MAVEN Python Toolkit Users' Guide

2017-June-14

Table of Contents

1. Toolkit Installation 1
1.1. System Requirements 1
1.2. Downloading the Toolkit 2
1.3. Updating the Toolkit 2
1.4. Required Data Directory Structure 2
1.5. Starting the Toolkit 3
1.6. Getting More Help 4
2. Toolkit Routines 4
2.1. Downloading Data Using the Toolkit 4
2.1.1. download_files 4
2.2. Reading data into Python Memory 6
2.2.1. read 6
2.2.2. read_model_results 8
2.3. Manipulating Key Parameter Data 9
2.3.1. insitu_search 9
2.3.2. bin 10
2.4. Plotting Key Parameter Data 12
2.4.1. plot 12
2.4.2. altplot 13
2.4.3. standards 14
2.4.4. corona 15
2.4.5. periapse 16
2.5. Interpolating Model Results 17
2.5.1. interpol_model 17
2.5.2. create_model_maps 18
2.6. Plotting of Key parameter data in 2D 19
2.6.1. map2d 19
A. Appendix: KP Data Structures in the ToolKit 20

1. Toolkit Installation

1.1. System Requirements

The MAVEN Toolkit currently requires Anaconda version 4 or above. It can be obtained from the following link:

https://www.continuum.io/downloads

Anaconda will install the python programming language, as well as numerous software libraries used for scientific computing. This toolkit is only available for python version 3.

1.2. Downloading the Toolkit

The easiest way to install the toolkit is simply typing the following command in the terminal:

```
>>pip install pydivide
```

This will install pydivide and all of the dependencies automatically.
The following command may also be necessary:

```
>>conda install -c bokeh nodejs
```

This will install nodejs for the plotting software.

The MAVEN Toolkit is also available for download at the MAVEN Science Data Center github page:

https://github.com/MAVENSDC

However, installing the toolkit this way will require changing your python path so that your Anaconda installation can find this library. It will also require installing all of the dependencies of the Toolkit manually. Unless you are familiar with python/Anaconda, it is highly recommended that you install the toolkit via the pip command above.

1.3. Updating the Toolkit

You can install the latest version of the toolkit by typing the following command in the terminal:

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

1.4. Required Data Directory Structure

The toolkit requires data files to be stored in a particular directory structure. This directory structure matches the SDC and SSL directory structure. However, a user is able to choose the root directory location for the data to be stored (will refer to this as the `ROOT_DATA_DIR`). When you first use a download or read procedure, you will be prompted to select this `ROOT_DATA_DIR`. After the selection is made, it is saved in a file (`mvn_toolkit_prefs.txt`), and can be changed later if desired. A user can also choose to set an environment variable `ROOT_DATA_DIR` instead of using the preferences file. The remaining directory structure will then be automatically generated under this `ROOT_DATA_DIR`. The toolkit download procedures will download files into this directory structure, and the read procedure expects to find data files there as well. The directory structure that is created (and required) looks like:

```
<ROOT_DATA_DIR>/maven/data/sci/kp/insitu/<YYYY>/<MM>/
                               kp/iuvs/<YYYY>/<MM>/
```

And if you choose to download the level 2 data using the Toolkit routine (**`download_files`**), the following will be created (and required):

```
<ROOT_DATA_DIR>/maven/data/sci/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>/
```

- **Note**: If you are on Windows, the forward slashes (/) will be back slashes (\).
- `<ROOT_DATA_DIR>` is chosen by the user.
- <YYYY> and <MM> will be created as files for those year/month exist.

1.5. Starting the Toolkit

The toolkit was developed under the assumption that most users would be using it in an interactive command line environment. To start up an interactive session of python, type the following command in the terminal:

```
>>IPython
```

The terminal should display that an interactive version of python has begun. Then to start a toolkit session, type:

```
>>import pydivide
```

The toolkit commands are now ready to be used by typing them into the terminal.

If you want to do more sophisticated data analysis than the toolkit currently offers, there are several other libraries you can import as well that provide functions similar to IDL and MATLAB. These libraries include:

```
>>import scipy
```

```
>>import numpy
>>import pandas
```

These packages are automatically included in an Anaconda installation, so you do not need to download them separately. Instructions on how to use these libraries are beyond the scope of this text, but helpful guides for these projects and others can be found here:

https://www.scipy.org/docs.html

NOTE: You do not need to import these libraries in order for the pydivide toolkit to work, you only need the initial "import pydivide" command.

1.6. Getting More Help

If you have any further problems or questions, either with installation or operation of the Toolkit, feel free to contact the developers at maven_divide@lasp.colorado.edu

2. Toolkit Routines

Here we list the routines available in the MAVEN Toolkit and provide static documentation for their usage. For each, we will list the procedure name, followed by a brief description of what the code does, including the required and optional arguments that can be passed to the procedure. Next, we display an example usage of the procedure, followed by a comprehensive listing of all of the required arguments, then the optional arguments.

2.1. Downloading Data Using the Toolkit

2.1.1. download_files

Description

This procedure downloads in-situ and/or IUVS Key Parameter (KP) data files from the MAVEN SDC web server. It can also download specific instrument data, though the toolkit does not specifically work with these files.

