PyDV Documentation

Release 3.0.4

Kevin Griffin, Mason Kwiat, Douglas Miller, Rachael Brooks

May 07, 2021

CONTENTS

1 Users 3

2 Developers 55

3 Indices and tables 87

Python Module Index 89

Index 91

The Python Data Visualizer (PyDV) is a replacement for ULTRA writtern in python. PyDV allows the presentation, manipulation, and analysis of 1D data sets, i.e. (x,y) pairs. Presentation refers to the capability to display, and make hard copies of data plots. Manipulation refers to the capability to excerpt, shift, and scale data sets. Analysis refers to the capability to combine data sets in various ways to create new data sets.

The principal object with which PyDV works is the curve. A curve is an object which consists of an array of x values, an array of y values, a number of points (the length of the x and y arrays), and an ASCII label. PyDV operates on curves.

CHAPTER

ONE

USERS

1.1 Release Notes

1.1.1 3.0.4

- Changed “ultra” to “pydv” in error messages.
- Changed “majorminor” to “both” in grid argument.
- Force interp num to be integers.

1.1.2 3.0.3

Enhancements

- Ability to read Sina curve sets.
- Ability to specify which curves will appear in the legend.

1.1.3 3.0.2

Bug Fixes

- Zoom settings from the User Interface are now persisted throughout the application.

Enhancements

- Allow simple math operations on curves that have been read in but not yet plotted.
- Enhanced the **image** command to allow the user to define the image resolution and transparency.
- Added the **menur** command that works like the **menu** command with the addition of allowing *start* and *stop* indices.
- Added the **listr** command that works like the **list** command with the addition of allowing *start* and *stop* indices.
- Added the **plotlayout** command that allows the user to adjust the plot layout parameters.

1.1.4 3.0.1

Bug Fixes

- Fixed the @ symbol range bug
- Fixed guilims command

Enhancements

- Added **labelcurve** command that allows users to add curve letter to the legend label
- Enhanced the **divide**, **multiply**, **add**, and **subtract** commands to support dividing by a real number
- Suppressed user warnings
- Added **border** command that turns plot border on or off
- Updated the link in the **About** dialog popup

Changes for PyDV Developers

- Moved repository to the LLNL Github organization

1.1.5 3.0

Python 3 port with bug fixes and a lot of minor code refactoring.

1.1.6 2.4.3

Bug Fixes

- Fixed the piece-wise constant integration bug
- Fit command bug fixed
- Added Doug Miller's fix for retrieving a curve by label

Enhancements

- Added the minorticks command. Minor ticks can now be made visible.
- Added the xtickcolor command to change the color of major and minor ticks on the x~axis
- Added the ytickcolor command to change the color of major and minor ticks on the y~axis
- Updated the xticklength command to support minor ticks
- Updated the xtickwidth command to support minor ticks
- Updated the yticklength command to support minor ticks
- Updated the ytickwidth command to support minor ticks
- Added the gridcolor command
- Added the gridstyle command

- Added the gridwidth command
- Added the random command
- Added the rev command
- Added the sort command
- Added the alpha command
- Added the gaussian command

1.1.7 2.4.2

Bug Fixes

- Fixed the FFT command to produce two curves for the complex and imaginary part like Ultra
- Corrected the 'off by one' index error for curves named with the '@' symbol
- The xtick commands now display the correct help information
- errorbar command works now

Enhancements

- Implemented the convol, convolb, and convolc commands like their Ultra equivalent
- Added the intensize and extensive commands
- Added the correl command
- Added the system command to allow passing commands to the operating system
- Allow the user to optionally throw away zero and negative values when using the log commands
- Updated the integrate command to use a new color for the new curve it produces
- Added the getymax/getymin commands
- Enhanced the convol commands to add the number of points to the label

1.1.8 2.4

Bug Fixes

- Display updates correctly after running a batch file
- Draw style command can now draw all of the step options (pre, post, mid)
- Fixed the lableFileNames command from adding the filename more than once
- For certain commands that create a new curve, ensured that all attributes were copied to the new curve
- Fixed a bug in the integrate command that ignored the upper and lower limits
- Fixed a bug in the subsample command. Also, enhanced it so the user needs to specify a curve(s)

Enhancements

- Added the dupx command
- Added the xindex command
- Added the append~curves command
- Added the average command
- Added the max command
- Added the min command
- Added the get~attributes command
- Added the stats command
- Removed unused 'Plot Name' column in the Menu dialog
- Piecewise constant plots are now supported
- The font size and font color can be changed only for the legend
- The getx and gety command now returns all the x~ and y~values for a given y~ or x~value respectively
- The .pdvrc file supports more default values (fontsize, lnwidth)
- Improved the syntax of the legend command
- Added the bkgcolor command that allows the use to change the background color of the plot, window, or both
- The menu and curve regex option is now done over the curve name and filename
- Both the x- and y-column can be specified when reading in an ULTRA text file

1.1.9 2.3

Bug Fixes

- Fixed the **getx** and **gety** commands to work with horizontal/vertical lines.
- Fixed the sign issue with subtracting curves.

Enhancements

- Added window to display the contents of the **list** command. You can also delete curves from this window.
- Allow figure size specification in **create_plot**.
- Enhanced the **list** command to use a regex for filtering the list.
- Display the **menu** command contents in a popup window. Can also plot and delete curves from the popup window.
- Enhanced the read command to filter the curves as they are read in. Also, the user can specify the number of matched curves to read in.
- Added the **getlabel** command that prints the given curve's label.
- Added the **getnumpoints** command that prints the given curve's number of points.
- Added the **kill** command that deletes specified entries from the menu.

1.1.10 2.2

Bug Fixes

- Fixed the interpolation function for two curves
- Got alias command working again by adding back the removed import new line

Enhancements

- Added convolvef math command that performs a convolution of two curves using the Fast Fourier transform method
- Added Fast Fourier Transform math command
- Added disp and dispx commands for displaying the curves y~ and x~values
- Enhanced the read command to optionally use a regular expression to filter the curves that are read in
- Created a method in the PyDV Python interface to filter curves using a regular expression
- Added handlelength command to control the length of lines in the legend
- Allow namewidth to be changed from the .pdvrc file
- Added documentation for the .pdrc file format

1.1.11 2.1

Bug Fixes

- Addition operator dropping down into the Python interpreter after execution
- Error when reading ULTRA files with an extra data item
- Geometry command not working

Enhancements

- Changing plot properties from the GUI are now persistent
- Added fontcolor command
- Added guilims command
- Added linemarker command
- Added markeredgecolor command
- Added markerfacecolor command
- Added drawstyle command

Changes for PyDV Developers

- Created compile and test scripts
- Integrated compile and test scripts with Bamboo

1.1.12 2.0

Bug Fixes

- Plot limits auto adjust fixed
- Cleaned up a lot of typos and errors in the help documentation

Enhancements

- Legend can be moved by clicking on it and dragging with the mouse
- Added style command that allows user to change the style of the plot
- Added showstyles command that lists all the available styles
- Added sinhx math command
- Added support for reading .csv files
- Created a Python interface (pydvpy) for PyDV functionality
- Turned Latex off by default
- Changed backend to Qt4Agg
- New 'About' dialogs with links to the PyDV confluence page, developer contact information and copyright details

Changes for PyDV Developers

- Setup a documentation framework with SPHINX
- Added an application icon

1.2 .pdvrc File Format

Note: The **.pdvrc** file allows the user to initialize PyDV settings at startup time. PyDV expects the **.pdvrc** file to be located in the User's Home Directory. The format of the **.pdvrc** file is 'key=value'. Below are all of the currently recognized settings.

1.2.1 xlabel=label

Set the label for the x-axis to label.

1.2.2 ylabel=label

Set the label for the y-axis to label.

1.2.3 title=str

Set the title for the plot to str.

1.2.4 key=ON | OFF

Show the legend if key=ON, otherwise hide it if key=OFF.

1.2.5 letters=ON | OFF

Show or hide letter markers on plotted curves.

1.2.6 geometry=val1 val2 val3 val4

Change the PyDV window size and location in pixels where val1 is the x-size, val2 is the y-size, val3 is the x-location, and val4 is the y-location.

1.2.7 initcommands=filename

Specify a file to run the initial commands.

1.2.8 namewidth=width

Change the width of teh first column of the menu and lst output.

1.2.9 fontsize=size

Change the font size

1.2.10 lnwidth=width

Change the default line width of the curves.

1.3 Getting started

This section gives a tutorial introduction to PyDV. A sample session is run which highlights the basic PyDV commands.

Note: In PyDV commands, spaces are used to delimit items on the input line. More precisely items on a command line are either space delimited, are preceded by a left parenthesis if the first item in a list, or terminated by a right parenthesis if the last item in a list. Semicolons may be used to stack multiple commands on a single interactive input line. In interactive mode, ranges of curve numbers or data-id's may be indicated using colon notation. For example, a:f or 5:9

1.3.1 Run PyDV (LLNL)

```
/usr/gapps/pydv/pdv      # Current version

/usr/gapps/pydv/3.0/pdv # Version specific
```

1.3.2 Run PyDV With a Command File

```
/usr/gapps/pydv/pdv -i <command-file>
```

1.3.3 Run PyDV In Column Format Mode

```
/usr/gapps/pydv/pdv -gnu <file.gnu>
```

1.3.4 Create a curve consisting of a straight line y=x over the interval (0,6.28).

```
[PyDV]: span 0 6.28
```

1.3.5 Print a list of curves on the display.

```
[PyDV]: lst
```

1.3.6 Take the sine of curve A

```
[PyDV]: sin a
```

1.3.7 Take the product of curves A and B

```
[PyDV]: * A B
```

1.3.8 Write all of the curves to an ASCII file

```
[PyDV]: save foo.txt a:c
```

1.3.9 Read file foo.txt

```
[PyDV]: rd foo.txt
```

1.3.10 Display curves in all open files

```
[PyDV]: menu
```

1.3.11 Display curves from the menu on

```
[PyDV]: cur 1 3
```

1.3.12 Shift curve A by one unit to the right

```
[PyDV]: dx A 1
```

1.3.13 Delete curve A from the display

```
[PyDV]: del A
```

1.3.14 Exit PyDV

```
[PyDV]: quit
```

1.4 I/O Commands

These commands access disk files either for reading or writing.

Note: **< >** = Required user input.

[< >] = Optional user input.

[PyDV] = Python Data Visualizer command-line prompt.

1.4.1 read

Read curves from the specified ASCII file and optionally filter by regex. The next available prefix (see the prefix command) is automatically assigned the menu index of the first curve in each data file read. For column oriented (.gnu) files optionally specify the x-column number before the file name. **Shortcut: rd**

```
[PyDV]: read [(regex) matches] [x-col] <filename>
```

1.4.2 readcsv

Read CSV data file. The next available prefix (see the prefix command) is automatically assigned the menu index of the first curve in each data file read. For column oriented (.gnu) files optionally specify the x-column number before the file name. **Shortcut: rdcsv**

```
[PyDV]: readcsv [x-col] <filename.csv>
```

1.4.3 readsina

Read all curves from Sina data file. PyDV assumes there is only one record in the Sina file, and if there are more than one then PyDV only reads the first. PyDV also assumes there is only one independent variable per curve_set; if there are more than one then PyDV may exhibit undefined behavior. The next available prefix (see the prefix command) is automatically assigned the menu index of the first curve in each data file read. **Shortcut: rdsina**

```
[PyDV]: readsina <filename.json>
```

1.4.4 run

Execute a list of commands from a file.

```
[PyDV]: run <filename>
```

1.4.5 save

Saves curves to a file in text format.

```
[PyDV]: save <filename> <curve-list>
```

1.4.6 savecsv

Save curves to file in comma separated values (CSV) format. Assumes all curves have the same x basis.

```
[PyDV]: savecsv <filename> <curve-list>
```

1.5 Math Operations

Note: **< >** = Required user input.

[ ] = Optional user input.

[PyDV]: = Python Data Visualizer command-line prompt.

1.5.1 abs

Take the absolute value of the y values of the curves. Modifies the existing curve.

```
[PyDV]: abs <curve-list>
```

1.5.2 absx

Take the absolute value of the x values of the curves. Modifies the existing curve.

```
[PyDV]: absx <curve-list>
```

1.5.3 acos

Take arccosine of y values of curves

```
[PyDV]: acos <curve-list>
```

1.5.4 acosh

Take hyperbolic arccosine of y values of curves.

```
[PyDV]: acosh <curve-list>
```

1.5.5 acoshx

Take hyperbolic arccosine of x values of curves.

```
[PyDV]: acoshx <curve-list>
```

1.5.6 acosx

Take arccosine of x values of curves .. code:

```
[PyDV]: acosx <curve-list>
```

1.5.7 add

Take the sum of curves. If the optional *value* is specified it will add the y-values of the curves by *value* (equivalent to using the **dy** command). **Shortcut: +**

Note: Adding curves by a number modifies the curve. If you want to create a new curve then copy the original curve first using the copy command.

```
[PyDV]: add <curve-list> [value]
```

1.5.8 add_h

Adds curves that have been read from a file but not yet plotted. **list-of-menu-numbers** are the index values displayed in the first column of the **menu** command.

```
[PyDV]: add_h <list-of-menu-numbers>
```

1.5.9 alpha

Find the alpha.

```
[PyDV]: alpha <calculated-a> <calculated-i> <response> [# points]
```

1.5.10 asin

Take arcsine of y values of curves

```
[PyDV]: asin <curve-list>
```

1.5.11 asinx

Take arcsine of x values of curves .. code:

```
[PyDV]: asinx <curve-list>
```

1.5.12 asinh

Take hyperbolic arcsine of y values of curves.

```
[PyDV]: asinh <curve-list>
```

1.5.13 asinhx

Take hyperbolic arcsine of x values of curves.

```
[PyDV]: asinhx <curve-list>
```

1.5.14 atan

Take arctangent of y values of curves.

```
[PyDV]: atan <curve-list>
```

1.5.15 atanx

Take arctangent of x values of curves.

```
[PyDV]: atanx <curve-list>
```

1.5.16 atan2

Take atan2 of two curves.

```
[PyDV]: atan2 <curve1> <curve2>
```

1.5.17 atanh

Take hyperbolic arctangent of y values of curves.

```
[PyDV]: atanh <curve-list>
```

1.5.18 atanhx

Take hyperbolic arctangent of x values of curves.

```
[PyDV]: atanhx <curve-list>
```

1.5.19 average

Average the specified curvelist over the intersection of their domains.

```
[PyDV]: average <curve-list>
```

1.5.20 convolve

Computes the convolution of the two given curves. This is similar to the slower **convolc** method in ULTRA that uses direct integration and minimal interpolations. **Shortcut:** convol

```
[PyDV]: convolve <curve1> <curve2> [points]
```

1.5.21 convolveb

Computes the convolution of the two given curves and normalizing the second curve by the area under the curve. This computes the integrals directly which avoid padding and aliasing problems associated with FFT methods (it is however slower). **Shortcut:** convolb

```
[PyDV]: convolveb <curve1> <curve2> [points]
```

1.5.22 convolvec

Computes the convolution of the two given curves with no normalization. This computes the integrals directly which avoid padding and aliasing problems associated with FFT methods (it is however slower). **Shortcut:** convolb

```
[PyDV]: convolveb <curve1> <curve2> [points]
```

1.5.23 correl - 2.4.2

Computes the cross-correlation of two curves.

```
[PyDV]: correl <curve1> <curve2>
```

