API Documentation

May 28, 2007

Contents

1 Package matplotlib

This is a matlab(TM) style functional interface the matplotlib.

The following matlab(TM) compatible commands are provided by

```
>>> from pylab import *
```

Plotting commands

```
axes     - Create a new axes
axhline  - draw a horizontal line across axes
axvline  - draw a vertical line across axes
axhspan  - draw a horizontal bar across axes
axvspan  - draw a vertical bar across axes
axis     - Set or return the current axis limits
bar      - make a bar chart
barh     - a horizontal bar chart
boxplot  - make a box and whisker plot
cla      - clear current axes
clf      - clear a figure window
close    - close a figure window
colorbar - add a colorbar to the current figure
cohere   - make a plot of coherence
contour  - make a contour plot
csd      - make a plot of cross spectral density
draw     - Force a redraw of the current figure
errorbar - make an errorbar graph
figlegend - make legend on the figure rather than the axes
figimage  - make a figure image
figtext   - add text in figure coords
figure   - create or change active figure
fill     - make filled polygons
gca      - return the current axes
gcf      - return the current figure
gci      - get the current image, or None
get      - get a handle graphics property
gray     - set the current colormap to gray
jet      - set the current colormap to jet
hist     - make a histogram
hold     - set the axes hold state
legend   - make an axes legend
loglog   - a log log plot
imread   - load image file into array
imshow   - plot image data
pcolor   - make a pseudocolor plot
plot     - make a line plot
psd      - make a plot of power spectral density
```

```
  rc       - control the default params
  savefig  - save the current figure
  scatter  - make a scatter plot
  set      - set a handle graphics property
  semilogx - log x axis
  semilogy - log y axis
  show     - show the figures
  specgram - a spectrogram plot
  stem     - make a stem plot
  subplot  - make a subplot (numrows, numcols, axesnum)
  table    - add a table to the plot
  text     - add some text at location x,y to the current axes
  title    - add a title to the current axes
  xlim     - set/get the xlimits
  ylim     - set/get the ylimits
  xticks   - set/get the xticks
  yticks   - set/get the yticks
  xlabel   - add an xlabel to the current axes
  ylabel   - add a ylabel to the current axes

Matrix commands

  cumprod   - the cumulative product along a dimension
  cumsum    - the cumulative sum along a dimension
  detrend   - remove the mean or besdt fit line from an array
  diag      - the k-th diagonal of matrix
  diff      - the n-th differnce of an array
  eig       - the eigenvalues and eigen vectors of v
  eye       - a matrix where the k-th diagonal is ones, else zero
  find      - return the indices where a condition is nonzero
  fliplr    - flip the rows of a matrix up/down
  flipud    - flip the columns of a matrix left/right
  linspace  - a linear spaced vector of N values from min to max inclusive
  ones      - an array of ones
  rand      - an array from the uniform distribution [0,1]
  randn     - an array from the normal distribution
  rot90     - rotate matrix k*90 degress counterclockwise
  squeeze   - squeeze an array removing any dimensions of length 1
  tri       - a triangular matrix
  tril      - a lower triangular matrix
  triu      - an upper triangular matrix
  vander    - the Vandermonde matrix of vector x
  svd       - singular value decomposition
  zeros     - a matrix of zeros

Probability

  levypdf   - The levy probability density function from the char. func.
```

```
  normpdf   - The Gaussian probability density function
  rand      - random numbers from the uniform distribution
  randn     - random numbers from the normal distribution

Statistics

  corrcoef  - correlation coefficient
  cov       - covariance matrix
  max       - the maximum along dimension m
  mean      - the mean along dimension m
  median    - the median along dimension m
  min       - the minimum along dimension m
  norm      - the norm of vector x
  prod      - the product along dimension m
  ptp       - the max-min along dimension m
  std       - the standard deviation along dimension m
  sum       - the sum along dimension m

Time series analysis

  bartlett  - M-point Bartlett window
  blackman  - M-point Blackman window
  cohere    - the coherence using average periodiogram
  csd       - the cross spectral density using average periodiogram
  fft       - the fast Fourier transform of vector x
  hamming   - M-point Hamming window
  hanning   - M-point Hanning window
  hist      - compute the histogram of x
  kaiser    - M length Kaiser window
  psd       - the power spectral density using average periodiogram
  sinc      - the sinc function of array x

Other

  angle     - the angle of a complex array
  polyfit   - fit x, y to an n-th order polynomial
  polyval   - evaluate an n-th order polynomial
  roots     - the roots of the polynomial coefficients in p
  trapz     - trapezoidal integration


Credits: The plotting commands were provided by
John D. Hunter <jdhunter@ace.bsd.uhicago.edu>

Most of the other commands are from the Numeric, MLab and FFT, with
the exception of those in mlab.py provided by matplotlib.
```

1.1 Modules

- **afm**: This is a python interface to Adobe Font Metrics Files.
(Section 2, p. 33)
- **agg** *(Section 3, p. 36)*
- **art3d**: Wrap 2D artists so that they can pretend to be 3D
(Section 4, p. 83)
- **artist** *(Section 5, p. 91)*
- **axes** *(Section 6, p. 99)*
- **axes3d**: 3D projection glued onto 2D Axes.
(Section 7, p. 173)
- **axis**: Classes for the ticks and x and y axis
(Section 8, p. 183)
- **axis3d** *(Section 9, p. 194)*
- **backend_bases**: Abstract base classes define the primitives that renderers and graphics contexts must implement to serve as a matplotlib backend
(Section 10, p. 196)
- **cbook**: A collection of utility functions and classes.
(Section 11, p. 214)
- **cm**: This module contains the instantiations of color mapping classes
(Section 12, p. 223)
- **collections**: Classes for the efficient drawing of large collections of objects that share most properties, eg a large number of line segments or polygons
(Section 13, p. 225)
- **colorbar**: Colorbar toolkit with two classes and a function:
ColorbarBase is the base class with full colorbar drawing functionality.
(Section 14, p. 239)
- **colors**: A class for converting color arguments to RGB or RGBA
This class instantiates a single instance colorConverter that is used to convert matlab color strings to RGB.
(Section 15, p. 242)
- **contour**: These are classes to support contour plotting and labelling for the axes class
(Section 16, p. 249)
- **dates**: Matplotlib provides sophisticated date plotting capabilites, standing on the shoulders of python datetime, the add-on modules pytz and dateutils.
(Section 17, p. 252)
- **dviread**: An experimental module for reading single-page dvi files output by TeX.
(Section 18, p. 264)
- **figure**: Figure class – add docstring here!
(Section 19, p. 267)
- **finance**: A collection of modules for collecting, analyzing and plotting financial data.
(Section 20, p. 282)
- **font_manager**: A module for finding, managing, and using fonts across-platforms.
(Section 21, p. 285)
- **image**: The image module supports basic image loading, rescaling and display operations.
(Section 22, p. 292)
- **legend**: Place a legend on the axes at location loc.
(Section 23, p. 298)
- **lines**: This module contains all the 2D line class which can draw with a variety of line styles, markers

1.2 Functions

checkdep_dvipng()

checkdep_ghostscript()

checkdep_pdftops()

checkdep_tex()

compare_versions(*a*, *b*)

return True if a is greater than b

get_backend()

get_configdir(**args*, ***kwargs*)

Return the string representing the configuration dir. If s is the special string _default_, use HOME/.matplotlib. s must be writable

get_data_path(**args*, ***kwargs*)

get the path to matplotlib data

get_home(**args*, ***kwargs*)

Find user's home directory if possible. Otherwise raise error.
:see: http://mail.python.org/pipermail/python-list/2005-February/263921.html

get_py2exe_datafiles()

interactive(*b*)

Set interactive mode to boolean b.
If b is True, then draw after every plotting command, eg, after xlabel

is_interactive()

Return true if plot mode is interactive

is_string_like(*obj*)

matplotlib_fname()

```
Return the path to the rc file

Search order:

 * current working dir
 * environ var MATPLOTLIBRC
 * HOME/.matplotlib/matplotlibrc
 * MATPLOTLIBDATA/matplotlibrc
```

rc(*group*, ***kwargs*)

```
Set the current rc params.  Group is the grouping for the rc, eg
for lines.linewidth the group is 'lines', for axes.facecolor, the
group is 'axes', and so on.  Group may also be a list or tuple
of group names, eg ('xtick','ytick').  kwargs is a list of
attribute name/value pairs, eg

  rc('lines', linewidth=2, color='r')

sets the current rc params and is equivalent to

  rcParams['lines.linewidth'] = 2
  rcParams['lines.color'] = 'r'

The following aliases are available to save typing for interactive
users
    'lw'  : 'linewidth'
    'ls'  : 'linestyle'
    'c'   : 'color'
    'fc'  : 'facecolor'
    'ec'  : 'edgecolor'
    'mew' : 'markeredgewidth'
    'aa'  : 'antialiased'

Thus you could abbreviate the above rc command as

      rc('lines', lw=2, c='r')


Note you can use python's kwargs dictionary facility to store
dictionaries of default parameters.  Eg, you can customize the
font rc as follows

  font = {'family' : 'monospace',
          'weight' : 'bold',
          'size'   : 'larger',
         }

  rc('font', **font)  # pass in the font dict as kwargs

This enables you to easily switch between several configurations.
Use rcdefaults to restore the default rc params after changes.
```

rc_params(*fail_on_error*=`False`)

Return the default params updated from the values in the rc file

rcdefaults()

Restore the default rc params - the ones that were created at matplotlib load time

tk_window_focus()

Return true if focus maintenance under TkAgg on win32 is on. This currently works only for python.exe and IPython.exe. Both IDLE and Pythonwin.exe fail badly when tk_window_focus is on.

use(*arg*)

Set the matplotlib backend to one of the known backends

validate_aspect(*s*)

validate_backend(*s*, *fail_on_err*=`True`)

validate_bool(*b*)

Convert b to a boolean or raise

validate_color(*s*)

return a valid color arg

validate_comma_sep_str(*s*)

return a list

validate_float(*s*)

convert s to float or raise

validate_fontsize(*s*)

validate_int(*s*)

convert s to int or raise

validate_key(*key*, *val*, *line*, *cnt*, *fname*, *fail_on_error*)

validate_numerix(*s*)

return "Numeric" or "numarray" or "numpy" or raise

validate_path_exists(*s*)

If s is a path, return s, else False

validate_ps_distiller(*s*)

validate_toolbar(*s*)

return toolbar string 'None', 'classic', 'toolbar2'

validate_usetex(*s*)

validate_verbose(*s*)

validate_verbose_fileo(*s*)

1.3 Class ExampleInfo

1.4 Class ExampleManager

1.4.1 Methods

get_examples(*self*)

get_info(*self, s*)

return an ExampleInfo instance from s, the string content of an example

1.4.2 Class Variables

Name	Description
baseurl	**Value:** `'http://matplotlib.sf.net'` *(type=`str`)*
subdir	**Value:** `'examples/widgets'` *(type=`str`)*
urls	**Value:** `['http://matplotlib.sf.net/examples', 'http://m-atplotlib.sf.net/examples/widg...` *(type=`list`)*

1.5 Class validate_nseq_float

1.5.1 Methods

__init__(*self, n*)

__call__(*self, s*)

return a seq of n floats or raise

1.6 Class validate_nseq_int

1.6.1 Methods

__init__(*self*, *n*)

__call__(*self*, *s*)

return a seq of n ints or raise

1.7 Class ValidateInStrings

1.7.1 Methods

__init__(*self*, *valid*, *ignorecase*=`False`)

valid is a list of legal strings

__call__(*self*, *s*)

1.8 Class ValidateInterval

Value must be in interval

1.8.1 Methods

__init__(*self*, *vmin*, *vmax*, *closedmin*=`True`, *closedmax*=`True`)

__call__(*self*, *s*)

1.9 Class Verbose

A class to handle reporting. Set the fileo attribute to any file instance to handle the output. Default is sys.stdout

1.9.1 Methods

__init__(*self*, *level*)

ge(*self*, *level*)

return true if self.level is >= level

report(*self*, *s*, *level*=`'helpful'`)

print message s to self.fileo if self.level>=level. Return value indicates whether a message was issued

set_level(*self*, *level*)

set the verbosity to one of the Verbose.levels strings

wrap(*self*, *fmt*, *func*, *level*=`'helpful'`, *always*=`True`)

return a callable function that wraps func and reports it output through the verbose handler if current verbosity level is higher than level
if always is True, the report will occur on every function call; otherwise only on the first time the function is called

1.9.2 Class Variables

Name	Description
i	**Value:** `3` *(type=int)*
level	**Value:** `'debug-annoying'` *(type=str)*
levels	**Value:** `('silent', 'helpful', 'debug', 'debug-annoying')` *(type=tuple)*
vald	**Value:** `{'debug': 2, 'debug-annoying': 3, 'silent': 0, -'helpful': 1}` *(type=dict)*

2 Module matplotlib.afm

```
This is a python interface to Adobe Font Metrics Files.  Although a
number of other python implementations exist (and may be more complete
than mine) I decided not to go with them because either they were
either

  1) copyighted or used a non-BSD compatible license

  2) had too many dependencies and I wanted a free standing lib

  3) Did more than I needed and it was easier to write my own than
     figure out how to just get what I needed from theirs

It is pretty easy to use, and requires only built-in python libs

    >>> from afm import AFM
    >>> fh = file('ptmr8a.afm')
    >>> afm = AFM(fh)
    >>> afm.string_width_height('What the heck?')
    (6220.0, 683)
    >>> afm.get_fontname()
    'Times-Roman'
    >>> afm.get_kern_dist('A', 'f')
    0
    >>> afm.get_kern_dist('A', 'y')
    -92.0
    >>> afm.get_bbox_char('!')
    [130, -9, 238, 676]
    >>> afm.get_bbox_font()
    [-168, -218, 1000, 898]


AUTHOR:
  John D. Hunter <jdhunter@ace.bsd.uchicago.edu>
```

2.1 Functions

parse_afm(*fh*)

Parse the Adobe Font Metics file in file handle fh Return value is a (dhead, dcmetrics, dkernpairs, dcomposite) tuple where
dhead : a parse_header dict dcmetrics : a parse_composites dict dkernpairs : a parse_kern_pairs dict, possibly {} dcomposite : a parse_composites dict , possibly {}

2.2 Class AFM

2.2.1 Methods

__init__(*self*, *fh*)

Parse the AFM file in file object fh

get_angle(*self*)

Return the fontangle as float

get_bbox_char(*self*, *c*, *isord*=`False`)

get_familyname(*self*)

Return the font family name, eg, Times

get_fontname(*self*)

Return the font name, eg, Times-Roman

get_fullname(*self*)

Return the font full name, eg, Times-Roman

get_height_char(*self*, *c*, *isord*=`False`)

Get the height of character c from the bounding box. This is the ink height (space is 0)

get_kern_dist(*self*, *c1*, *c2*)

Return the kerning pair distance (possibly 0) for chars c1 and c2

get_name_char(*self*, *c*)

Get the name of the character, ie, ';' is 'semicolon'

get_str_bbox(*self*, *s*)

Return the string bounding box

get_weight(*self*)

Return the font weight, eg, 'Bold' or 'Roman'

get_width_char(*self*, *c*, *isord*=`False`)

Get the width of the character from the character metric WX field

string_width_height(*self*, *s*)

Return the string width (including kerning) and string height as a w,h tuple

3 Module matplotlib.agg

3.1 Class binary_data

__builtin__.object
    binary_data

3.1.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.1.2 Properties

Name	Description
data	
size	

3.1.3 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'data': <built-in function binary_data_data_get>, 'size': <built-in function... *(type=dict)*
__swig_setmethods__	**Value:** {'data': <built-in function binary_data_data_set>, 'size': <built-in function... *(type=dict)*

3.2 Class buffer

__builtin__.object
    buffer

3.2.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

to_string(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.2.2 Properties

Name	Description
data	
freemem	
height	
stride	
width	

3.2.3 Class Variables

Name	Description
__swig_getmethods__	**Value:** {`'width': <built-in function buffer_width_get>,-'stride': <built-in function ...` *(type=`dict`)*
__swig_setmethods__	**Value:** {`'data': <built-in function buffer_data_set>, '-freemem': <built-in function b...` *(type=`dict`)*

3.3 Class conv_adaptor_vcgen_curve

__builtin__.object
conv_adaptor_vcgen_curve

Known Subclasses: conv_stroke_curve

3.3.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

generator(**args*)

markers(**args*)

rewind(**args*)

set_source(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.3.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.4 Class conv_adaptor_vcgen_curvetrans

__builtin__.object
└ **conv_adaptor_vcgen_curvetrans**

3.4.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

generator(**args*)

markers(**args*)

rewind(**args*)

set_source(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.4.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.5 Class conv_adaptor_vcgen_path

__builtin__.object

conv_adaptor_vcgen_path

Known Subclasses: conv_stroke_path

3.5.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

generator(**args*)

markers(**args*)

rewind(**args*)

set_source(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.5.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.6 Class conv_adaptor_vcgen_transcurve

__builtin__.object
└ **conv_adaptor_vcgen_transcurve**

3.6.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

generator(**args*)

markers(**args*)

rewind(**args*)

set_source(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.6.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.7 Class conv_adaptor_vcgen_transpath

__builtin__.object
└─ **conv_adaptor_vcgen_transpath**

Known Subclasses: conv_stroke_transpath

3.7.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

generator(**args*)

markers(**args*)

rewind(**args*)

set_source(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.7.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.8 Class conv_curve_path

__builtin__.object
conv_curve_path

3.8.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

approximation_scale(**args*)

rewind(**args*)

set_source(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.8.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.9 Class conv_curve_trans

__builtin__.object
conv_curve_trans

3.9.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

approximation_scale(**args*)

rewind(**args*)

set_source(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.9.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.10 Class conv_stroke_curve

__builtin__.object
matplotlib.agg.conv_adaptor_vcgen_curve
conv_stroke_curve

3.10.1 Methods

__init__(*self*, **args*)
Overrides: matplotlib.agg.conv_adaptor_vcgen_curve.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

approximation_scale(**args*)

inner_line_join(**args*)

inner_miter_limit(**args*)

line_cap(**args*)

line_join(**args*)

miter_limit(**args*)

miter_limit_theta(**args*)

shorten(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__
Inherited from conv_adaptor_vcgen_curve: generator, markers, rewind, set_source, vertex

3.10.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.11 Class conv_stroke_curvetrans

__builtin__.object

conv_stroke_curvetrans

3.11.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

approximation_scale(**args*)

inner_line_join(**args*)

inner_miter_limit(**args*)

line_cap(**args*)

line_join(**args*)

miter_limit(**args*)

miter_limit_theta(**args*)

shorten(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.11.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.12 Class conv_stroke_path

__builtin__.object

matplotlib.agg.conv_adaptor_vcgen_path

conv_stroke_path

3.12.1 Methods

__init__(*self*, **args*)
Overrides: matplotlib.agg.conv_adaptor_vcgen_path.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

approximation_scale(**args*)

inner_line_join(**args*)

inner_miter_limit(**args*)

line_cap(**args*)

line_join(**args*)

miter_limit(**args*)

miter_limit_theta(**args*)

shorten(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__
Inherited from conv_adaptor_vcgen_path: generator, markers, rewind, set_source, vertex

3.12.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.13 Class conv_stroke_transcurve

```
__builtin__.object
        |
        conv_stroke_transcurve
```

3.13.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

approximation_scale(**args*)

inner_line_join(**args*)

inner_miter_limit(**args*)

line_cap(**args*)

line_join(**args*)

miter_limit(**args*)

miter_limit_theta(**args*)

shorten(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.13.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.14 Class conv_stroke_transpath

__builtin__.object

matplotlib.agg.conv_adaptor_vcgen_transpath

conv_stroke_transpath

3.14.1 Methods

__init__(*self*, **args*)
Overrides: matplotlib.agg.conv_adaptor_vcgen_transpath.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

approximation_scale(**args*)

inner_line_join(**args*)

inner_miter_limit(**args*)

line_cap(**args*)

line_join(**args*)

miter_limit(**args*)

miter_limit_theta(**args*)

shorten(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__
Inherited from conv_adaptor_vcgen_transpath: generator, markers, rewind, set_source, vertex

3.14.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.15 Class conv_transform_curve

```
__builtin__.object ─┐
                    conv_transform_curve
```

3.15.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

rewind(**args*)

set_source(**args*)

transformer(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.15.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.16 Class conv_transform_path

__builtin__.object
 └ **conv_transform_path**

3.16.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

rewind(**args*)

set_source(**args*)

transformer(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.16.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.17 Class null_markers

__builtin__.object

null_markers

3.17.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

add_vertex(**args*)

prepare_src(**args*)

remove_all(**args*)

rewind(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.17.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.18 Class order_abgr

__builtin__.object
order_abgr

3.18.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.18.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*
A	**Value:** 0 *(type=*`int`*)*
B	**Value:** 1 *(type=*`int`*)*
G	**Value:** 2 *(type=*`int`*)*
R	**Value:** 3 *(type=*`int`*)*
rgba_tag	**Value:** 4 *(type=*`int`*)*

3.19 Class order_argb

__builtin__.object
order_argb

3.19.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self, name, value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.19.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*
A	**Value:** `0` *(type=`int`)*
B	**Value:** `3` *(type=`int`)*
G	**Value:** `2` *(type=`int`)*
R	**Value:** `1` *(type=`int`)*
rgba_tag	**Value:** `4` *(type=`int`)*

3.20 Class order_bgr

__builtin__.object
order_bgr

3.20.1 Methods

__init__(*self, *args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self, name*)

__setattr__(*self, name, value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.20.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*
B	**Value:** `0` *(type=`int`)*
G	**Value:** `1` *(type=`int`)*
R	**Value:** `2` *(type=`int`)*
rgb_tag	**Value:** `3` *(type=`int`)*

3.21 Class order_bgra

__builtin__.object
  order_bgra

3.21.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.21.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*
A	**Value:** `3` *(type=*`int`*)*
B	**Value:** `0` *(type=*`int`*)*
G	**Value:** `1` *(type=*`int`*)*
R	**Value:** `2` *(type=*`int`*)*
rgba_tag	**Value:** `4` *(type=*`int`*)*

3.22 Class order_rgb

__builtin__.object
  order_rgb

3.22.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.22.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*
B	**Value:** 2 *(type=int)*
G	**Value:** 1 *(type=int)*
R	**Value:** 0 *(type=int)*
rgb_tag	**Value:** 3 *(type=int)*

3.23 Class order_rgba

__builtin__.object
  order_rgba

3.23.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.23.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*
A	**Value:** 3 *(type=int)*
B	**Value:** 2 *(type=int)*
G	**Value:** 1 *(type=int)*
R	**Value:** 0 *(type=int)*
rgba_tag	**Value:** 4 *(type=int)*

3.24 Class path_storage

```
__builtin__.object ─┐
                    path_storage
```

3.24.1 Methods

__init__(*self*, **args*) Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

add_poly(**args*)

add_vertex(**args*)

arc_rel(**args*)

arc_to(**args*)

arrange_orientations(**args*)

arrange_orientations_all_paths(**args*)

close_polygon(**args*)

command(**args*)

copy_from(**args*)

curve3(**args*)

curve3_rel(**args*)

curve4(**args*)

curve4_rel(**args*)

end_poly(**args*)

flip_x(**args*)

flip_y(**args*)

last_vertex(**args*)

line_rel(**args*)

line_to(**args*)

modify_command(**args*)

modify_vertex(**args*)

move_rel(**args*)

move_to(**args*)

prev_vertex(**args*)

rel_to_abs(**args*)

remove_all(**args*)

rewind(**args*)

start_new_path(**args*)

total_vertices(**args*)

vertex(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.24.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.25 Class pixel64_type

__builtin__.object
└ **pixel64_type**

3.25.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.25.2 Properties

Name	Description
c	

3.25.3 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'c': <built-in function pixel64_type_c_get>} *(type=`dict`)*
__swig_setmethods__	**Value:** {'c': <built-in function pixel64_type_c_set>} *(type=`dict`)*

3.26 Class pixel_format_rgba

__builtin__.object
└ **pixel_format_rgba**

3.26.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self, name*)

__setattr__(*self, name, value*)

attach(**args*)

blend_color_hspan(**args*)

blend_color_vspan(**args*)

blend_hline(**args*)

blend_pixel(**args*)

blend_solid_hspan(**args*)

blend_solid_vspan(**args*)

blend_vline(**args*)

copy_color_hspan(**args*)

copy_from(**args*)

copy_hline(**args*)

copy_pixel(**args*)

copy_vline(**args*)

demultiply(**args*)

height(**args*)

pixel(**args*)

premultiply(**args*)

row(**args*)

span(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.26.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*
base_mask	**Value:** 255 *(type=int)*
base_shift	**Value:** 8 *(type=int)*
base_size	**Value:** 256 *(type=int)*

3.27 Class point_type

__builtin__.object

point_type

3.27.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.27.2 Properties

Name	Description
x	
y	

3.27.3 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'y': <built-in function point_type_y_get>, 'x'-: <built-in function point_typ... *(type=`dict`)*
__swig_setmethods__	**Value:** {'y': <built-in function point_type_y_set>, 'x'-: <built-in function point_typ... *(type=`dict`)*

3.28 Class rasterizer_scanline_aa

__builtin__.object
└─ **rasterizer_scanline_aa**

3.28.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

add_path(**args*)

add_vertex(**args*)

apply_gamma(**args*)

calculate_alpha(**args*)

clip_box(**args*)

close_polygon(**args*)

filling_rule(**args*)

hit_test(**args*)

line_to(**args*)

line_to_d(**args*)

max_x(**args*)

max_y(**args*)

min_x(**args*)

min_y(**args*)

move_to(**args*)

move_to_d(**args*)

reset(**args*)

reset_clipping(**args*)

rewind_scanlines(**args*)

sort(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.28.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.29 Class rect

__builtin__.object
└ **rect**

3.29.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

clip(**args*)

is_valid(**args*)

normalize(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.29.2 Properties

Name	Description
x1	
x2	
y1	
y2	

3.29.3 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'y1': <built-in function rect_y1_get>, 'x2': - <built-in function rect_x2_get>... *(type=`dict`)*
__swig_setmethods__	**Value:** {'y1': <built-in function rect_y1_set>, 'x2': - <built-in function rect_x2_set>... *(type=`dict`)*

3.30 Class rect_d

__builtin__.object

rect_d

3.30.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

clip(**args*)

is_valid(**args*)

normalize(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.30.2 Properties

Name	Description
x1	
x2	
y1	
y2	

3.30.3 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'y1': <built-in function rect_d_y1_get>, 'x2':-<built-in function rect_d_x2_... *(type=dict)*
__swig_setmethods__	**Value:** {'y1': <built-in function rect_d_y1_set>, 'x2':-<built-in function rect_d_x2_... *(type=dict)*

3.31 Class renderer_base_rgba

__builtin__.object
└ **renderer_base_rgba**

3.31.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self, name, value*)

attach(*args*)

blend_bar(*args*)

blend_color_hspan(*args*)

blend_color_hspan_no_clip(*args*)

blend_color_vspan(*args*)

blend_color_vspan_no_clip(*args*)

blend_hline(*args*)

blend_pixel(*args*)

blend_solid_hspan(*args*)

blend_solid_vspan(*args*)

blend_vline(*args*)

bounding_clip_box(*args*)

bounding_xmax(*args*)

bounding_xmin(*args*)

bounding_ymax(*args*)

bounding_ymin(*args*)

clear(*args*)

clear_rgba(*args*)

clear_rgba8(*args*)

clip_box(*args*)

clip_box_naked(*args*)

clip_rect_area(**args*)

copy_bar(**args*)

copy_color_hspan(**args*)

copy_color_hspan_no_clip(**args*)

copy_from(**args*)

copy_hline(**args*)

copy_pixel(**args*)

copy_vline(**args*)

first_clip_box(**args*)

height(**args*)

inbox(**args*)

next_clip_box(**args*)

pixel(**args*)

ren(**args*)

reset_clipping(**args*)

span(**args*)

width(**args*)

xmax(**args*)

xmin(**args*)

ymax(**args*)

ymin(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.31.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.32 Class renderer_scanline_aa_solid_rgba

__builtin__.object
renderer_scanline_aa_solid_rgba

3.32.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

attach(**args*)

color(**args*)

color_rgba(**args*)

color_rgba8(**args*)

prepare(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.32.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.33 Class renderer_scanline_bin_solid_rgba

__builtin__.object
└ **renderer_scanline_bin_solid_rgba**

3.33.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

attach(**args*)

color(**args*)

color_rgba(**args*)

color_rgba8(**args*)

prepare(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.33.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.34 Class rendering_buffer

__builtin__.object
└ **rendering_buffer**

3.34.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

attach(**args*)

attachb(**args*)

buf(**args*)

clear(**args*)

copy_from(**args*)

height(**args*)

next_row(**args*)

row(**args*)

rows(**args*)

stride(**args*)

stride_abs(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.34.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.35 Class rgba

__builtin__.object
└ **rgba**

3.35.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

clear(**args*)

demultiply(**args*)

gradient(**args*)

opacity(**args*)

premultiply(**args*)

transparent(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.35.2 Static Methods

from_wavelength(*...*)

no_color(*...*)

3.35.3 Properties

Name	Description
a	
b	
g	

continued on next page

Name	Description
r	

3.35.4 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'a': <built-in function rgba_a_get>, 'b': <built-in function rgba_b_get>, 'g... *(type=`dict`)*
__swig_setmethods__	**Value:** {'a': <built-in function rgba_a_set>, 'r': <built-in function rgba_r_set>, 'b... *(type=`dict`)*

3.36 Class rgba16

__builtin__.object

rgba16

3.36.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

clear(**args*)

demultiply(**args*)

gradient(**args*)

opacity(**args*)

premultiply(**args*)

transparent(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.36.2 Static Methods

from_wavelength(...)

no_color(...)

3.36.3 Properties

Name	Description
a	
b	
g	
r	

3.36.4 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'a': <built-in function rgba16_a_get>, 'b': <-built-in function rgba16_b_get>... *(type=dict)*
__swig_setmethods__	**Value:** {'a': <built-in function rgba16_a_set>, 'r': <-built-in function rgba16_r_set>... *(type=dict)*
base_mask	**Value:** 65535 *(type=int)*
base_shift	**Value:** 16 *(type=int)*
base_size	**Value:** 65536 *(type=int)*

3.37 Class rgba8

__builtin__.object
└ **rgba8**

3.37.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

clear(**args*)

demultiply(**args*)

gradient(**args*)

opacity(**args*)

premultiply(**args*)

transparent(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.37.2 Static Methods

from_wavelength(*...*)

no_color(*...*)

3.37.3 Properties

Name	Description
a	
b	
g	
r	

3.37.4 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'a': <built-in function rgba8_a_get>, 'b': <b-uilt-in function rgba8_b_get>, ... *(type=dict)*
__swig_setmethods__	**Value:** {'a': <built-in function rgba8_a_set>, 'r': <b-uilt-in function rgba8_r_set>, ... *(type=dict)*
base_mask	**Value:** 255 *(type=int)*
base_shift	**Value:** 8 *(type=int)*
base_size	**Value:** 256 *(type=int)*

3.38 Class scanline32_bin

```
__builtin__.object ─┐
                    scanline32_bin
```

3.38.1 Methods

__init__(*self*, **args*) Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

add_cell(**args*)

add_cells(**args*)

add_span(**args*)

finalize(**args*)

num_spans(**args*)

reset(**args*)

reset_spans(**args*)

y(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.38.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.39 Class scanline_bin

```
__builtin__.object ─┐
                    scanline_bin
```

3.39.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

add_cell(**args*)

add_cells(**args*)

add_span(**args*)

finalize(**args*)

num_spans(**args*)

reset(**args*)

reset_spans(**args*)

y(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.39.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.40 Class scanline_p8

__builtin__.object
└── **scanline_p8**

3.40.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

add_cell(**args*)

add_cells(**args*)

add_span(**args*)

begin(**args*)

finalize(**args*)

num_spans(**args*)

reset(**args*)

reset_spans(**args*)

y(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.40.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.41 Class trans_affine

__builtin__.object
└ **trans_affine**

Known Subclasses: trans_affine_rotation, trans_affine_scaling, trans_affine_skewing, trans_affine_translation

3.41.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__eq__(**args*)

__getattr__(*self*, *name*)

__imul__(**args*)

__invert__(**args*)

__mul__(**args*)

__ne__(**args*)

__setattr__(*self*, *name*, *value*)

as_vec6(**args*)

determinant(**args*)

flip_x(**args*)

flip_y(**args*)

get_rotation(**args*)

get_scaling(**args*)

get_translation(**args*)

inverse_transform(**args*)

invert(**args*)

is_equal(**args*)

is_identity(**args*)

load_from(**args*)

multiply(**args*)

parl_to_parl(**args*)

parl_to_rect(**args*)

premultiply(**args*)

rect_to_parl(**args*)

reset(**args*)

scale(**args*)

transform(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.41.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.42 Class trans_affine_rotation

__builtin__.object
matplotlib.agg.trans_affine
trans_affine_rotation

3.42.1 Methods

__init__(*self*, **args*)
Overrides: matplotlib.agg.trans_affine.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__
Inherited from trans_affine: __eq__, __imul__, __invert__, __mul__, __ne__, as_vec6, determinant, flip_x, flip_y, get_rotation, get_scaling, get_translation, inverse_transform, invert, is_equal, is_identity, load_from, multiply, parl_to_parl, parl_to_rect, premultiply, rect_to_parl, reset, scale, transform

3.42.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.43 Class trans_affine_scaling

__builtin__.object
matplotlib.agg.trans_affine
trans_affine_scaling

3.43.1 Methods

__init__(*self*, **args*)
Overrides: matplotlib.agg.trans_affine.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__
Inherited from trans_affine: __eq__, __imul__, __invert__, __mul__, __ne__, as_vec6, determinant, flip_x, flip_y, get_rotation, get_scaling, get_translation, inverse_transform, invert, is_equal, is_identity, load_from, multiply, parl_to_parl, parl_to_rect, premultiply, rect_to_parl, reset, scale, transform

3.43.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.44 Class trans_affine_skewing

```
__builtin__.object
    |
    matplotlib.agg.trans_affine
        |
        trans_affine_skewing
```

3.44.1 Methods

__init__(*self*, **args*)
Overrides: matplotlib.agg.trans_affine.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__
Inherited from trans_affine: __eq__, __imul__, __invert__, __mul__, __ne__, as_vec6, determinant, flip_x, flip_y, get_rotation, get_scaling, get_translation, inverse_transform, invert, is_equal, is_identity, load_from, multiply, parl_to_parl, parl_to_rect, premultiply, rect_to_parl, reset, scale, transform

3.44.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=dict)*
__swig_setmethods__	**Value:** {} *(type=dict)*

3.45 Class trans_affine_translation

```
__builtin__.object
    |
    matplotlib.agg.trans_affine
        |
        trans_affine_translation
```

3.45.1 Methods

__init__(*self*, **args*)
Overrides: matplotlib.agg.trans_affine.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__
Inherited from trans_affine: __eq__, __imul__, __invert__, __mul__, __ne__, as_vec6, determinant, flip_x, flip_y, get_rotation, get_scaling, get_translation, inverse_transform, invert, is_equal, is_identity, load_from, multiply, parl_to_parl, parl_to_rect, premultiply, rect_to_parl, reset, scale, transform

3.45.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=`dict`)*
__swig_setmethods__	**Value:** {} *(type=`dict`)*

3.46 Class vcgen_stroke

__builtin__.object
vcgen_stroke

3.46.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

add_vertex(**args*)

approximation_scale(**args*)

inner_line_join(**args*)

inner_miter_limit(**args*)

line_cap(**args*)

line_join(**args*)

miter_limit(**args*)

miter_limit_theta(**args*)

remove_all(**args*)

rewind(**args*)

shorten(**args*)

vertex(**args*)

width(**args*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.46.2 Class Variables

Name	Description
__swig_getmethods__	**Value:** {} *(type=*`dict`*)*
__swig_setmethods__	**Value:** {} *(type=*`dict`*)*

3.47 Class vertex_type

__builtin__.object
vertex_type

3.47.1 Methods

__init__(*self*, **args*)
Overrides: __builtin__.object.__init__

__del__(*self*)

__getattr__(*self, name*)

__setattr__(*self, name, value*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __str__

3.47.2 Properties

Name	Description
cmd	
x	
y	

3.47.3 Class Variables

Name	Description
__swig_getmethods__	**Value:** {'y': <built-in function vertex_type_y_get>, 'x-': <built-in function vertex_t... *(type=dict)*
__swig_setmethods__	**Value:** {'y': <built-in function vertex_type_y_set>, 'x-': <built-in function vertex_t... *(type=dict)*

4 Module matplotlib.art3d

Wrap 2D artists so that they can pretend to be 3D

4.1 Functions

draw(*text*, *renderer*)

draw_linec(*self*, *renderer*)

draw_polyc(*self*, *renderer*)

get_colors(*c*, *num*)

Stretch the color argument to provide the required number num

image_draw(*image*, *renderer*)

iscolor(*c*)

juggle_axes(*xs*, *ys*, *zs*, *dir*)

Depending on the direction of the plot re-order the axis
This is so that 2d plots can be plotted along any direction.

line_draw(*self*, *renderer*)

Draw a 2D line as a 3D line

owrap(*text*)

patch_draw(*self*, *renderer*)

set_line_data(*line*, *xs*, *ys*, *zs*)

set_text_data(*text*, *x*, *y*, *z*)

text_draw(*self*, *renderer*)

wrap_2d_fn(*patch*, *zs*, *dir*=`'z'`, *fn*=`<function patch_draw at 0x856a7d4>`)

wrap_image(*image*, *extent*)

wrap_line(*line*, *zs*, *dir*='z')

Wrap a 2D line so that it draws as a 3D line

wrap_patch(*patch*, *zs*, *dir*='z')

wrap_text(*text*, *zs*, *dir*='z')

zalpha(*colors*, *zs*)

Modify the alphas of the color list according to depth

4.2 Class Line2DCollectionW

matplotlib.art3d.Wrap2D
└ **Line2DCollectionW**

4.2.1 Methods

__init__(*self*, *inst*, *z*=0, *dir*='z')
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.3 Class Line2DW

matplotlib.art3d.Wrap2D
└ **Line2DW**

4.3.1 Methods

__init__(*self*, *inst*, *z*=0, *dir*='z')
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.4 Class Line3D

matplotlib.artist.Artist
　　matplotlib.lines.Line2D
　　　　Line3D

Make a 2D line pretend to be 3D

4.4.1 Methods

__init__(*self*, *xs*, *ys*, *zs*, **args*, ***kwargs*)
Overrides: matplotlib.lines.Line2D.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.lines.Line2D.draw

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Line2D: get_aa, get_antialiased, get_c, get_color, get_dash_capstyle, get_dash_joinstyle, get_linestyle, get_linewidth, get_ls, get_lw, get_marker, get_markeredgecolor, get_markeredgewidth, get_markerfacecolor, get_markersize, get_mec, get_mew, get_mfc, get_ms, get_solid_capstyle, get_solid_joinstyle, get_window_extent, get_xdata, get_ydata, is_dashed, pick, recache, set_aa, set_antialiased, set_axes, set_c, set_color, set_dash_capstyle, set_dash_joinstyle, set_dashes, set_data, set_linestyle, set_linewidth, set_ls, set_lw, set_marker, set_markeredgecolor, set_markeredgewidth, set_markerfacecolor, set_markersize, set_mec, set_mew, set_mfc, set_ms, set_solid_capstyle, set_solid_joinstyle, set_xdata, set_ydata, update_from

4.4.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Line2D: filled_markers *(p. 301)*, validCap *(p. 301)*, validJoin *(p. 301)*, zorder *(p. 301)*	

4.5 Class Line3DCollection

matplotlib.art3d.Wrap2D
matplotlib.art3d.Line3DCollectionW
　　Line3DCollection

4.5.1 Methods

__init__(*self*, *segments*, **args*, ***kwargs*)
Overrides: matplotlib.art3d.Line3DCollectionW.__init__

Inherited from Line3DCollectionW: draw3d
Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.6 Class Line3DCollectionW

matplotlib.art3d.Wrap2D
Line3DCollectionW

Known Subclasses: Line3DCollection

4.6.1 Methods

__init__(*self*, *inst*, *segments*)
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.7 Class oLine3DCollection

matplotlib.artist.Artist
matplotlib.collections.Collection
matplotlib.cm.ScalarMappable
matplotlib.collections.LineCollection
oLine3DCollection

4.7.1 Methods

__init__(*self*, *segments*, **args*, ***kwargs*)
Overrides: matplotlib.collections.LineCollection.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.collections.LineCollection.draw

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pick, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from LineCollection: color, get_colors, get_dashes, get_linestyle, get_linewidth, get_transoffset, get_verts, set_alpha, set_color, set_linestyle, set_linewidth, set_segments, set_verts, update_scalarmappable

4.7.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from LineCollection: zorder *(p. 227)*	

4.8 Class oText3D

matplotlib.artist.Artist
  matplotlib.text.Text
    oText3D

4.8.1 Methods

__init__(*self*, *x*=`0`, *y*=`0`, *z*=`0`, *text*=`''`, *dir*=`'z'`, **args*, ***kwargs*)
Overrides: matplotlib.text.Text.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.text.Text.draw

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Text: get_color, get_font_properties, get_fontname, get_fontsize, get_fontstyle, get_fontweight, get_ha, get_horizontalalignment, get_name, get_position, get_prop_tup, get_rotation, get_rotation_matrix, get_size, get_style, get_text, get_va, get_verticalalignment, get_weight, get_window_extent, is_math_text, pick, set_backgroundcolor, set_bbox, set_color, set_family, set_fontname, set_fontproperties, set_fontsize, set_fontstyle, set_fontweight, set_ha, set_horizontalalignment, set_ma, set_multialignment, set_name, set_position, set_rotation, set_size, set_style, set_text, set_va, set_variant, set_verticalalignment, set_weight, set_x, set_y, update_from

4.8.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Text: zorder *(p. 511)*	

4.9 Class Patch3D

matplotlib.art3d.Wrap2D ─┐
Patch3D

4.9.1 Methods

__init__(*self*, *inst*, *zs*, *dir*='z')
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.10 Class Patch3DCollectionW

matplotlib.art3d.Wrap2D ─┐
Patch3DCollectionW

4.10.1 Methods

__init__(*self*, *inst*, *zs*, *dir*='z')
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.11 Class Poly3DCollection

matplotlib.art3d.Wrap2D ─┐
Poly3DCollection

4.11.1 Methods

__init__(*self*, *segments*, **args*, ***kwargs*)
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

get_vector(*self*)

optimise points for projection

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.12 Class Poly3DCollectionW

matplotlib.art3d.Wrap2D
└ **Poly3DCollectionW**

4.12.1 Methods

__init__(*self*, *inst*, *zs*=`None`, *dir*=`'z'`)
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.13 Class Text3D

matplotlib.art3d.Wrap2D
└ matplotlib.art3d.Text3DW
  └ **Text3D**

4.13.1 Methods

__init__(*self*, *x*=`0`, *y*=`0`, *z*=`0`, *text*=`''`, *dir*=`'z'`)
Overrides: matplotlib.art3d.Text3DW.__init__

Inherited from Text3DW: draw3d
Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.14 Class Text3DW

matplotlib.art3d.Wrap2D ─┐
Text3DW

Known Subclasses: Text3D

Wrap a 2D text object and make it look vaguely 3D

4.14.1 Methods

__init__(*self*, *inst*, *z*=0, *dir*='z')
Overrides: matplotlib.art3d.Wrap2D.__init__

draw3d(*self*, *renderer*)
Overrides: matplotlib.art3d.Wrap2D.draw3d

Inherited from Wrap2D: __getattr__, __setattr__, call_draw3d, remember

4.15 Class Wrap2D

Known Subclasses: Line2DCollectionW, Line2DW, Line3DCollectionW, Patch3D, Patch3DCollectionW, Poly3DCollection, Poly3DCollectionW, Text3DW

Wrapper which wraps a 2D object and makes it 3D

Artists are normally rendered by calling the draw method, this class causes call_draw3d to be called instead. This in turn calls draw3d which should play with the 2D coordinates and eventually call the original self.draw method through self.orig_draw.

overrides the draw method with draw3d remembers the original draw method of the wrapped 2d instance

4.15.1 Methods

__init__(*self*, *inst2d*)

__getattr__(*self*, *k*)

__setattr__(*self*, *k*, *v*)

call_draw3d(*self*, *renderer*)

draw3d(*self*, *renderer*)

remember(*self*, **attrs*)

Remember some attributes in the wrapped class

5 Module matplotlib.artist

5.1 Functions

get(*o*, **args*, ***kwargs*)

```
Return the value of handle property s

h is an instance of a class, eg a Line2D or an Axes or Text.
if s is 'somename', this function returns

  o.get_somename()

getp can be used to query all the gettable properties with getp(o)
Many properties have aliases for shorter typing, eg 'lw' is an
alias for 'linewidth'.  In the output, aliases and full property
names will be listed as

  property or  alias = value

eg

  linewidth or lw = 2
```

getp(*o*, **args*)

```
Return the value of handle property s

h is an instance of a class, eg a Line2D or an Axes or Text.
if s is 'somename', this function returns

  o.get_somename()

getp can be used to query all the gettable properties with getp(o)
Many properties have aliases for shorter typing, eg 'lw' is an
alias for 'linewidth'.  In the output, aliases and full property
names will be listed as

  property or  alias = value

eg

  linewidth or lw = 2
```

kwdoc(*a*)

setp(*h*, **args*, ***kwargs*)

matplotlib supports the use of setp ("set property") and getp to set and get object properties, as well as to do introspection on the object For example, to set the linestyle of a line to be dashed, you can do

```
>>> line, = plot([1,2,3])
>>> setp(line, linestyle='--')
```

If you want to know the valid types of arguments, you can provide the name of the property you want to set without a value

```
>>> setp(line, 'linestyle')
    linestyle: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' ]
```

If you want to see all the properties that can be set, and their possible values, you can do

```
>>> setp(line)
    ... long output listing omitted'
```

setp operates on a single instance or a list of instances. If you are in query mode introspecting the possible values, only the first instance in the sequence is used. When actually setting values, all the instances will be set. Eg, suppose you have a list of two lines, the following will make both lines thicker and red

```
>>> x = arange(0,1.0,0.01)
>>> y1 = sin(2*pi*x)
>>> y2 = sin(4*pi*x)
>>> lines = plot(x, y1, x, y2)
>>> setp(lines, linewidth=2, color='r')
```

setp works with the matlab(TM) style string/value pairs or with python kwargs. For example, the following are equivalent

```
>>> setp(lines, 'linewidth', 2, 'color', r')  # matlab style
>>> setp(lines, linewidth=2, color='r')       # python style
```

5.2 Class Artist

Known Subclasses: Axes, AxesImage, Axis, Collection, Figure, FigureImage, Legend, Line2D, Patch, QuiverKey, Table, Text, Tick

Abstract base class for someone who renders into a FigureCanvas

5.2.1 Methods

__init__(*self*)

add_callback(*self*, *func*)

convert_xunits(*self*, *x*)

for artists in an axes, if the xaxis as units support, convert x using xaxis unit type

convert_yunits(*self*, *y*)

for artists in an axes, if the yaxis as units support, convert y using yaxis unit type

draw(*self*, *renderer*, **args*, ***kwargs*)

Derived classes drawing method

get_alpha(*self*)

Return the alpha value used for blending - not supported on all backends

get_animated(*self*)

return the artist's animated state

get_axes(*self*)

return the axes instance the artist resides in, or None

get_clip_box(*self*)

Return artist clipbox

get_clip_on(*self*)

Return whether artist uses clipping

get_clip_path(*self*)

Return artist clip path

get_figure(*self*)

return the figure instance

get_label(*self*)

get_picker(*self*)

return the Pickeration instance used by this artist

get_transform(*self*)

return the Transformation instance used by this artist

get_visible(*self*)

return the artist's visiblity

get_zorder(*self*)

have_units(*self*)

return True if units are set on the x or y axes

is_figure_set(*self*)

is_transform_set(*self*)

Artist has transform explicity let

pchanged(*self*)

fire event when property changed

pick(*self*, *mouseevent*)

the user picked location x,y; if this Artist is within picker "pick epsilon" of x,y fire off a pick event

pickable(*self*)

return True if self is pickable

remove_callback(*self*, *oid*)

set(*self*, ***kwargs*)

A tkstyle set command, pass kwargs to set properties

set_alpha(*self*, *alpha*)

Set the alpha value used for blending - not supported on all backends
ACCEPTS: float

set_animated(*self*, *b*)

set the artist's animation state
ACCEPTS: [True | False]

set_axes(*self*, *axes*)

set the axes instance the artist resides in, if any
ACCEPTS: an axes instance

set_clip_box(*self*, *clipbox*)

Set the artist's clip Bbox
ACCEPTS: a matplotlib.transform.Bbox instance

set_clip_on(*self*, *b*)

Set whether artist uses clipping
ACCEPTS: [True | False]

set_clip_path(*self*, *path*)

Set the artist's clip path
ACCEPTS: an agg.path_storage instance

set_figure(*self*, *fig*)

Set the figure instance the artist belong to
ACCEPTS: a matplotlib.figure.Figure instance

set_label(*self*, *s*)

Set the line label to s for auto legend
ACCEPTS: any string

set_lod(*self*, *on*)

Set Level of Detail on or off. If on, the artists may examine things like the pixel width of the axes and draw a subset of their contents accordingly
ACCEPTS: [True | False]

set_picker(*self*, *picker*)

```
set the epsilon for picking used by this artist

picker can be one of the following:

  None -  picking is disabled for this artist (default)

  boolean - if True then picking will be enabled and the
    artist will fire a pick event if the mouse event is over
    the artist

  float - if picker is a number it is interpreted as an
    epsilon tolerance in points and the the artist will fire
    off an event if it's data is within epsilon of the mouse
    event.  For some artists like lines and patch collections,
    the artist may provide additional data to the pick event
    that is generated, eg the indices of the data within
    epsilon of the pick event

  function - if picker is callable, it is a user supplied
    function which determines whether the artist is hit by the
    mouse event.

      hit, props = picker(artist, mouseevent)

    to determine the hit test.  if the mouse event is over the
    artist, return hit=True and props is a dictionary of
    properties you want added to the PickEvent attributes

ACCEPTS: [None|float|boolean|callable]
```

set_transform(*self*, *t*)

set the Transformation instance used by this artist
ACCEPTS: a matplotlib.transform transformation instance

set_visible(*self*, *b*)

set the artist's visiblity
ACCEPTS: [True | False]

set_zorder(*self*, *level*)

