PyTplot Documentation

Release

Laboratory for Atmospheric and Space Physics

July 20, 2017

Table of Contents

1 Introduction

Pytplot is a python package which aims to mimic the functionality of the IDL "tplot" libraries. The primary routine (tplot) generates HTML files for the specified plots, and automatically opens the files in a Qt interface.

These plots have several user interaction tools built in, such as zooming and panning. The can be exported as standalone HTML files (to retain their interactivity) or as static PNG files.

Pytplot can be used in python scripts, or interactively through IPython and the Jupyter notebook.

1.1 Install Python

You will need the Anaconda distribution of Python 3 in order to run pytplot.

Anaconda comes with a suite of packages that are useful for data science.

1.2 Running PyTplot

To start using pytplot in a similar manner to IDL tplot, start up an interactive environment through the terminal command:

```
ipython
```

or, if you prefer the jupyter interactive notebook:

```
jupyter notebook
```

then, just import the package by typing the command:

```
import pytplot
```

2 Storing Data in Memory

2.1 store_data

store_data (*name*, *data=None*, *delete=False*)

This function creates a "Tplot Variable" based on the inputs, and stores this data in memory. Tplot Variables store all of the information needed to generate a plot.

Parameters:

name : *str*

Name of the tplot variable that will be created

data : *dict*

A python dictionary object.

'x' should be a 1-dimensional array that represents the data's x axis. Typically this data is time, represented in seconds since epoch (January 1st 1970)
'y' should be the data values. This can be 2 dimensions if multiple lines or a spectrogram are desired.
'v' is optional, and is only used for spectrogram plots. This will be a list of bins to be used. If this is provided, then 'y' should have dimensions of x by z.
'x' and 'y' can be any data format that can be read in by the pandas module. Python lists, numpy arrays, or any pandas data type will all work.

delete : *bool, optional*
Deletes the tplot variable matching the "name" parameter

Note If you want to combine multiple tplot variables into one, simply supply the list of tplot variables to the "data" parameter. This will cause the data to overlay when plotted.

Returns:
None

Examples:

```
>>> # Store a single line
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
```

```
>>> # Store a two lines
>>> x_data = [1,2,3,4,5]
>>> y_data = [[1,5],[2,4],[3,3],[4,2],[5,1]]
>>> pytplot.store_data("Variable2", data={'x':x_data, 'y':y_data})
```

```
>>> # Store a specrogram
>>> x_data = [1,2,3]
>>> y_data = [ [1,2,3] , [4,5,6], [7,8,9] ]
>>> v_data = [1,2,3]
>>> pytplot.store_data("Variable3", data={'x':x_data, 'y':y_data,
'v':v_data})
```

```
>>> # Combine two different line plots
>>> pytplot.store_data("Variable1and2", data=['Variable1', 'Variable2'])
```

2.2 tplot_rename

`tplot_rename` (*old_name*, *new_name*)
This function will rename tplot variables that are already stored in memory.

Parameters:

old_name : *str*
Old name of the Tplot Variable

new_name : *str*
New name of the Tplot Variable

Returns:
None

Examples:

```
>>> # Rename Variable 1 to Variable 2
>>> import pytplot
>>> pytplot.tplot_rename("Variable1", "Variable2")
```

2.3 del_data

del_data (*name*)
This function will delete tplot variables that are already stored in memory.

Parameters:

name : *str*
Name of the tplot variable to be deleted

Returns:
None

Examples:

```
>>> # Delete Variable 1
>>> import pytplot
>>> pytplot.del_data("Varaible1")
```

3 Retrieveing Data

3.1 get_data

get_data (*name*)
This function will get extract the data from the Tplot Variables stored in memory.

Parameters:

name : *str*
Name of the tplot variable

Returns:
time_val : pandas dataframe index data_val : list

Examples:

```
>>> # Retrieve the data from Variable 1
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> time, data = pytplot.get_data("Variable1")
```

3.2 get_timespan

get_timespan (*name*)

This function will get extract the time span from the Tplot Variables stored in memory.

Parameters:

name : *str*
Name of the tplot variable

Returns:

time_begin : *float*
The beginning of the time series

time_end : *float*
The end of the time series

Examples:

```
>>> # Retrieve the time span from Variable 1
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> time1, time2 = pytplot.get_timespan("Variable1")
```

3.3 get_ylimits

get_ylimits (*name, trg=None*)

This function will get extract the y limites from the Tplot Variables stored in memory.

