pydivide

Jan 09, 2020

Contents

CHAPTER 1

Introduction

1.1 What is PyDIVIDE?

PyDIVIDE a toolkit for manipulating and visualizing data from the MAVEN mission. The toolkit started in IDL, but the core functionality has been ported over into python. The tool has primarily been focused on the Key Parameter dataset in the past, but due to the limited nature of that dataset will soon focus more on the Level 2 data.

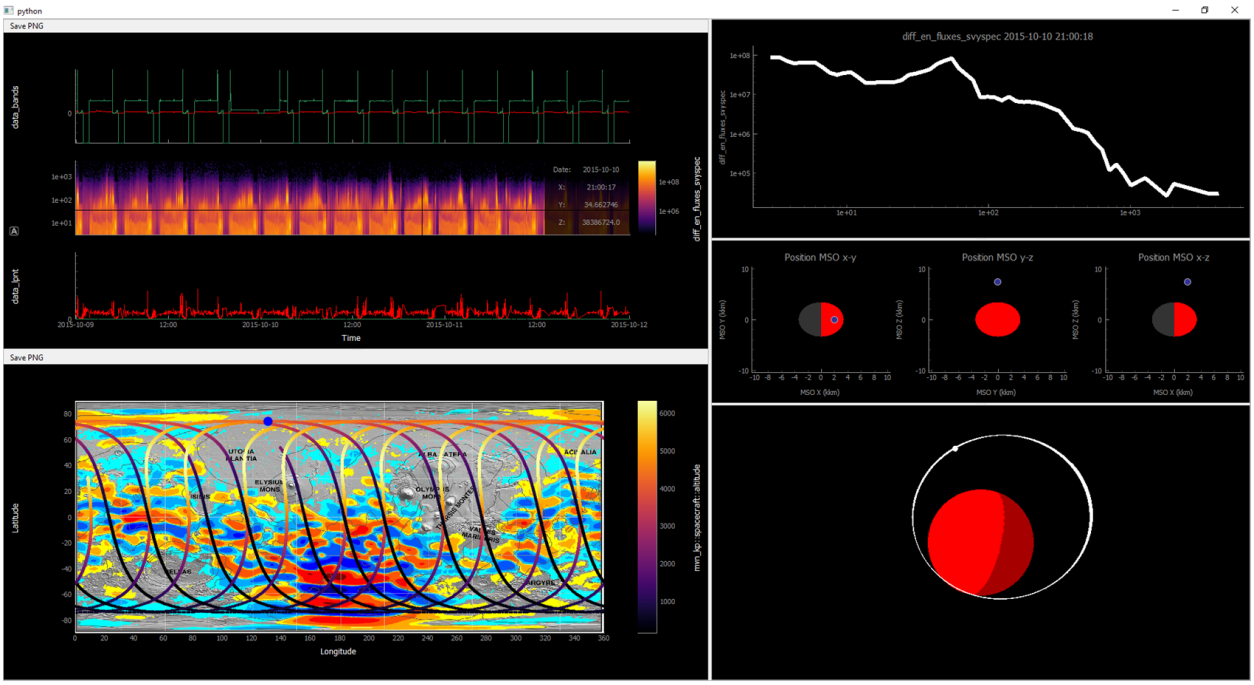
It can plot using either Qt via pytqtgraph, or by creating HTML files via Bokeh.

1.2 What does it do?

There are a few major things the tool does:

- Downloads data files from the MAVEN Science Data Center
- Reads in data from Key Parameter data files
- Plots data in a variety of ways (Time, Altitude, Lat vs Lon, etc)
- Reads in data files from models of the Martian Atmosphere
- Provides time series data analysis/manipulation routines

1.3 Pyqtgraph Sample



CHAPTER 2

Getting Started

2.1 System Requirements

The MAVEN PyDIVIDE toolkit currently requires Anaconda 5.0 or above. Anaconda will install Python, as well as numerous software libraries for scientific computing. This toolkit is only compatible with Python 3.5 or above.

To install the PyDIVIDE toolkit, type the following command into the local terminal/Anaconda Prompt terminal.

    pip install pydivide

This will automatically install most of the required dependencies. The following is necessary for the Bokeh HTML plots:

    conda install -c bokeh nodejs

And finally, though this is not required, for 3D visualization you will need to get PyOpenGL

    pip install pyopengl

The PyDIVIDE toolkit can also be downloaded from the MAVEN Science Data Center GitHub page, https://github.com/MAVENSDC. This will require manual installation of all dependencies of PyDIVIDE. It is recommended that PyDIVIDE be installed via the pip command above.

2.2 Updating the Toolkit

The latest version of PyDIVIDE can be installed by typing the following command into the terminal

```
pip install pydivide --upgrade
```

2.3 Running PyDIVIDE

An IDE is the recommended way to run PyDIVIDE procedures; however, they can also be run from the terminal. To start an interactive session of Python, enter the following commands into the terminal

```
ipython
import pydivide
```

2.4 Data Storage

PyDIVIDE requires all data files to be stored in an automatically-created directory structure. This has a similar format to the SDC and SSL directory structures. The root directory for data storage can be chosen by the user. When first running a download_files or read procedure, the user will be prompted to select the root_data_dir. After the directory is selected, it is saved in mvn_toolkit_prefs.txt, and can later be changed manually as desired. After the first selection of the directory, the user will not be prompted by download_files or read again. download_files will place files into the chosen directory structure, and read will pull data files from that directory structure.

Note: If you have pyspedas installed (installed by default when installed via pip), then pydivide will use the pyspedas data directory. The default is C://Datapy, but can be changed with pyspedas.set_prefs('data_dir', '/insert/path/for/your/data')

While you do not necessarily need to know this to use the toolkit, the data directories are structured as such

<root_data_dir>/maven/data/sci /kp/insitu/YYYY/MM/ /kp/iuvs/YYYY/MM/ /sta/l2/YYYY/MM/ /sep/l2/YYYY/MM/ /swi/l2/YYYY/MM/ /swe/l2/YYYY/MM/ /lpw/l2/YYYY/MM/ /mag/l2/YYYY/MM/ /iuv/l2/YYYY/MM/ /ngi/l2/YYYY/MM/ /euv/l2/YYYY/MM/ /acc/l2/YYYY/MM/

CHAPTER 3

Reading in Data

This page will describe how to obtain and read in the data. For reading/loading Level 2 files, PySPEDAS is currently required.

3.1 Download Data

pydivide.**download_files**(*filenames=None*, *instruments=None*, *list_files=False*, *level='l2'*, *insitu=True*, *iuvs=False*, *new_files=False*, *start_date='2014-01-01'*, *end_date='2020-01-01'*, *update_prefs=False*, *only_update_prefs=False*, *exclude_orbit_file=False*, *local_dir=None*, *unittest=False*, *crustal_download=True*)

Download data files from the MAVEN SDC web server. Compatible with KP files or instrument-specific data downloads. insitu, iuvs, or at least one instrument must be specified.