Set the zorder for the artist
ACCEPTS: any number

update(*self*, *props*)

update_from(*self, other*)

copy properties from other to self

5.2.2 Class Variables

Name	Description
aname	**Value:** `'Artist'` *(type=`str`)*
zorder	**Value:** `0` *(type=`int`)*

5.3 Class ArtistInspector

A helper class to insect an Artist and return information about it's settable properties and their current values

5.3.1 Methods

__init__(*self, o*)

Initialize the artist inspector with an artist or sequence of artists. Id a sequence is used, we assume it is a homogeneous sequence (all Artists are of the same type) and it is your responsibility to make sure this is so.

aliased_name(*self, s*)

return 'PROPNAME or alias' if s has an alias, else return PROPNAME.
Eg for the line markerfacecolor property, which has an alias, return 'markerfacecolor or mfc' and for the transform property, which does not, return 'transform'

get_aliases(*self*)

```
get a dict mapping fullname -> alias for each alias in o.
Eg for lines: {'markerfacecolor': 'mfc',
               'linewidth'      : 'lw',
               }
```

get_setters(*self*)

Get the attribute strings with setters for object h. Eg, for a line, return ['markerfacecolor', 'linewidth', ...]

get_valid_values(*self, attr*)

get the legal arguments for the setter associated with attr
This is done by querying the doc string of the function set_attr for a line that begins with ACCEPTS:
Eg, for a line linestyle, return ['-' | '–' | '-.' | ':' | 'steps' | 'None']

is_alias(*self*, *o*)

return true if method object o is an alias for another function

pprint_getters(*self*)

return the getters and actual values as list of strings'

pprint_setters(*self*, *prop*=`None`, *leadingspace*=`2`)

if prop is None, return a list of strings of all settable properies and their valid values
if prop is not None, it is a valid property name and that property will be returned as a string of property : valid values

6 Module matplotlib.axes

6.1 Functions

delete_masked_points(**args*)

Find all masked points in a set of arguments, and return the arguments with only the unmasked points remaining.
The overall mask is calculated from any masks that are present. If a mask is found, any argument that does not have the same dimensions is left unchanged; therefore the argument list may include arguments that can take string or array values, for example.
Array arguments must have the same length; masked arguments must be one-dimensional.
Written as a helper for scatter, but may be more generally useful.

makeValue(*v*)

6.2 Class Axes

matplotlib.artist.Artist
└ **Axes**

Known Subclasses: Axes3DI, PolarAxes, Subplot

The Axes contains most of the figure elements: Axis, Tick, Line2D, Text, Polygon etc, and sets the coordinate system

6.2.1 Methods

__init__(*self*, *fig*, *rect*, *axisbg*=`None`, *frameon*=`True`, *sharex*=`None`, *sharey*=`None`, *label*=`''`, ***kwargs*)

Build an Axes instance in Figure with rect=[left, bottom, width,height in Figure coords
adjustable: ['box' | 'datalim'] alpha: the alpha transparency anchor: ['C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W'] aspect: ['auto' | 'equal' | aspect_ratio] autoscale_on: boolean - whether or not to autoscale the viewlim axis_bgcolor: any matplotlib color - see help(colors) axisbelow: draw the grids and ticks below the other artists cursor_props: a (float, color) tuple figure: a Figure instance frame_on: a boolean - draw the axes frame label: the axes label navigate: True|False navigate_mode: the navigation toolbar button status: 'PAN', 'ZOOM', or None position: [left, bottom, width,height in Figure coords sharex : an Axes instance to share the x-axis with sharey : an Axes instance to share the y-axis with title: the title string visible: a boolean - whether the axes is visible xlabel: the xlabel xlim: (xmin, xmax) view limits xscale: ['log' | 'linear'] xticklabels: sequence of strings xticks: sequence of floats ylabel: the ylabel strings ylim: (ymin, ymax) view limits yscale: ['log' | 'linear'] yticklabels: sequence of strings yticks: sequence of floats

Overrides: matplotlib.artist.Artist.__init__

acorr(*self*, *x*, ***kwargs*)

```
ACORR(x, normed=False, detrend=detrend_none, usevlines=False,
      maxlags=None, **kwargs)
Plot the autocorrelation of x.  If normed=True, normalize the
data but the autocorrelation at 0-th lag.  x is detrended by
the detrend callable (default no normalization.
data are plotted as plot(lags, c, **kwargs)
return value is lags, c, line where lags are a length
2*maxlags+1 lag vector, c is the 2*maxlags+1 auto correlation
vector, and line is a Line2D instance returned by plot.  The
default linestyle is None and the default marker is 'o',
though these can be overridden with keyword args.  The cross
correlation is performed with numerix cross_correlate with
mode=2.
If usevlines is True, Axes.vlines rather than Axes.plot is used
to draw vertical lines from the origin to the acorr.
Otherwise the plotstyle is determined by the kwargs, which are
Line2D properties.  If usevlines, the return value is lags, c,
linecol, b where linecol is the LineCollection and b is the x-axis
if usevlines=True, kwargs are passed onto Axes.vlines
if usevlines=False, kwargs are passed onto Axes.plot
maxlags is a positive integer detailing the number of lags to show.
The default value of None will return all (2*len(x)-1) lags.
See the respective function for documentation on valid kwargs
```

add_artist(*self*, *a*)

Add any artist to the axes

add_collection(*self*, *collection*, *autolim*=`False`)

add a Collection instance to Axes

add_line(*self*, *line*)

Add a line to the list of plot lines

add_patch(*self*, *p*)

Add a patch to the list of Axes patches; the clipbox will be set to the Axes clipping box. If the transform is not set, it wil be set to self.transData.

add_table(*self*, *tab*)

Add a table instance to the list of axes tables

annotate(*self*, **args*, ***kwargs*)

```
annotate(self, s, xy, textloc,
 xycoords='data', textcoords='data',
 lineprops=None,
 markerprops=None
 **props)
alpha: float
animated: [True | False]
axes: an axes instance
backgroundcolor: any matplotlib color
bbox: rectangle prop dict plus key 'pad' which is a pad in points
clip_box: a matplotlib.transform.Bbox instance
clip_on: [True | False]
clip_path: an agg.path_storage instance
color: any matplotlib color
family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
figure: a matplotlib.figure.Figure instance
fontproperties: a matplotlib.font_manager.FontProperties instance
horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
label: any string
lod: [True | False]
multialignment: ['left' | 'right' | 'center' ]
name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
picker: [None|float|boolean|callable]
position: (x,y)
rotation: [ angle in degrees 'vertical' | 'horizontal'
size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
style or fontstyle: [ 'normal' | 'italic' | 'oblique']
text: string or anything printable with '%s' conversion
transform: a matplotlib.transform transformation instance
variant: [ 'normal' | 'small-caps' ]
verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
visible: [True | False]
weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
x: float
y: float
zorder: any number
```

apply_aspect(*self*, *data_ratio*=None)

Use self._aspect and self._adjustable to modify the axes box or the view limits. The data_ratio kwarg is set to 1 for polar axes. It is used only when _adjustable is 'box'.

arrow(*self*, *x*, *y*, *dx*, *dy*, ***kwargs*)

```
Draws arrow on specified axis from (x,y) to (x+dx,y+dy).
Optional kwargs control the arrow properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

autoscale_view(*self*, *tight*=`False`, *scalex*=`True`, *scaley*=`True`)

autoscale the view limits using the data limits. You can selectively autoscale only a single axis, eg, the xaxis by setting scaley to False. The autoscaling preserves any axis direction reversal that has already been done.

axhline(*self*, *y*=0, *xmin*=0, *xmax*=1, ***kwargs*)

```
AXHLINE(y=0, xmin=0, xmax=1, **kwargs)
Axis Horizontal Line
Draw a horizontal line at y from xmin to xmax.  With the default
values of xmin=0 and xmax=1, this line will always span the horizontal
extent of the axes, regardless of the xlim settings, even if you
change them, eg with the xlim command.  That is, the horizontal extent
is in axes coords: 0=left, 0.5=middle, 1.0=right but the y location is
in data coordinates.
Return value is the Line2D instance.  kwargs are the same as kwargs to
plot, and can be used to control the line properties.  Eg
  # draw a thick red hline at y=0 that spans the xrange
  axhline(linewidth=4, color='r')
  # draw a default hline at y=1 that spans the xrange
  axhline(y=1)
  # draw a default hline at y=.5 that spans the the middle half of
  # the xrange
  axhline(y=.5, xmin=0.25, xmax=0.75)
Valid kwargs are Line2D properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

axhspan(*self*, *ymin*, *ymax*, *xmin*=0, *xmax*=1, ***kwargs*)

```
AXHSPAN(ymin, ymax, xmin=0, xmax=1, **kwargs)
Axis Horizontal Span.  ycoords are in data units and x
coords are in axes (relative 0-1) units
Draw a horizontal span (regtangle) from ymin to ymax.  With the
default values of xmin=0 and xmax=1, this always span the xrange,
regardless of the xlim settings, even if you change them, eg with the
xlim command.  That is, the horizontal extent is in axes coords:
0=left, 0.5=middle, 1.0=right but the y location is in data
coordinates.
kwargs are the kwargs to Patch, eg
  antialiased, aa
  linewidth,   lw
  edgecolor,   ec
  facecolor,   fc
the terms on the right are aliases
Return value is the patches.Polygon instance.
    #draws a gray rectangle from y=0.25-0.75 that spans the horizontal
    #extent of the axes
    axhspan(0.25, 0.75, facecolor='0.5', alpha=0.5)
Valid kwargs are Polygon properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

axis(*self*, **v*, ***kwargs*)

Convenience method for manipulating the x and y view limits and the aspect ratio of the plot.
kwargs are passed on to set_xlim and set_ylim – see their docstrings for details

axvline(*self*, *x*=0, *ymin*=0, *ymax*=1, ***kwargs*)

```
AXVLINE(x=0, ymin=0, ymax=1, **kwargs)
Axis Vertical Line
Draw a vertical line at x from ymin to ymax.  With the default values
of ymin=0 and ymax=1, this line will always span the vertical extent
of the axes, regardless of the xlim settings, even if you change them,
eg with the xlim command.  That is, the vertical extent is in axes
coords: 0=bottom, 0.5=middle, 1.0=top but the x location is in data
coordinates.
Return value is the Line2D instance.  kwargs are the same as
kwargs to plot, and can be used to control the line properties.  Eg
    # draw a thick red vline at x=0 that spans the yrange
    l = axvline(linewidth=4, color='r')
    # draw a default vline at x=1 that spans the yrange
    l = axvline(x=1)
    # draw a default vline at x=.5 that spans the the middle half of
    # the yrange
    axvline(x=.5, ymin=0.25, ymax=0.75)
Valid kwargs are Line2D properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

axvspan(*self*, *xmin*, *xmax*, *ymin*=0, *ymax*=1, ***kwargs*)

```
AXVSPAN(xmin, xmax, ymin=0, ymax=1, **kwargs)
axvspan : Axis Vertical Span.  xcoords are in data units and y coords
are in axes (relative 0-1) units
Draw a vertical span (regtangle) from xmin to xmax.  With the default
values of ymin=0 and ymax=1, this always span the yrange, regardless
of the ylim settings, even if you change them, eg with the ylim
command.  That is, the vertical extent is in axes coords: 0=bottom,
0.5=middle, 1.0=top but the y location is in data coordinates.
kwargs are the kwargs to Patch, eg
  antialiased, aa
  linewidth,   lw
  edgecolor,   ec
  facecolor,   fc
the terms on the right are aliases
return value is the patches.Polygon instance.
    # draw a vertical green translucent rectangle from x=1.25 to 1.55 that
    # spans the yrange of the axes
    axvspan(1.25, 1.55, facecolor='g', alpha=0.5)
Valid kwargs are Polygon properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

bar(*self*, *left*, *height*, *width*=0.80000000000000004, *bottom*=None, *color*=None, *edgecolor*=None, *linewidth*=None, *yerr*=None, *xerr*=None, *ecolor*=None, *capsize*=3, *align*='edge', *orientation*='vertical', *log*=False, ***kwargs*)

```
BAR(left, height, width=0.8, bottom=0,
    color=None, edgecolor=None, linewidth=None,
    yerr=None, xerr=None, ecolor=None, capsize=3,
    align='edge', orientation='vertical', log=False)
Make a bar plot with rectangles bounded by
  left, left+width, bottom, bottom+height
        (left, right, bottom and top edges)
left, height, width, and bottom can be either scalars or sequences
Return value is a list of Rectangle patch instances
    left - the x coordinates of the left sides of the bars
    height - the heights of the bars
Optional arguments:
    width - the widths of the bars
    bottom - the y coordinates of the bottom edges of the bars
    color - the colors of the bars
    edgecolor - the colors of the bar edges
    linewidth - width of bar edges; None means use default
        linewidth; 0 means don't draw edges.
    xerr and yerr, if not None, will be used to generate errorbars
    on the bar chart
    ecolor specifies the color of any errorbar
    capsize (default 3) determines the length in points of the error
    bar caps
    align = 'edge' (default) | 'center'
    orientation = 'vertical' | 'horizontal'
    log = False | True - False (default) leaves the orientation
            axis as-is; True sets it to log scale
For vertical bars, align='edge' aligns bars by their left edges in
left, while 'center' interprets these values as the x coordinates of
the bar centers. For horizontal bars, 'edge' aligns bars by their
bottom edges in bottom, while 'center' interprets these values as the
y coordinates of the bar centers.
The optional arguments color, edgecolor, linewidth, xerr, and yerr can
be either scalars or sequences of length equal to the number of bars.
This enables you to use bar as the basis for stacked bar charts, or
candlestick plots.
Optional kwargs:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
```

barh(*self*, *bottom*, *width*, *height*=0.80000000000000004, *left*=None, ***kwargs*)

```
BARH(bottom, width, height=0.8, left=0, **kwargs)
Make a horizontal bar plot with rectangles bounded by
  left, left+width, bottom, bottom+height
        (left, right, bottom and top edges)
bottom, width, height, and left can be either scalars or sequences
Return value is a list of Rectangle patch instances
    bottom - the vertical positions of the bottom edges of the bars
    width - the lengths of the bars
Optional arguments:
    height - the heights (thicknesses) of the bars
    left - the x coordinates of the left edges of the bars
    color - the colors of the bars
    edgecolor - the colors of the bar edges
    linewidth - width of bar edges; None means use default
        linewidth; 0 means don't draw edges.
    xerr and yerr, if not None, will be used to generate errorbars
    on the bar chart
    ecolor specifies the color of any errorbar
    capsize (default 3) determines the length in points of the error
    bar caps
    align = 'edge' (default) | 'center'
    log = False | True - False (default) leaves the horizontal
            axis as-is; True sets it to log scale
Setting align='edge' aligns bars by their bottom edges in bottom,
while 'center' interprets these values as the y coordinates of the bar
centers.
The optional arguments color, edgecolor, linewidth, xerr, and yerr can
be either scalars or sequences of length equal to the number of bars.
This enables you to use barh as the basis for stacked bar charts, or
candlestick plots.
Optional kwargs:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

boxplot(*self*, *x*, *notch*=0, *sym*='b+', *vert*=1, *whis*=1.5, *positions*=None, *widths*=None)

```
boxplot(x, notch=0, sym='+', vert=1, whis=1.5,
        positions=None, widths=None)

Make a box and whisker plot for each column of x or
each vector in sequence x.
The box extends from the lower to upper quartile values
of the data, with a line at the median.  The whiskers
extend from the box to show the range of the data.  Flier
points are those past the end of the whiskers.

notch = 0 (default) produces a rectangular box plot.
notch = 1 will produce a notched box plot

sym (default 'b+') is the default symbol for flier points.
Enter an empty string ('') if you don't want to show fliers.

vert = 1 (default) makes the boxes vertical.
vert = 0 makes horizontal boxes.  This seems goofy, but
that's how Matlab did it.

whis (default 1.5) defines the length of the whiskers as
a function of the inner quartile range.  They extend to the
most extreme data point within ( whis*(75%-25%) ) data range.

positions (default 1,2,...,n) sets the horizontal positions of
the boxes. The ticks and limits are automatically set to match
the positions.

widths is either a scalar or a vector and sets the width of
each box. The default is 0.5, or 0.15*(distance between extreme
positions) if that is smaller.

x is an array or a sequence of vectors.

Returns a list of the lines added.
```

broken_barh(*self*, *xranges*, *yrange*, ***kwargs*)

```
A collection of horizontal bars spanning yrange with a sequence of
xranges
xranges : sequence of (xmin, xwidth)
yrange  : (ymin, ywidth)
kwargs are collections.BrokenBarHCollection properties
        alpha: float
        animated: [True | False]
        array: unknown
        axes: an axes instance
        clim: a length 2 sequence of floats
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cmap: a colormap
        color: matplotlib color arg or sequence of rgba tuples
        colorbar: unknown
        edgecolor: matplotlib color arg or sequence of rgba tuples
        facecolor: matplotlib color arg or sequence of rgba tuples
        figure: a matplotlib.figure.Figure instance
        label: any string
        linewidth: float or sequence of floats
        lod: [True | False]
        norm: unknown
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
these can either be a single argument, ie facecolors='black'
or a sequence of arguments for the various bars, ie
facecolors='black', 'red', 'green'
```

cla(*self*)

Clear the current axes

clabel(*self*, *CS*, **args*, ***kwargs*)

```
clabel(CS, **kwargs) - add labels to line contours in CS,
       where CS is a ContourSet object returned by contour.

clabel(CS, V, **kwargs) - only label contours listed in V

keyword arguments:

* fontsize = None: as described in http://matplotlib.sf.net/fonts.html

* colors = None:

   - a tuple of matplotlib color args (string, float, rgb, etc),
     different labels will be plotted in different colors in the order
     specified

   - one string color, e.g. colors = 'r' or colors = 'red', all labels
     will be plotted in this color

   - if colors == None, the color of each label matches the color
     of the corresponding contour

* inline = True: controls whether the underlying contour is removed
             (inline = True) or not (False)

* fmt = '%1.3f': a format string for the label
```

clear(*self*)

clear the axes

cohere(*self*, *x*, *y*, *NFFT*=256, *Fs*=2, *detrend*=`<function detrend_none at 0x8413b1c>`, *window*=`<function window_hanning at 0x8413a04>`, *noverlap*=0, ***kwargs*)

```
COHERE(x, y, NFFT=256, Fs=2, detrend=detrend_none,
      window=window_hanning, noverlap=0, **kwargs)
cohere the coherence between x and y.  Coherence is the normalized
cross spectral density
  Cxy = |Pxy|^2/(Pxx*Pyy)
The return value is (Cxy, f), where f are the frequencies of the
coherence vector.
See the PSD help for a description of the optional parameters.
kwargs are applied to the lines
Returns the tuple Cxy, freqs
Refs: Bendat & Piersol -- Random Data: Analysis and Measurement
  Procedures, John Wiley & Sons (1986)
kwargs control the Line2D properties of the coherence plot:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

connect(*self*, *s*, *func*)

```
Register observers to be notified when certain events occur.  Register
with callback functions with the following signatures.  The function
has the following signature

    func(ax)  # where ax is the instance making the callback.

The following events can be connected to:

  'xlim_changed','ylim_changed'

The connection id is is returned - you can use this with
disconnect to disconnect from the axes event
```

contour(*self*, **args*, ***kwargs*)

```
contour and contourf draw contour lines and filled contours,
respectively.  Except as noted, function signatures and return
values are the same for both versions.

contourf differs from the Matlab (TM) version in that it does not
    draw the polygon edges, because the contouring engine yields
    simply connected regions with branch cuts.  To draw the edges,
    add line contours with calls to contour.


Function signatures

contour(Z) - make a contour plot of an array Z. The level
         values are chosen automatically.

contour(X,Y,Z) - X,Y specify the (x,y) coordinates of the surface

contour(Z,N) and contour(X,Y,Z,N) - contour N automatically-chosen
         levels.

contour(Z,V) and contour(X,Y,Z,V) - draw len(V) contour lines,
         at the values specified in sequence V

contourf(..., V) - fill the (len(V)-1) regions between the
         values in V

contour(Z, **kwargs) - Use keyword args to control colors, linewidth,
            origin, cmap ... see below

X, Y, and Z must be arrays with the same dimensions.
Z may be a masked array, but filled contouring may not handle
           internal masked regions correctly.

C = contour(...) returns a ContourSet object.


Optional keyword args are shown with their defaults below (you must
use kwargs for these):

    * colors = None; or one of the following:
      - a tuple of matplotlib color args (string, float, rgb, etc),
      different levels will be plotted in different colors in the order
      specified

      -  one string color, e.g. colors = 'r' or colors = 'red', all levels
      will be plotted in this color

      - if colors == None, the colormap specified by cmap will be used

    * alpha=1.0 : the alpha blending value

    * cmap = None: a cm Colormap instance from matplotlib.cm.
      - if cmap == None and colors == None, a default Colormap is used.

    * norm = None: a matplotlib.colors.Normalize instance for
      scaling data values to colors.
```

contourf(*self*, **args*, ***kwargs*)

```
contour and contourf draw contour lines and filled contours,
respectively.  Except as noted, function signatures and return
values are the same for both versions.

contourf differs from the Matlab (TM) version in that it does not
    draw the polygon edges, because the contouring engine yields
    simply connected regions with branch cuts.  To draw the edges,
    add line contours with calls to contour.


Function signatures

contour(Z) - make a contour plot of an array Z. The level
         values are chosen automatically.

contour(X,Y,Z) - X,Y specify the (x,y) coordinates of the surface

contour(Z,N) and contour(X,Y,Z,N) - contour N automatically-chosen
         levels.

contour(Z,V) and contour(X,Y,Z,V) - draw len(V) contour lines,
         at the values specified in sequence V

contourf(..., V) - fill the (len(V)-1) regions between the
         values in V

contour(Z, **kwargs) - Use keyword args to control colors, linewidth,
            origin, cmap ... see below

X, Y, and Z must be arrays with the same dimensions.
Z may be a masked array, but filled contouring may not handle
           internal masked regions correctly.

C = contour(...) returns a ContourSet object.


Optional keyword args are shown with their defaults below (you must
use kwargs for these):

    * colors = None; or one of the following:
      - a tuple of matplotlib color args (string, float, rgb, etc),
      different levels will be plotted in different colors in the order
      specified

      -  one string color, e.g. colors = 'r' or colors = 'red', all levels
      will be plotted in this color

      - if colors == None, the colormap specified by cmap will be used

    * alpha=1.0 : the alpha blending value

    * cmap = None: a cm Colormap instance from matplotlib.cm.
      - if cmap == None and colors == None, a default Colormap is used.

    * norm = None: a matplotlib.colors.Normalize instance for
      scaling data values to colors.
```

csd(*self*, *x*, *y*, *NFFT*=256, *Fs*=2, *detrend*=<function detrend_none at 0x8413b1c>, *window*=<function window_hanning at 0x8413a04>, *noverlap*=0, ***kwargs*)

```
CSD(x, y, NFFT=256, Fs=2, detrend=detrend_none,
    window=window_hanning, noverlap=0, **kwargs)
The cross spectral density Pxy by Welches average periodogram method.
The vectors x and y are divided into NFFT length segments.  Each
segment is detrended by function detrend and windowed by function
window.  The product of the direct FFTs of x and y are averaged over
each segment to compute Pxy, with a scaling to correct for power loss
due to windowing.
See the PSD help for a description of the optional parameters.
Returns the tuple Pxy, freqs.  Pxy is the cross spectrum (complex
valued), and 10*log10(|Pxy|) is plotted
Refs:
  Bendat & Piersol -- Random Data: Analysis and Measurement
    Procedures, John Wiley & Sons (1986)
kwargs control the Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

disconnect(*self*, *cid*)

disconnect from the Axes event.

draw(*self*, *renderer*=`None`, *inframe*=`False`)

Draw everything (plot lines, axes, labels)

Overrides: matplotlib.artist.Artist.draw

draw_artist(*self*, *a*)

This method can only be used after an initial draw which caches the renderer. It is used to efficiently update Axes data (axis ticks, labels, etc are not updated)

errorbar(*self*, *x*, *y*, *yerr*=None, *xerr*=None, *fmt*='b-', *ecolor*=None, *capsize*=3, *barsabove*=False, ***kwargs*)

```
ERRORBAR(x, y, yerr=None, xerr=None,
         fmt='b-', ecolor=None, capsize=3, barsabove=False)
Plot x versus y with error deltas in yerr and xerr.
Vertical errorbars are plotted if yerr is not None
Horizontal errorbars are plotted if xerr is not None
xerr and yerr may be any of:
    a rank-0, Nx1 Numpy array  - symmetric errorbars +/- value
    an N-element list or tuple - symmetric errorbars +/- value
    a rank-1, Nx2 Numpy array  - asymmetric errorbars -column1/+column2
Alternatively, x, y, xerr, and yerr can all be scalars, which
plots a single error bar at x, y.
    fmt is the plot format symbol for y.  if fmt is None, just
    plot the errorbars with no line symbols.  This can be useful
    for creating a bar plot with errorbars
    ecolor is a matplotlib color arg which gives the color the
    errorbar lines; if None, use the marker color.
    capsize is the size of the error bar caps in points
    barsabove, if True, will plot the errorbars above the plot symbols
    - default is below
    kwargs are passed on to the plot command for the markers.
      So you can add additional key=value pairs to control the
      errorbar markers.  For example, this code makes big red
      squares with thick green edges
      >>> x,y,yerr = rand(3,10)
      >>> errorbar(x, y, yerr, marker='s',
                   mfc='red', mec='green', ms=20, mew=4)
     mfc, mec, ms and mew are aliases for the longer property
     names, markerfacecolor, markeredgecolor, markersize and
     markeredgewith.
valid kwargs for the marker properties are
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
```

fill(*self*, **args*, ***kwargs*)

```
FILL(*args, **kwargs)
plot filled polygons.  *args is a variable length argument, allowing
for multiple x,y pairs with an optional color format string; see plot
for details on the argument parsing.  For example, all of the
following are legal, assuming ax is an Axes instance:
  ax.fill(x,y)            # plot polygon with vertices at x,y
  ax.fill(x,y, 'b' )      # plot polygon with vertices at x,y in blue
An arbitrary number of x, y, color groups can be specified, as in
  ax.fill(x1, y1, 'g', x2, y2, 'r')
Return value is a list of patches that were added
The same color strings that plot supports are supported by the fill
format string.
kwargs control the Polygon properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

format_coord(*self*, *x*, *y*)

return a format string formatting the x, y coord

format_xdata(*self*, *x*)

Return x string formatted. This function will use the attribute self.fmt_xdata if it is callable, else will fall back on the xaxis major formatter

format_ydata(*self*, *y*)

Return y string formatted. This function will use the attribute self.fmt_ydata if it is callable, else will fall back on the yaxis major formatter

get_adjustable(*self*)

get_anchor(*self*)

get_aspect(*self*)

get_autoscale_on(*self*)

Get whether autoscaling is applied on plot commands

get_axis_bgcolor(*self*)

Return the axis background color

get_axisbelow(*self*)

Get whether axist below is true or not

get_child_artists(*self*)

Return a list of artists the axes contains. Deprecated

get_children(*self*)

return a list of child artists

get_cursor_props(*self*)

return the cursor props as a linewidth, color tuple where linewidth is a float and color is an RGBA tuple

get_frame(*self*)

Return the axes Rectangle frame

get_frame_on(*self*)

Get whether the axes rectangle patch is drawn

get_images(*self*)

return a list of Axes images contained by the Axes

get_legend(*self*)

Return the Legend instance, or None if no legend is defined

get_lines(*self*)

Return a list of lines contained by the Axes

get_navigate(*self*)

Get whether the axes responds to navigation commands

get_navigate_mode(*self*)

Get the navigation toolbar button status: 'PAN', 'ZOOM', or None

get_position(*self*, *original*=`False`)

Return the axes rectangle left, bottom, width, height

get_renderer_cache(*self*)

get_window_extent(*self*, **args*, ***kwargs*)

get the axes bounding box in display space; args and kwargs are empty

get_xaxis(*self*)

Return the XAxis instance

get_xgridlines(*self*)

Get the x grid lines as a list of Line2D instances

get_xlim(*self*)

Get the x axis range [xmin, xmax]

get_xscale(*self*)

return the xaxis scale string: log or linear

get_xticklabels(*self*)

Get the xtick labels as a list of Text instances

get_xticklines(*self*)

Get the xtick lines as a list of Line2D instances

get_xticks(*self*)

Return the x ticks as a list of locations

get_yaxis(*self*)

Return the YAxis instance

get_ygridlines(*self*)

Get the y grid lines as a list of Line2D instances

get_ylim(*self*)

Get the y axis range [ymin, ymax]

get_yscale(*self*)

return the yaxis scale string: log or linear

get_yticklabels(*self*)

Get the ytick labels as a list of Text instances

get_yticklines(*self*)

Get the ytick lines as a list of Line2D instances

get_yticks(*self*)

Return the y ticks as a list of locations

grid(*self*, *b*=None, ***kwargs*)

```
GRID(self, b=None, **kwargs)
Set the axes grids on or off; b is a boolean
if b is None and len(kwargs)==0, toggle the grid state.  if
kwargs are supplied, it is assumed that you want a grid and b
is thus set to True
kawrgs are used to set the grid line properties, eg
  ax.grid(color='r', linestyle='-', linewidth=2)
Valid Line2D kwargs are
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

has_data(*self*)

Return true if any artists have been added to axes.
This should not be used to determine whether the dataLim need to be updated, and may not actually be useful for anything.

hist(*self*, *x*, *bins*=10, *normed*=0, *bottom*=None, *align*='edge', *orientation*='vertical', *width*=None, *log*=False, ***kwargs*)

```
HIST(x, bins=10, normed=0, bottom=None,
     align='edge', orientation='vertical', width=None,
     log=False, **kwargs)
Compute the histogram of x.  bins is either an integer number of
bins or a sequence giving the bins.  x are the data to be binned.
The return values is (n, bins, patches)
If normed is true, the first element of the return tuple will
be the counts normalized to form a probability density, ie,
n/(len(x)*dbin).  In a probability density, the integral of
the histogram should be one (we assume equally spaced bins);
you can verify that with
  # trapezoidal integration of the probability density function
  from matplotlib.mlab import trapz
  pdf, bins, patches = ax.hist(...)
  print trapz(bins, pdf)
align = 'edge' | 'center'.  Interprets bins either as edge
or center values
orientation = 'horizontal' | 'vertical'.  If horizontal, barh
will be used and the "bottom" kwarg will be the left edges.
width: the width of the bars.  If None, automatically compute
the width.
log: if True, the histogram axis will be set to a log scale
kwargs are used to update the properties of the
hist Rectangles:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

hlines(*self*, *y*, *xmin*, *xmax*, *colors*=`'k'`, *linestyle*=`'solid'`, *label*=`''`, ***kwargs*)

HLINES(y, xmin, xmax, colors='k', linestyle='solid', **kwargs) plot horizontal lines at each y from xmin to xmax. xmin or xmax can be scalars or len(x) numpy arrays. If they are scalars, then the respective values are constant, else the widths of the lines are determined by xmin and xmax colors is a line collections color args, either a single color or a len(x) list of colors linestyle is one of solid|dashed|dashdot|dotted Returns the LineCollection that was added

hold(*self*, *b*=`None`)

```
HOLD(b=None)

Set the hold state.  If hold is None (default), toggle the
hold state.  Else set the hold state to boolean value b.

Eg
    hold()      # toggle hold
    hold(True)  # hold is on
    hold(False) # hold is off


When hold is True, subsequent plot commands will be added to
the current axes.  When hold is False, the current axes and
figure will be cleared on the next plot command
```

imshow(*self*, *X*, *cmap*=None, *norm*=None, *aspect*=None, *interpolation*=None, *alpha*=1.0, *vmin*=None, *vmax*=None, *origin*=None, *extent*=None, *shape*=None, *filternorm*=1, *filterrad*=4.0, *imlim*=None, ***kwargs*)

```
IMSHOW(X, cmap=None, norm=None, aspect=None, interpolation=None,
       alpha=1.0, vmin=None, vmax=None, origin=None, extent=None)

IMSHOW(X) - plot image X to current axes, resampling to scale to axes
            size (X may be numarray/Numeric array or PIL image)

IMSHOW(X, **kwargs) - Use keyword args to control image scaling,
colormapping etc. See below for details


Display the image in X to current axes.  X may be a float array, a
UInt8 array or a PIL image. If X is an array, X can have the following
shapes:

    MxN    : luminance (grayscale, float array only)

    MxNx3  : RGB (float or UInt8 array)

    MxNx4  : RGBA (float or UInt8 array)

The value for each component of MxNx3 and MxNx4 float arrays should be
in the range 0.0 to 1.0; MxN float arrays may be normalised.

A matplotlib.image.AxesImage instance is returned

The following kwargs are allowed:

  * cmap is a cm colormap instance, eg cm.jet.  If None, default to rc
    image.cmap value (Ignored when X has RGB(A) information)

  * aspect is one of: auto, equal, or a number.  If None, default to rc
    image.aspect value

  * interpolation is one of:

    'nearest', 'bilinear', 'bicubic', 'spline16', 'spline36',
    'hanning', 'hamming', 'hermite', 'kaiser', 'quadric',
    'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc',
    'lanczos', 'blackman'

    if interpolation is None, default to rc
    image.interpolation.  See also th the filternorm and
    filterrad parameters

  * norm is a matplotlib.colors.Normalize instance; default is
    normalization().  This scales luminance -> 0-1 (only used for an
    MxN float array).
```



```
  * vmin and vmax are used to scale a luminance image to 0-1.  If
    either is None, the min and max of the luminance values will be
    used.  Note if you pass a norm instance, the settings for vmin and
    vmax will be ignored.

  * alpha = 1.0 : the alpha blending value
```

in_axes(*self*, *xwin*, *ywin*)

return True is the point xwin, ywin (display coords) are in the Axes

ishold(*self*)

return the HOLD status of the axes

legend(*self*, **args*, ***kwargs*)

```
LEGEND(*args, **kwargs)

Place a legend on the current axes at location loc.  Labels are a
sequence of strings and loc can be a string or an integer specifying
the legend location

USAGE:

  Make a legend with existing lines

  >>> legend()

  legend by itself will try and build a legend using the label
  property of the lines/patches/collections.  You can set the label of
  a line by doing plot(x, y, label='my data') or line.set_label('my
  data'). If label is set to '_nolegend_', the item will not be shown
  in legend.

    # automatically generate the legend from labels
    legend( ('label1', 'label2', 'label3') )

    # Make a legend for a list of lines and labels
    legend( (line1, line2, line3), ('label1', 'label2', 'label3') )

    # Make a legend at a given location, using a location argument
    # legend( LABELS, LOC )  or
    # legend( LINES, LABELS, LOC )
    legend( ('label1', 'label2', 'label3'), loc='upper left')
    legend( (line1, line2, line3),  ('label1', 'label2', 'label3'), loc=2)

The location codes are

  'best' : 0,
  'upper right'  : 1, (default)
  'upper left'   : 2,
  'lower left'   : 3,
  'lower right'  : 4,
  'right'        : 5,
  'center left'  : 6,
  'center right' : 7,
  'lower center' : 8,
  'upper center' : 9,
  'center'       : 10,

If none of these are suitable, loc can be a 2-tuple giving x,y
in axes coords, ie,

  loc = 0, 1 is left top
  loc = 0.5, 0.5 is center, center

and so on.  The following kwargs are supported:

isaxes=True           # whether this is an axes legend
numpoints = 4         # the number of points in the legend line
prop = FontProperties(size='smaller')  # the font property
pad = 0.2             # the fractional whitespace inside the legend border
```

loglog(*self*, **args*, ***kwargs*)

```
LOGLOG(*args, **kwargs)
Make a loglog plot with log scaling on the a and y axis.  The args
to semilog x are the same as the args to plot.  See help plot for
more info.
Optional keyword args supported are any of the kwargs
supported by plot or set_xscale or set_yscale.  Notable, for
log scaling:
  * basex: base of the x logarithm
  * subsx: the location of the minor ticks; None defaults to
    autosubs, which depend on the number of decades in the
    plot; see set_xscale for details
  * basey: base of the y logarithm
  * subsy: the location of the minor yticks; None defaults to
    autosubs, which depend on the number of decades in the
    plot; see set_yscale for details
The remaining valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

matshow(*self*, *Z*, ***kwargs*)

```
Plot a matrix as an image.

The matrix will be shown the way it would be printed,
with the first row at the top.  Row and column numbering
is zero-based.

Argument:
    Z   anything that can be interpreted as a 2-D array

kwargs: all are passed to imshow.  matshow sets defaults
for extent, origin, interpolation, and aspect; use care
in overriding the extent and origin kwargs, because they
interact.  (Also, if you want to change them, you probably
should be using imshow directly in your own version of
matshow.)

Returns: an AxesImage instance
```

panx(*self*, *numsteps*)

Pan the x axis numsteps (plus pan right, minus pan left)

pany(*self*, *numsteps*)

Pan the x axis numsteps (plus pan up, minus pan down)

pcolor(*self*, **args*, ***kwargs*)

```
pcolor(*args, **kwargs): pseudocolor plot of a 2-D array
Function signatures
  pcolor(C, **kwargs)
  pcolor(X, Y, C, **kwargs)
C is the array of color values
X and Y, if given, specify the (x,y) coordinates of the colored
quadrilaterals; the quadrilateral for C[i,j] has corners at
(X[i,j],Y[i,j]), (X[i,j+1],Y[i,j+1]), (X[i+1,j],Y[i+1,j]),
(X[i+1,j+1],Y[i+1,j+1]).  Ideally the dimensions of X and Y
should be one greater than those of C; if the dimensions are the
same, then the last row and column of C will be ignored.
Note that the the column index corresponds to the x-coordinate,
and the row index corresponds to y; for details, see
the "Grid Orientation" section below.
If either or both of X and Y are 1-D arrays or column vectors,
they will be expanded as needed into the appropriate 2-D arrays,
making a rectangular grid.
X,Y and C may be masked arrays.  If either C[i,j], or one
of the vertices surrounding C[i,j] (X or Y at [i,j],[i+1,j],
[i,j+1],[i=1,j+1]) is masked, nothing is plotted.
Optional keyword args are shown with their defaults below (you must
use kwargs for these):
  * cmap = cm.jet : a cm Colormap instance from matplotlib.cm.
    defaults to cm.jet
  * norm = Normalize() : matplotlib.colors.Normalize instance
    is used to scale luminance data to 0,1.
  * vmin=None and vmax=None : vmin and vmax are used in conjunction
    with norm to normalize luminance data.  If either are None, the
    min and max of the color array C is used.  If you pass a norm
    instance, vmin and vmax will be None
  * shading = 'flat' : or 'faceted'.  If 'faceted', a black grid is
    drawn around each rectangle; if 'flat', edges are not drawn
  * alpha=1.0 : the alpha blending value
Return value is a matplotlib.collections.PatchCollection
object
Grid Orientation
    The orientation follows the Matlab(TM) convention: an
    array C with shape (nrows, ncolumns) is plotted with
    the column number as X and the row number as Y, increasing
    up; hence it is plotted the way the array would be printed,
    except that the Y axis is reversed.  That is, C is taken
    as C(y,x).
    Similarly for meshgrid:
        x = arange(5)
        y = arange(3)
        X, Y = meshgrid(x,y)
    is equivalent to
        X = array([[0, 1, 2, 3, 4],
                   [0, 1, 2, 3, 4],
                   [0, 1, 2, 3, 4]])
        Y = array([[0, 0, 0, 0, 0],
                   [1, 1, 1, 1, 1],
                   [2, 2, 2, 2, 2]])
    so if you have
        C = rand( len(x), len(y))
```

pcolor_classic(*self*, **args*)

pcolor_classic is no longer available; please use pcolor, which is a drop-in replacement.

pcolormesh(*self*, **args*, ***kwargs*)

```
PCOLORMESH(*args, **kwargs)
Function signatures
  PCOLORMESH(C) - make a pseudocolor plot of matrix C
  PCOLORMESH(X, Y, C) - a pseudo color plot of C on the matrices X and Y
  PCOLORMESH(C, **kwargs) - Use keyword args to control colormapping and
                        scaling; see below
C may be a masked array, but X and Y may not.  Masked array support
is implemented via cmap and norm; in contrast, pcolor simply does
not draw quadrilaterals with masked colors or vertices.
Optional keyword args are shown with their defaults below (you must
use kwargs for these):
  * cmap = cm.jet : a cm Colormap instance from matplotlib.cm.
    defaults to cm.jet
  * norm = Normalize() : matplotlib.colors.Normalize instance
    is used to scale luminance data to 0,1.  Instantiate it
    with clip=False if C is a masked array.
  * vmin=None and vmax=None : vmin and vmax are used in conjunction
    with norm to normalize luminance data.  If either are None, the
    min and max of the color array C is used.
  * shading = 'flat' : or 'faceted'.  If 'faceted', a black grid is
    drawn around each rectangle; if 'flat', edge colors are same as
    face colors
  * alpha=1.0 : the alpha blending value
Return value is a matplotlib.collections.PatchCollection
object
See pcolor for an explantion of the grid orientation and the
expansion of 1-D X and/or Y to 2-D arrays.
kwargs can be used to control the QuadMesh polygon collection properties:
        alpha: float
        animated: [True | False]
        array: unknown
        axes: an axes instance
        clim: a length 2 sequence of floats
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cmap: a colormap
        color: matplotlib color arg or sequence of rgba tuples
        colorbar: unknown
        edgecolor: matplotlib color arg or sequence of rgba tuples
        facecolor: matplotlib color arg or sequence of rgba tuples
        figure: a matplotlib.figure.Figure instance
        label: any string
        linewidth: float or sequence of floats
        lod: [True | False]
        norm: unknown
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

pick(*self*, **args*)

pick(mouseevent)
each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

Overrides: matplotlib.artist.Artist.pick

pie(*self*, *x*, *explode*=None, *labels*=None, *colors*=None, *autopct*=None, *pctdistance*=0.59999999999999998, *shadow*=False)

```
PIE(x, explode=None, labels=None,
    colors=('b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'),
    autopct=None, pctdistance=0.6, shadow=False)

Make a pie chart of array x.  The fractional area of each wedge is
given by x/sum(x).  If sum(x)<=1, then the values of x give the
fractional area directly and the array will not be normalized.

  - explode, if not None, is a len(x) array which specifies the
    fraction of the radius to offset that wedge.

  - colors is a sequence of matplotlib color args that the pie chart
    will cycle.

  - labels, if not None, is a len(x) list of labels.

  - autopct, if not None, is a string or function used to label the
    wedges with their numeric value.  The label will be placed inside
    the wedge.  If it is a format string, the label will be fmt%pct.
    If it is a function, it will be called

  - pctdistance is the ratio between the center of each pie slice
    and the start of the text generated by autopct.  Ignored if autopct
    is None; default is 0.6.

  - shadow, if True, will draw a shadow beneath the pie.

The pie chart will probably look best if the figure and axes are
square.  Eg,

  figure(figsize=(8,8))
  ax = axes([0.1, 0.1, 0.8, 0.8])

Return value:

  If autopct is None, return a list of (patches, texts), where patches
  is a sequence of matplotlib.patches.Wedge instances and texts is a
  list of the label Text instnaces

  If autopct is not None, return (patches, texts, autotexts), where
  patches and texts are as above, and autotexts is a list of text
  instances for the numeric labels
```

plot(*self*, **args*, ***kwargs*)

```
PLOT(*args, **kwargs)
Plot lines and/or markers to the Axes.  *args is a variable length
argument, allowing for multiple x,y pairs with an optional format
string.  For example, each of the following is legal
    plot(x,y)            # plot x and y using the default line style and color
    plot(x,y, 'bo')      # plot x and y using blue circle markers
    plot(y)              # plot y using x as index array 0..N-1
    plot(y, 'r+')        # ditto, but with red plusses
If x and/or y is 2-Dimensional, then the corresponding columns
will be plotted.
An arbitrary number of x, y, fmt groups can be specified, as in
a.plot(x1, y1, 'g^', x2, y2, 'g-')
Return value is a list of lines that were added.
The following line styles are supported:
    -     : solid line
    --    : dashed line
    -.    : dash-dot line
    :     : dotted line
    .     : points
    ,     : pixels
    o     : circle symbols
    ^     : triangle up symbols
    v     : triangle down symbols
    <     : triangle left symbols
    >     : triangle right symbols
    s     : square symbols
    +     : plus symbols
    x     : cross symbols
    D     : diamond symbols
    d     : thin diamond symbols
    1     : tripod down symbols
    2     : tripod up symbols
    3     : tripod left symbols
    4     : tripod right symbols
    h     : hexagon symbols
    H     : rotated hexagon symbols
    p     : pentagon symbols
    |     : vertical line symbols
    _     : horizontal line symbols
    steps : use gnuplot style 'steps' # kwarg only
The following color abbreviations are supported
    b  : blue
    g  : green
    r  : red
    c  : cyan
    m  : magenta
    y  : yellow
    k  : black
    w  : white
In addition, you can specify colors in many weird and
wonderful ways, including full names 'green', hex strings
'#008000', RGB or RGBA tuples (0,1,0,1) or grayscale
intensities as a string '0.8'.
Line styles and colors are combined in a single format string, as in
'bo' for blue circles.
The **kwargs can be used to set line properties (any property that has
```

plot_date(*self, x, y, fmt=*'bo', *tz=*None, *xdate=*True, *ydate=*False, ***kwargs*)

```
PLOT_DATE(x, y, fmt='bo', tz=None, xdate=True, ydate=False, **kwargs)
Similar to the plot() command, except the x or y (or both) data
is considered to be dates, and the axis is labeled accordingly.
x or y (or both) can be a sequence of dates represented as
float days since 0001-01-01 UTC.
fmt is a plot format string.
tz is the time zone to use in labelling dates.  Defaults to rc value.
If xdate is True, the x-axis will be labeled with dates.
If ydate is True, the y-axis will be labeled with dates.
Note if you are using custom date tickers and formatters, it
may be necessary to set the formatters/locators after the call
to plot_date since plot_date will set the default tick locator
to AutoDateLocator (if the tick locator is not already set to
a DateLocator instance) and the default tick formatter to
AutoDateFormatter (if the tick formatter is not already set to
a DateFormatter instance).
Valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
See matplotlib.dates for helper functions date2num, num2date
and drange for help on creating the required floating point dates
```

psd(*self*, *x*, *NFFT*=256, *Fs*=2, *detrend*=<function detrend_none at 0x8413b1c>, *window*=<function window_hanning at 0x8413a04>, *noverlap*=0, ***kwargs*)

```
PSD(x, NFFT=256, Fs=2, detrend=detrend_none,
    window=window_hanning, noverlap=0, **kwargs)
The power spectral density by Welches average periodogram method.  The
vector x is divided into NFFT length segments.  Each segment is
detrended by function detrend and windowed by function window.
noperlap gives the length of the overlap between segments.  The
absolute(fft(segment))**2 of each segment are averaged to compute Pxx,
with a scaling to correct for power loss due to windowing.  Fs is the
sampling frequency.
    NFFT is the length of the fft segment; must be a power of 2
    Fs is the sampling frequency.
    detrend - the function applied to each segment before fft-ing,
      designed to remove the mean or linear trend.  Unlike in matlab,
      where the detrend parameter is a vector, in matplotlib is it a
      function.  The mlab module defines detrend_none, detrend_mean,
      detrend_linear, but you can use a custom function as well.
    window - the function used to window the segments.  window is a
      function, unlike in matlab(TM) where it is a vector.  mlab defines
      window_none, window_hanning, but you can use a custom function
      as well.
    noverlap gives the length of the overlap between segments.
Returns the tuple Pxx, freqs
For plotting, the power is plotted as 10*log10(pxx)) for decibels,
though pxx itself is returned
Refs:
  Bendat & Piersol -- Random Data: Analysis and Measurement
  Procedures, John Wiley & Sons (1986)
kwargs control the Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
```

quiver(*self*, **args*, ***kw*)

```
Plot a 2-D field of arrows.

Function signatures:

    quiver(U, V, **kw)
    quiver(U, V, C, **kw)
    quiver(X, Y, U, V, **kw)
    quiver(X, Y, U, V, C, **kw)

Arguments:

    X, Y give the x and y coordinates of the arrow locations
        (default is tail of arrow; see 'pivot' kwarg)
    U, V give the x and y components of the arrow vectors
    C is an optional array used to map colors to the arrows

    All arguments may be 1-D or 2-D arrays or sequences.
    If X and Y are absent, they will be generated as a uniform grid.
    If U and V are 2-D arrays but X and Y are 1-D, and if
        len(X) and len(Y) match the column and row dimensions
        of U, then X and Y will be expanded with meshgrid.

Keyword arguments (default given first):

  * units = 'width' | 'height' | 'dots' | 'inches' | 'x' | 'y'
            arrow units; the arrow dimensions *except for length*
            are in multiples of this unit.
  * scale = None | float
            data units per arrow unit, e.g. m/s per plot width;
            a smaller scale parameter makes the arrow longer.
            If None, a simple autoscaling algorithm is used, based
            on the average vector length and the number of vectors.

    Arrow dimensions and scales can be in any of several units:

    'width' or 'height': the width or height of the axes
    'dots' or 'inches':  pixels or inches, based on the figure dpi
    'x' or 'y': X or Y data units

    In all cases the arrow aspect ratio is 1, so that if U==V the angle
    of the arrow on the plot is 45 degrees CCW from the X-axis.

    The arrows scale differently depending on the units, however.
    For 'x' or 'y', the arrows get larger as one zooms in; for other
    units, the arrow size is independent of the zoom state.  For
    'width or 'height', the arrow size increases with the width and
    height of the axes, respectively, when the the window is resized;
    for 'dots' or 'inches', resizing does not change the arrows.
```



```
  * width = ?       shaft width in arrow units; default depends on
                        choice of units, above, and number of vectors;
                        a typical starting value is about
                        0.005 times the width of the plot.
  * headwidth = 3    head width as multiple of shaft width
  * headlength = 5   head length as multiple of shaft width
```

quiver2(*self*, **args*, ***kw*)

```
Plot a 2-D field of arrows.

