being affected and their relation to the sun direction suggests that the issue is more likely due to UV contamination than an electronic glitch. There are also issues with saturation in the ecliptic plane due to the detectors large geometry factor and the high flux rate of the solar wind. The energy bins show a pattern suggesting these bins may relate to the bins in the double-sweep mode, since the energies go from  $\sim 28$  keV down to  $\sim 5$  keV then start over again. Regardless, the counts associated with these data bins are clearly not physical as they often exceed 2-3 orders of magnitude above the counts in the rest of the detector bins<sup>24</sup>. If the data structures are retrieved on a Sun Machine, the error in the energy bin values will occur as a NaN and the corresponding data values will be too large to be physically reliable. If the data structures are retrieved on a Mac, the bad values turn into very large numbers. The issue is results from the definition of big and little endian for the Mac Intel and Sun Machine shared object libraries.

Table I: Pesa High Bad Data Bins

Mapcode (Hex)	Mapcode (Long)	# of Bins	Bin Elements
D4A4	54436	121	[0,1,8,9,16,17,24,25,35,36,43,44,51,52,59,60]
D4FE	54526	97	[0,1,5,6,10,11,15,16,23,24,28,29,33,34,38,39]
D5EC	54764	56	[0,1,2,8,9,10,19,20,21,27,28,29]
D6BB	54971	65	Unknown
D65D	54877	88	Unknown

The glitches occur consistently in the same data bins, regardless of day or year. The bins, however, depend upon the mode that the Pesa High detector is in. I have found the following bins to be an issue for the sample modes seen in Table I. The only two modes I have yet to find a *glitch* in are the two where the number of bins equals 65 or 88. This is largely due to the fact that I almost never find Pesa High in a sample mode with these number of data bins.

I have written two programs to help deal with this issue, **pesa\_high\_bad\_bins.pro** and **pesa\_high\_str\_fill.pro**. Both programs attempt to handle this issue and they are called by multiple other programs in my list of added libraries to the SSL software.

### B.1 Finding the Pesa High “Glitch” in IDL

The glitches can be found easily enough with rather little effort. To do so, read in an array of PH or PHB data structures. Then use the following commands in IDL to look for issues:

```
UMN> twph = get_ph([unix time of interest])
UMN> nbins = twph.NBINS
UMN> energy = twph.ENERGY*1d-3 ;=> (keV)
UMN> mform = '(";",I3.3," : ",f6.2,8f9.2)'
UMN> FOR j=0L, nbins - 1L DO BEGIN $
UMN>   jstr = STRTRIM(STRING(FORMAT=mform,j,energy[0L:8L,j]),2L) & $
UMN>   IF (energy[8L,j] GT 4e0) THEN PRINT, jstr
```

which results in the following output:

```
;000 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;001 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;008 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;009 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;016 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;017 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
```

<sup>24</sup>Perhaps this is a combination of UV contamination and the high solar wind flux.

```
;024 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;025 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;035 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;036 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;043 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;044 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;051 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;052 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;059 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43
;060 : 28.43 21.15 15.73 11.70 8.70 6.47 4.82 NaN 28.43 .
```

Notice that the seventh (indexing from zero since in IDL) element is *Not A Number* or *NaN*. The eighth element seems to start back at the highest energy and repeat the pattern, as seen below with the following commands:

```
UMN> twph = get_ph([unix time of interest])
UMN> nbins = twph.NBINS
UMN> energy = twph.ENERGY*1d-3 ;=> (keV)
UMN> mform = '(",",I3.3," : ",f6.2,5f9.2)'
UMN> FOR j=0L, nbins - 1L DO BEGIN $
UMN>   jstr = STRTRIM(STRING(FORMAT=mform,j,energy[9L:14L,j]),2L) & $
UMN>   IF (energy[8L,j] GT 4e0) THEN PRINT, jstr
```