Example Usage

– Download all available KP data files for the in situ instruments between 1 January 2015 and 31 January 2015, inclusive:

```
>> pydivide.download_files(start_date='2015-01-01', end_date='2015-01-31', insitu=True)
```

– List all available CDF in-situ KP files on the server:

```
>> pydivide.download_files(insitu=True, list_files=True)
```

– Download all **new** (those files on the server that are not found in your local data directory) text (ASCII) IUVS KP files through 6 April 2015:

```
>> pydivide.download_files(iuvs=True, new_files=True, end_date='2015-04-
06')
```

– List to screen all available Level 2 data files for the SWIA instrument.

```
>> pydivide.download_files(instruments='swi', list_files=True, level='l2')
```

– List all available Level 2 data files for the SWIA instrument 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 available Level 2 data files for the NGIMS, STATIC, and EUV instruments that currently exist at the SDC server, but not in your local data directory.

```
>> pydivide.download_files(instruments=['ngi', 'sta', 'euv'],
new_files=True)
```

Required Arguments

Either **`insitu`**, **`iuvs`**, or at least one **`instrument`** **must** be specified.

List of all accepted Arguments

- **`insitu`**: Boolean variable. Search/download in-situ KP data files
- **`iuvs`**: Boolean variable. Search/download IUVS KP data files
- **`cdf_files`**: Boolean variable. Search/download CDF formatted data files (`*.cdf`)
- **`text_files`**: Boolean variable. Search/download ASCII formatted data files (`*.tab`)
- **`new_files`**: Boolean variable. Search/download only files that exist on the server
- **`list_files`**: Boolean variable. I.e., "dry run." List to standard output (usually, the screen) the files that would be downloaded based on the provided arguments; but do not download any data.
- **`update_prefs`**: Boolean variable. Before searching/downloading data, open up a dialogue window to allow the user to update the `mvn_toolkit_prefs.txt` file containing the location of the ROOT_DATA_DIR. Once the new path is selected, the search/download will proceed according to the remaining arguments.

- **`only_update_prefs`**: Boolean variable. As `update_prefs=True`; but do not attempt to search download and data.
- **`exclude_orbit_file`**: Boolean variable. Do not download an updated version of the orbit number file from http://naif.jpl.nasa.gov/naif/.
- **`filenames`**: scalar or an array of specific filename strings to search/download. The full filename must be provided (e.g., `mvn_kp_insitu_20150129_v00_r01.tab`); wildcards are **not** recognized.
- **`start_date`**: (format='`YYYY-MM-DD`') Search/download only data from `start_date` (inclusive) to present.
- **`end_date`**: (format='`YYYY-MM-DD`') Search/download only data from prior to `end_date` (inclusive).
- **`local_dir`**: Specify a directory to which to download files. This overrides (but does not overwrite) the target listed in the `mvn_toolkit_prefs.txt` file.
- **`instruments`**: Scalar or an array of 3-character string abbreviations (the instrument-specific directory names in the directory structure) of instruments for which the data are to be downloaded/searched.
- **`level`**: A string that specifies what level data you wish to download. This will only work when an instrument is specified. Options include l0, l1, l1a, l1b, l1c, l2, l2a, l2b, l3, l3a, l3b, l3c. By default, level 2 data is downloaded. Level 2 is generally the level where data is calibrated sufficiently for science use.

2.2. Reading data into Python Memory

2.2.1. read

Description

This procedure ingests a subset of locally available KP data into one or two data structures in python memory (depending on the provided arguments). The data structure variables thus produced are the primary inputs to the various plotting routines contained within this Toolkit.

In its simplest form, the read routine will quickly and efficiently return all MAVEN KP data with a single command. Through the use of a variety of keywords and options, however, the user may use the same routine to extract any subset of KP data according to their needs.

The first time a user calls **`read`**, a dialog will appear that ask for the location for ROOT_DATA_DIR, where on their machine MAVEN KP data is stored (see Required Directory Structure). If a user has already run a download routine, then this selection has already been made and saved.

After the first time, the Toolkit routines will remember this location and not prompt the user again. However, should the user wish to change this, or access

KP data in an alternate directory, they can re-enter this dialog via optional keywords (below).

Example Usage

– Read one day's worth (10 April 2015) of in-situ and IUVS KP data

```
>> insitu, iuvs = pydivide.read('2015-04-10')
```

– Read in-situ and IUVS KP data from a given start time (1:05:01 on 5 April 2015) to a given end time (14:22:11 on 22 April 2015). **NB, the procedure also accepts time/date strings in the following convention: 'yyyy-mm-dd hh:mm:ss'**.

```
>> insitu, iuvs = pydivide.read(['2015-04-10 01:05:01', '2015-05-22
14:22:11'])
```

– Read in five days (19 April 2015 to 24 April 2015) of in-situ and IUVS data, downloading any new files from the SDC server that are not already stored locally

```
>> insitu, iuvs = pydivide.read(['2015-04-19 00:00:00', '2015-04-24
00:00:00'], download_new=True)
```

Required Arguments

`time`:
The user is required to provide a constraint on the window for which data is to be retrieved from the local disk and stored in the local ipython data structure(s). These constraints may be provided in one of following formats:

1. As a date string (format='YYYY-MM-DD')
2. As a date/time string (format='YYYY-MM-DD HH:MM:SS')

At a minimum, the user is required to input a single time from which the KP data will be read. If the user inputs a single time, then the routine will read data for the default time period, beginning at the entered time. Alternately, the user may enter time as a two-element array that corresponds to the beginning and end times to be read. **N.B.**, *the default read period, if only a single time or orbit is entered by the user, is defined a single day (86400 seconds) if a date and time is entered.*

Returns

`insitu_output`:
This user-defined variable will be the name of the structure returned that contains all the in-situ instrument KP data as well as the spacecraft position and orientation information. The INSITU_OUTPUT structure is always filled with some data, even if only IUVS data is requested via keyword because it also contains the spacecraft position and orientation information that is needed for later visualization.