Function signatures:

    quiver(U, V, **kw)
    quiver(U, V, C, **kw)
    quiver(X, Y, U, V, **kw)
    quiver(X, Y, U, V, C, **kw)

Arguments:

    X, Y give the x and y coordinates of the arrow locations
        (default is tail of arrow; see 'pivot' kwarg)
    U, V give the x and y components of the arrow vectors
    C is an optional array used to map colors to the arrows

    All arguments may be 1-D or 2-D arrays or sequences.
    If X and Y are absent, they will be generated as a uniform grid.
    If U and V are 2-D arrays but X and Y are 1-D, and if
        len(X) and len(Y) match the column and row dimensions
        of U, then X and Y will be expanded with meshgrid.

Keyword arguments (default given first):

  * units = 'width' | 'height' | 'dots' | 'inches' | 'x' | 'y'
            arrow units; the arrow dimensions *except for length*
            are in multiples of this unit.
  * scale = None | float
            data units per arrow unit, e.g. m/s per plot width;
            a smaller scale parameter makes the arrow longer.
            If None, a simple autoscaling algorithm is used, based
            on the average vector length and the number of vectors.

    Arrow dimensions and scales can be in any of several units:

    'width' or 'height': the width or height of the axes
    'dots' or 'inches':  pixels or inches, based on the figure dpi
    'x' or 'y': X or Y data units

    In all cases the arrow aspect ratio is 1, so that if U==V the angle
    of the arrow on the plot is 45 degrees CCW from the X-axis.

    The arrows scale differently depending on the units, however.
    For 'x' or 'y', the arrows get larger as one zooms in; for other
    units, the arrow size is independent of the zoom state.  For
    'width or 'height', the arrow size increases with the width and
    height of the axes, respectively, when the the window is resized;
    for 'dots' or 'inches', resizing does not change the arrows.
```



```
  * width = ?       shaft width in arrow units; default depends on
                        choice of units, above, and number of vectors;
                        a typical starting value is about
                        0.005 times the width of the plot.
  * headwidth = 3    head width as multiple of shaft width
  * headlength = 5   head length as multiple of shaft width
```

quiver_classic(*self*, *U*, *V*, **args*, ***kwargs*)

QUIVER(X, Y, U, V) QUIVER(U, V) QUIVER(X, Y, U, V, S) QUIVER(U, V, S) QUIVER(..., color=None, width=1.0, cmap=None, norm=None)
Make a vector plot (U, V) with arrows on a grid (X, Y)
If X and Y are not specified, U and V must be 2D arrays. Equally spaced X and Y grids are then generated using the meshgrid command.
color can be a color value or an array of colors, so that the arrows can be colored according to another dataset. If cmap is specified and color is 'length', the colormap is used to give a color according to the vector's length.
If color is a scalar field, the colormap is used to map the scalar to a color If a colormap is specified and color is an array of color triplets, then the colormap is ignored
width is a scalar that controls the width of the arrows
if S is specified it is used to scale the vectors. Use S=0 to disable automatic scaling. If S!=0, vectors are scaled to fit within the grid and then are multiplied by S.

quiverkey(*self*, **args*, ***kw*)

```
Add a key to a quiver plot.

Function signature:
    quiverkey(Q, X, Y, U, label, **kw)

Arguments:
    Q is the Quiver instance returned by a call to quiver.
    X, Y give the location of the key; additional explanation follows.
    U is the length of the key
    label is a string with the length and units of the key

Keyword arguments (default given first):
  * coordinates = 'axes' | 'figure' | 'data' | 'inches'
        Coordinate system and units for X, Y: 'axes' and 'figure'
        are normalized coordinate systems with 0,0 in the lower
        left and 1,1 in the upper right; 'data' are the axes
        data coordinates (used for the locations of the vectors
        in the quiver plot itself); 'inches' is position in the
        figure in inches, with 0,0 at the lower left corner.
  * color overrides face and edge colors from Q.
  * labelpos = 'N' | 'S' | 'E' | 'W'
        Position the label above, below, to the right, to the left
        of the arrow, respectively.
  * labelsep = 0.1 inches distance between the arrow and the label
  * labelcolor (defaults to default Text color)
  * fontproperties is a dictionary with keyword arguments accepted
        by the FontProperties initializer: family, style, variant,
        size, weight

    Any additional keyword arguments are used to override vector
    properties taken from Q.

    The positioning of the key depends on X, Y, coordinates, and
    labelpos.  If labelpos is 'N' or 'S', X,Y give the position
    of the middle of the key arrow.  If labelpos is 'E', X,Y
    positions the head, and if labelpos is 'W', X,Y positions the
    tail; in either of these two cases, X,Y is somewhere in the middle
    of the arrow+label key object.
```

redraw_in_frame(*self*)

This method can only be used after an initial draw which caches the renderer. It is used to efficiently update Axes data (axis ticks, labels, etc are not updated)

relim(*self*)

recompute the datalimits based on current artists

scatter(*self*, *x*, *y*, *s*=20, *c*='b', *marker*='o', *cmap*=None, *norm*=None, *vmin*=None, *vmax*=None, *alpha*=1.0, *linewidths*=None, *faceted*=True, *verts*=None, ***kwargs*)

```
SCATTER(x, y, s=20, c='b', marker='o', cmap=None, norm=None,
    vmin=None, vmax=None, alpha=1.0, linewidths=None,
    faceted=True, **kwargs)
Supported function signatures:
    SCATTER(x, y, **kwargs)
    SCATTER(x, y, s, **kwargs)
    SCATTER(x, y, s, c, **kwargs)
Make a scatter plot of x versus y, where x, y are 1-D sequences
of the same length, N.
Arguments s and c can also be given as kwargs; this is encouraged
for readability.
    s is a size in points^2.  It is a scalar
      or an array of the same length as x and y.
    c is a color and can be a single color format string,
      or a sequence of color specifications of length N,
      or a sequence of N numbers to be mapped to colors
      using the cmap and norm specified via kwargs (see below).
      Note that c should not be a single numeric RGB or RGBA
      sequence because that is indistinguishable from an array
      of values to be colormapped. c can be a 2-D array in which
      the rows are RGB or RGBA, however.
The marker can be one of
    's' : square
    'o' : circle
    '^' : triangle up
    '>' : triangle right
    'v' : triangle down
    '<' : triangle left
    'd' : diamond
    'p' : pentagram
    'h' : hexagon
    '8' : octagon
If marker is None and verts is not None, verts is a sequence
of (x,y) vertices for a custom scatter symbol.
s is a size argument in points squared.
Any or all of x, y, s, and c may be masked arrays, in which
case all masks will be combined and only unmasked points
will be plotted.
Other keyword args; the color mapping and normalization arguments will
on be used if c is an array of floats
  * cmap = cm.jet : a colors.Colormap instance from matplotlib.cm.
    defaults to rc image.cmap
  * norm = Normalize() : matplotlib.colors.Normalize instance
    is used to scale luminance data to 0,1.
  * vmin=None and vmax=None : vmin and vmax are used in conjunction
    with norm to normalize luminance data.  If either are None, the
    min and max of the color array C is used.  Note if you pass a norm
    instance, your settings for vmin and vmax will be ignored
  * alpha =1.0 : the alpha value for the patches
  * linewidths, if None, defaults to (lines.linewidth,).  Note
    that this is a tuple, and if you set the linewidths
    argument you must set it as a sequence of floats, as
    required by RegularPolyCollection -- see
    matplotlib.collections.RegularPolyCollection for details
 * faceted: if True, will use the default edgecolor for the
```

scatter_classic(*self*, *x*, *y*, *s*=`None`, *c*=`'b'`)

scatter_classic is no longer available; please use scatter. To help in porting, for comparison to the scatter docstring, here is the scatter_classic docstring:
SCATTER_CLASSIC(x, y, s=None, c='b')
Make a scatter plot of x versus y. s is a size (in data coords) and can be either a scalar or an array of the same length as x or y. c is a color and can be a single color format string or an length(x) array of intensities which will be mapped by the colormap jet.
If size is None a default size will be used

semilogx(*self*, **args*, ***kwargs*)

```
SEMILOGX(*args, **kwargs)
Make a semilog plot with log scaling on the x axis.  The args to
semilog x are the same as the args to plot.  See help plot for more
info.
Optional keyword args supported are any of the kwargs supported by
plot or set_xscale.  Notable, for log scaling:
    * basex: base of the logarithm
    * subsx: the location of the minor ticks; None defaults to
      autosubs, which depend on the number of decades in the
      plot; see set_xscale for details
The remaining valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

semilogy(*self*, **args*, ***kwargs*)

```
SEMILOGY(*args, **kwargs):
Make a semilog plot with log scaling on the y axis.  The args to
semilogy are the same as the args to plot.  See help plot for more
info.
Optional keyword args supported are any of the kwargs supported by
plot or set_yscale.  Notable, for log scaling:
    * basey: base of the logarithm
    * subsy: a sequence of the location of the minor ticks;
      None defaults to autosubs, which depend on the number of
      decades in the plot; see set_yscale for details
The remaining valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

set_adjustable(*self*, *adjustable*)

ACCEPTS: ['box' | 'datalim']

set_anchor(*self*, *anchor*)

ACCEPTS: ['C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W']

set_aspect(*self*, *aspect*, *adjustable*=None, *anchor*=None)

```
aspect:
   'auto'   -  automatic; fill position rectangle with data
   'normal' -  same as 'auto'; deprecated
   'equal'  -  same scaling from data to plot units for x and y
    num     -  a circle will be stretched such that the height
               is num times the width. aspect=1 is the same as
               aspect='equal'.

adjustable:
    'box'      - change physical size of axes
    'datalim'  - change xlim or ylim

anchor:
    'C'     - centered
    'SW'    - lower left corner
    'S'     - middle of bottom edge
    'SE'    - lower right corner
         etc.

ACCEPTS: ['auto' | 'equal' | aspect_ratio]
```

set_autoscale_on(*self*, *b*)

Set whether autoscaling is applied on plot commands
ACCEPTS: True|False

set_axis_bgcolor(*self*, *color*)

set the axes background color
ACCEPTS: any matplotlib color - see help(colors)

set_axis_off(*self*)

turn off the axis
ACCEPTS: void

set_axis_on(*self*)

turn on the axis
ACCEPTS: void

set_axisbelow(*self*, *b*)

Set whether the axis ticks and gridlines are above or below most artists
ACCEPTS: True|False

set_cursor_props(*self*, **args*)

Set the cursor property as ax.set_cursor_props(linewidth, color) OR ax.set_cursor_props((linewidth, color))
ACCEPTS: a (float, color) tuple

set_figure(*self*, *fig*)

Set the Axes figure
ACCEPTS: a Figure instance

Overrides: matplotlib.artist.Artist.set_figure

set_frame_on(*self*, *b*)

Set whether the axes rectangle patch is drawn
ACCEPTS: True|False

set_navigate(*self*, *b*)

Set whether the axes responds to navigation toolbar commands
ACCEPTS: True|False

set_navigate_mode(*self*, *b*)

Set the navigation toolbar button status; this is not a user-API function.

set_position(*self*, *pos*, *which*=`'both'`)

```
Set the axes position with pos = [left, bottom, width, height]
in relative 0,1 coords

There are two position variables: one which is ultimately
used, but which may be modified by apply_aspect, and a second
which is the starting point for apply_aspect.

which = 'active' to change the first;
        'original' to change the second;
        'both' to change both

ACCEPTS: len(4) sequence of floats
```

set_title(*self*, *label*, *fontdict*=None, ***kwargs*)

```
SET_TITLE(label, fontdict=None, **kwargs):
Set the title for the axes.  See the text docstring for information
of how override and the optional args work
kwargs are Text properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: str
```

set_xlabel(*self*, *xlabel*, *fontdict*=None, ***kwargs*)

```
SET_XLABEL(xlabel, fontdict=None, **kwargs)
Set the label for the xaxis.  See the text docstring for information
of how override and the optional args work.
Valid kwargs are Text properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: str
```

set_xlim(*self*, *xmin*=None, *xmax*=None, *emit*=False, ***kwargs*)

set_xlim(self, *args, **kwargs):
Set the limits for the xaxis; v = [xmin, xmax]
set_xlim((valmin, valmax)) set_xlim(valmin, valmax) set_xlim(xmin=1) # xmax unchanged
set_xlim(xmax=1) # xmin unchanged
Valid kwargs:
xmin : the min of the xlim xmax : the max of the xlim emit : notify observers of lim change
Returns the current xlimits as a length 2 tuple
ACCEPTS: len(2) sequence of floats

set_xscale(*self*, *value*, *basex*=10, *subsx*=None)

```
SET_XSCALE(value, basex=10, subsx=None)

Set the xscaling: 'log' or 'linear'

If value is 'log', the additional kwargs have the following meaning

    * basex: base of the logarithm

    * subsx: a sequence of the location of the minor ticks;
      None defaults to autosubs, which depend on the number of
      decades in the plot.  Eg for base 10, subsx=(1,2,5) will
      put minor ticks on 1,2,5,11,12,15,21, ....To turn off
      minor ticking, set subsx=[]

ACCEPTS: ['log' | 'linear' ]
```

set_xticklabels(*self*, *labels*, *fontdict*=None, ***kwargs*)

```
SET_XTICKLABELS(labels, fontdict=None, **kwargs)
Set the xtick labels with list of strings labels Return a list of axis
text instances.
kwargs set the Text properties.  Valid properties are
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: sequence of strings
```

set_xticks(*self*, *ticks*)

Set the x ticks with list of ticks
ACCEPTS: sequence of floats

set_ylabel(*self*, *ylabel*, *fontdict*=None, ***kwargs*)

```
SET_YLABEL(ylabel, fontdict=None, **kwargs)
Set the label for the yaxis
See the text doctstring for information of how override and
the optional args work
Valid kwargs are Text properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: str
```

set_ylim(*self*, *ymin*=None, *ymax*=None, *emit*=False, ***kwargs*)

set_ylim(self, *args, **kwargs):
Set the limits for the yaxis; v = [ymin, ymax]
set_ylim((valmin, valmax)) set_ylim(valmin, valmax) set_ylim(ymin=1) # ymax unchanged
set_ylim(ymax=1) # ymin unchanged
Valid kwargs:
ymin : the min of the ylim ymax : the max of the ylim emit : notify observers of lim change
Returns the current ylimits as a length 2 tuple
ACCEPTS: len(2) sequence of floats

set_yscale(*self*, *value*, *basey*=10, *subsy*=None)

```
SET_YSCALE(value, basey=10, subsy=None)

Set the yscaling: 'log' or 'linear'

If value is 'log', the additional kwargs have the following meaning

    * basey: base of the logarithm

    * subsy: a sequence of the location of the minor ticks;
      None defaults to autosubs, which depend on the number of
      decades in the plot.  Eg for base 10, subsy=(1,2,5) will
      put minor ticks on 1,2,5,11,12,15, 21, ....To turn off
      minor ticking, set subsy=[]

ACCEPTS: ['log' | 'linear']
```

set_yticklabels(*self*, *labels*, *fontdict*=None, ***kwargs*)

```
SET_YTICKLABELS(labels, fontdict=None, **kwargs)
Set the ytick labels with list of strings labels.  Return a list of
Text instances.
kwargs set Text properties for the labels.  Valid properties are
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: sequence of strings
```

set_yticks(*self*, *ticks*)

Set the y ticks with list of ticks
ACCEPTS: sequence of floats

specgram(*self*, *x*, *NFFT*=256, *Fs*=2, *detrend*=<function detrend_none at 0x8413b1c>, *window*=<function window_hanning at 0x8413a04>, *noverlap*=128, *cmap*=None, *xextent*=None)

```
SPECGRAM(x, NFFT=256, Fs=2, detrend=detrend_none,
         window=window_hanning, noverlap=128,
         cmap=None, xextent=None)

Compute a spectrogram of data in x.  Data are split into NFFT length
segements and the PSD of each section is computed.  The windowing
function window is applied to each segment, and the amount of overlap
of each segment is specified with noverlap.

    * cmap is a colormap; if None use default determined by rc

    * xextent is the image extent in the xaxes xextent=xmin, xmax -
      default 0, max(bins), 0, max(freqs) where bins is the return
      value from matplotlib.matplotlib.mlab.specgram

    * See help(psd) for information on the other keyword arguments.

Return value is (Pxx, freqs, bins, im), where

    bins are the time points the spectrogram is calculated over

    freqs is an array of frequencies

    Pxx is a len(times) x len(freqs) array of power

    im is a matplotlib.image.AxesImage.

Note: If x is real (i.e. non-complex) only the positive spectrum is
shown.  If x is complex both positive and negative parts of the
spectrum are shown.
```

spy(*self*, *Z*, *precision*=None, *marker*=None, *markersize*=None, *aspect*='equal', ***kwargs*)

```
spy(Z) plots the sparsity pattern of the 2-D array Z

If precision is None, any non-zero value will be plotted;
else, values of absolute(Z)>precision will be plotted.

The array will be plotted as it would be printed, with
the first index (row) increasing down and the second
index (column) increasing to the right.

By default aspect is 'equal' so that each array element
occupies a square space; set the aspect kwarg to 'auto'
to allow the plot to fill the plot box, or to any scalar
number to specify the aspect ratio of an array element
directly.

Two plotting styles are available: image or marker. Both
are available for full arrays, but only the marker style
works for scipy.sparse.spmatrix instances.

If marker and markersize are None, an image will be
returned and any remaining kwargs are passed to imshow;
else, a Line2D object will be returned with the value
of marker determining the marker type, and any remaining
kwargs passed to the axes plot method.

If marker and markersize are None, useful kwargs include:
    cmap
    alpha
See documentation for imshow() for details.
For controlling colors, e.g. cyan background and red marks, use:
    cmap = matplotlib.colors.ListedColormap(['c','r'])

If marker or markersize is not None, useful kwargs include:
    marker
    markersize
    color
See documentation for plot() for details.

Useful values for marker include:
    's'  square (default)
    'o'  circle
    '.'  point
    ','  pixel
```

stem(*self*, *x*, *y*, *linefmt*=`'b-'`, *markerfmt*=`'bo'`, *basefmt*=`'r-'`)

STEM(x, y, linefmt='b-', markerfmt='bo', basefmt='r-')
A stem plot plots vertical lines (using linefmt) at each x location from the baseline to y, and places a marker there using markerfmt. A horizontal line at 0 is is plotted using basefmt
Return value is (markerline, stemlines, baseline) .
See http://www.mathworks.com/access/helpdesk/help/techdoc/ref/stem.html for details and examples/stem_plot.py for a demo.

table(*self*, ***kwargs*)

```
TABLE(cellText=None, cellColours=None,
      cellLoc='right', colWidths=None,
      rowLabels=None, rowColours=None, rowLoc='left',
      colLabels=None, colColours=None, colLoc='center',
      loc='bottom', bbox=None):
Add a table to the current axes.  Returns a table instance.  For
finer grained control over tables, use the Table class and add it
to the axes with add_table.
Thanks to John Gill for providing the class and table.
kwargs control the Table properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        figure: a matplotlib.figure.Figure instance
        fontsize: a float in points
        label: any string
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

text(*self*, *x*, *y*, *s*, *fontdict*=None, *withdash*=False, ***kwargs*)

```
TEXT(x, y, s, fontdict=None, **kwargs)
Add text in string s to axis at location x,y (data coords)
  fontdict is a dictionary to override the default text properties.
  If fontdict is None, the defaults are determined by your rc
  parameters.
  withdash=True will create a TextWithDash instance instead
  of a Text instance.
Individual keyword arguments can be used to override any given
parameter
    text(x, y, s, fontsize=12)
The default transform specifies that text is in data coords,
alternatively, you can specify text in axis coords (0,0 lower left and
1,1 upper right).  The example below places text in the center of the
axes
    text(0.5, 0.5,'matplotlib',
         horizontalalignment='center',
         verticalalignment='center',
         transform = ax.transAxes,
    )
You can put a rectangular box around the text instance (eg to
set a background color) by using the keyword bbox.  bbox is a
dictionary of matplotlib.patches.Rectangle properties (see help
for Rectangle for a list of these).  For example
 text(x, y, s, bbox=dict(facecolor='red', alpha=0.5))
Valid kwargs are Text properties
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
```

ticklabel_format(*self*, ****kwargs**)

```
Convenience method for manipulating the ScalarFormatter
used by default for linear axes.

kwargs:
    style = 'sci' (or 'scientific') or 'plain';
                plain turns off scientific notation
    axis = 'x', 'y', or 'both'

Only the major ticks are affected.
If the method is called when the ScalarFormatter is not
the one being used, an AttributeError will be raised with
no additional error message.

Additional capabilities and/or friendlier error checking may be added.
```

toggle_log_lineary(*self*)

toggle between log and linear on the y axis

update_datalim(*self*, *xys*)

Update the data lim bbox with seq of xy tups or equiv. 2-D array

update_datalim_numerix(*self*, *x*, *y*)

Update the data lim bbox with seq of xy tups

vlines(*self*, *x*, *ymin*, *ymax*, *colors*=`'k'`, *linestyle*=`'solid'`, *label*=`''`, ***kwargs*)

```
VLINES(x, ymin, ymax, color='k')
Plot vertical lines at each x from ymin to ymax.  ymin or ymax can be
scalars or len(x) numpy arrays.  If they are scalars, then the
respective values are constant, else the heights of the lines are
determined by ymin and ymax
colors is a line collections color args, either a single color
or a len(x) list of colors
linestyle is one of solid|dashed|dashdot|dotted
Returns the LineCollection that was added
kwargs are LineCollection properties:
        alpha: float or sequence of floats
        animated: [True | False]
        array: unknown
        axes: an axes instance
        clim: a length 2 sequence of floats
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cmap: a colormap
        color: matplotlib color arg or sequence of rgba tuples
        colorbar: unknown
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle: ['solid' | 'dashed', 'dashdot', 'dotted' |  (offset, on-off-dash-seq) ]
        linewidth: float or sequence of floats
        lod: [True | False]
        norm: unknown
        picker: [None|float|boolean|callable]
        segments: unknown
        transform: a matplotlib.transform transformation instance
        verts: unknown
        visible: [True | False]
        zorder: any number
```

xaxis_date(*self*, *tz*=`None`)

Sets up x-axis ticks and labels that treat the x data as dates.
tz is the time zone to use in labeling dates. Defaults to rc value.

xcorr(*self*, *x*, *y*, *normed*=`False`, *detrend*=`<function detrend_none at 0x8413b1c>`, *usevlines*=`False`, *maxlags*=`None`, ***kwargs*)

XCORR(x, y, normed=False, detrend=detrend_none, usevlines=False, **kwargs): Plot the cross correlation between x and y. If normed=True, normalize the data but the cross correlation at 0-th lag. x and y are detrended by the detrend callable (default no normalization. x and y must be equal length data are plotted as plot(lags, c, **kwargs) return value is lags, c, line where lags are a length 2*maxlags+1 lag vector, c is the 2*maxlags+1 auto correlation vector, and line is a Line2D instance returned by plot. The default linestyle is None and the default marker is 'o', though these can be overridden with keyword args. The cross correlation is performed with numerix cross_correlate with mode=2. If usevlines is True, Axes.vlines rather than Axes.plot is used to draw vertical lines from the origin to the acorr. Otherwise the plotstyle is determined by the kwargs, which are Line2D properties. If usevlines, the return value is lags, c, linecol, b where linecol is the LineCollection and b is the x-axis if usevlines=True, kwargs are passed onto Axes.vlines if usevlines=False, kwargs are passed onto Axes.plot maxlags is a positive integer detailing the number of lags to show. The default value of None will return all (2*len(x)-1) lags. See the respective function for documentation on valid kwargs

yaxis_date(*self*, *tz*=`None`)

Sets up y-axis ticks and labels that treat the y data as dates.
tz is the time zone to use in labeling dates. Defaults to rc value.

zoomx(*self*, *numsteps*)

Zoom in on the x xaxis numsteps (plus for zoom in, minus for zoom out)

zoomy(*self*, *numsteps*)

Zoom in on the x xaxis numsteps (plus for zoom in, minus for zoom out)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

6.2.2 Class Variables

Name	Description
scaled	**Value:** `{0: 'linear', 1: 'log'}` *(type=`dict`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

6.3 Class PolarAxes

matplotlib.artist.Artist
  matplotlib.axes.Axes
    PolarAxes

Known Subclasses: PolarSubplot

```
Make a PolarAxes.  The rectangular bounding box of the axes is given by


   PolarAxes(position=[left, bottom, width, height])

where all the arguments are fractions in [0,1] which specify the
fraction of the total figure window.

axisbg is the color of the axis background

Attributes:
  thetagridlines  : a list of Line2D for the theta grids
  rgridlines      : a list of Line2D for the radial grids
  thetagridlabels : a list of Text for the theta grid labels
  rgridlabels     : a list of Text for the theta grid labels
```

6.3.1 Methods

__init__(*self*, **args*, ***kwarg*)

See Axes base class for args and kwargs documentation

Overrides: matplotlib.axes.Axes.__init__

autoscale_view(*self*, *scalex*=`True`, *scaley*=`True`)

set the view limits to include all the data in the axes

Overrides: matplotlib.axes.Axes.autoscale_view

cla(*self*)

Clear the current axes

Overrides: matplotlib.axes.Axes.cla

draw(*self*, *renderer*)
Overrides: matplotlib.axes.Axes.draw

format_coord(*self*, *theta*, *r*)

return a format string formatting the coordinate

Overrides: matplotlib.axes.Axes.format_coord

get_children(*self*)

return a list of child artists

Overrides: matplotlib.axes.Axes.get_children

get_rmax(*self*)

get the maximum radius in the view limits dimension

get_xscale(*self*)

return the xaxis scale string

Overrides: matplotlib.axes.Axes.get_xscale

get_yscale(*self*)

return the yaxis scale string

Overrides: matplotlib.axes.Axes.get_yscale

grid(*self*, *b*)

Set the axes grids on or off; b is a boolean

Overrides: matplotlib.axes.Axes.grid

has_data(*self*)

return true if any artists have been added to axes

Overrides: matplotlib.axes.Axes.has_data

regrid(*self*, *rmax*)

set_rgrids(*self*, *radii*, *labels*=None, *angle*=22.5, *rpad*=0.050000000000000003, ***kwargs*)

```
set the radial locations and labels of the r grids
The labels will appear at radial distances radii at angle
labels, if not None, is a len(radii) list of strings of the
labels to use at each angle.
if labels is None, the self.rformatter will be used
rpad is a fraction of the max of radii which will pad each of
the radial labels in the radial direction.

Return value is a list of lines, labels where the lines are
matplotlib.Line2D instances and the labels are matplotlib.Text
instances
kwargs control the rgrid Text label properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: sequence of floats
```

set_rmax(*self*, *rmax*)

set_thetagrids(*self*, *angles*, *labels*=None, *fmt*='%d', *frac*=1.1000000000000001, ***kwargs*)

```
set the angles at which to place the theta grids (these
gridlines are equal along the theta dimension).  angles is in
degrees
labels, if not None, is a len(angles) list of strings of the
labels to use at each angle.
if labels is None, the labels with be fmt%angle
frac is the fraction of the polar axes radius at which to
place the label (1 is the edge).Eg 1.05 isd outside the axes
and 0.95 is inside the axes
Return value is a list of lines, labels where the lines are
matplotlib.Line2D instances and the labels are matplotlib.Text
instances:
kwargs are optional text properties for the labels
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: sequence of floats
```

set_xlabel(*self*, *xlabel*, *fontdict*=None, ***kwargs*)

xlabel not implemented

Overrides: matplotlib.axes.Axes.set_xlabel

set_xlim(*self*, *xmin*=None, *xmax*=None, *emit*=True)

set the xlimits ACCEPTS: len(2) sequence of floats

Overrides: matplotlib.axes.Axes.set_xlim

set_ylabel(*self*, *ylabel*, *fontdict*=None, ***kwargs*)

ylabel not implemented

Overrides: matplotlib.axes.Axes.set_ylabel

set_ylim(*self*, *ymin*=None, *ymax*=None, *emit*=True)

set the ylimits ACCEPTS: len(2) sequence of floats

Overrides: matplotlib.axes.Axes.set_ylim

table(*self*, **args*, ***kwargs*)

TABLE(*args, **kwargs) Not implemented for polar axes

Overrides: matplotlib.axes.Axes.table

toggle_log_lineary(*self*)

toggle between log and linear axes ignored for polar

Overrides: matplotlib.axes.Axes.toggle_log_lineary

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

Inherited from Axes: acorr, add_artist, add_collection, add_line, add_patch, add_table, annotate, apply_aspect, arrow, axhline, axhspan, axis, axvline, axvspan, bar, barh, boxplot, broken_barh, clabel, clear, cohere, connect, contour, contourf, csd, disconnect, draw_artist, errorbar, fill, format_xdata, format_ydata, get_adjustable, get_anchor, get_aspect, get_autoscale_on, get_axis_bgcolor, get_axisbelow, get_child_artists, get_cursor_props, get_frame, get_frame_on, get_images, get_legend, get_lines, get_navigate, get_navigate_mode, get_position, get_renderer_cache, get_window_extent, get_xaxis, get_xgridlines, get_xlim, get_xticklabels, get_xticklines, get_xticks, get_yaxis, get_ygridlines, get_ylim, get_yticklabels, get_yticklines, get_yticks, hist, hlines, hold, imshow, in_axes, ishold, legend, loglog, matshow, panx, pany, pcolor, pcolor_classic, pcolormesh, pick, pie, plot, plot_date, psd, quiver, quiver2, quiver_classic, quiverkey, redraw_in_frame, relim, scatter, scatter_classic, semilogx, semilogy, set_adjustable, set_anchor, set_aspect, set_autoscale_on, set_axis_bgcolor, set_axis_off, set_axis_on, set_axisbelow, set_cursor_props, set_figure, set_frame_on, set_navigate, set_navigate_mode, set_position, set_title, set_xscale, set_xticklabels, set_xticks, set_yscale, set_yticklabels, set_yticks, specgram, spy, stem, text, ticklabel_format, update_datalim, update_datalim_numerix, vlines, xaxis_date, xcorr, yaxis_date, zoomx, zoomy

6.3.2 Class Variables

Name	Description
RESOLUTION	**Value:** `100` *(type=`int`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)* **Inherited from Axes:** scaled *(p. 99)*	

6.4 Class PolarSubplot

matplotlib.artist.Artist

matplotlib.axes.Axes

matplotlib.axes.PolarAxes

matplotlib.axes.SubplotBase

PolarSubplot

```
Create a polar subplot with

  PolarSubplot(numRows, numCols, plotNum)

where plotNum=1 is the first plot number and increasing plotNums
fill rows first.  max(plotNum)==numRows*numCols

You can leave out the commas if numRows<=numCols<=plotNum<10, as
in

  Subplot(211)    # 2 rows, 1 column, first (upper) plot
```

6.4.1 Methods

__init__(*self*, *fig*, **args*, ***kwargs*)

fig is a figure instance
args is a varargs to specify the subplot

Overrides: matplotlib.axes.SubplotBase.__init__ extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Axes: acorr, add_artist, add_collection, add_line, add_patch, add_table, annotate, apply_aspect, arrow, axhline, axhspan, axis, axvline, axvspan, bar, barh, boxplot, broken_barh, clabel, clear, cohere, connect, contour, contourf, csd, disconnect, draw_artist, errorbar, fill, format_xdata, format_ydata,

get_adjustable, get_anchor, get_aspect, get_autoscale_on, get_axis_bgcolor, get_axisbelow, get_child_artists, get_cursor_props, get_frame, get_frame_on, get_images, get_legend, get_lines, get_navigate, get_navigate_mode, get_position, get_renderer_cache, get_window_extent, get_xaxis, get_xgridlines, get_xlim, get_xticklabels, get_xticklines, get_xticks, get_yaxis, get_ygridlines, get_ylim, get_yticklabels, get_yticklines, get_yticks, hist, hlines, hold, imshow, in_axes, ishold, legend, loglog, matshow, panx, pany, pcolor, pcolor_classic, pcolormesh, pick, pie, plot, plot_date, psd, quiver, quiver2, quiver_classic, quiverkey, redraw_in_frame, relim, scatter, scatter_classic, semilogx, semilogy, set_adjustable, set_anchor, set_aspect, set_autoscale_on, set_axis_bgcolor, set_axis_off, set_axis_on, set_axisbelow, set_cursor_props, set_figure, set_frame_on, set_navigate, set_navigate_mode, set_position, set_title, set_xscale, set_xticklabels, set_xticks, set_yscale, set_yticklabels, set_yticks, specgram, spy, stem, text, ticklabel_format, update_datalim, update_datalim_numerix, vlines, xaxis_date, xcorr, yaxis_date, zoomx, zoomy
Inherited from PolarAxes: autoscale_view, cla, draw, format_coord, get_children, get_rmax, get_xscale, get_yscale, grid, has_data, regrid, set_rgrids, set_rmax, set_thetagrids, set_xlabel, set_xlim, set_ylabel, set_ylim, table, toggle_log_lineary
Inherited from SubplotBase: change_geometry, get_geometry, is_first_col, is_first_row, is_last_col, is_last_row, label_outer, update_params

6.4.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	
Inherited from Axes: scaled *(p. 99)*	
Inherited from PolarAxes: RESOLUTION *(p. 164)*	

6.5 Class Subplot

matplotlib.artist.Artist

matplotlib.axes.Axes

matplotlib.axes.SubplotBase

Subplot

```
Emulate matlab's(TM) subplot command, creating axes with

  Subplot(numRows, numCols, plotNum)

where plotNum=1 is the first plot number and increasing plotNums
fill rows first.  max(plotNum)==numRows*numCols

You can leave out the commas if numRows<=numCols<=plotNum<10, as
in

  Subplot(211)    # 2 rows, 1 column, first (upper) plot
```

6.5.1 Methods

__init__(*self*, *fig*, **args*, ***kwargs*)

See Axes base class documentation for args and kwargs

Overrides: matplotlib.axes.SubplotBase.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Axes: acorr, add_artist, add_collection, add_line, add_patch, add_table, annotate, apply_aspect, arrow, autoscale_view, axhline, axhspan, axis, axvline, axvspan, bar, barh, boxplot, broken_barh, cla, clabel, clear, cohere, connect, contour, contourf, csd, disconnect, draw, draw_artist, errorbar, fill, format_coord, format_xdata, format_ydata, get_adjustable, get_anchor, get_aspect, get_autoscale_on, get_axis_bgcolor, get_axisbelow, get_child_artists, get_children, get_cursor_props, get_frame, get_frame_on, get_images, get_legend, get_lines, get_navigate, get_navigate_mode, get_position, get_renderer_cache, get_window_extent, get_xaxis, get_xgridlines, get_xlim, get_xscale, get_xticklabels, get_xticklines, get_xticks, get_yaxis, get_ygridlines, get_ylim, get_yscale, get_yticklabels, get_yticklines, get_yticks, grid, has_data, hist, hlines, hold, imshow, in_axes, ishold, legend, loglog, matshow, panx, pany, pcolor, pcolor_classic, pcolormesh, pick, pie, plot, plot_date, psd, quiver, quiver2, quiver_classic, quiverkey, redraw_in_frame, relim, scatter, scatter_classic, semilogx, semilogy, set_adjustable, set_anchor, set_aspect, set_autoscale_on, set_axis_bgcolor, set_axis_off, set_axis_on, set_axisbelow, set_cursor_props, set_figure, set_frame_on, set_navigate, set_navigate_mode, set_position, set_title, set_xlabel, set_xlim, set_xscale, set_xticklabels, set_xticks, set_ylabel, set_ylim, set_yscale, set_yticklabels, set_yticks, specgram, spy, stem, table, text, ticklabel_format, toggle_log_lineary, update_datalim, update_datalim_numerix, vlines, xaxis_date, xcorr, yaxis_date, zoomx, zoomy
Inherited from SubplotBase: change_geometry, get_geometry, is_first_col, is_first_row, is_last_col, is_last_row, label_outer, update_params

6.5.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	
Inherited from Axes: scaled *(p. 99)*	

6.6 Class SubplotBase

Known Subclasses: PolarSubplot, Subplot

```
Emulate matlab's(TM) subplot command, creating axes with

  Subplot(numRows, numCols, plotNum)

where plotNum=1 is the first plot number and increasing plotNums
fill rows first.  max(plotNum)==numRows*numCols
```

```
You can leave out the commas if numRows<=numCols<=plotNum<10, as
in

  Subplot(211)    # 2 rows, 1 column, first (upper) plot
```

6.6.1 Methods

__init__(*self*, *fig*, **args*)

fig is a figure instance
args is a varargs to specify the subplot

change_geometry(*self*, *numrows*, *numcols*, *num*)

change subplot geometry, eg from 1,1,1 to 2,2,3

get_geometry(*self*)

get the subplot geometry, eg 2,2,3

is_first_col(*self*)

is_first_row(*self*)

is_last_col(*self*)

is_last_row(*self*)

label_outer(*self*)

set the visible property on ticklabels so xticklabels are visible only if the subplot is in the last row and yticklabels are visible only if the subplot is in the first column

update_params(*self*)

update the subplot position from fig.subplotpars

7 Module matplotlib.axes3d

3D projection glued onto 2D Axes.

Axes3D

7.1 Functions

get_test_data(*delta*=0.050000000000000003)

sensible_format_data(*self*, *value*)

Used to generate more comprehensible numbers in status bar

test_bar2D()

test_contour()

test_plot()

test_polys()

test_scatter()

test_scatter2D()

test_surface()

test_wire()

7.2 Class Axes3D

Wrapper for Axes3DI

Provides set_xlim, set_ylim etc.

2D functions can be caught here and mapped to their 3D approximations.

This should probably be the case for plot etc...

7.2.1 Methods

__init__(*self*, *fig*, **args*, ***kwargs*)

__getattr__(*self*, *k*)

__setattr__(*self*, *k*, *v*)

add_3DCollection(*self*, *patches*)

add_collection(*self*, *polys*, *zs*=`None`, *dir*=`'z'`)

bar(*self*, *left*, *height*, *z*=`0`, *dir*=`'z'`, **args*, ***kwargs*)

scatter(*self*, *xs*, *ys*, *zs*=`None`, *dir*=`'z'`, **args*, ***kwargs*)

set_xlim(*self*, **args*, ***kwargs*)

set_ylim(*self*, **args*, ***kwargs*)

set_zlim(*self*, **args*, ***kwargs*)

text(*self*, *x*, *y*, *text*, **args*, ***kwargs*)

7.3 Class Axes3DI

matplotlib.artist.Artist
    matplotlib.axes.Axes
        Axes3DI

Wrap an Axes object

The x,y data coordinates, which are manipulated by set_xlim and set_ylim are used as the target view coordinates by the 3D transformations. These coordinates are mostly invisible to the outside world.

set_w_xlim, set_w_ylim and set_w_zlim manipulate the 3D world coordinates which are scaled to represent the data and are stored in the xy_dataLim, zz_datalim bboxes.

The axes representing the x,y,z world dimensions are self.w_xaxis, self.w_yaxis and self.w_zaxis. They can probably be controlled in more or less the normal ways.

7.3.1 Methods

__init__(*self*, *fig*, *rect*=`[0.0, 0.0, 1.0, 1.0]`, **args*, ***kwargs*)
Overrides: matplotlib.axes.Axes.__init__

add_lines(*self*, *lines*, **args*, ***kwargs*)

ahvline(*self*, *x*, *y*)

ahvxplane(*self*, *x*)

ahvyplane(*self*, *y*)

auto_scale_xyz(*self*, *X*, *Y*, *Z*=`None`, *had_data*=`None`)

autoscale_view(*self*, *scalex*=`True`, *scaley*=`True`, *scalez*=`True`)
Overrides: matplotlib.axes.Axes.autoscale_view

button_press(*self*, *event*)

button_release(*self*, *event*)

clabel(*self*, **args*, ***kwargs*)
Overrides: matplotlib.axes.Axes.clabel

contour3D(*self*, *X*, *Y*, *Z*, **args*, ***kwargs*)

contourf3D(*self*, *X*, *Y*, *Z*, **args*, ***kwargs*)

create_axes(*self*)

draw(*self*, *renderer*)
Overrides: matplotlib.axes.Axes.draw

format_coord(*self*, *xd*, *yd*)

Given the 2D view coordinates attempt to guess a 3D coordinate
Looks for the nearest edge to the point and then assumes that the point is at the same z location as the nearest point on the edge.

Overrides: matplotlib.axes.Axes.format_coord

format_xdata(*self*, *x*)

Return x string formatted. This function will use the attribute self.fmt_xdata if it is callable, else will fall back on the xaxis major formatter

Overrides: matplotlib.axes.Axes.format_xdata

format_ydata(*self*, *y*)

Return y string formatted. This function will use the attribute self.fmt_ydata if it is callable, else will fall back on the yaxis major formatter

Overrides: matplotlib.axes.Axes.format_ydata

format_zdata(*self*, *z*)

Return y string formatted. This function will use the attribute self.fmt_ydata if it is callable, else will fall back on the yaxis major formatter

get_axis_position(*self*)

get_proj(*self*)

Create the projection matrix from the current viewing position.
elev stores the elevation angle in the z plane azim stores the azimuth angle in the x,y plane
dist is the distance of the eye viewing point from the object point.

get_w_lims(*self*)

get_w_xlim(*self*)

get_w_ylim(*self*)

get_w_zlim(*self*)

mouse_init(*self*)

nset_xlim(*self*, **args*)

nset_ylim(*self*, **args*)

on_move(*self*, *event*)

Mouse moving
button-1 rotates button-3 zooms

panx(*self*, *numsteps*)

Pan the x axis numsteps (plus pan right, minus pan left)

Overrides: matplotlib.axes.Axes.panx extit(inherited documentation)

pany(*self*, *numsteps*)

Pan the x axis numsteps (plus pan up, minus pan down)

Overrides: matplotlib.axes.Axes.pany extit(inherited documentation)

plot(*self*, **args*, ***kwargs*)

```
PLOT(*args, **kwargs)
Plot lines and/or markers to the Axes.  *args is a variable length
argument, allowing for multiple x,y pairs with an optional format
string.  For example, each of the following is legal
    plot(x,y)            # plot x and y using the default line style and color
    plot(x,y, 'bo')      # plot x and y using blue circle markers
    plot(y)              # plot y using x as index array 0..N-1
    plot(y, 'r+')        # ditto, but with red plusses
If x and/or y is 2-Dimensional, then the corresponding columns
will be plotted.
An arbitrary number of x, y, fmt groups can be specified, as in
a.plot(x1, y1, 'g^', x2, y2, 'g-')
Return value is a list of lines that were added.
The following line styles are supported:
    -     : solid line
    --    : dashed line
    -.    : dash-dot line
    :     : dotted line
    .     : points
    ,     : pixels
    o     : circle symbols
    ^     : triangle up symbols
    v     : triangle down symbols
    <     : triangle left symbols
    >     : triangle right symbols
    s     : square symbols
    +     : plus symbols
    x     : cross symbols
    D     : diamond symbols
    d     : thin diamond symbols
    1     : tripod down symbols
    2     : tripod up symbols
    3     : tripod left symbols
    4     : tripod right symbols
    h     : hexagon symbols
    H     : rotated hexagon symbols
    p     : pentagon symbols
    |     : vertical line symbols
    _     : horizontal line symbols
    steps : use gnuplot style 'steps' # kwarg only
The following color abbreviations are supported
    b  : blue
    g  : green
    r  : red
    c  : cyan
    m  : magenta
    y  : yellow
    k  : black
    w  : white
In addition, you can specify colors in many weird and
wonderful ways, including full names 'green', hex strings
'#008000', RGB or RGBA tuples (0,1,0,1) or grayscale
intensities as a string '0.8'.
Line styles and colors are combined in a single format string, as in
'bo' for blue circles.
The **kwargs can be used to set line properties (any property that has
```

plot3D(*self*, *xs*, *ys*, *zs*, **args*, ***kwargs*)

plot3d(*self*, *xs*, *ys*, *zs*, **args*, ***kwargs*)

plot_surface(*self*, *X*, *Y*, *Z*, **args*, ***kwargs*)

plot_wireframe(*self*, *X*, *Y*, *Z*, **args*, ***kwargs*)

really_set_xlim(*self*, *vmin*, *vmax*)

really_set_ylim(*self*, *vmin*, *vmax*)

scatter3D(*self*, *xs*, *ys*, *zs*, **args*, ***kwargs*)

scatter3d(*self*, *xs*, *ys*, *zs*, **args*, ***kwargs*)

set_top_view(*self*)

set_w_xlim(*self*, **args*, ***kwargs*)

set_w_ylim(*self*, **args*, ***kwargs*)

set_w_zlim(*self*, **args*, ***kwargs*)

set_xlabel(*self*, *xlabel*, *fontdict*=None, ***kwargs*)

```
SET_XLABEL(xlabel, fontdict=None, **kwargs)
Set the label for the xaxis.  See the text docstring for information
of how override and the optional args work.
Valid kwargs are Text properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: str
```

Overrides: matplotlib.axes.Axes.set_xlabel extit(inherited documentation)

set_ylabel(*self*, *ylabel*, *fontdict*=None, ***kwargs*)

```
SET_YLABEL(ylabel, fontdict=None, **kwargs)
Set the label for the yaxis
See the text doctstring for information of how override and
the optional args work
Valid kwargs are Text properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
ACCEPTS: str
```

Overrides: matplotlib.axes.Axes.set_ylabel extit(inherited documentation)

set_zlabel(*self*, *zlabel*, *fontdict*=None, ***kwargs*)

text3D(*self*, *x*, *y*, *z*, *s*, **args*, ***kwargs*)

tunit_cube(*self*, *vals*=None, *M*=None)

tunit_edges(*self*, *vals*=`None`, *M*=`None`)

unit_cube(*self*, *vals*=`None`)

update_datalim(*self*, *xys*)

Update the data lim bbox with seq of xy tups or equiv. 2-D array

Overrides: matplotlib.axes.Axes.update_datalim extit(inherited documentation)

update_datalim_numerix(*self*, *x*, *y*)

Update the data lim bbox with seq of xy tups

Overrides: matplotlib.axes.Axes.update_datalim_numerix extit(inherited documentation)

view_init(*self*, *elev*, *azim*)

vlim_argument(*self*, *get_lim*, **args*)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Axes: acorr, add_artist, add_collection, add_line, add_patch, add_table, annotate, apply_aspect, arrow, axhline, axhspan, axis, axvline, axvspan, bar, barh, boxplot, broken_barh, cla, clear, cohere, connect, contour, contourf, csd, disconnect, draw_artist, errorbar, fill, get_adjustable, get_anchor, get_aspect, get_autoscale_on, get_axis_bgcolor, get_axisbelow, get_child_artists, get_children, get_cursor_props, get_frame, get_frame_on, get_images, get_legend, get_lines, get_navigate, get_navigate_mode, get_position, get_renderer_cache, get_window_extent, get_xaxis, get_xgridlines, get_xlim, get_xscale, get_xticklabels, get_xticklines, get_xticks, get_yaxis, get_ygridlines, get_ylim, get_yscale, get_yticklabels, get_yticklines, get_yticks, grid, has_data, hist, hlines, hold, imshow, in_axes, ishold, legend, loglog, matshow, pcolor, pcolor_classic, pcolormesh, pick, pie, plot_date, psd, quiver, quiver2, quiver_classic, quiverkey, redraw_in_frame, relim, scatter, scatter_classic, semilogx, semilogy, set_adjustable, set_anchor, set_aspect, set_autoscale_on, set_axis_bgcolor, set_axis_off, set_axis_on, set_axisbelow, set_cursor_props, set_figure, set_frame_on, set_navigate, set_navigate_mode, set_position, set_title, set_xlim, set_xscale, set_xticklabels, set_xticks, set_ylim, set_yscale, set_yticklabels, set_yticks, specgram, spy, stem, table, text, ticklabel_format, toggle_log_lineary, vlines, xaxis_date, xcorr, yaxis_date, zoomx, zoomy

7.3.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	
Inherited from Axes: scaled *(p. 99)*	

7.4 Class Scaler

7.4.1 Methods

__init__(*self*, *points*)

update(*self*, *lims*)

8 Module matplotlib.axis

Classes for the ticks and x and y axis

8.1 Class Axis

matplotlib.artist.Artist
Axis

Known Subclasses: XAxis, YAxis

```
Public attributes
  transData - transform data coords to display coords
  transAxis - transform axis coords to display coords
```

8.1.1 Methods

__init__(*self*, *axes*)

Init the axis with the parent Axes instance

Overrides: matplotlib.artist.Artist.__init__

cla(*self*)

clear the current axis

convert_units(*self*, *x*)

draw(*self*, *renderer*, **args*, ***kwargs*)

Draw the axis lines, grid lines, tick lines and labels

Overrides: matplotlib.artist.Artist.draw

get_children(*self*)

get_data_interval(*self*)

return the Interval instance for this axis data limits

get_gridlines(*self*)

Return the grid lines as a list of Line2D instance

get_label(*self*)

Return the axis label as a Text instance

Overrides: matplotlib.artist.Artist.get_label

get_major_formatter(*self*)

Get the formatter of the major ticker

get_major_locator(*self*)

Get the locator of the major ticker

get_major_ticks(*self*)

get the tick instances; grow as necessary

get_minor_formatter(*self*)

Get the formatter of the minor ticker

get_minor_locator(*self*)

Get the locator of the minor ticker

get_minor_ticks(*self*)

get the minor tick instances; grow as necessary

get_offset_text(*self*)

Return the axis offsetText as a Text instance

get_ticklabels(*self*)

Return a list of Text instances for ticklabels

get_ticklines(*self*)

Return the ticklines lines as a list of Line2D instance

get_ticklocs(*self*)

Get the tick locations in data coordinates as a Numeric array

get_units(*self*)

return the units for axis

get_view_interval(*self*)

return the Interval instance for this axis view limits

grid(*self*, *b*=None, *which*='major', ***kwargs*)

```
Set the axis grid on or off; b is a boolean use which =
'major' | 'minor' to set the grid for major or minor ticks

if b is None and len(kwargs)==0, toggle the grid state.  If
kwargs are supplied, it is assumed you want the grid on and b
will be set to True

kwargs are used to set the line properties of the grids, eg,

  xax.grid(color='r', linestyle='-', linewidth=2)
```

have_units(*self*)

return True if units are set on the x or y axes

Overrides: matplotlib.artist.Artist.have_units extit(inherited documentation)

pan(*self*, *numsteps*)

Pan numticks (can be positive or negative)

pick(*self*, *mouseevent*)

pick(mouseevent)
each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

Overrides: matplotlib.artist.Artist.pick

set_major_formatter(*self*, *formatter*)