Parameters

- **filenames** – str/list of str Specific filename strings to search/download.
- **instruments** – str/list of str - swe, swi, ngi, euv, lpw, iuv, rse, sta, sep, acc Instruments from which you want to download data.
- **list_files** – bool (True/False) If true, lists the files instead of downloading them.
- **level** – str Data level to download.
- **insitu** – bool (True/False) If true, specifies only insitu files.
- **iuvs** – bool (True/False) If true,
- **new_files** – bool (True/False) Checks downloaded files and only downloads those that haven't already been downloaded.
- **start_date** – str String that is the start date for downloading data (YYYY-MM-DD)
- **end_date** – str String that is the end date for downloading data (YYYY-MM-DD)
- **update_prefs** – bool (True/False) If true, updates where you want to store data locally

- **only_update_prefs** – bool (True/False) If true, *only* updates where to store dat alocally, doesn't download files.
- **exclude_orbit_file** – bool (True/False) If true, won't download the latest orbit tables.
- **local_dir** – str If indicated, specifies where to download files for a specific implementation of this function.
- **unittest** – bool If True, will not actually download files. If False (default) files will be downloaded.
- **crustal_download** – bool If True (default), when insitu files are downloaded, any crustal files will also be downloaded. If False, crustal files will not be downloaded when insitu files are downloaded.

Returns None

Examples

```
>>> # Download all available insitu data between 2015-01-01 and 2015-01-31,␣
↪inclusive:
>>> pydivide.download_files(start_date='2015-01-01', end_date='2015-01-31',␣
↪insitu=True)
```

```
>>> # List all available CDF insitu KP files on the server:
>>> pydivide.download_files(insitu=True, list_files=True)
```

```
>>> # Download all new IUVS files from 6 April 2015 not found in the local␣
↪directory.
>>> pydivide.download_files(iuvs=True, new_files=True, end_date='2015-04-06')
```

```
>>> # List all available Level 2 data files for SWIA.
>>> pydivide.download_files(instruments='swi', list_files=True, level='l2')
```

```
>>> # List all available Level 2 data files for SWIA for the month of January␣
↪2015.
>>> pydivide.download_files(start_date='2015-01-01', end_date='2015-01-31',␣
↪instruments='swi', list_files=True, level='l2')
```

```
>>> # Download all new Level 2 data files for NGIMS, STATIC, and EUV.
>>> pydivide.download_files(instruments=['ngi','sta','euv'], new_files=True)
```

3.2 Read KP Data

pydivide.**read**(*filename=None*, *input_time=None*, *instruments=None*, *insitu_only=False*, *specified_files_only=False*)

Read in a given filename in situ file into a dictionary object Optional keywords maybe used to downselect instruments returned and the time windows.

Parameters

- **filename** – str/list of str Name of the in situ KP file(s) to read in.

- **input_time** – list of str/int Set a time bounds/filter on the data, must be length 2 with the first value being the start time, and the second value being the end time.
- **instruments** – Optional keyword listing the instruments to include in the returned dictionary/structure.
- **insitu_only** – Optional keyword that allows you to specify that you only want to download insitu files.
- **specified_files_only** – Optional keyword that allows you to specify you only want filenames given in 'filename' to be read in, not other files close in date/time as well.

Returns A dictionary (data structure) containing up to all of the columns included in a MAVEN in-situ Key parameter data file.

Examples

```
>>> # Retrieve insitu and IUVS data for LPW and MAG on 2015-12-26.
>>> insitu,iuvs = pydivide.read('2015-12-26', instruments=['lpw','mag'])
```

```
>>> # Retrieve only insitu data for all instruments on 2017-06-19.
>>> insitu = pydivide.read('2017-06-19', insitu_only=True)
```

3.3 Read Model Results

pydivide.**read_model_results**(*file*)

Reads results of specified simulation into a dictionary object containing sub-directories for metadata, dimension information, and model tracers. This function can read any of the models currently on the MAVEN SDC website with the .nc extension, which can be found here: https://lasp.colorado.edu/maven/sdc/public/pages/models.html The desired model must be downloaded prior to running this procedure

Parameters **file** – str Simulation result file name to be read

Returns

Dictionary roughly structured as follows

_ **meta** () _ longsubsol _ ls _ etc

_ **dim** _ lat/x _ lon/y _ alt/z

_ **variable1** _ dim_order (x,y,z or z,y,x for example) _ data

_ **variable2** _ dim_order _ data

... _ variableN

Examples

```
>>> # Read the University of Michigan group's ionospheric model for mean solar␣
↪activity (F10.7 = 130).
>>> model = pydivide.read_model_results('<dir_path>/MGITM_LS270_F130_150519.nc')
```

CHAPTER 4

Plotting Insitu KP Data

This page will describe how to plot the KP data

4.1 Altitiude Plots

pydivide.**altplot**(*kp*, *parameter=None*, *time=None*, *errors=None*, *sameplot=True*, *list=False*, *title='Altitude Plot'*, *ylog=False*, *qt=True*)

Plot the provided data plotted against spacecraft altitude. If time is not provided plot entire data set.

Parameters

- **kp** – dict insitu kp data structure/dictionary read from file(s)
- **parameter** – list of str/int The parameter(s) to be plotted. Can be provided as integers (by index) or strings (by name: inst.obs). If a single parameter is provided, it must be an int or str. If several are provided it must be a list. A list may contain a mixture of data types.
- **time** – list of str Two-element list of strings or integers indicating the range of Time to be plotted. At present, there are no checks on whether provided Times are within provided data
- **sameplot** – bool if True, put all curves on same axes if False, generate new axes for each plot
- **list** – bool Lists all Key Parameters instead of plotting
- **title** – str The Title to give the plot
- **ylog** – bool Displays the log of the y axis
- **qt** – bool If true, plots with qt. Else creates an HTML page with bokeh.

Returns None

Examples

```
>>> # Plot LPW.ELECTRON_DENSITY against spacecraft altitude.
>>> pydivide.altplot(insitu, parameter=['LPW.ELECTRON_DENSITY','MAG.MSO_Y'],␣
↪qt=False, ylog=True)
```

4.2 Time Series Plots

pydivide.**plot**(*kp*, *parameter=None*, *time=None*, *errors=None*, *sameplot=True*, *list=False*, *title=''*, *qt=True*, *exec_qt=True*, *log=False*)

Plot time-series data from insitu data structure.

Parameters

- **kp** – dict insitu kp data structure/dictionary read from file(s)
- **parameter** – list of str/int The parameter(s) to be plotted. Can be provided as integers (by index) or strings (by name: inst.obs). If a single parameter is provided, it must be an int or str. If several are provided it must be a list. A list may contain a mixture of data types.
- **time** – list of str Two-element list of strings or integers indicating the range of Time to be plotted. At present, there are no checks on whether provided Times are within provided data
- **sameplot** – bool if True, put all curves on same axes if False, generate new axes for each plot
- **list** – bool Lists all Key Parameters instead of plotting
- **title** – str The Title to give the plot
- **qt** – bool If true, plots with qt. Else creates an HTML page with bokeh.
- **exec_qt** – bool If False, does not run the event loop for pyqtgraph.