`iuvs_output`:

This user-defined variable will be the name of the structure retuned that contains all the IUVS KP data.

List of all accepted Arguments

- `time`: Must be the **first** argument after the procedure call.
  - Description above in "Required Arguments"
- `insitu_only`: Boolean variable. Setting this keyword will read in only the KP data from the in-situ instruments. If this keyword is set, the third required argument (`iuvs_output`) becomes unnecessary and is ignored.
- `Instruments`: Scalar or an array of 3-character string abbreviations (the instrument-specific directory names in the directory structure) of instruments for which the data are to be downloaded/searched.

2.2.2. read_model_results

Description

This procedure reads the results of a given simulation result into python memory as a dictionary object containing sub-dictionaries for metadata, dimension information, and model tracers. This function can read in any of the models currently on the MAVEN Science Data Center with the ".nc" extension:

https://lasp.colorado.edu/maven/sdc/public/pages/models.html

You will need to download the desired model before running this procedure.

Example Usage

– Read the Michigan group's Ionospheric model for Mars Season: 270° and Mean Solar Activity (check these).

```
>> model = pydivide.read_model_results('
/path/to/file/MGITM_L2270_F130.nc')
```

– Read the LATMOS group's Ionospheric model for Mars Season 270°, and Solar Maximum levels of solar activity.

```
>> model =
pydivide.read_model_results('/path/to/file/Heliosares_Ionos_LS270_SolMax.nc'
)
```

Required Arguments

`filename`:
The file name of the simulation result that is to be read in.

Returns

`output`:

This user-defined variable will be the name of the structure returned that contains the simulation results as a dictionary object. Roughly, the output will look like:

```
output
    \_ 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
    ...
    \_ variable
```

For example, the data for variable 1 can be accessed via the command:

```
>>output['variable1']['data']
```

And the subsolar longitude can be accessed with:

```
>> output['meta']['longsubsol']
```

2.3. Manipulating Key Parameter Data

Once the MAVEN KP data have been read into python memory, all data fields may be searched for values that fall within defined parameters. These searches may be run simultaneously. For example, it is possible to use these procedures (see examples below) to find all data records when the spacecraft was between altitudes of 1000km and 2000km, and the STATIC measured O^+ densities greater than 3000cm^{-3}.

2.3.1. insitu_search

Description

Search an existing in-situ data structure for data consistent with a set of requirements, and output a new data structure containing only those data consistent with the down-selection.

Example Usage

– Find all data records that have a STATIC measured O^+ density greater than 3000cm^{-3} and less than 1000000 cm^{-3}, and store the results in `insitu_new`.

```
>> insitu_new = pydivide.insitu_search(insitu,
parameter='static.oplus_density', min=3000, max=1000000)
```

Required Arguments

insitu_in:

The previously created in-situ KP data structure from which a subset of data are to be extracted.

Returns

insitu_out: This user-defined variable will be the name of the structure returned that contains all of the requested extracted subset of in-situ KP data.

Either **parameter** or **/list** **must** be present.

List of all accepted Arguments

- **insitu_in**: The input in-situ key parameter data structure produced by a previous call to **read** or **insitu_search**.
- **list**: Boolean variable. Display an ordered list of all parameters present in the input data structure, `insitu_in`. The items are listed by index, and by instrument followed by name. N.B., if this keyword is present, no down-selection of data based on any provided criteria will be performed, and there will be no output data structure.
- will be performed, and there will be no output data structure.
- **parameter**: Either the name or index (see the `/list` keyword) of the Key Parameter to be searched. This may be a single integer or string, if searching on a single parameter, or an array of integers or strings to search on multiple parameters simultaneously.
- **min**: This is the minimum value for a given search criteria. If not included, then the minimum is assumed to be negative infinity. Like the `parameter` keyword, this may be either a single value (if `parameter` is only a single name) or an array of values, where each corresponds to the respective `parameter` name.
- **max**: This is the maximum value for a given search criteria. If not included, then the maximum is assumed to be infinity. Like the `parameter` keyword, this may be either a single value (if `parameter` is only a single name) or an array of values, where each corresponds to the respective parameter name.

2.3.2. bin

Description

This routine provides the user with a convenient and efficient way to bin in-situ Key Parameters in one to eight defined dimensions. These guide dimensions may be any of the other Key Parameters within the data structure. The size of each bin is user definable and the output bins may be averages, standard deviations, and medians.