Set the formatter of the major ticker
ACCEPTS: A Formatter instance

set_major_locator(*self*, *locator*)

Set the locator of the major ticker
ACCEPTS: a Locator instance

set_minor_formatter(*self*, *formatter*)

Set the formatter of the minor ticker
ACCEPTS: A Formatter instance

set_minor_locator(*self*, *locator*)

Set the locator of the minor ticker
ACCEPTS: a Locator instance

set_ticklabels(*self*, *ticklabels*, **args*, ***kwargs*)

Set the text values of the tick labels. Return a list of Text instances.
ACCEPTS: sequence of strings

set_ticks(*self*, *ticks*)

Set the locations of the tick marks from sequence ticks
ACCEPTS: sequence of floats

set_units(*self*, *u*)

set the units for axis
ACCEPTS: a units tag

update_units(*self*, *data*)

introspect data for units converter and update the axis.converter instance if necessary. Return true is data is registered for unit conversion

zoom(*self*, *direction*)

Zoom in/out on axis; if direction is >0 zoom in, else zoom out

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_picker, get_transform, get_visible, get_zorder, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

8.1.2 Class Variables

Name	Description
LABELPAD	**Value:** `5` *(type=`int`)*
OFFSETTEXTPAD	**Value:** `3` *(type=`int`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

8.2 Class Tick

matplotlib.artist.Artist
    Tick

Known Subclasses: XTick, YTick

```
Abstract base class for the axis ticks, grid lines and labels

1 refers to the bottom of the plot for xticks and the left for yticks
```

```
2 refers to the top of the plot for xticks and the right for yticks

Publicly accessible attributes

  tick1line  : a Line2D instance
  tick2line  : a Line2D instance
  gridline   : a Line2D instance
  label1     : a Text instance
  label2     : a Text instance
  gridOn     : a boolean which determines whether to draw the tickline
  tick1On    : a boolean which determines whether to draw the 1st tickline
  tick2On    : a boolean which determines whether to draw the 2nd tickline
  label1On   : a boolean which determines whether to draw tick label
  label2On   : a boolean which determines whether to draw tick label
```

8.2.1 Methods

__init__(*self*, *axes*, *loc*, *label*, *size*=`None`, *gridOn*=`None`, *tick1On*=`True`, *tick2On*=`True`, *label1On*=`True`, *label2On*=`False`, *major*=`True`)

bbox is the Bound2D bounding box in display coords of the Axes loc is the tick location in data coords size is the tick size in relative, axes coords

Overrides: matplotlib.artist.Artist.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_children(*self*)

get_loc(*self*)

Return the tick location (data coords) as a scalar

get_pad(*self*, *val*)

Get the value of the tick label pad in points

get_view_interval(*self*)

return the view Interval instance for the axis tjis tick is ticking

pick(*self*, *mouseevent*)

pick(mouseevent)
each child artist will fire a pick event if mouseevent is over the artist and the artist has picker set

Overrides: matplotlib.artist.Artist.pick

set_label(*self*, *s*)

Set the text of ticklabel
ACCEPTS: str

Overrides: matplotlib.artist.Artist.set_label

set_label1(*self*, *s*)

Set the text of ticklabel
ACCEPTS: str

set_label2(*self*, *s*)

Set the text of ticklabel2
ACCEPTS: str

set_pad(*self*, *val*)

Set the tick label pad in points
ACCEPTS: float

set_xy(*self*, *loc*)

Set the location of tick in data coords with scalar loc
ACCEPTS: float

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

8.2.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

8.3 Class Ticker

8.3.1 Class Variables

Name	Description
formatter	**Value:** `None` *(type=`NoneType`)*
locator	**Value:** `None` *(type=`NoneType`)*

8.4 Class XAxis

matplotlib.artist.Artist
matplotlib.axis.Axis
XAxis

Known Subclasses: Axis

8.4.1 Methods

get_data_interval(*self*)

return the Interval instance for this axis data limits

Overrides: matplotlib.axis.Axis.get_data_interval

get_label_position(*self*)

Return the label position (top or bottom)

get_ticks_position(*self*)

Return the ticks position (top, bottom, default or unknown)

get_view_interval(*self*)

return the Interval instance for this axis view limits

Overrides: matplotlib.axis.Axis.get_view_interval

set_label_position(*self*, *position*)

Set the label position (top or bottom)
ACCEPTS: ['top' | 'bottom']

set_ticks_position(*self*, *position*)

Set the ticks position (top, bottom, both or default) both sets the ticks to appear on both positions, but does not change the tick labels. default resets the tick positions to the default: ticks on both positions, labels at bottom.
ACCEPTS: ['top' | 'bottom' | 'both' | 'default']

tick_bottom(*self*)

use ticks only on bottom

tick_top(*self*)

use ticks only on top

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_picker, get_transform, get_visible, get_zorder, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Axis: __init__, cla, convert_units, draw, get_children, get_gridlines, get_label, get_major_formatter, get_major_locator, get_major_ticks, get_minor_formatter, get_minor_locator, get_minor_ticks, get_offset_text, get_ticklabels, get_ticklines, get_ticklocs, get_units, grid, have_units, pan, pick, set_major_formatter, set_major_locator, set_minor_formatter, set_minor_locator, set_ticklabels, set_ticks, set_units, update_units, zoom

8.4.2 Class Variables

Name	Description
__name__	**Value:** `'XAxis'` *(type=`str`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	
Inherited from Axis: LABELPAD *(p. 183)*, OFFSETTEXTPAD *(p. 183)*	

8.5 Class XTick

matplotlib.artist.Artist
matplotlib.axis.Tick
XTick

Contains all the Artists needed to make an x tick - the tick line, the label text and the grid line

8.5.1 Methods

get_data_interval(*self*)

return the Interval instance for this axis data limits

get_view_interval(*self*)

return the Interval instance for this axis view limits

Overrides: matplotlib.axis.Tick.get_view_interval

update_position(*self*, *loc*)

Set the location of tick in data coords with scalar loc

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder,

have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Tick: __init__, draw, get_children, get_loc, get_pad, pick, set_label, set_label1, set_label2, set_pad, set_xy

8.5.2 Class Variables

Name	Description
__name__	**Value:** `'XTick'` *(type=`str`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

8.6 Class YAxis

matplotlib.artist.Artist
  matplotlib.axis.Axis
    YAxis

8.6.1 Methods

get_data_interval(*self*)

return the Interval instance for this axis data limits

Overrides: matplotlib.axis.Axis.get_data_interval

get_label_position(*self*)

Return the label position (left or right)

get_ticks_position(*self*)

Return the ticks position (left, right, both or unknown)

get_view_interval(*self*)

return the Interval instance for this axis view limits

Overrides: matplotlib.axis.Axis.get_view_interval

set_label_position(*self*, *position*)

Set the label position (left or right)
ACCEPTS: ['left' | 'right']

set_offset_position(*self*, *position*)

set_ticks_position(*self*, *position*)

Set the ticks position (left, right, both or default) both sets the ticks to appear on both positions, but does not change the tick labels. default resets the tick positions to the default: ticks on both positions, labels on the left.
ACCEPTS: ['left' | 'right' | 'both' | 'default']

tick_left(*self*)

use ticks only on left

tick_right(*self*)

use ticks only on right

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_picker, get_transform, get_visible, get_zorder, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Axis: __init__, cla, convert_units, draw, get_children, get_gridlines, get_label, get_major_formatter, get_major_locator, get_major_ticks, get_minor_formatter, get_minor_locator, get_minor_ticks, get_offset_text, get_ticklabels, get_ticklines, get_ticklocs, get_units, grid, have_units, pan, pick, set_major_formatter, set_major_locator, set_minor_formatter, set_minor_locator, set_ticklabels, set_ticks, set_units, update_units, zoom

8.6.2 Class Variables

Name	Description
__name__	**Value:** `'YAxis'` *(type=`str`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)* **Inherited from Axis:** LABELPAD *(p. 183)*, OFFSETTEXTPAD *(p. 183)*	

8.7 Class YTick

matplotlib.artist.Artist
  matplotlib.axis.Tick
    YTick

Contains all the Artists needed to make a Y tick - the tick line, the label text and the grid line

8.7.1 Methods

get_data_interval(*self*)

return the Interval instance for this axis data limits

get_view_interval(*self*)

return the Interval instance for this axis view limits

Overrides: matplotlib.axis.Tick.get_view_interval

update_position(*self*, *loc*)

Set the location of tick in data coords with scalar loc

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Tick: __init__, draw, get_children, get_loc, get_pad, pick, set_label, set_label1, set_label2, set_pad, set_xy

8.7.2 Class Variables

Name	Description
__name__	**Value:** `'YTick'` *(type=*`str`*)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

9 Module matplotlib.axis3d

9.1 Functions

norm_angle(*a*)

Return angle between -180 and +180

text_update_coords(*self*, *renderer*)

Modified method update_coords from TextWithDash
I could not understand the original text offset calculations and it gave bad results for the angles I was using. This looks better, although the text bounding boxes look a little inconsistent

tick_update_position(*tick*, *x*, *y*, *z*, *angle*)

9.2 Class Axis

matplotlib.artist.Artist
- matplotlib.axis.Axis
  - matplotlib.axis.XAxis
    - **Axis**

9.2.1 Methods

__init__(*self*, *adir*, *v_intervalx*, *d_intervalx*, *axes*, **args*, ***kwargs*)
Overrides: matplotlib.axis.Axis.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.axis.Axis.draw

get_data_interval(*self*)

return the Interval instance for this axis data limits

Overrides: matplotlib.axis.XAxis.get_data_interval

get_major_ticks(*self*)

get the tick instances; grow as necessary

Overrides: matplotlib.axis.Axis.get_major_ticks extit(inherited documentation)

get_tick_positions(*self*)

get_view_interval(*self*)

return the Interval instance for this axis view limits

Overrides: matplotlib.axis.XAxis.get_view_interval

set_pane_bg(*self*, *xys*)

set_pane_fg(*self*, *xys*)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_picker, get_transform, get_visible, get_zorder, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from Axis: cla, convert_units, get_children, get_gridlines, get_label, get_major_formatter, get_major_locator, get_minor_formatter, get_minor_locator, get_minor_ticks, get_offset_text, get_ticklabels, get_ticklines, get_ticklocs, get_units, grid, have_units, pan, pick, set_major_formatter, set_major_locator, set_minor_formatter, set_minor_locator, set_ticklabels, set_ticks, set_units, update_units, zoom
Inherited from XAxis: get_label_position, get_ticks_position, set_label_position, set_ticks_position, tick_bottom, tick_top

9.2.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	
Inherited from Axis: LABELPAD *(p. 183)*, OFFSETTEXTPAD *(p. 183)*	
Inherited from XAxis: __name__ *(p. 189)*	

10 Module matplotlib.backend_bases

Abstract base classes define the primitives that renderers and graphics contexts must implement to serve as a matplotlib backend

10.1 Class Cursors

10.1.1 Class Variables

Name	Description
HAND	**Value:** 0 *(type=int)*
MOVE	**Value:** 3 *(type=int)*
POINTER	**Value:** 1 *(type=int)*
SELECT_REGION	**Value:** 2 *(type=int)*

10.2 Class DrawEvent

matplotlib.backend_bases.Event ─┐
DrawEvent

```
An event triggered by a draw operation on the canvas

Attributes are
  name
  canvas
  renderer - the Renderer instance
```

10.2.1 Methods

__init__(*self*, *name*, *canvas*, *renderer*) Overrides: matplotlib.backend_bases.Event.__init__

10.3 Class Event

Known Subclasses: DrawEvent, LocationEvent, PickEvent, ResizeEvent

A matplotlib event. Attach additional attributes as defined in FigureCanvas.connect. The following attributes are defined and shown with their default values name # the event name canvas # the FigureCanvas instance generating the event

10.3.1 Methods

__init__(*self*, *name*, *canvas*, *guiEvent*=None)

10.4 Class FigureCanvasBase

Known Subclasses: FigureCanvasAgg, FigureCanvasQT

```
The canvas the figure renders into.

Public attribute

  figure - A Figure instance
```

10.4.1 Methods

__init__(*self*, *figure*)

---

blit(*self*, *bbox*=`None`)

blit the canvas in bbox (default entire canvas)

---

button_press_event(*self*, *x*, *y*, *button*, *guiEvent*=`None`)

Backend derived classes should call this function on any mouse button press. x,y are the canvas coords: 0,0 is lower, left. button and key are as defined in MouseEvent

---

button_release_event(*self*, *x*, *y*, *button*, *guiEvent*=`None`)

Backend derived classes should call this function on any mouse button release. x,y are the canvas coords: 0,0 is lower, left. button and key are as defined in MouseEvent

---

draw(*self*, **args*, ***kwargs*)

Render the figure

---

draw_cursor(*self*, *event*)

Draw a cursor in the event.axes if inaxes is not None. Use native GUI drawing for efficiency if possible

---

draw_event(*self*, *renderer*)

---

draw_idle(*self*, **args*, ***kwargs*)

draw only if idle; defaults to draw but backends can overrride

---

get_width_height(*self*)

return the figure width and height in points or pixels (depending on the backend), truncated to integers

---

key_press_event(*self*, *key*, *guiEvent*=`None`)

key_release_event(*self*, *key*, *guiEvent*=None)

motion_notify_event(*self*, *x*, *y*, *guiEvent*=None)

Backend derived classes should call this function on any motion-notify-event. x,y are the canvas coords: 0,0 is lower, left. button and key are as defined in MouseEvent

mpl_connect(*self*, *s*, *func*)

```
Connect event with string s to func.  The signature of func is

  def func(event)

where event is a MplEvent.  The following events are recognized

'resize_event',
'draw_event',
'key_press_event',
'key_release_event',
'button_press_event',
'button_release_event',
'motion_notify_event',
'pick_event',


For the three events above, if the mouse is over the axes,
the variable event.inaxes will be set to the axes it is over,
and additionally, the variables event.xdata and event.ydata
will be defined.  This is the mouse location in data coords.
See backend_bases.MplEvent.

return value is a connection id that can be used with
mpl_disconnect
```

mpl_disconnect(*self*, *cid*)

disconnect callback id cid

pick_event(*self*, *mouseevent*, *artist*, ***kwargs*)

This method will be called by artists who are picked and will fire off PickEvent callbacks registered listeners

print_figure(*self*, *filename*, *dpi*=`None`, *facecolor*=`'w'`, *edgecolor*=`'w'`, *orientation*=`'portrait'`, ***kwargs*)

Render the figure to hardcopy. Set the figure patch face and edge colors. This is useful because some of the GUIs have a gray figure face color background and you'll probably want to override this on hardcopy. filename - can also be a file object on image backends orientation - only currently applies to PostScript printing. dpi - the dots per inch to save the figure in; if None, use savefig.dpi facecolor - the facecolor of the figure edgecolor - the edgecolor of the figure orientation - 'landscape' | 'portrait' (not supported on all backends)

resize(*self*, *w*, *h*)

set the canvas size in pixels

resize_event(*self*)

switch_backends(*self*, *FigureCanvasClass*)

instantiate an instance of FigureCanvasClass
This is used for backend switching, eg, to instantiate a FigureCanvasPS from a FigureCanvasGTK. Note, deep copying is not done, so any changes to one of the instances (eg, setting figure size or line props), will be reflected in the other

10.4.2 Class Variables

Name	Description
events	**Value:** `('resize_event', 'draw_event', 'key_press_event', 'key_release_event', 'butto...` *(type=`tuple`)*

10.5 Class FigureManagerBase

Known Subclasses: FigureManagerQT

Helper class for matlab mode, wraps everything up into a neat bundle

Public attibutes canvas - A FigureCanvas instance num - The figure number

10.5.1 Methods

__init__(*self*, *canvas*, *num*)

destroy(*self*)

full_screen_toggle(*self*)

key_press(*self*, *event*)

resize(*self*, *w*, *h*)

For gui backends: resize window in pixels

show_popup(*self*, *msg*)

Display message in a popup – GUI only

10.6 Class GraphicsContextBase

An abstract base class that provides color, line styles, etc...

10.6.1 Methods

__init__(*self*)

copy_properties(*self*, *gc*)

Copy properties from gc to self

get_alpha(*self*)

Return the alpha value used for blending - not supported on all backends

get_antialiased(*self*)

Return true if the object should try to do antialiased rendering

get_capstyle(*self*)

Return the capstyle as a string in ('butt', 'round', 'projecting')

get_clip_path(*self*)

Return the clip path

get_clip_rectangle(*self*)

Return the clip rectangle as (left, bottom, width, height)

get_dashes(*self*)

Return the dash information as an offset dashlist tuple The dash list is a even size list that gives the ink on, ink off in pixels. See p107 of to postscript BLUEBOOK for more info
Default value is None

get_hatch(*self*)

Gets the current hatch style

get_joinstyle(*self*)

Return the line join style as one of ('miter', 'round', 'bevel')

get_linestyle(*self*, *style*)

Return the linestyle: one of ('solid', 'dashed', 'dashdot', 'dotted').

get_linewidth(*self*)

Return the line width in points as a scalar

get_rgb(*self*)

returns a tuple of three floats from 0-1. color can be a matlab format string, a html hex color string, or a rgb tuple

set_alpha(*self*, *alpha*)

Set the alpha value used for blending - not supported on all backends

set_antialiased(*self*, *b*)

True if object should be drawn with antialiased rendering

set_capstyle(*self*, *cs*)

Set the capstyle as a string in ('butt', 'round', 'projecting')

set_clip_path(*self*, *path*)

Set the clip path

set_clip_rectangle(*self*, *rectangle*)

Set the clip rectangle with sequence (left, bottom, width, height)

set_dashes(*self*, *dash_offset*, *dash_list*)

Set the dash style for the gc. dash_offset is the offset (usually 0). dash_list specifies the on-off sequence as points (None, None) specifies a solid line

set_foreground(*self*, *fg*, *isRGB*=`False`)

Set the foreground color. fg can be a matlab format string, a html hex color string, an rgb unit tuple, or a float between 0 and 1. In the latter case, grayscale is used.
The GraphicsContext converts colors to rgb internally. If you know the color is rgb already, you can set isRGB to True to avoid the performace hit of the conversion

set_graylevel(*self*, *frac*)

Set the foreground color to be a gray level with frac frac

set_hatch(*self*, *hatch*)

Sets the hatch style for filling

set_joinstyle(*self*, *js*)

Set the join style to be one of ('miter', 'round', 'bevel')

set_linestyle(*self*, *style*)

Set the linestyle to be one of ('solid', 'dashed', 'dashdot', 'dotted').

set_linewidth(*self*, *w*)

Set the linewidth in points

10.6.2 Class Variables

Name	Description
dashd	**Value:** `{'solid': (None, None), 'dashed': (0, (6.0, 6.0-)), 'dotted': (0, (1.0, 3.0)),...` *(type=`dict`)*

10.7 Class KeyEvent

matplotlib.backend_bases.Event

matplotlib.backend_bases.LocationEvent

KeyEvent

A key event (key press, key release).

Attach additional attributes as defined in FigureCanvas.connect.

The following attributes are defined and shown with their default values

x = None # x position - pixels from left of canvas y = None # y position - pixels from bottom of canvas key = None # the key pressed: None, chr(range(255), shift, win, or control inaxes = None # the Axes instance if mouse us over axes xdata = None # x coord of mouse in data coords ydata = None # y coord of mouse in data coords

This interface may change slightly when better support for modifier keys is included

10.7.1 Methods

__init__(*self*, *name*, *canvas*, *key*, *x*=`0`, *y*=`0`, *guiEvent*=`None`)
Overrides: matplotlib.backend_bases.LocationEvent.__init__

10.7.2 Class Variables

Name	Description
Inherited from LocationEvent: button *(p. 203)*, inaxes *(p. 203)*, x *(p. 203)*, xdata *(p. 203)*, y *(p. 203)*, ydata *(p. 203)*	

10.8 Class LocationEvent

matplotlib.backend_bases.Event
└ **LocationEvent**

Known Subclasses: KeyEvent, MouseEvent

A event that has a screen location

The following additional attributes are defined and shown with their default values

x = None # x position - pixels from left of canvas y = None # y position - pixels from bottom of canvas inaxes = None # the Axes instance if mouse us over axes xdata = None # x coord of mouse in data coords ydata = None # y coord of mouse in data coords

10.8.1 Methods

__init__(*self*, *name*, *canvas*, *x*, *y*, *guiEvent*=`None`)

x, y in figure coords, 0,0 = bottom, left button pressed None, 1, 2, 3

Overrides: matplotlib.backend_bases.Event.__init__

10.8.2 Class Variables

Name	Description
button	**Value:** `None` *(type=NoneType)*
inaxes	**Value:** `None` *(type=NoneType)*
x	**Value:** `None` *(type=NoneType)*
xdata	**Value:** `None` *(type=NoneType)*
y	**Value:** `None` *(type=NoneType)*
ydata	**Value:** `None` *(type=NoneType)*

10.9 Class MouseEvent

matplotlib.backend_bases.Event

matplotlib.backend_bases.LocationEvent

MouseEvent

A mouse event (button_press_event, button_release_event, motion_notify_event).

The following attributes are defined and shown with their default values

x = None # x position - pixels from left of canvas y = None # y position - pixels from bottom of canvas button = None # button pressed None, 1, 2, 3 key = None # the key pressed: None, chr(range(255), shift, win, or control inaxes = None # the Axes instance if mouse us over axes xdata = None # x coord of mouse in data coords ydata = None # y coord of mouse in data coords

10.9.1 Methods

__init__(*self*, *name*, *canvas*, *x*, *y*, *button*=`None`, *key*=`None`, *guiEvent*=`None`)

x, y in figure coords, 0,0 = bottom, left button pressed None, 1, 2, 3

Overrides: matplotlib.backend_bases.LocationEvent.__init__

10.9.2 Class Variables

Name	Description
button	**Value:** `None` *(type=NoneType)*
inaxes	**Value:** `None` *(type=NoneType)*
x	**Value:** `None` *(type=NoneType)*
xdata	**Value:** `None` *(type=NoneType)*
y	**Value:** `None` *(type=NoneType)*
ydata	**Value:** `None` *(type=NoneType)*

10.10 Class NavigationToolbar2

Known Subclasses: NavigationToolbar2QT

```
Base class for the navigation cursor, version 2
```

```
backends must implement a canvas that handles connections for
'button_press_event' and 'button_release_event'.  See
FigureCanvas.connect for more information


They must also define

 * save_figure - save the current figure

 * set_cursor - if you want the pointer icon to change

 * _init_toolbar - create your toolbar widget

 * draw_rubberband (optional) : draw the zoom to rect
   "rubberband" rectangle

* press : (optional) whenever a mouse button is pressed, you'll be
   notified with the event

* release : (optional) whenever a mouse button is released,
   you'll be notified with the event

* dynamic_update (optional) dynamically update the window while
  navigating

* set_message (optional) - display message

* set_history_buttons (optional) - you can change the history
   back / forward buttons to indicate disabled / enabled state.

That's it, we'll do the rest!
```

10.10.1 Methods

__init__(*self*, *canvas*)

---

back(*self*, **args*)

move back up the view lim stack

---

drag_pan(*self*, *event*)

the drag callback in pan/zoom mode

draw(*self*)

redraw the canvases, update the locators

draw_rubberband(*self*, *event*, *x0*, *y0*, *x1*, *y1*)

draw a rectangle rubberband to indicate zoom limits

dynamic_update(*self*)

forward(*self*, **args*)

move forward in the view lim stack

home(*self*, **args*)

restore the original view

mouse_move(*self*, *event*)

pan(*self*, **args*)

Activate the pan/zoom tool. pan with left button, zoom with right

press(*self*, *event*)

this will be called whenver a mouse button is pressed

press_pan(*self*, *event*)

the press mouse button in pan/zoom mode callback

press_zoom(*self*, *event*)

the press mouse button in zoom to rect mode callback

push_current(*self*)

push the current view limits and position onto the stack

release(*self*, *event*)

this will be called whenever mouse button is released

release_pan(*self*, *event*)

the release mouse button callback in pan/zoom mode

release_zoom(*self*, *event*)

the release mouse button callback in zoom to rect mode

save_figure(*self*, **args*)

save the current figure

set_cursor(*self*, *cursor*)

Set the current cursor to one of the backend_bases.Cursors enums values

set_history_buttons(*self*)

enable or disable back/forward button

set_message(*self*, *s*)

display a message on toolbar or in status bar

update(*self*)

reset the axes stack

zoom(*self*, **args*)

activate zoom to rect mode

10.11 Class PickEvent

matplotlib.backend_bases.Event
└─ **PickEvent**

a pick event, fired when the user picks a location on the canvas sufficiently close to an artist.

Attrs: all the Event attrs plus mouseevent : the MouseEvent that generated the pick artist : the artist picked

extra class dependent attrs – eg a Line2D pick may define different extra attributes than a PatchCollection pick event

10.11.1 Methods

__init__(*self*, *name*, *canvas*, *mouseevent*, *artist*, *guiEvent*=`None`, ***kwargs*)
Overrides: matplotlib.backend_bases.Event.__init__

10.12 Class RendererBase

An abstract base class to handle drawing/rendering operations

10.12.1 Methods

__init__(*self*)

---

close_group(*self*, *s*)

close a grouping element with label s Is only currently used by backend_svg

---

draw_arc(*self*, *gc*, *rgbFace*, *x*, *y*, *width*, *height*, *angle1*, *angle2*, *rotation*)

Draw an arc using GraphicsContext instance gcEdge, centered at x,y, with width and height and angles from 0.0 to 360.0 0 degrees is at 3-o'clock positive angles are anti-clockwise draw rotated 'rotation' degrees anti-clockwise about x,y
If the color rgbFace is not None, fill the arc with it.

---

draw_image(*self*, *x*, *y*, *im*, *bbox*)

Draw the Image instance into the current axes; x is the distance in pixels from the left hand side of the canvas. y is the distance from the origin. That is, if origin is upper, y is the distance from top. If origin is lower, y is the distance from bottom
bbox is a matplotlib.transforms.BBox instance for clipping, or None

---

draw_line(*self*, *gc*, *x1*, *y1*, *x2*, *y2*)

Draw a single line from x1,y1 to x2,y2

draw_line_collection(*self*, *segments*, *transform*, *clipbox*, *colors*, *linewidths*, *linestyle*, *antialiaseds*, *offsets*, *transOffset*)

This is a function for optimized line drawing. If you need to draw many line segments with similar properties, it is faster to avoid the overhead of all the object creation etc. The lack of total configurability is compensated for with efficiency. Hence we don't use a GC and many of the line props it supports. See matplotlib.collections for more details.
segments is a sequence of (line0, line1, line2), where linen = is an Mx2 array with columns x, y. Each line can be a different length
transform is used to Transform the lines
clipbox is a xmin, ymin, width, height clip rect
colors is a tuple of RGBA tuples
linewidths is a tuple of linewidths *** really should be called 'dashes' not 'linestyle', since we call gc.set_dashes() not gc.set_linestyle() ***
linestyle is an (offset, onoffseq) tuple or None,None for solid
antialiseds is a tuple of ones or zeros indicating whether the segment should be aa or not
offsets, if not None, is an Nx2 array of x,y offsets to translate the lines by after transform is used to transform the offset coords
This function could be overridden in the backend to possibly implement faster drawing, but it is already much faster than using draw_lines() by itself.

draw_lines(*self*, *gc*, *x*, *y*, *transform*=`None`)

x and y are equal length arrays, draw lines connecting each point in x, y

draw_point(*self*, *gc*, *x*, *y*)

Draw a single point at x,y Where 'point' is a device-unit point (or pixel), not a matplotlib point

draw_poly_collection(*self*, *verts*, *transform*, *clipbox*, *facecolors*, *edgecolors*, *linewidths*, *antialiaseds*, *offsets*, *transOffset*)

Draw a polygon collection
verts are a sequence of polygon vectors, where each polygon vector is a sequence of x,y tuples of vertices
facecolors and edgecolors are a sequence of RGBA tuples linewidths are a sequence of linewidths
antialiaseds are a sequence of 0,1 integers whether to use aa
If a linewidth is zero or an edgecolor alpha is zero, the line will be omitted; similarly, the fill will be omitted if the facecolor alpha is zero.

draw_polygon(*self*, *gc*, *rgbFace*, *points*)

Draw a polygon using the GraphicsContext instance gc. points is a len vertices tuple, each element giving the x,y coords a vertex
If the color rgbFace is not None, fill the polygon with it

draw_quad_mesh(*self*, *meshWidth*, *meshHeight*, *colors*, *xCoords*, *yCoords*, *clipbox*, *transform*, *offsets*, *transOffset*, *showedges*)

Draw a quadrilateral mesh See documentation in QuadMesh class in collections.py for details

draw_rectangle(*self*, *gcEdge*, *rgbFace*, *x*, *y*, *width*, *height*)

Draw a non-filled rectangle using the GraphicsContext instance gcEdge, with lower left at x,y with width and height.
If rgbFace is not None, fill the rectangle with it.

draw_regpoly_collection(*self*, *clipbox*, *offsets*, *transOffset*, *verts*, *sizes*, *facecolors*, *edgecolors*, *linewidths*, *antialiaseds*)

Draw a regular poly collection
offsets - is a sequence is x,y tuples transOffset - maps this to display coords
verts - are the vertices of the regular polygon at the origin
sizes are the area of the circle that circumscribes the polygon in points^2
facecolors and edgecolors are a sequence of RGBA tuples linewidths are a sequence of linewidths
antialiaseds are a sequence of 0,1 integers whether to use aa

draw_tex(*self*, *gc*, *x*, *y*, *s*, *prop*, *angle*, *ismath*=`'TeX!'`)

draw_text(*self*, *gc*, *x*, *y*, *s*, *prop*, *angle*, *ismath*=`False`)

```
Draw the text.Text instance s at x,y (display coords) with font
properties instance prop at angle in degrees, using GraphicsContext gc

**backend implementers note**

When you are trying to determine if you have gotten your bounding box
right (which is what enables the text layout/alignment to work
properly), it helps to change the line in text.py

          if 0: bbox_artist(self, renderer)

to if 1, and then the actual bounding box will be blotted along with
your text.
```

flipy(*self*)

return true if y small numbers are top for renderer Is used for drawing text (text.py) and images (image.py) only

get_canvas_width_height(*self*)

return the canvas width and height in display coords

get_image_magnification(*self*)

Get the factor by which to magnify images passed to draw_image. Allows a backend to have images at a different resolution to other artists.

get_texmanager(*self*)

get_text_extent(*self*, *text*)

Get the text extent in window coords

get_text_width_height(*self*, *s*, *prop*, *ismath*)

get the width and height in display coords of the string s with FontPropertry prop

new_gc(*self*)

Return an instance of a GraphicsContextBase

open_group(*self*, *s*)

open a grouping element with label s Is only currently used by backend_svg

option_image_nocomposite(*self*)

overwrite this method for renderers that do not necessarily want to rescale and composite raster images. (like SVG)

points_to_pixels(*self*, *points*)

Convert points to display units points - a float or a numerix array of float return points converted to pixels

You need to override this function (unless your backend doesn't have a dpi, eg, postscript or svg). Some imaging systems assume some value for pixels per inch. points to pixels = points * pixels_per_inch/72.0 * dpi/72.0

strip_math(*self*, *s*)

10.13 Class ResizeEvent

matplotlib.backend_bases.Event ─┐
ResizeEvent

```
An event triggered by a canvas resize

Attributes are
  name
```

```
canvas
width   # width of the canvas in pixels
height  # height of the canvas in pixels
```

10.13.1 Methods

__init__(*self*, *name*, *canvas*)
Overrides: matplotlib.backend_bases.Event.__init__

11 Module matplotlib.cbook

A collection of utility functions and classes. Many (but not all) from the Python Cookbook – hence the name cbook

11.1 Functions

allequal(*seq*)

return true if all elements of seq compare equal. If seq is 0 or 1 length, return True

allpairs(*x*)

return all possible pairs in sequence x
Condensed by Alex Martelli from this thread on c.l.python
http://groups.google.com/groups?q=all+pairs+group:*python*&hl=en&lr=&ie=UTF-8&selm=mailman.4028.1096403649.5135.python-list%40python.org&rnum=1

alltrue(*seq*)

dedent(*s*)

Remove excess indentation from docstrings.
Discards any leading blank lines, then removes up to n whitespace characters from each line, where n is the number of leading whitespace characters in the first line. It differs from textwrap.dedent in its deletion of leading blank lines and its use of the first non-blank line to determine the indentation.

dict_delall(*d*, *keys*)

delete all of the keys from the dict d

exception_to_str(*s*=`None`)

finddir(*o*, *match*, *case*=`False`)

return all attributes of o which match string in match. if case is True require an exact case match.

flatten(*seq*, *scalarp*=`<function is_scalar at 0x83b38b4>`)

this generator flattens nested containers such as

```
>>> l=( ('John', 'Hunter'), (1,23), [[[[42,(5,23)]]]])
```

so that

```
>>> for i in flatten(l): print i,
John Hunter 1 23 42 5 23
```

By: Composite of Holger Krekel and Luther Blissett From:
http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/121294 and Recipe 1.12 in cookbook

get_recursive_filelist(*args*)

Recurs all the files and dirs in args ignoring symbolic links and return the files as a list of strings

get_split_ind(*seq*, *N*)

seq is a list of words. Return the index into seq such that len(' '.join(seq[:ind])<=N

is_file_like(*obj*)

is_numlike(*obj*)

is_scalar(*obj*)

is_string_like(*obj*)

iterable(*obj*)

listFiles(*root*, *patterns*='*', *recurse*=1, *return_folders*=0)

Recursively list files from Parmar and Martelli in the Python Cookbook

mkdirs(*newdir*, *mode*=511)

onetrue(*seq*)

pieces(*seq*, *num*=2)

Break up the seq into num tuples

popall(*seq*)

empty a list

popd(*d*, **args*)

Should behave like python2.3 pop method; d is a dict
returns value for key and deletes item; raises a KeyError if key # is not in dict val = popd(d, key)
returns value for key if key exists, else default. Delete key, # val item if it exists. Will not raise a KeyError val = popd(d, key, default)

report_memory(*i*=0)

return the memory consumed by process

reverse_dict(*d*)

reverse the dictionary – may lose data if values are not uniq!

soundex(*name*, *len*=4)

soundex module conforming to Odell-Russell algorithm

strip_math(*s*)

remove latex formatting from mathtext

unique(*x*)

Return a list of unique elements of x

wrap(*prefix*, *text*, *cols*)

wrap text with prefix at length cols

11.2 Class Bunch

```
Often we want to just collect a bunch of stuff together, naming each
item of the bunch; a dictionary's OK for that, but a small do- nothing
class is even handier, and prettier to use.  Whenever you want to
group a few variables:

  >>> point = Bunch(datum=2, squared=4, coord=12)
  >>> point.datum

  By: Alex Martelli
  From: http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/52308
```

11.2.1 Methods

__init__(*self*, ***kwds*)

11.3 Class CallbackRegistry

```
Handle registering and disconnecting for a set of signals and
callbacks

   signals = 'eat', 'drink', 'be merry'

   def oneat(x):
       print 'eat', x
```

```
def ondrink(x):
    print 'drink', x

callbacks = CallbackRegistry(signals)

ideat = callbacks.connect('eat', oneat)
iddrink = callbacks.connect('drink', ondrink)

#tmp = callbacks.connect('drunk', ondrink) # this will raise a ValueError

callbacks.process('drink', 123)    # will call oneat
callbacks.process('eat', 456)      # will call ondrink
callbacks.process('be merry', 456) # nothing will be called
callbacks.disconnect(ideat)        # disconnect oneat
callbacks.process('eat', 456)      # nothing will be called
```

11.3.1 Methods

__init__(*self*, *signals*)

signals is a sequence of valid signals

connect(*self*, *s*, *func*)

register func to be called when a signal s is generated func will be called with args and kwargs

disconnect(*self*, *cid*)

disconnect the callback registered with callback id cid

process(*self*, *s*, **args*, ***kwargs*)

process signal s. All of the functions registered to receive callbacks on s will be called with *args and **kwargs

11.4 Class maxdict

__builtin__.object
    __builtin__.dict
        maxdict

A dictionary with a maximum size; this doesn't override all the relevant methods to contrain size, just setitem, so use with caution

11.4.1 Methods

__init__(*self*, *maxsize*)
Overrides: __builtin__.dict.__init__

__setitem__(*self*, *k*, *v*)
Overrides: __builtin__.dict.__setitem__

Inherited from dict: __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __getitem__, __gt__, __hash__, __iter__, __le__, __len__, __lt__, __ne__, __new__, __repr__, clear, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__
Inherited from type: fromkeys

11.5 Class MemoryMonitor

11.5.1 Methods

__init__(*self*, *nmax*=20000)

__call__(*self*)

clear(*self*)

plot(*self*, *i0*=0, *isub*=1)

report(*self*, *segments*=4)

xy(*self*, *i0*=0, *isub*=1)

11.6 Class Null

Null objects always and reliably "do nothing."

11.6.1 Methods

__init__(*self*, **args*, ***kwargs*)

__call__(*self*, **args*, ***kwargs*)

__delattr__(*self*, *name*)

__getattr__(*self*, *name*)

__nonzero__(*self*)

__repr__(*self*)

__setattr__(*self*, *name*, *value*)

__str__(*self*)

11.7 Class RingBuffer

class that implements a not-yet-full buffer

11.7.1 Methods

__init__(*self*, *size_max*)

__get_item__(*self*, *i*)

append(*self*, *x*)

append an element at the end of the buffer

get(*self*)

Return a list of elements from the oldest to the newest.

11.8 Class silent_list

__builtin__.object
└─ __builtin__.list
   └─ **silent_list**

override repr when returning a list of matplotlib artists to prevent long, meaningless output. This is meant to be used for a homogeneous list of a give type

11.8.1 Methods

__init__(*self*, *type*, *seq*=`None`)
Overrides: __builtin__.list.__init__

__repr__(*self*)
Overrides: __builtin__.list.__repr__

__str__(*self*)
Overrides: __builtin__.object.__str__

Inherited from list: __add__, __contains__, __delitem__, __delslice__, __eq__, __ge__, __getattribute__, __getitem__, __getslice__, __gt__, __hash__, __iadd__, __imul__, __iter__, __le__, __len__, __lt__, __mul__, __ne__, __new__, __reversed__, __rmul__, __setitem__, __setslice__, append, count, extend, index, insert, pop, remove, reverse, sort
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__

11.9 Class Sorter

Sort by attribute or item

Example usage: sort = Sorter()

list = [(1, 2), (4, 8), (0, 3)] dict = [{'a': 3, 'b': 4}, {'a': 5, 'b': 2}, {'a': 0, 'b': 0}, {'a': 9, 'b': 9}]

sort(list) # default sort sort(list, 1) # sort by index 1 sort(dict, 'a') # sort a list of dicts by key 'a'

11.9.1 Methods

__call__(*self*, *data*, *itemindex*=`None`, *inplace*=`1`)

byAttribute(*self*, *data*, *attributename*, *inplace*=`1`)

byItem(*self*, *data*, *itemindex*=`None`, *inplace*=`1`)

sort(*self*, *data*, *itemindex*=`None`, *inplace*=`1`)

11.10 Class Stack

Implement a stack where elements can be pushed on and you can move back and forth. But no pop. Should mimic home / back / forward in a browser

11.10.1 Methods

__init__(*self*, *default*=`None`)

__call__(*self*)

return the current element, or None

back(*self*)

move the position back and return the current element

bubble(*self*, *o*)

raise o to the top of the stack and return o. o must be in the stack

clear(*self*)

empty the stack

empty(*self*)

forward(*self*)

move the position forward and return the current element

home(*self*)

push the first element onto the top of the stack

push(*self*, *o*)

push object onto stack at current position - all elements occurring later than the current position are discarded

remove(*self*, *o*)

remove element o from the stack

11.11 Class Xlator

__builtin__.object
    __builtin__.dict
        Xlator

All-in-one multiple-string-substitution class

Example usage:

text = "Larry Wall is the creator of Perl" adict = { "Larry Wall" : "Guido van Rossum", "creator" : "Benevolent Dictator for Life", "Perl" : "Python", }

print multiple_replace(adict, text)

xlat = Xlator(adict) print xlat.xlat(text)

11.11.1 Methods

__call__(*self*, *match*)

Handler invoked for each regex match

xlat(*self*, *text*)

Translate text, returns the modified text.

Inherited from dict: __init__, __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __getitem__, __gt__, __hash__, __iter__, __le__, __len__, __lt__, __ne__, __new__, __repr__, __setitem__, clear, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__
Inherited from type: fromkeys

12 Module matplotlib.cm

This module contains the instantiations of color mapping classes

12.1 Functions

get_cmap(*name*=None, *lut*=None)

Get a colormap instance, defaulting to rc values if name is None

12.2 Class ScalarMappable

Known Subclasses: AxesImage, ColorbarBase, ContourSet, FigureImage, LineCollection, PatchCollection

This is a mixin class to support scalar -> RGBA mapping. Handles normalization and colormapping

12.2.1 Methods

__init__(*self*, *norm*=None, *cmap*=None)

norm is a colors.Norm instance to map luminance to 0-1 cmap is a cm colormap instance

add_observer(*self*, *mappable*)

whenever the norm, clim or cmap is set, call the notify instance of the mappable observer with self. This is designed to allow one image to follow changes in the cmap of another image

autoscale(*self*)

Autoscale the scalar limits on the norm instance using the current array

autoscale_None(*self*)

Autoscale the scalar limits on the norm instance using the current array, changing only limits that are None

changed(*self*)

Call this whenever the mappable is changed so observers can update state

get_array(*self*)

Return the array

get_clim(*self*)

return the min, max of the color limits for image scaling

notify(*self*, *mappable*)

If this is called then we are pegged to another mappable. Update our cmap, norm, alpha from the other mappable.

set_array(*self*, *A*)

Set the image array from numeric/numarray A

set_clim(*self*, *vmin*=`None`, *vmax*=`None`)

set the norm limits for image scaling; if vmin is a length2 sequence, interpret it as (vmin, vmax) which is used to support setp
ACCEPTS: a length 2 sequence of floats

set_cmap(*self*, *cmap*)

set the colormap for luminance data
ACCEPTS: a colormap

set_colorbar(*self*, *im*, *ax*)

set the colorbar image and axes associated with mappable

set_norm(*self*, *norm*)

set the normalization instance

to_rgba(*self*, *x*, *alpha*=`1.0`)

Return a normalized rgba array corresponding to x. If x is already an rgb or rgba array, return it unchanged.

13 Module matplotlib.collections

Classes for the efficient drawing of large collections of objects that share most properties, eg a large number of line segments or polygons

The classes are not meant to be as flexible as their single element counterparts (eg you may not be able to select all line styles) but they are meant to be fast for common use cases (eg a bunch of solid line segemnts)

13.1 Class BrokenBarHCollection

matplotlib.artist.Artist

matplotlib.collections.Collection

matplotlib.cm.ScalarMappable

matplotlib.collections.PatchCollection

matplotlib.collections.PolyCollection

BrokenBarHCollection

A colleciton of horizontal bars spanning yrange with a sequence of xranges

13.1.1 Methods

__init__(*self*, *xranges*, *yrange*, ***kwargs*)

```
xranges : sequence of (xmin, xwidth)
yrange  : ymin, ywidth
    Valid PatchCollection kwargs are:

      edgecolors=None,
      facecolors=None,
      linewidths=None,
      antialiaseds = None,
      offsets = None,
      transOffset = identity_transform(),
      norm = None,  # optional for ScalarMappable
      cmap = None,  # ditto

    offsets and transOffset are used to translate the patch after
    rendering (default no offsets)

    If any of edgecolors, facecolors, linewidths, antialiaseds are
    None, they default to their patch.* rc params setting, in sequence
    form.
```

Overrides: matplotlib.collections.PolyCollection.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from PatchCollection: get_transformed_patches, get_transoffset, pick, set_alpha, set_color, set_edgecolor, set_facecolor, set_linewidth, update_scalarmappable
Inherited from PolyCollection: draw, get_verts, set_verts

13.1.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from PatchCollection: zorder *(p. 230)*	

13.2 Class Collection

matplotlib.artist.Artist
└ **Collection**

Known Subclasses: LineCollection, PatchCollection

```
All properties in a collection must be sequences.  The
property of the ith element of the collection is the

  prop[i % len(props)].

This implies that the properties cycle if the len of props is less
than the number of elements of the collection.  A length 1
property is shared by all the elements of the collection

All color args to a collection are sequences of rgba tuples
```

13.2.1 Methods

__init__(*self*)
Overrides: matplotlib.artist.Artist.__init__

get_verts(*self*)

return seq of (x,y) in collection

Inherited from Artist: add_callback, convert_xunits, convert_yunits, draw, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pick, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

13.2.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

13.3 Class LineCollection

matplotlib.artist.Artist

matplotlib.collections.Collection

matplotlib.cm.ScalarMappable

LineCollection

Known Subclasses: oLine3DCollection

All parameters must be sequences. The property of the ith line segment is the prop[i % len(props)], ie the properties cycle if the len of props is less than the number of sements

13.3.1 Methods

__init__(*self*, *segments*, *linewidths*=None, *colors*=None, *antialiaseds*=None, *linestyle*='solid', *offsets*=None, *transOffset*=None, *norm*=None, *cmap*=None, ***kwargs*)

```
segments is a sequence of ( line0, line1, line2), where
linen = (x0, y0), (x1, y1), ... (xm, ym), or the
equivalent numerix array with two columns.
Each line can be a different length.

colors must be a tuple of RGBA tuples (eg arbitrary color
strings, etc, not allowed).

antialiaseds must be a sequence of ones or zeros

linestyles is a string or dash tuple. Legal string values are
  solid|dashed|dashdot|dotted.  The dash tuple is (offset, onoffseq)
  where onoffseq is an even length tuple of on and off ink in points.

If linewidths, colors, or antialiaseds is None, they default to
their rc params setting, in sequence form.

If offsets and transOffset are not None, then
offsets are transformed by transOffset and applied after
the segments have been transformed to display coordinates.

If offsets is not None but transOffset is None, then the
offsets are added to the segments before any transformation.
In this case, a single offset can be specified as offsets=(xo,yo),
and this value will be
added cumulatively to each successive segment, so as
to produce a set of successively offset curves.

norm = None,  # optional for ScalarMappable
cmap = None,  # ditto

The use of ScalarMappable is optional.  If the ScalarMappable
matrix _A is not None (ie a call to set_array has been made), at
draw time a call to scalar mappable will be made to set the colors.
```

Overrides: matplotlib.collections.Collection.__init__

color(*self*, *c*)

Set the color(s) of the line collection. c can be a matplotlib color arg (all patches have same color), or a a sequence or rgba tuples; if it is a sequence the patches will cycle through the sequence
ACCEPTS: matplotlib color arg or sequence of rgba tuples

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_colors(*self*)

get_dashes(*self*)

get_linestyle(*self*)

get_linewidth(*self*)

get_transoffset(*self*)

get_verts(*self*, *dataTrans*=None)

Return vertices in data coordinates. The calculation is incomplete in general; it is based on the segments or the offsets, whichever is using dataTrans as its transformation, so it does not take into account the combined effect of segments and offsets.

Overrides: matplotlib.collections.Collection.get_verts

set_alpha(*self*, *alpha*)

Set the alpha tranpancies of the collection. Alpha can be a float, in which case it is applied to the entire collection, or a sequence of floats
ACCEPTS: float or sequence of floats

Overrides: matplotlib.artist.Artist.set_alpha

set_color(*self*, *c*)

Set the color(s) of the line collection. c can be a matplotlib color arg (all patches have same color), or a a sequence or rgba tuples; if it is a sequence the patches will cycle through the sequence
ACCEPTS: matplotlib color arg or sequence of rgba tuples

set_linestyle(*self*, *ls*)

Set the linestyles(s) for the collection. ACCEPTS: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq)]

set_linewidth(*self*, *lw*)

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence
ACCEPTS: float or sequence of floats

set_segments(*self*, *segments*)

set_verts(*self*, *segments*)

update_scalarmappable(*self*)
If the scalar mappable array is not none, update colors from scalar data

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pick, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba

13.3.2 Class Variables

Name	Description
zorder	**Value:** 2 *(type=`int`)*
Inherited from Artist: aname *(p. 92)*	

13.4 Class PatchCollection

matplotlib.artist.Artist

matplotlib.collections.Collection

matplotlib.cm.ScalarMappable

PatchCollection

Known Subclasses: PolyCollection, QuadMesh, RegularPolyCollection

```
Base class for filled regions such as PolyCollection etc.
It must be subclassed to be usable.

kwargs are:

      edgecolors=None,
      facecolors=None,
      linewidths=None,
      antialiaseds = None,
      offsets = None,
      transOffset = identity_transform(),
      norm = None,  # optional for ScalarMappable
      cmap = None,  # ditto

offsets and transOffset are used to translate the patch after
rendering (default no offsets)
```

```
If any of edgecolors, facecolors, linewidths, antialiaseds are
None, they default to their patch.* rc params setting, in sequence
form.

The use of ScalarMappable is optional.  If the ScalarMappable
matrix _A is not None (ie a call to set_array has been made), at
draw time a call to scalar mappable will be made to set the face
colors.
```

13.4.1 Methods

__init__(*self*, *edgecolors*=None, *facecolors*=None, *linewidths*=None, *antialiaseds*=None, *offsets*=None, *transOffset*=None, *norm*=None, *cmap*=None)

```
Create a PatchCollection
Valid PatchCollection kwargs are:

  edgecolors=None,
  facecolors=None,
  linewidths=None,
  antialiaseds = None,
  offsets = None,
  transOffset = identity_transform(),
  norm = None,  # optional for ScalarMappable
  cmap = None,  # ditto

offsets and transOffset are used to translate the patch after
rendering (default no offsets)

If any of edgecolors, facecolors, linewidths, antialiaseds are
None, they default to their patch.* rc params setting, in sequence
form.
```

Overrides: matplotlib.collections.Collection.__init__

get_transformed_patches(*self*)

get a sequence of the polygons in the collection in display (transformed) space
The ith element in the returned sequence is a list of x,y vertices defining the ith polygon

get_transoffset(*self*)

pick(*self*, *mouseevent*)

fire a pick event with the index into the data if the mouse click is within the patch

Overrides: matplotlib.artist.Artist.pick

set_alpha(*self*, *alpha*)

Set the alpha tranpancies of the collection. Alpha must be a float.
ACCEPTS: float

Overrides: matplotlib.artist.Artist.set_alpha

set_color(*self*, *c*)

Set both the edgecolor and the facecolor. See set_facecolor and set_edgecolor.
ACCEPTS: matplotlib color arg or sequence of rgba tuples

set_edgecolor(*self*, *c*)

Set the facecolor(s) of the collection. c can be a matplotlib color arg (all patches have same color), or a a sequence or rgba tuples; if it is a sequence the patches will cycle through the sequence
ACCEPTS: matplotlib color arg or sequence of rgba tuples

set_facecolor(*self*, *c*)

Set the facecolor(s) of the collection. c can be a matplotlib color arg (all patches have same color), or a a sequence or rgba tuples; if it is a sequence the patches will cycle through the sequence
ACCEPTS: matplotlib color arg or sequence of rgba tuples

set_linewidth(*self*, *lw*)

Set the linewidth(s) for the collection. lw can be a scalar or a sequence; if it is a sequence the patches will cycle through the sequence
ACCEPTS: float or sequence of floats

update_scalarmappable(*self*)

If the scalar mappable array is not none, update facecolors from scalar data

Inherited from Artist: add_callback, convert_xunits, convert_yunits, draw, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from Collection: get_verts

13.4.2 Class Variables

Name	Description
zorder	**Value:** `1` *(type=`int`)*
Inherited from Artist: aname *(p. 92)*	

13.5 Class PolyCollection

```
matplotlib.artist.Artist ─┐
matplotlib.collections.Collection ─┐
  matplotlib.cm.ScalarMappable ────┘
matplotlib.collections.PatchCollection ─┐
                                        PolyCollection
```

Known Subclasses: BrokenBarHCollection, Quiver

13.5.1 Methods

__init__(*self*, *verts*, ***kwargs*)

```
verts is a sequence of ( verts0, verts1, ...) where verts_i is
a sequence of xy tuples of vertices, or an equivalent
numerix array of shape (nv,2).
    Valid PatchCollection kwargs are:

      edgecolors=None,
      facecolors=None,
      linewidths=None,
      antialiaseds = None,
      offsets = None,
      transOffset = identity_transform(),
      norm = None,  # optional for ScalarMappable
      cmap = None,  # ditto

    offsets and transOffset are used to translate the patch after
    rendering (default no offsets)

    If any of edgecolors, facecolors, linewidths, antialiaseds are
    None, they default to their patch.* rc params setting, in sequence
    form.
```

Overrides: matplotlib.collections.PatchCollection.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_verts(*self*, *dataTrans*=`None`)

Return vertices in data coordinates. The calculation is incomplete in general; it is based on the vertices or the offsets, whichever is using dataTrans as its transformation, so it does not take into account the combined effect of segments and offsets.