Returns : None

Examples

```
>>> # Plot SWIA H+ density.
>>> pydivide.plot(insitu,parameter='swia.hplus_density')
>>> # Plot SWIA H+ density and altitude in the same window.
>>> pydivide.plot(insitu,parameter=['swia.hplus_density', 'spacecraft.altitude'],
↪sameplot=True)
```

4.3 Standard Plots

pydivide.**standards**(*kp*, *list_plots=False*, *all_plots=False*, *euv=False*, *mag_mso=False*, *mag_geo=False*, *mag_cone=False*, *mag_dir=False*, *ngims_neutral=False*, *ngims_ions=False*, *eph_angle=False*, *eph_geo=False*, *eph_mso=False*, *swea=False*, *sep_ion=False*, *sep_electron=False*, *wave=False*, *plasma_den=False*, *plasma_temp=False*, *swia_h_vel=False*, *static_h_vel=False*, *static_o2_vel=False*, *static_flux=False*, *static_energy=False*, *sun_bar=False*, *solar_wind=False*, *ionosphere=False*, *sc_pot=False*, *altitude=False*, *title='Standard Plots'*, *qt=True*)

Generate all or a subset of 25 standardized plots, created from insitu KP data on the MAVEN SDC website

Parameters

- **kp** – dict insitu kp data structure/dictionary read from file(s)
- **mag_mso** – bool magnetic field, MSO coordinates
- **mag_geo** – bool magnetic field, geographic coordinates
- **mag_cone** – bool magnetic clock and cone angles, MSO coordinates
- **mag_dir** – bool magnetic field, radial/horizontal/northward/eastward components
- **ngims_neutral** – bool neutral atmospheric component densities
- **ngims_ions** – bool ionized atmospheric component densities
- **eph_angle** – bool spacecraft ephemeris information
- **eph_geo** – bool spacecraft position, geographic coordinates
- **eph_mso** – bool spacecraft position, MSO coordinates
- **swea** – bool electron parallel/anti-parallel fluxes
- **sep_ion** – bool ion energy flux
- **sep_electron** – bool electron energy flux
- **wave** – bool electric field wave power
- **plasma_den** – bool plasma density
- **plasma_temp** – bool plasma temperature
- **swia_h_vel** – bool H+ flow velocity, SWIA MSO coordinates
- **static_h_vel** – bool H+ flow velocity, STATIC MSO coordinates
- **static_o2_vel** – bool O2+ flow velocity, STATIC MSO coordinates
- **static_flux** – bool H+/He++ and pick-up ion omni-directional flux
- **static_energy** – bool H+/He++ and pick-up ion characteristic energy
- **sun_bar** – bool MAVEN sunlight indicator
- **solar_wind** – bool solar wind dynamic pressure
- **ionosphere** – bool electron spectrum shape parameter
- **altitude** – bool spacecraft altitude
- **sc_pot** – bool spacecraft potential
- **list** – bool Lists all Key Parameters instead of plotting

- **title** – str The Title to give the plot
- **qt** – bool If true, plots with qt. Else creates an HTML page with bokeh.
- **exec_qt** – bool If False, does not run the event loop for pyqtgraph.

Returns : None

Examples :

```
>>> # Solar Orbital coordinates (x, y, z, magnitude), standard spacecraft␣
↪ephemeris
>>> # information (sub-spacecraft lat/lon, subsolar lat/lon, local solar time,
↪ solar
>>> # zenith angle, Mars season)
>>> # omni-directional flux.
>>> insitu,iuvs = pydivide.read(input_time=['2017-06-19','2017-06-20'])
>>> pydivide.standards(insitu, mag_mso=True, eph_angle=True, title='Example␣
↪Title')
```

4.4 Map Plots

pydivide.**map2d**(*kp*, *parameter=None*, *time=None*, *list=False*, *subsolar=False*, *mso=False*, *basemap=None*, *alpha=None*, *title='MAVEN Mars'*, *qt=True*, *exec_qt=True*)

Produces a 2D map of Mars, either in the planetocentric or MSO coordinate system, with the MAVEN orbital projection and a variety of basemaps. Spacecraft orbital path may be colored by a given insitu KP data value

Parameters

- **kp** – dict insitu kp data structure/dictionary read from file(s)
- **parameter** – list of str/int The parameter(s) to be plotted. Can be provided as integers (by index) or strings (by name: inst.obs). If a single parameter is provided, it must be an int or str. If several are provided it must be a list. A list may contain a mixture of data types.
- **time** – list of str Two-element list of strings or integers indicating the range of Time to be plotted. At present, there are no checks on whether provided Times are within provided data
- **color_table** – str Specifies color table to use for plotting
- **subsolar** – bool Plot path of subsolar point
- **mso** – bool Plot using MSO map projection
- **map_limit** – list Set the bounding box on the map in lat/lon coordinates [x0,y0,x1,y1]
- **basemap** – str Name of the basemap on which the spacecraft data with be overlaid. Choices are • 'mdim': Mars Digital Image Model • 'mola': Mars Topography (color) • 'mola_bw': Mars Topography (black and white) • 'mag': Mars Crustal Magnetism • '<dir_path>/file.png': User-defined basemap
- **sameplot** – bool if True, put all curves on same axes if False, generate new axes for each plot
- **list** – bool Lists all Key Parameters instead of plotting
- **title** – str The Title to give the plot
- **ylog** – bool Displays the log of the y axis
- **qt** – bool If true, plots with qt. Else creates an HTML page with bokeh.

- **exec_qt** – bool If False, does not run the event loop for pyqtgraph.

Returns : None

Examples

```
>>> # Plot spacecraft altitude along MAVEN surface orbital track.
>>> pydivide.map2d(insitu, 'spacecraft.altitude')
```

```
>>> # Plot spacecraft altitude along MAVEN surface orbital track using MOLA␣
↪altimetry basemap; plot subsolar point path.
>>> pydivide.map2d(insitu, 'spacecraft.altitude', basemap='mola', subsolar=True)
```

CHAPTER 5

Plotting IUVS KP Data

5.1 IUVS Corona Plots

pydivide.**corona**(*iuvs*, *sameplot=True*, *density=True*, *radiance=True*, *orbit_num=None*, *species=None*, *log=False*, *title='IUVS Corona Observations'*, *qt=True*, *exec_qt=True*)

Plot IUVS Corona Scan data against spacecraft altitude.

Parameters

- **iuvs** – dict iuvs kp data structure/dictionary read from file(s)
- **orbit_num** – list of int The orbit numbers to plot from the IUVS data structure
- **species** – list of str The species to plot. Values can be

  Density - CO2, CO2+, O, N2, C, N, H Radiance - CO2pUVD, CO, H, O_1304, O_1356, O_2972, C_1561, C_1657, N_1493, N2, NO
- **radiance** – bool If true, plots the radiance
- **density** – bool If true, plots the density
- **sameplot** – bool if True, put all curves on same axes if False, generate new axes for each plot
- **title** – str The Title to give the plot
- **ylog** – bool Displays the log of the y axis
- **qt** – bool If true, plots with qt. Else creates an HTML page with bokeh.
- **exec_qt** – bool If False, does not run the event loop for pyqtgraph.

