SPEDAS Particle Tools Development Guide

A guide for developers on velocity distribution function tools in SPEDAS
version 1.1

Eric Grimes
(based on the work of many previous developers/scientists)

egrimes@igpp.ucla.edu

Overview

- PGS
- 2D Slices
- ISEE 3D

PGS

Particle Get Spec

PGS plug-ins allow users to generate energy/gyro-phase spectra, PADs, and moments from velocity distribution functions

PGS Plug-in Overview

Routine	Purpose
xxx_part_products	Core routine
xxx_part_getspec	Wrapper for xxx_part_products; loads distribution and support data into tplot variables
xxx_get_yyy_dist	Converts distribution tplot variables into 3D particle structures
xxx_convert_flux_units	Converts distribution (df_cm, df_km) units to/from flux, eflux units
xxx_pgs_clean_data	Reforms items in structure to [energy, angle], strips unneeded items from structure
xxx_pgs_clean_support	Transforms magnetic field to same coordinate system as the distribution, and interpolates B-field, S/C potential, S/C position to the time stamps of the distribution
xxx_pgs_make_fac	Generates the field aligned coordinate transformation matrix

Key mission specific routines

xxx: mission (e.g., thm, mms, erg, etc)

yyy: instrument (e.g., FPI, HPCA, ESA, etc)

xxx_part_getspec: wrapper routine; loads distribution and support data required by xxx_part_products based on the user's keywords, then calls xxx_part_products

↓ calls

xxx_part_products: core routine

↓ calls

xxx_get_yyy_dist: instrument-specific function that returns SPEDAS 3D distribution data structures for calculations

suggestion: start with xxx_part_products and a short crib sheet that works with xxx_part_products to get it to work - then turn that crib sheet into xxx_part_getspec

xxx_part_getspec

(or a short crib sheet to start off with)

```
xxx_part_getspec, trange=['2015-12-15', '2015-12-16'], instrument='yyy', species='i'
```

1) Load ion distribution data for YYY instrument

2) (not needed to start) Load B-field, S/C position, S/C potential, YYY velocity data (all depending on if they're needed in xxx_part_products); B-field, S/C position and YYY velocity data will be required for various FAC transformations; S/C potential data are required for moments

3) Pass required tplot variables and options to xxx_part_products

xxx_part_products

1) Validate input

2) Find timestamps for the requested trange (with xxx_get_yyy_dist); this is so that you can interpolate the support data to the same times as the distribution data

3) Prepare the support data

4) Loop over time to build the spectrograms/moments

5) Create the tplot variables

Before expanding on these, we should introduce the function that turns your distribution data (stored in tplot variables) into the standard 3D data structure used by SPEDAS particle routines: xxx_get_yyy_dist

xxx_get_yyy_dist

```
dist = xxx_get_yyy_dist(tname [,index] [,/times] [,/structure])
```

Input:

- tname: Tplot variable containing the distribution data
- index: Index of time sample to return
- times: Flag to return full array of times
- structure: Flag to return a structure array instead of a pointer

note: if ‘index’ parameter isn’t specified, all distribution data should be returned (this will allow the routine to be used with our spd_slice2d tools as well)

Output (returns):

By default: pointer to structure containing 3D distribution data and metadata (more on this next)

If /structure keyword is specified: the above structure itself without a pointer (less efficient)

If /times keyword is specified: array of time samples (the X values from tname) - for (2) on previous slide

3D Data Structure for Particle Routines

Example from FAST FPI L2 (all probably required):

```
PROJECT_NAME     STRING    'MMS'
SPACECRAFT       STRING    '1'
DATA_NAME        STRING    'FPI Electron'
UNITS_NAME       STRING    'df_cm'
UNITS_PROCEDURE  STRING    ''  ---------> only placeholder required
SPECIES          STRING    'e'
VALID            BYTE         1 ---------> only placeholder required
CHARGE           FLOAT          -1.00000
MASS             FLOAT       5.68566e-06 -----------> eV/(km/s)^2
TIME             DOUBLE       1.4501376e+09
END_TIME         DOUBLE       1.4501376e+09
DATA             FLOAT     Array[32, 32, 16]
BINS             FLOAT     Array[32, 32, 16]---------> bins are 0 (not valid)
                                                        or 1 (valid)
ENERGY           FLOAT     Array[32, 32, 16]
DENERGY          FLOAT     Array[32, 32, 16]
NENERGY          LONG                    32
NBINS            LONG                   512 ---> per energy, azimuth*elevation
PHI              FLOAT     Array[32, 32, 16]
DPHI             FLOAT     Array[32, 32, 16]
THETA            FLOAT     Array[32, 32, 16]  ------> [32, 32, 16] is:
DTHETA           FLOAT     Array[32, 32, 16]          [energy, azimuth, elevation]
```

See mms_get_fpi_dist.pro for full example of FPI on MMS

3D Data Structure for Particle Routines

Important Note!