Example Usage

– Bin the STATIC O$^+$ characteristic energy according to spacecraft latitude and longitude, at one degree resolution in latitude, and two degree resolution in longitude. N.B., this assumes the entire KP data table has been read in.

```
>> output = pydivide.bin(insitu, parameter = 'static.oplus_char_energy',
bin_by=['spacecraft.geo_latitude', 'spacecraft.geo_longitude'] , avg=True,
binsize=[2,1])
```

– Bin the SWIA H$^+$ density according to spacecraft altitude, with 10km resolution, returning the averaged value (to `output`) and its standard deviation (to `output_std`) in each bin.

```
>> output, output_std = pydivide.bin(insitu, parameter =
'swia.hplus_density', bin_by='spacecraft.altitude', binsize=10 , avg=True,
std=True)
```

Required Arguments

insitu_in:
The first argument must be an in-situ key parameter data structure created from **read**.

to_bin:
The second argument lists the Key Parameter to be binned. Only one key parameter may be binned at a time by this procedure.

bin_by:
The third parameter lists the parameters – by index or name – by which to bin the requested key parameter.

binsize:
Keyword that accepts the array defining the bin size to use for each of the binning dimensions. The number of elements of `binsize` must equal the number of elements in `bin_by`.

NOTE: At least one of the following arguments must be set as well: avg, std, median, density.

Returns

output:

The requested key parameter binned according to the requested dimensions is output to this user-supplied variable. This procedure can output up to 4 variables, one for each of the keywords: avg, std, median, density.

List of all accepted Arguments

- **`avg`**: Boolean variable. Calculate the average within each bin and return the information in an array.
- **`std`**: Boolean variable. Calculate the standard deviation within each bin and return the information in an array.
- **`median`**: Boolean variable. Calculate the median within each bin and return the information in an array.
- **`density`**: Boolean variable. Array containing the number of values of the input data parameter that fall within each bin.
- **`mins`**: Array of minimum values for each provided binning (number of elements of `mins` must equal number of elements of `by_bin`).
- **`maxs`**: Array of maximum values for each provided binning (number of elements of `maxs` must equal number of elements of `by_bin`).

2.4. Plotting Key Parameter Data

These are procedures that will produce "traditional" Abscissa versus Ordinate plots of one or more parameters against time or altitude. In each case, the data structure previously created by **`read`** must be provided to the procedure as an argument.

2.4.1. plot

Description

Plot time series data from a MAVEN in-situ KP data structure. At present, this procedure will not work with IUVS KP data.

Example Usage

– Plot the H+ density from SWIA against time, identifying the parameter by name.

```
>> pydivide.plot(insitu, parameter='swia.hplus_density')
```

– Create three plots of three attributes all on the same plot

```
>> pydivide.plot(insitu, parameter=['swia.hplus_density',
'static.oplus_char_energy', 'spacecraft.altitude'] , SamePlot=True)
```

– List all KP data attributes present in a data structure. This is useful if you do not know the name of the attribute you wish to plot, or its index number.

```
>> pydivide.plot(insitu, list=True)
```

– Create a plot of H^+ density from SWIA, for a subset of the data contained within the read in data structure, using the `time` keyword to limit the plotted data to between 2 and 12 UTC on April 10th

```
>> pydivide.plot(insitu, parameter='swia.hplus_density', time=['2016-04-10
02:00:00', '2016-04-10 12:00:00'])
```

Required Arguments

kp_data:
The first argument must be an in-situ key parameter data structure created from **read**, or **insitu_search**.

parameter:
At least one KP parameter must be provided. This can be either passed as the name(s) of the parameter(s), such as `swia.hplus_density`, or `spacecraft.altitude`, or the index or indices of the parameters, the values of which can be obtained by using the **list** keyword, described below.

List of all accepted Arguments

- **kp_data**: The input in-situ key parameter data structure produced by a previous call to **read** or **search**.
- **parameter**: The Key Parameter value(s) to be plotted. The full list can be found in Appendix A, or at the command line by using the **/list** keyword (see below).
- **list**: Boolean variable. Display an ordered list of all parameters present in the data structure. The items are listed by index, and by instrument followed by name. If instead this keyword is assigned to a variable, then the list is stored in that variable as an array of strings.
- **time**: Define a range of times to be plotted, this keyword will accept strings as scalars or arrays. See the `time` keyword under "Required Arguments" in **read** for additional details.
- **SamePlot**: Boolean variable. Set true to plot everything on one chart, and false to create a stack of charts (True by default).
- **title:** String variable that sets the title of your plot.

2.4.2. altplot

Description

Generate a plot of in-situ key parameter data plotted versus altitude, rather than time. Altitude is plotted on the x-axis.

Example Usage

– Exactly the same as **plot**.

Required Arguments

See **plot** above.

List of all accepted Arguments

See **plot** above.

2.4.3. standards

Description

There are twenty-five standardized plots created from the in-situ KP data found on the MAVEN SDC website. This procedure generates a direct graphics window that contains all, or a subset of, those twenty-five plots, from the data provided by the user. Most of the plotted parameters are directly from the Key Parameter data; though in some cases, the plotted quantities of interest have been derived.

Example Usage

- Plot all twenty-five standard plots. Normally, you will not want to do this, since this command will generate a single window with 25 rows of very narrow plots. But it might be useful in a quick-look case, so this keyword has been retained.

```
>> pydivide.standards(insitu, all_plots=True)
```

- Generate a figure containing only three plots: the Magnetic field standard plot in Mars Solar Orbital coordinates (x, y, z, and magnitude) the standard spacecraft ephemeris information (sub-spacecraft latitude/longitude, subsolar latitude/longitude, local solar time, solar zenith angle, and Mars season), and the H^+/He^{++} and pick-up ion omni-directional fluxes from STATIC. Also customize the title.

```
>> pydivide.standards(insitu, mag_mso=True, eph_angle=True,
static_flux=True, title='Random Plots')
```

Required Arguments

insitu:

The first argument must be an in-situ key parameter data structure created from **read**.