Parameters:

name : *str*
Name of the tplot variable

trg : *list, optional*
The time range that you would like to look in

Returns:

ymin : *float*
The minimum value of y

ymax : *float*
The maximum value of y

Examples:

```
>>> # Retrieve the y limits from Variable 1
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> y1, y2 = pytplot.get_ylimits("Variable1")
```

3.4 tplot_names

tplot_names ( )

This function will print out and return a list of all current Tplot Variables stored in the memory.

Parameters:

None

Returns:

list : *list of str*

A list of all Tplot Variables stored in the memory

Examples:

```
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> tnames = pyplot.tplot_names()
0 : Variable 1
```

4 Setting Plot Options

4.1 options

options (*name*, *option*, *value*)

This function allows the user to set a large variety of options for individual plots.

Parameters:

name : *str*

Name of the tplot variable

option : *str*

The name of the option. See section below

value : *str/int/float/list*

The value of the option. See section below.

Options:

Options	Value type	Notes
Color	str/list	Red, Orange, Yellow, Green, Blue, etc
Colormap	str/list	https://matplotlib.org/examples/color/colormaps_reference.html
Spec	int	1 sets the Tplot Variable to spectrogram mode, 0 reverts
Alt	int	1 sets the Tplot Variable to altitude plot mode, 0 reverts
Map	int	1 sets the Tplot Variable to latitude/longitude mode, 0 reverts
ylog	int	1 sets the y axis to log scale, 0 reverts

zlog	int	1 sets the z axis to log scale, 0 reverts (spectrograms only)
legend_names	list	A list of strings that will be used to identify the lines
line_style	str	solid_line, dot, dash, dash_dot, dash_dot_dot_dot, long_dash
name	str	The title of the plot
panel_size	flt	Number between (0,1], representing the percent size of the plot
basemap	str	Full path and name of a background image for "Map" plots
alpha	flt	Number between [0,1], gives the transparancy of the plot lines
yrange	flt list	Two numbers that give the y axis range of the plot
zrange	flt list	Two numbers that give the z axis range of the plot
ytitle	str	Title shown on the y axis
ztitle	str	Title shown on the z axis. Spec plots only.

Returns:

None

Examples:

```
>>> # Change the y range of Variable1
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> pytplot.options('Variable1', 'yrange', [2,4])
```

```
>>> # Change Variable1 to use a log scale
>>> pytplot.options('Variable1', 'ylog', 1)
```

```
>>> # Change the line color of Variable1
>>> pytplot.options('Variable1', 'ylog', 1)
```

4.2 tplot_options

`tplot_options` (*option*, *value*)

This function allows the user to set several global options for the generated plots.

Parameters:

option : *str*

The name of the option. See section below

value : *str/int/float/list*

The value of the option. See section below.

Options:

Options	Value type	Notes
title	str	Title of the the entire output
title_size	int	Font size of the output
wsize	[int, int]	[height, width], pixel size of the plot window
title_align	int	Offset position in pixels of the title

var_label	srt	Name of the tplot variable to be used as another x axis
alt_range	[flt, flt]	The min and max altitude to be plotted on all alt plots

Returns:

None

Examples:

```
>>> # Set the plot title
>>> import pytplot
>>> pytplot.tplot_options('title', 'SWEA Data for Orbit 1563')
```

```
>>> # Set the window size
>>> pytplot.tplot_options('wsize', [1000,500])
```

4.3 timebar

timebar (*t*, *varname=None*, *databar=False*, *delete=False*, *color='black'*, *thick=1*, *dash=False*)

This function will add a vertical bar to all time series plots. This is useful if you want to bring attention to a specific time.