SPEDAS uses presumed particle trajectories; if angles are stored in look direction of instrument, they'll need to be converted, e.g.,

```
dist.phi = (dist.phi + 180) mod 360
dist.theta = -dist.theta
```

xxx_part_products

1) ~~Validate input~~

2) ~~Find timestamps for the requested trange (with xxx_get_yyy_dist); this is so that you can interpolate the support data to the same times as the distribution data~~

3) Prepare the support data

4) Loop over time to build the spectrograms/moments

5) Create the tplot variables

Prepare Support Data

1) Generate the field aligned coordinate transformation matrix
2) Transform and interpolate magnetic field data
3) Interpolate spacecraft potential data
4) Prepare additional support data (e.g., photoelectron correction models, …)

See mms_pgs_make_fac.pro/mms_pgs_clean_support.pro for MMS examples

xxx_part_products

1) ~~Validate input~~

2) ~~Find timestamps for the requested trange (with xxx_get_yyy_dist); this is so that you can interpolate the support data to the same times as the distribution data~~

3) ~~Prepare the support data~~

4) Loop over time to build the spectrograms/moments

5) Create the tplot variables

Loop over time to build the spectrograms/moments

1) Return the structure for each time index, e.g.,

```
dist = xxx_get_yyy_dist(tvarname, time_idx[i], /structure)
```

2) Convert units (typically defaults to output in eflux); see: mms_convert_flux_units

3) Reform items in dist from [energy, theta, phi] to [energy, angle], then remove unneeded fields from structure; see: mms_pgs_clean_data; return new structure, e.g., clean_dist

(cache clean_dist.bins if FAC requested!)

4) Apply phi, theta, and energy limits (spd_pgs_limit_range)

5) Calculate moments (spd_pgs_moments)

6) Build theta spectrogram (spd_pgs_make_theta_spec)

7) Build phi spectrogram (spd_pgs_make_phi_spec)

8) Build energy spectrogram (spd_pgs_make_e_spec)

(if FAC, PA, or gyro are requested)

9) Perform transformation to FAC, regrid data, and apply limits in new coords (spd_pgs_do_fac, spd_pgs_regrid, spd_pgs_limit_range)

10) Build pitch angle spectrogram (spd_pgs_make_theta_spec)

11) Build gyrophase spectrogram (spd_pgs_make_phi_spec)

12) Build energy spectrogram from field aligned distribution (spd_pgs_make_e_spec)

13) Calculate FAC moments (spd_pgs_moments)

xxx_part_products

1) ~~Validate input~~

2) ~~Find timestamps for the requested trange (with xxx_get_yyy_dist); this is so that you can interpolate the support data to the same times as the distribution data~~

3) ~~Prepare the support data~~

4) ~~Loop over time to build the spectrograms/moments~~

5) Create the tplot variables

Create the tplot variables

1) Create spectrograms with spd_pgs_make_tplot

2) Create moments with spd_pgs_moments_tplot

Extra

Units

flux	# / (cm^2 * s * sr * eV)
eflux	eV / (cm^2 * s * sr * eV)
df_cm	s^3 / cm^6
df_km	s^3 / km^6

Converting the Units

xxx_convert_flux_units
Step 1: determine the scaling factor between DF and flux units

```
;get mass of species
case species_lc of
   'i': A=1;H+
   'proton': A=1;H+
   'hplus': A=1;H+
   'heplus': A=4;He+
   'heplusplus': A=4;He++
   'oplus': A=16;O+
   'oplusplus': A=16;O++
   'e': A=1d/1836;e-
   else: message, 'Unknown species: '+species_lc
endcase

;scaling factor between df and flux units
flux_to_df = A^2 * 0.5447d * 1d6
```