Overrides: matplotlib.collections.Collection.get_verts

set_verts(*self*, *verts*)

This allows one to delay initialization of the vertices.

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from PatchCollection: get_transformed_patches, get_transoffset, pick, set_alpha, set_color, set_edgecolor, set_facecolor, set_linewidth, update_scalarmappable

13.5.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from PatchCollection: zorder *(p. 230)*	

13.6 Class QuadMesh

matplotlib.artist.Artist

matplotlib.collections.Collection

matplotlib.cm.ScalarMappable

matplotlib.collections.PatchCollection

QuadMesh

Class for the efficient drawing of a quadrilateral mesh. A quadrilateral mesh consists of a grid of vertices. The dimensions of this array are (meshWidth+1, meshHeight+1). Each vertex in the mesh has a different set of "mesh coordinates" representing its position in the topology of the mesh. For any values (m, n) such that 0 <= m <= meshWidth and 0 <= n <= meshHeight, the vertices at mesh coordinates (m, n), (m, n+1), (m+1, n+1), and (m+1, n) form one of the quadrilaterals in the mesh. There are thus (meshWidth * meshHeight) quadrilaterals in the mesh. The mesh need not be regular and the polygons need not be convex. A quadrilateral mesh is represented by a (2 x ((meshWidth + 1) * (meshHeight + 1))) Numeric array 'coordinates' where each row is the X and Y coordinates of one of the vertices. To define the function that maps from a data point to its corresponding color, use the set_cmap() function. Each of these arrays is indexed in row-major order by the mesh coordinates of the vertex (or the mesh coordinates of the lower left vertex, in the case of the colors). For example, the first entry in coordinates is the coordinates of the vertex at mesh coordinates (0, 0), then the one at (0, 1), then at (0, 2) .. (0, meshWidth), (1, 0), (1, 1), and so on.

13.6.1 Methods

__init__(*self*, *meshWidth*, *meshHeight*, *coordinates*, *showedges*)
Overrides: matplotlib.collections.PatchCollection.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_verts(*self*, *dataTrans*=`None`)

return seq of (x,y) in collection

Overrides: matplotlib.collections.Collection.get_verts extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from PatchCollection: get_transformed_patches, get_transoffset, pick, set_alpha, set_color, set_edgecolor, set_facecolor, set_linewidth, update_scalarmappable

13.6.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from PatchCollection: zorder *(p. 230)*	

13.7 Class RegularPolyCollection

matplotlib.artist.Artist

matplotlib.collections.Collection

matplotlib.cm.ScalarMappable

matplotlib.collections.PatchCollection

RegularPolyCollection

Known Subclasses: StarPolygonCollection

13.7.1 Methods

__init__(*self*, *dpi*, *numsides*, *rotation*=0, *sizes*=(1,), ***kwargs*)

```
Draw a regular polygon with numsides.
* dpi is the figure dpi instance, and is required to do the
  area scaling.
* numsides: the number of sides of the polygon
* sizes gives the area of the circle circumscribing the
  regular polygon in points^2
* rotation is the rotation of the polygon in radians
    Valid PatchCollection kwargs are:

      edgecolors=None,
      facecolors=None,
      linewidths=None,
      antialiaseds = None,
      offsets = None,
      transOffset = identity_transform(),
      norm = None,  # optional for ScalarMappable
      cmap = None,  # ditto

    offsets and transOffset are used to translate the patch after
    rendering (default no offsets)

    If any of edgecolors, facecolors, linewidths, antialiaseds are
    None, they default to their patch.* rc params setting, in sequence
    form.

Example: see examples/dynamic_collection.py for complete example
offsets = nx.mlab.rand(20,2)
facecolors = [cm.jet(x) for x in nx.mlab.rand(20)]
black = (0,0,0,1)
collection = RegularPolyCollection(
    fig.dpi,
    numsides=5, # a pentagon
    rotation=0,
    sizes=(50,),
    facecolors = facecolors,
    edgecolors = (black,),
    linewidths = (1,),
    offsets = offsets,
    transOffset = ax.transData,
    )
```

Overrides: matplotlib.collections.PatchCollection.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

---

get_transformed_patches(*self*)

get a sequence of the polygons in the collection in display (transformed) space
The ith element in the returned sequence is a list of x,y vertices defining the ith polygon

Overrides: matplotlib.collections.PatchCollection.get_transformed_patches extit(inherited documentation)

---

get_verts(*self*, *dataTrans*=`None`)

Return vertices in data coordinates. The calculation is incomplete; it uses only the offsets, and only if _transOffset is dataTrans.

Overrides: matplotlib.collections.Collection.get_verts

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from PatchCollection: get_transoffset, pick, set_alpha, set_color, set_edgecolor, set_facecolor, set_linewidth, update_scalarmappable

13.7.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from PatchCollection: zorder *(p. 230)*	

13.8 Class StarPolygonCollection

matplotlib.artist.Artist

matplotlib.collections.Collection

matplotlib.cm.ScalarMappable

matplotlib.collections.PatchCollection

matplotlib.collections.RegularPolyCollection

StarPolygonCollection

13.8.1 Methods

__init__(*self*, *dpi*, *numsides*, *rotation*=0, *sizes*=(1,), ***kwargs*)

```
Draw a regular star like Polygone with numsides.
* dpi is the figure dpi instance, and is required to do the
  area scaling.
* numsides: the number of sides of the polygon
* sizes gives the area of the circle circumscribing the
  regular polygon in points^2
* rotation is the rotation of the polygon in radians
    Valid PatchCollection kwargs are:

      edgecolors=None,
      facecolors=None,
      linewidths=None,
      antialiaseds = None,
      offsets = None,
      transOffset = identity_transform(),
      norm = None,  # optional for ScalarMappable
      cmap = None,  # ditto

    offsets and transOffset are used to translate the patch after
    rendering (default no offsets)

    If any of edgecolors, facecolors, linewidths, antialiaseds are
    None, they default to their patch.* rc params setting, in sequence
    form.
```

Overrides: matplotlib.collections.RegularPolyCollection.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from PatchCollection: get_transoffset, pick, set_alpha, set_color, set_edgecolor, set_facecolor, set_linewidth, update_scalarmappable
Inherited from RegularPolyCollection: draw, get_transformed_patches, get_verts

13.8.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from PatchCollection: zorder *(p. 230)*	

14 Module matplotlib.colorbar

```
Colorbar toolkit with two classes and a function:

    ColorbarBase is the base class with full colorbar drawing functionality.
        It can be used as-is to make a colorbar for a given colormap;
        a mappable object (e.g., image) is not needed.

    Colorbar is the derived class for use with images or contour plots.

    make_axes is a function for resizing an axes and adding a second axes
        suitable for a colorbar

The Figure.colorbar() method uses make_axes and Colorbar; the pylab.colorbar()
function is a thin wrapper over Figure.colorbar().
```

14.1 Functions

make_axes(*parent*, ***kw*)

```
Resize and reposition a parent axes, and return a child
axes suitable for a colorbar.

cax, kw = make_axes(parent, **kw)

Keyword arguments may include the following (with defaults):
    orientation = 'vertical'  or 'horizontal'

    fraction    = 0.15; fraction of original axes to use for colorbar
    pad         = 0.05 if vertical, 0.15 if horizontal; fraction
                          of original axes between colorbar and
                          new image axes
    shrink      = 1.0; fraction by which to shrink the colorbar
    aspect      = 20; ratio of long to short dimensions



All but the first of these are stripped from the input kw set.

Returns (cax, kw), the child axes and the reduced kw dictionary.
```

14.2 Class Colorbar

matplotlib.cm.ScalarMappable
  matplotlib.colorbar.ColorbarBase
    Colorbar

14.2.1 Methods

__init__(*self*, *ax*, *mappable*, ***kw*)
Overrides: matplotlib.colorbar.ColorbarBase.__init__

add_lines(*self*, *CS*)

Add the lines from a non-filled ContourSet to the colorbar.

Overrides: matplotlib.colorbar.ColorbarBase.add_lines

notify(*self*, *mappable*)

Manually change any contour line colors. This is called when the image or contour plot to which this colorbar belongs is changed.

Overrides: matplotlib.cm.ScalarMappable.notify

Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from ColorbarBase: draw_all, set_alpha, set_label

14.3 Class ColorbarBase

matplotlib.cm.ScalarMappable
  ColorbarBase

Known Subclasses: Colorbar

14.3.1 Methods

__init__(*self*, *ax*, *cmap*=`None`, *norm*=`None`, *alpha*=`1.0`, *values*=`None`, *boundaries*=`None`, *orientation*=`'vertical'`, *extend*=`'neither'`, *spacing*=`'uniform'`, *ticks*=`None`, *format*=`None`, *drawedges*=`False`, *filled*=`True`)
Overrides: matplotlib.cm.ScalarMappable.__init__

add_lines(*self*, *levels*, *colors*, *linewidths*)

Draw lines on the colorbar.

draw_all(*self*)

Calculate any free parameters based on the current cmap and norm, and do all the drawing.

set_alpha(*self*, *alpha*)

set_label(*self*, *label*, ***kw*)

Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba

15 Module matplotlib.colors

```
A class for converting color arguments to RGB or RGBA

This class instantiates a single instance colorConverter that is used
to convert matlab color strings to RGB.  RGB is a tuple of float RGB
values in the range 0-1.

Commands which take color arguments can use several formats to specify
the colors.  For the basic builtin colors, you can use a single letter

      b  : blue
      g  : green
      r  : red
      c  : cyan
      m  : magenta
      y  : yellow
      k  : black
      w  : white

Gray shades can be given as a string encoding a float in the 0-1
range, e.g.,

    color = '0.75'

For a greater range of colors, you have two options.  You can specify
the color using an html hex string, as in

      color = '#eeefff'

or you can pass an R,G,B tuple, where each of R,G,B are in the range
[0,1].

Finally, legal html names for colors, like 'red', 'burlywood' and
'chartreuse' are supported.
```

15.1 Functions

hex2color(*s*)

Take a hex string 's' and return the corresponding rgb 3-tuple Example: #efefef -> (0.93725, 0.93725, 0.93725)

is_color_like(*c*)

makeMappingArray(*N*, *data*)

Create an N-element 1-d lookup table
data represented by a list of x,y0,y1 mapping correspondences. Each element in this list represents how a value between 0 and 1 (inclusive) represented by x is mapped to a corresponding value between 0 and 1 (inclusive). The two values of y are to allow for discontinuous mapping functions (say as might be found in a sawtooth) where y0 represents the value of y for values of x <= to that given, and y1 is the value to be used for x > than that given). The list must start with x=0, end with x=1, and all values of x must be in increasing order. Values between the given mapping points are determined by simple linear interpolation.
The function returns an array "result" where result[x*(N-1)] gives the closest value for values of x between 0 and 1.

rgb2hex(*rgb*)

Given a len 3 rgb tuple of 0-1 floats, return the hex string

15.2 Class ColorConverter

15.2.1 Methods

to_rgb(*self*, *arg*)

```
Returns an RGB tuple of three floats from 0-1.

arg can be an RGB or RGBA sequence or a string in any of several forms:
    1) a letter from the set 'rgbcmykw'
    2) a hex color string, like '#00FFFF'
    3) a standard name, like 'aqua'
    4) a float, like '0.4', indicating gray on a 0-1 scale

if arg is RGBA, the A will simply be discarded.
```

to_rgba(*self*, *arg*, *alpha*=`None`)

Returns an RGBA tuple of four floats from 0-1.
For acceptable values of arg, see to_rgb. If arg is an RGBA sequence and alpha is not None, alpha will replace the original A.

to_rgba_list(*self*, *c*, *alpha*=`None`)

Returns a list of rgba tuples.
Accepts a single mpl color spec or a sequence of specs. If the sequence is a list, the list items are changed in place.

15.2.2 Class Variables

Name	Description
cache	**Value:** {} *(type=`dict`)*
colors	**Value:** `{'c': (0.0, 0.75, 0.75), 'b': (0.0, 0.0, 1.0), 'g': (0.0, 0.5, 0.0), 'k': (0....` *(type=`dict`)*

15.3 Class Colormap

Known Subclasses: LinearSegmentedColormap

```
Base class for all scalar to rgb mappings

Important methods:
    set_bad()
    set_under()
    set_over()
```

15.3.1 Methods

__init__(*self*, *name*, *N*=`256`)

Public class attributes: self.N: number of rgb quantization levels self.name: name of colormap

__call__(*self*, *X*, *alpha*=`1.0`)

X is either a scalar or an array (of any dimension). If scalar, a tuple of rgba values is returned, otherwise an array with the new shape = oldshape+(4,). If the X-values are integers, then they are used as indices into the array. If they are floating point, then they must be in the interval (0.0, 1.0). Alpha must be a scalar.

is_gray(*self*)

set_bad(*self*, *color*=`'k'`, *alpha*=`1.0`)

Set color to be used for masked values.

set_over(*self*, *color*=`'k'`, *alpha*=`1.0`)

Set color to be used for high out-of-range values. Requires norm.clip = False

set_under(*self*, *color*=`'k'`, *alpha*=`1.0`)

Set color to be used for low out-of-range values. Requires norm.clip = False

15.4 Class LinearSegmentedColormap

matplotlib.colors.Colormap
LinearSegmentedColormap

Known Subclasses: ListedColormap

Colormap objects based on lookup tables using linear segments.

The lookup transfer function is a simple linear function between defined intensities. There is no limit to the number of segments that may be defined. Though as the segment intervals start containing fewer and fewer array locations, there will be inevitable quantization errors

15.4.1 Methods

__init__(*self*, *name*, *segmentdata*, *N*=`256`)

Create color map from linear mapping segments
segmentdata argument is a dictionary with a red, green and blue entries. Each entry should be a list of x, y0, y1 tuples. See makeMappingArray for details

Overrides: matplotlib.colors.Colormap.__init__

Inherited from Colormap: __call__, is_gray, set_bad, set_over, set_under

15.5 Class ListedColormap

matplotlib.colors.Colormap
matplotlib.colors.LinearSegmentedColormap
ListedColormap

Colormap object generated from a list of colors.

Color boundaries are evenly spaced. This is intended for simulating indexed color selection, but may be useful for generating special colormaps also.

15.5.1 Methods

__init__(*self*, *colors*, *name*=`'from_list'`, *N*=`None`)
Overrides: matplotlib.colors.LinearSegmentedColormap.__init__

Inherited from Colormap: __call__, is_gray, set_bad, set_over, set_under

15.6 Class LogNorm

matplotlib.colors.Normalize ─┐
LogNorm

Normalize a given value to the 0-1 range on a log scale

15.6.1 Methods

__call__(*self*, *value*, *clip*=`None`)
Overrides: matplotlib.colors.Normalize.__call__

inverse(*self*, *value*)
Overrides: matplotlib.colors.Normalize.inverse

Inherited from Normalize: __init__, autoscale, autoscale_None, scaled

15.7 Class NoNorm

matplotlib.colors.Normalize ─┐
NoNorm

Dummy replacement for Normalize, for the case where we want to use indices directly in a ScalarMappable.

15.7.1 Methods

__call__(*self*, *value*, *clip*=`None`)
Overrides: matplotlib.colors.Normalize.__call__

inverse(*self*, *value*)
Overrides: matplotlib.colors.Normalize.inverse

Inherited from Normalize: __init__, autoscale, autoscale_None, scaled

15.8 Class NoNorm

matplotlib.colors.Normalize ─┐
NoNorm

Dummy replacement for Normalize, for the case where we want to use indices directly in a ScalarMappable.

15.8.1 Methods

__call__(*self*, *value*, *clip*=`None`)
Overrides: matplotlib.colors.Normalize.__call__

inverse(*self*, *value*)
Overrides: matplotlib.colors.Normalize.inverse

Inherited from Normalize: __init__, autoscale, autoscale_None, scaled

15.9 Class Normalize

Known Subclasses: LogNorm, NoNorm

Normalize a given value to the 0-1 range

15.9.1 Methods

__init__(*self*, *vmin*=`None`, *vmax*=`None`, *clip*=`True`)

If vmin or vmax is not given, they are taken from the input's minimum and maximum value respectively. If clip is True and the given value falls outside the range, the returned value will be 0 or 1, whichever is closer. Returns 0 if vmin==vmax. Works with scalars or arrays, including masked arrays. If clip is True, masked values are set to 1; otherwise they remain masked.

__call__(*self*, *value*, *clip*=`None`)

autoscale(*self*, *A*)

Set vmin, vmax to min, max of A.

autoscale_None(*self*, *A*)

autoscale only None-valued vmin or vmax

inverse(*self*, *value*)

scaled(*self*)

return true if vmin and vmax set

15.10 Class Normalize

Known Subclasses: LogNorm, NoNorm

Normalize a given value to the 0-1 range

15.10.1 Methods

__init__(*self*, *vmin*=`None`, *vmax*=`None`, *clip*=`True`)

If vmin or vmax is not given, they are taken from the input's minimum and maximum value respectively. If clip is True and the given value falls outside the range, the returned value will be 0 or 1, whichever is closer. Returns 0 if vmin==vmax. Works with scalars or arrays, including masked arrays. If clip is True, masked values are set to 1; otherwise they remain masked.

__call__(*self*, *value*, *clip*=`None`)

autoscale(*self*, *A*)

Set vmin, vmax to min, max of A.

autoscale_None(*self*, *A*)

autoscale only None-valued vmin or vmax

inverse(*self*, *value*)

scaled(*self*)

return true if vmin and vmax set

16 Module matplotlib.contour

These are classes to support contour plotting and labelling for the axes class

16.1 Class ContourLabeler

Known Subclasses: ContourSet

Mixin to provide labelling capability to ContourSet

16.1.1 Methods

break_linecontour(*self*, *linecontour*, *rot*, *labelwidth*, *ind*)

break a contour in two contours at the location of the label

clabel(*self*, **args*, ***kwargs*)

```
clabel(CS, **kwargs) - add labels to line contours in CS,
       where CS is a ContourSet object returned by contour.

clabel(CS, V, **kwargs) - only label contours listed in V

keyword arguments:

* fontsize = None: as described in http://matplotlib.sf.net/fonts.html

* colors = None:

   - a tuple of matplotlib color args (string, float, rgb, etc),
     different labels will be plotted in different colors in the order
     specified

   - one string color, e.g. colors = 'r' or colors = 'red', all labels
     will be plotted in this color

   - if colors == None, the color of each label matches the color
     of the corresponding contour

* inline = True: controls whether the underlying contour is removed
             (inline = True) or not (False)

* fmt = '%1.3f': a format string for the label
```

get_label_coords(*self*, *distances*, *XX*, *YY*, *ysize*, *lw*)

labels are ploted at a location with the smallest dispersion of the contour from a straight line unless there's another label nearby, in which case the second best place on the contour is picked up if there's no good place a label isplotted at the beginning of the contour

get_label_width(*self*, *lev*, *fmt*, *fsize*)

get the width of the label in points

get_text(*self*, *lev*, *fmt*)

get the text of the label

labels(*self*, *inline*)

locate_label(*self*, *linecontour*, *labelwidth*)

find a good place to plot a label (relatively flat part of the contour) and the angle of rotation for the text object

print_label(*self*, *linecontour*, *labelwidth*)

if contours are too short, don't plot a label

set_label_props(*self*, *label*, *text*, *color*)

set the label properties - color, fontsize, text

too_close(*self*, *x*, *y*, *lw*)

if there's a label already nearby, find a better place

16.2 Class ContourSet

matplotlib.contour.ContourLabeler
matplotlib.cm.ScalarMappable
ContourSet

```
Create and store a set of contour lines or filled regions.

User-callable method: clabel

Useful attributes:
    ax - the axes object in which the contours are drawn
    collections - a silent_list of LineCollections or PolyCollections
```

```
levels - contour levels
layers - same as levels for line contours; half-way between
         levels for filled contours.  See _process_colors method.
```

16.2.1 Methods

__init__(*self*, *ax*, **args*, ***kwargs*)

Draw contour lines or filled regions, depending on whether keyword arg 'filled' is False (default) or True. The first argument of the initializer must be an axes object. The remaining arguments and keyword arguments are described in ContourSet.contour_doc.

Overrides: matplotlib.cm.ScalarMappable.__init__

changed(*self*)

Call this whenever the mappable is changed so observers can update state

Overrides: matplotlib.cm.ScalarMappable.changed extit(inherited documentation)

get_alpha(*self*)

For compatibility with artists, return self.alpha

set_alpha(*self*, *alpha*)

For compatibility with artists, set self.alpha

Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from ContourLabeler: break_linecontour, clabel, get_label_coords, get_label_width, get_text, labels, locate_label, print_label, set_label_props, too_close

16.2.2 Class Variables

Name	Description
contour_doc	**Value:** `"\n        contour and contourf draw contour lines and filled contours,\n    ...` *(type=`str`)*

17 Module matplotlib.dates

```
Matplotlib provides sophisticated date plotting capabilites, standing
on the shoulders of python datetime, the add-on modules pytz and
dateutils.  datetime objects are converted to floating point numbers
which represent the number of days since 0001-01-01 UTC.  The helper
functions date2num, num2date and drange are used to facilitate easy
conversion to and from datetime and numeric ranges.

A wide range of specific and general purpose date tick locators and
formatters are provided in this module.  See matplotlib.tickers for
general information on tick locators and formatters.  These are
described below.

All the matplotlib date converters, tickers and formatters are
timezone aware, and the default timezone is given by the timezone
parameter in your matplotlibrc file.  If you leave out a tz timezone
instance, the default from your rc file will be assumed.  If you want
to use a custom time zone, pass a matplotlib.pytz.timezone instance
with the tz keyword argument to num2date, plot_date, and any custom
date tickers or locators you create.  See http://pytz.sourceforge.net
for information on pytz and timezone handling.

dateutils https://moin.conectiva.com.br/DateUtil the code to handle
date ticking, making it easy to place ticks on any kinds of dates -
see examples below.

Date tickers -

  Most of the date tickers can locate single or multiple values.  Eg

    # tick on mondays every week
    loc = WeekdayLocator(byweekday=MO, tz=tz)

    # tick on mondays and saturdays
    loc = WeekdayLocator(byweekday=(MO, SA))

  In addition, most of the constructors take an interval argument.

    # tick on mondays every second week
    loc = WeekdayLocator(byweekday=MO, interval=2)

  The rrule locator allows completely general date ticking

    # tick every 5th easter
    rule = rrulewrapper(YEARLY, byeaster=1, interval=5)
    loc = RRuleLocator(rule)
```

```
  Here are all the date tickers

    * MinuteLocator  - locate minutes

    * HourLocator    - locate hours

    * DayLocator     - locate specifed days of the month

    * WeekdayLocator - Locate days of the week, eg MO, TU

    * MonthLocator   - locate months, eg 7 for july

    * YearLocator    - locate years that are multiples of base

    * RRuleLocator - locate using a matplotlib.dates.rrulewrapper.
        The rrulewrapper is a simple wrapper around a dateutils.rrule
        https://moin.conectiva.com.br/DateUtil which allow almost
        arbitrary date tick specifications.  See
        examples/date_demo_rrule.py


Date formatters

  DateFormatter - use strftime format strings

  DateIndexFormatter - date plots with implicit x indexing.
```

17.1 Functions

date2num(*d*)

d is either a datetime instance or a sequence of datetimes
return value is a floating point number (or sequence of floats) which gives number of days (fraction part represents hours, minutes, seconds) since 0001-01-01 00:00:00 UTC

drange(*dstart*, *dend*, *delta*)

Return a date range as float gregorian ordinals. dstart and dend are datetime instances. delta is a datetime.timedelta instance

epoch2num(*e*)

convert an epoch or sequence of epochs to the new date format, days since 0001

hours(*h*)

return hours as days

minutes(*m*)

return minutes as days

mx2num(*mxdates*)

Convert mx datetime instance (or sequence of mx instances) to the new date format,

num2date(*x*, *tz*=`None`)

x is a float value which gives number of days (fraction part represents hours, minutes, seconds) since 0001-01-01 00:00:00 UTC
Return value is a datetime instance in timezone tz (default to rcparams TZ value)
if x is a sequence, a sequence of datetimes will be returned

num2epoch(*d*)

convert days since 0001 to epoch. d can be a number or sequence

seconds(*s*)

return seconds as days

weeks(*w*)

return weeks as days

17.2 Class DateFormatter

matplotlib.ticker.TickHelper
  matplotlib.ticker.Formatter
    DateFormatter

Tick location is seconds since the epoch. Use a strftime format string

python only supports datetime strftime formatting for years greater than 1900. Thanks to Andrew Dalke, Dalke Scientific Software who contributed the strftime code below to include dates earlier than this year

17.2.1 Methods

__init__(*self*, *fmt*, *tz*=`None`)

fmt is an strftime format string; tz is the tzinfo instance

__call__(*self*, *x*, *pos*=0)

Return the format for tick val x at position pos; pos=None indicated unspecified

Overrides: matplotlib.ticker.Formatter.__call__ extit(inherited documentation)

set_tzinfo(*self*, *tz*)

strftime(*self*, *dt*, *fmt*)

Inherited from Formatter: format_data, format_data_short, get_offset, set_locs
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.2.2 Class Variables

Name	Description
illegal_s	**Value:** `<_sre.SRE_Pattern object at 0x85fc458>` *(type=`SRE_Pattern`)*
Inherited from Formatter: locs *(p. 525)* **Inherited from TickHelper:** dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.3 Class DateLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
DateLocator

Known Subclasses: RRuleLocator, YearLocator, AutoDateLocator

17.3.1 Methods

__init__(*self*, *tz*=None)

tz is the tzinfo instance

datalim_to_dt(*self*)

nonsingular(*self*, *vmin*, *vmax*)

set_tzinfo(*self*, *tz*)

viewlim_to_dt(*self*)

Inherited from Locator: __call__, autoscale, pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.3.2 Class Variables

Name	Description
hms0d	**Value:** {'byminute': 0, 'byhour': 0, 'bysecond': 0} *(type=dict)*
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.4 Class DayLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
matplotlib.dates.DateLocator
matplotlib.dates.RRuleLocator
DayLocator

Make ticks on occurances of each day of the month, eg 1, 15, 30

17.4.1 Methods

__init__(*self*, *bymonthday*=None, *interval*=1, *tz*=None)

mark every day in bymonthday; bymonthday can be an int or sequence
Default is to tick every day of the month - bymonthday=range(1,32)

Overrides: matplotlib.dates.RRuleLocator.__init__

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from RRuleLocator: __call__, autoscale
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.4.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.5 Class HourLocator

matplotlib.ticker.TickHelper
  matplotlib.ticker.Locator
    matplotlib.dates.DateLocator
      matplotlib.dates.RRuleLocator
        HourLocator

Make ticks on occurances of each hour

17.5.1 Methods

__init__(*self*, *byhour*=`None`, *interval*=`1`, *tz*=`None`)

mark every hour in byhour; byhour can be an int or sequence. Default is to tick every hour - byhour=range(24)
interval is the interval between each iteration. Eg, if interval=2, mark every second occurance

Overrides: matplotlib.dates.RRuleLocator.__init__

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from RRuleLocator: __call__, autoscale
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.5.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.6 Class IndexDateFormatter

matplotlib.ticker.TickHelper
  matplotlib.ticker.Formatter
    IndexDateFormatter

Use with IndexLocator to cycle format strings by index.

17.6.1 Methods

__init__(*self*, *t*, *fmt*, *tz*=None)
t is a sequence of dates floating point days). fmt is a strftime format string

__call__(*self*, *x*, *pos*=0)
Return the label for time x at position pos Overrides: matplotlib.ticker.Formatter.__call__

Inherited from Formatter: format_data, format_data_short, get_offset, set_locs
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.6.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.7 Class MinuteLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
matplotlib.dates.DateLocator
matplotlib.dates.RRuleLocator
MinuteLocator

Make ticks on occurances of each minute

17.7.1 Methods

__init__(*self*, *byminute*=None, *interval*=1, *tz*=None)
mark every minute in byminute; byminute can be an int or sequence. default is to tick every minute - byminute=range(60) interval is the interval between each iteration. Eg, if interval=2, mark every second occurance Overrides: matplotlib.dates.RRuleLocator.__init__

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from RRuleLocator: __call__, autoscale
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.7.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.8 Class MonthLocator

matplotlib.ticker.TickHelper
  matplotlib.ticker.Locator
    matplotlib.dates.DateLocator
      matplotlib.dates.RRuleLocator
        MonthLocator

Make ticks on occurances of each month month, eg 1, 3, 12

17.8.1 Methods

__init__(*self*, *bymonth*=`None`, *bymonthday*=`1`, *interval*=`1`, *tz*=`None`)

mark every month in bymonth; bymonth can be an int or sequence. default is range(1,13), ie every month interval is the interval between each iteration. Eg, if interval=2, mark every second occurance

Overrides: matplotlib.dates.RRuleLocator.__init__

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from RRuleLocator: __call__, autoscale
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.8.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.9 Class RRuleLocator

matplotlib.ticker.TickHelper

matplotlib.ticker.Locator

matplotlib.dates.DateLocator

RRuleLocator

Known Subclasses: DayLocator, HourLocator, MinuteLocator, MonthLocator, SecondLocator, Weekday-Locator

17.9.1 Methods

__init__(*self*, *o*, *tz*=`None`)
Overrides: matplotlib.dates.DateLocator.__init__

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__ extit(inherited documentation)

autoscale(*self*)

Set the view limits to include the data range

Overrides: matplotlib.ticker.Locator.autoscale

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.9.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.10 Class SecondLocator

matplotlib.ticker.TickHelper
  matplotlib.ticker.Locator
    matplotlib.dates.DateLocator
      matplotlib.dates.RRuleLocator
        SecondLocator

Make ticks on occurances of each second

17.10.1 Methods

__init__(*self*, *bysecond*=`None`, *interval*=`1`, *tz*=`None`)

mark every second in bysecond; bysecond can be an int or sequence. Default is to tick every second bysecond = range(60)
interval is the interval between each iteration. Eg, if interval=2, mark every second occurance

Overrides: matplotlib.dates.RRuleLocator.__init__

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from RRuleLocator: __call__, autoscale
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.10.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.11 Class WeekdayLocator

matplotlib.ticker.TickHelper
  matplotlib.ticker.Locator
    matplotlib.dates.DateLocator
      matplotlib.dates.RRuleLocator
        WeekdayLocator

Make ticks on occurances of each weekday

17.11.1 Methods

__init__(*self*, *byweekday*=`1`, *interval*=`1`, *tz*=`None`)
mark every weekday in byweekday; byweekday can be a number or sequence elements of byweekday must be one of MO, TU, WE, TH, FR, SA, SU, the constants from dateutils.rrule interval specifies the number of weeks to skip. Ie interval=2 plots every second week
Overrides: matplotlib.dates.RRuleLocator.__init__

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from RRuleLocator: __call__, autoscale
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.11.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

17.12 Class YearLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
matplotlib.dates.DateLocator
YearLocator

Make ticks on a given day of each year that is a multiple of base.

Examples: # Tick every year on Jan 1st locator = YearLocator()

Tick every 5 years on July 4th locator = YearLocator(5, month=7, day=4)

17.12.1 Methods

__init__(*self*, *base*=`1`, *month*=`1`, *day*=`1`, *tz*=`None`)
mark years that are multiple of base on a given month and day (default jan 1)
Overrides: matplotlib.dates.DateLocator.__init__

__call__(*self*)
Return the locations of the ticks
Overrides: matplotlib.ticker.Locator.__call__ extit(inherited documentation)

autoscale(*self*)

Set the view limits to include the data range

Overrides: matplotlib.ticker.Locator.autoscale

Inherited from DateLocator: datalim_to_dt, nonsingular, set_tzinfo, viewlim_to_dt
Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

17.12.2 Class Variables

Name	Description
Inherited from DateLocator: hms0d *(p. 255)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

18 Module matplotlib.dviread

```
An experimental module for reading single-page dvi files output by
TeX. Several limitations make this not (currently) useful as a
general-purpose dvi preprocessor. The idea is that the file has a
single page with only a single formula or other piece of text.

Interface:

   dvi = Dvi(filename)
   dvi.read()
   text, boxes = dvi.output(72)
   for x,y,font,glyph in text:
       fontname, pointsize = dvi.fontinfo(font)
       ...
   for x,y,height,width in boxes:
       ...
```

18.1 Class Dvi

__builtin__.object
└─ **Dvi**

18.1.1 Methods

__init__(*self*, *filename*)
Overrides: __builtin__.object.__init__

arg(*self*, *nbytes*, *signed*=`False`)

bop(*self*, *c0*, *c1*, *c2*, *c3*, *c4*, *c5*, *c6*, *c7*, *c8*, *c9*, *p*)

dispatch(*self*, *byte*)

down(*self*, *a*)

down_y(*self*, *new_y*)

down_z(*self*, *new_z*)

eop(*self*)

fnt_def(*self*, *k*, *c*, *s*, *d*, *a*, *l*, *n*)

fnt_num(*self*, *k*)

fontinfo(*self*, *f*)

Name and size in (Adobe) points.

nop(*self*)

output(*self*, *dpi*)

Return lists of text and box objects transformed into a standard Cartesian coordinate system at the given dpi value. The coordinates are floating point numbers, but otherwise precision is not lost and coordinate values are not clipped to integers.

pop(*self*)

post(*self*)

post_post(*self*)

pre(*self*, *i*, *num*, *den*, *mag*, *comment*)

push(*self*)

put_char(*self*, *char*)

put_rule(*self*, *a*, *b*)

read(*self*, *debug*=`False`)

right(*self*, *b*)

right_w(*self*, *new_w*)

right_x(*self*, *new_x*)

set_char(*self*, *char*)

set_rule(*self*, *a*, *b*)

xxx(*self*, *special*)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

18.2 Class Tfm

```
__builtin__.object ─┐
                    Tfm
```

18.2.1 Methods

__init__(*self*, *filename*)
Overrides: __builtin__.object.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

19 Module matplotlib.figure

Figure class – add docstring here!

19.1 Functions

figaspect(*arg*)

```
Create a figure with specified aspect ratio.  If arg is a number,
use that aspect ratio.  If arg is an array, figaspect will
determine the width and height for a figure that would fit array
preserving aspect ratio.  The figure width, height in inches are
returned.  Be sure to create an axes with equal with and height,
eg

Example usage:

  # make a figure twice as tall as it is wide
  w, h = figaspect(2.)
  fig = Figure(figsize=(w,h))
  ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
  ax.imshow(A, **kwargs)


  # make a figure with the proper aspect for an array
  A = rand(5,3)
  w, h = figaspect(A)
  fig = Figure(figsize=(w,h))
  ax = fig.add_axes([0.1, 0.1, 0.8, 0.8])
  ax.imshow(A, **kwargs)

Thanks to Fernando Perez for this function
```

19.2 Class Figure

matplotlib.artist.Artist
└─ **Figure**

19.2.1 Methods

__init__(*self*, *figsize*=None, *dpi*=None, *facecolor*=None, *edgecolor*=None, *linewidth*=1.0, *frameon*=True, *subplotpars*=None)

figsize is a w,h tuple in inches dpi is dots per inch subplotpars is a SubplotParams instance, defaults to rc

Overrides: matplotlib.artist.Artist.__init__

add_axes(*self*, **args*, ***kwargs*)

```
Add an a axes with axes rect [left, bottom, width, height] where all
quantities are in fractions of figure width and height.  kwargs are
legal Axes kwargs plus "polar" which sets whether to create a polar axes
    rect = l,b,w,h
    add_axes(rect)
    add_axes(rect, frameon=False, axisbg='g')
    add_axes(rect, polar=True)
    add_axes(ax)   # add an Axes instance
If the figure already has an axes with key *args, *kwargs then it will
simply make that axes current and return it.  If you do not want this
behavior, eg you want to force the creation of a new axes, you must
use a unique set of args and kwargs.  The artist "label" attribute has
been exposed for this purpose.  Eg, if you want two axes that are
otherwise identical to be added to the figure, make sure you give them
unique labels:
    add_axes(rect, label='axes1')
    add_axes(rect, label='axes2')
The Axes instance will be returned
The following kwargs are supported:
        adjustable: ['box' | 'datalim']
        alpha: float
        anchor: ['C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W']
        animated: [True | False]
        aspect: ['auto' | 'equal' | aspect_ratio]
        autoscale_on: True|False
        axes: an axes instance
        axis_bgcolor: any matplotlib color - see help(colors)
        axis_off: void
        axis_on: void
        axisbelow: True|False
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cursor_props: a (float, color) tuple
        figure: a Figure instance
        frame_on: True|False
        label: any string
        lod: [True | False]
        navigate: True|False
        navigate_mode: unknown
        picker: [None|float|boolean|callable]
        position: len(4) sequence of floats
        title: str
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xlabel: str
        xlim: len(2) sequence of floats
        xscale: ['log' | 'linear' ]
        xticklabels: sequence of strings
        xticks: sequence of floats
        ylabel: str
        ylim: len(2) sequence of floats
        yscale: ['log' | 'linear']
        yticklabels: sequence of strings
        yticks: sequence of floats
        zorder: any number
```

add_axobserver(*self*, *func*)

whenever the axes state change, func(self) will be called

add_subplot(*self*, **args*, ***kwargs*)

```
Add a subplot.  Examples
    add_subplot(111)
    add_subplot(212, axisbg='r')  # add subplot with red background
    add_subplot(111, polar=True)  # add a polar subplot
    add_subplot(sub)              # add Subplot instance sub
kwargs are legal Axes kwargs plus"polar" which sets whether to create a
polar axes.  The Axes instance will be returned.
If the figure already has a subplot with key *args, *kwargs then it will
simply make that subplot current and return it
The following kwargs are supported:
        adjustable: ['box' | 'datalim']
        alpha: float
        anchor: ['C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W']
        animated: [True | False]
        aspect: ['auto' | 'equal' | aspect_ratio]
        autoscale_on: True|False
        axes: an axes instance
        axis_bgcolor: any matplotlib color - see help(colors)
        axis_off: void
        axis_on: void
        axisbelow: True|False
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cursor_props: a (float, color) tuple
        figure: a Figure instance
        frame_on: True|False
        label: any string
        lod: [True | False]
        navigate: True|False
        navigate_mode: unknown
        picker: [None|float|boolean|callable]
        position: len(4) sequence of floats
        title: str
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xlabel: str
        xlim: len(2) sequence of floats
        xscale: ['log' | 'linear' ]
        xticklabels: sequence of strings
        xticks: sequence of floats
        ylabel: str
        ylim: len(2) sequence of floats
        yscale: ['log' | 'linear']
        yticklabels: sequence of strings
        yticks: sequence of floats
        zorder: any number
```

autofmt_xdate(*self*, *bottom*=`0.20000000000000001`, *rotation*=`30`, *ha*=`'right'`)

A common use case is a number of subplots with shared xaxes where the x-axis is date data. The ticklabels are often long,and it helps to rotate them on the bottom subplot and turn them off on other subplots. This function will raise a RuntimeError if any of the Axes are not Subplots.
bottom : the bottom of the subplots for subplots_adjust rotation: the rotation of the xtick labels ha : the horizontal alignment of the xticklabels

clear(*self*)

Clear the figure

clf(*self*)

Clear the figure

colorbar(*self*, *mappable*, *cax*=None, ***kw*)

```
Create a colorbar for a ScalarMappable instance.

        Documentation for the pylab thin wrapper:
Add a colorbar to a plot.

Function signatures:

    colorbar(**kwargs)

    colorbar(mappable, **kwargs)

    colorbar(mappable, cax, **kwargs)

The optional arguments mappable and cax may be included in the kwargs;
they are image, ContourSet, etc. to which the colorbar applies, and
the axes object in which the colorbar will be drawn.  Defaults are
the current image and a new axes object created next to that image
after resizing the image.

kwargs are in two groups:
    axes properties:

        fraction    = 0.15; fraction of original axes to use for colorbar
        pad         = 0.05 if vertical, 0.15 if horizontal; fraction
                              of original axes between colorbar and
                              new image axes
        shrink      = 1.0; fraction by which to shrink the colorbar
        aspect      = 20; ratio of long to short dimensions



    colorbar properties:

        extend='neither', 'both', 'min', 'max'
                If not 'neither', make pointed end(s) for out-of-range
                values.  These are set for a given colormap using the
                colormap set_under and set_over methods.
        spacing='uniform', 'proportional'
                Uniform spacing gives each discrete color the same space;
                proportional makes the space proportional to the data interval.
        ticks=None, list of ticks, Locator object
                If None, ticks are determined automatically from the input.
        format=None, format string, Formatter object
                If none, the ScalarFormatter is used.
                If a format string is given, e.g. '%.3f', that is used.
                An alternative Formatter object may be given instead.
        drawedges=False, True
                If true, draw lines at color boundaries.

        The following will probably be useful only in the context of
        indexed colors (that is, when the mappable has norm=NoNorm()),
        or other unusual circumstances.

        boundaries=None or a sequence
        values=None or a sequence which must be of length 1 less than the
                sequence of boundaries.
                For each region delimited by adjacent entries in
```

delaxes(*self*, *a*)

remove a from the figure and update the current axes

draw(*self*, *renderer*)

Render the figure using Renderer instance renderer

Overrides: matplotlib.artist.Artist.draw

draw_artist(*self*, *a*)

draw artist only – this is available only after the figure is drawn

figimage(*self*, *X*, *xo*=0, *yo*=0, *alpha*=1.0, *norm*=None, *cmap*=None, *vmin*=None, *vmax*=None, *origin*=None)

```
FIGIMAGE(X) # add non-resampled array to figure

FIGIMAGE(X, xo, yo) # with pixel offsets

FIGIMAGE(X, **kwargs) # control interpolation ,scaling, etc

Add a nonresampled figure to the figure from array X.  xo and yo are
offsets in pixels

X must be a float array

    If X is MxN, assume luminance (grayscale)
    If X is MxNx3, assume RGB
    If X is MxNx4, assume RGBA

The following kwargs are allowed:

  * cmap is a cm colormap instance, eg cm.jet.  If None, default to
    the rc image.cmap valuex

  * norm is a matplotlib.colors.Normalize instance; default is
    normalization().  This scales luminance -> 0-1

  * vmin and vmax are used to scale a luminance image to 0-1.  If
    either is None, the min and max of the luminance values will be
    used.  Note if you pass a norm instance, the settings for vmin and
    vmax will be ignored.

  * alpha = 1.0 : the alpha blending value

  * origin is either 'upper' or 'lower', which indicates where the [0,0]
    index of the array is in the upper left or lower left corner of
    the axes.  Defaults to the rc image.origin value

This complements the axes image (Axes.imshow) which will be resampled
to fit the current axes.  If you want a resampled image to fill the
entire figure, you can define an Axes with size [0,1,0,1].

A image.FigureImage instance is returned.
```

gca(*self*, ***kwargs*)

```
Return the current axes, creating one if necessary
The following kwargs are supported
        adjustable: ['box' | 'datalim']
        alpha: float
        anchor: ['C', 'SW', 'S', 'SE', 'E', 'NE', 'N', 'NW', 'W']
        animated: [True | False]
        aspect: ['auto' | 'equal' | aspect_ratio]
        autoscale_on: True|False
        axes: an axes instance
        axis_bgcolor: any matplotlib color - see help(colors)
        axis_off: void
        axis_on: void
        axisbelow: True|False
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cursor_props: a (float, color) tuple
        figure: a Figure instance
        frame_on: True|False
        label: any string
        lod: [True | False]
        navigate: True|False
        navigate_mode: unknown
        picker: [None|float|boolean|callable]
        position: len(4) sequence of floats
        title: str
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xlabel: str
        xlim: len(2) sequence of floats
        xscale: ['log' | 'linear' ]
        xticklabels: sequence of strings
        xticks: sequence of floats
        ylabel: str
        ylim: len(2) sequence of floats
        yscale: ['log' | 'linear']
        yticklabels: sequence of strings
        yticks: sequence of floats
        zorder: any number
```

get_axes(*self*)

return the axes instance the artist resides in, or None

Overrides: matplotlib.artist.Artist.get_axes extit(inherited documentation)

get_children(*self*)

get a list of artists contained in the figure

get_dpi(*self*)

Return the dpi as a float

get_edgecolor(*self*)

Get the edge color of the Figure rectangle

get_facecolor(*self*)

Get the face color of the Figure rectangle

get_figheight(*self*)

Return the figheight as a float

get_figwidth(*self*)

Return the figwidth as a float

get_frameon(*self*)

get the boolean indicating frameon

get_size_inches(*self*)

get_window_extent(*self*, **args*, ***kwargs*)

get the figure bounding box in display space; kwargs are void

hold(*self*, *b*=`None`)

Set the hold state. If hold is None (default), toggle the hold state. Else set the hold state to boolean value b.
Eg hold() # toggle hold hold(True) # hold is on hold(False) # hold is off

legend(*self*, *handles*, *labels*, *loc*, ***kwargs*)

```
Place a legend in the figure.  Labels are a sequence of
strings, handles is a sequence of line or patch instances, and
loc can be a string or an integer specifying the legend
location

USAGE:
  legend( (line1, line2, line3),
          ('label1', 'label2', 'label3'),
          'upper right')

The LOC location codes are

  'best' : 0,          (currently not supported, defaults to upper right)
  'upper right'  : 1,  (default)
  'upper left'   : 2,
  'lower left'   : 3,
  'lower right'  : 4,
  'right'        : 5,
  'center left'  : 6,
  'center right' : 7,
  'lower center' : 8,
  'upper center' : 9,
  'center'       : 10,

loc can also be an (x,y) tuple in figure coords, which
specifies the lower left of the legend box.  figure coords are
(0,0) is the left, bottom of the figure and 1,1 is the right,
top.

The legend instance is returned.  The following kwargs are supported:

isaxes=True           # whether this is an axes legend
numpoints = 4         # the number of points in the legend line
prop = FontProperties(size='smaller')  # the font property
pad = 0.2             # the fractional whitespace inside the legend border
markerscale = 0.6     # the relative size of legend markers vs. original
shadow                # if True, draw a shadow behind legend
labelsep = 0.005     # the vertical space between the legend entries
handlelen = 0.05     # the length of the legend lines
handletextsep = 0.02 # the space between the legend line and legend text
axespad = 0.02       # the border between the axes and legend edge
```

pick(*self*, *mouseevent*)

the user picked location x,y; if this Artist is within picker "pick epsilon" of x,y fire off a pick event

Overrides: matplotlib.artist.Artist.pick extit(inherited documentation)

savefig(*self*, **args*, ***kwargs*)

```
SAVEFIG(fname, dpi=None, facecolor='w', edgecolor='w',
orientation='portrait', papertype=None, format=None):

Save the current figure.

fname - the filename to save the current figure to.  The
        output formats supported depend on the backend being
        used.  and are deduced by the extension to fname.
        Possibilities are eps, jpeg, pdf, png, ps, svg.  fname
        can also be a file or file-like object - cairo backend
        only.  dpi - is the resolution in dots per inch.  If
        None it will default to the value savefig.dpi in the
        matplotlibrc file

facecolor and edgecolor are the colors of the figure rectangle

orientation is either 'landscape' or 'portrait' - not supported on
all backends; currently only on postscript output

papertype is is one of 'letter', 'legal', 'executive', 'ledger', 'a0'
through 'a10', or 'b0' through 'b10' - only supported for postscript
output

format - one of 'pdf', 'png', 'ps', 'svg'. It is used to specify the
         output when fname is a file or file-like object - cairo
         backend only.
```

sca(*self*, *a*)

Set the current axes to be a and return a

set_canvas(*self*, *canvas*)

Set the canvas the contains the figure
ACCEPTS: a FigureCanvas instance

set_dpi(*self*, *val*)

Set the dots-per-inch of the figure
ACCEPTS: float

set_edgecolor(*self*, *color*)

Set the edge color of the Figure rectangle
ACCEPTS: any matplotlib color - see help(colors)

set_facecolor(*self*, *color*)

Set the face color of the Figure rectangle
ACCEPTS: any matplotlib color - see help(colors)

set_figheight(*self*, *val*)

Set the height of the figure in inches
ACCEPTS: float

set_figsize_inches(*self*, **args*, ***kwargs*)

set_figwidth(*self*, *val*)

Set the width of the figure in inches
ACCEPTS: float

set_frameon(*self*, *b*)

Set whether the figure frame (background) is displayed or invisible
ACCEPTS: boolean

set_size_inches(*self*, **args*, ***kwargs*)

```
set_size_inches(w,h, forward=False)

Set the figure size in inches

Usage: set_size_inches(self, w,h)  OR
       set_size_inches(self, (w,h) )

optional kwarg forward=True will cause the canvas size to be
automatically updated; eg you can resize the figure window
from the shell

WARNING: forward=True is broken on all backends except GTK*

ACCEPTS: a w,h tuple with w,h in inches
```

subplots_adjust(*self*, **args*, ***kwargs*)

```
subplots_adjust(self, left=None, bottom=None, right=None, top=None,
                wspace=None, hspace=None)
fig.subplots_adjust(left=None, bottom=None, right=None, wspace=None, hspace=None):
Update the SubplotParams with kwargs (defaulting to rc where
None) and update the subplot locations
```

text(*self*, *x*, *y*, *s*, **args*, ***kwargs*)

```
Add text to figure at location x,y (relative 0-1 coords) See
the help for Axis text for the meaning of the other arguments
kwargs control the Text properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
```

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

19.2.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

19.3 Class SubplotParams

A class to hold the parameters for a subplot

19.3.1 Methods

__init__(*self*, *left*=`None`, *bottom*=`None`, *right*=`None`, *top*=`None`, *wspace*=`None`, *hspace*=`None`)

All dimensions are fraction of the figure width or height. All values default to their rc params
The following attributes are available:
left : the left side of the subplots of the figure right : the right side of the subplots of the figure bottom : the bottom of the subplots of the figure top : the top of the subplots of the figure wspace : the amount of width reserved for blank space between subplots hspace : the amount of height reserved for white space between subplots
validate : make sure the params are in a legal state (left<right, etc)

update(*self*, *left*=`None`, *bottom*=`None`, *right*=`None`, *top*=`None`, *wspace*=`None`, *hspace*=`None`)

Update the current values. If any kwarg is None, default to the current value, if set, otherwise to rc

20 Module matplotlib.finance

A collection of modules for collecting, analyzing and plotting financial data. User contributions welcome!