1.5.24 cos

Take the cosine of the y values of the curves.

```
[PyDV]: cos <curve-list>
```

1.5.25 cosx

Take the cosine of the x values of the curves.

```
[PyDV]: cosx <curve-list>
```

1.5.26 cosh

Take hyperbolic cosine of y values of curves.

```
[PyDV]: cosh <curve-list>
```

1.5.27 coshx

Take hyperbolic cosine of x values of curves.

```
[PyDV]: coshx <curve-list>
```

1.5.28 dx

Shift x values of curves by a constant.

```
[PyDV]: dx <curve-list> <value>
```

1.5.29 dy

Shift y values of curves by a constant.

```
[PyDV]: dy <curve-list> <value>
```

1.5.30 divide

Take quotient of curves. If the optional *value* is specified it will divide the y-values of the curves by *value* (equivalent to using the **divy** command). **Shortcuts:** /, div

---

Note: Dividing curves by a number modifies the curve. If you want to create a new curve then copy the original curve first using the copy command.

```
[PyDV]: divide <curve-list> [value]
```

1.5.31 divide_h

Divides curves that have been read from a file but not yet plotted. **list-of-menu-numbers** are the index values displayed in the first column of the **menu** command.

```
[PyDV]: divide_h <list-of-menu-numbers>
```

1.5.32 divx

Procedure: Divide x values of curves by a constant.

```
[PyDV]: divx <curve-list> <value>
```

1.5.33 divy

Procedure: Divide y values of curves by a constant.

```
[PyDV]: divy <curve-list> <value>
```

1.5.34 error-bar

Plot error bars on the given curve.

```
[PyDV]: errorbar <curve> <y-error-curve> <y+error-curve> [x-error-curve x+error-
↪curve] [point-skip]
```

1.5.35 errorrange

Plot shaded error region on given curve, **Shortcut: error-range**

```
[PyDV]: errorrange <curve> <y-error-curve> <y+error-curve>
```

1.5.36 exp

e**y, exponentiate y values of the curves.

```
[PyDV]: exp <curve-list>
```

1.5.37 expx

e**y, exponentiate x values of the curves.

```
[PyDV]: expx <curve-list>
```

1.5.38 fft

Compute the one-dimensional discrete Fourier Transform for the y-values of the curves.

```
[PyDV]: fft <curve-list>
```

1.5.39 fftx

Compute the one-dimensional discrete Fourier Transform for the x-values of the curves.

```
[PyDV]: fftx <curve-list>
```

1.5.40 gaussian

Generate a gaussian function.

```
[PyDV]: gaussian <amplitude> <width> <center> [<# points> [<# half-widths>]]
```

1.5.41 j0

Take the zeroth order Bessel function of y values of curves

```
[PyDV]: j0 <curve-list>
```

1.5.42 j0x

Take the zeroth order Bessel function of x values of curves

```
[PyDV]: j0x <curve-list>
```

1.5.43 j1

Take the first order Bessel function of y values of curves

```
[PyDV]: j1 <curve-list>
```

1.5.44 j1x

Take the first order Bessel function of x values of curves

```
[PyDV]: j1x <curve-list>
```

1.5.45 jn

Take the nth order Bessel function of y values of curves

```
[PyDV]: jn <curve-list> n
```

1.5.46 jnx

Take the nth order Bessel function of x values of curves

```
[PyDV]: jnx <curve-list> n
```

1.5.47 L1

Makes new curve that is the L1 norm of two args; the L1 norm is integral(|curve1 - curve2|) over the interval [xmin,xmax]. Also prints value of integral to command-line.

```
[PyDV]: L1 <curve1> <curve2> [<xmin> <xmax>]
```

1.5.48 L2

Makes new curve that is the L2 norm of two args; the L2 norm is integral((curve1 - curve2)**2)**(1/2) over the interval [xmin,xmax]. Also prints value of integral to command-line.

```
[PyDV]: L2 <curve1> <curve2> [<xmin> <xmax>]
```

1.5.49 log

Take the natural logarithm of the y values of the curves. If the optional argument *keep-neg-vals* is set to false, then zero and negative y-values will be discarded. *keep-neg-vals* is true by default. **Shortcut: ln**

```
[PyDV]: log <curve-list> [keep-neg-vals: True | False]
```

1.5.50 logx

Take the natural logarithm of the x values of the curves. If the optional argument *keep-neg-vals* is set to false, then zero and negative x-values will be discarded. *keep-neg-vals* is true by default. **Shortcut: lnx**

```
[PyDV]: logx <curve-list> [keep-neg-vals: True | False]
```

1.5.51 log10

Take the base 10 logarithm of the y values of the curves. If the optional argument *keep-neg-vals* is set to false, then zero and negative y-values will be discarded. *keep-neg-vals* is true by default.

```
[PyDV]: log10 <curve-list> [keep-neg-vals: True | False]
```

1.5.52 log10x

Take the base 10 logarithm of the x values of the curves. If the optional argument *keep-neg-vals* is set to false, then zero and negative y-values will be discarded. *keep-neg-vals* is true by default.

```
[PyDV]: log10x <curve-list> [keep-neg-vals: True | False]
```

1.5.53 makeintensive - 2.4.2

Set the y-values such that y[i] = y[i] / (x[i+1] - x[i]). **Shortcut: mkint**

```
[PyDV]: makeintensive <curve-list>
```

1.5.54 makeextensive - 2.4.2

Set the y-values such that y[i] = y[i] * (x[i+1] - x[i]). **Shortcut: mkext**

```
[PyDV]: makeextensive <curve-list>
```

1.5.55 max

Makes a new curve with max y values of curves passed in curvelist.

```
[PyDV]: max <curve-list>
```

1.5.56 min

Makes a new curve with min y values of curves passed in curvelist.

```
[PyDV]: min <curve-list>
```

1.5.57 mx

Scale the x values of the curves by a fixed value.

```
[PyDV]: mx <curve-list> <value>
```

1.5.58 multiply

Take the product of curves. If the optional *value* is specified it will multiply the y-values of the curves by *value* (equivalent to using the **my** command). **Shortcuts:** *, mult

---

Note: Multiplying curves by a number modifies the curve. If you want to create a new curve then copy the original curve first using the copy command.

```
[PyDV]: multiply <curve-list> [value]
```

1.5.59 multiply_h

Multiplies curves that have been read from a file but not yet plotted. **list-of-menu-numbers** are the index values displayed in the first column of the **menu** command.

```
[PyDV]: multiply_h <list-of-menu-numbers>
```

1.5.60 my

Scale the y values of the curves by a fixed value.

```
[PyDV]: my <curve-list> <value>
```

1.5.61 norm

Makes a new curve that is the norm of two args. Also prints the value of the integral to command line.

```
[PyDV]: norm <curve> <curve> <p> <xmin> <xmax>
```

Note: The p-norm is (integral((curve1 - curve2)**p)**(1/p) over the interval [xmin, xmax], where p = order.

1.5.62 powa

Raise a fixed value, a, to the power of the y values of the curves.

```
[PyDV]: powa <curve-list> <a>
```

1.5.63 powax

Raise a fixed value, a, to the power of the x values of the curves.

```
[PyDV]: powax <curve-list> <a>
```

1.5.64 powr

Raise the y values of the curves to a fixed power p.

```
[PyDV]: powr <curve-list> <p>
```

1.5.65 powrx

Raise the x values of the curves to a fixed power p.

```
[PyDV]: powrx <curve-list> <p>
```

1.5.66 recip

Take the reciprocal of the y values of the curves.

```
[PyDV]: recip <curve-list>
```

1.5.67 recipx

Take the reciprocal of the x values of the curves.

```
[PyDV]: recipx <curve-list>
```

1.5.68 sin

Take the sine of the y values of the curve

```
[PyDV]: sin <curve-list>
```

1.5.69 sinx

Take the sine of the x values of the curve

```
[PyDV]: sinx <curve-list>
```

1.5.70 sinh

Take the hyperbolic sine of the y values of the curve

```
[PyDV]: sinh <curve-list>
```

1.5.71 smooth

Smooth the curve to the given degree.

```
[PyDV]: smooth <curve-list> [smooth-factor]
```

1.5.72 sqr

Take the square of the y values of the curves.

```
[PyDV]: sqr <curve-list>
```

1.5.73 sqrx

Take the square of the x values of the curves.

```
[PyDV]: sqrx <curve-list>
```

1.5.74 sqrt

Take the squre root of the y values of the curves.

```
[PyDV]: sqrt <curve-list>
```

1.5.75 sqrtx

Take the squre root of the x values of the curves.

```
[PyDV]: sqrtx <curve-list>
```

1.5.76 subtract

Take the difference of curves. A single curve can be specified, resulting in the negating of its y-values. If the optional *value* is specified it will subtract the y-values of the curves by *value* (similar to using the **dy** command). **Shortcuts:** –, sub

---

Note: Subtracting curves by a number modifies the curve. If you want to create a new curve then copy the original curve first using the copy command.

```
[PyDV]: subtract <curve-list> [value]
```

1.5.77 subtract_h

Subtracts curves that have been read from a file but not yet plotted. **list-of-menu-numbers** are the index values displayed in the first column of the **menu** command.

```
[PyDV]: subtract_h <list-of-menu-numbers>
```

1.5.78 tan

Take the tangent of y values of curves

```
[PyDV]: tan <curve-list>
```

1.5.79 tanx

Take the tangent of x values of curves

```
[PyDV]: tanx <curve-list>
```

1.5.80 tanh

Take the hyperbolic tangent of y values of curves

```
[PyDV]: tanh <curve-list>
```

1.5.81 tanhx

Take the hyperbolic tangent of x values of curves

```
[PyDV]: tanhx <curve-list>
```

1.5.82 xmax

Filter out points in curves whose x-values greater than limit

```
[PyDV]: xmax <curve-list> <limit>
```

1.5.83 xmin

Filter out points in curves whose x-values less than limit

```
[PyDV]: xmin <curve-list> <limit>
```

1.5.84 y0

Take the zeroth order Bessel function of the second kind of the y values of the curves.

```
[PyDV]: y0 <curve-list>
```

1.5.85 y0x

Take the zeroth order Bessel function of the second kind of the x values of the curves.

```
[PyDV]: y0x <curve-list>
```

1.5.86 y1

Take the first order Bessel function of the second kind of the y values of the curves.

```
[PyDV]: y1 <curve-list>
```

1.5.87 y1x

Take the first order Bessel function of the second kind of the x values of the curves.

```
[PyDV]: y1x <curve-list>
```

1.5.88 ymax

Filter out points in curves whose y-values greater than limit

```
[PyDV]: ymax <curve-list> <limit>
```

1.5.89 ymin

Filter out points in curves whose y-values less than limit

```
[PyDV]: ymin <curve-list> <limit>
```

1.5.90 yminmax

Trim the selected curves. **Shortcut: ymm**

```
[PyDV]: yminmax <curve-list> <low-limit> <high-lim>
```

1.5.91 yn

Take the nth order Bessel function of the second kind of y values of curves

```
[PyDV]: yn <curve-list> <n>
```

1.5.92 ynx

Take the nth order Bessel function of the second kind of x values of curves

```
[PyDV]: ynx <curve-list> <n>
```

1.5.93 derivative

Take the derivative of curves. **Shortcut:** der

```
[PyDV]: derivative <curve-list>
```

1.5.94 diffMeasure

Compare two curves. For the given curves a fractional difference measure and its average is computed

```
[PyDV]: diffMeasure <curve1> <curve2> [tolerance]
```

1.5.95 fit

Make new curve that is polynomial fit to argument. n=1 by default, logy means take log(y-values) before fitting, logx means take log(x-values) before fitting

```
[PyDV]: fit <curve> [n] [logx] [logy]
```

1.5.96 integrate

Compute the definite integral of each curve in the list over the specified domain. **Shortcut:** int

```
[PyDV]: integrate <curve-list> [low-limit high-limit]
```

1.5.97 span

Generates a straight line of slope 1 and y intercept 0 in the specified domain with an optional number of points

```
[PyDV]: span <xmin> <xmax> [points]
```

1.5.98 vs

Plot the range of the first curve against the range of the second curve

```
[PyDV]: vs <curve1> <curve2>
```

1.6 Environmental Inquiry Commands

These functions are provided to gain access to information about the state of the PyDV session. Information such as the state of global variables and system help packages is made available through these functions.

Note: **< >** = Required user input.

[ ] = Optional user input.

[PyDV]: = Python Data Visualizer command-line prompt.

1.6.1 help

Return infroamtion about the specified command, variable, or command category. If no argument is supplied, return a list of available commands.

```
[PyDV]: help [command]
```

1.6.2 list

Return a list of the curves currently displayed. A regular expression may be supplied for matching against the curve label to be listed. **Shortcut: lst**

```
[PyDV]: list <label-pattern>
```

1.6.3 listr

Return a list of the curves currently displayed in range from **start** to **stop**. If **stop** is not specified, it will be set to the end of the plot list. **Shortcut: lstr**

```
[PyDV]: listr <start> [stop]
```

1.6.4 listannot

List current annotations.

```
[PyDV]: listannot
```

1.6.5 menu

Return a selection of the curves available for plotting. A regular expression may be supplied for matching against the curve label to be listed.

```
[PyDV]: menu <label-pattern>
```

1.6.6 menur

Return a selection of the curves available for plotting in the range from **start** to **stop**. If **stop** is not specified, it will be set to the end of the curve list.

```
[PyDV]: menur <start> [stop]
```

1.6.7 system

Allows passing commands to the operating system. **Shortcut: ! or shell**

```
[PyDV]: system <command>
```

1.7 Curve Inquiry Commands

These functions provide a mechanism for obtaining information about specified curves.

Note: **< >** = Required user input.

[ ] = Optional user input.

[PyDV]: = Python Data Visualizer command-line prompt.

1.7.1 disp

Display the y-values in the specified curve(s).

```
[PyDV]: disp <curve-list>
```

1.7.2 dispx

Display the x-values in the specified curve(s).

```
[PyDV]: dispx <curve-list>
```

1.7.3 eval

Evaluate mathematical operations on curves.

```
[PyDV]: eval <curve-operations>
```

1.7.4 getattributes

Return (to the terminal) the attributes of a curve, such as: color, style, width, etc.

```
[PyDV]: getattributes <curve>
```

1.7.5 getdomain

Return (to the terminal) the domains for the list of curves.

```
[PyDV]: getdomain <curve-list>
```

1.7.6 getlabel

Return (to the terminal) the given curve's label.

```
[PyDV]: getlabel <curve>
```

1.7.7 getnumpoints

Display the number of points for the given curve.

```
[PyDV]: getnumpoints <curve>
```

1.7.8 getrange

Return (to the terminal) the ranges for the list of curves.

```
[PyDV]: getrange <curve-list>
```

1.7.9 getx

Return the x values for a given y

```
[PyDV]: getx <curve-list> <y-value>
```

1.7.10 gety

Return the y values for a given x

```
[PyDV]: gety <curve-list> <x-value>
```

1.7.11 stats

Calculate the mean and standard deviation for the curves and display the results on the terminal.

```
[PyDV]: stats <curve-list>
```

1.7.12 getymin - 2.4.2

Return the minimum y-value for the curve within the specified domain. If no domain is given, then the full domain range is used.

```
[PyDV]: getymin <curve> [<xmin> <xmax>]
```

1.7.13 getymax - 2.4.2

Return the maximum y-value for the curve within the specified domain. If no domain is given, then the full domain range is used.

```
[PyDV]: getymax <curve> [<xmin> <xmax>]
```

1.8 Environmental Control Commands

These functions allow you to manipulate the environment of PyDV on a global level. They are useful to avoid repeated use of other commands or to change the state of PyDV in dramatic ways.