Returns : None

Examples

```
>>> # Plot CO2 density vs spacecraft altitude.
>>> insitu, iuvs = pydivide.read(input_time=['2016-02-01', '2016-02-28'])
>>> pydivide.periapse(iuvs, species='N2', orbit=2726, log=True, density=True,␣
↪radiance=False, qt=False)
```

5.2 IUVS Occultation Plots

pydivide.**occultation**(*iuvs*, *sameplot=True*, *orbit_num=None*, *species=None*, *log=False*, *title='IUVS Occultation Observations'*, *qt=True*, *exec_qt=True*)

Plot IUVS Stellar Occultation data against spacecraft altitude.

Parameters

- **iuvs** – dict iuvs kp data structure/dictionary read from file(s)
- **orbit_num** – list of int The orbit numbers to plot from the IUVS data structure
- **species** – list of str The species to plot. Values can be "CO2", "O2", "O3", and "Temp."
- **sameplot** – bool if True, put all curves on same axes if False, generate new axes for each plot
- **list** – bool Lists all Key Parameters instead of plotting
- **title** – str The Title to give the plot
- **ylog** – bool Displays the log of the y axis
- **qt** – bool If true, plots with qt. Else creates an HTML page with bokeh.
- **exec_qt** – bool If False, does not run the event loop for pyqtgraph.

Returns : None

Examples

```
>>> # Plot CO2 density vs spacecraft altitude.
>>> insitu, iuvs = pydivide.read(input_time=['2016-01-01', '2016-01-31'])
>>> pydivide.occultation(iuvs, log=True, species=['CO2'], qt=False)
```

5.3 IUVS Limb Plots

pydivide.**periapse**(*iuvs*, *sameplot=True*, *density=True*, *radiance=True*, *orbit_num=None*, *species=None*, *obs_num=None*, *log=False*, *title='IUVS Periapse Observations'*, *qt=True*, *exec_qt=True*)

Plot IUVS Limb Scan data against spacecraft altitude.

Parameters

- **iuvs** – dict iuvs kp data structure/dictionary read from file(s)
- **orbit_num** – list of int The orbit numbers to plot from the IUVS data structure
- **species** – list of str The species to plot. Values can be

Density - CO2, CO2+, O, N2, C, N, H Radiance - CO2pUVD, CO, H, O_1304, O_1356, O_2972, C_1561, C_1657, N_1493, N2, NO

- **radiance** – bool If true, plots the radiance
- **density** – bool If true, plots the density
- **sameplot** – bool if True, put all curves on same axes if False, generate new axes for each plot
- **title** – str The Title to give the plot
- **ylog** – bool Displays the log of the y axis
- **qt** – bool If true, plots with qt. Else creates an HTML page with bokeh.
- **exec_qt** – bool If False, does not run the event loop for pyqtgraph.

Returns : None

Examples

```
>>> # Plot CO2 density vs spacecraft altitude.
>>> insitu, iuvs = pydivide.read(input_time=['2016-02-01', '2016-02-28'])
>>> pydivide.periapse(iuvs, species='CO', orbit_num=2726, log=True, density=False,
↪ radiance=True, qt=False)
```

CHAPTER 6

Plotting Level 2 Data

PyDIVIDE is currently in a strange place right now, and in the future may further depend on PySPEDAS or even become a part of that library. Currently, we have only one routine to display level 2 data.

6.1 Additional Requirements

All level 2 data plotting routines will rely on PySPEDAS for loading the data into memory (into xarrays). This library can be installed via

```
pip install pyspedas
```

6.2 Full Plots

This routine was created as a way to uniquely visualize the MAVEN data, as well as display the capabilities of PySPEDAS and PyTplot. Currently, it only works with the pyqtgraph visualization library, but hopefully a bokeh window will be added in the future.

If you would like the fully interactive 3D window to appear, you must first run

```
pip install pyopengl
```

pydivide.**fullplot**(*instruments=None*, *level='l2'*, *type=None*, *start_date='2014-01-01'*, *end_date='2014-01-02'*, *tplot_names=''*, *filenames=None*, *insitu=None*, *parameter=''*, *auto_yes=True*)

Plot any insitu Level 2 or KP data from MAVEN. Downloads files found into PySPEDAS and loads them into memory via PyTplot. Then creates an interactive plot window including spectrogram slicer, MAVEN's location and orbit in MSO coordinates, and MAVEN's location in GEO coordinates, especially relative to the crustal magnetic fields.

Parameters

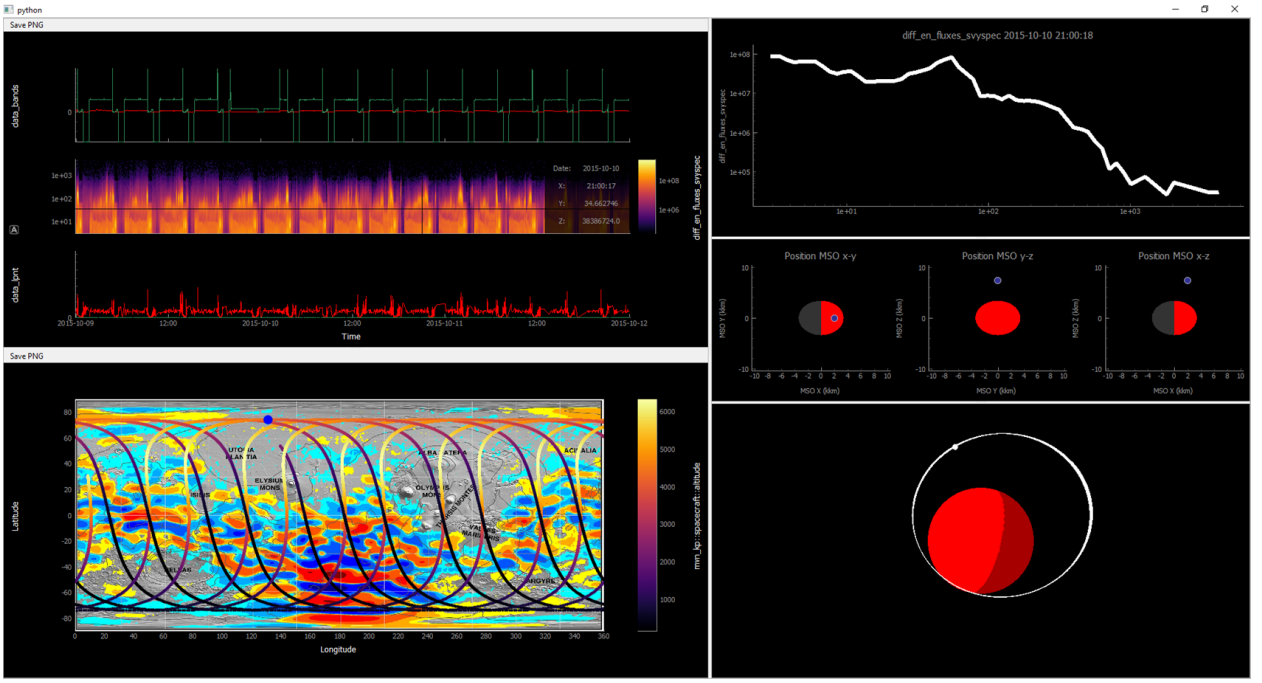
- **instruments** – str/list of str Instruments from which you want to download data. Accepted values are any combination of: sta, swi, swe, lpw, euv, ngi, iuv, mag, sep, rse
- **type** – str/list of str The observation/file type of the instruments to load. If None, all file types are loaded. Otherwise, a file will only be loaded into tplot if its descriptor matches one of the strings in this field. See the instrument SIS for more detail on types.
- **tplot_names** – list of str The tplot names to plot. Also not needed, use only if the variables are already loaded into memory. For example, if you want to load in data with this fullplot procedure but modify the variables with pytplot.options or the pytplot.tplot_math routines, you can re-plot the data by specifying the specific pytplot variables.
- **filenames** – str/list of str ['yyyy-mm-dd'] List of files to load
- **start_date** – str String that is the start date for downloading data (YYYY-MM-DD), or the orbit number
- **end_date** – str String that is the end date for downloading data (YYYY-MM-DD), or the orbit number
- **kp** – dict insitu kp data structure/dictionary read from file(s). This is not required, only needed if you want to plot variables from this data structure.
- **parameter** – list of str/int If the above kp data structure is given, this variable will be the parameters to plot (see the pydivide.plot function)