20.1 Functions

candlestick(*ax*, *quotes*, *width*=`0.20000000000000001`, *colorup*=`'k'`, *colordown*=`'r'`, *alpha*=`1.0`)

quotes is a list of (time, open, close, high, low, ...) tuples. As long as the first 5 elements of the tuples are these values, the tuple can be as long as you want (eg it may store volume).
time must be in float days format - see date2num
Plot the time, open, close, high, low as a vertical line ranging from low to high. Use a rectangular bar to represent the open-close span. If close >= open, use colorup to color the bar, otherwise use colordown
ax : an Axes instance to plot to width : fraction of a day for the rectangle width colorup : the color of the rectangle where close >= open colordown : the color of the rectangle where close < open alpha : the rectangle alpha level
return value is lines, patches where lines is a list of lines added and patches is a list of the rectangle patches added

candlestick2(*ax*, *opens*, *closes*, *highs*, *lows*, *width*=`4`, *colorup*=`'k'`, *colordown*=`'r'`, *alpha*=`0.75`)

Represent the open, close as a bar line and high low range as a vertical line.
ax : an Axes instance to plot to width : the bar width in points colorup : the color of the lines where close >= open colordown : the color of the lines where close < open alpha : bar transparency
return value is lineCollection, barCollection

fetch_historical_yahoo(*ticker*, *date1*, *date2*, *cachename*=`None`)

Fetch historical data for ticker between date1 and date2. date1 and date2 are datetime instances
Ex: fh = fetch_historical_yahoo('^GSPC', d1, d2)
cachename is the name of the local file cache. If None, will default to the md5 hash or the url (which incorporates the ticker and date range)
a file handle is returned

index_bar(*ax*, *vals*, *facecolor*=`'b'`, *edgecolor*=`'l'`, *width*=`4`, *alpha*=`1.0`)

Add a bar collection graph with height vals (-1 is missing).
ax : an Axes instance to plot to width : the bar width in points alpha : bar transparency

parse_yahoo_historical(*fh*, *asobject*=`False`, *adjusted*=`True`)

Parse the historical data in file handle fh from yahoo finance and return results as a list of
d, open, close, high, low, volume
where d is a floating poing representation of date, as returned by date2num
if adjust=True, use adjusted prices

plot_day_summary(*ax*, *quotes*, *ticksize*=3, *colorup*=`'k'`, *colordown*=`'r'`)

quotes is a list of (time, open, close, high, low, ...) tuples
Represent the time, open, close, high, low as a vertical line ranging from low to high. The left tick is the open and the right tick is the close.
time must be in float date format - see date2num
ax : an Axes instance to plot to ticksize : open/close tick marker in points colorup : the color of the lines where close >= open colordown : the color of the lines where close < open return value is a list of lines added

plot_day_summary2(*ax*, *opens*, *closes*, *highs*, *lows*, *ticksize*=4, *colorup*=`'k'`, *colordown*=`'r'`)

Represent the time, open, close, high, low as a vertical line ranging from low to high. The left tick is the open and the right tick is the close.
ax : an Axes instance to plot to ticksize : size of open and close ticks in points colorup : the color of the lines where close >= open colordown : the color of the lines where close < open
return value is a list of lines added

quotes_historical_yahoo(*ticker*, *date1*, *date2*, *asobject*=`False`, *adjusted*=`True`, *cachename*=`None`)

```
Get historical data for ticker between date1 and date2.  date1 and
date2 are datetime instances

results are a list of tuples

  (d, open, close, high, low, volume)

where d is a floating poing representation of date, as returned by date2num

if asobject is True, the return val is an object with attrs date,
open, close, high, low, volume, which are equal length arrays

if adjust=True, use adjusted prices

Ex:
sp = f.quotes_historical_yahoo('^GSPC', d1, d2, asobject=True, adjusted=True)
returns = (sp.open[1:] - sp.open[:-1])/sp.open[1:]
[n,bins,patches] = hist(returns, 100)
mu = mean(returns)
sigma = std(returns)
x = normpdf(bins, mu, sigma)
plot(bins, x, color='red', lw=2)

cachename is the name of the local file cache.  If None, will
default to the md5 hash or the url (which incorporates the ticker
and date range)
```

volume_overlay(*ax*, *opens*, *closes*, *volumes*, *colorup*=`'k'`, *colordown*=`'r'`, *width*=`4`, *alpha*=`1.0`)

Add a volume overlay to the current axes. The opens and closes are used to determine the color of the bar. -1 is missing. If a value is missing on one it must be missing on all
ax : an Axes instance to plot to width : the bar width in points colorup : the color of the lines where close >= open colordown : the color of the lines where close < open alpha : bar transparency

volume_overlay2(*ax*, *closes*, *volumes*, *colorup*=`'k'`, *colordown*=`'r'`, *width*=`4`, *alpha*=`1.0`)

Add a volume overlay to the current axes. The closes are used to determine the color of the bar. -1 is missing. If a value is missing on one it must be missing on all
ax : an Axes instance to plot to width : the bar width in points colorup : the color of the lines where close >= open colordown : the color of the lines where close < open alpha : bar transparency
nb: first point is not displayed - it is used only for choosing the right color

volume_overlay3(*ax*, *quotes*, *colorup*=`'k'`, *colordown*=`'r'`, *width*=`4`, *alpha*=`1.0`)

Add a volume overlay to the current axes. quotes is a list of (d, open, close, high, low, volume) and close-open is used to determine the color of the bar
kwarg width : the bar width in points colorup : the color of the lines where close1 >= close0 colordown : the color of the lines where close1 < close0 alpha : bar transparency

21 Module matplotlib.font_manager

```
A module for finding, managing, and using fonts across-platforms.

This module provides a single FontManager that can be shared across
backends and platforms.  The findfont() method returns the best
TrueType (TTF) font file in the local or system font path that matches
the specified FontProperties.  The FontManager also handles Adobe Font
Metrics (AFM) font files for use by the PostScript backend.

The design is based on the W3C Cascading Style Sheet, Level 1 (CSS1)
font specification (http://www.w3.org/TR/1998/REC-CSS2-19980512/ ).
Future versions may implement the Level 2 or 2.1 specifications.


KNOWN ISSUES

  - documentation
  - font variant is untested
  - font stretch is incomplete
  - font size is incomplete
  - font size_adjust is incomplete
  - default font algorithm needs improvement and testing
  - setWeights function needs improvement
  - 'light' is an invalid weight value, remove it.
  - update_fonts not implemented


Authors   : John Hunter <jdhunter@ace.bsd.uchicago.edu>
            Paul Barrett <Barrett@STScI.Edu>
Copyright : John Hunter (2004,2005), Paul Barrett (2004,2005)
License   : matplotlib license (PSF compatible)
            The font directory code is from ttfquery,
            see license/LICENSE_TTFQUERY.
```

21.1 Functions

add_filename(*fontdict*, *prop*, *fname*)

A function to add a font file name to the font dictionary using the FontKey properties. If a font property has no dictionary, then create it.

afmFontProperty(*font*)

A function for populating the FontKey by extracting information from the AFM font file.

createFontDict(*fontfiles*, *fontext*=`'ttf'`)

A function to create a dictionary of font file paths. The default is to create a dictionary for TrueType fonts. An AFM font dictionary can optionally be created.

findSystemFonts(*fontpaths*=`None`, *fontext*=`'ttf'`)

Search for fonts in the specified font paths, or use the system paths if none given. A list of TrueType fonts are returned by default with AFM fonts as an option.

OSXFontDirectory()

Return the system font directories for OS X.

OSXInstalledFonts(*directory*=`None`, *fontext*=`None`)

Get list of font files on OS X - ignores font suffix by default

pickle_dump(*data*, *filename*)

Equivalent to pickle.dump(data, open(filename, 'w')) but closes the file to prevent filehandle leakage.

pickle_load(*filename*)

Equivalent to pickle.load(open(filename, 'r')) but closes the file to prevent filehandle leakage.

setWeights(*font*)

A function to populate missing values in a font weight dictionary. This proceedure is necessary since the font finding algorithm always matches on the weight property.

ttfdict_to_fnames(*d*)

flatten a ttfdict to all the filenames it contains

ttfFontProperty(*font*)

A function for populating the FontKey by extracting information from the TrueType font file.

weight_as_number(*weight*)

Return the weight property as a numeric value. String values are converted to their corresponding numeric value.

win32FontDirectory()

Return the user-specified font directory for Win32.

win32InstalledFonts(*directory*=`None`, *fontext*=`'ttf'`)

Search for fonts in the specified font directory, or use the system directories if none given. A list of TrueType fonts are returned by default with AFM fonts as an option.

x11FontDirectory()

Return the system font directories for X11.

21.2 Class FontKey

__builtin__.object
└ **FontKey**

A class for storing Font properties. It is used when populating the font dictionary.

21.2.1 Methods

__init__(*self*, *name*=`''`, *style*=`'normal'`, *variant*=`'normal'`, *weight*=`'normal'`, *stretch*=`'normal'`, *size*=`'medium'`)
Overrides: __builtin__.object.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __repr__, __setattr__, __str__

21.3 Class FontManager

On import, the FontManager creates a dictionary of TrueType fonts based on the font properties: name, style, variant, weight, stretch, and size. The findfont() method searches this dictionary for a font file name that exactly matches the font properties of the specified text. If none is found, a default font is returned. By updating the dictionary with the properties of the found font, the font dictionary can act like a font cache.

21.3.1 Methods

__init__(*self*, *size*=`None`, *weight*=`'normal'`)

findfont(*self*, *prop*, *fontext*=`'ttf'`)

Search the font dictionary for a font that exactly or closely matches the specified font properties. See the FontProperties class for a description.
The properties are searched in the following order: name, style, variant, weight, stretch, and size. The font weight always matches returning the closest weight, and the font size always matches for scalable fonts. An oblique style font will be used inplace of a missing italic style font if present. See the W3C Cascading Style Sheet, Level 1 (CSS1; http://www.w3.org/TR/1998/REC-CSS2-19980512/) documentation for a description of the font finding algorithm.

get_default_size(*self*)

Return the default font size.

get_default_weight(*self*)

Return the default font weight.

set_default_size(*self*, *size*)

Set the default font size in points. The initial value is set by font.size in rc.

set_default_weight(*self*, *weight*)

Set the default font weight. The initial value is 'normal'.

update_fonts(*self*, *filenames*)

Update the font dictionary with new font files. Currently not implemented.

21.4 Class FontProperties

```
A class for storing and manipulating font properties.

The font properties are those described in the W3C Cascading Style
Sheet, Level 1 (CSS1; http://www.w3.org/TR/1998/REC-CSS2-19980512/)
font specification.  The six properties are:

  family  - A list of font names in decreasing order of priority.
            The last item is the default font name and is given the
            name of the font family, either serif, sans-serif,
            cursive, fantasy, and monospace.
  style   - Either normal, italic or oblique.
  variant - Either normal or small-caps.
  stretch - Either an absolute value of ultra-condensed, extra-
            condensed, condensed, semi-condensed, normal, semi-
            expanded, expanded, extra-expanded or ultra-expanded;
            or a relative value of narrower or wider.
            This property is currently not implemented and is set to
            normal.
  weight  - A numeric value in the range 100, 200, 300, ..., 900.
  size    - Either an absolute value of xx-small, x-small, small,
            medium, large, x-large, xx-large; or a relative value
            of smaller or larger; or an absolute font size, e.g. 12;
            or scalable.

The default font property for TrueType fonts is: sans-serif, normal,
normal, normal, 400, scalable.
```

```
The preferred usage of font sizes is to use the absolute values, e.g.
large, instead of absolute font sizes, e.g. 12.  This approach allows
all text sizes to be made larger or smaller based on the font manager's
default font size, i.e. by using the set_default_size() method of the
font manager.

Examples:

  #  Load default font properties
  >>> p = FontProperties()
  >>> p.get_family()
  ['Bitstream Vera Sans', 'Lucida Grande', 'Verdana', 'Geneva', 'Lucida', 'Arial', 'Helvetica', 'sans-se

  #  Change font family to 'fantasy'
  >>> p.set_family('fantasy')
  >>> p.get_family()
  ['Comic Sans MS', 'Chicago', 'Charcoal', 'Impact', 'Western', 'fantasy']

  #  Make these fonts highest priority in font family
  >>> p.set_name(['foo', 'fantasy', 'bar', 'baz'])
  Font name 'fantasy' is a font family. It is being deleted from the list.
  >>> p.get_family()
  ['foo', 'bar', 'baz', 'Comic Sans MS', 'Chicago', 'Charcoal', 'Impact', 'Western', 'fantasy']
```

21.4.1 Methods

__init__(*self*, *family*=None, *style*=None, *variant*=None, *weight*=None, *stretch*=None, *size*=None, *fname*=None)

__hash__(*self*)

__str__(*self*)

copy(*self*)

Return a deep copy of self

get_family(*self*)

Return a list of font names that comprise the font family.

get_name(*self*)

Return the name of the font that best matches the font properties.

get_size(*self*)

Return the font size.

get_size_in_points(*self*, *parent_size*=`None`)

Return the size property as a numeric value. String values are converted to their corresponding numeric value.

get_stretch(*self*)

Return the font stretch or width. Options are: normal, narrow, condensed, or wide.

get_style(*self*)

Return the font style. Values are: normal, italic or oblique.

get_variant(*self*)

Return the font variant. Values are: normal or small-caps.

get_weight(*self*)

Return the font weight. See the FontProperties class for a a list of possible values.

set_family(*self*, *family*)

Change the font family. Options are: serif, sans-serif, cursive, fantasy, or monospace.

set_name(*self*, *names*)

Add one or more font names to the font family list. If the font name is already in the list, then the font is given a higher priority in the font family list. To change the font family, use the set_family() method.

set_size(*self*, *size*)

Set the font size.

set_stretch(*self*, *stretch*)

Set the font stretch or width. Options are: normal, narrow, condensed, or wide.

set_style(*self*, *style*)

Set the font style. Values are: normal, italic or oblique.

set_variant(*self*, *variant*)

Set the font variant. Values are: normal or small-caps.

set_weight(*self*, *weight*)

Set the font weight. See the FontProperties class for a a list of possible values.

22 Module matplotlib.image

The image module supports basic image loading, rescaling and display operations.

22.1 Functions

imread(*fname*)

return image file in fname as numerix array
Return value is a MxNx4 array of 0-1 normalized floats

pil_to_array(*pilImage*)

22.2 Class AxesImage

matplotlib.artist.Artist
matplotlib.cm.ScalarMappable
AxesImage

Known Subclasses: NonUniformImage

22.2.1 Methods

__init__(*self*, *ax*, *cmap*=`None`, *norm*=`None`, *interpolation*=`None`, *origin*=`None`, *extent*=`None`, *filternorm*=`1`, *filterrad*=`4.0`, ***kwargs*)

interpolation and cmap default to their rc settings
cmap is a colors.Colormap instance norm is a colors.Normalize instance to map luminance to 0-1
extent is data axes (left, right, bottom, top) for making image plots registered with data plots. Default is to label the pixel centers with the zero-based row and column indices.
Additional kwargs are matplotlib.artist properties

Overrides: matplotlib.artist.Artist.__init__

changed(*self*)

Call this whenever the mappable is changed so observers can update state

Overrides: matplotlib.cm.ScalarMappable.changed

draw(*self*, *renderer*, **args*, ***kwargs*)

Derived classes drawing method

Overrides: matplotlib.artist.Artist.draw extit(inherited documentation)

get_extent(*self*)

get the image extent: left, right, bottom, top

get_filternorm(*self*)

return the filternorm setting

get_filterrad(*self*)

return the filterrad setting

get_interpolation(*self*)

Return the interpolation method the image uses when resizing.
One of
'bicubic', 'bilinear', 'blackman100', 'blackman256', 'blackman64', 'nearest', 'sinc144', 'sinc256', 'sinc64', 'spline16', 'spline36'

get_size(*self*)

Get the numrows, numcols of the input image

make_image(*self*, *magnification*=`1.0`)

pick(*self*, *mouseevent*)

return true if the data coords of mouse click are within the extent of the image

Overrides: matplotlib.artist.Artist.pick

set_alpha(*self*, *alpha*)

Set the alpha value used for blending - not supported on all backends
ACCEPTS: float

Overrides: matplotlib.artist.Artist.set_alpha

set_array(*self*, *A*)

retained for backwards compatibility - use set_data instead
ACCEPTS: numeric/numarray/PIL Image A

Overrides: matplotlib.cm.ScalarMappable.set_array

set_data(*self*, *A*, *shape*=`None`)

Set the image array
ACCEPTS: numpy/PIL Image A

set_filternorm(*self*, *filternorm*)

Set whether the resize filter norms the weights – see help for imshow
ACCEPTS: 0 or 1

set_filterrad(*self*, *filterrad*)

Set the resize filter radius only applicable to some interpolation schemes – see help for imshow
ACCEPTS: positive float

set_interpolation(*self*, *s*)

Set the interpolation method the image uses when resizing.
ACCEPTS: ['bicubic' | 'bilinear' | 'blackman100' | 'blackman256' | 'blackman64', 'nearest' | 'sinc144' | 'sinc256' | 'sinc64' | 'spline16' | 'spline36']

write_png(*self*, *fname*, *noscale*=`False`)

Write the image to png file with fname

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, get_array, get_clim, notify, set_clim, set_cmap, set_colorbar, set_norm, to_rgba

22.2.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

22.3 Class FigureImage

matplotlib.artist.Artist
matplotlib.cm.ScalarMappable
FigureImage

22.3.1 Methods

__init__(*self*, *fig*, *cmap*=`None`, *norm*=`None`, *offsetx*=`0`, *offsety*=`0`, *origin*=`None`)

cmap is a colors.Colormap instance norm is a colors.Normalize instance to map luminance to 0-1

Overrides: matplotlib.artist.Artist.__init__

---

draw(*self*, *renderer*, **args*, ***kwargs*)

Derived classes drawing method

Overrides: matplotlib.artist.Artist.draw extit(inherited documentation)

---

get_size(*self*)

Get the numrows, numcols of the input image

---

make_image(*self*, *magnification*=`1.0`)

---

write_png(*self*, *fname*)

Write the image to png file with fname

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pick, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba

22.3.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

22.4 Class NonUniformImage

matplotlib.artist.Artist
matplotlib.cm.ScalarMappable
matplotlib.image.AxesImage
NonUniformImage

22.4.1 Methods

__init__(*self*, *ax*, *cmap*=`None`, *norm*=`None`, *extent*=`None`)
Overrides: matplotlib.image.AxesImage.__init__

---

get_extent(*self*)

get the image extent: left, right, bottom, top

Overrides: matplotlib.image.AxesImage.get_extent extit(inherited documentation)

---

make_image(*self*, *magnification*=`1.0`)
Overrides: matplotlib.image.AxesImage.make_image

---

set_array(*self*, **args*)
Overrides: matplotlib.image.AxesImage.set_array

---

set_cmap(*self*, *cmap*)

set the colormap for luminance data
ACCEPTS: a colormap

Overrides: matplotlib.cm.ScalarMappable.set_cmap extit(inherited documentation)

---

set_data(*self*, *x*, *y*, *A*)
Overrides: matplotlib.image.AxesImage.set_data

---

set_filternorm(*self*, *s*)
Overrides: matplotlib.image.AxesImage.set_filternorm

---

set_filterrad(*self*, *s*)
Overrides: matplotlib.image.AxesImage.set_filterrad

---

set_interpolation(*self*, *s*)

Set the interpolation method the image uses when resizing.
ACCEPTS: ['bicubic' | 'bilinear' | 'blackman100' | 'blackman256' | 'blackman64', 'nearest' | 'sinc144' | 'sinc256' | 'sinc64' | 'spline16' | 'spline36']

Overrides: matplotlib.image.AxesImage.set_interpolation extit(inherited documentation)

---

set_norm(*self*, *norm*)

set the normalization instance

Overrides: matplotlib.cm.ScalarMappable.set_norm extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible,

set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, get_array, get_clim, notify, set_clim, set_colorbar, to_rgba
Inherited from AxesImage: changed, draw, get_filternorm, get_filterrad, get_interpolation, get_size, pick, set_alpha, write_png

22.4.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

23 Module matplotlib.legend

```
Place a legend on the axes at location loc.  Labels are a
sequence of strings and loc can be a string or an integer
specifying the legend location

The location codes are

  'best'         : 0,
  'upper right'  : 1,  (default)
  'upper left'   : 2,
  'lower left'   : 3,
  'lower right'  : 4,
  'right'        : 5,
  'center left'  : 6,
  'center right' : 7,
  'lower center' : 8,
  'upper center' : 9,
  'center'       : 10,

Return value is a sequence of text, line instances that make
up the legend
```

23.1 Functions

line_cuts_bbox(*line*, *bbox*)

Return True if and only if line cuts bbox.

23.2 Class Legend

matplotlib.artist.Artist
└ **Legend**

```
Place a legend on the axes at location loc.  Labels are a
sequence of strings and loc can be a string or an integer
specifying the legend location

The location codes are

  'best'         : 0,
  'upper right'  : 1,  (default)
  'upper left'   : 2,
  'lower left'   : 3,
  'lower right'  : 4,
  'right'        : 5,
```

```
  'center left'  : 6,
  'center right' : 7,
  'lower center' : 8,
  'upper center' : 9,
  'center'       : 10,

Return value is a sequence of text, line instances that make
up the legend
```

23.2.1 Methods

__init__(*self*, *parent*, *handles*, *labels*, *loc*, *isaxes*=None, *numpoints*=None, *prop*=None, *pad*=None, *markerscale*=None, *labelsep*=None, *handlelen*=None, *handletextsep*=None, *axespad*=None, *shadow*=None)

```
parent                # the artist that contains the legend
  handles               # a list of artists (lines, patches) to add to the legend
  labels                # a list of strings to label the legend
  loc                   # a location code
  isaxes=True           # whether this is an axes legend
  numpoints = 4         # the number of points in the legend line
  prop = FontProperties(size='smaller')  # the font property
  pad = 0.2             # the fractional whitespace inside the legend border
  markerscale = 0.6     # the relative size of legend markers vs. original
  shadow                # if True, draw a shadow behind legend

The following dimensions are in axes coords
  labelsep = 0.005     # the vertical space between the legend entries
  handlelen = 0.05     # the length of the legend lines
  handletextsep = 0.02 # the space between the legend line and legend text
  axespad = 0.02       # the border between the axes and legend edge
```

Overrides: matplotlib.artist.Artist.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

draw_frame(*self*, *b*)

b is a boolean. Set draw frame to b

get_frame(*self*)

return the Rectangle instance used to frame the legend

get_lines(*self*)

return a list of lines.Line2D instances in the legend

get_patches(*self*)

return a list of patch instances in the legend

get_texts(*self*)

return a list of text.Text instance in the legend

get_window_extent(*self*)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pick, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

23.2.2 Class Variables

Name	Description
codes	**Value:** `{'right': 5, 'center': 10, 'lower left': 3, 'center right': 7, 'upper left': ...` *(type=`dict`)*
zorder	**Value:** `5` *(type=`int`)*
Inherited from Artist: aname *(p. 92)*	

24 Module matplotlib.lines

This module contains all the 2D line class which can draw with a variety of line styles, markers and colors

24.1 Functions

unmasked_index_ranges(*mask*, *compressed*=True)

```
Calculate the good data ranges in a masked 1-D array, based on mask.

Returns Nx2 array with each row the start and stop indices
for slices of the compressed array corresponding to each of N
uninterrupted runs of unmasked values.
If optional argument compressed is False, it returns the
start and stop indices into the original array, not the
compressed array.
Returns None if there are no unmasked values.

Example:

y = ma.array(arange(5), mask = [0,0,1,0,0])
#ii = unmasked_index_ranges(y.mask())
ii = unmasked_index_ranges(ma.getmask(y))
    # returns [[0,2,] [2,4,]]

y.compressed().filled()[ii[1,0]:ii[1,1]]
    # returns array [3,4,]
    # (The 'filled()' method converts the masked array to a numerix array.)

#i0, i1 = unmasked_index_ranges(y.mask(), compressed=False)
i0, i1 = unmasked_index_ranges(ma.getmask(y), compressed=False)
    # returns [[0,3,] [2,5,]]

y.filled()[ii[1,0]:ii[1,1]]
    # returns array [3,4,]
```

24.2 Class Line2D

matplotlib.artist.Artist
    Line2D

Known Subclasses: Line3D

24.2.1 Methods

__init__(*self*, *xdata*, *ydata*, *linewidth*=None, *linestyle*=None, *color*=None, *marker*=None, *markersize*=None, *markeredgewidth*=None, *markeredgecolor*=None, *markerfacecolor*=None, *antialiased*=None, *dash_capstyle*=None, *solid_capstyle*=None, *dash_joinstyle*=None, *solid_joinstyle*=None, ***kwargs*)

```
Create a Line2D instance with x and y data in sequences xdata,
ydata

The kwargs are Line2D properties:
  alpha: float
  animated: [True | False]
  antialiased or aa: [True | False]
  clip_box: a matplotlib.transform.Bbox instance
  clip_on: [True | False]
  color or c: any matplotlib color
  dash_capstyle: ['butt' | 'round' | 'projecting']
  dash_joinstyle: ['miter' | 'round' | 'bevel']
  dashes: sequence of on/off ink in points
  data: (array xdata, array ydata)
  figure: a matplotlib.figure.Figure instance
  label: any string
  linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
  linewidth or lw: float value in points
  lod: [True | False]
  marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
  markeredgecolor or mec: any matplotlib color
  markeredgewidth or mew: float value in points
  markerfacecolor or mfc: any matplotlib color
  markersize or ms: float
  solid_capstyle: ['butt' | 'round' |  'projecting']
  solid_joinstyle: ['miter' | 'round' | 'bevel']
  transform: a matplotlib.transform transformation instance
  visible: [True | False]
  xdata: array
  ydata: array
  zorder: any number
```

Overrides: matplotlib.artist.Artist.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_aa(*self*)

alias for get_antialiased

get_antialiased(*self*)

get_c(*self*)

alias for get_color

get_color(*self*)

get_dash_capstyle(*self*)

Get the cap style for dashed linestyles

get_dash_joinstyle(*self*)

Get the join style for dashed linestyles

get_linestyle(*self*)

get_linewidth(*self*)

get_ls(*self*)

alias for get_linestyle

get_lw(*self*)

alias for get_linewidth

get_marker(*self*)

get_markeredgecolor(*self*)

get_markeredgewidth(*self*)

get_markerfacecolor(*self*)

get_markersize(*self*)

get_mec(*self*)

alias for get_markeredgecolor

get_mew(*self*)

alias for get_markeredgewidth

get_mfc(*self*)

alias for get_markerfacecolor

get_ms(*self*)

alias for get_markersize

get_solid_capstyle(*self*)

Get the cap style for solid linestyles

get_solid_joinstyle(*self*)

Get the join style for solid linestyles

get_window_extent(*self*, *renderer*)

get_xdata(*self*, *orig*=`True`)

return the xdata; if orig is true return the original data, else the processed data

get_ydata(*self*, *orig*=`True`)

return the ydata; if orig is true return the original data, else the processed data

is_dashed(*self*)

return True if line is dashstyle

pick(*self*, *mouseevent*)

If mouseevent is over data that satisifies the picker, fire off a backend_bases.PickEvent with the additional attribute "ind" which is a sequence of indices into the data that meet the criteria

Overrides: matplotlib.artist.Artist.pick

recache(*self*)

set_aa(*self*, *val*)

alias for set_antialiased

set_antialiased(*self*, *b*)

True if line should be drawin with antialiased rendering
ACCEPTS: [True | False]

set_axes(*self*, *ax*)
Overrides: matplotlib.artist.Artist.set_axes

set_c(*self*, *val*)

alias for set_color

set_color(*self*, *color*)

Set the color of the line
ACCEPTS: any matplotlib color

set_dash_capstyle(*self*, *s*)

Set the cap style for dashed linestyles ACCEPTS: ['butt' | 'round' | 'projecting']

set_dash_joinstyle(*self*, *s*)

Set the join style for dashed linestyles ACCEPTS: ['miter' | 'round' | 'bevel']

set_dashes(*self*, *seq*)

Set the dash sequence, sequence of dashes with on off ink in points. If seq is empty or if seq = (None, None), the linestyle will be set to solid.
ACCEPTS: sequence of on/off ink in points

set_data(*self*, **args*)

Set the x and y data
ACCEPTS: (array xdata, array ydata)

set_linestyle(*self*, *linestyle*)

Set the linestyle of the line
ACCEPTS: ['-' | '–' | '-.' | ':' | 'steps' | 'None' | ' ' | "]

set_linewidth(*self*, *w*)

Set the line width in points
ACCEPTS: float value in points

set_ls(*self*, *val*)

alias for set_linestyle

set_lw(*self*, *val*)

alias for set_linewidth

set_marker(*self*, *marker*)

```
Set the line marker

ACCEPTS: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
         | '<' | '>' | 'D' | 'H' | '^' | '_' | 'd'
         | 'h' | 'o' | 'p' | 's' | 'v' | 'x' | '|'
         | TICKUP | TICKDOWN | TICKLEFT | TICKRIGHT
         | 'None' | ' ' | '' ]
```

set_markeredgecolor(*self*, *ec*)

Set the marker edge color
ACCEPTS: any matplotlib color

set_markeredgewidth(*self*, *ew*)

Set the marker edge width in points
ACCEPTS: float value in points

set_markerfacecolor(*self*, *fc*)

Set the marker face color
ACCEPTS: any matplotlib color

set_markersize(*self*, *sz*)

Set the marker size in points
ACCEPTS: float

set_mec(*self*, *val*)

alias for set_markeredgecolor

set_mew(*self*, *val*)

alias for set_markeredgewidth

set_mfc(*self*, *val*)

alias for set_markerfacecolor

set_ms(*self*, *val*)

alias for set_markersize

set_solid_capstyle(*self*, *s*)

Set the cap style for solid linestyles ACCEPTS: ['butt' | 'round' | 'projecting']

set_solid_joinstyle(*self*, *s*)

Set the join style for solid linestyles ACCEPTS: ['miter' | 'round' | 'bevel']

set_xdata(*self*, *x*)

Set the data array for x
ACCEPTS: array

set_ydata(*self*, *y*)

Set the data array for y
ACCEPTS: array

update_from(*self*, *other*)

copy properties from other to self

Overrides: matplotlib.artist.Artist.update_from

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update

24.2.2 Class Variables

Name	Description
filled_markers	**Value:** `('o', '^', 'v', '<', '>', 's', 'd', 'D', 'h', -'H', 'p')` *(type=`tuple`)*
validCap	**Value:** `('butt', 'round', 'projecting')` *(type=`tuple`)*
validJoin	**Value:** `('miter', 'round', 'bevel')` *(type=`tuple`)*
zorder	**Value:** `2` *(type=`int`)*
Inherited from Artist: aname *(p. 92)*	

25 Module matplotlib.mathtext

```
OVERVIEW

  mathtext is a module for parsing TeX expressions and drawing them
  into a matplotlib.ft2font image buffer.  You can draw from this
  buffer into your backend.

  A large set of the TeX symbols are provided (see below).
  Subscripting and superscripting are supported, as well as the
  over/under style of subscripting with \sum, \int, etc.

  The module uses pyparsing to parse the TeX expression, an so can
  handle fairly complex TeX expressions Eg, the following renders
  correctly

  s = r'$\cal{R}\prod_{i=\alpha\cal{B}}^\infty a_i\rm{sin}(2 \pi f x_i)$'

  The fonts \cal, \rm, \it, and \tt are allowed.

  The following accents are provided: \hat, \breve, \grave, \bar,
  \acute, \tilde, \vec, \dot, \ddot.  All of them have the same
  syntax, eg to make an overbar you do \bar{o} or to make an o umlaut
  you do \ddot{o}.  The shortcuts are also provided, eg: \"o \'e \`e
  \~n \.x \^y

  The spacing elements \ , \/ and \hspace{num} are provided.  \/
  inserts a small space, and \hspace{num} inserts a fraction of the
  current fontsize.  Eg, if num=0.5 and the fontsize is 12.0,
  hspace{0.5} inserts 6 points of space



  If you find TeX expressions that don't parse or render properly,
  please email me, but please check KNOWN ISSUES below first.

REQUIREMENTS

  mathtext requires matplotlib.ft2font.  Set BUILD_FT2FONT=True in
  setup.py.  See BACKENDS below for a summary of availability by
  backend.

LICENSING:

  The computer modern fonts this package uses are part of the BaKoMa
  fonts, which are (now) free for commercial and noncommercial use and
  redistribution; see license/LICENSE_BAKOMA in the matplotlib src
  distribution for redistribution requirements.
```

```
USAGE:

  See http://matplotlib.sourceforge.net/tutorial.html#mathtext for a
  tutorial introduction.

  Any text element (xlabel, ylabel, title, text, etc) can use TeX
  markup, as in

    xlabel(r'$\Delta_i$')
           ^
        use raw strings

  The $ symbols must be the first and last symbols in the string.  Eg,
  you cannot do

    r'My label $x_i$'.

  but you can change fonts, as in

    r'$\rm{My label} x_i$'

  to achieve the same effect.

  A large set of the TeX symbols are provided.  Subscripting and
  superscripting are supported, as well as the over/under style of
  subscripting with \sum, \int, etc.


  Allowed TeX symbols:

  \/ \Delta \Downarrow \Gamma \Im \LEFTangle \LEFTbrace \LEFTbracket
  \LEFTparen \Lambda \Leftarrow \Leftbrace \Leftbracket \Leftparen
  \Leftrightarrow \Omega \P \Phi \Pi \Psi \RIGHTangle \RIGHTbrace
  \RIGHTbracket \RIGHTparen \Re \Rightarrow \Rightbrace \Rightbracket
  \Rightparen \S \SQRT \Sigma \Sqrt \Theta \Uparrow \Updownarrow
  \Upsilon \Vert \Xi \aleph \alpha \approx \angstrom \ast \asymp
  \backslash \beta \bigcap \bigcirc \bigcup \bigodot \bigoplus
  \bigotimes \bigtriangledown \bigtriangleup \biguplus \bigvee
  \bigwedge \bot \bullet \cap \cdot \chi \circ \clubsuit \coprod \cup
  \dag \dashv \ddag \delta \diamond \diamondsuit \div \downarrow \ell
  \emptyset \epsilon \equiv \eta \exists \flat \forall \frown \gamma
  \geq \gg \heartsuit \hspace \imath \in \infty \int \iota \jmath
  \kappa \lambda \langle \lbrace \lceil \leftangle \leftarrow
  \leftbrace \leftbracket \leftharpoondown \leftharpoonup \leftparen
  \leftrightarrow \leq \lfloor \ll \mid \mp \mu \nabla \natural
  \nearrow \neg \ni \nu \nwarrow \odot \oint \omega \ominus \oplus
  \oslash \otimes \phi \pi \pm \prec \preceq \prime \prod \propto \psi
```

```
  \rangle \rbrace \rceil \rfloor \rho \rightangle \rightarrow
  \rightbrace \rightbracket \rightharpoondown \rightharpoonup
  \rightparen \searrow \sharp \sigma \sim \simeq \slash \smile
  \spadesuit \sqcap \sqcup \sqrt \sqsubseteq \sqsupseteq \subset
  \subseteq \succ \succeq \sum \supset \supseteq \swarrow \tau \theta
  \times \top \triangleleft \triangleright \uparrow \updownarrow
  \uplus \upsilon \varepsilon \varphi \varphi \varrho \varsigma
  \vartheta \vdash \vee \vert \wedge \wp \wr \xi \zeta


BACKENDS

  mathtext currently works with GTK, Agg, GTKAgg, TkAgg and WxAgg and
  PS, though only horizontal and vertical rotations are supported in
  *Agg

  mathtext now embeds the TrueType computer modern fonts into the PS
  file, so what you see on the screen should be what you get on paper.

  Backends which don't support mathtext will just render the TeX
  string as a literal.  Stay tuned.


KNOWN ISSUES:

 - nested subscripts, eg, x_i_j not working; but you can do x_{i_j}
 - nesting fonts changes in sub/superscript groups not parsing
 - I would also like to add a few more layout commands, like \frac.

Author    : John Hunter <jdhunter@ace.bsd.uchicago.edu>
Copyright : John Hunter (2004,2005)
License   : matplotlib license (PSF compatible)
```

25.1 Functions

font_open(*filename*)

get_type1_name(*symbol*)

get_type1_name(symbol) -> string
Returns the the Type1 name of symbol. symbol can be a single unicode character, or a TeX command (i.e. r'\pi').

get_unicode_index(*symbol*)

get_unicode_index(symbol) -> integer
Return the integer index (from the Unicode table) of symbol. symbol can be a single unicode character, a TeX command (i.e. r'\pi'), or a Type1 symbol name (i.e. 'phi').

25.2 Class BakomaPDFFonts

matplotlib.mathtext.Fonts
matplotlib.mathtext.BakomaPSFonts
BakomaPDFFonts

Hack of BakomaPSFonts for PDF support.

25.2.1 Methods

render(*self*, *ox*, *oy*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.BakomaPSFonts.render

Inherited from BakomaPSFonts: __init__, get_metrics, set_canvas_size
Inherited from Fonts: get_kern

25.2.2 Class Variables

Name	Description
Inherited from BakomaPSFonts: basepath *(p. 311)*, facenames *(p. 311)*, fontmap *(p. 311)*	

25.3 Class BakomaPSFonts

matplotlib.mathtext.Fonts
BakomaPSFonts

Known Subclasses: BakomaPDFFonts

Use the Bakoma postscript fonts for rendering to backend_ps

25.3.1 Methods

__init__(*self*)

get_metrics(*self*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.get_metrics

render(*self*, *ox*, *oy*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.render

set_canvas_size(*self*, *w*, *h*, *pswriter*)

Dimension the drawing canvas; may be a noop

Overrides: matplotlib.mathtext.Fonts.set_canvas_size

Inherited from Fonts: get_kern

25.3.2 Class Variables

Name	Description
basepath	**Value:** `'/home/jdhunter/dev/lib/python2.5/site-packages-/matplotlib/mpl-data/fonts/ttf'` *(type=`str`)*
facenames	**Value:** `('cmmi10', 'cmsy10', 'cmex10', 'cmtt10', 'cmr10')` *(type=`tuple`)*
fontmap	**Value:** `{None: 'cmmi10', 'rm': 'cmr10', 'tt': 'cmtt10',-'it': 'cmmi10', 'cal': 'cmsy10'}` *(type=`dict`)*

25.4 Class BakomaTrueTypeFonts

matplotlib.mathtext.Fonts
└ **BakomaTrueTypeFonts**

Use the Bakoma true type fonts for rendering

25.4.1 Methods

__init__(*self*, *useSVG*=`False`)

get_metrics(*self*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.get_metrics

render(*self*, *ox*, *oy*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.render

set_canvas_size(*self*, *w*, *h*)

Dimension the drawing canvas; may be a noop

Overrides: matplotlib.mathtext.Fonts.set_canvas_size

Inherited from Fonts: get_kern

25.4.2 Class Variables

Name	Description
basepath	**Value:** `'/home/jdhunter/dev/lib/python2.5/site-packages-/matplotlib/mpl-data/fonts/ttf'` *(type=str)*
fnames	**Value:** `('cmmi10', 'cmsy10', 'cmex10', 'cmtt10', 'cmr10')` *(type=tuple)*
fontmap	**Value:** `{None: 'cmmi10', 'rm': 'cmr10', 'tt': 'cmtt10',-'it': 'cmmi10', 'cal': 'cmsy10'}` *(type=dict)*

25.5 Class BakomaUnicodeFonts

matplotlib.mathtext.Fonts

matplotlib.mathtext.UnicodeFonts

BakomaUnicodeFonts

A class that simulates Unicode support in the BaKoMa fonts

25.5.1 Methods

Inherited from Fonts: get_kern
Inherited from UnicodeFonts: __init__, get_metrics, render, set_canvas_size

25.5.2 Class Variables

Name	Description
filenamesd	**Value:** `{None: 'cmmi10.ttf', 'rm': 'cmr10.ttf', 'tt': '-cmtt10.ttf', 'it': 'cmmi10.ttf...` *(type=dict)*

25.6 Class CMUUnicodeFonts

matplotlib.mathtext.Fonts

matplotlib.mathtext.UnicodeFonts

CMUUnicodeFonts

A class representing Computer Modern Unicode Fonts, made by Andrey V. Panov panov /at/ canopus. iacp. dvo. ru They are distributed under the X11 License.

25.6.1 Methods

Inherited from Fonts: get_kern
Inherited from UnicodeFonts: __init__, get_metrics, render, set_canvas_size

25.7 Class DummyFonts

matplotlib.mathtext.Fonts
  └ **DummyFonts**

dummy class for debugging parser

25.7.1 Methods

get_metrics(*self*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.get_metrics

Inherited from Fonts: get_kern, render, set_canvas_size

25.8 Class Element

Known Subclasses: GroupElement, SpaceElement, SymbolElement

25.8.1 Methods

__init__(*self*)

__repr__(*self*)

advance(*self*)

get the horiz advance

centerx(*self*)

centery(*self*)

height(*self*)

get the element height: ymax-ymin

padx(*self*)

pady(*self*)

render(*self*)

render to the fonts canvas

set_font(*self*, *font*)

set the font (one of tt, it, rm , cal)

set_origin(*self*, *ox*, *oy*)

set_padx(*self*, *pad*)

set the y padding in points

set_pady(*self*, *pad*)

set the y padding in points

set_scale(*self*, *scale*)

scale the element by scale

set_size_info(*self*, *fontsize*, *dpi*)

width(*self*)

get the element width: xmax-xmin

xmax(*self*)

get the xmax of ink rect

xmin(*self*)

get the xmin of ink rect

ymax(*self*)

get the ymax of ink rect

ymin(*self*)

get the ymin of ink rect

25.8.2 Class Variables

Name	Description
dpi	**Value:** 72 *(type=int)*
font	**Value:** 'it' *(type=str)*
fontsize	**Value:** 12 *(type=int)*

25.9 Class ExpressionElement

matplotlib.mathtext.Element
matplotlib.mathtext.GroupElement
ExpressionElement

The entire mathtext expression

25.9.1 Methods

__repr__(*self*)
Overrides: matplotlib.mathtext.GroupElement.__repr__

Inherited from Element: centerx, centery, padx, pady, set_padx, set_pady, set_scale
Inherited from GroupElement: __init__, advance, height, render, set_font, set_origin, set_size_info, width, xmax, xmin, ymax, ymin

25.9.2 Class Variables

Name	Description
Inherited from Element: dpi *(p. 314)*, font *(p. 314)*, fontsize *(p. 314)*	

25.10 Class Fonts

Known Subclasses: BakomaPSFonts, BakomaTrueTypeFonts, DummyFonts, StandardPSFonts, UnicodeFonts

An abstract base class for fonts that want to render mathtext

The class must be able to take symbol keys and font file names and return the character metrics as well as do the drawing

25.10.1 Methods

get_kern(*self*, *facename*, *symleft*, *symright*, *fontsize*, *dpi*)

Get the kerning distance for font between symleft and symright.
facename is one of tt, it, rm, cal or None
sym is a single symbol(alphanum, punct) or a special symbol like \sigma.

get_metrics(*self*, *facename*, *sym*, *fontsize*, *dpi*)

```
facename is one of tt, it, rm, cal or None

sym is a single symbol(alphanum, punct) or a special symbol
like \sigma.

fontsize is in points

Return object has attributes - see
http://www.freetype.org/freetype2/docs/tutorial/step2.html for
a pictoral representation of these attributes

  advance
  height
  width
  xmin, xmax, ymin, ymax  - the ink rectangle of the glyph
```

render(*self*, *ox*, *oy*, *facename*, *sym*, *fontsize*, *dpi*)

set_canvas_size(*self*, *w*, *h*)

Dimension the drawing canvas; may be a noop

25.11 Class GroupElement

matplotlib.mathtext.Element
    GroupElement

Known Subclasses: ExpressionElement

A group is a collection of elements

25.11.1 Methods

__init__(*self*, *elements*)
Overrides: matplotlib.mathtext.Element.__init__

__repr__(*self*)
Overrides: matplotlib.mathtext.Element.__repr__

advance(*self*)

get the horiz advance

Overrides: matplotlib.mathtext.Element.advance

height(*self*)

get the element height: ymax-ymin

Overrides: matplotlib.mathtext.Element.height

render(*self*)

render to the fonts canvas

Overrides: matplotlib.mathtext.Element.render

set_font(*self*, *font*)

set the font (one of tt, it, rm , cal)

Overrides: matplotlib.mathtext.Element.set_font

set_origin(*self*, *ox*, *oy*)
Overrides: matplotlib.mathtext.Element.set_origin

set_size_info(*self*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Element.set_size_info

width(*self*)

get the element width: xmax-xmin

Overrides: matplotlib.mathtext.Element.width

xmax(*self*)

get the max ink in x

Overrides: matplotlib.mathtext.Element.xmax

xmin(*self*)

get the minimum ink in x

Overrides: matplotlib.mathtext.Element.xmin

ymax(*self*)

get the max ink in y

Overrides: matplotlib.mathtext.Element.ymax

ymin(*self*)

get the minimum ink in y

Overrides: matplotlib.mathtext.Element.ymin

Inherited from Element: centerx, centery, padx, pady, set_padx, set_pady, set_scale

25.11.2 Class Variables

Name	Description
Inherited from Element: dpi *(p. 314)*, font *(p. 314)*, fontsize *(p. 314)*	

25.12 Class Handler

25.12.1 Methods

accent(*self*, *s*, *loc*, *toks*)

clear(*self*)

composite(*self*, *s*, *loc*, *toks*)

expression(*self*, *s*, *loc*, *toks*)

font(*self*, *s*, *loc*, *toks*)

group(*self*, *s*, *loc*, *toks*)

is_overunder(*self*, *prev*)

space(*self*, *s*, *loc*, *toks*)

subscript(*self*, *s*, *loc*, *toks*)

subsuperscript(*self*, *s*, *loc*, *toks*)

superscript(*self*, *s*, *loc*, *toks*)

symbol(*self*, *s*, *loc*, *toks*)

25.12.2 Class Variables

Name	Description
symbols	**Value:** `[]` *(type=`list`)*

25.13 Class math_parse_s_ft2font_common

Parse the math expression s, return the (bbox, fonts) tuple needed to render it.

fontsize must be in points

return is width, height, fonts

25.13.1 Methods

__init__(*self*, *output*)

__call__(*self*, *s*, *dpi*, *fontsize*, *angle*=`0`)

25.13.2 Class Variables

Name	Description
major	**Value:** `2` *(type=`int`)*
minor1	**Value:** `5` *(type=`int`)*
minor2	**Value:** `1` *(type=`int`)*
tmp	**Value:** `0` *(type=`int`)*

25.14 Class MyUnicodeFonts

matplotlib.mathtext.Fonts
matplotlib.mathtext.UnicodeFonts
MyUnicodeFonts

25.14.1 Methods

__init__(*self*)
Overrides: matplotlib.mathtext.UnicodeFonts.__init__

Inherited from Fonts: get_kern
Inherited from UnicodeFonts: get_metrics, render, set_canvas_size

25.15 Class SpaceElement

matplotlib.mathtext.Element
└ **SpaceElement**

blank horizontal space

25.15.1 Methods

__init__(*self*, *space*, *height*=0)

space is the amount of blank space in fraction of fontsize height is the height of the space in fraction of fontsize

Overrides: matplotlib.mathtext.Element.__init__

advance(*self*)

get the horiz advance

Overrides: matplotlib.mathtext.Element.advance

height(*self*)

get the element height: ymax-ymin

Overrides: matplotlib.mathtext.Element.height

set_font(*self*, *f*)
Overrides: matplotlib.mathtext.Element.set_font

width(*self*)

get the element width: xmax-xmin

Overrides: matplotlib.mathtext.Element.width

xmax(*self*)

get the max ink in x

Overrides: matplotlib.mathtext.Element.xmax

xmin(*self*)

get the minimum ink in x

Overrides: matplotlib.mathtext.Element.xmin

ymax(*self*)

get the max ink in y

Overrides: matplotlib.mathtext.Element.ymax

ymin(*self*)

get the minimum ink in y

Overrides: matplotlib.mathtext.Element.ymin

Inherited from Element: __repr__, centerx, centery, padx, pady, render, set_origin, set_padx, set_pady, set_scale, set_size_info

25.15.2 Class Variables

Name	Description
Inherited from Element: dpi *(p. 314)*, font *(p. 314)*, fontsize *(p. 314)*	

25.16 Class StandardPSFonts

matplotlib.mathtext.Fonts
└ **StandardPSFonts**

Use the standard postscript fonts for rendering to backend_ps

25.16.1 Methods

__init__(*self*)

get_kern(*self*, *font*, *symleft*, *symright*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.get_kern

get_metrics(*self*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.get_metrics

render(*self*, *ox*, *oy*, *font*, *sym*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.render

set_canvas_size(*self*, *w*, *h*, *pswriter*)

Dimension the drawing canvas; may be a noop

Overrides: matplotlib.mathtext.Fonts.set_canvas_size

25.16.2 Class Variables

Name	Description
basepath	**Value:** `'/home/jdhunter/dev/lib/python2.5/site-packages-/matplotlib/mpl-data/fonts/afm'` *(type=`str`)*

continued on next page

Name	Description
fnames	**Value:** `('psyr', 'pncri8a', 'pcrr8a', 'pncr8a', 'pzcmi8a')` *(type=`tuple`)*
fontmap	**Value:** `{'rm': 'pncr8a', 'tt': 'pcrr8a', 'it': 'pncri8a-', 'cal': 'pzcmi8a'}` *(type=`dict`)*

25.17 Class SymbolElement

matplotlib.mathtext.Element
└ **SymbolElement**

25.17.1 Methods

__init__(*self*, *sym*)
Overrides: matplotlib.mathtext.Element.__init__

__repr__(*self*)
Overrides: matplotlib.mathtext.Element.__repr__

advance(*self*)

get the horiz advance

Overrides: matplotlib.mathtext.Element.advance

height(*self*)

get the element height: ymax-ymin

Overrides: matplotlib.mathtext.Element.height

render(*self*)

render to the fonts canvas

Overrides: matplotlib.mathtext.Element.render

set_font(*self*, *font*)

set the font (one of tt, it, rm , cal)

Overrides: matplotlib.mathtext.Element.set_font

set_origin(*self*, *ox*, *oy*)
Overrides: matplotlib.mathtext.Element.set_origin

set_size_info(*self*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Element.set_size_info

width(*self*)

get the element width: xmax-xmin

Overrides: matplotlib.mathtext.Element.width

xmax(*self*)

get the max ink in x

Overrides: matplotlib.mathtext.Element.xmax

xmin(*self*)

get the minimum ink in x

Overrides: matplotlib.mathtext.Element.xmin

ymax(*self*)

get the max ink in y

Overrides: matplotlib.mathtext.Element.ymax

ymin(*self*)

get the minimum ink in y

Overrides: matplotlib.mathtext.Element.ymin

Inherited from Element: centerx, centery, padx, pady, set_padx, set_pady, set_scale

25.17.2 Class Variables

Name	Description
Inherited from Element: dpi *(p. 314)*, font *(p. 314)*, fontsize *(p. 314)*	

25.18 Class UnicodeFonts

matplotlib.mathtext.Fonts
└ **UnicodeFonts**

Known Subclasses: BakomaUnicodeFonts, CMUUnicodeFonts, MyUnicodeFonts

```
An abstract base class for handling Unicode fonts.

Specific terminology:
 * fontface: an FT2Font object, corresponding to a facename
 * facename: a string that defines the (type)face's name - 'rm', 'it' etc.
```

```
 * filename: a string that is used for generating a fontface object
 * symbol*: a single Unicode character or a TeX command,
    or to be precise, a TeX symbol command like lpha (but not
rac) or
    even a Type1/PS name
 * filenamesd: a dict that maps the face's name to the filename:
    filenamesd = { 'cal' : 'fontnamecal.ext',
                  'rm'  : 'fontnamerm.ext',
                  'tt'  : 'fontnamett.ext',
                  'it'  : 'fontnameit.ext',
                  None  : 'fontnamesmth.ext'}
    filenamesd should be declared as a class atribute
 * glyphdict: a dict used for caching of glyph specific data
 * fonts: a dict of facename -> fontface pairs
 * charmaps: a dict of facename -> charmap pairs. Charmap maps character
   codes to glyph indices
 * glyphmaps: a dict of facename -> glyphmap pairs. A glyphmap is an
    inverted charmap
 * output: a string in ['Agg','SVG','PS'], coresponding to the backends
 * index: Fontfile specific index of a glyph.
```

25.18.1 Methods

__init__(*self*, *output*='Agg')

get_metrics(*self*, *facename*, *symbol*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.get_metrics

render(*self*, *ox*, *oy*, *facename*, *symbol*, *fontsize*, *dpi*)
Overrides: matplotlib.mathtext.Fonts.render

set_canvas_size(*self*, *w*, *h*, *pswriter*=None)

Dimension the drawing canvas; may be a noop

Overrides: matplotlib.mathtext.Fonts.set_canvas_size

Inherited from Fonts: get_kern

26 Module matplotlib.mathtext2

```
Supported commands:
-------------------
 * _, ^, to any depth
 * commands for typesetting functions (\sin, \cos etc.),
 * commands for changing the current font (\rm, \cal etc.),
 * Space/kern commands "\ ", \thinspace
 * \frac

Small TO-DO's:
--------------
 * Display braces etc. \} not working (displaying wierd characters) etc.
 * better placing of sub/superscripts. F_1^1y_{1_{2_{3_{4}^3}}3}1_23
 * implement crampedness (or is it smth. else?). y_1 vs. y_1^1
 * add better italic correction. F^1
 * implement other space/kern commands

TO-DO's:
--------
 * \over, \above, \choose etc.
 * Add support for other backends
```

26.1 Functions

break_up_commands(*texstring*)

Breaks up a string (mustn't contain any groupings) into a list of commands and pure text.

get_args(*command, texgroup, env, num_args*)

Returns the arguments needed by a TeX command

get_first_word(*texstring*)

get_font(*env*)

get_frac_bar_height(*env*)

get_kern(*first, second*)

get_space(*env*)

group_split(*texstring*)

Splits the string into three parts based on the grouping delimiters, and returns them as a list.

handle_char(*uniindex*, *env*)

handle_command(*command*, *texgroup*, *env*, *allowsetters*=`False`)

Handles TeX commands that don't have backward propagation, and aren't setting anything in the environment.

handle_scripts(*firsttype*, *texgroup*, *env*)

handle_tokens(*texgroup*, *env*, *box*=`<class matplotlib.mathtext2.Hbox at 0x8d39c8c>`)

Scans the entire (tex)group to handle tokens. Tokens are other groups, commands, characters, kerns etc. Used recursively.

infer_face(*env*, *item*)

is_command(*item*)

math_parse_s_ft2font(*s*, *dpi*, *fontsize*, *angle*=`0`, *output*=`'AGG'`)

This function is called by the backends

math_parse_s_ft2font1(*s*, *dpi*, *fontsize*, *angle*=`0`)

Used only for testing

math_parse_s_ft2font_svg(*s*, *dpi*, *fontsize*, *angle*=`0`)

normalize_tex(*texstring*)

Normalizes the whole TeX expression (that is: prepares it for parsing)

parse_tex(*texstring*)

remove_comments(*texstring*)

split_command(*texstring*)

Splits a texstring into a command part and a pure text (as a list) part

to_list(*texstring*)

Parses the normalized tex string and returns a list. Used recursively.