Note: **< >** = Required user input.

[ ] = Optional user input.

[PyDV]: = Python Data Visualizer command-line prompt.

1.8.1 alias

Define a synonym for an existing command.

```
[PyDV]: alias <command> <alias>
```

1.8.2 custom

Load a file of custom functions to extend PyDV. Functions must be of the form **'def do_commandname(self, line): ...'**

```
[PyDV]: custom <file-name>
```

1.8.3 debug

Show debug tracebacks if True

```
[PyDV]: debug on | off
```

1.8.4 drop

Start the Python Interactive Console

```
[PyDV]: drop
```

1.8.5 erase

Erase all curves on the screen but leave the limits untouched. **Shortcut: era**

```
[PyDV]: erase
```

1.8.6 kill

Delete the specified entries from the menu.

```
[PyDV]: kill [all | number-list]
```

1.8.7 namewidth

Change the width of the first column of the **menu** and **lst** output.

```
[PyDV]: namewidth <integer>
```

1.8.8 quit

Exit PyDV. **Shortcut: q**

```
[PyDV]: quit
```

1.9 Plot Control Commands

These functions control the plotting characteristics of PyDV which affect all displayed curves.

Note: **< >** = Required user input.

[ ] = Optional user input.

[PyDV]: = Python Data Visualizer command-line prompt.

1.9.1 annot

Display text on the plot at point (x, y).

```
[PyDV]: annot <text> <x> <y>
```

1.9.2 border

Show the border if **on** or **1**, otherwise hide the border. The **color-name** determines the color of the border. By default, the border color is black.

```
[PyDV]: border <on | 1 | off | 0> [color-name]
```

1.9.3 bkgcolor

Change the background color of the plot, window, or both

```
[PyDV]: bkgcolor <[plot | window] color-name | reset>
```

1.9.4 dashstyle

Set the style of dash or dot dash lines. The python list is a list of integers, alternating how many pixels to turn on and off, for example:

[2, 2] : Two pixels on, two off (will result in a dot pattern).

[4, 2, 2, 2] : 4 on, 2 off, 2 on, 2 off (results in a dash-dot pattern).

[4, 2, 2, 2, 4, 2] : Gives a dash-dot-dash pattern.

[4, 2, 2, 2, 2, 2] : Gives a dash-dot-dot pattern.

See matplotlib 'set_dashes' command for more information.

```
[PyDV]: dashstyle <curve-list> <[...]>
```

1.9.5 dataid

Show curve identifiers if True. **Alternative Form: data-id**

```
[PyDV]: dataid <on | off>
```

1.9.6 delannot

Delete annotations from list.

```
[PyDV]: delannot <number-list-of-annotations>
```

1.9.7 domain

Set the domain for plotting. Using de (for default) will let the curves determine the domain.

```
[PyDV]: domain <low-lim> <high-lim>

              OR

[PyDV]: domain de
```

1.9.8 fontcolor

Change the font color of given plot component.

```
[PyDV]: fontcolor [<component: xlabel | ylabel | title | xaxis | yaxis>] <color-name>
```

1.9.9 fontsize

Change the font size of given component, or overall scaling factor.

```
[PyDV]: fontsize [<component: title | xlabel | ylabel | key | tick | curve |␣
↪annotation>] <numerical-size | small | medium | large | default>
```

1.9.10 fontstyle

Set the fontstyle family.

```
[PyDV]: fontstyle <serif | sans-serfif | monospace>
```

1.9.11 geometry

Change the PyDV window size and location in pixels.

```
[PyDV]: geometry <xsize> <ysize> <xlocation> <ylocation>
```

1.9.12 grid

Set whether or not to draw grid lines on the graph. Default is off.

```
[PyDV]: grid <on | off>
```

1.9.13 gridcolor

Set the color of the grid.

```
[PyDV]: gridcolor <color-name>
```

1.9.14 gridstyle

Set the line style for the grid.

```
[PyDV]: gridstyle <style: solid | dash | dot | dashdot>
```

1.9.15 gridwidth

Set the grid line width in points.

```
[PyDV]: gridwidth <width>
```

1.9.16 guilims

Set whether or not to use the GUI min/max values for the X and Y limits. Default is off.

```
[PyDV]: guilims <on | off>
```

1.9.17 handlelength

Adjust the length of the line(s) in the legend.

```
[PyDV]: handlelength <length>
```

1.9.18 image

Save the current figure to an image file. All parameters are optional. The default value for **filename** is *plot*, the default value for **filetype** is *pdf* and the default value for **transparent** is *False*. **dpi** is the resolution in dots per inch and the default value is the figure's dpi value.

```
[PyDV]: image [filename=plot] [filetype=pdf: png | ps | pdf | svg]␣
↪[transparent=False: True | False] [dpi]
```

1.9.19 label

Change the key and list label for a curve.

```
[PyDV]: label <curve> <new-label>
```

1.9.20 labelcurve

Add curve letter to the legend label if **on**, otherwise hide curve letter if **off**.

```
[PyDV]: labelcurve <on | off>
```

1.9.21 labelfilenames

Change the key and list labels for all curves to append the filename. This command only affects the curves plotted at the time of execution. Any new curve will need to have this command run again to append the filename.

```
[PyDV]: labelfilenames
```

1.9.22 latex

Use LaTeX font rendering if True

```
[PyDV]: latex on | off
```

1.9.23 legend

Show/Hide the legend with on | off or set legend position with ur, ul, ll, lr, cl, cr, uc, lc. Specify the number of columns to use in the legend. Specify curves to add to or remove from the legend using the *hide* or *show* keywords followed by the ids of the curves. Note: Commands after *hide*/*show* will not be processed, so make sure these are the last in the command list. **Shortcuts: leg, key**

```
[PyDV]: legend <on | off> [position] [<number of columns>] [<show/hide curve ids]
```

1.9.24 lnstyle

Set the line style of the specified curves.

```
[PyDV]: lnstyle <curve-list> <style: solid | dash | dot | dotdash>
```

1.9.25 lnwidth

Set the line widths of the specified curves. A line width of 0 will give the thinnest line which the host graphics system supports.

```
[PyDV]: lnwidth <curve-list> <width>
```

1.9.26 marker

Set the marker symbol and scale (optionally) for scatter plots. You can also use any of the matplotlib supported marker types as well. See the matplotlib documentation on markers for further information.

```
[PyDV]: marker <curve-list> <marker-style: + | . | circle | square | diamond> [marker-
↪size]
```

1.9.27 minorticks

Minor ticks are not visible by default. On will make the minor ticks visible and off will hide the minor ticks.

```
[PyDV]: minorticks <on | off>
```

1.9.28 movefront

Move the given curves so they are plotted on top.

```
[PyDV]: movefront <curve-list>
```

1.9.29 plotlayout

Adjust the plot layout parameters. Where **left** is the position of the left edge of the plot as a fraction of the figure width, **right** is the position of the right edge of the plot, as a fraction of the figure width, **top** is the position of the top edge of the plot, as a fraction of the figure height and **bottom** is the position of the bottom edge of the plot, as a fraction of the figure height. Alternatively, *de* will revert to the default plot layout values.

If no arguments are given, the plot's current layout settings will be displayed.

```
[PyDV]: plotlayout [<left> <right> <top> <bottom> || de]
```

1.9.30 range

Set the range for plotting. Using de (for default) will let the curves determine the range. **Shortcut: ran**

```
[PyDV]: range <low-lim> <high-lim> | de
```

1.9.31 style

Use matplotlib style settings from a style specification. The style name of **default** (if available) is reserved for reverting back to the default style settings.

```
[PyDV]: style <style-name>
```

1.9.32 ticks

Set the maximum number of major ticks on the axes.

```
[PyDV]: ticks <quantity> | de
```

1.9.33 title

Set a title for the plot

```
[PyDV]: title <title-name>
```

1.9.34 update

Update the plot after each command if True.

```
[PyDV]: update on | off
```

1.9.35 xlabel

Set a label for the x axis

```
[PyDV]: xlabel <label-name>
```

1.9.36 xlogscale

Set log scale on or off for the x-axis. **Alternative Form: x-log-scale**, **Shortcut: xls**

```
[PyDV]: xlogscale <on | off>
```

1.9.37 xtickcolor

Set the color of the ticks on the x-axis. Default is to apply to major ticks only.

```
[PyDV]: xticks <de | color> [which: major | minor | both]
```

1.9.38 xticks

Set the locations of major ticks on the x-axis

```
[PyDV]: xticks de | <number> | <list of locations> | <list of locations, list of␣
↪labels>
```

1.9.39 xtickformat

Set the format of major ticks on the x axis. Default is plain.

```
[PyDV]: xtickformat <plain | sci | exp | 10**>
```

1.9.40 xticklength

Set the length (in points) of x ticks on the axis. Default is apply to major ticks only.

```
[PyDV]: xticklength <number> [which: major | minor | both]
```

1.9.41 xtickwidth

Set the width (in points) of x ticks on the x axis. Default is to apply to major ticks only.

```
[PyDV]: xtickwidth <number> [which: major | minor | both]
```

1.9.42 ylabel

Set a label for the y axis

```
[PyDV]: ylabel <label-name>
```

1.9.43 ylogscale

Set log scale on or off for the y-axis. **Alternative Form: y-log-scale**, **Shortcut: yls**

```
[PyDV]: ylogscale <on | off>
```

1.9.44 ytickcolor

Set the color of the ticks on the y-axis. Default is to apply to major ticks only.

```
[PyDV]: ytickcolor <de | color> [which: major | minor | both]
```

1.9.45 ytickformat

Set the format of major ticks on the y axis. Default is plain.

```
[PyDV]: ytickformat <plain | sci | exp | 10**>
```

1.9.46 yticklength

Set the length (in points) of y ticks on the y axis. Default is to apply to major ticks only.

```
[PyDV]: yticklength <number> [which: major | minor | both]
```

1.9.47 ytickwidth

Set the width (in points) of y ticks on the y axis. Default is to apply to major ticks only.

```
[PyDV]: ytickwidth <number> [which: major | minor | both]
```

1.9.48 yticks

Set the locations of major ticks on the y axis.

```
[PyDV]: yticks de | <number> | <list of locations> | <list of locations, list of␣
↪labels>
```

1.10 Curve Control Commands

These functions control the individual curves that are currently being displayed. They range in type from controlling the appearance of the curve to deleting it. They also include the "non-mathematical" mechanisms which may generate curves.

Note: **< >** = Required user input.

[ ] = Optional user input.

[PyDV]: = Python Data Visualizer command-line prompt.

1.10.1 appendcurves - 2.4

Merge a list of curves over the union of their domains. Where the domains overlap, take the average of the curve's y-values.

```
[PyDV]: appendcurves <curve-list>
```

1.10.2 color

Set the color of curves. Color names can be "blue", "red", etc., or "#eb70aa", a 6 digit set of hexadecimal red-green-blue values (RRGGBB). The entire set of HTML-standard color names is available. Type *showcolormap* to see the available named colors.

```
[PyDV]: color <curve-list> <color>
```

1.10.3 curve

Select curves from the menu for plotting. **Shortcut: cur**

```
[PyDV]: curve [menu <regex>] <list-of-menu-numbers>
```

1.10.4 dupx - 2.4

Duplicate x-values so that y=x for each of the specified curves.

```
[PyDV]: dupx <curve-list>
```

1.10.5 linemarker

Set the marker symbol for the curves.

```
[PyDV]: linemarker <curve-list> <marker-style: + | . | circle | square | diamond> [
↪<marker-size>]
```

Note: When setting this value through the interface or the curve object directly, use ONLY matplotlib supported marker types. Matplotlib marker types are also supported here as well. See matplotlib documentation on markers for further information.

1.10.6 markerfacecolor

Set the markerface color of curves. Color names can be "blue", "red", etc, or "#eb70aa", a 6 digit set of hexadecimal red-green-blue values (RRGGBB). The entire set of HTML-standard color names is available. Try "showcolormap" to see the available named colors.

```
[PyDV]: markerfacecolor <curve-list> <color-name>
```

1.10.7 markeredgecolor

Set the markeredge color of curves. Color names can be "blue", "red", etc, or "#eb70aa", a 6 digit set of hexadecimal red-green-blue values (RRGGBB). The entire set of HTML-standard color names is available. Try "showcolormap" to see the available named colors.

```
[PyDV]: markeredgecolor <curve-list> <color-name>
```

1.10.8 showcolormap

Show the available named colors.

```
[PyDV]: showcolormap
```

1.10.9 showstyles

Show the available plot styles.

```
[PyDV]: showstyles
```

1.10.10 copy

Copy and plot the given curves

```
[PyDV]: copy <curve-list>
```

1.10.11 del

Delete the specified curves. **Shortcut: del**

```
[PyDV]: delete <curve-list>
```

1.10.12 dupx - 2.4

Duplicate the x-values such that y=x for each of the given curves

```
[PyDV]: dupx <curve-list>
```

1.10.13 hide

Hide the specified curves from view.

```
[PyDV]: hide <curve-list>
```

1.10.14 line

Generate a line with y = mx + b and an optional number of points.

```
[PyDV]: line <m> <b> <xmin> <xmax> [# pts]
```

1.10.15 linespoints

Plot curves as linespoints plots.

```
[PyDV]: linespoints <curve-list> on | off
```

1.10.16 makecurve

Generate a curve from two lists of numbers. Each list must be delimited by parentheses. **Alternative Form: makecurve**

```
[PyDV]: makecurve (<list of x-values) (<list of y-values>)
```

1.10.17 newcurve

Creates a new curve from an expression.

```
[PyDV]: newcurve <numpy expression>
```

Note: For convenience, both math and numpy modules have been imported into the namespace. Just FYI, this feature is way outside the ULTRA syntax that PyDV is mostly based on. EXAMPLE:

[PyDV]: newcurve sin(a.x*2*pi)/(h.y**2)

This creates a new curve according to the above expression. **Shortcut: nc**

Warning:

- Currently, newcurve is hard-wired to only handle single-letter labels. Curve names used in the expression cannot be the @N type we use after we run out of letters. Sorry (April 2015).
- A common error is to forget the .x or .y on the curve label name.
- All the arrays in your expression have to span the same domain! Currently (4/2015), newcurve will generate a curve from different domains (with no error message), and that curve will almost certainly not be what you intended.