Types:

Returns : None

Examples

```
>>> # Plots EUV Bands, LPW LP-IV, and MAG SS data on Jan 01 2015
>>> pydivide.fullplot(instruments=['euv', 'lpw', 'mag'], type=['bands', 'lpnt',
↪'ss1s'], start_date='2015-01-01', end_date='2015-01-02')
```



CHAPTER 7

Manipulating KP Data

7.1 Resample

pydivide.**resample**(*kp*, *time*, *sc_only=False*)

Modifies KP structure index to user specified time via interpolation.

Parameters

- **kp** – struct KP insitu data structure read from file(s).
- **time** – list Specifies subset of insitu KP data for resampling. time must be expressed in the format 'YYYY-MM-DD HH:MM:SS'.

Examples

```
>>> # Resample insitu time to 2016-06-20 coarse survey 3D file time.
>>> swi_cdf = cdflib.CDF('<dir_path>/mvn_swi_l2_coarsesvy3d_20160620_v01_r00.cdf')
>>> newtime = swi_cdf.varget('time_unix')
>>> insitu_resampled = pydivide.resample(insitu, newtime)
```

```
>>> # Resamples an entire day of data to just 3 points
>>> import pytplot
>>>insitu, iuvs = pydivide.read(input_time=['2016-02-18', '2016-02-19'])
>>> x = pydivide.resample(insitu, [pytplot.tplot_utilities.str_to_int('2016-02-
↪18T05:00:00'),
>>>                        pytplot.tplot_utilities.str_to_int('2016-02-18T10:00:00
↪'),
>>>                        pytplot.tplot_utilities.str_to_int('2016-02-18T15:00:00
↪')])
```

7.2 Bin

pydivide.**bin**(*kp*, *parameter=None*, *bin_by=None*, *mins=None*, *maxs=None*, *binsize=None*, *std=False*, *avg=False*, *density=False*, *median=False*, *unittest=False*)

Bins insitu Key Parameters by up to 8 different parameters, specified within the data structure. Necessary that at least one of avg, std, median, or density be specified.

Parameters

- **kp** – struct KP insitu data structure read from file(s).
- **parameter** – str Key Parameter to be binned. Only one may be binned at a time.
- **bin_by** – int, str Parameters (index or name) by which to bin the specified Key Parameter.
- **binsize** – int, list
- **size for each binning dimension. Number of elements must be equal to those in bin_by.** (*Bin*) –
- **mins** – int, list Minimum value(s) for each binning scheme. Number of elements must be equal to those in bin_by.
- **maxs** – int, list 7 Maximum value(s) for each binning scheme. Number of elements must be equal to those in bin_by.
- **avg** – bool Calculate average per bin.
- **std** – bool Calculate standard deviation per bin.
- **density** – bool Returns number of items in each bin.
- **median** – bool Calculate median per bin.

Returns This procedures outputs up to 4 arrays to user-defined variables, corresponding to avg, std, median, and density.

Examples: >>> # Bin STATIC O+ characteristic energy by spacecraft latitude (1° resolution) and longitude (2° resolution). >>> output_avg = pydivide.bin(insitu, parameter='static.oplus_char_energy', bin_by=['spacecraft.geo_latitude', 'spacecraft.geo_longitude'], avg=True,binsize=[2,1])

```
>>> # Bin SWIA H+ density by spacecraft altitude (10km resolution), return␣
↪average value and standard deviation for each bin.
>>> output_avg,output_std = pydivide.bin(insitu, parameter='swia.hplus_density',␣
↪bin_by='spacecraft.altitude', binsize=10,avg=True,std=True)
```

CHAPTER 8

Model Manipulation

This page will describe some of the routines to view Model results

8.1 Create Model Maps

pydivide.**create_model_maps**(*altitude*, *variable=None*, *model=None*, *file=None*, *numcontours=25*, *fill=False*, *ct='viridis'*, *transparency=1*, *nearest=False*, *linear=True*, *savefig=True*)

Generates a .png contour map of a model at a specific altitude. These can be used as a background in map2d. The models must be downloaded manually from the SDC website: https://lasp.colorado.edu/maven/sdc/public/pages/models.html.