Parameters:

t : *flt/list*

The time in seconds since Jan 01 1970 to place the vertical bar. If a list of numbers are supplied, multiple bars will be created. If "databar" is set, then "t" becomes the point on the y axis to place a horizontal bar.

varname : *str/list, optional*

The variable(s) to add the vertical bar to. If not set, the default is to add it to all current plots.

databar : *bool, optional*

This will turn the timebar into a horizontal data bar. If this is set True, then variable "t" becomes the point on the y axis to place a horizontal bar.

delete : *bool, optional*

If set to True, at lease one varname must be supplied. The timebar at point "t" for variable "varname" will be removed.

color : *str*

The color of the bar

thick : *int*

The thickness of the bar

dash : *bool*

If set to True, the bar is dashed rather than solid

Returns:

None

Examples:

```
>>> # Place a green time bar at 2017-07-17 00:00:00
>>> import pytplot
```

```
>>> pytplot.timebar(1500249600, color='grean)
```

```
>>> # Place a dashed data bar at 5500 on the y axis
>>> pytplot.timebar(5500, dashed=True, databar=True)
```

4.4 timespan

timespan (*t1, dt, keyword='days'*)

This function will set the time range for all time series plots. This is a wrapper for the function "xlim" to better handle time axes.

Parameters:

t1 : *flt/str*

The time to start all time series plots. Can be given in seconds since epoch, or as a string in the format "YYYY-MM-DD HH:MM:SS"

dt : *flt*

The time duration of the plots. Default is number of days.

keyword : *str*

Sets the units of the "dt" variable. Days, hours, minutes, and seconds are all accepted.

Returns:

None

Examples:

```
>>> # Set the timespan to be 2017-07-17 00:00:00 plus 1 day
>>> import pytplot
>>> pytplot.timespan(1500249600, 1)
```

```
>>> # The same as above, but using different inputs
>>> pytplot.timespan("2017-07-17 00:00:00", 24, keyword='hours')
```

4.5 timestamp

timestamp (*val*)

This function will turn on a time stamp that shows up at the bottom of every generated plot.

Parameters:

val : *str*

A string that can either be 'on' or 'off'.

Returns:

None

Examples:

```
>>> # Turn on the timestamp
>>> import pytplot
>>> pytplot.timestamp('on')
```

4.6 xlim

xlim (*min*, *max*)

This function will set the x axis range for all time series plots

Parameters:

min : *flt*

The time to start all time series plots. Can be given in seconds since epoch, or as a string in the format "YYYY-MM-DD HH:MM:SS"

max : *flt*

The time to end all time series plots. Can be given in seconds since epoch, or as a string in the format "YYYY-MM-DD HH:MM:SS"

Returns:

None

Examples:

```
>>> # Set the timespan to be 2017-07-17 00:00:00 plus 1 day
>>> import pytplot
>>> pytplot.xlim(1500249600, 1500249600 + 86400)
```

```
>>> # The same as above, but using different inputs
>>> pytplot.xlim("2017-07-17 00:00:00", "2017-07-18 00:00:00")
```

4.7 ylim

ylim (*name*, *min*, *max*)

This function will set the y axis range displayed for a specific tplot variable.

Parameters:

name : *str*

The name of the tplot variable that you wish to set y limits for.

min : *flt*

The start of the y axis.

max : *flt*

The end of the y axis.

Returns:

None

Examples:

```
>>> # Change the y range of Variable1
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> pytplot.ylim('Variable1', 2, 4)
```

4.8 zlim

zlim (*name*, *min*, *max*)

This function will set the z axis range displayed for a specific tplot variable. This is only used for spec plots, where the z axis represents the magnitude of the values in each bin.

Parameters:

name : *str*
The name of the tplot variable that you wish to set z limits for.

min : *flt*
The start of the z axis.

max : *flt*
The end of the z axis.

Returns:
None

Examples:

```
>>> # Change the z range of Variable1
>>> import pytplot
>>> x_data = [1,2,3]
>>> y_data = [ [1,2,3] , [4,5,6], [7,8,9] ]
>>> v_data = [1,2,3]
>>> pytplot.store_data("Variable3", data={'x':x_data, 'y':y_data,
'v':v_data})
>>> pytplot.zlim('Variable1', 2, 3)
```

5 Plotting Data

5.1 tplot

tplot (*name*, *var_label=None*, *auto_color=True*, *interactive=False*, *combine_axes=True*, *nb=False*, *save_file=None*, *gui=False*, *qt=True*)

This is the function used to display the tplot variables stored in memory. The default output is to show the plots stacked on top of one another inside a GUI window. The GUI window has the option to export the plots in either PNG or HTML formats.