26.2 Class Char

matplotlib.mathtext2.Renderer
 Char

A class that implements rendering of a single character.

26.2.1 Methods

__init__(*self*, *env*, *char*, *uniindex*=None)
Overrides: matplotlib.mathtext2.Renderer.__init__

hrender(*self*, *x*, *y*)
Overrides: matplotlib.mathtext2.Renderer.hrender

Inherited from Renderer: vrender

26.3 Class Environment

Class used for representing the TeX environment variables

26.3.1 Methods

__init__(*self*)

copy(*self*)

26.4 Class Fraction

matplotlib.mathtext2.Renderer
 Fraction

A class for rendering a fraction.

26.4.1 Methods

__init__(*self*, *env*, *num*, *den*)
Overrides: matplotlib.mathtext2.Renderer.__init__

hrender(*self*, *x*, *y*)
Overrides: matplotlib.mathtext2.Renderer.hrender

Inherited from Renderer: vrender

26.5 Class Hbox

matplotlib.mathtext2.Renderer
└ **Hbox**

A class that corresponds to a TeX hbox.

26.5.1 Methods

__init__(*self*, *env*, *texlist*=`[]`)
Overrides: matplotlib.mathtext2.Renderer.__init__

hrender(*self*, *x*, *y*)
Overrides: matplotlib.mathtext2.Renderer.hrender

Inherited from Renderer: vrender

26.6 Class Kern

matplotlib.mathtext2.Renderer
└ **Kern**

Class that implements the rendering of a Kern.

26.6.1 Methods

__init__(*self*, *env*, *hadvance*)
Overrides: matplotlib.mathtext2.Renderer.__init__

__repr__(*self*)

Inherited from Renderer: hrender, vrender

26.7 Class Line

matplotlib.mathtext2.Renderer
└ **Line**

Class that implements the rendering of a line.

26.7.1 Methods

__init__(*self*, *env*, *width*, *height*)
Overrides: matplotlib.mathtext2.Renderer.__init__

hrender(*self*, *x*, *y*)
Overrides: matplotlib.mathtext2.Renderer.hrender

Inherited from Renderer: vrender

26.8 Class Renderer

Known Subclasses: Char, Fraction, Hbox, Kern, Line, Scripted, Vbox

Abstract class that implements the rendering methods

26.8.1 Methods

__init__(*self*, *env*)

hrender(*self*, *x*, *y*)

vrender(*self*, *x*, *y*)

26.9 Class Scripted

matplotlib.mathtext2.Renderer
  Scripted

Used for creating elements that have sub/superscripts

26.9.1 Methods

__init__(*self*, *env*, *nuc*=`None`, *type*=`'ord'`, *sub*=`None`, *sup*=`None`)
Overrides: matplotlib.mathtext2.Renderer.__init__

__repr__(*self*)

hrender(*self*, *x*, *y*)
Overrides: matplotlib.mathtext2.Renderer.hrender

Inherited from Renderer: vrender

26.10 Class Scriptfactors

__builtin__.object
  __builtin__.dict
    Scriptfactors

Used for returning the factor with wich you should multiply the fontsize to get the font size of the script

26.10.1 Methods

__getitem__(*self*, *key*)
Overrides: __builtin__.dict.__getitem__

Inherited from dict: __init__, __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __gt__, __hash__, __iter__, __le__, __len__, __lt__, __ne__, __new__, __repr__, __setitem__, clear, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__
Inherited from type: fromkeys

26.11 Class TexParseError

__builtin__.object
exceptions.BaseException
  exceptions.Exception
    TexParseError

26.11.1 Methods

Inherited from object: __hash__, __reduce_ex__
Inherited from BaseException: __delattr__, __getattribute__, __getitem__, __getslice__, __reduce__, __repr__, __setattr__, __setstate__, __str__
Inherited from Exception: __init__, __new__

26.11.2 Class Variables

Name	Description
Inherited from BaseException: args *(p. ??)*, message *(p. ??)*	

26.12 Class Vbox

matplotlib.mathtext2.Renderer

Vbox

A class representing a vertical box. ref is the index of the texlist element whose origin will be used as the vbox origin. The default is ref=-1. If ref is None, then ref is set to be the middle index (if the number of items in the list is even, a new list element is inserted (as Kern(0)) to make the list odd). The box is rendered top down - the last element of the list is rendered at the bottom.

26.12.1 Methods

__init__(*self*, *env*, *texlist*=`[]`, *ref*=`None`)
Overrides: matplotlib.mathtext2.Renderer.__init__

hrender(*self*, *x*, *y*)
Overrides: matplotlib.mathtext2.Renderer.hrender

vrender(*self*, *x*, *y*)
Overrides: matplotlib.mathtext2.Renderer.vrender

27 Module matplotlib.mlab

```
Numerical python functions written for compatability with matlab(TM)
commands with the same names.

  Matlab(TM) compatible functions:

    * cohere - Coherence (normalized cross spectral density)

    * conv     - convolution

    * corrcoef - The matrix of correlation coefficients

    * csd - Cross spectral density uing Welch's average periodogram

    * detrend -- Remove the mean or best fit line from an array

    * find - Return the indices where some condition is true

    * linspace -- Linear spaced array from min to max

    * hist -- Histogram

    * polyfit - least squares best polynomial fit of x to y

    * polyval - evaluate a vector for a vector of polynomial coeffs

    * prctile - find the percentiles of a sequence

    * prepca - Principal Component's Analysis

    * psd - Power spectral density uing Welch's average periodogram

    * rk4 - A 4th order runge kutta integrator for 1D or ND systems

    * vander - the Vandermonde matrix

    * trapz - trapeziodal integration

  Functions that don't exist in matlab(TM), but are useful anyway:

    * cohere_pairs - Coherence over all pairs.  This is not a matlab
      function, but we compute coherence a lot in my lab, and we
      compute it for alot of pairs.  This function is optimized to do
      this efficiently by caching the direct FFTs.

Credits:
```

```
Unless otherwise noted, these functions were written by
Author: John D. Hunter <jdhunter@ace.bsd.uchicago.edu>

Some others are from the Numeric documentation, or imported from
MLab or other Numeric packages
```

27.1 Functions

amap(*fn*, **args*)

amap(function, sequence[, sequence, ...]) -> array.
Works like map(), but it returns an array. This is just a convenient shorthand for Numeric.array(map(...))

approx_real(*x*)

approx_real(x) : returns x.real if |x.imag| < |x.real| * _eps_approx. This function is needed by sqrtm and allows further functions.

base_repr(*number*, *base*=2, *padding*=0)

Return the representation of a number in any given base.

binary_repr(*number*, *max_length*=1025)

Return the binary representation of the input number as a string.
This is more efficient than using base_repr with base 2.
Increase the value of max_length for very large numbers. Note that on 32-bit machines, 2**1023 is the largest integer power of 2 which can be converted to a Python float.

bivariate_normal(*X*, *Y*, *sigmax*=1.0, *sigmay*=1.0, *mux*=0.0, *muy*=0.0, *sigmaxy*=0.0)

Bivariate gaussan distribution for equal shape X, Y
http://mathworld.wolfram.com/BivariateNormalDistribution.html

center_matrix(*M*, *dim*=0)

Return the matrix M with each row having zero mean and unit std
if dim=1, center columns rather than rows

cohere(*x*, *y*, *NFFT*=256, *Fs*=2, *detrend*=`<function detrend_none at 0x8413b1c>`, *window*=`<function window_hanning at 0x8413a04>`, *noverlap*=0)

cohere the coherence between x and y. Coherence is the normalized cross spectral density
Cxy = |Pxy|^2/(Pxx*Pyy)
The return value is (Cxy, f), where f are the frequencies of the coherence vector. See the docs for psd and csd for information about the function arguments NFFT, detrend, window, noverlap, as well as the methods used to compute Pxy, Pxx and Pyy.
Returns the tuple Cxy, freqs

cohere_pairs(*X*, *ij*, *NFFT*=256, *Fs*=2, *detrend*=<function detrend_none at 0x8413b1c>, *window*=<function window_hanning at 0x8413a04>, *noverlap*=0, *preferSpeedOverMemory*=True, *progressCallback*=<function donothing_callback at 0x8413d14>, *returnPxx*=False)

```
Cxy, Phase, freqs = cohere_pairs( X, ij, ...)

Compute the coherence for all pairs in ij.  X is a
numSamples,numCols Numeric array.  ij is a list of tuples (i,j).
Each tuple is a pair of indexes into the columns of X for which
you want to compute coherence.  For example, if X has 64 columns,
and you want to compute all nonredundant pairs, define ij as

  ij = []
  for i in range(64):
      for j in range(i+1,64):
          ij.append( (i,j) )

The other function arguments, except for 'preferSpeedOverMemory'
(see below), are explained in the help string of 'psd'.

Return value is a tuple (Cxy, Phase, freqs).

  Cxy -- a dictionary of (i,j) tuples -> coherence vector for that
    pair.  Ie, Cxy[(i,j) = cohere(X[:,i], X[:,j]).  Number of
    dictionary keys is len(ij)

  Phase -- a dictionary of phases of the cross spectral density at
    each frequency for each pair.  keys are (i,j).

  freqs -- a vector of frequencies, equal in length to either the
    coherence or phase vectors for any i,j key.  Eg, to make a coherence
    Bode plot:

      subplot(211)
      plot( freqs, Cxy[(12,19)])
      subplot(212)
      plot( freqs, Phase[(12,19)])

For a large number of pairs, cohere_pairs can be much more
efficient than just calling cohere for each pair, because it
caches most of the intensive computations.  If N is the number of
pairs, this function is O(N) for most of the heavy lifting,
whereas calling cohere for each pair is O(N^2).  However, because
of the caching, it is also more memory intensive, making 2
additional complex arrays with approximately the same number of
elements as X.

The parameter 'preferSpeedOverMemory', if false, limits the
caching by only making one, rather than two, complex cache arrays.
This is useful if memory becomes critical.  Even when
preferSpeedOverMemory is false, cohere_pairs will still give
significant performace gains over calling cohere for each pair,
and will use subtantially less memory than if
preferSpeedOverMemory is true.  In my tests with a 43000,64 array
over all nonredundant pairs, preferSpeedOverMemory=1 delivered a
33% performace boost on a 1.7GHZ Athlon with 512MB RAM compared
with preferSpeedOverMemory=0.  But both solutions were more than
10x faster than naievly crunching all possible pairs through
```

conv(*x*, *y*, *mode*=2)

convolve x with y

corrcoef(**args*)

corrcoef(X) where X is a matrix returns a matrix of correlation coefficients for each numrows observations and numcols variables.
corrcoef(x,y) where x and y are vectors returns the matrix or correlation coefficients for x and y.
Numeric arrays can be real or complex
The correlation matrix is defined from the covariance matrix C as
r(i,j) = C[i,j] / sqrt(C[i,i]*C[j,j])

csd(*x*, *y*, *NFFT*=256, *Fs*=2, *detrend*=<function detrend_none at 0x8413b1c>, *window*=<function window_hanning at 0x8413a04>, *noverlap*=0)

```
The cross spectral density Pxy by Welches average periodogram
method.  The vectors x and y are divided into NFFT length
segments.  Each segment is detrended by function detrend and
windowed by function window.  noverlap gives the length of the
overlap between segments.  The product of the direct FFTs of x and
y are averaged over each segment to compute Pxy, with a scaling to
correct for power loss due to windowing.  Fs is the sampling
frequency.

NFFT must be a power of 2

window can be a function or a vector of length NFFT. To create
window vectors see numpy.blackman, numpy.hamming, numpy.bartlett,
scipy.signal, scipy.signal.get_window etc.

Returns the tuple Pxy, freqs



Refs:
  Bendat & Piersol -- Random Data: Analysis and Measurement
    Procedures, John Wiley & Sons (1986)
```

detrend(*x*, *key*=None)

detrend_linear(*x*)

Return x minus best fit line; 'linear' detrending

detrend_mean(*x*)

Return x minus the mean(x)

detrend_none(*x*)

Return x: no detrending

diagonal_matrix(*diag*)

Return square diagonal matrix whose non-zero elements are given by the input array.

dist(*x*, *y*)

return the distance between two points

dist_point_to_segment(*p*, *s0*, *s1*)

get the distance of a point to a segment.
p, s0, s1 are xy sequences
This algorithm from
http://softsurfer.com/Archive/algorithm_0102/algorithm_0102.htm#Distance%20to%20Ray%20or%20Segment

donothing_callback(**args*)

entropy(*y*, *bins*)

Return the entropy of the data in y
\sum p_i log2(p_i) where p_i is the probability of observing y in the ith bin of bins. bins can be a number of bins or a range of bins; see hist
Compare S with analytic calculation for a Gaussian x = mu + sigma*randn(200000) Sanalytic = 0.5 * (1.0 + log(2*pi*sigma**2.0))

exp_safe(*x*)

Compute exponentials which safely underflow to zero.
Slow but convenient to use. Note that NumArray will introduce proper floating point exception handling with access to the underlying hardware.

fftsurr(*x*, *detrend*=`<function detrend_none at 0x8413b1c>`, *window*=`<function window_none at 0x8413a3c>`)

Compute an FFT phase randomized surrogate of x

find(*condition*)

Return the indices where condition is true

fix(*x*)

Rounds towards zero. x_rounded = fix(x) rounds the elements of x to the nearest integers towards zero. For negative numbers is equivalent to ceil and for positive to floor.

frange(*xini*, *xfin*=`None`, *delta*=`None`, ***kw*)

frange([start,] stop[, step, keywords]) -> array of floats
Return a Numeric array() containing a progression of floats. Similar to arange(), but defaults to a closed interval.
frange(x0, x1) returns [x0, x0+1, x0+2, ..., x1]; start defaults to 0, and the endpoint *is included*. This behavior is different from that of range() and arange(). This is deliberate, since frange will probably be more useful for generating lists of points for function evaluation, and endpoints are often desired in this use. The usual behavior of range() can be obtained by setting the keyword 'closed=0', in this case frange() basically becomes arange().
When step is given, it specifies the increment (or decrement). All arguments can be floating point numbers.
frange(x0,x1,d) returns [x0,x0+d,x0+2d,...,xfin] where xfin<=x1.
frange can also be called with the keyword 'npts'. This sets the number of points the list should contain (and overrides the value 'step' might have been given). arange() doesn't offer this option.
Examples: >>> frange(3) array([0., 1., 2., 3.]) >>> frange(3,closed=0) array([0., 1., 2.]) >>> frange(1,6,2) array([1, 3, 5]) >>> frange(1,6.5,npts=5) array([1. , 2.375, 3.75 , 5.125, 6.5])

fromfunction_kw(*function*, *dimensions*, ***kwargs*)

Drop-in replacement for fromfunction() from Numerical Python.
Allows passing keyword arguments to the desired function.
Call it as (keywords are optional): fromfunction_kw(MyFunction, dimensions, keywords)
The function MyFunction() is responsible for handling the dictionary of keywords it will recieve.

get_sparse_matrix(*M*, *N*, *frac*=`0.10000000000000001`)

return a MxN sparse matrix with frac elements randomly filled

get_xyz_where(*Z*, *Cond*)

Z and Cond are MxN matrices. Z are data and Cond is a boolean matrix where some condition is satisfied. Return value is x,y,z where x and y are the indices into Z and z are the values of Z at those indices. x,y,z are 1D arrays

hist(*y*, *bins*=`10`, *normed*=`0`)

Return the histogram of y with bins equally sized bins. If bins is an array, use the bins. Return value is (n,x) where n is the count for each bin in x
If normed is False, return the counts in the first element of the return tuple. If normed is True, return the probability density n/(len(y)*dbin)
If y has rank>1, it will be raveled. If y is masked, only the unmasked values will be used. Credits: the Numeric 22 documentation

identity(*n*, *rank*=2, *typecode*='l')

```
identity(n,r) returns the identity matrix of shape (n,n,...,n) (rank r).

For ranks higher than 2, this object is simply a multi-index Kronecker
delta:
                    /  1  if i0=i1=...=iR,
id[i0,i1,...,iR] = -|
                    \  0  otherwise.

Optionally a typecode may be given (it defaults to 'l').

Since rank defaults to 2, this function behaves in the default case (when
only n is given) like the Numeric identity function.
```

inside_poly(*points*, *verts*)

points is a sequence of x,y points verts is a sequence of x,y vertices of a poygon
return value is a sequence on indices into points for the points that are inside the polygon

ispower2(*n*)

Returns the log base 2 of n if n is a power of 2, zero otherwise.
Note the potential ambiguity if n==1: 2**0==1, interpret accordingly.

l1norm(*a*)

Return the l1 norm of a, flattened out.
Implemented as a separate function (not a call to norm() for speed).

l2norm(*a*)

Return the l2 norm of a, flattened out.
Implemented as a separate function (not a call to norm() for speed).

levypdf(*x*, *gamma*, *alpha*)

Returm the levy pdf evaluated at x for params gamma, alpha

liaupunov(*x*, *fprime*)

x is a very long trajectory from a map, and fprime returns the derivative of x. Return lambda = 1/n\sum ln|fprime(x_i)|. See Sec 10.5 Strogatz (1994)"Nonlinear Dynamics and Chaos".

linspace(*xmin*, *xmax*, *N*)

load(*fname*, *comments*='#', *delimiter*=None, *converters*=None, *skiprows*=0, *usecols*=None, *unpack*=False)

```
Load ASCII data from fname into an array and return the array.

The data must be regular, same number of values in every row

fname can be a filename or a file handle.  Support for gzipped files is
automatic, if the filename ends in .gz

matfile data is not currently supported, but see
Nigel Wade's matfile ftp://ion.le.ac.uk/matfile/matfile.tar.gz

Example usage:

  X = load('test.dat')  # data in two columns
  t = X[:,0]
  y = X[:,1]

Alternatively, you can do the same with "unpack"; see below

  X = load('test.dat')    # a matrix of data
  x = load('test.dat')    # a single column of data

comments - the character used to indicate the start of a comment
in the file

delimiter is a string-like character used to seperate values in the
file. If delimiter is unspecified or none, any whitespace string is
a separator.

converters, if not None, is a dictionary mapping column number to
a function that will convert that column to a float.  Eg, if
column 0 is a date string: converters={0:datestr2num}

skiprows is the number of rows from the top to skip

usecols, if not None, is a sequence of integer column indexes to
extract where 0 is the first column, eg usecols=(1,4,5) to extract
just the 2nd, 5th and 6th columns

unpack, if True, will transpose the matrix allowing you to unpack
into named arguments on the left hand side

    t,y = load('test.dat', unpack=True) # for  two column data
    x,y,z = load('somefile.dat', usecols=(3,5,7), unpack=True)

See examples/load_demo.py which exeercises many of these options.
```

log2(*x*, *ln2*=`0.69314718055994529`)

Return the log(x) in base 2.
This is a _slow_ function but which is guaranteed to return the correct integer value if the input is an ineger exact power of 2.

logspace(*xmin*, *xmax*, *N*)

longest_contiguous_ones(*x*)

return the indicies of the longest stretch of contiguous ones in x, assuming x is a vector of zeros and ones.

longest_ones(*x*)

return the indicies of the longest stretch of contiguous ones in x, assuming x is a vector of zeros and ones. If there are two equally long stretches, pick the first

mean(*x*, *dim*=`None`)

mean_flat(*a*)

Return the mean of all the elements of a, flattened out.

meshgrid(*x*, *y*)

```
For vectors x, y with lengths Nx=len(x) and Ny=len(y), return X, Y
where X and Y are (Ny, Nx) shaped arrays with the elements of x
and y repeated to fill the matrix

EG,

  [X, Y] = meshgrid([1,2,3], [4,5,6,7])

   X =
     1   2   3
     1   2   3
     1   2   3
     1   2   3


   Y =
     4   4   4
     5   5   5
     6   6   6
     7   7   7
```

mfuncC(*f*, *x*)

mfuncC(f, x) : matrix function with possibly complex eigenvalues. Note: Numeric defines (v,u) = eig(x) => x*u.T = u.T * Diag(v) This function is needed by sqrtm and allows further functions.

movavg(*x*, *n*)

compute the len(n) moving average of x

norm(*x*, *y*=2)

```
Norm of a matrix or a vector according to Matlab.
The description is taken from Matlab:

    For matrices...
      NORM(X) is the largest singular value of X, max(svd(X)).
      NORM(X,2) is the same as NORM(X).
      NORM(X,1) is the 1-norm of X, the largest column sum,
                      = max(sum(abs((X)))).
      NORM(X,inf) is the infinity norm of X, the largest row sum,
                      = max(sum(abs((X')))).
      NORM(X,'fro') is the Frobenius norm, sqrt(sum(diag(X'*X))).
      NORM(X,P) is available for matrix X only if P is 1, 2, inf or 'fro'.

    For vectors...
      NORM(V,P) = sum(abs(V).^P)^(1/P).
      NORM(V) = norm(V,2).
      NORM(V,inf) = max(abs(V)).
      NORM(V,-inf) = min(abs(V)).
```

normpdf(*x*, **args*)

Return the normal pdf evaluated at x; args provides mu, sigma

orth(*A*)

```
Orthogonalization procedure by Matlab.
The description is taken from its help:

    Q = ORTH(A) is an orthonormal basis for the range of A.
    That is, Q'*Q = I, the columns of Q span the same space as
    the columns of A, and the number of columns of Q is the
    rank of A.
```

polyfit(*x*, *y*, *N*)

```
Do a best fit polynomial of order N of y to x.  Return value is a
vector of polynomial coefficients [pk ... p1 p0].  Eg, for N=2

  p2*x0^2 +  p1*x0 + p0 = y1
  p2*x1^2 +  p1*x1 + p0 = y1
  p2*x2^2 +  p1*x2 + p0 = y2
  .....
  p2*xk^2 +  p1*xk + p0 = yk


Method: if X is a the Vandermonde Matrix computed from x (see
http://mathworld.wolfram.com/VandermondeMatrix.html), then the
polynomial least squares solution is given by the 'p' in

  X*p = y

where X is a len(x) x N+1 matrix, p is a N+1 length vector, and y
is a len(x) x 1 vector

This equation can be solved as

  p = (XT*X)^-1 * XT * y

where XT is the transpose of X and -1 denotes the inverse.

For more info, see
http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html,
but note that the k's and n's in the superscripts and subscripts
on that page.  The linear algebra is correct, however.

See also polyval
```

polyval(*p*, *x*)

```
y = polyval(p,x)

p is a vector of polynomial coeffients and y is the polynomial
evaluated at x.

Example code to remove a polynomial (quadratic) trend from y:

  p = polyfit(x, y, 2)
  trend = polyval(p, x)
  resid = y - trend

See also polyfit
```

prctile(*x*, *p*=`(0.0, 25.0, 50.0, 75.0, 100.0)`)

Return the percentiles of x. p can either be a sequence of percentil values or a scalar. If p is a sequence the i-th element of the return sequence is the p(i)-th percentile of x

prctile_rank(*x*, *p*)

return the for each element in x, return the rank 0..len(p) . Eg if p=(25, 50, 75), the return value will be a len(x) array with values in [0,1,2,3] where 0 indicates the value is less than the 25th percentile, 1 indicates the value is >= the 25th and < 50th percentile, ... and 3 indicates the value is above the 75th percentile cutoff
p is either an array of percentiles in [0..100] or a scalar which indicates how many quantiles of data you want ranked

prepca(*P*, *frac*=`0`)

```
Compute the principal components of P.  P is a numVars x
numObservations numeric array.  frac is the minimum fraction of
variance that a component must contain to be included

Return value are
Pcomponents : a num components x num observations numeric array
Trans       : the weights matrix, ie, Pcomponents = Trans*P
fracVar     : the fraction of the variance accounted for by each
              component returned
```

psd(*x*, *NFFT*=256, *Fs*=2, *detrend*=<function detrend_none at 0x8413b1c>, *window*=<function window_hanning at 0x8413a04>, *noverlap*=0)

```
The power spectral density by Welches average periodogram method.
The vector x is divided into NFFT length segments.  Each segment
is detrended by function detrend and windowed by function window.
noperlap gives the length of the overlap between segments.  The
absolute(fft(segment))**2 of each segment are averaged to compute Pxx,
with a scaling to correct for power loss due to windowing.  Fs is
the sampling frequency.

-- NFFT must be a power of 2
-- detrend is a functions, unlike in matlab where it is a vector.
-- window can be a function or a vector of length NFFT. To create window
   vectors see numpy.blackman, numpy.hamming, numpy.bartlett,
   scipy.signal, scipy.signal.get_window etc.
-- if length x < NFFT, it will be zero padded to NFFT


Returns the tuple Pxx, freqs

Refs:
  Bendat & Piersol -- Random Data: Analysis and Measurement
    Procedures, John Wiley & Sons (1986)
```

rank(*x*)

Returns the rank of a matrix. The rank is understood here as the an estimation of the number of linearly independent rows or columns (depending on the size of the matrix). Note that numerix.mlab.rank() is not equivalent to Matlab's rank. This function is!

rem(*x*, *y*)

Remainder after division. rem(x,y) is equivalent to x - y.*fix(x./y) in case y is not zero. By convention, rem(x,0) returns None. We keep the convention by Matlab: "The input x and y must be real arrays of the same size, or real scalars."

rk4(*derivs, y0, t*)

```
Integrate 1D or ND system of ODEs from initial state y0 at sample
times t.  derivs returns the derivative of the system and has the
signature

 dy = derivs(yi, ti)

Example 1 :

    ## 2D system
    # Numeric solution
    def derivs6(x,t):
        d1 =  x[0] + 2*x[1]
        d2 =  -3*x[0] + 4*x[1]
        return (d1, d2)
    dt = 0.0005
    t = arange(0.0, 2.0, dt)
    y0 = (1,2)
    yout = rk4(derivs6, y0, t)

Example 2:

    ## 1D system
    alpha = 2
    def derivs(x,t):
        return -alpha*x + exp(-t)

    y0 = 1
    yout = rk4(derivs, y0, t)


If you have access to scipy, you should probably be using the
scipy.integrate tools rather than this function
```

rms_flat(*a*)

Return the root mean square of all the elements of a, flattened out.

save(*fname, X, fmt=*`'%.18e'`*, delimiter=*`' '`)

Save the data in X to file fname using fmt string to convert the data to strings
fname can be a filename or a file handle. If the filename ends in .gz, the file is automatically saved in compressed gzip format. The load() command understands gzipped files transparently.
Example usage:
save('test.out', X) # X is an array save('test1.out', (x,y,z)) # x,y,z equal sized 1D arrays save('test2.out', x) # x is 1D save('test3.out', x, fmt='%1.4e') # use exponential notation
delimiter is used to separate the fields, eg delimiter ',' for comma-separated values

segments_intersect(*s1*, *s2*)

Return True if s1 and s2 intersect. s1 and s2 are defines as
s1: (x1, y1), (x2, y2) s2: (x3, y3), (x4, y4)

slopes(*x*, *y*)

SLOPES calculate the slope y'(x) Given data vectors X and Y SLOPES calculates Y'(X), i.e the slope of a curve Y(X). The slope is estimated using the slope obtained from that of a parabola through any three consecutive points.
This method should be superior to that described in the appendix of A CONSISTENTLY WELL BEHAVED METHOD OF INTERPOLATION by Russel W. Stineman (Creative Computing July 1980) in at least one aspect:
Circles for interpolation demand a known aspect ratio between x- and y-values. For many functions, however, the abscissa are given in different dimensions, so an aspect ratio is completely arbitrary.
The parabola method gives very similar results to the circle method for most regular cases but behaves much better in special cases
Norbert Nemec, Institute of Theoretical Physics, University or Regensburg, April 2006 Norbert.Nemec at physik.uni-regensburg.de
(inspired by a original implementation by Halldor Bjornsson, Icelandic Meteorological Office, March 2006 halldor at vedur.is)

specgram(*x*, *NFFT*=`256`, *Fs*=`2`, *detrend*=`<function detrend_none at 0x8413b1c>`, *window*=`<function window_hanning at 0x8413a04>`, *noverlap*=`128`)

Compute a spectrogram of data in x. Data are split into NFFT length segements and the PSD of each section is computed. The windowing function window is applied to each segment, and the amount of overlap of each segment is specified with noverlap.
window can be a function or a vector of length NFFT. To create window vectors see numpy.blackman, numpy.hamming, numpy.bartlett, scipy.signal, scipy.signal.get_window etc.
See pdf for more info.
If x is real (i.e. non-Complex) only the positive spectrum is given. If x is Complex then the complete spectrum is given.
The returned times are the midpoints of the intervals over which the ffts are calculated

sqrtm(*x*)

Returns the square root of a square matrix. This means that s=sqrtm(x) implies s*s = x. Note that s and x are matrices.

stineman_interp(*xi*, *x*, *y*, *yp*=None)

```
STINEMAN_INTERP Well behaved data interpolation.  Given data
vectors X and Y, the slope vector YP and a new abscissa vector XI
the function stineman_interp(xi,x,y,yp) uses Stineman
interpolation to calculate a vector YI corresponding to XI.

Here's an example that generates a coarse sine curve, then
interpolates over a finer abscissa:

  x = linspace(0,2*pi,20);  y = sin(x); yp = cos(x)
  xi = linspace(0,2*pi,40);
  yi = stineman_interp(xi,x,y,yp);
  plot(x,y,'o',xi,yi)

The interpolation method is described in the article A
CONSISTENTLY WELL BEHAVED METHOD OF INTERPOLATION by Russell
W. Stineman. The article appeared in the July 1980 issue of
Creative computing with a note from the editor stating that while
they were

  not an academic journal but once in a while something serious
  and original comes in adding that this was
  "apparently a real solution" to a well known problem.

For yp=None, the routine automatically determines the slopes using
the "slopes" routine.

X is assumed to be sorted in increasing order

For values xi[j] < x[0] or xi[j] > x[-1], the routine tries a
extrapolation.  The relevance of the data obtained from this, of
course, questionable...

original implementation by Halldor Bjornsson, Icelandic
Meteorolocial Office, March 2006 halldor at vedur.is

completely reworked and optimized for Python by Norbert Nemec,
Institute of Theoretical Physics, University or Regensburg, April
2006 Norbert.Nemec at physik.uni-regensburg.de
```

sum_flat(*a*)

Return the sum of all the elements of a, flattened out.
It uses a.flat, and if a is not contiguous, a call to ravel(a) is made.

trapz(*x*, *y*)

vander(*x*, *N*=None)

X = vander(x,N=None)
The Vandermonde matrix of vector x. The i-th column of X is the the i-th power of x. N is the maximum power to compute; if N is None it defaults to len(x).

window_hanning(*x*)

return x times the hanning window of len(x)

window_none(*x*)

No window function; simply return x

zeros_like(*a*)

Return an array of zeros of the shape and typecode of a.

27.2 Class FIFOBuffer

A FIFO queue to hold incoming x, y data in a rotating buffer using numerix arrrays under the hood. It is assumed that you will call asarrays much less frequently than you add data to the queue – otherwise another data structure will be faster

This can be used to support plots where data is added from a real time feed and the plot object wants grab data from the buffer and plot it to screen less freqeuently than the incoming

If you set the dataLim attr to a matplotlib BBox (eg ax.dataLim), the dataLim will be updated as new data come in

TODI: add a grow method that will extend nmax

27.2.1 Methods

__init__(*self*, *nmax*)

buffer up to nmax points

add(*self*, *x*, *y*)

add scalar x and y to the queue

asarrays(*self*)

return x and y as arrays; their length will be the len of data added or nmax

last(*self*)

get the last x, y or None, None if no data set

register(*self*, *func*, *N*)

call func everytime N events are passed; func signature is func(fifo)

update_datalim_to_current(*self*)

update the datalim in the current data in the fifo

28 Module matplotlib.nxutils

29 Module matplotlib.patches

29.1 Functions

bbox_artist(*artist*, *renderer*, *props*=`None`, *fill*=`True`)

This is a debug function to draw a rectangle around the bounding box returned by get_window_extent of an artist, to test whether the artist is returning the correct bbox
props is a dict of rectangle props with the additional property 'pad' that sets the padding around the bbox in points

draw_bbox(*bbox*, *renderer*, *color*=`'k'`, *trans*=`None`)

This is a debug function to draw a rectangle around the bounding box returned by get_window_extent of an artist, to test whether the artist is returning the correct bbox

29.2 Class Arrow

matplotlib.artist.Artist
matplotlib.patches.Patch
matplotlib.patches.Polygon
Arrow

An arrow patch

29.2.1 Methods

__init__(*self*, *x*, *y*, *dx*, *dy*, *width*=1.0, ***kwargs*)

```
Draws an arrow, starting at (x,y), direction and length
given by (dx,dy) the width of the arrow is scaled by width

Valid kwargs are:
          alpha: float
  animated: [True | False]
  antialiased or aa: [True | False]
  clip_box: a matplotlib.transform.Bbox instance
  clip_on: [True | False]
  edgecolor or ec: any matplotlib color
  facecolor or fc: any matplotlib color
  figure: a matplotlib.figure.Figure instance
  fill: [True | False]
  hatch: unknown
  label: any string
  linewidth or lw: float
  lod: [True | False]
  transform: a matplotlib.transform transformation instance
  visible: [True | False]
  zorder: any number
```

Overrides: matplotlib.patches.Polygon.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from
Inherited from Polygon: get_verts

29.2.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.3 Class Circle

matplotlib.artist.Artist
  matplotlib.patches.Patch
    matplotlib.patches.Ellipse
      Circle

A circle patch

29.3.1 Methods

__init__(*self*, *xy*, *radius*=5, ***kwargs*)

```
Create true circle at center xy=(x,y) with given radius;
unlike circle polygon which is a polygonal approcimation, this
uses splines and is much closer to a scale free circle
Valid kwargs are:
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          clip_box: a matplotlib.transform.Bbox instance
          clip_on: [True | False]
          edgecolor or ec: any matplotlib color
          facecolor or fc: any matplotlib color
          figure: a matplotlib.figure.Figure instance
          fill: [True | False]
          hatch: unknown
          label: any string
          linewidth or lw: float
          lod: [True | False]
          transform: a matplotlib.transform transformation instance
          visible: [True | False]
          zorder: any number
```

Overrides: matplotlib.patches.Ellipse.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Ellipse: draw, get_verts
Inherited from Patch: get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from

29.3.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.4 Class CirclePolygon

matplotlib.artist.Artist
  matplotlib.patches.Patch
matplotlib.patches.RegularPolygon
  CirclePolygon

A circle patch

29.4.1 Methods

__init__(*self*, *xy*, *radius*=5, *resolution*=20, ***kwargs*)

```
Create a circle at xy=(x,y) with radius given by 'radius'
Valid kwargs are:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

Overrides: matplotlib.patches.RegularPolygon.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update

Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from
Inherited from RegularPolygon: get_verts

29.4.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.5 Class Ellipse

matplotlib.artist.Artist
  matplotlib.patches.Patch
    Ellipse

Known Subclasses: Circle

A scale-free ellipse

29.5.1 Methods

__init__(*self*, *xy*, *width*, *height*, *angle*=`0.0`, ***kwargs*)

xy - center of ellipse width - length of horizontal axis height - length of vertical axis angle - rotation in degrees (anti-clockwise)
Valid kwargs are: %(Patch)s

Overrides: matplotlib.patches.Patch.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.patches.Patch.draw

get_verts(*self*)

Return the vertices of the patch

Overrides: matplotlib.patches.Patch.get_verts extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch,

get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from

29.5.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.6 Class FancyArrow

matplotlib.artist.Artist
  matplotlib.patches.Patch
    matplotlib.patches.Polygon
      FancyArrow

Like Arrow, but lets you set head width and head height independently.

29.6.1 Methods

__init__(*self*, *x*, *y*, *dx*, *dy*, *width*=`0.001`, *length_includes_head*=`False`, *head_width*=`None`, *head_length*=`None`, *shape*=`'full'`, *overhang*=`0`, *head_starts_at_zero*=`False`, ***kwargs*)

```
Returns a new Arrow.

length_includes_head: True if head is counted in calculating the length.

shape: ['full', 'left', 'right']

overhang: distance that the arrow is swept back (0 overhang means
triangular shape).

head_starts_at_zero: if True, the head starts being drawn at coordinate
0 instead of ending at coordinate 0.

Valid kwargs are:
          alpha: float
  animated: [True | False]
  antialiased or aa: [True | False]
  clip_box: a matplotlib.transform.Bbox instance
  clip_on: [True | False]
  edgecolor or ec: any matplotlib color
  facecolor or fc: any matplotlib color
  figure: a matplotlib.figure.Figure instance
  fill: [True | False]
  hatch: unknown
  label: any string
  linewidth or lw: float
  lod: [True | False]
  transform: a matplotlib.transform transformation instance
  visible: [True | False]
  zorder: any number
```

Overrides: matplotlib.patches.Polygon.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from
Inherited from Polygon: get_verts

29.6.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.7 Class Patch

matplotlib.artist.Artist ─┐
Patch

Known Subclasses: Ellipse, Polygon, Rectangle, RegularPolygon, Shadow

A patch is a 2D thingy with a face color and an edge color

If any of edgecolor, facecolor, linewidth, or antialiased are None, they default to their rc params setting

29.7.1 Methods

__init__(*self*, *edgecolor*=`None`, *facecolor*=`None`, *linewidth*=`None`, *antialiased*=`None`, *hatch*=`None`, *fill*=`1`, ***kwargs*)

The following kwarg properties are supported alpha: float animated: [True | False] antialiased or aa: [True | False] clip_box: a matplotlib.transform.Bbox instance clip_on: [True | False] edgecolor or ec: any matplotlib color facecolor or fc: any matplotlib color figure: a matplotlib.figure.Figure instance fill: [True | False] hatch: unknown label: any string linewidth or lw: float lod: [True | False] transform: a matplotlib.transform transformation instance visible: [True | False] zorder: any number

Overrides: matplotlib.artist.Artist.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_aa(*self*)

alias for get_antialiased

get_antialiased(*self*)

get_ec(*self*)

alias for get_edgecolor

get_edgecolor(*self*)

get_facecolor(*self*)

get_fc(*self*)

alias for get_facecolor

get_fill(*self*)

return whether fill is set

get_hatch(*self*)

return the current hatching pattern

get_linewidth(*self*)

get_lw(*self*)

alias for get_linewidth

get_verts(*self*)

Return the vertices of the patch

get_window_extent(*self*, *renderer*=`None`)

pick(*self*, *mouseevent*)

if the mouse click is inside the vertices defining the patch, fire off a backend_bases.PickEvent

Overrides: matplotlib.artist.Artist.pick

set_antialiased(*self*, *aa*)

Set whether to use antialiased rendering
ACCEPTS: [True | False]

set_ec(*self*, *val*)

alias for set_edgecolor

set_edgecolor(*self*, *color*)

Set the patch edge color
ACCEPTS: any matplotlib color

set_facecolor(*self*, *color*)

Set the patch face color
ACCEPTS: any matplotlib color

set_fc(*self*, *val*)

alias for set_facecolor

set_fill(*self*, *b*)

Set whether to fill the patch
ACCEPTS: [True | False]

set_hatch(*self*, *h*)

```
Set the hatching pattern

hatch can be one of:
/   - diagonal hatching
\   - back diagonal
|   - vertical
-   - horizontal
#   - crossed
x   - crossed diagonal
letters can be combined, in which case all the specified
hatchings are done
if same letter repeats, it increases the density of hatching
in that direction

CURRENT LIMITATIONS:
1. Hatching is supported in the PostScript
backend only.

2. Hatching is done with solid black lines of width 0.
```

set_linewidth(*self*, *w*)

Set the patch linewidth in points
ACCEPTS: float

set_lw(*self*, *val*)

alias for set_linewidth

update_from(*self*, *other*)

copy properties from other to self

Overrides: matplotlib.artist.Artist.update_from extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update

29.7.2 Class Variables

Name	Description
zorder	**Value:** `1` *(type=int)*
Inherited from Artist: aname *(p. 92)*	

29.8 Class Polygon

```
matplotlib.artist.Artist
    matplotlib.patches.Patch
        Polygon
```

Known Subclasses: Arrow, FancyArrow, Wedge, YAArrow

A general polygon patch.

29.8.1 Methods

__init__(*self*, *xy*, ***kwargs*)

```
xy is a sequence of (x,y) 2 tuples
Valid kwargs are:
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          clip_box: a matplotlib.transform.Bbox instance
          clip_on: [True | False]
          edgecolor or ec: any matplotlib color
          facecolor or fc: any matplotlib color
          figure: a matplotlib.figure.Figure instance
          fill: [True | False]
          hatch: unknown
          label: any string
          linewidth or lw: float
          lod: [True | False]
          transform: a matplotlib.transform transformation instance
          visible: [True | False]
          zorder: any number

See Patch documentation for additional kwargs
```

Overrides: matplotlib.patches.Patch.__init__

get_verts(*self*)

Return the vertices of the patch

Overrides: matplotlib.patches.Patch.get_verts extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from

29.8.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.9 Class PolygonInteractor

```
An polygon editor.