Parameters

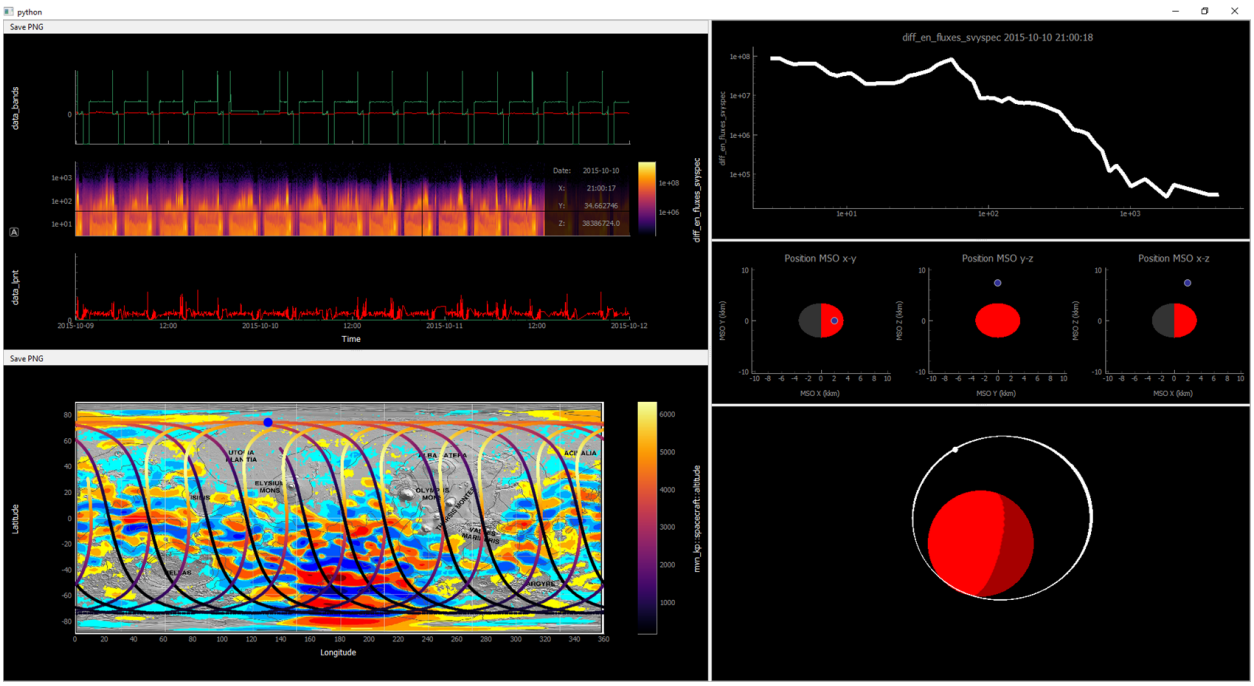
- **altitude** – int Specified altitude of output map.
- **variable** – str Plots specified chemical species (Appendix A).
- **model** – dict Model variable produced from prior call to read_model_results.
- **file** – str If model not provided (produced from read_model_results), full path to model can be set and read.
- **numContours** – int Specifies number of contour lines. Default is 25.
- **fill** – bool If True, fills in contour levels instead of generating lines.
- **ct** – str Sets color table. Valid color tables can be found here: https://matplotlib.org/examples/color/colormaps_reference.html
- **transparency** – int, float Sets transparency between [0,1] inclusive. 0 is completely transparent, and 1 is completely opaque.
- **nearest** – bool If True, instead of interpolating nearby values, this returns the value of the nearest neighbor altitude.
- **linear** – bool If True, performs linear interpolation between 2 altitude layers.

- **saveFig** – bool If True, saves figure as .png file.

Returns None

Examples

```
>>> # Interpolate all model tracers to spacecraft trajectory using nearest␣
↪neighbor interpolation.
>>> pydivide.create_model_maps(altitude=170, file = '<dir_path>/MAMPS_LS180_F130_
↪081216.nc', variable='geo_x', saveFig=True)
```



8.2 Interpolate Model

pydivide.**interpol_model**(*kp*, *model=None*, *file=None*, *nearest=False*)

Reads in MAVEN's position from insitu, and determines the value of the models at those points.

There are 3 scenarios to interpolate: 1) MSO coordinate system with latitude/longitude/altitude 2) GEO coordinate system with latitude/longitude/altitude 3) MSO coordinate system with x/y/z NOTE: GEO x/y/z coordinate system is not allowed

In all 3 cases, everything is converted to an MSO lat/lon/alt coordinate system. For interpolation purposes, the atmosphere acts like a cube with dimensions lat*lon*alt This makes it so the interpolation is weighted more accurately. A point that is 1 degree of lat/lon away will have as much influence as a point that is 1 kilometer higher or lower

Parameters

- **kp** – struct KP insitu data structure read from file(s).
- **model** – str Source of simulation data to be interpolated.
- **file** – str If model not provided, can specify the full path to the model.

- **nearest** – bool If True, instead of interpolating nearby values, this returns the value of the nearest neighbor altitude.

Returns Numpy array of data representative of what the spacecraft would have measured if it were travelling through the model

Examples

```
>>> # Interpolate all model tracers to the spacecraft trajectory using nearest␣
↪neighbor interpolation.
>>> # results = pydivide.interpol_model(insitu, file='<dir_path>/Elew_18_06_14_
↪t00600.nc', nearest=True)
```

CHAPTER 9

Key Parameters to Tplot Variables

PyDIVIDE ultimately uses PyTplot to do plotting routines. If you would prefer to work with the KP data in PyTplot directly, then this is the function for you.

pydivide.**tplot_varcreate**(*insitu*, *instruments=None*, *observations=None*)

Can turn an insitu object from the pydivide.read procedure into pytplot variables.

Parameters

- **insitu** – dict The dictionary object that gets created from an insitu read procedure
- **instruments** – str Specific instruments to be loaded into pytplot
- **observations** – str Specific observations to be loaded into pytplot

Returns A string list of the created tplot variables

Examples

```
>>> # Load MAG and NGIMS data into PyTplot
>>> tvars = pydivide.tplot_varcreate(insitu, instruments=['MAG', 'NGIMS'])
```

The variables created are stored in a global dictionary of xarray objects, and given names in the form

```
mvn::kp::{instrument}::{observation}
```

For example: “mvn::kp::mag::mso_x”

CHAPTER 10

Insitu KP Data Structure

Insitu parameters are callable either by name or number listed below. IUVS parameters are only callable by name.

1. TimeString
2. Time
3. Orbit
4. IOflag

10.1 LPW

5. LPW.ELECTRON_DENSITY
6. LPW.ELECTRON_DENSITY_QUAL_MIN
7. LPW.ELECTRON_DENSITY_QUAL_MAX
8. LPW.ELECTRON_TEMPERATURE
9. LPW.ELECTRON_TEMPERATURE_QUAL_MIN
10. LPW.ELECTRON_TEMPERATURE_QUAL_MAX
11. LPW.SPACECRAFT_POTENTIAL
12. LPW.SPACECRAFT_POTENTIAL_QUAL_MIN
13. LPW.SPACECRAFT_POTENTIAL_QUAL_MAX
14. LPW.EWAVE_LOW_FREQ
15. LPW.EWAVE_LOW_FREQ_QUAL
16. LPW.EWAVE_MID_FREQ
17. LPW.EWAVE_MID_FREQ_QUAL
18. LPW.EWAVE_HIGH_FREQ

19. LPW.EWAVE_HIGH_FREQ_QUAL

10.2 EUV

20. EUV.IRRADIANCE_LOW
21. EUV.IRRADIANCE_LOW_QUAL
22. EUV.IRRADIANCE_MID
23. EUV.IRRADIANCE_MID_QUAL
24. EUV.IRRADIANCE_LYMAN
25. EUV.IRRADIANCE_LYMAN_QUAL

10.3 SWEA

26. SWEA.SOLAR_WIND_ELECTRON_DENSITY
27. SWEA.SOLAR_WIND_ELECTRON_DENSITY_QUAL
28. SWEA.SOLAR_WIND_ELECTRON_TEMPERATURE
29. SWEA.SOLAR_WIND_ELECTRON_TEMPERATURE_QUAL
30. SWEA.ELECTRON_PARALLEL_FLUX_LOW
31. SWEA.ELECTRON_PARALLEL_FLUX_LOW_QUAL
32. SWEA.ELECTRON_PARALLEL_FLUX_MID
33. SWEA.ELECTRON_PARALLEL_FLUX_MID_QUAL
34. SWEA.ELECTRON_PARALLEL_FLUX_HIGH
35. SWEA.ELECTRON_PARALLEL_FLUX_HIGH_QUAL
36. SWEA.ELECTRON_ANTI_PARALLEL_FLUX_LOW
37. SWEA.ELECTRON_ANTI_PARALLEL_FLUX_LOW_QUAL
38. SWEA.ELECTRON_ANTI_PARALLEL_FLUX_MID
39. SWEA.ELECTRON_ANTI_PARALLEL_FLUX_MID_QUAL
40. SWEA.ELECTRON_ANTI_PARALLEL_FLUX_HIGH
41. SWEA.ELECTRON_ANTI_PARALLEL_FLUX_HIGH_QUAL
42. SWEA.ELECTRON_SPECTRUM_SHAPE_PARAMETER
43. SWEA.ELECTRON_SPECTRUM_SHAPE_PARAMETER_QUAL