1.10.18 random

Generate random y values between -1 and 1 for the specified curves.

```
[PyDV]: random <curve-list>
```

1.10.19 redo

Redo the last undo curve operation.

```
[PyDV]: redo
```

1.10.20 reid

Relabel all the curves in order. **Alternative Form: re-id**

```
[PyDV]: reid
```

1.10.21 rev

Swap x and y values for the specified curves. You may want to sort after this one.

```
[PyDV]: rev <curve-list>
```

1.10.22 scatter

Plot curves as scatter diagrams or connected lines.

```
[PyDV]: scatter <curve-list> <on | off>
```

1.10.23 show

Reveal the specified curves hidden by the hide command

```
[PyDV]: show <curve-list>
```

1.10.24 sort

Sort the specified curves so that their points are plotted in order of ascending x values.

```
[PyDV]: sort <curve-list>
```

1.10.25 subsample

Subsample the curves by the optional stride. Default value for stride is 2.

```
[PyDV]: subsample <curve-list> [stride]
```

1.10.26 undo

Undo the last operation on plotted curves.

```
[PyDV]: undo
```

1.10.27 xindex - 2.4

Create curves with y-values vs. integer index values.

```
[PyDV]: xindex <curve-list>
```

1.10.28 xminmax

Trim the specified curves. **Shortcut: xmm**

```
[PyDV]: xminmax <curve-list> <low-lim> <high-lim>
```

1.11 UI Features

The PyDV User Interface (UI) has many different windows and toolbar items that provide information and alternative ways of executing PyDV commands (see Figure 1.1). Below is a description of the different UI components that are available.

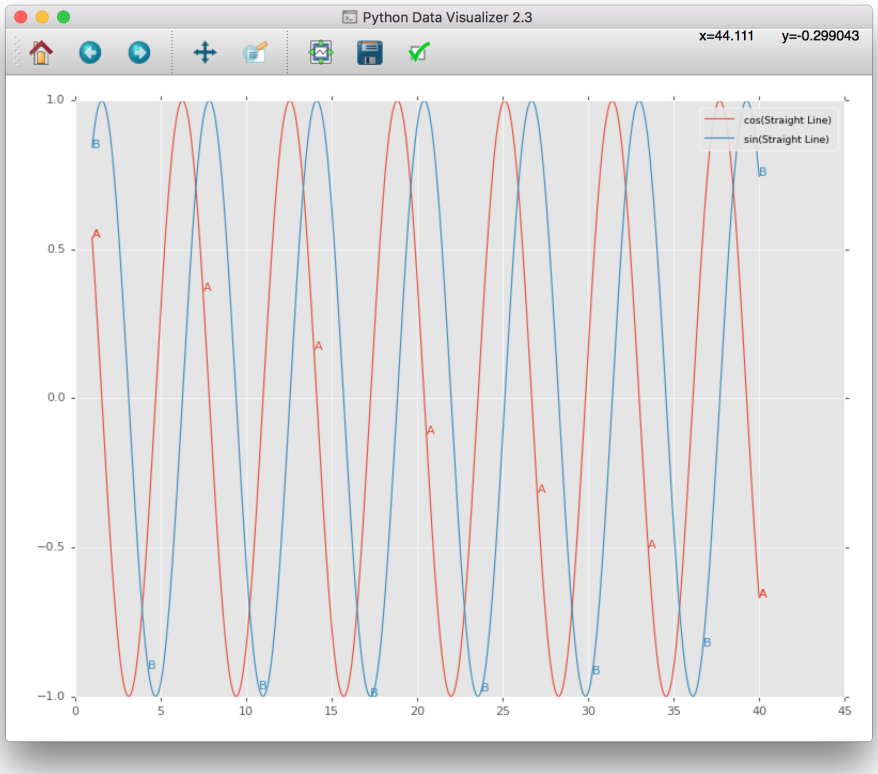

Fig. 1.1: PyDV User Interface

1.11.1 Toolbar

An annotated toolbar for the PyDV UI is shown in Figure 1.2. Button **1** allows the user to reset the plot window to the original view, button **2** allows the user to step back to the previous view, and button **3** allows the user to step through to the next view. Button **4** allows panning the axis with the left mouse button and zooming the axis with the right mouse button. Button **5** allows zooming to a rectangle. Button **6** allows configuring of the subplots, button **7** is for saving the displayed figure, and button **8** is for editing curve lines and axes parameters. The area highlighted in **9** tracks the mouse location over the plot.

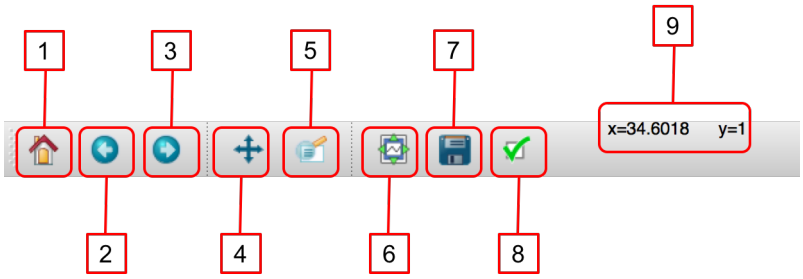

Fig. 1.2: UI Toolbar

1.11.2 Help Menu

The **Help** menu contains three items for displaying information to the user. The **Copyright ...** menu item displays the copyright information for PyDV (see Figure 1.3). The **About PyDV** window, Figure 1.4, displays information about the current release of PyDV and provides developer contact information. The **About Qt** window, Figure 1.5, displays information about the *Qt* version that PyDV is using.

1.11.3 View Menu

The **View** menu contains the **List** and **Menu** dialogs. The **List** dialog displays the information from the **list** command and also allows the user to deleted selected curves from the plot window (see Figure 1.6).

The **Menu** dialog lists the available curves read from a file. The user can delete or plot curves from this dialog (see Figure 1.7).

1.11.4 PyDV Colormap

You can display the available colors to use in PyDV for commands like **xtickcolor** by using the **showcolormap** command.

```
[PyDV]: showcolormap
```

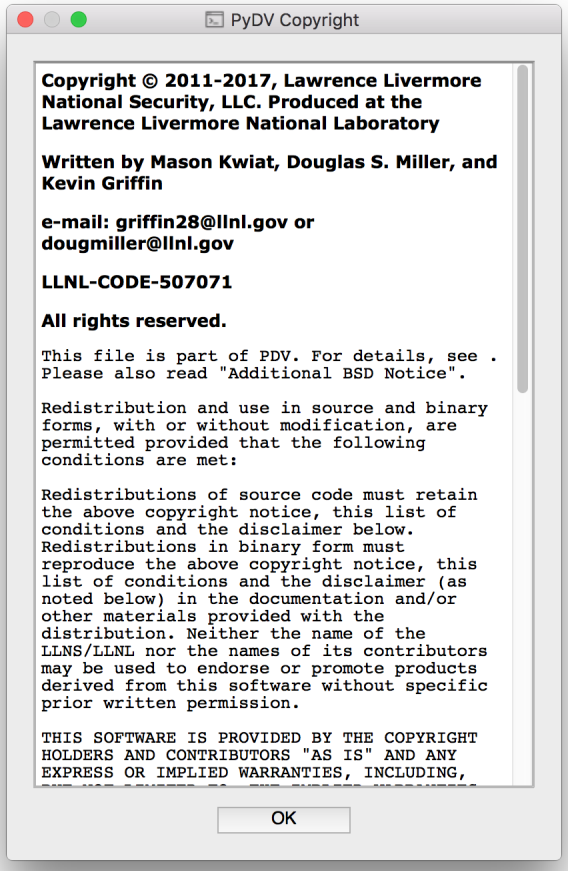

Fig. 1.3: PyDV Copyright

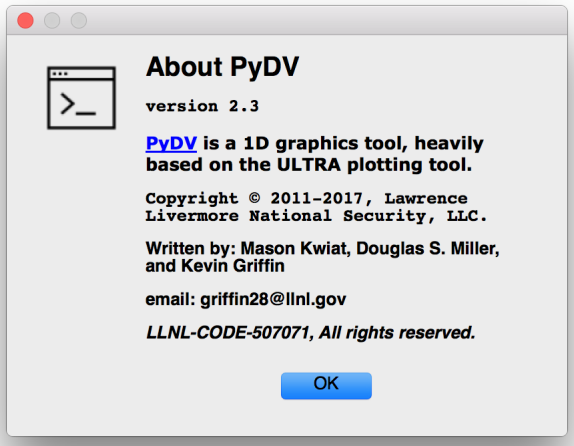

Fig. 1.4: About PyDV Window

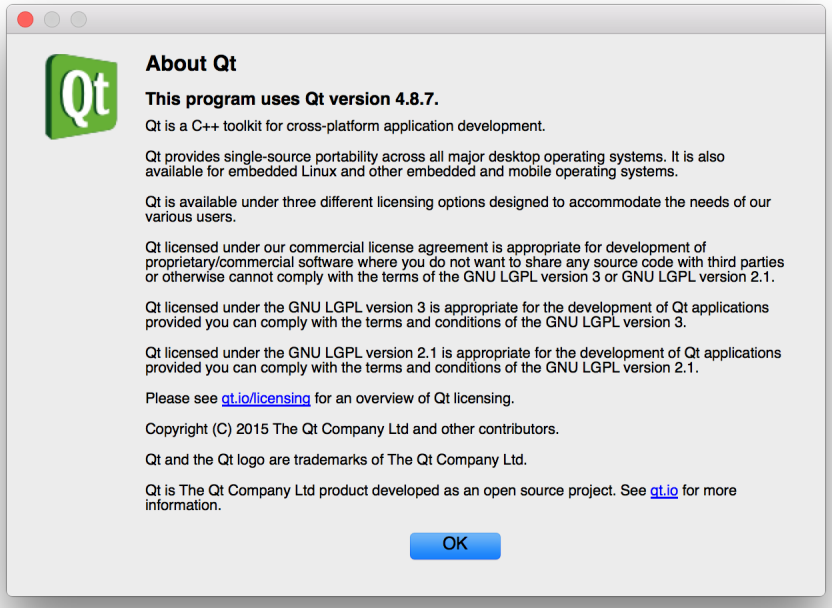

Fig. 1.5: About Qt Window

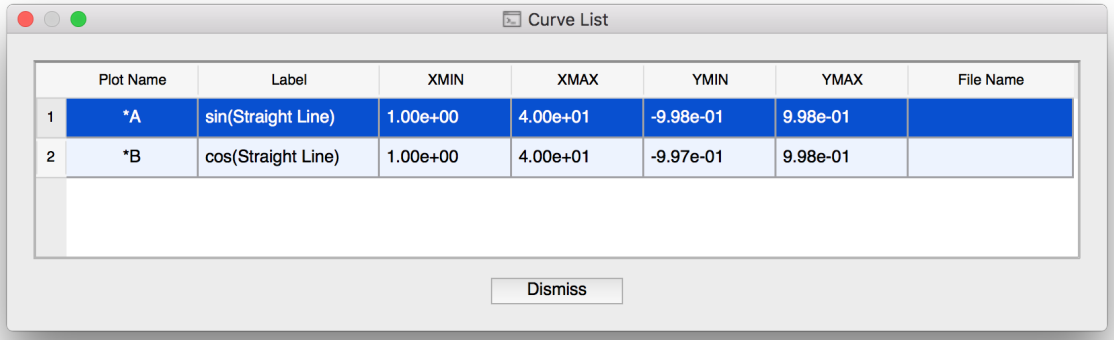

Fig. 1.6: List of plotted curves

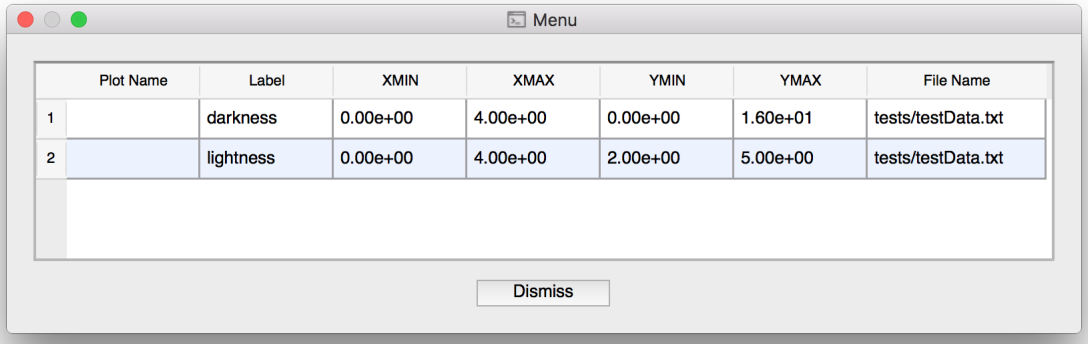

Fig. 1.7: List of available curves

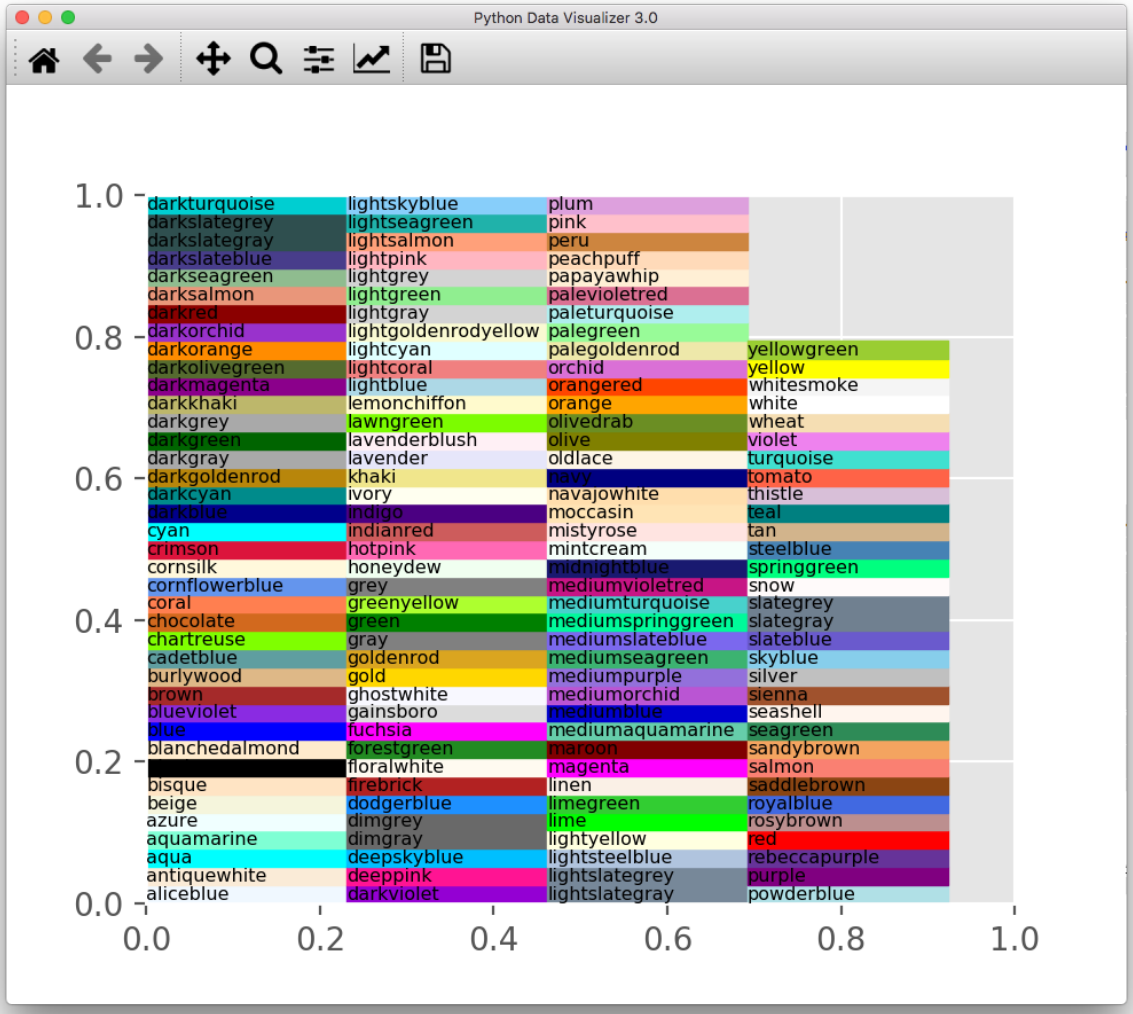

Fig. 1.8: Display of the available colors to use in PyDV

1.11.5 Plot Tickmarks

The plot tickmarks can be modified as desired. You can change the width, length, color, and number. Below is an example of changing the x-axis major tickmarks to red, the y-axis minor tickmarks to green, and the width of the y-axis minor tickmarks. The corresponding PyDV commands are also shown below.