Data Selection Keywords:

The following keywords identify which among the data that feed the standardized plots are to be presented in the generated figure. Each of these data selection keywords may be used in conjunction with any of the other data selection keywords. I.e., they are not exclusive, but additive keywords. **At least one of these must be set**:

- **all_plots**: Generate all 25 plots
- **euv**: EUV irradiance in each of three bands
- **mag_mso**: Magnetic field, MSO coordinates

  - `mag_geo`: Magnetic field, Geographic coordinates
  - `mag_cone`: Magnetic clock and cone angles, MSO coordinates
  - `mag_dir`: Magnetic field: radial, horizontal, northward, and eastward components
  - `ngims_neutral`: Neutral atmospheric component densities
  - `ngims_ions`: Ionized atmospheric component densities
  - `eph_angle`: Spacecraft ephemeris information
  - `eph_geo`: Spacecraft position in geographic coordinates
  - `eph_mso`: Spacecraft position in MSO coordinates
  - `swea`: electron parallel/anti-parallel fluxes
  - `sep_ion`: Ion Energy fluxes
  - `sep_electron`: Electron Energy fluxes
  - `wave`: Electric field wave power
  - `plasma_den`: Plasma densities
  - `plasma_temp`: Plasma Temperatures
  - `swia_h_vel`: H^+ Flow velocity in MSO coordinates from SWIA
  - `static_h_vel`: H^+ flow velocity in MSO coordinates from STATIC
  - `static_o2_vel`: O_2^+ flow velocity in MSO coords from STATIC
  - `static_flux`: H^+/He^{++} and Pick-up Ion omni-directional fluxes
  - `static_energy`: H^+/He^{++} and Pick-up Ion characteristic energies
  - `sun_bar`: Indication of whether MAVEN is in sunlight
  - `solar_wind`: solar wind dynamic pressure
  - `ionosphere`: Electron Spectrum shape parameter
  - `altitude`: Spacecraft altitude
  - `sc_pot`: Spacecraft potential

List of all accepted Arguments

- `insitu`: The input in-situ key parameter data structure produced by a previous call to `read` or `insitu_search`.
- `list_plots`: Display a list of all available plots and a brief description
- `title`: The title of the plots

2.4.4. corona

Description

Create altitude plots of the corona limb scans from IUVS Key Parameter files. These scans contain radiance and density data of various species.

Example Usage

– Plot all IUVS radiance and density data for all species on one plot

```
>> pydivide.corona(iuvs)
```

– Plot IUVS radiance and density data for H, O and O_1304 for orbit numbers 2540 and 2546. Also make the title Testing1234.

```
>> pydivide.corona(iuvs, species = ['H', 'O', 'O_1304'], orbit_num = [2540,
2546], title='Testing1234')
```

Required Arguments

kp_data:
The first argument must be an IUVS key parameter data structure created from **read**.

List of all accepted Arguments

- **kp_data**: The input IUVS key parameter data structure produced by a previous call to **read**.
- **density**: Boolean variable. Plot the density, default is true.
- **radiance**: Boolean variable. Plot the radiance, default is true.
- **orbit_num**: Integer or list of integers to specify the orbit number(s) plotted.
- **species**: A string or list of strings to only plot certain chemical species.
- **log**: Boolean variable. Set true to turn into a log plot.
- **SamePlot**: Boolean variable. Set true to plot everything on one chart, and false to create a stack of charts (True by default).

2.4.5. periapse

Description

Create altitude plots of the periapse limb scans from IUVS Key Parameter files. These scans contain radiance and density data of various species.

Example Usage

– Plot the N2 density and radiance data for orbit 1307 on a log plot

```
>> pydivide.periapse(iuvs, log=True, species='N2', orbit_num=1307)
```

Required Arguments

kp_data:
The first argument must be an IUVS key parameter data structure created from **read**.

List of all accepted Arguments

- **kp_data**: The input IUVS key parameter data structure produced by a previous call to **read**.
- **density**: Boolean variable. Plot the density, default is true.
- **radiance**: Boolean variable. Plot the radiance, default is true.
- **orbit_num**: Integer or list of integers to specify the orbit number(s) plotted.

- **obs_num**: Integer or list of integers to specify the observation number(s) plotted. There are up to 3 periapse observations per orbit.
- **species**: A string or list of strings to only plot certain chemical species.
- **log**: Boolean variable. Set true to turn into a log plot.
- **SamePlot**: Boolean variable. Set true to plot everything on one chart, and false to create a stack of charts (True by default).

2.5. Interpolating Model Results

These are procedures that are designed to read in outputs from simulations of the Martian Ionosphere, Exosphere, and Thermosphere, and interpolate them to either the spacecraft trajectory of a given in-situ key parameter data structure, or a given altitude.

2.5.1. interpol_model

Description

Given the structure containing the model results read in using **read_model_results**, and an in-situ key parameter data structure as input, this procedure produces a data structure containing all of the simulation's parameters interpolated to their values at the positions (latitude, longitude, altitude) taken from the spacecraft ephemeris.