10.4 SWIA

44. SWIA.HPLUS_DENSITY
45. SWIA.HPLUS_DENSITY_QUAL
46. SWIA.HPLUS_FLOW_VELOCITY_MSO_X

47. SWIA.HPLUS_FLOW_VELOCITY_MSO_X_QUAL
48. SWIA.HPLUS_FLOW_VELOCITY_MSO_Y
49. SWIA.HPLUS_FLOW_VELOCITY_MSO_Y_QUAL
50. SWIA.HPLUS_FLOW_VELOCITY_MSO_Z
51. SWIA.HPLUS_FLOW_VELOCITY_MSO_Z_QUAL
52. SWIA.HPLUS_TEMPERATURE
53. SWIA.HPLUS_TEMPERATURE_QUAL
54. SWIA.SOLAR_WIND_DYNAMIC_PRESSURE
55. SWIA.SOLAR_WIND_DYNAMIC_PRESSURE_QUAL

10.5 STATIC

56. STATIC.STATIC_QUALITY_FLAG
57. STATIC.HPLUS_DENSITY
58. STATIC.HPLUS_DENSITY_QUAL
59. STATIC.OPLUS_DENSITY
60. STATIC.OPLUS_DENSITY_QUAL
61. STATIC.O2PLUS_DENSITY
62. STATIC.O2PLUS_DENSITY_QUAL
63. STATIC.HPLUS_TEMPERATURE
64. STATIC.HPLUS_TEMPERATURE_QUAL
65. STATIC.OPLUS_TEMPERATURE
66. STATIC.OPLUS_TEMPERATURE_QUAL
67. STATIC.O2PLUS_TEMPERATURE
68. STATIC.O2PLUS_TEMPERATURE_QUAL
69. STATIC.O2PLUS_FLOW_VELOCITY_MAVEN_APP_X
70. STATIC.O2PLUS_FLOW_VELOCITY_MAVEN_APP_X_QUAL
71. STATIC.O2PLUS_FLOW_VELOCITY_MAVEN_APP_Y
72. STATIC.O2PLUS_FLOW_VELOCITY_MAVEN_APP_Y_QUAL
73. STATIC.O2PLUS_FLOW_VELOCITY_MAVEN_APP_Zz
74. STATIC.O2PLUS_FLOW_VELOCITY_MAVEN_APP_Z_QUAL
75. STATIC.O2PLUS_FLOW_VELOCITY_MSO_X
76. STATIC.O2PLUS_FLOW_VELOCITY_MSO_X_QUAL
77. STATIC.O2PLUS_FLOW_VELOCITY_MSO_Y
78. STATIC.O2PLUS_FLOW_VELOCITY_MSO_Y_QUAL
79. STATIC.O2PLUS_FLOW_VELOCITY_MSO_Z

80. STATIC.O2PLUS_FLOW_VELOCITY_MSO_Z_QUAL
81. STATIC.HPLUS_OMNI_DIRECTIONAL_FLUX
82. STATIC.HPLUS_CHARACTERISTIC_ENERGY
83. STATIC.HPLUS_CHARACTERISTIC_ENERGY_QUAL
84. STATIC.HEPLUS_OMNI_DIRECTIONAL_FLUX
85. STATIC.HEPLUS_CHARACTERISTIC_ENERGY
86. HSTATIC.HEPLUS_CHARACTERISTIC_ENERGY_QUAL
87. STATIC.OPLUS_OMNI_DIRECTIONAL_FLUX
88. STATIC.OPLUS_CHARACTERISTIC_ENERGY
89. STATIC.OPLUS_CHARACTERISTIC_ENERGY_QUAL
90. STATIC.O2PLUS_OMNI_DIRECTIONAL_FLUX
91. STATIC.O2PLUS_CHARACTERISTIC_ENERGY
92. STATIC.O2PLUS_CHARACTERISTIC_ENERGY_QUAL
93. STATIC.HPLUS_CHARACTERISTIC_DIRECTION_MSO_X
94. STATIC.HPLUS_CHARACTERISTIC_DIRECTION_MSO_Y
95. STATIC.HPLUS_CHARACTERISTIC_DIRECTION_MSO_Z
96. STATIC.HPLUS_CHARACTERISTIC_ANGULAR_WIDTH
97. STATIC.HPLUS_CHARACTERISTIC_ANGULAR_WIDTH_QUAL
98. STATIC.DOMINANT_PICKUP_ION_CHARACTERISTIC_DIRECTION_MSO_X
99. STATIC.DOMINANT_PICKUP_ION_CHARACTERISTIC_DIRECTION_MSO_Y
100. STATIC.DOMINANT_PICKUP_ION_CHARACTERISTIC_DIRECTION_MSO_Z
101. STATIC.DOMINANT_PICKUP_ION_CHARACTERISTIC_ANGULAR_WIDTH
102. STATIC.DOMINANT_PICKUP_ION_CHARACTERISTIC_ANGULAR_WIDTH_QUAL

10.6 SEP

103. SEP.ION_ENERGY_FLUX__FOV_1_F
104. SEP.ION_ENERGY_FLUX__FOV_1_F_QUAL
105. SEP.ION_ENERGY_FLUX__FOV_1_R
106. SEP.ION_ENERGY_FLUX__FOV_1_R_QUAL
107. SEP.ION_ENERGY_FLUX__FOV_2_F
108. SEP.ION_ENERGY_FLUX__FOV_2_F_QUAL
109. SEP.ION_ENERGY_FLUX__FOV_2_R
110. SEP.ION_ENERGY_FLUX__FOV_2_R_QUAL
111. SEP.ELECTRON_ENERGY_FLUX___FOV_1_F
112. SEP.ELECTRON_ENERGY_FLUX___FOV_1_F_QUAL

113. SEP.ELECTRON_ENERGY_FLUX___FOV_1_R
114. SEP.ELECTRON_ENERGY_FLUX___FOV_1_R_QUAL
115. SEP.ELECTRON_ENERGY_FLUX___FOV_2_F
116. **SEP.ELECTRON_ENERGY_FLUX__**

10.7 MAG

131. MAG.MSO_X
132. MAG.MSO_X_QUAL
133. MAG.MSO_Y
134. MAG.MSO_Y_QUAL
135. MAG.MSO_Z
136. MAG.MSO_Z_QUAL
137. MAG.GEO_X
138. MAG.GEO_X_QUAL
139. MAG.GEO_Y
140. MAG.GEO_Y_QUAL
141. MAG.GEO_Z
142. MAG.GEO_Z_QUAL
143. MAG.RMS_DEVIATION
144. MAG.RMS_DEVIATION_QUAL