Note This plotting routine uses the python Bokeh library, which creates plots using HTML and Javascript. Bokeh is technically still in beta, so future patches to Bokeh may require updates to this function.

Parameters:

name : *str / list*
List of tplot variables that will be plotted

var_label : *str, optional*
The name of the tplot variable you would like as a second x axis.

auto_color : *bool, optional*
Automatically color the plot lines.

interactive : *bool, optional*
If True, a secondary interactive plot will be generated next to spectrogram plots. Mousing over the spectrogram will display a slice of data from that time on the interactive chart.

combine_axis : *bool, optional*
If True, the axes are combined so that they all display the same x range. This also enables scrolling/zooming/panning on one plot to affect all of the other plots simultaneously.

nb : *bool, optional*
If True, the plot will be displayed inside of a current Jupyter notebook session.

save_file : *str, optional*
A full file name and path. If this option is set, the plot will be automatically saved to the file name provided in an HTML format. The plots can then be opened and viewed on any browser without any requirements.

gui : *bool, optional*
If True, then this function will output the 2 HTML components of the generated plots as string variables. This is useful if you are embedded the plots in your own GUI. For more information, see http://bokeh.pydata.org/en/latest/docs/user_guide/embed.html

qt : *bool, optional*
If True, then this function will display the plot inside of the Qt window. From this window, you can choose to export the plots as either an HTML file, or as a PNG.

Returns:
None

Examples:

```
>>> #Plot a single line
>>> import pytplot
>>> x_data = [2,3,4,5,6]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> pytplot.tplot("Variable1")
```

```
>>> #Display two plots
>>> x_data = [1,2,3,4,5]
>>> y_data = [[1,5],[2,4],[3,3],[4,2],[5,1]]
>>> pytplot.store_data("Variable2", data={'x':x_data, 'y':y_data})
>>> pytplot.tplot(["Variable1", "Variable2"])
```

```
>>> #Display 2 plots, using Variable1 as another x axis
>>> x_data = [1,2,3]
>>> y_data = [ [1,2,3] , [4,5,6], [7,8,9] ]
>>> v_data = [1,2,3]
>>> pytplot.store_data("Variable3", data={'x':x_data, 'y':y_data,
'v':v_data})
```

```
>>> pytplot.options("Variable3", 'spec', 1)
>>> pytplot.tplot(["Variable2", "Variable3"], var_label='Variable1')
```

```
>>> #Plot all 3 tplot variables, sending the output to an HTML file
>>> pytplot.tplot(["Variable1", "Variable2", "Variable3"],
save_file='C:/temp/pytplot_example.html')
```

```
>>> #Plot all 3 tplot variables, sending the HTML output to a pair of
strings
>>> div, component = pytplot.tplot(["Variable1", "Variable2",
"Variable3"], gui=True)
```

6 Saving and Restoring Sessions

6.1 tplot_save

tplot_save (*names, filename=None*)

This function will save tplot variables into a single file by using the python "pickle" function. This file can then be "restored" using tplot_restore. This is useful if you want to end the pytplot session, but save all of your data/options. All variables and plot options can be read back into tplot with the "tplot_restore" command.

Parameters:

names : *str/list*

A string or a list of strings of the tplot variables you would like saved.

filename : *str, optional*

The filename where you want to save the file.

Returns:

None

Examples:

```
>>> # Save a single tplot variable
>>> import pytplot
>>> x_data = [1,2,3,4,5]
>>> y_data = [1,2,3,4,5]
>>> pytplot.store_data("Variable1", data={'x':x_data, 'y':y_data})
>>> pytplot.ylim('Variable1', 2, 4)
>>> pytplot.save('Variable1', filename='C:/temp/variable1.pytplot')
```

6.2 tplot_restore

tplot_restore (*filename*)

This function will restore tplot variables that have been saved with the "tplot_save" command.

Note This function is compatible with the IDL tplot_save routine. If you have a ".tplot" file generated from IDL, this procedure will restore the data contained in the file. Not all plot options will transfer over at this time.

Parameters:

filename : *str*

The file name and full path generated by the "tplot_save" command.

Returns:

None

Examples:

```
>>> # Restore the saved data from the tplot_save example
>>> import pytplot
>>> pytplot.restore('C:/temp/variable1.pytplot')
```