Example Usage

– Interpolate all model tracers to the spacecraft trajectory using nearest neighbor interpolation.

```
>> results = pydivide.interpol_model(insitu, file =
'/path/to/file/Elew_18_06_14_t00600.nc', nearest=True)
```

Required Arguments

kp_data:
The first argument must be an in-situ key parameter data structure created from **read**, or **insitu_search**.

model:
The second argument provides the source of the simulation data to be interpolated to the spacecraft trajectory.

file:
If "model" is not provided, you can provide the full path to a model and the script will read in the model file.

Returns

output:
An array of values with the interpolated values at each point

List of all accepted Arguments

- **nearest**: Boolean variable. If set to True, instead of interpolating the nearby values, it will just return the value of the nearest neighbor point to the spacecraft

2.5.2. create_model_maps

Description

Given the path to a model file, this will generate a png file contour map of a model at a specific altitude. These maps can be used as a background in the 2D maps described in the next section.

Example Usage

– Interpolate all model tracers to the spacecraft trajectory using nearest neighbor interpolation.

```
>> pydivide.create_model_maps(altitude=170, file =
'/path/to/file/MGITM_LS090_F070_150812.nc')
```

Required Arguments

altitude:
The altitude of output map

file:
If "model" is not provided, you can provide the full path to a model and the script will read in the model file.

Returns

A png file will be created in the same directory as the model file provided to the function.

List of all accepted Arguments

- **nearest**: Boolean variable. If set to True, instead of interpolating the nearby values, it will just return the value of the nearest neighbor altitude.
- **linear**: Boolean variable. If set to True, will perform a simple linear interpolation between two altitude layers. Default is True.
- **transparency**: Numerical value between zero and one. Zero is a completely transparent map.
- **ct**: A string variable that sets the color tables to use. A list of allowed colortables can be found here: https://matplotlib.org/examples/color/colormaps_reference.html
- **fill:** Boolean variable. If set to True, fills in the contour levels rather than generate lines.
- **numContour:** Specify the number of contour lines. The default is 25.

2.6. Plotting of Key parameter data in 2D

2.6.1. map2d

Description

This routine will produce a 2d map of Mars, either in planetocentric or the MSO coordinate system, with the MAVEN orbital projection and a variety of basemaps. The spacecraft orbital path may be colored by a given in-situ Key Parameter data value.

Example Usage

– Plot the spacecraft altitude along the MAVEN orbital track along the surface.

```
>> pydivide.map2d(insitu, 'spacecraft.altitude')
```

– Plot the spacecraft altitude along the MAVEN orbital track along the surface, using the MOLA altimetry basemap, and also plot the path of the subsolar point.

```
>> pydivide.map2d(insitu, 'spacecraft.altitude', basemap='mola',
subsolar=True)
```

– Plot the CO_2^+ density from NGIMS along the MAVEN orbital track along the surface, limiting the displayed map domain to ±60° latitude and 90° to 270° longitude in MSO coordinates.

```
>> pydivide.map2d(insitu, 'ngims.co2plus_density’',  map_limit=[-
60,90,60,270], mso=True)
```

Required Arguments

insitu_data: The in-situ Key Parameter data structure
parameter: In-situ Key Parameter by which to color the spacecraft trajectory.

List of all accepted Arguments

- **parameter**: In-situ Key Parameter by which to color the spacecraft trajectory. I
- **time**: This keyword enables the user to plot a subset of the in-situ KP data. By default, the all of the data contained within the passed structure are plotted. The user can choose the plotted time range in the format [yyyy-mm-dd hh:mm:ss, yyyy-mm-dd hh:mm:ss].
- **basemap:** The name of the basemap to display upon which the spacecraft data will be overplotted. If not included, a basic lat/lon grid is used as the backdrop. Choices include:
    - 'MDIM': The Mars Digital Image Model.
    - 'MOLA': Mars Topography in color.
    - 'MOLA_BW': Mars topography in black and white.
    - 'MAG': Mars crustal magnetism.

    - '/path/to/file.png': User-defined basemap.
- `subsolar`: Boolean variable. Plot the path of the subsolar point along the surface of Mars.
- `mso`: Boolean variable. Plot using the MSO map projection.
- `list`: Boolean variable. Display an ordered list of all parameters present in the data structure. The items are listed by index, and by instrument followed by name. N.B., No data will be plotted if this keyword is provided; all plotting keywords will be ignored.
- `map_limit:` A list of 4 numbers that specify the bounding box displayed on the map: [x0, y0, x1, y1]
- `alpha:` A number that sets the transparency of the trajectory.
- `title:` A string that sets the title of the generated plot

A. Appendix: KP Data Structures in the ToolKit

The Key Parameter data read in by `read` are held within python as a dictionary of instruments, with each dictionary key referring to a "dataframe" object. The name of the dictionary is defined by the user at the time that `read` is used to read the raw Key Parameter data files, or at the time that `insitu_search` is used to select a subset of an existing structure of Key Parameter data. For the rest of this section, this name will be assumed to be **`kp_data`**. Not alldictionary keys/ dataframes will be present, according to the subsetting performed either during the `read` or `search`. Full in-situ and IUVS Key Parameter data structures have the following form. Dataframe columns are in lower case, while dictionary keys are listed in CAPS and boldface. For more information, refer to Table 13 of the MAVEN In-Situ Instruments Key Parameters SIS document.