10.8 NGIMS

145. NGIMS.HE_DENSITY
146. NGIMS.HE_DENSITY_PRECISION
147. NGIMS.HE_DENSITY_QUAL
148. NGIMS.O_DENSITY
149. NGIMS.O_DENSITY_PRECISION
150. NGIMS.O_DENSITY_QUAL
151. NGIMS.CO_DENSITY
152. NGIMS.CO_DENSITY_PRECISION
153. NGIMS.CO_DENSITY_QUAL
154. NGIMS.N2_DENSITY
155. NGIMS.N2_DENSITY_PRECISION
156. NGIMS.N2_DENSITY_QUAL
157. NGIMS.NO_DENSITY

158. NGIMS.NO_DENSITY_PRECISION
159. NGIMS.NO_DENSITY_QUAL
160. NGIMS.AR_DENSITY
161. NGIMS.AR_DENSITY_PRECISION
162. NGIMS.AR_DENSITY_QUAL
163. NGIMS.CO2_DENSITY
164. NGIMS.CO2_DENSITY_PRECISION
165. NGIMS.CO2_DENSITY_QUAL
166. NGIMS.O2PLUS_DENSITY
167. NGIMS.O2PLUS_DENSITY_PRECISION
168. NGIMS.O2PLUS_DENSITY_QUAL
169. NGIMS.CO2PLUS_DENSITY
170. NGIMS.CO2PLUS_DENSITY_PRECISION
171. NGIMS.CO2PLUS_DENSITY_QUAL
172. NGIMS.NOPLUS_DENSITY
173. NGIMS.NOPLUS_DENSITY_PRECISION
174. NGIMS.NOPLUS_DENSITY_QUAL
175. NGIMS.OPLUS_DENSITY
176. NGIMS.OPLUS_DENSITY_PRECISION
177. NGIMS.OPLUS_DENSITY_QUAL
178. NGIMS.CO2PLUS_N2PLUS_DENSITY
179. NGIMS.CO2PLUS_N2PLUS_DENSITY_PRECISION
180. NGIMS.CO2PLUS_N2PLUS_DENSITY_QUAL
181. NGIMS.CPLUS_DENSITY
182. NGIMS.CPLUS_DENSITY_PRECISION
183. NGIMS.CPLUS_DENSITY_QUAL
184. NGIMS.OHPLUS_DENSITY
185. NGIMS.OHPLUS_DENSITY_PRECISION
186. NGIMS.OHPLUS_DENSITY_QUAL
187. NGIMS.NPLUS_DENSITY
188. NGIMS.NPLUS_DENSITY_PRECISION
189. NGIMS.NPLUS_DENSITY_QUAL

10.9 APP

190. APP.ATTITUDE_GEO_X
191. APP.ATTITUDE_GEO_Y
192. APP.ATTITUDE_GEO_Z
193. APP.ATTITUDE_MSO_X
194. APP.ATTITUDE_MSO_Y
195. APP.ATTITUDE_MSO_Z

10.10 SPACECRAFT

196. SPACECRAFT.GEO_X
197. SPACECRAFT.GEO_Y
198. SPACECRAFT.GEO_Z
199. SPACECRAFT.MSO_X
200. SPACECRAFT.MSO_Y
201. SPACECRAFT.MSO_Z
202. SPACECRAFT.SUB_SC_LONGITUDE
203. SPACECRAFT.SUB_SC_LATITUDE
204. SPACECRAFT.SZA
205. SPACECRAFT.LOCAL_TIME
206. SPACECRAFT.ALTITUDE
207. SPACECRAFT.ATTITUDE_GEO_X
208. SPACECRAFT.ATTITUDE_GEO_Y
209. SPACECRAFT.ATTITUDE_GEO_Z
210. SPACECRAFT.ATTITUDE_MSO_X
211. SPACECRAFT.ATTITUDE_MSO_Y
212. SPACECRAFT.ATTITUDE_MSO_Z
213. SPACECRAFT.MARS_SEASON
214. SPACECRAFT.MARS_SUN_DISTANCE
215. SPACECRAFT.SUBSOLAR_POINT_GEO_LONGITUDE
216. SPACECRAFT.SUBSOLAR_POINT_GEO_LATITUDE
217. SPACECRAFT.SUBMARS_POINT_SOLAR_LONGITUDE
218. SPACECRAFT.SUBMARS_POINT_SOLAR_LATITUDE
219. SPACECRAFT.T11
220. SPACECRAFT.T12
221. SPACECRAFT.T13

222. SPACECRAFT.T21
223. SPACECRAFT.T22
224. SPACECRAFT.T23
225. SPACECRAFT.T31
226. SPACECRAFT.T32
227. SPACECRAFT.T33
228. SPACECRAFT.SPACECRAFT_T11
229. SPACECRAFT.SPACECRAFT_T12
230. SPACECRAFT.SPACECRAFT_T13
231. SPACECRAFT.SPACECRAFT_T21
232. SPACECRAFT.SPACECRAFT_T22
233. SPACECRAFT.SPACECRAFT_T23
234. SPACECRAFT.SPACECRAFT_T31
235. SPACECRAFT.SPACECRAFT_T32
236. SPACECRAFT.SPACECRAFT_T33

CHAPTER 11

IUVS KP Data Structure

11.1 ALL

- TIME_START
- TIME_STOP
- SZA
- LOCAL_TIME
- LAT
- LON
- LAT_MSO
- LON_MSO
- ORBIT_NUMBER
- MARS_SEASON_LS
- SPACECRAFT_GEO
- SPACECRAFT_MSO
- SUN_GEO
- SPACECRAFT_GEO_LONGITUDE
- SPACECRAFT_GEO_LATITUDE
- SPACECRAFT_MSO_LONGITUDE
- SPACECRAFT_MSO_LATITUDE
- SUBSOLAR_POINT_GEO_LONGITUDE
- SUBSOLAR_POINT_GEO_LATITUDE
- SPACECRAFT_SZA

- SPACECRAFT_LOCAL_TIME
- SPACECRAFT_ALTITUDE
- MARS_SUN_DISTANCE

11.2 PERIAPSE1/PERIAPSE2/PERIAPSE3

- SCALE_HEIGHT
- DENSITY
- RADIANCE
- TEMPERATURE
- ALT

11.3 APOAPSE

- OZONE_DEPTH
- AURORAL_INDEX
- DUST_DEPTH
- RADIANCE
- SZA_BP
- LOCAL_TIME_BP
- LON_BINS
- LAT_BINS

11.4 CORONA_LORES_HIGH

- HALF_INT_DISTANCE
- TEMPERATURE
- DENSITY
- RADIANCE
- ALT

11.5 OCCULTATION

- CO2
- O2
- O3
- TEMPERATURE

11.6 Measured Chemical Species

- C
- C_1561
- C_1657
- CO Cameron
- CO2
- CO2+
- CO2pUVD
- H
- N
- N2
- N_1493
- NO
- O
- O2
- O3
- O_1304
- O_1356
- O_2972

Index