which results in the following output:

```
;000 : 21.15 15.73 11.70 8.70 6.47 4.82
;001 : 21.15 15.73 11.70 8.70 6.47 4.82
;008 : 21.15 15.73 11.70 8.70 6.47 4.82
;009 : 21.15 15.73 11.70 8.70 6.47 4.82
;016 : 21.15 15.73 11.70 8.70 6.47 4.82
;017 : 21.15 15.73 11.70 8.70 6.47 4.82
;024 : 21.15 15.73 11.70 8.70 6.47 4.82
;025 : 21.15 15.73 11.70 8.70 6.47 4.82
;035 : 21.15 15.73 11.70 8.70 6.47 4.82
;036 : 21.15 15.73 11.70 8.70 6.47 4.82
;043 : 21.15 15.73 11.70 8.70 6.47 4.82
;044 : 21.15 15.73 11.70 8.70 6.47 4.82
;051 : 21.15 15.73 11.70 8.70 6.47 4.82
;052 : 21.15 15.73 11.70 8.70 6.47 4.82
;059 : 21.15 15.73 11.70 8.70 6.47 4.82
;060 : 21.15 15.73 11.70 8.70 6.47 4.82 .
```

The glitch appears always at the same azimuthal angles,  $\phi$  (degrees), in the Pesa High data structures. To find them, follow a similar procedure as outlined above, but change *energy* to *phi* in the variable definition of *jstr*. The angles are as follows:

```
;000 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;001 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;002 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;008 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;009 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;010 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;019 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;020 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
```

```
;021 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;027 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;028 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;029 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
```

for a case with 56 data bins on 04/03/1996,

```
;000 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;001 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;005 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;006 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;010 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;011 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;015 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;016 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;023 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;024 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;028 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;029 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;033 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;034 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;038 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;039 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
```

for a case with 97 data bins on 11/24/2001, and for a case with 121 data bins on 04/06/2000,

```
;000 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;001 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;008 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;009 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;016 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;017 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;024 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;025 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;035 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;036 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;043 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;044 : 179.62 178.88 178.12 177.38 176.62 175.88 175.12 174.38 173.62
;051 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;052 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;059 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88
;060 : 190.88 190.12 189.38 188.62 187.88 187.12 186.38 185.62 184.88 .
```

In each case we see a repeating pattern, as with the data bins, consistent with the spacecraft rotation rate remaining approximately the same, within the angular resolution of the Pesa High detector.

## C Appendix: Conversion Factors and Constants

There are a number of complicated and unexplained numerical constants in the programs *mom\_sum.pro*, *mom\_translate.pro*, and *mom3d.pro* which are neither explained nor defined by a variable. This appendix is an attempt to help illustrate and enlighten those who are unfortunate enough to have access only to the uncommented code.

The first thing to note is that the number associated with the tag name *MASS* in any 3DP data structure appears unrecognizable to most. The number derives from the mass of the particle in units of  $eV/c^2$  and the value of  $c^2$  (speed of light squared) is in  $\text{km}^2/\text{s}^2$  as illustrated in Equation 2a for an electron:

$$mass = \frac{510,990.6 \text{ eV}/c^2}{(2.99792458 \times 10^5 \text{ km/s})^2} \quad (2a)$$

$$= 5.6967578 \times 10^{-6} \text{ eV}/(\text{km/s})^2 . \quad (2b)$$

The next important unreferenced number is used in *mom\_sum.pro* and *mom3d.pro* to deal with particle charges. The number is:

$$0.010438871 \frac{\text{eV}}{(\text{km/s})^2} = \frac{938.27231 \times 10^6 \text{ eV}/c^2}{(2.99792458 \times 10^5 \text{ km/s})^2} \quad (3)$$

which is used for determining the fractional charge (units of proton charge as  $\text{eV}/(\text{km/s})^2$ ) of the input particle type. For electrons, the rounded value of the particle mass over this charge conversion constant is zero, thus the program assigns a value of -1 to the variable *charge*. The normalization factor used to convert the zeroth moment to a density is defined by:

$$n_o \equiv \sqrt{\left| \frac{2 * q}{M_s} \right|} = 593.09544 \text{ (for electrons)} \quad (4)$$

where  $q$  is a normalized charge and  $M_s$  is the particle species mass ( $\text{eV}/(\text{km/s})^2$ ).