- **KP_DATA**
    - **INSITU**
        - time_string
        - time
        - orbit
        - inbound/outbound flag
        - **SPACECRAFT**
            - GEO_x
            - GEO_y
            - GEO_z
            - MSO_x
            - MSO_y
            - MSO_z
            - GEO_longitude
            - GEO_latitude
            - SZA
            - Local_time

- Altitude
- Attitude_geo_x
- Attitude_geo_y
- Attitude_geo_z
- Attitude_mso_x
- Attitude_mso_y
- Attitude_mso_z
- Mars_season
- Mars_sun_distance
- Subsolar_point_GEO_longitude
- Subsolar_point_GEO_latitude
- SubMars_point_Solar_longitude
- SubMars_point_solar_latitude

▪ **APP**
- Attitude_geo_x
- Attitude_geo_y
- Attitude_geo_z
- Attitude_mso_x
- Attitude_mso_y
- Attitude_mso_z

▪ **LPW**
- Electron_density
- Electron_density_qual_min
- Electron_density_qual_max
- Electron_temperature
- Electron_temperature_qual_min
- Electron_temperature_qual_max
- Spacecraft_potential
- Spacecraft_potential_qual_min
- Spacecraft_potential_qual_max
- Ewave_low
- Ewave_low_qual
  - "0" = perfect
  - "100" = 100% error
- Ewave_mid
- Ewave_mid_qual
  - "0" = perfect
  - "100" = 100% error
- Ewave_high
- Ewave_high_qual
  - "0" = perfect
  - "100" = 100% error

▪ **EUV**

    - Quality flags contain: “0” = good data, “1” = off-nominal pointing, “2” = aperture closed
  - irradiance_low
  - irradiance_low_qual
  - irradiance_mid
  - irradiance_mid_qual
  - irradiance_lyman
  - irradiance_lyman_qual
- **MAG**
    - All quality flags are 0=normal data; 1=abnormal data.
  - MSOx
  - MSOx_qual
  - MSOy
  - MSOy_qual
  - MSOz
  - MSOz_qual
  - GEOx
  - GEOx_qual
  - GEOy
  - GEOy_qual
  - GEOz
  - GEOz_qual
  - RMS
  - RMS_qual
- **NGIMS**
    - All densities are abundances or upper limits in cc^{-1}. All quality flags are % error (1 sigma). Quality flag of “-1” indicates density is an upper limit.
  - He_density
  - He_density_qual
  - O_density
  - O_density_qual
  - CO_density
  - CO_density_qual
  - N2_density
  - N2_density_qual
  - NO_density
  - NO_density_qual
  - Ar_density
  - Ar_density_qual
  - CO2_density
  - CO2_density_qual
  - O2plus_density
  - O2plus_density_qual

- CO2plus_density
- CO2plus_density_qual
- NOplus_density
- NOplus_density_qual
- Oplus_density
- Oplus_density_qual
- CONplus_density
- CONplus_density_qual
- Cplus_density
- Cplus_density_qual
- OHplus_density
- OHplus_density_qual
- Nplus_density
- Nplus_density_qual

▪ **SEP**

  – Energy fluxes and their quality flags are in units of eV/cm^2/s. Quality flags are standard uncertainty in ion energy flux based on Poisson statistics.

- Ion_energy_flux_1
- Ion_energy_flux_1_qual
- Ion_energy_flux_2
- Ion_energy_flux_2_qual
- Ion_energy_flux_3
- Ion_energy_flux_3_qual
- Ion_energy_flux_4
- Ion_energy_flux_4_qual
- Electron_energy_flux_1
- Electron_energy_flux_1_qual
- Electron_energy_flux_2
- Electron_energy_flux_2_qual
- Electron_energy_flux_3
- Electron_energy_flux_3_qual
- Electron_energy_flux_4
- Electron_energy_flux_4_qual
- Look_direction_1_MSOx
- Look_direction_1_MSOy
- Look_direction_1_MSOz
- Look_direction_2_MSOx
- Look_direction_2_MSOy
- Look_direction_2_MSOz
- Look_direction_3_MSOx
- Look_direction_3_MSOy
- Look_direction_3_MSOz
- Look_direction_4_MSOx

        - Look_direction_4_MSOy
        - Look_direction_4_MSOz
- **STATIC**
    - STATIC Quality Flag
    - CO2plus_density
    - CO2plus_density_qual
    - Oplus_density
    - Oplus_density_qual
    - O2plus_density
    - O2plus_density_qual
    - CO2plus_temperatrure
    - CO2plus_temperature_qual
    - Oplus_temperature
    - Oplus_temperature_qual
    - O2plus_temperature
    - O2plus_temperature_qual
    - O2plus_flow_v_appx
    - O2plus_flow_v_appx_qual
    - O2plus_flow_v_appy
    - O2plus_flow_v_appy_qual
    - O2plus_flow_v_appz
    - O2plus_flow_v_appz_qual
    - O2plus_flow_v_MSOx
    - O2plus_flow_v_MSOx_qual
    - O2plus_flow_v_MSOy
    - O2plus_flow_v_MSOy_qual
    - O2plus_flow_v_MSOz
    - O2plus_flow_v_MSOz_qual
    - Hplus_omni_flux
    - Hplus_char_energy
    - Hplus_char_energy_qual
    - Heplus_omni_flux
    - Heplus_char_energy
    - Heplus_char_energy_qual
    - Oplus_omni_flux
    - Oplus_char_energy
    - Oplus_char_energy_qual
    - O2plus_omni_flux
    - O2plus_char_energy
    - O2plus_char_energy_qual
    - Hplus_char_dir_MSOx
    - Hplus_char_dir_MSOy
    - Hplus_char_dir_MSOz
    - Hplus_char_angular_width