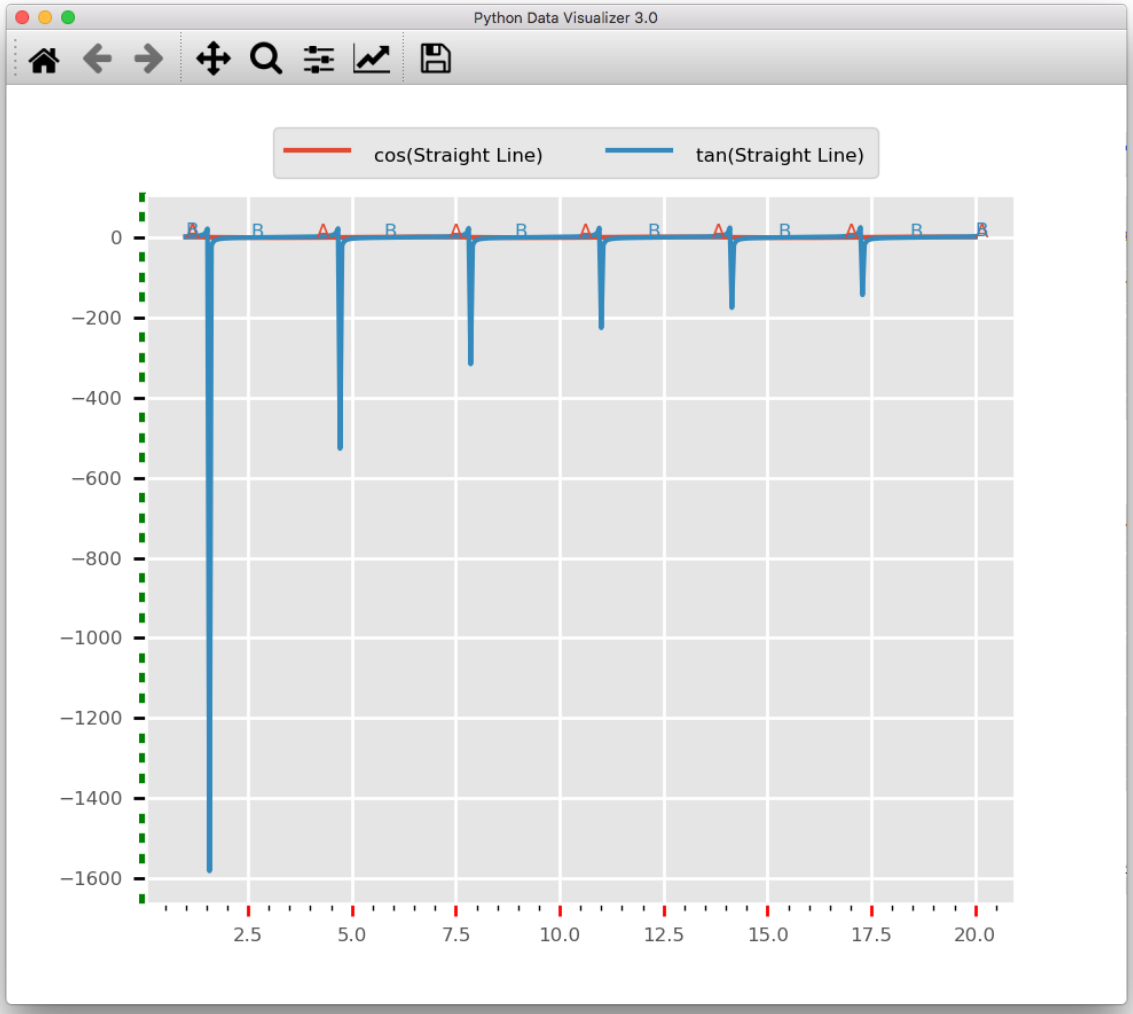

Fig. 1.9: Example of modifying the plot tickmarks

```
[PyDV]: minorticks on
[PyDV]: xtickcolor red major
[PyDV]: ytickcolor green minor
[PyDV]: ytickwidth 3 minor
```

1.11.6 Border

The border that outlines the plot can be shown or hidden. You can also change the color of the border line. Below is an example of showing and hiding the border and also setting its color to yellow.

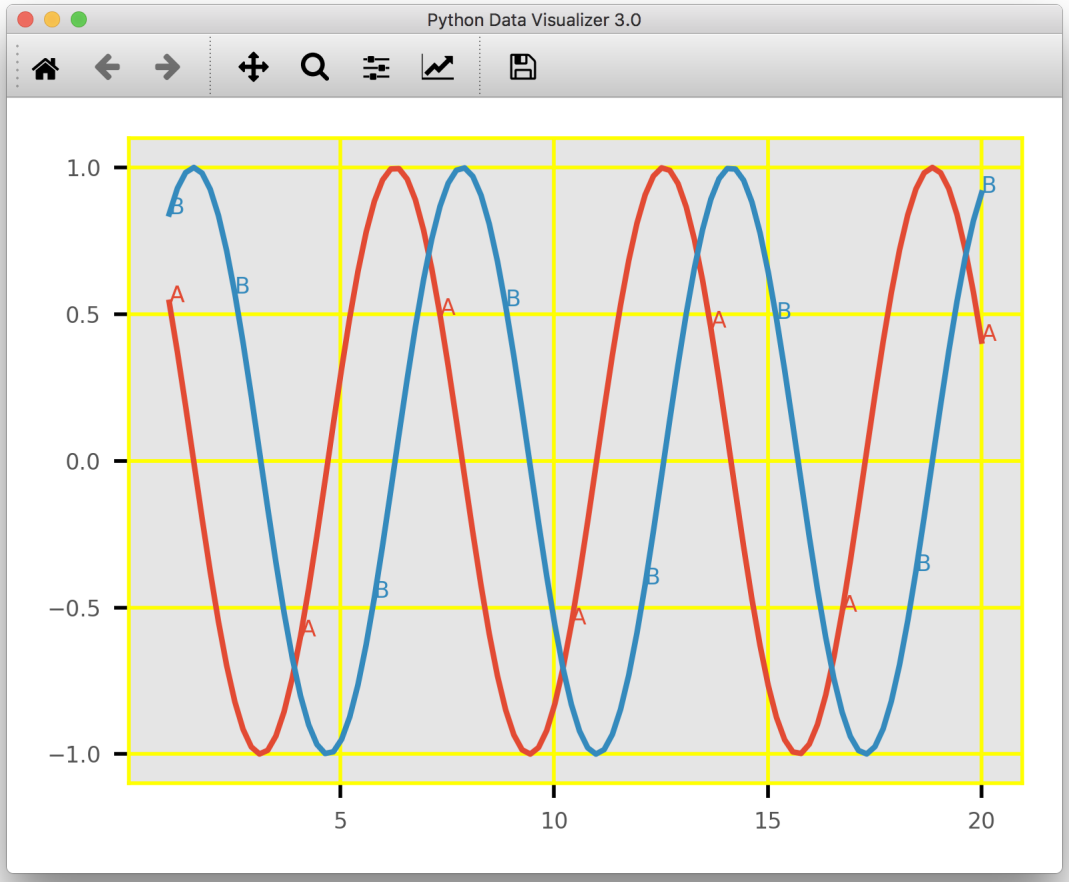

Fig. 1.10: Plot with yellow border and yellow grid

```
[PyDV]: span 1 20
[PyDV]: span 1 20
[PyDV]: cos a
[PyDV]: sin b
[PyDV]: gridcolor yellow
[PyDV]: border on yellow
```

```
[PyDV]: border off
```

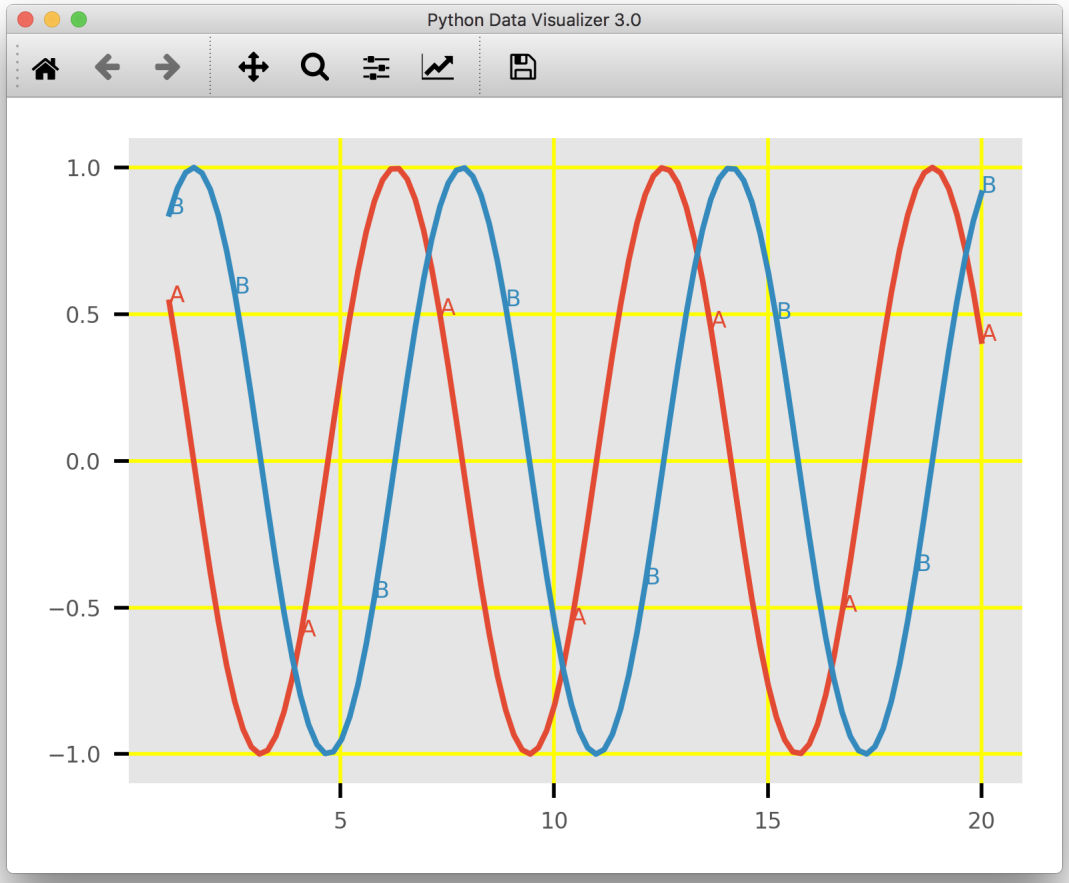

Fig. 1.11: Plot with yellow border hidden

CHAPTER

TWO

DEVELOPERS

2.1 PyDV API Specification

2.1.1 pydvpy module

A python interface for PyDV functionality.

```
>>> import pydvpy as pydvif
```

pydvpy.**abs**(*curvelist*)

Take the absolute value of the y values of the Curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.abs(curves) OR
```

```
>>> pydvif.abs(curves[0])
```

Parameters **curvelist** (*Curve or list*) – the Curve or list of curves

pydvpy.**absx**(*curvelist*)

Take the absolute value of the x values of the Curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.absx(curves) OR
```

```
>>> pydvif.absx(curves[0])
```

Parameters **curvelist** (*Curve or list*) – the Curve or list of curves

pydvpy.**acos**(*curvelist*)

Take the arccosine of y values of a Curve or list of Curves

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.acos(curves) OR
```

```
>>> pydvif.acos(curves[0])
```

Parameters **curvelist** (*Curve or list*) – The Curve or list of curves

pydvpy.**acosh**(*curvelist*)

Take the hyperbolic arccosine of y values of a Curve or list of Curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.acosh(curves) OR
```

```
>>> pydvif.acosh(curves[0])
```

Parameters **curvelist** (*Curve or list*) – The Curve or list of curves

pydvpy.**acoshx**(*curvelist*)

Take the hyperbolic arccosine of x values of a Curve or list of Curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.acoshx(curves) OR
```

```
>>> pydvif.acoshx(curves[0])
```

Parameters **curvelist** (*Curve or list*) – The Curve or list of curves

pydvpy.**acosx**(*curvelist*)

Take the arccosine of x values of a Curve or list of Curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.acosx(curves) OR
```

```
>>> pydvif.acosx(curves[0])
```

Parameters **curvelist** (*Curve or list*) – The Curve or list of curves

pydvpy.**add**(*curvelist*)

Add one or more curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> c = pydvif.add(curves)
```

Parameters **curvelist** (*list*) – The list of curves

Returns curve – the curve containing the sum of the curves in curvelist

pydvpy.**alpha**(*ac*, *ig*, *res*, *npts=- 1*)

pydvpy.**appendcurves**(*curvelist*)

Merge two or more curves over the union of their domains. Where domains overlap, take the average of the curve's y-values.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> newcurve = pydvif.appendcurve(curves)
```

Parameters **curvelist** (*list*) – the specified curves

Returns Curve – the merging of the two curves c1 and c2

pydvpy.**asin**(*curvelist*)

Take the arcsine of y values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.asin(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**asinh**(*curvelist*)

Take the hyperbolic arcsine of y values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.asinh(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**asinhx**(*curvelist*)

Take the hyperbolic arcsine of x values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.asinhx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**asinx**(*curvelist*)

Take the arcsine of x values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.asinx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**atan**(*curvelist*)

Take the arctangent of y values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.atan(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**atan2**(*c1*, *c2*, *t=None*)

Perform the atan2 method for a pair of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.atan2(curves[0], curves[1])   OR
```

```
>>> pydvif.atan2(curves[0], curves[1], tuple(['A', 'B']))
```

Parameters

- **c1** (*curve*) – the first curve
- **c2** (*curve*) – the second curve
- **t** (*tuple*) – A tuple containing exactly two values to insert into the name string for the new curve

Returns curve – a new curve with the results from this operation

pydvpy.**atanh**(*curvelist*)

Take the hyperbolic arctangent of y values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.atanh(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**atanhx**(*curvelist*)

Take the hyperbolic arctangent of x values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.atanhx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**atanx**(*curvelist*)

Take the arctangent of x values of a single curve or curves in a list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.atanx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**average_curve**(*curvelist*)
Average the specified curves over the intersection of their domains.

Parameters **curvelist** – the specified curves

Returns Curve – a new curve with the average values over the intersection of the domains of the specified curves.

pydvpy.**convolve**(*c1*, *c2*, *npts=100*)
Compute and return the convolution of two real curves: `- - (g*h)(x) = Int(-inf, inf, dt, g(t)*h(x-t))` - The Fourier Transform is used to perform the convolution.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> newcurve = pydvif.convolve(curves[0], curves[1])
```

Parameters

- **c1** (*Curve*) – (N,) The first curve
- **c2** (*Curve*) – (M,) The second curve
- **npts** (*int*) – the number of points used to create a uniform temporal spacing

Returns Curve – the convolution of the two curves c1 and c2

pydvpy.**convolve_int**(*c1*, *c2*, *norm=True*, *npts=100*)
Computes the convolution of the two curves (c1, c2). The integrals are computed directly which avoid padding and aliasing problems associated with FFT methods (it is however slower).