## References

- Aptekar, R. L., et al. (1995), Konus-W Gamma-Ray Burst Experiment for the GGS Wind Spacecraft, *Space Sci. Rev.*, 71, 265–272, doi:10.1007/BF00751332.
- Bougeret, J.-L., et al. (1995), Waves: The Radio and Plasma Wave Investigation on the Wind Spacecraft, *Space Sci. Rev.*, 71, 231–263, doi:10.1007/BF00751331.
- Desch, M. (2005), Wind: Understanding Interplanetary Dynamics, nASA Goddard Space Flight Center, Online: <http://istp.gsfc.nasa.gov/wind.shtml>.
- Ergun, R. E., et al. (1998), Wind Spacecraft Observations of Solar Impulsive Electron Events Associated with Solar Type III Radio Bursts, *Astrophys. J.*, 503, 435–+, doi:10.1086/305954.
- Feldman, W. C., R. C. Anderson, S. J. Bame, S. P. Gary, J. T. Gosling, D. J. McComas, M. F. Thomsen, G. Paschmann, and M. M. Hoppe (1983), Electron velocity distributions near the earth's bow shock, *J. Geophys. Res.*, 88, 96–110, doi:10.1029/JA088iA01p00096.
- Gloeckler, G., et al. (1995), The Solar Wind and Suprathermal Ion Composition Investigation on the Wind Spacecraft, *Space Sci. Rev.*, 71, 79–124, doi:10.1007/BF00751327.
- Kasper, J. C. (2007), Interplanetary Shock Database, harvard-Smithsonian Center for Astrophysics, Online: <http://www.cfa.harvard.edu/shocks/>.
- Lepping, R. P., et al. (1995), The Wind Magnetic Field Investigation, *Space Sci. Rev.*, 71, 207–229, doi:10.1007/BF00751330.
- Lin, R. P., et al. (1995), A Three-Dimensional Plasma and Energetic Particle Investigation for the Wind Spacecraft, *Space Sci. Rev.*, 71, 125–153, doi:10.1007/BF00751328.
- McFadden, J. P., et al. (2007), In-Flight Instrument Calibration and Performance Verification, *ISSI Sci. Rep. Ser.*, 7, 277–385.
- Ogilvie, K. W., et al. (1995), SWE, A Comprehensive Plasma Instrument for the Wind Spacecraft, *Space Sci. Rev.*, 71, 55–77, doi:10.1007/BF00751326.
- Owens, A., et al. (1995), A High-Resolution GE Spectrometer for Gamma-Ray Burst Astronomy, *Space Sci. Rev.*, 71, 273–296, doi:10.1007/BF00751333.
- Wilson III, L. B., C. A. Cattell, P. J. Kellogg, K. Goetz, K. Kersten, J. C. Kasper, A. Szabo, and K. Meziane (2009), Low-frequency whistler waves and shocklets observed at quasi-perpendicular interplanetary shocks, *J. Geophys. Res.*, 114, A10106, doi:10.1029/2009JA014376.
- Wilson III, L. B., C. A. Cattell, P. J. Kellogg, K. Goetz, K. Kersten, J. C. Kasper, A. Szabo, and M. Wilber (2010), Large-amplitude electrostatic waves observed at a supercritical interplanetary shock, *J. Geophys. Res.*, 115, A12104, doi:10.1029/2010JA015332.
- Wilson III, L. B., et al. (2012), Observations of electromagnetic whistler precursors at supercritical interplanetary shocks, *Geophys. Res. Lett.*, 39, L08109, doi:10.1029/2012GL051581.
- Wilson III, L. B., et al. (2013a), Electromagnetic waves and electron anisotropies downstream of supercritical interplanetary shocks, *J. Geophys. Res.*, 118, 5–16, doi:10.1029/2012JA018167.
- Wilson III, L. B., et al. (2013b), Shocklets, SLAMS, and field-aligned ion beams in the terrestrial foreshock, *J. Geophys. Res.*, 118, 957–966, doi:10.1029/2012JA018186.
- Wüest, M., Evans, D. S., & von Steiger, R. (Ed.) (2007), *Calibration of Particle Instruments in Space Physics*, ESA Publications Division, Keplerlaan 1, 2200 AG Noordwijk, The Netherlands.