- Hplus_char_angular_width_qual
- Dominant_Pickup_ion_char_dir_MSOx
- Dominant_Pickup_ion_char_dir_MSOx_qual
- Dominant_Pickup_ion_char_dir_MSOy
- Dominant_Pickup_ion_char_dir_MSOy_qual
- Dominant_Pickup_ion_char_dir_MSOz
- Dominant_Pickup_ion_char_dir_MSOz_qual
- Dominant_Pickup_ion_char_angular_width
- Dominant_Pickup_ion_char_angular_width_qual

▪ **SWEA**

  – Unless noted, quality flags all reflect 'Statistical Uncertainty"

- Solarwind_e_density
- Solarwind_e_density_qual
- Solarwind_e_temperature
- Solarwind_e_temperature_qual
- Electron_parallel_flux_low
- Electron_parallel_flux_low_qual
- Electron_parallel_flux_mid
- Electron_parallel_flux_mid_qual
- Electron_parallel_flux_high
- Electron_parallel_flux_high_qual
- Electron_antiparallel_flux_low
- Electron_antiparallel_flux_low_qual
- Electron_antiparallel_flux_mid
- Electron_antiparallel_flux_mid_qual
- Electron_antiparallel_flux_high
- Electron_antiparallel_flux_high_qual
- Electron_spectrum_shape
- Electron_spectrum_shape _qual
  – Floating point number from 0 to 1. "Zero" means no evidence for ionospheric electrons, "one" means no evidence for solar wind electrons.

▪ **SWIA**

  – Unless noted, quality flags are 0 for bad, and 1 for good, indicating whether the distribution is well-measured and decommutation parameters are definite.

- Hplus_density
- Hplus_density_qual
- Hplus_flow_velocity_MSOx
- Hplus_flow_velocity_MSOx_qual
- Hplus_flow_velocity_MSOy
- Hplus_flow_velocity_MSOy_qual
- Hplus_flow_velocity_MSOz
- Hplus_flow_velocity_MSOz_qual

        - Hplus_temperature
        - Hplus_temperature_qual
        - Solarwind_dynamic_pressure
        - Solarwind_dynamic_pressure_qual
    - **IUVS**
        - **PERIAPSE# (NOTE: “#” will either be 1, 2 or 3)**
            - Time_start
            - Time_stop
            - Scale_height
            - scale_height_unc
            - density
            - density_unc
            - density_sys_unc
            - Radiance
            - Radiance_unc
            - Radiance_sys_unc
            - Temperature
            - Temperature_unc
            - Sza
            - Local_time
            - Lat
            - Lon
            - Lat_mso
            - Lon_mso
            - Orbit_number
            - Mars_season_ls
            - Spacecraft_geo
            - Spacecraft_mso
            - Sun_geo
            - Sun_mso
            - Spacecraft_geo_longitude
            - Spacecraft_geo_latitude
            - Spacecraft_mso_longitude
            - Spacecraft_mso_latitude
            - Subsolar_point_geo_longitude
            - Subsolar_point_geo_latitude
            - Subsolar_point_mso_longitude
            - Subsolar_point_mso_latitude
            - Spacecraft_sza
            - Spacecraft_local_time
            - Spacecraft_altitude
            - Mars_sun_distance
        - **CORONA_LORES_HIGH**
            - Time_start

- Time_stop
- Scale_height
- scale_height_err
- density
- density_err
- Radiance
- Radiance_err
- Temperature
- Temperature_err
- Sza
- Local_time
- Lat
- Lon
- Lat_mso
- Lon_mso
- Orbit_number
- Mars_season_ls
- Spacecraft_geo
- Spacecraft_mso
- Sun_geo
- Sun_mso
- Spacecraft_geo_longitude
- Spacecraft_geo_latitude
- Spacecraft_mso_longitude
- Spacecraft_mso_latitude
- Subsolar_point_geo_longitude
- Subsolar_point_geo_latitude
- Subsolar_point_mso_longitude
- Subsolar_point_mso_latitude
- Spacecraft_sza
- Spacecraft_local_time
- Spacecraft_altitude
- Mars_sun_distance

- **APOAPSE**
  - Time_start
  - Time_stop
  - Ozone_depth
  - Ozone_depth_err
  - Auroral_index
  - Dust_depth
  - Dust_depth_err
  - Radiance
  - Radiance_err

- Sza_bp
- Local_time_bp
- Sza
- Local_time
- Lat
- Lon
- Lat_mso
- Lon_mso
- Orbit_number
- Mars_season_ls
- Spacecraft_geo
- Spacecraft_mso
- Sun_geo
- Sun_mso
- Spacecraft_geo_longitude
- Spacecraft_geo_latitude
- Spacecraft_mso_longitude
- Spacecraft_mso_latitude
- Subsolar_point_geo_longitude
- Subsolar_point_geo_latitude
- Subsolar_point_mso_longitude
- Subsolar_point_mso_latitude
- Spacecraft_sza
- Spacecraft_local_time
- Spacecraft_altitude
- Mars_sun_distance