Parameters

- **c1** (*Curve*) – (N,) The first curve
- **c2** (*Curve*) – (M,) The second curve
- **norm** (*bool*) – if true then the result is normalized to unit area.
- **npts** (*int*) – the number of points

Returns nc: Curve – the convolution of the two curves c1 and c2

pydvpy.**convolveb**(*c1*, *c2*, *npts=100*)
Computes the convolution of the two given curves: `- - (g*h)(x) = Int(-inf, inf, dt*g(t)*h(x-t)) / Int(-inf, inf, dt*h(t))` - This computes the integrals directly which avoid padding and aliasing problems associated with FFT methods (it is however slower).

Parameters

- **c1** (*Curve*) – (N,) The first curve
- **c2** (*Curve*) – (M,) The second curve
- **npts** (*int*) – the number of points

Returns Curve – the convolution of the two curves c1 and c2 using integration and normalizing c2

pydvpy.**convolvec**(*c1*, *c2*, *npts=100*)
Computes the convolution of the two given curves: `- - (g*h)(x) = Int(-inf, inf, dt*g(t)*h(x-t)) / Int(-inf, inf, dt*h(t))` - This computes the integrals directly which avoid padding and aliasing problems associated with FFT methods (it is however slower).

Parameters

- **c1** (Curve) – (N,) The first curve
- **c2** (Curve) – (M,) The second curve
- **npts** (*int*) – the number of points

Returns Curve – the convolution of the two curves c1 and c2 using integration and no normalization

pydvpy.**correlate**(*c1*, *c2*, *mode='valid'*)

Computes the cross-correlation of two 1D sequences (c1.y and c2.y) as defined by numpy.correlate.

Parameters

- **c1** (Curve) – The first curve with 1D input sequence c1.y
- **c2** (Curve) – The second curve with 1D input sequence c2.y
- **mode** (*'full'(default), 'same' or 'valid'*) –

  full: By default, mode is 'full'. This returns the convolution at each point of overlap, with an output shape of (N+M-1,). At the end-points of the convolution, the signals do not overlap completely, and boundary effects may be seen.

  same: Mode 'same' returns output of length `max(M, N)`. Boundary effects are still visible.

  valid: Mode 'valid' returns output of length `max(M, N) - min(M, N) + 1`. The convolution product is only given for points where the signals overlap completely. Values outside the signal boundary have no effect.

Returns Curve – the cross-correlation of c1.y and c2.y

pydvpy.**cos**(*curvelist*)

Take the cosine of y values of a Curve or list of Curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.cos(curves) OR
```

```
>>> pydvif.cos(curves[0])
```

Parameters **curvelist** (Curve *or list*) – The Curve or list of Curves

pydvpy.**cosh**(*curvelist*)

Take the hyperbolic cosine of y values of a Curve or list of Curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.cosh(curves) OR
```

```
>>> pydvif.cosh(curves[0])
```

Parameters **curvelist** (Curve *or list*) – The Curve or list of curves

pydvpy.**coshx**(*curvelist*)

Take the hyperbolic cosine of x values of a Curve or list of Curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.coshx(curves) OR
```

```
>>> pydvif.coshx(curves[0])
```

Parameters **curvelist** (*Curve or list*) – The Curve or list of curves

pydvpy.**cosx**(*curvelist*)

Take the cosine of x values of a Curve or list of Curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.cosx(curves) OR
```

```
>>> pydvif.cosx(curves[0])
```

Parameters **curvelist** (*Curve or list*) – The Curve or list of Curves

pydvpy.**create_plot**(*curvelist*, ***kwargs*)

Create a plot from of the curves in curvelist. The available keyword arguments are: * Filename: fname='myFile' * Save Format: ftype='pdf' * Plot Title: title='My Title' * X-Axis Label: xlabel='X' * Y-Axis Label: ylabel='Y' * Show/Hide Plot Legend: legend=True * Plot Style: stylename='ggplot' * Show X-Axis in log scale: xls=True * Show Y-Axis in log scale: yls=True * Set the width of the figure in inches: fwidth=1.2 * Set the height of the figure in inches: fheight=2.1

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> plot1 = pydvif.create_plot(curves, fname='myPlot1')
```

```
>>> plot2 = pydvif.create_plot(curves, fname='myPlot2', ftype='pdf', fwidth=10.1,␣
↪fheight=11.3, title='My Plot', xlabel='X', ylabel='Y', legend=True, stylename=
↪'ggplot')
```

Parameters

- **curvelist** (*list*) – The curve or list of curves to plot
- **kwargs** (*dict*) – The keyword arguments to modify the plot.

Returns matplotlib.pyplot – the plot of the curves

pydvpy.**derivative**(*c*, *eo=1*)

Take the derivative of the curve.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> newCurve = pydvif.derivative(curves[0])
```

Parameters

- **c** (*Curve*) – The curve
- **eo** (*int, optional*) – edge_order, gradient is calculated using N-th order accurate differences at the boundaries. Default: 1.

Returns A new curve representing the derivate of c

pydvpy.**diffMeasure**(*c1*, *c2*, *tol=1e-08*)

Compare two curves. For the given curves a fractional difference measure and its average are computed.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> c1, c2  = pydvif.diffMeasure(curves[0], curves[1])
```

```
>>> curves.append(c1)
```

```
>>> curves.append(c2)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters

- **c1** (Curve) – The first curve
- **c2** (Curve) – The second curve
- **tol** (*float*) – The tolerance

Returns tuple – Two curves representing the fractional difference measure and its average

pydvpy.**disp**(*c*, *domain=True*)

Create a string formatted list of the curve's x-values if domain is True, otherwise y-values.

```
>>> c = pydvif.span(1, 10)
```

```
>>> yvalues = pydvif.disp(c, False)
```

Parameters

- **c** – The given curve
- **domain** (*bool, optional*) – if True, display the x-values of the curve. Otherwise, display the y-values of the curve

Returns list – The list of x- or y-values as strings

pydvpy.**divide**(*curvelist*)

Take quotient of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> c = pydvif.divide(curves)
```

Parameters **curvelist** (*list*) – The list of curves

Returns curve – the curve containing the quotient of the curves

pydvpy.**divx**(*curvelist*, *value*)

Divide x values of the curve(s) by a constant value.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.divx(curves, 4)
```

Parameters

- **curvelist** (*Curve or list*) – The curve or curvelist
- **value** (*float*) – The divisor

pydvpy.**divy**(*curvelist*, *value*)

Divide y values of the curve(s) by a constant value.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.divy(curves, 4)
```

Parameters

- **curvelist** (*Curve or list*) – The curve or curvelist
- **value** (*float*) – The divisor

pydvpy.**dupx**(*curvelist*)

Duplicate the x-values such that y = x for each of the given curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.dupx(curves)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters **curvelist** (*Curve or list*) – The curve or list of curves

pydvpy.**dx**(*curvelist*, *value*)

Shift x values of a curve or list of curves by a constant value.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.dx(curves, 4) OR
```

```
>>> pydvif.dx(curves[0], 4)
```

Parameters

- **curvelist** (*Curve or list*) – A curve or curvelist
- **value** (*float*) – The amount to shift the x values by

pydvpy.**dy**(*curvelist*, *value*)

Shift y values of a curve or list of curves by a constant value.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.dy(curves, 4) OR
```

```
>>> pydvif.dy(curves[0], 4)
```

Parameters

- **curvelist** (*Curve or list*) – A curve or curvelist
- **value** (*float*) – The amount to shift the y values by

pydvpy.**errorbar**(*scur*, *cury1*, *cury2*, *curx1=None*, *curx2=None*, *mod=1*)

Plot error bars on the given curve.

```
>>> curves = list()
```

```
>>> curves.append(pydvif.span(1,10))
```

```
>>> curves.append(pydvif.span(1,10))
```

```
>>> curves.append(pydvif.span(1,10))
```

```
>>> pydvif.dy(curves[0], 0.25)
```

```
>>> pydvif.dy(curves[2], -0.25)
```

```
>>> pydvif.errorbar(curves[1], curves[0], curves[2])
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters

- **scur** (*Curve*) – The given curve
- **cury1** (*Curve*) – y-error-curve
- **cury2** (*Curve*) – y+error-curve
- **curx1** (*Curve*) – x-error-curve
- **curx2** (*Curve*) – x+error-curve
- **mod** (*int*) – point-skip

pydvpy.**errorrange**(*scur*, *cury1*, *cury2*)

Plot shaded error region on given curve.

```
>>> curves = list()
```

```
>>> curves.append(pydvif.span(1,10))
```

```
>>> curves.append(pydvif.span(1,10))
```

```
>>> curves.append(pydvif.span(1,10))
```

```
>>> pydvif.dy(curves[0], 0.25)
```

```
>>> pydvif.dy(curves[2], -0.25)
```

```
>>> pydvif.errorrange(curves[1], curves[0], curves[2])
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters

- **scur** (Curve) – The given curve
- **cury1** (Curve) – y-error-curve
- **cury2** (Curve) – y+error-curve

pydvpy.**exp**(*curvelist*)

Exponentiate y values of the Curve or list of curves (e**y).

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.exp(curves) OR
```

```
>>> pydvif.exp(curves[0])
```

Parameters **curvelist** (Curve *or list*) – the Curve or list of curves

pydvpy.**expx**(*curvelist*)

Exponentiate x values of the Curve or list of curves (e**x).

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.expx(curves) OR
```

```
>>> pydvif.expx(curves[0])
```

Parameters **curvelist** (Curve *or list*) – the Curve or list of curves

pydvpy.**fft**(*c*, *n=None*, *axis=- 1*, *norm=None*)

Compute the one-dimensional discrete Fourier Transform for the x- or y-values of c.

This function computes the one-dimensional *n*-point discrete Fourier Transform (DFT) with the efficient Fast Fourier Transform (FFT) algorithm [CT].

Raises IndexError: if *axes* is larger than the last axis of *a*.

Notes: FFT (Fast Fourier Transform) refers to a way the discrete Fourier Transform (DFT) can be calculated efficiently, by using symmetries in the calculated terms. The symmetry is highest when *n* is a power of 2, and the transform is therefore most efficient for these sizes.

The DFT is defined, with the conventions used in this implementation, in the documentation for the *numpy.fft* module.

Citation: Cooley, James W., and John W. Tukey, 1965, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.* 19: 297-301.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> realcurve, imagcurve = pydvif.fft(curves[0])
```

Parameters

- **c** (Curve) – Curve with x- or y-values as input array, can be complex.
- **n** (*int, optional*) – Length of the transformed axis of the output. If *n* is smaller than the length of the input, the input is cropped. If it is larger, the input is padded with zeros. If *n* is not given, the length of the input along the axis specified by *axis* is used.
- **axis** (*int, optional*) – Axis over which to compute the FFT. If not given, the last axis is used.
- **norm** (*None, "ortho", optional*) – Normalization mode (see *numpy.fft*). Default is None.

Returns Curve tuple – Two curves with the real and imaginary parts.

pydvpy.**filtercurves**(*curvelist*, *pattern*)

Filters the list of curves based on the regular expression pattern.

```
>>> curves = pydvif.filtercurves(curves, "*_name")
```

Parameters

- **curvelist** (Curve) – the list of curves
- **pattern** (*str*) – the regular expression pattern

Returns list – The list of filtered curves from curvelist based on the regular expression pattern

pydvpy.**fit**(*c*, *n=1*, *logx=False*, *logy=False*)

Make a new curve that is a polynomial fit to curve c.

```
>>> curves = list()
```

```
>>> curves.append(pydvif.span(1,10))
```

```
>>> pydvif.sin(curves)
```

```
>>> curves.append(pydvif.fit(curves[0], 2))
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters

- **c** (Curve) – The curve to fit
- **n** (*int*) – Degree of the fitting polynomial
- **logx** (*bool*) – Take the log(x-values) before fitting if True
- **logy** (*bool*) – Take the log(y-values) before fitting if True

Returns Curve – The fitting polynomial

pydvpy.**gaussian**(*amp*, *wid*, *center*, *num=100*, *nsd=3*)
Generate a gaussian function.

```
>>> curve = pydvif.gaussian(5, 10, 0)
```

```
>>> pydvif.create_plot(curve, legend=True, stylename='ggplot')
```

Parameters

- **amp** (*float*) – amplitude
- **wid** (*float*) – width
- **center** (*float*) – center
- **num** (*int*) – optional, number of points
- **nsd** (*float*) – optional, number of half-widths

Returns Curve – representing the gaussian function

pydvpy.**get_styles**()
Get the list of available plot styles.

Returns list – the list of available style names or an empty list if no styles exist.

pydvpy.**getdomain**(*curvelist*)
Get domain of the curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> domains = pydvif.getdomain(curves)
```

```
>>> plotname, minx, maxx = domains[0]
```

Parameters **curvelist** (*Curve or list*) – The given curve or list of curves

Returns list – A list of tuples where each tuple contains the curve name, minimum x, and maximum x

pydvpy.**getnumpoints**(*curve*)
Return the given curve's number of points.

Parameters **curve** – The given curve

Returns int – the number of points in curve

pydvpy.**getrange**(*curvelist*)
Get the range of the curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> ranges = pydvif.getrange(curves)
```

```
>>> plotname, miny, maxy = ranges[0]
```

Parameters **curvelist** (*Curve or list*) – The given curve or list of curves

Returns list – A list of tuples where each tuple contains the curve name, minimum y, and maximum y

pydvpy.**getx**(*c*, *value*)

Get the x values of the curve for a given y.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> vals = pydvif.getx(curves[0], 4)
```

```
>>> x, y = vals[0]
```

Parameters

- **c** (Curve) – The curve
- **value** (*float*) – y value

Returns list – A list of tuples where each tuple contains the x value, and the given y

pydvpy.**gety**(*c*, *value*)

Get the y values of the curve for a given x.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> vals = pydvif.gety(curves[0], 2)
```

```
>>> x, y = vals[0]
```

Parameters

- **c** (Curve) – The curve
- **value** (*float*) – x value

Returns list – A list of tuples where each tuple contains the y value, and the given x

pydvpy.**getymax**(*c*, *xmin=None*, *xmax=None*)

Get the maximum y-value for the curve within the specified domain.

Parameters

- **c** – the curve
- **xmin** (*float, optional*) – the minimum x-value for the sub-domain
- **xmax** (*float, optional*) – the maximum x-value for the sub-domain

Returns str – curve name ymax – the maximum y-value for the specified domain

pydvpy.**getymin**(*c*, *xmin=None*, *xmax=None*)

Get the minimum y-value for the curve within the specified domain.

Parameters

- **c** – the curve
- **xmin** (*float, optional*) – the minimum x-value for the sub-domain
- **xmax** (*float, optional*) – the maximum x-value for the sub-domain

Returns str – curve name ymin – the minimum y-value for the specified domain

pydvpy.**integrate**(*curvelist*, *low=None*, *high=None*)

Take the integral of the curve or curves in curvelist.

Parameters

- **curvelist** (*curve or list*) – A curve or list of curves
- **low** (*float*) – The lower limit
- **high** (*float*) – The maximum limit

Returns list – the list of integrated curves

pydvpy.**j0**(*curvelist*)

Take the Bessel function of the first kind of the zeroth order for the y values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**j0x**(*curvelist*)

Take the Bessel function of the first kind of the zeroth order for the x values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**j1**(*curvelist*)

Take the Bessel function of the first kind of the first order for the y values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**j1x**(*curvelist*)

Take the Bessel function of the first kind of the first order for the x values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**jn**(*curvelist*, *n*)

Take the Bessel function of the first kind of the nth order for the y values of curves in curvelist.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **n** (*float*) – The order

pydvpy.**jnx**(*curvelist*, *n*)

Take the Bessel function of the first kind of the nth order for the x values of curves in curvelist.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **n** (*float*) – The order

pydvpy.**l1**(*c1*, *c2*, *xmin=None*, *xmax=None*)

Make a new curve that is the L1 norm of curve c1 and curve c2. The L1-norm is the integral(|c1 - c2|) over the interval [xmin, xmax].