Converting the Units

xxx_convert_flux_units
Step 2: do the calculation

```
;convert between km^6 and cm^6 for df_km
cm_to_km = 1d30

;calculation will be kept simple and stable as possible by
;pre-determining the final exponent of each scaling factor
;rather than multiplying by all applicable in/out factors
;these exponents should always be integers!
;    [energy, flux_to_df, cm_to_km]
in = [0,0,0]
out = [0,0,0]

;get input/output scaling exponents
case units_in of
  'flux': in = [1,0,0]
  'eflux':
  'df_km': in = [2,-1,0]
  'df_cm': in = [2,-1,1]
  'df': message, 'df units no longer supported - use df_km or df_cm instead'
  else: message, 'Unknown input units: '+units_in
endcase

case units_out of
  'flux':out = -[1,0,0]
  'eflux':
  'df_km': out = -[2,-1,0]
  'df_cm': out = -[2,-1,1]
  'df': message, 'df units no longer supported - use df_km or df_cm instead'
  else: message, 'Unknown output units: '+units_out
endcase

exp = in + out

;ensure everything is double prec first for numerical stability
;  -target field won't be mutated since it's part of a structure
output.data = double(dist.data) * double(dist.energy)^exp[0] * (flux_to_df^exp[1] * cm_to_km^exp[2])

output.units_name = strlowcase(units)
```

Reforming the data for calculations

xxx_pgs_clean_data

```
dims = dimen(data.data)

output= {  $
  dims: dims, $
  time: data.time, $
  end_time:data.end_time, $
  charge:data.charge, $
  mass:data.mass,$
  species: data.species, $
  magf:[0.,0.,0.],$
  sc_pot:0.,$
  scaling:fltarr(dims[0],dims[1]*dims[2])+1,$
  units:units,$
  data: reform(data.data,dims[0],dims[1]*dims[2]), $
  bins: reform(data.bins,dims[0],dims[1]*dims[2]), $
  energy: reform(data.energy,dims[0],dims[1]*dims[2]), $
  denergy: reform(data.denergy,dims[0],dims[1]*dims[2]), $
  phi:reform(data.phi,dims[0],dims[1]*dims[2]), $
  dphi:reform(data.dphi,dims[0],dims[1]*dims[2]), $
  theta:reform(data.theta,dims[0],dims[1]*dims[2]), $
  dtheta:reform(data.dtheta,dims[0],dims[1]*dims[2]) $
}
```

xxx_part_getspec standard keywords

Keyword	Purpose
probe/instrument/data_rate/etc	mission specific parameters
trange	time range
outputs	data products to generate, e.g., ’energy’, ‘phi’, ‘theta’, ‘gyro’, ‘pa’ and ‘moments’
units	output units (df_cm, df_km, flux, eflux); defaults to eflux
phi	apply limits to phi angles
theta	apply limits to theta angles
pitch	apply limits to pitch angles
gyro	apply limits to gyro phase angles
regrid	2 element array specifying resampling resolution of FAC distribution
no_regrid	flag to skip regridding step when converting to FAC
datagap	setting for tplot variables, controls how long a gap must be before it is drawn
fac_type	field aligned coordinate system variant. Existing options: ‘phigeo’, ‘mphigeo’, ‘xgse’

2D Slices

2D slice plug-ins allow users to generate/plot slices through velocity distribution functions

2D Slice Plug-in Overview

Routine	Purpose
xxx_part_slice2d	Wrapper routine for spd_slice2d; loads distribution and support data into tplot variables
xxx_get_yyy_dist	Converts distribution tplot variables into 3D particle structures (see PGS section for more details)

xxx_part_slice2d

1) Load distribution and support data

2) Return the structure or array of structures for the requested trange (xxx_get_yyy_dist)

3) Pass the output of (2) and the user's keywords to spd_slice2d; spd_slice2d accepts structures, arrays of structures, or points to either of these

4) (optional) Pass the output of (3) to spd_slice2d_plot to generate the plot (should also allow the user to pass graphics keywords); this is optional - you could also return the slices and allow the user to manually pass them to spd_slice2d_plot (this is how THEMIS works)

See mms_part_slice2d.pro/thm_part_slice2d.pro for examples

ISEE 3D

ISEE 3D plug-ins allow users to visualize the velocity distribution functions in 3D

ISEE 3D Plug-in Overview

Routine	Purpose
xxx_part_isee3d	Wrapper routine for ISEE_3D; loads distribution and support data into tplot variables and converts into ISEE_3D compatible data structures using spd_dist_to_hash
xxx_get_yyy_dist	Converts distribution tplot variables into 3D particle structures (see PGS section for more details)

xxx_part_isee3d

1) Load distribution and support data

2) Return the structure or array of structures for the requested trange (xxx_get_yyy_dist)

3) Pass the output of (2) to spd_dist_to_hash

4) Pass the output of (3) to isee_3d in the data keyword, along with the support data and user's keywords

See mms_part_isee3d.pro for an example