```
>>> c = pydvif.l1(curve1, curve2)
```

```
>>> c2 = pydvif.l1(curve1, curve2, 1.1, 10.9)
```

Parameters

- **c1** (Curve) – The first curve

- **c2** (Curve) – The second curve
- **xmin** (*float*) – the minimum x value to perform the L1 norm
- **xmax** (*float*) – the maximum x value to perform the L1 norm

Returns Curve A new curve that is the L1 norm of c1 and c2

pydvpy.**l2**(*c1*, *c2*, *xmin=None*, *xmax=None*)

Make a new curve that is the L2 norm of curve c1 and curve c2. The L2-norm is (integral((c1 - c2)**2)**(1/2) over the interval [xmin, xmax].

```
>>> c = pydvif.l2(curve1, curve2)
```

```
>>> c2 = pydvif.l2(curve1, curve2, 3.1, 30.9)
```

Parameters

- **c1** (Curve) – The first curve
- **c2** (Curve) – The second curve
- **xmin** (*float*) – the minimum x value to perform the L2 norm
- **xmax** (*float*) – the maximum x value to perform the L2 norm

Returns Curve A new curve that is the L2 norm of c1 and c2

pydvpy.**line**(*m*, *b*, *xmin*, *xmax*, *numpts=100*)

Generate a line with y = mx + b and an optional number of points.

```
>>> curves = list()
```

```
>>> curves.append(pydvif.line(2, 5, 0, 10))
```

```
>>> pydvif.create_plot(curves, legend=True, stylename='ggplot')
```

Parameters

- **m** (*float*) – The slope
- **b** (*float*) – The y-intercept
- **xmin** (*float*) – The minimum x value
- **xmax** (*float*) – The maximum x value
- **numpts** (*int*) – The number of points to use for the new line

Returns Curve – The curve representing the newly created line

pydvpy.**log**(*curvelist*, *keep=True*)

Take the natural logarithm of y values of the Curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.log(curves) OR
```

```
>>> pydvif.log(curves[0])
```

Parameters

- **curvelist** (*Curve or list*) – the Curve or list of curves
- **keep** (*optional, boolean*) – flag to determine whether or not to discard zero or negative y-values before taking the log. keep is True by default.

pydvpy.**log10**(*curvelist*, *keep=True*)

Take the base 10 logarithm of y values of a Curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.log10(curves) OR
```

```
>>> pydvif.log10(curves[0])
```

Parameters

- **curvelist** (*Curve or list*) – the Curve or list of curves
- **keep** (*optional, boolean*) – flag to determine whether or not to discard zero or negative y-values before taking the base 10 logarithm. keep is True by default.

pydvpy.**log10x**(*curvelist*, *keep=True*)

Take the base 10 logarithm of x values of a Curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.log10x(curves) OR
```

```
>>> pydvif.log10x(curves[0])
```

Parameters

- **curvelist** (*Curve or list*) – the Curve or list of curves
- **keep** (*optional, boolean*) – flag to determine whether or not to discard zero or negative y-values before taking the base 10 logarithm. keep is True by default.

pydvpy.**logx**(*curvelist*, *keep=True*)

Take the natural logarithm of x values of the Curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.logx(curves) OR
```

```
>>> pydvif.logx(curves[0])
```

Parameters

- **curvelist** (*Curve or list*) – the Curve or list of curves

- **keep** (*optional, boolean*) – flag to determine whether or not to discard zero or negative x-values before taking the log. keep is True by default.

pydvpy.**makecurve**(*x*, *y*, *name='Curve'*, *fname=''*, *xlabel=''*, *ylabel=''*, *title=''*)

Generate a curve from two lists of numbers.

```
>>> c1 = pydvif.makecurve([1, 2, 3, 4], [5, 10, 15, 20])
```

```
>>> c2 = pydvif.makecurve([1, 2, 3, 4], [7, 8, 9, 10], 'Line')
```

Parameters

- **x** (*list*) – list of x values
- **y** (*list*) – list of y values
- **name** (*str*) – the name of the new curve
- **fname** (*str*) – the name of the file containing this curves data.

Returns curve – the curve generated from the x and y list of values.

pydvpy.**makeextensive**(*curvelist*)

Set the y-values such that `y[i] *= (x[i+1] - x[i])`

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.makeextensive(curves)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters **curvelist** (*Curve or list*) – The curve or list of curves

pydvpy.**makeintensive**(*curvelist*)

Set the y-values such that y[i] /= (x[i+1] - x[i]).

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.makeintensive(curves)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters **curvelist** (*Curve or list*) – The curve or list of curves

pydvpy.**max_curve**(*curvelist*)

Construct a curve from the maximum y values of the intersection of the curves domain.

Parameters **curvelist** – the specified curves

Returns Curve – a new curve with the maximum y-values over the intersection of the domains of the specified curves.

pydvpy.**min_curve**(*curvelist*)

Construct a curve from the minimum y values of the intersection of the curves domain.

Parameters **curvelist** – the specified curves

Returns Curve – a new curve with the minimum y-values over the intersection of the domains of the specified curves.

pydvpy.**multiply**(*curvelist*)

Take product of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> c = pydvif.multiply(curves)
```

Parameters **curvelist** (*list*) – The list of curves

Returns Curve – the curve containing the product of the curves

pydvpy.**mx**(*curvelist*, *value*)

Scale x values of a curve or list of curves by a constant value.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.mx(curves, 4) OR
```

```
>>> pydvif.mx(curves[0], 4)
```

Parameters

- **curvelist** (*Curve or list*) – A curve or curvelist
- **value** (*float*) – The amount to scale the x values by

pydvpy.**my**(*curvelist*, *value*)

Scale y values of a curve or list of curves by a constant value.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.my(curves, 4) OR
```

```
>>> pydvif.my(curves[0], 4)
```

Parameters

- **curvelist** (*Curve or list*) – A curve or curvelist
- **value** (*float*) – The amount to scale the y values by

pydvpy.**norm**(*c1*, *c2*, *p*, *xmin=None*, *xmax=None*)

Make a new curve that is the p-norm of curve c1 and curve c2.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> c = pydvif.norm(curves[0], curves[1], 'inf')
```

```
>>> curves.append(c)
```

Parameters

- **c1** (Curve) – The first curve
- **c2** (Curve) – The second curve
- **p** (*str*) – the order (e.g., 'inf', '3', '5')
- **xmin** (*float*) – the minimum x value to perform the p-norm
- **xmax** (*float*) – the maximum x value to perform the p-norm

Returns Curve A new curve that is the p-norm of c1 and c2

pydvpy.**powa**(*curvelist*, *a*)

Raise a fixed value, a, to the power of the y values of the Curve or list of curves. y = a^y

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.powa(curves, 2) OR
```

```
>>> pydvif.powa(curves[0], 2)
```

Parameters

- **curvelist** (Curve *or list*) – the Curve or list of curves
- **a** (*float*) – the fixed value

pydvpy.**powax**(*curvelist*, *a*)

Raise a fixed value, a, to the power of the x values of the Curve or curves. x = a^x

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.powax(curves, 4.2) OR
```

```
>>> pydvif.powax(curves[0], 4.2)
```

Parameters

- **curvelist** (Curve *or list*) – the Curve or list of curves
- **a** (*float*) – the fixed value

pydvpy.**powr**(*curvelist*, *a*)

Raise a the y values of a curve or list of curves to a fixed power, y = y^a.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.powr(curves, 4.2) OR
```

```
>>> pydvif.powr(curves[0], 4.2)
```

Parameters

- **curvelist** (*curve or list*) – the curve or list of curves
- **a** (*float*) – the fixed value

pydvpy.**powrx**(*curvelist*, *a*)

Raise a the x values of a curve or list of curves to a fixed power, x = x^a.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.powrx(curves, 4.2) OR
```

```
>>> pydvif.powrx(curves[0], 4.2)
```

Parameters

- **curvelist** (*curve or list*) – the curve or list of curves
- **a** (*float*) – the fixed value

pydvpy.**random**(*curve*)

Generate random y values between -1 and 1 for the specified curves.

```
>>> c = pydvif.span(1, 10)
```

```
>>> pydvif.random(c)
```

Parameters **curve** (Curve) – The curve to sort

pydvpy.**read**(*fname*, *gnu=False*, *xcol=0*, *verbose=False*, *pattern=None*, *matches=None*)

Read the file and add parsed curves to a curvelist

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> curves = pydvif.read('testData.txt', False, 0, False, '*_name', 20)
```

Parameters

- **fname** (*str*) – ULTRA filename
- **gnu** (*bool*) – optional, flag to determine if the file is a column oriented (.gnu) file.
- **xcol** (*int*) – optional, x-column number for column oriented (.gnu) files
- **verbose** (*bool*) – optional, prints the error stacktrace when True
- **pattern** (*str*) – optional, the regular expression pattern
- **matches** (*int*) – optional, maximum number of times to match pattern, if specified

Returns list – the list of curves from the file matching pattern, if specified

pydvpy.**readcsv**(*fname*, *xcol=0*, *verbose=False*)

Load a csv (comma separated values) data file, add parsed curves to a curvelist. '#' is the comment character. First uncommented line must be the column labels. We assume the first column is the x-data, every other column is y-data. We also assume all columns are the same length.

```
>>> curves = readcsv('testData.csv')
```

Parameters

- **fname** (*str*) – csv filename

- **xcol** (*int*) – x-column number for column oriented (.gnu) files
- **verbose** (*bool*) – prints the error stacktrace when True

Returns list – the list of curves from the csv file

pydvpy.**readsina**(*fname*, *verbose=False*)

Load a Sina JSON data file, add parsed curves to a curvelist.

We assume JSON conforming to the Sina schema, with each curve defined in a curve_set. We assume there is only one record, and if there are more then we only read the first one. We also assume only one independent variable per curve_set; if there are more than one, then PyDV may exhibit undefined behavior.

```
>>> curves = readsina('testData.json')
```

Parameters

- **fname** (*str*) – Sina JSON filename
- **verbose** (*bool*) – prints the error stacktrace when True

Returns list: the list of curves from the sina file

pydvpy.**recip**(*curvelist*)

Take the reciprocal of the y values of the curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.recip(curves[1])
```

```
>>> pydvif.create_plot(curves, legend=True, stylename='ggplot')
```

Parameters **curvelist** (*Curve or list*) – The curve or list of curves

pydvpy.**recipx**(*curvelist*)

Take the reciprocal of the x values of the curve or list of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.dx(curves, 2)
```

```
>>> pydvif.recipx(curves)
```

```
>>> pydvif.create_plot(curves, legend=True, stylename='ggplot')
```

Parameters **curvelist** (*Curve or list*) – The curve or list of curves

Returns

pydvpy.**rev**(*curve*)

Swap x and y values for the specified curves. You may want to sort after this one.

```
>>> c = pydvif.span(1, 10)
```

```
>>> pydvif.rev(c)
```

Parameters **curve** (Curve) – The curve to sort

pydvpy.**save**(*fname*, *curvelist*, *verbose=False*)

Saves the given Curve or list of Curves to a file named fname.

```
>>> curves = list()
```

```
>>> curves.append(pydvif.makecurve([1, 2, 3, 4], [5, 10, 15, 20]))
```

```
>>> pydvif.save('myfile.txt', curves) OR
```

```
>>> pydvif.save('myfile.txt', curves[0])
```

Parameters

- **fname** (*str*) – ULTRA filename
- **curvelist** (Curve *or list*) – The curve or list of curves to save
- **verbose** (*bool*) – prints the error stacktrace when True

pydvpy.**savecsv**(*fname*, *curvelist*, *verbose=False*)

Saves the Curve or list of Curves to file in comma separated values (csv) format. Assumes all curves have the same x basis.

```
>>> curves = list()
```

```
>>> curves.append(pydvif.makecurve([1, 2, 3, 4], [5, 10, 15, 20]))
```

```
>>> pydvif.savecsv('myfile.csv', curves)
```

Parameters

- **fname** (*str*) – ULTRA filename
- **curvelist** (*list*) – The Curve or list of Curves to save
- **verbose** (*bool*) – prints the error stacktrace when True

pydvpy.**sin**(*curvelist*)

Take the sine of y values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.sin(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**sinh**(*curvelist*)

Take the hyperbolic sine of y values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.sinh(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**sinhx**(*curvelist*)

Take the hyperbolic sine of x values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.sinhx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**sinx**(*curvelist*)

Take the sine of x values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.sinx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**smooth**(*curvelist*, *factor=1*)

Smooth the curve to the given degree.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.smooth(curves, 4)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters

- **curvelist** (Curve *or list*) – The curve or list of curves
- **factor** (*int*) – The smooth factor

pydvpy.**sort**(*curve*)

Sort the specified curve so that their points are plotted in order of ascending x values.

```
>>> c = pydvif.span(1, 10)
```

```
>>> pydvif.sort(c)
```

Parameters **curve** (Curve) – The curve to sort

pydvpy.**span**(*xmin*, *xmax*, *numpts=100*)

Generates a straight line of slope 1 and y intercept 0 in the specified domain with an optional number of points.

```
>>> c = pydvif.span(1, 10)
```

Parameters

- **xmin** (*float*) – The minimum x value
- **xmax** (*float*) – The maximum x value

- **numpts** (*int*) – The number of points used to plot the line

Returns curve – the curve object representing the straight line.

pydvpy.**sqr**(*curvelist*)
Take the square of the y values in a curve or list of curves.

Parameters **curvelist** (*curve or list*) – the curve or list of curves

pydvpy.**sqrt**(*curvelist*)
Take the square root of the y values in a curve or list of curves.

Parameters **curvelist** (*curve or list*) – the curve or list of curves

pydvpy.**sqrtx**(*curvelist*)
Take the square root of the x values in a curve or list of curves.

Parameters **curvelist** (*curve or list*) – the curve or list of curves

pydvpy.**sqrx**(*curvelist*)
Take the square of the x values in a curve or list of curves.

Parameters **curvelist** (*curve or list*) – the curve or list of curves

pydvpy.**subsample**(*curvelist*, *stride=2*, *verbose=False*)
Subsample the curve or list of curves, i.e., reduce to every nth value.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.subsample(curves, 4)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters

- **curvelist** (*Curve or list*) – The curve or list of curves
- **stride** (*int*) – The step size through the array
- **verbose** (*bool*) – If True additional information will be printed to stdout

pydvpy.**subtract**(*curvelist*)
Take difference of curves.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> c = pydvif.subtract(curves)
```

Parameters **curvelist** (*list*) – The list of curves

Returns curve – the curve containing the difference of the curves

pydvpy.**tan**(*curvelist*)
Take the tangent of y values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.tan(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**tanh**(*curvelist*)

Take the hyperbolic tangent of y values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.tanh(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**tanhx**(*curvelist*)

Take the hyperbolic tangent of x values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.tanhx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**tanx**(*curvelist*)

Take the tangent of x values of a single curve or multiple curves in list.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.tanx(curves)
```

Parameters **curvelist** (*curve or list*) – A single curve or a list of curves

pydvpy.**vs**(*c1*, *c2*)

Create a new curve that will plot as the range of the first curve against the range of the second curve.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> c1 = pydvif.vs(curves[0], curves[1])
```

```
>>> curves.append(c1)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters

- **c1** (Curve) – The first curve
- **c2** (Curve) – The second curve

Returns Curve – the new curve

pydvpy.**xindex**(*curvelist*)

Create curves with y-values vs. integer index values.

```
>>> curves = pydvif.read('testData.txt')
```

```
>>> pydvif.xindex(curves)
```

```
>>> pydvif.create_plot(curves, legend=True)
```

Parameters **curvelist** (*Curve or list*) – The curve or list of curves

pydvpy.**xmax**(*curvelist*, *limit*)
Filter out points in the curve or list of curves whose x values are greater than limit.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **limit** (*float*) – The maximum value

pydvpy.**xmin**(*curvelist*, *min*)
Filter out points in the curve or list of curves whose x values are less than min.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **min** (*float*) – The minimum value

pydvpy.**xminmax**(*curvelist*, *min*, *max*)
Filter out points in the curve or list of curves whose x values are less than min or greater than max.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **min** (*float*) – The minimum value
- **max** (*float*) – The maximum value

pydvpy.**y0**(*curvelist*)
Take the Bessel function of the second kind of the zeroth order for the y values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**y0x**(*curvelist*)
Take the Bessel function of the second kind of the zeroth order for the x values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**y1**(*curvelist*)
Take the Bessel function of the second kind of the first order for the y values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**y1x**(*curvelist*)
Take the Bessel function of the second kind of the first order for the x values of curves in curvelist.

Parameters **curvelist** (*curve or list*) – The curve or list of curves

pydvpy.**ymax**(*curvelist*, *max*)
Filter out points in the curve or list of curves whose y values are greater than limit.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **max** (*float*) – The maximum value

pydvpy.**ymin**(*curvelist*, *min*)
Filter out points in the curve or list of curves whose y values are less than min.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **min** (*float*) – The minimum value

pydvpy.**yminmax**(*curvelist*, *min*, *max*)
Filter out points in the curve or list of curves whose y values are less than min or greater than max.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **min** (*float*) – The minimum value
- **max** (*float*) – The maximum value

pydvpy.**yn**(*curvelist*, *n*)
Take the Bessel function of the second kind of order n for the y values of curves in curvelist.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **n** (*int*) – The order

pydvpy.**ynx**(*curvelist*, *n*)
Take the Bessel function of the second kind of order n for the x values of curves in curvelist.

Parameters

- **curvelist** (*curve or list*) – The curve or list of curves
- **n** (*int*) – The order

2.1.2 curve module

class curve.**Curve**(*filename=''*, *name=''*)
Bases: object

color = ''

copy()

dashes = None

drawstyle = 'default'

ebar = None

edited = False

erange = None

filename = ''

hidden = False

legend_show = True

linespoints = False

linestyle = '-'

linewidth = None

marker = '.'

markeredgecolor = None

markerfacecolor = None

markersize = 3

markerstyle = None

name = ''

normalize()

plotname = ''

plotprecedence = 0

scatter = False

title = ''

x = array([], dtype=float64)

xlabel = ''

y = array([], dtype=float64)

ylabel = ''

curve.**append**(*a*, *b*)

Merge curve a and curve b over the union of their domains. Where domains overlap, take the average of the curve's y-values.

Parameters

- **a** (*curve*) – Curve A
- **b** (*curve*) – Curve B

Returns a new curve resulting from the merging of curve a and curve b

curve.**getinterp**(*a*, *b*, *left=None*, *right=None*, *samples=100*, *match='domain'*)

Gets the interpolated and domain matched versions of the two curves.

Parameters

- **a** (*curve*) – Curve A
- **b** (*curve*) – Curve B
- **left** (*float, optional*) – Value to return for *x* < *a.x[0]*, default is *a.y[0]*.
- **right** – Value to return for *x* > *a.x[-1]*, default is *a.y[-1]*.
- **{'domain','step'},optional** (*match*) – A string indicating how to interpolate the two curves

Type right: float, optional

Returns curve pair – the interpolated and domain matched versions of a and b

curve.**interp1d**(*a*, *num=100*, *retstep=False*)

Gets the interpolated values of the curve with the specified number of samples.

Parameters

- **a** (*curve*) – Curve A
- **num** – Number of samples to generate. Default is 100. Must be non-negative.

- **retstep** – return the spacing between samples

Type num: int, optional

Type retstep: bool, optional

Returns ia: curve – the interpolated and dimensions matched version of a step: float, optional – only returned if retstep is True. Size of the spacing between samples

2.1.3 pdv module

2.1.4 pdvplot module

class pdvplot.**Plotter**(*pydvcmd*)

Bases: PySide2.QtWidgets.QMainWindow

canvas = None

closeEvent(*self*, *event: PySide2.QtGui.QCloseEvent*) → None

defaultPlotLayout = None

fig = None

figcolor = 'white'

plotChanged = False

showCurvelistDialog()

Shows a dialog with the output of the list command in a table.

showMenuDialog()

Shows a dialog with the output of the menu command in a table.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

style = 'ggplot'

updateDialogs()

Updates the list and menu dialogs if visible.

updatePlotGeometry(*geometry='de'*)

Updates the size and location of the window. Using an action to trigger the update to ensure that the resizing is happening on the main GUI thread.

2.1.5 pdvutil module

pdvutil.**getCurveIndex**(*plotname*, *plotlist*)

pdvutil.**get_actual_index**(*origref*, *val*)

pdvutil.**getletterargs**(*line*)

pdvutil.**getnumberargs**(*line*, *filelist*)

pdvutil.**parsemath**(*line*, *plotlist*, *commander*, *xdomain*)

pdvutil.**truncate**(*string*, *size*)

2.1.6 pdvnavbar module

class pdvnavbar.**PyDVToolbar**(*canvas_*, *parent_*, *coordinates_*)

Bases: matplotlib.backends.backend_qt5.NavigationToolbar2QT

edit_parameters()

staticMetaObject = <PySide2.QtCore.QMetaObject object>

zoom(**args*)

Toggle zoom to rect mode.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

curve, 82

p

pdvnavbar, 85
pdvplot, 84
pdvutil, 84
pydvpy, 55

INDEX

A

abs() (*in module pydvpy*), 55
absx() (*in module pydvpy*), 55
acos() (*in module pydvpy*), 55
acosh() (*in module pydvpy*), 56
acoshx() (*in module pydvpy*), 56
acosx() (*in module pydvpy*), 56
add() (*in module pydvpy*), 56
alpha() (*in module pydvpy*), 56
append() (*in module curve*), 83
appendcurves() (*in module pydvpy*), 56
asin() (*in module pydvpy*), 57
asinh() (*in module pydvpy*), 57
asinhx() (*in module pydvpy*), 57
asinx() (*in module pydvpy*), 57
atan() (*in module pydvpy*), 57
atan2() (*in module pydvpy*), 58
atanh() (*in module pydvpy*), 58
atanhx() (*in module pydvpy*), 58
atanx() (*in module pydvpy*), 58
average_curve() (*in module pydvpy*), 58

C

canvas (*pdvplot.Plotter attribute*), 84
closeEvent() (*pdvplot.Plotter method*), 84
color (*curve.Curve attribute*), 82
convolve() (*in module pydvpy*), 59
convolve_int() (*in module pydvpy*), 59
convolveb() (*in module pydvpy*), 59
convolvec() (*in module pydvpy*), 59
copy() (*curve.Curve method*), 82
correlate() (*in module pydvpy*), 60
cos() (*in module pydvpy*), 60
cosh() (*in module pydvpy*), 60
coshx() (*in module pydvpy*), 60
cosx() (*in module pydvpy*), 61
create_plot() (*in module pydvpy*), 61
curve
    module, 82
Curve (*class in curve*), 82

D

dashes (*curve.Curve attribute*), 82
defaultPlotLayout (*pdvplot.Plotter attribute*), 84
derivative() (*in module pydvpy*), 61
diffMeasure() (*in module pydvpy*), 62
disp() (*in module pydvpy*), 62
divide() (*in module pydvpy*), 62
divx() (*in module pydvpy*), 62
divy() (*in module pydvpy*), 63
drawstyle (*curve.Curve attribute*), 82
dupx() (*in module pydvpy*), 63
dx() (*in module pydvpy*), 63
dy() (*in module pydvpy*), 63

E

ebar (*curve.Curve attribute*), 82
edit_parameters() (*pdvnavbar.PyDVToolbar method*), 85
edited (*curve.Curve attribute*), 82
erange (*curve.Curve attribute*), 82
errorbar() (*in module pydvpy*), 64
errorrange() (*in module pydvpy*), 64
exp() (*in module pydvpy*), 65
expx() (*in module pydvpy*), 65

F

fft() (*in module pydvpy*), 65
fig (*pdvplot.Plotter attribute*), 84
figcolor (*pdvplot.Plotter attribute*), 84
filename (*curve.Curve attribute*), 82
filtercurves() (*in module pydvpy*), 66
fit() (*in module pydvpy*), 66

G

gaussian() (*in module pydvpy*), 66
get_actual_index() (*in module pdvutil*), 84
get_styles() (*in module pydvpy*), 67
getCurveIndex() (*in module pdvutil*), 84
getdomain() (*in module pydvpy*), 67
getinterp() (*in module curve*), 83
getletterargs() (*in module pdvutil*), 84
getnumberargs() (*in module pdvutil*), 84

getnumpoints() (*in module pydvpy*), 67
getrange() (*in module pydvpy*), 67
getx() (*in module pydvpy*), 68
gety() (*in module pydvpy*), 68
getymax() (*in module pydvpy*), 68
getymin() (*in module pydvpy*), 68

H

hidden (*curve.Curve attribute*), 82

I

integrate() (*in module pydvpy*), 69
interp1d() (*in module curve*), 83

J

j0() (*in module pydvpy*), 69
j0x() (*in module pydvpy*), 69
j1() (*in module pydvpy*), 69
j1x() (*in module pydvpy*), 69
jn() (*in module pydvpy*), 69
jnx() (*in module pydvpy*), 69

L

l1() (*in module pydvpy*), 69
l2() (*in module pydvpy*), 70
legend_show (*curve.Curve attribute*), 82
line() (*in module pydvpy*), 70
linespoints (*curve.Curve attribute*), 82
linestyle (*curve.Curve attribute*), 82
linewidth (*curve.Curve attribute*), 82
log() (*in module pydvpy*), 70
log10() (*in module pydvpy*), 71
log10x() (*in module pydvpy*), 71
logx() (*in module pydvpy*), 71

M

makecurve() (*in module pydvpy*), 72
makeextensive() (*in module pydvpy*), 72
makeintensive() (*in module pydvpy*), 72
marker (*curve.Curve attribute*), 83
markeredgecolor (*curve.Curve attribute*), 83
markerfacecolor (*curve.Curve attribute*), 83
markersize (*curve.Curve attribute*), 83
markerstyle (*curve.Curve attribute*), 83
max_curve() (*in module pydvpy*), 72
min_curve() (*in module pydvpy*), 72
module
    curve, 82
    pdvnavbar, 85
    pdvplot, 84
    pdvutil, 84
    pydvpy, 55
multiply() (*in module pydvpy*), 73
mx() (*in module pydvpy*), 73
my() (*in module pydvpy*), 73

N

name (*curve.Curve attribute*), 83
norm() (*in module pydvpy*), 73
normalize() (*curve.Curve method*), 83

P

parsemath() (*in module pdvutil*), 84
pdvnavbar
    module, 85
pdvplot
    module, 84
pdvutil
    module, 84
plotChanged (*pdvplot.Plotter attribute*), 84
plotname (*curve.Curve attribute*), 83
plotprecedence (*curve.Curve attribute*), 83
Plotter (*class in pdvplot*), 84
powa() (*in module pydvpy*), 74
powax() (*in module pydvpy*), 74
powr() (*in module pydvpy*), 74
powrx() (*in module pydvpy*), 74
pydvpy
    module, 55
PyDVToolbar (*class in pdvnavbar*), 85

R

random() (*in module pydvpy*), 75
read() (*in module pydvpy*), 75
readcsv() (*in module pydvpy*), 75
readsina() (*in module pydvpy*), 76
recip() (*in module pydvpy*), 76
recipx() (*in module pydvpy*), 76
rev() (*in module pydvpy*), 76

S

save() (*in module pydvpy*), 77
savecsv() (*in module pydvpy*), 77
scatter (*curve.Curve attribute*), 83
showCurvelistDialog() (*pdvplot.Plotter method*), 84
showMenuDialog() (*pdvplot.Plotter method*), 84
sin() (*in module pydvpy*), 77
sinh() (*in module pydvpy*), 77
sinhx() (*in module pydvpy*), 78
sinx() (*in module pydvpy*), 78
smooth() (*in module pydvpy*), 78
sort() (*in module pydvpy*), 78
span() (*in module pydvpy*), 78
sqr() (*in module pydvpy*), 79
sqrt() (*in module pydvpy*), 79

sqrtx() (*in module pydvpy*), 79
sqrx() (*in module pydvpy*), 79
staticMetaObject (*pdvnavbar.PyDVToolbar attribute*), 85
staticMetaObject (*pdvplot.Plotter attribute*), 84
style (*pdvplot.Plotter attribute*), 84
subsample() (*in module pydvpy*), 79
subtract() (*in module pydvpy*), 79

T

tan() (*in module pydvpy*), 79
tanh() (*in module pydvpy*), 80
tanhx() (*in module pydvpy*), 80
tanx() (*in module pydvpy*), 80
title (*curve.Curve attribute*), 83
truncate() (*in module pdvutil*), 84

U

updateDialogs() (*pdvplot.Plotter method*), 84
updatePlotGeometry() (*pdvplot.Plotter method*), 84

V

vs() (*in module pydvpy*), 80

X

x (*curve.Curve attribute*), 83
xindex() (*in module pydvpy*), 80
xlabel (*curve.Curve attribute*), 83
xmax() (*in module pydvpy*), 81
xmin() (*in module pydvpy*), 81
xminmax() (*in module pydvpy*), 81

Y

y (*curve.Curve attribute*), 83
y0() (*in module pydvpy*), 81
y0x() (*in module pydvpy*), 81
y1() (*in module pydvpy*), 81
y1x() (*in module pydvpy*), 81
ylabel (*curve.Curve attribute*), 83
ymax() (*in module pydvpy*), 81
ymin() (*in module pydvpy*), 81
yminmax() (*in module pydvpy*), 82
yn() (*in module pydvpy*), 82
ynx() (*in module pydvpy*), 82

Z

zoom() (*pdvnavbar.PyDVToolbar method*), 85