Key-bindings

  't' toggle vertex markers on and off.  When vertex markers are on,
      you can move them, delete them

  'd' delete the vertex under point

  'i' insert a vertex at point.  You must be within epsilon of the
      line connecting two existing vertices
```

29.9.1 Methods

__init__(*self*, *poly*)

button_press_callback(*self*, *event*)

whenever a mouse button is pressed

button_release_callback(*self*, *event*)

whenever a mouse button is released

get_ind_under_point(*self*, *event*)

get the index of the vertex under point if within epsilon tolerance

key_press_callback(*self*, *event*)

whenever a key is pressed

motion_notify_callback(*self*, *event*)

on mouse movement

poly_changed(*self*, *poly*)

this method is called whenever the polygon object is called

29.9.2 Class Variables

Name	Description
epsilon	**Value:** `5` *(type=`int`)*
showverts	**Value:** `True` *(type=`bool`)*

29.10 Class Rectangle

matplotlib.artist.Artist
  matplotlib.patches.Patch
    Rectangle

Known Subclasses: Cell

Draw a rectangle with lower left at xy=(x,y) with specified width and height

29.10.1 Methods

__init__(*self*, *xy*, *width*, *height*, ***kwargs*)

```
xy is an x,y tuple lower, left
width and height are width and height of rectangle
fill is a boolean indicating whether to fill the rectangle
Valid kwargs are:
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          clip_box: a matplotlib.transform.Bbox instance
          clip_on: [True | False]
          edgecolor or ec: any matplotlib color
          facecolor or fc: any matplotlib color
          figure: a matplotlib.figure.Figure instance
          fill: [True | False]
          hatch: unknown
          label: any string
          linewidth or lw: float
          lod: [True | False]
          transform: a matplotlib.transform transformation instance
          visible: [True | False]
          zorder: any number
```

Overrides: matplotlib.patches.Patch.__init__

get_height(*self*)

Return the height of the rectangle

get_verts(*self*)

Return the vertices of the rectangle

Overrides: matplotlib.patches.Patch.get_verts

get_width(*self*)

Return the width of the rectangle

get_x(*self*)

Return the left coord of the rectangle

get_y(*self*)

Return the bottom coord of the rectangle

set_bounds(*self*, **args*)

Set the bounds of the rectangle: l,b,w,h
ACCEPTS: (left, bottom, width, height)

set_height(*self*, *h*)

Set the width rectangle
ACCEPTS: float

set_width(*self*, *w*)

Set the width rectangle
ACCEPTS: float

set_x(*self*, *x*)

Set the left coord of the rectangle
ACCEPTS: float

set_y(*self*, *y*)

Set the bottom coord of the rectangle
ACCEPTS: float

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from

29.10.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.11 Class RegularPolygon

matplotlib.artist.Artist
  matplotlib.patches.Patch
    RegularPolygon

Known Subclasses: CirclePolygon

A regular polygon patch.

29.11.1 Methods

__init__(*self*, *xy*, *numVertices*, *radius*=5, *orientation*=0, ***kwargs*)

```
xy is a length 2 tuple (the center)
numVertices is the number of vertices.
radius is the distance from the center to each of the vertices.
orientation is in radians and rotates the polygon.
Valid kwargs are:
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          clip_box: a matplotlib.transform.Bbox instance
          clip_on: [True | False]
          edgecolor or ec: any matplotlib color
          facecolor or fc: any matplotlib color
          figure: a matplotlib.figure.Figure instance
          fill: [True | False]
          hatch: unknown
          label: any string
          linewidth or lw: float
          lod: [True | False]
          transform: a matplotlib.transform transformation instance
          visible: [True | False]
          zorder: any number
```

Overrides: matplotlib.patches.Patch.__init__

get_verts(*self*)

Return the vertices of the patch

Overrides: matplotlib.patches.Patch.get_verts extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from

29.11.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.12 Class Shadow

matplotlib.artist.Artist
 └ matplotlib.patches.Patch
  └ **Shadow**

29.12.1 Methods

__init__(*self*, *patch*, *ox*, *oy*, *props*=None, ***kwargs*)

```
Create a shadow of the patch offset by ox, oy.  props, if not None is
a patch property update dictionary.  If None, the shadow will have
have the same color as the face, but darkened
kwargs are
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          clip_box: a matplotlib.transform.Bbox instance
          clip_on: [True | False]
          edgecolor or ec: any matplotlib color
          facecolor or fc: any matplotlib color
          figure: a matplotlib.figure.Figure instance
          fill: [True | False]
          hatch: unknown
          label: any string
          linewidth or lw: float
          lod: [True | False]
          transform: a matplotlib.transform transformation instance
          visible: [True | False]
          zorder: any number
```

Overrides: matplotlib.patches.Patch.__init__

get_verts(*self*)

Return the vertices of the patch

Overrides: matplotlib.patches.Patch.get_verts extit(inherited documentation)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible,

set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from

29.12.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.13 Class Wedge

matplotlib.artist.Artist
  matplotlib.patches.Patch
    matplotlib.patches.Polygon
      Wedge

29.13.1 Methods

__init__(*self*, *center*, *r*, *theta1*, *theta2*, *dtheta*=0.10000000000000001, ***kwargs*)

```
Draw a wedge centered at x,y tuple center with radius r that
sweeps theta1 to theta2 (angles)
dtheta is the resolution in degrees
Valid kwargs are:
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          clip_box: a matplotlib.transform.Bbox instance
          clip_on: [True | False]
          edgecolor or ec: any matplotlib color
          facecolor or fc: any matplotlib color
          figure: a matplotlib.figure.Figure instance
          fill: [True | False]
          hatch: unknown
          label: any string
          linewidth or lw: float
          lod: [True | False]
          transform: a matplotlib.transform transformation instance
          visible: [True | False]
          zorder: any number
```

Overrides: matplotlib.patches.Polygon.__init__

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from
Inherited from Polygon: get_verts

29.13.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

29.14 Class YAArrow

matplotlib.artist.Artist
  matplotlib.patches.Patch
    matplotlib.patches.Polygon
      YAArrow

Yet another arrow class

This is an arrow that is defined in display space and has a tip at x1,y1 and a base at x2, y2.

29.14.1 Methods

__init__(*self*, *dpi*, *xytip*, *xybase*, *width*=4, *frac*=0.10000000000000001, *headwidth*=12, ***kwargs*)

```
xytip : (x,y) location of arrow tip
xybase : (x,y) location the arrow base mid point
dpi : the figure dpi instance (fig.dpi)
width : the width of the arrow in points
frac  : the fraction of the arrow length occupied by the head
headwidth : the width of the base of the arrow head in points
Valid kwargs are:
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          clip_box: a matplotlib.transform.Bbox instance
          clip_on: [True | False]
          edgecolor or ec: any matplotlib color
          facecolor or fc: any matplotlib color
          figure: a matplotlib.figure.Figure instance
          fill: [True | False]
          hatch: unknown
          label: any string
          linewidth or lw: float
          lod: [True | False]
          transform: a matplotlib.transform transformation instance
          visible: [True | False]
          zorder: any number
```

Overrides: matplotlib.patches.Polygon.__init__

get_verts(*self*)

Return the vertices of the patch

Overrides: matplotlib.patches.Polygon.get_verts extit(inherited documentation)

getpoints(*self*, *x1*, *y1*, *x2*, *y2*, *k*)

for line segment defined by x1,y1 and x2,y2, return the points on the line that is perpendicular to the line and intersects x2,y2 and the distance from x2,y2 ot the returned points is k

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Patch: draw, get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch, get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from

29.14.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

30 Module matplotlib.proj3d

Various transforms used for by the 3D code

30.1 Functions

inv_transform(*xs*, *ys*, *zs*, *M*)

line2d(*p0*, *p1*)

Return 2D equation of line in the form ax+by+c = 0

line2d_dist(*l*, *p*)

Distance from line to point line is a tuple of coefficients a,b,c

line2d_seg_dist(*p1*, *p2*, *p0*)

distance(s) from line defined by p1 - p2 to point(s) p0
p0[0] = x(s) p0[1] = y(s)
intersection point p = p1 + u*(p2-p1) and intersection point lies within segement if u is between 0 and 1

mod(*v*)

3d vector length

persp_transformation(*zfront*, *zback*)

proj_points(*points*, *M*)

proj_trans_clip_points(*points*, *M*)

proj_trans_points(*points*, *M*)

proj_transform(*xs*, *ys*, *zs*, *M*)

Transform the points by the projection matrix

proj_transform_clip(*xs*, *ys*, *zs*, *M*)

Transform the points by the projection matrix and return the clipping result returns txs,tys,tzs,tis

proj_transform_vec(*vec*, *M*)

proj_transform_vec_clip(*vec*, *M*)

rot_x(*V*, *alpha*)

test_lines_dists()

test_proj()

test_proj_draw_axes(*M*, *s*=`1`)

test_proj_make_M(*E*=`None`)

test_rot()

test_world()

transform(*xs*, *ys*, *zs*, *M*)

Transform the points by the projection matrix

vec_pad_ones(*xs*, *ys*, *zs*)

view_transformation(*E*, *R*, *V*)

world_transformation(*xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax*)

31 Module matplotlib.pylab

```
This is a matlab(TM) style interface to matplotlib.

The following plotting commands are provided; some of these do not
exist in matlab(TM) but have proven themselves to be useful nonetheless.
The majority of them, however, have matlab analogs

_Plotting commands
  acorr     - plot the autocorrelation function
  annotate  - annotate something in the figure
  arrow     - add an arrow to the axes
  axes      - Create a new axes
  axhline   - draw a horizontal line across axes
  axvline   - draw a vertical line across axes
  axhspan   - draw a horizontal bar across axes
  axvspan   - draw a vertical bar across axes
  axis      - Set or return the current axis limits
  bar       - make a bar chart
  barh      - a horizontal bar chart
  broken_barh - a set of horizontal bars with gaps
  box       - set the axes frame on/off state
  boxplot   - make a box and whisker plot
  cla       - clear current axes
  clabel    - label a contour plot
  clf       - clear a figure window
  clim      - adjust the color limits of the current image
  close     - close a figure window
  colorbar  - add a colorbar to the current figure
  cohere    - make a plot of coherence
  contour   - make a contour plot
  contourf  - make a filled contour plot
  csd       - make a plot of cross spectral density
  delaxes   - delete an axes from the current figure
  draw      - Force a redraw of the current figure
  errorbar  - make an errorbar graph
  figlegend - make legend on the figure rather than the axes
  figimage  - make a figure image
  figtext   - add text in figure coords
  figure   - create or change active figure
  fill     - make filled polygons
  gca      - return the current axes
  gcf      - return the current figure
  gci      - get the current image, or None
  getp      - get a handle graphics property
  grid     - set whether gridding is on
  hist     - make a histogram
  hold     - set the axes hold state
```

```
ioff     - turn interaction mode off
ion      - turn interaction mode on
isinteractive - return True if interaction mode is on
imread   - load image file into array
imshow   - plot image data
ishold   - return the hold state of the current axes
legend   - make an axes legend
loglog   - a log log plot
matshow  - display a matrix in a new figure preserving aspect
pcolor   - make a pseudocolor plot
pcolormesh - make a pseudocolor plot using a quadrilateral mesh
pie      - make a pie chart
plot     - make a line plot
plot_date - plot dates
pie      - pie charts
polar    - make a polar plot on a PolarAxes
psd      - make a plot of power spectral density
quiver   - make a direction field (arrows) plot
rc       - control the default params
rgrids   - customize the radial grids and labels for polar
savefig  - save the current figure
scatter  - make a scatter plot
setp      - set a handle graphics property
semilogx - log x axis
semilogy - log y axis
show     - show the figures
specgram - a spectrogram plot
spy      - plot sparsity pattern using markers or image
stem     - make a stem plot
subplot  - make a subplot (numrows, numcols, axesnum)
subplots_adjust - change the params controlling the subplot positions of current figure
subplot_tool - launch the subplot configuration tool
table    - add a table to the plot
text     - add some text at location x,y to the current axes
thetagrids - customize the radial theta grids and labels for polar
title    - add a title to the current axes
xcorr   - plot the autocorrelation function of x and y
xlim     - set/get the xlimits
ylim     - set/get the ylimits
xticks   - set/get the xticks
yticks   - set/get the yticks
xlabel   - add an xlabel to the current axes
ylabel   - add a ylabel to the current axes

autumn - set the default colormap to autumn
bone   - set the default colormap to bone
cool   - set the default colormap to cool
copper - set the default colormap to copper
```

```
  flag   - set the default colormap to flag
  gray   - set the default colormap to gray
  hot    - set the default colormap to hot
  hsv    - set the default colormap to hsv
  jet    - set the default colormap to jet
  pink   - set the default colormap to pink
  prism  - set the default colormap to prism
  spring - set the default colormap to spring
  summer - set the default colormap to summer
  winter - set the default colormap to winter
  spectral - set the default colormap to spectral

_Event handling

  connect - register an event handler
  disconnect - remove a connected event handler

_Matrix commands

  cumprod   - the cumulative product along a dimension
  cumsum    - the cumulative sum along a dimension
  detrend   - remove the mean or besdt fit line from an array
  diag      - the k-th diagonal of matrix
  diff      - the n-th differnce of an array
  eig       - the eigenvalues and eigen vectors of v
  eye       - a matrix where the k-th diagonal is ones, else zero
  find      - return the indices where a condition is nonzero
  fliplr    - flip the rows of a matrix up/down
  flipud    - flip the columns of a matrix left/right
  linspace  - a linear spaced vector of N values from min to max inclusive
  meshgrid  - repeat x and y to make regular matrices
  ones      - an array of ones
  rand      - an array from the uniform distribution [0,1]
  randn     - an array from the normal distribution
  rot90     - rotate matrix k*90 degress counterclockwise
  squeeze   - squeeze an array removing any dimensions of length 1
  tri       - a triangular matrix
  tril      - a lower triangular matrix
  triu      - an upper triangular matrix
  vander    - the Vandermonde matrix of vector x
  svd       - singular value decomposition
  zeros     - a matrix of zeros

_Probability

  levypdf   - The levy probability density function from the char. func.
  normpdf   - The Gaussian probability density function
  rand      - random numbers from the uniform distribution
```

```
  randn     - random numbers from the normal distribution

_Statistics

  corrcoef  - correlation coefficient
  cov       - covariance matrix
  amax       - the maximum along dimension m
  mean      - the mean along dimension m
  median    - the median along dimension m
  amin       - the minimum along dimension m
  norm      - the norm of vector x
  prod      - the product along dimension m
  ptp       - the max-min along dimension m
  std       - the standard deviation along dimension m
  asum       - the sum along dimension m

_Time series analysis

  bartlett  - M-point Bartlett window
  blackman  - M-point Blackman window
  cohere    - the coherence using average periodiogram
  csd       - the cross spectral density using average periodiogram
  fft       - the fast Fourier transform of vector x
  hamming   - M-point Hamming window
  hanning   - M-point Hanning window
  hist      - compute the histogram of x
  kaiser    - M length Kaiser window
  psd       - the power spectral density using average periodiogram
  sinc      - the sinc function of array x

_Dates

  date2num  - convert python datetimes to numeric representation
  drange    - create an array of numbers for date plots
  num2date  - convert numeric type (float days since 0001) to datetime

_Other

  angle     - the angle of a complex array
  load     - load ASCII data into array
  polyfit   - fit x, y to an n-th order polynomial
  polyval   - evaluate an n-th order polynomial
  roots     - the roots of the polynomial coefficients in p
  save      - save an array to an ASCII file
  trapz     - trapezoidal integration

__end
```

```
Credits: The plotting commands were provided by
John D. Hunter <jdhunter@ace.bsd.uchicago.edu>

Most of the other commands are from Numeric, MLab and FFT, with the
exception of those in mlab.py provided by matplotlib.
```

31.1 Functions

acorr(**args, **kwargs*)

```
ACORR(x, normed=False, detrend=detrend_none, usevlines=False,
      maxlags=None, **kwargs)
Plot the autocorrelation of x.  If normed=True, normalize the
data but the autocorrelation at 0-th lag.  x is detrended by
the detrend callable (default no normalization.
data are plotted as plot(lags, c, **kwargs)
return value is lags, c, line where lags are a length
2*maxlags+1 lag vector, c is the 2*maxlags+1 auto correlation
vector, and line is a Line2D instance returned by plot.  The
default linestyle is None and the default marker is 'o',
though these can be overridden with keyword args.  The cross
correlation is performed with numerix cross_correlate with
mode=2.
If usevlines is True, Axes.vlines rather than Axes.plot is used
to draw vertical lines from the origin to the acorr.
Otherwise the plotstyle is determined by the kwargs, which are
Line2D properties.  If usevlines, the return value is lags, c,
linecol, b where linecol is the LineCollection and b is the x-axis
if usevlines=True, kwargs are passed onto Axes.vlines
if usevlines=False, kwargs are passed onto Axes.plot
maxlags is a positive integer detailing the number of lags to show.
The default value of None will return all (2*len(x)-1) lags.
See the respective function for documentation on valid kwargs

Addition kwargs: hold = [True|False] overrides default hold state
```

annotate(**args*, ***kwargs*)

```
annotate(self, s, xy, textloc,
 xycoords='data', textcoords='data',
 lineprops=None,
 markerprops=None
 **props)
alpha: float
animated: [True | False]
axes: an axes instance
backgroundcolor: any matplotlib color
bbox: rectangle prop dict plus key 'pad' which is a pad in points
clip_box: a matplotlib.transform.Bbox instance
clip_on: [True | False]
clip_path: an agg.path_storage instance
color: any matplotlib color
family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
figure: a matplotlib.figure.Figure instance
fontproperties: a matplotlib.font_manager.FontProperties instance
horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
label: any string
lod: [True | False]
multialignment: ['left' | 'right' | 'center' ]
name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
picker: [None|float|boolean|callable]
position: (x,y)
rotation: [ angle in degrees 'vertical' | 'horizontal'
size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
style or fontstyle: [ 'normal' | 'italic' | 'oblique']
text: string or anything printable with '%s' conversion
transform: a matplotlib.transform transformation instance
variant: [ 'normal' | 'small-caps' ]
verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
visible: [True | False]
weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
x: float
y: float
zorder: any number
```

arrow(**args*, ***kwargs*)

```
Draws arrow on specified axis from (x,y) to (x+dx,y+dy).
Optional kwargs control the arrow properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

autumn()

set the default colormap to autumn and apply to current image if any. See help(colormaps) for more information

axes(**args*, ***kwargs*)

```
Add an axes at positon rect specified by::

axes() by itself creates a default full subplot(111) window axis

axes(rect, axisbg='w') where rect=[left, bottom, width, height] in
normalized (0,1) units.  axisbg is the background color for the
axis, default white

axes(h) where h is an axes instance makes h the
current axis An Axes instance is returned

kwargs:

  axisbg=color   : the axes background color
  frameon=False  : don't display the frame
  sharex=otherax : the current axes shares xaxis attribute with otherax
  sharey=otherax : the current axes shares yaxis attribute with otherax
  polar=True|False : use a polar axes or not

Examples

  examples/axes_demo.py places custom axes.
  examples/shared_axis_demo.py uses sharex and sharey
```

axhline(**args, **kwargs*)

```
AXHLINE(y=0, xmin=0, xmax=1, **kwargs)
Axis Horizontal Line
Draw a horizontal line at y from xmin to xmax.  With the default
values of xmin=0 and xmax=1, this line will always span the horizontal
extent of the axes, regardless of the xlim settings, even if you
change them, eg with the xlim command.  That is, the horizontal extent
is in axes coords: 0=left, 0.5=middle, 1.0=right but the y location is
in data coordinates.
Return value is the Line2D instance.  kwargs are the same as kwargs to
plot, and can be used to control the line properties.  Eg
  # draw a thick red hline at y=0 that spans the xrange
  axhline(linewidth=4, color='r')
  # draw a default hline at y=1 that spans the xrange
  axhline(y=1)
  # draw a default hline at y=.5 that spans the the middle half of
  # the xrange
  axhline(y=.5, xmin=0.25, xmax=0.75)
Valid kwargs are Line2D properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

axhspan(**args*, ***kwargs*)

```
AXHSPAN(ymin, ymax, xmin=0, xmax=1, **kwargs)
Axis Horizontal Span.  ycoords are in data units and x
coords are in axes (relative 0-1) units
Draw a horizontal span (regtangle) from ymin to ymax.  With the
default values of xmin=0 and xmax=1, this always span the xrange,
regardless of the xlim settings, even if you change them, eg with the
xlim command.  That is, the horizontal extent is in axes coords:
0=left, 0.5=middle, 1.0=right but the y location is in data
coordinates.
kwargs are the kwargs to Patch, eg
  antialiased, aa
  linewidth,   lw
  edgecolor,   ec
  facecolor,   fc
the terms on the right are aliases
Return value is the patches.Polygon instance.
    #draws a gray rectangle from y=0.25-0.75 that spans the horizontal
    #extent of the axes
    axhspan(0.25, 0.75, facecolor='0.5', alpha=0.5)
Valid kwargs are Polygon properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

axis(**v, **kwargs*)

Set/Get the axis properties:

```
  v = axis()  returns the current axes as v = [xmin, xmax, ymin, ymax]

  axis(v) where v = [xmin, xmax, ymin, ymax] sets the min and max
    of the x and y axes

  axis('off') turns off the axis lines and labels

  axis('equal') changes limits of x or y axis so that equal
    increments of x and y have the same length; a circle
    is circular.

  axis('scaled') achieves the same result by changing the
    dimensions of the plot box instead of the axis data
    limits.

  axis('tight') changes x and y axis limits such that all data is
    shown. If all data is already shown, it will move it to the center
    of the figure without modifying (xmax-xmin) or (ymax-ymin). Note
    this is slightly different than in matlab.

  axis('image') is 'scaled' with the axis limits equal to the
    data limits.

  axis('auto') or 'normal' (deprecated) restores default behavior;
    axis limits are automatically scaled to make the data fit
    comfortably within the plot box.

 if len(*v)==0, you can pass in xmin, xmax, ymin, ymax as kwargs
 selectively to alter just those limits w/o changing the others.
 See help(xlim) and help(ylim) for more information

 The xmin, xmax, ymin, ymax tuple is returned
```

axvline(**args, **kwargs*)

```
AXVLINE(x=0, ymin=0, ymax=1, **kwargs)
Axis Vertical Line
Draw a vertical line at x from ymin to ymax.  With the default values
of ymin=0 and ymax=1, this line will always span the vertical extent
of the axes, regardless of the xlim settings, even if you change them,
eg with the xlim command.  That is, the vertical extent is in axes
coords: 0=bottom, 0.5=middle, 1.0=top but the x location is in data
coordinates.
Return value is the Line2D instance.  kwargs are the same as
kwargs to plot, and can be used to control the line properties.  Eg
    # draw a thick red vline at x=0 that spans the yrange
    l = axvline(linewidth=4, color='r')
    # draw a default vline at x=1 that spans the yrange
    l = axvline(x=1)
    # draw a default vline at x=.5 that spans the the middle half of
    # the yrange
    axvline(x=.5, ymin=0.25, ymax=0.75)
Valid kwargs are Line2D properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

axvspan(**args*, ***kwargs*)

```
AXVSPAN(xmin, xmax, ymin=0, ymax=1, **kwargs)
axvspan : Axis Vertical Span.  xcoords are in data units and y coords
are in axes (relative 0-1) units
Draw a vertical span (regtangle) from xmin to xmax.  With the default
values of ymin=0 and ymax=1, this always span the yrange, regardless
of the ylim settings, even if you change them, eg with the ylim
command.  That is, the vertical extent is in axes coords: 0=bottom,
0.5=middle, 1.0=top but the y location is in data coordinates.
kwargs are the kwargs to Patch, eg
  antialiased, aa
  linewidth,   lw
  edgecolor,   ec
  facecolor,   fc
the terms on the right are aliases
return value is the patches.Polygon instance.
    # draw a vertical green translucent rectangle from x=1.25 to 1.55 that
    # spans the yrange of the axes
    axvspan(1.25, 1.55, facecolor='g', alpha=0.5)
Valid kwargs are Polygon properties
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

bar(**args*, ***kwargs*)

```
BAR(left, height, width=0.8, bottom=0,
    color=None, edgecolor=None, linewidth=None,
    yerr=None, xerr=None, ecolor=None, capsize=3,
    align='edge', orientation='vertical', log=False)
Make a bar plot with rectangles bounded by
  left, left+width, bottom, bottom+height
        (left, right, bottom and top edges)
left, height, width, and bottom can be either scalars or sequences
Return value is a list of Rectangle patch instances
    left - the x coordinates of the left sides of the bars
    height - the heights of the bars
Optional arguments:
    width - the widths of the bars
    bottom - the y coordinates of the bottom edges of the bars
    color - the colors of the bars
    edgecolor - the colors of the bar edges
    linewidth - width of bar edges; None means use default
        linewidth; 0 means don't draw edges.
    xerr and yerr, if not None, will be used to generate errorbars
    on the bar chart
    ecolor specifies the color of any errorbar
    capsize (default 3) determines the length in points of the error
    bar caps
    align = 'edge' (default) | 'center'
    orientation = 'vertical' | 'horizontal'
    log = False | True - False (default) leaves the orientation
            axis as-is; True sets it to log scale
For vertical bars, align='edge' aligns bars by their left edges in
left, while 'center' interprets these values as the x coordinates of
the bar centers. For horizontal bars, 'edge' aligns bars by their
bottom edges in bottom, while 'center' interprets these values as the
y coordinates of the bar centers.
The optional arguments color, edgecolor, linewidth, xerr, and yerr can
be either scalars or sequences of length equal to the number of bars.
This enables you to use bar as the basis for stacked bar charts, or
candlestick plots.
Optional kwargs:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

barh(**args*, ***kwargs*)

```
BARH(bottom, width, height=0.8, left=0, **kwargs)
Make a horizontal bar plot with rectangles bounded by
  left, left+width, bottom, bottom+height
        (left, right, bottom and top edges)
bottom, width, height, and left can be either scalars or sequences
Return value is a list of Rectangle patch instances
    bottom - the vertical positions of the bottom edges of the bars
    width - the lengths of the bars
Optional arguments:
    height - the heights (thicknesses) of the bars
    left - the x coordinates of the left edges of the bars
    color - the colors of the bars
    edgecolor - the colors of the bar edges
    linewidth - width of bar edges; None means use default
        linewidth; 0 means don't draw edges.
    xerr and yerr, if not None, will be used to generate errorbars
    on the bar chart
    ecolor specifies the color of any errorbar
    capsize (default 3) determines the length in points of the error
    bar caps
    align = 'edge' (default) | 'center'
    log = False | True - False (default) leaves the horizontal
            axis as-is; True sets it to log scale
Setting align='edge' aligns bars by their bottom edges in bottom,
while 'center' interprets these values as the y coordinates of the bar
centers.
The optional arguments color, edgecolor, linewidth, xerr, and yerr can
be either scalars or sequences of length equal to the number of bars.
This enables you to use barh as the basis for stacked bar charts, or
candlestick plots.
Optional kwargs:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

bone()

set the default colormap to bone and apply to current image if any. See help(colormaps) for more information

box(*on*=`None`)

Turn the axes box on or off according to 'on'
If on is None, toggle state

boxplot(**args*, ***kwargs*)

```
boxplot(x, notch=0, sym='+', vert=1, whis=1.5,
        positions=None, widths=None)
Make a box and whisker plot for each column of x or
each vector in sequence x.
The box extends from the lower to upper quartile values
of the data, with a line at the median.  The whiskers
extend from the box to show the range of the data.  Flier
points are those past the end of the whiskers.
notch = 0 (default) produces a rectangular box plot.
notch = 1 will produce a notched box plot
sym (default 'b+') is the default symbol for flier points.
Enter an empty string ('') if you don't want to show fliers.
vert = 1 (default) makes the boxes vertical.
vert = 0 makes horizontal boxes.  This seems goofy, but
that's how Matlab did it.
whis (default 1.5) defines the length of the whiskers as
a function of the inner quartile range.  They extend to the
most extreme data point within ( whis*(75%-25%) ) data range.
positions (default 1,2,...,n) sets the horizontal positions of
the boxes. The ticks and limits are automatically set to match
the positions.
widths is either a scalar or a vector and sets the width of
each box. The default is 0.5, or 0.15*(distance between extreme
positions) if that is smaller.
x is an array or a sequence of vectors.
Returns a list of the lines added.

Addition kwargs: hold = [True|False] overrides default hold state
```

broken_barh(**args*, ***kwargs*)

```
A collection of horizontal bars spanning yrange with a sequence of
xranges
xranges : sequence of (xmin, xwidth)
yrange  : (ymin, ywidth)
kwargs are collections.BrokenBarHCollection properties
        alpha: float
        animated: [True | False]
        array: unknown
        axes: an axes instance
        clim: a length 2 sequence of floats
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cmap: a colormap
        color: matplotlib color arg or sequence of rgba tuples
        colorbar: unknown
        edgecolor: matplotlib color arg or sequence of rgba tuples
        facecolor: matplotlib color arg or sequence of rgba tuples
        figure: a matplotlib.figure.Figure instance
        label: any string
        linewidth: float or sequence of floats
        lod: [True | False]
        norm: unknown
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
these can either be a single argument, ie facecolors='black'
or a sequence of arguments for the various bars, ie
facecolors='black', 'red', 'green'

Addition kwargs: hold = [True|False] overrides default hold state
```

cla(**args*, ***kwargs*)

Clear the current axes

clabel(**args*, ***kwargs*)

```
clabel(CS, **kwargs) - add labels to line contours in CS,
       where CS is a ContourSet object returned by contour.
clabel(CS, V, **kwargs) - only label contours listed in V
keyword arguments:
* fontsize = None: as described in http://matplotlib.sf.net/fonts.html
* colors = None:
   - a tuple of matplotlib color args (string, float, rgb, etc),
     different labels will be plotted in different colors in the order
     specified
   - one string color, e.g. colors = 'r' or colors = 'red', all labels
     will be plotted in this color
   - if colors == None, the color of each label matches the color
     of the corresponding contour
* inline = True: controls whether the underlying contour is removed
             (inline = True) or not (False)
* fmt = '%1.3f': a format string for the label

Addition kwargs: hold = [True|False] overrides default hold state
```

clf()

Clear the current figure

clim(*vmin*=None, *vmax*=None)

Set the color limits of the current image
To apply clim to all axes images do
clim(0, 0.5)
If either vmin or vmax is None, the image min/max respectively will be used for color scaling.
If you want to set the clim of multiple images, use, for example for im in gca().get_images():
im.set_clim(0, 0.05)

close(**args*)

Close a figure window
close() by itself closes the current figure
close(num) closes figure number num
close(h) where h is a figure handle(instance) closes that figure
close('all') closes all the figure windows

cohere(**args*, ***kwargs*)

```
COHERE(x, y, NFFT=256, Fs=2, detrend=detrend_none,
      window=window_hanning, noverlap=0, **kwargs)
cohere the coherence between x and y.  Coherence is the normalized
cross spectral density
  Cxy = |Pxy|^2/(Pxx*Pyy)
The return value is (Cxy, f), where f are the frequencies of the
coherence vector.
See the PSD help for a description of the optional parameters.
kwargs are applied to the lines
Returns the tuple Cxy, freqs
Refs: Bendat & Piersol -- Random Data: Analysis and Measurement
  Procedures, John Wiley & Sons (1986)
kwargs control the Line2D properties of the coherence plot:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

colorbar(*mappable*=None, *cax*=None, ***kw*)

```
Add a colorbar to a plot.

Function signatures:

    colorbar(**kwargs)

    colorbar(mappable, **kwargs)

    colorbar(mappable, cax, **kwargs)

The optional arguments mappable and cax may be included in the kwargs;
they are image, ContourSet, etc. to which the colorbar applies, and
the axes object in which the colorbar will be drawn.  Defaults are
the current image and a new axes object created next to that image
after resizing the image.

kwargs are in two groups:
    axes properties:

        fraction    = 0.15; fraction of original axes to use for colorbar
        pad         = 0.05 if vertical, 0.15 if horizontal; fraction
                              of original axes between colorbar and
                              new image axes
        shrink      = 1.0; fraction by which to shrink the colorbar
        aspect      = 20; ratio of long to short dimensions


    colorbar properties:

        extend='neither', 'both', 'min', 'max'
                If not 'neither', make pointed end(s) for out-of-range
                values.  These are set for a given colormap using the
                colormap set_under and set_over methods.
        spacing='uniform', 'proportional'
                Uniform spacing gives each discrete color the same space;
                proportional makes the space proportional to the data interval.
        ticks=None, list of ticks, Locator object
                If None, ticks are determined automatically from the input.
        format=None, format string, Formatter object
                If none, the ScalarFormatter is used.
                If a format string is given, e.g. '%.3f', that is used.
                An alternative Formatter object may be given instead.
        drawedges=False, True
                If true, draw lines at color boundaries.

        The following will probably be useful only in the context of
        indexed colors (that is, when the mappable has norm=NoNorm()),
        or other unusual circumstances.

        boundaries=None or a sequence
        values=None or a sequence which must be of length 1 less than the
                sequence of boundaries.
                For each region delimited by adjacent entries in
                boundaries, the color mapped to the corresponding
                value in values will be used.
```

colorbar_classic(*mappable*=`None`, *cax*=`None`, *orientation*=`'vertical'`, *tickfmt*=`'%1.1f'`, *cspacing*=`'proportional'`, *clabels*=`None`, *drawedges*=`False`, *edgewidth*=`0.5`, *edgecolor*=`'k'`)

Create a colorbar for mappable; if mappable is None, use current image.
tickfmt is a format string to format the colorbar ticks
cax is a colorbar axes instance in which the colorbar will be placed. If None, as default axesd will be created resizing the current aqxes to make room for it. If not None, the supplied axes will be used and the other axes positions will be unchanged.
orientation is the colorbar orientation: one of 'vertical' | 'horizontal'
cspacing controls how colors are distributed on the colorbar. if cspacing == 'linear', each color occupies an equal area on the colorbar, regardless of the contour spacing. if cspacing == 'proportional' (Default), the area each color occupies on the the colorbar is proportional to the contour interval. Only relevant for a Contour image.
clabels can be a sequence containing the contour levels to be labelled on the colorbar, or None (Default). If clabels is None, labels for all contour intervals are displayed. Only relevant for a Contour image.
if drawedges == True, lines are drawn at the edges between each color on the colorbar. Default False.
edgecolor is the line color delimiting the edges of the colors on the colorbar (if drawedges == True). Default black ('k')
edgewidth is the width of the lines delimiting the edges of the colors on the colorbar (if drawedges == True). Default 0.5
return value is the colorbar axes instance

colormaps()

```
matplotlib provides the following colormaps.

  autumn bone cool copper flag gray hot hsv jet pink prism
  spring summer winter spectral

You can set the colormap for an image, pcolor, scatter, etc,
either as a keyword argumentdef con

>>> imshow(X, cmap=cm.hot)

or post-hoc using the corresponding pylab interface function

>>> imshow(X)
>>> hot()
>>> jet()

In interactive mode, this will update the colormap allowing you to
see which one works best for your data.
```

colors()

```
This is a do nothing function to provide you with help on how
matplotlib handles colors.

Commands which take color arguments can use several formats to
specify the colors.  For the basic builtin colors, you can use a
single letter

  b  : blue
  g  : green
  r  : red
  c  : cyan
  m  : magenta
  y  : yellow
  k  : black
  w  : white


For a greater range of colors, you have two options.  You can
specify the color using an html hex string, as in

  color = '#eeefff'

or you can pass an R,G,B tuple, where each of R,G,B are in the
range [0,1].

You can also use any legal html name for a color, like 'red',
'burlywood' and 'chartreuse'

The example below creates a subplot with a dark
slate gray background

   subplot(111, axisbg=(0.1843, 0.3098, 0.3098))

Here is an example that creates a pale turqoise title

  title('Is this the best color?', color='#afeeee')
```

connect(*s*, *func*)

```
Connect event with string s to func.  The signature of func is
  def func(event)
where event is a MplEvent.  The following events are recognized
'resize_event',
'draw_event',
'key_press_event',
'key_release_event',
'button_press_event',
'button_release_event',
'motion_notify_event',
'pick_event',


For the three events above, if the mouse is over the axes,
the variable event.inaxes will be set to the axes it is over,
and additionally, the variables event.xdata and event.ydata
will be defined.  This is the mouse location in data coords.
See backend_bases.MplEvent.

return value is a connection id that can be used with
mpl_disconnect
```

contour(**args*, ***kwargs*)

```
contour and contourf draw contour lines and filled contours,
respectively.  Except as noted, function signatures and return
values are the same for both versions.
contourf differs from the Matlab (TM) version in that it does not
    draw the polygon edges, because the contouring engine yields
    simply connected regions with branch cuts.  To draw the edges,
    add line contours with calls to contour.
Function signatures
contour(Z) - make a contour plot of an array Z. The level
         values are chosen automatically.
contour(X,Y,Z) - X,Y specify the (x,y) coordinates of the surface
contour(Z,N) and contour(X,Y,Z,N) - contour N automatically-chosen
         levels.
contour(Z,V) and contour(X,Y,Z,V) - draw len(V) contour lines,
         at the values specified in sequence V
contourf(..., V) - fill the (len(V)-1) regions between the
         values in V
contour(Z, **kwargs) - Use keyword args to control colors, linewidth,
            origin, cmap ... see below
X, Y, and Z must be arrays with the same dimensions.
Z may be a masked array, but filled contouring may not handle
           internal masked regions correctly.
C = contour(...) returns a ContourSet object.
Optional keyword args are shown with their defaults below (you must
use kwargs for these):
    * colors = None; or one of the following:
      - a tuple of matplotlib color args (string, float, rgb, etc),
      different levels will be plotted in different colors in the order
      specified
      -  one string color, e.g. colors = 'r' or colors = 'red', all levels
      will be plotted in this color
      - if colors == None, the colormap specified by cmap will be used
    * alpha=1.0 : the alpha blending value
    * cmap = None: a cm Colormap instance from matplotlib.cm.
      - if cmap == None and colors == None, a default Colormap is used.
    * norm = None: a matplotlib.colors.Normalize instance for
      scaling data values to colors.
      - if norm == None, and colors == None, the default
        linear scaling is used.
    * origin = None: 'upper'|'lower'|'image'|None.
      If 'image', the rc value for image.origin will be used.
      If None (default), the first value of Z will correspond
      to the lower left corner, location (0,0).
      This keyword is active only if contourf is called with
      one or two arguments, that is, without explicitly
      specifying X and Y.
    * extent = None: (x0,x1,y0,y1); also active only if X and Y
      are not specified.  If origin is not None, then extent is
      interpreted as in imshow: it gives the outer pixel boundaries.
      In this case, the position of Z[0,0] is the center of the
      pixel, not a corner.
      If origin is None, then (x0,y0) is the position of Z[0,0],
      and (x1,y1) is the position of Z[-1,-1].
    * locator = None: an instance of a ticker.Locator subclass;
      default is MaxNLocator.  It is used to determine the
      contour levels if they are not given explicitly via the
```

contourf(**args*, ***kwargs*)

```
contour and contourf draw contour lines and filled contours,
respectively.  Except as noted, function signatures and return
values are the same for both versions.
contourf differs from the Matlab (TM) version in that it does not
    draw the polygon edges, because the contouring engine yields
    simply connected regions with branch cuts.  To draw the edges,
    add line contours with calls to contour.
Function signatures
contour(Z) - make a contour plot of an array Z. The level
         values are chosen automatically.
contour(X,Y,Z) - X,Y specify the (x,y) coordinates of the surface
contour(Z,N) and contour(X,Y,Z,N) - contour N automatically-chosen
         levels.
contour(Z,V) and contour(X,Y,Z,V) - draw len(V) contour lines,
         at the values specified in sequence V
contourf(..., V) - fill the (len(V)-1) regions between the
         values in V
contour(Z, **kwargs) - Use keyword args to control colors, linewidth,
            origin, cmap ... see below
X, Y, and Z must be arrays with the same dimensions.
Z may be a masked array, but filled contouring may not handle
           internal masked regions correctly.
C = contour(...) returns a ContourSet object.
Optional keyword args are shown with their defaults below (you must
use kwargs for these):
    * colors = None; or one of the following:
      - a tuple of matplotlib color args (string, float, rgb, etc),
      different levels will be plotted in different colors in the order
      specified
      -  one string color, e.g. colors = 'r' or colors = 'red', all levels
      will be plotted in this color
      - if colors == None, the colormap specified by cmap will be used
    * alpha=1.0 : the alpha blending value
    * cmap = None: a cm Colormap instance from matplotlib.cm.
      - if cmap == None and colors == None, a default Colormap is used.
    * norm = None: a matplotlib.colors.Normalize instance for
      scaling data values to colors.
      - if norm == None, and colors == None, the default
        linear scaling is used.
    * origin = None: 'upper'|'lower'|'image'|None.
      If 'image', the rc value for image.origin will be used.
      If None (default), the first value of Z will correspond
      to the lower left corner, location (0,0).
      This keyword is active only if contourf is called with
      one or two arguments, that is, without explicitly
      specifying X and Y.
    * extent = None: (x0,x1,y0,y1); also active only if X and Y
      are not specified.  If origin is not None, then extent is
      interpreted as in imshow: it gives the outer pixel boundaries.
      In this case, the position of Z[0,0] is the center of the
      pixel, not a corner.
      If origin is None, then (x0,y0) is the position of Z[0,0],
      and (x1,y1) is the position of Z[-1,-1].
    * locator = None: an instance of a ticker.Locator subclass;
      default is MaxNLocator.  It is used to determine the
      contour levels if they are not given explicitly via the
```

cool()

set the default colormap to cool and apply to current image if any. See help(colormaps) for more information

copper()

set the default colormap to copper and apply to current image if any. See help(colormaps) for more information

csd(**args*, ***kwargs*)

```
CSD(x, y, NFFT=256, Fs=2, detrend=detrend_none,
    window=window_hanning, noverlap=0, **kwargs)
The cross spectral density Pxy by Welches average periodogram method.
The vectors x and y are divided into NFFT length segments.  Each
segment is detrended by function detrend and windowed by function
window.  The product of the direct FFTs of x and y are averaged over
each segment to compute Pxy, with a scaling to correct for power loss
due to windowing.
See the PSD help for a description of the optional parameters.
Returns the tuple Pxy, freqs.  Pxy is the cross spectrum (complex
valued), and 10*log10(|Pxy|) is plotted
Refs:
  Bendat & Piersol -- Random Data: Analysis and Measurement
    Procedures, John Wiley & Sons (1986)
kwargs control the Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

delaxes(**args*)

delaxes(ax) - remove ax from the current figure. If ax doesn't exist an error will be raised.
delaxes(): delete the current axes

disconnect(*cid*)

disconnect callback id cid

draw()

redraw the current figure

errorbar(**args, **kwargs*)

```
ERRORBAR(x, y, yerr=None, xerr=None,
         fmt='b-', ecolor=None, capsize=3, barsabove=False)
Plot x versus y with error deltas in yerr and xerr.
Vertical errorbars are plotted if yerr is not None
Horizontal errorbars are plotted if xerr is not None
xerr and yerr may be any of:
    a rank-0, Nx1 Numpy array  - symmetric errorbars +/- value
    an N-element list or tuple - symmetric errorbars +/- value
    a rank-1, Nx2 Numpy array  - asymmetric errorbars -column1/+column2
Alternatively, x, y, xerr, and yerr can all be scalars, which
plots a single error bar at x, y.
    fmt is the plot format symbol for y.  if fmt is None, just
    plot the errorbars with no line symbols.  This can be useful
    for creating a bar plot with errorbars
    ecolor is a matplotlib color arg which gives the color the
    errorbar lines; if None, use the marker color.
    capsize is the size of the error bar caps in points
    barsabove, if True, will plot the errorbars above the plot symbols
    - default is below
    kwargs are passed on to the plot command for the markers.
      So you can add additional key=value pairs to control the
      errorbar markers.  For example, this code makes big red
      squares with thick green edges
      >>> x,y,yerr = rand(3,10)
      >>> errorbar(x, y, yerr, marker='s',
                   mfc='red', mec='green', ms=20, mew=4)
     mfc, mec, ms and mew are aliases for the longer property
     names, markerfacecolor, markeredgecolor, markersize and
     markeredgewith.
valid kwargs for the marker properties are
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform.transformation instance
```

figimage(**args*, ***kwargs*)

```
FIGIMAGE(X) # add non-resampled array to figure
FIGIMAGE(X, xo, yo) # with pixel offsets
FIGIMAGE(X, **kwargs) # control interpolation ,scaling, etc
Add a nonresampled figure to the figure from array X.  xo and yo are
offsets in pixels
X must be a float array
    If X is MxN, assume luminance (grayscale)
    If X is MxNx3, assume RGB
    If X is MxNx4, assume RGBA
The following kwargs are allowed:
  * cmap is a cm colormap instance, eg cm.jet.  If None, default to
    the rc image.cmap valuex
  * norm is a matplotlib.colors.Normalize instance; default is
    normalization().  This scales luminance -> 0-1
  * vmin and vmax are used to scale a luminance image to 0-1.  If
    either is None, the min and max of the luminance values will be
    used.  Note if you pass a norm instance, the settings for vmin and
    vmax will be ignored.
  * alpha = 1.0 : the alpha blending value
  * origin is either 'upper' or 'lower', which indicates where the [0,0]
    index of the array is in the upper left or lower left corner of
    the axes.  Defaults to the rc image.origin value
This complements the axes image (Axes.imshow) which will be resampled
to fit the current axes.  If you want a resampled image to fill the
entire figure, you can define an Axes with size [0,1,0,1].
A image.FigureImage instance is returned.

Addition kwargs: hold = [True|False] overrides default hold state
```

figlegend(*handles*, *labels*, *loc*, ***kwargs*)

```
Place a legend in the figure.  Labels are a sequence of
strings, handles is a sequence of line or patch instances, and
loc can be a string r an integer specifying the legend
location

USAGE:
  legend( (line1, line2, line3),
          ('label1', 'label2', 'label3'),
          'upper right')

See help(legend) for information about the location codes

A matplotlib.legend.Legend instance is returned
```

figtext(**args*, ***kwargs*)

```
Add text to figure at location x,y (relative 0-1 coords) See
the help for Axis text for the meaning of the other arguments
kwargs control the Text properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key 'pad' which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
        label: any string
        lod: [True | False]
        multialignment: ['left' | 'right' | 'center' ]
        name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees 'vertical' | 'horizontal'
        size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
        style or fontstyle: [ 'normal' | 'italic' | 'oblique']
        text: string or anything printable with '%s' conversion
        transform: a matplotlib.transform transformation instance
        variant: [ 'normal' | 'small-caps' ]
        verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
        visible: [True | False]
        weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
        x: float
        y: float
        zorder: any number
```

figure(*num*=None, *figsize*=None, *dpi*=None, *facecolor*=None, *edgecolor*=None, *frameon*=True, *FigureClass*=<class matplotlib.figure.Figure at 0x8659fbc>, ***kwargs*)

```
figure(num = None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')


Create a new figure and return a handle to it.  If num=None, the figure
number will be incremented and a new figure will be created.  The returned
figure objects have a .number attribute holding this number.

If num is an integer, and figure(num) already exists, make it
active and return the handle to it.  If figure(num) does not exist
it will be created.  Numbering starts at 1, matlab style

  figure(1)

If you are creating many figures, make sure you explicitly call "close"
on the figures you are not using, because this will enable pylab
to properly clean up the memory.

kwargs:

  figsize - width x height in inches; defaults to rc figure.figsize
  dpi     - resolution; defaults to rc figure.dpi
  facecolor - the background color; defaults to rc figure.facecolor
  edgecolor - the border color; defaults to rc figure.edgecolor

rcParams gives the default values from the matplotlibrc file

FigureClass is a Figure or derived class that will be passed on to
new_figure_manager in the backends which allows you to hook custom
Figureclasses into the pylab interface.  Additional kwargs will be
passed on to your figure init function
```

fill(**args*, ***kwargs*)

```
FILL(*args, **kwargs)
plot filled polygons.  *args is a variable length argument, allowing
for multiple x,y pairs with an optional color format string; see plot
for details on the argument parsing.  For example, all of the
following are legal, assuming ax is an Axes instance:
  ax.fill(x,y)            # plot polygon with vertices at x,y
  ax.fill(x,y, 'b' )      # plot polygon with vertices at x,y in blue
An arbitrary number of x, y, color groups can be specified, as in
  ax.fill(x1, y1, 'g', x2, y2, 'r')
Return value is a list of patches that were added
The same color strings that plot supports are supported by the fill
format string.
kwargs control the Polygon properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

flag()

set the default colormap to flag and apply to current image if any. See help(colormaps) for more information

gca(***kwargs*)

```
Return the current axis instance.  This can be used to control
axis properties either using set or the Axes methods.

Example:

  plot(t,s)
  set(gca(), 'xlim', [0,10])  # set the x axis limits

or

  plot(t,s)
  a = gca()
  a.set_xlim([0,10])          # does the same
```

gcf()

Return a handle to the current figure

gci()

get the current ScalarMappable instance (image or patch collection), or None if no images or patch collections have been defined. The commands imshow and figimage create images instances, and the commands pcolor and scatter create patch collection instances

get_current_fig_manager()

get_plot_commands()

gray()

set the default colormap to gray and apply to current image if any. See help(colormaps) for more information

grid(**args*, ***kwargs*)

```
GRID(self, b=None, **kwargs)
Set the axes grids on or off; b is a boolean
if b is None and len(kwargs)==0, toggle the grid state.  if
kwargs are supplied, it is assumed that you want a grid and b
is thus set to True
kawrgs are used to set the grid line properties, eg
  ax.grid(color='r', linestyle='-', linewidth=2)
Valid Line2D kwargs are
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
```

hist(**args*, ***kwargs*)

```
HIST(x, bins=10, normed=0, bottom=None,
     align='edge', orientation='vertical', width=None,
     log=False, **kwargs)
Compute the histogram of x.  bins is either an integer number of
bins or a sequence giving the bins.  x are the data to be binned.
The return values is (n, bins, patches)
If normed is true, the first element of the return tuple will
be the counts normalized to form a probability density, ie,
n/(len(x)*dbin).  In a probability density, the integral of
the histogram should be one (we assume equally spaced bins);
you can verify that with
  # trapezoidal integration of the probability density function
  from matplotlib.mlab import trapz
  pdf, bins, patches = ax.hist(...)
  print trapz(bins, pdf)
align = 'edge' | 'center'.  Interprets bins either as edge
or center values
orientation = 'horizontal' | 'vertical'.  If horizontal, barh
will be used and the "bottom" kwarg will be the left edges.
width: the width of the bars.  If None, automatically compute
the width.
log: if True, the histogram axis will be set to a log scale
kwargs are used to update the properties of the
hist Rectangles:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        edgecolor or ec: any matplotlib color
        facecolor or fc: any matplotlib color
        figure: a matplotlib.figure.Figure instance
        fill: [True | False]
        hatch: unknown
        label: any string
        linewidth or lw: float
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

hlines(**args*, ***kwargs*)

HLINES(y, xmin, xmax, colors='k', linestyle='solid', **kwargs) plot horizontal lines at each y from xmin to xmax. xmin or xmax can be scalars or len(x) numpy arrays. If they are scalars, then the respective values are constant, else the widths of the lines are determined by xmin and xmax colors is a line collections color args, either a single color or a len(x) list of colors linestyle is one of solid|dashed|dashdot|dotted Returns the LineCollection that was added
Addition kwargs: hold = [True|False] overrides default hold state

hold(*b*=`None`)

Set the hold state. If hold is None (default), toggle the hold state. Else set the hold state to boolean value b.
Eg hold() # toggle hold hold(True) # hold is on hold(False) # hold is off
When hold is True, subsequent plot commands will be added to the current axes. When hold is False, the current axes and figure will be cleared on the next plot command

hot()

set the default colormap to hot and apply to current image if any. See help(colormaps) for more information

hsv()

set the default colormap to hsv and apply to current image if any. See help(colormaps) for more information

imread(**args*, ***kwargs*)

return image file in fname as numerix array Return value is a MxNx4 array of 0-1 normalized floats

imshow(**args, **kwargs*)

```
IMSHOW(X, cmap=None, norm=None, aspect=None, interpolation=None,
       alpha=1.0, vmin=None, vmax=None, origin=None, extent=None)
IMSHOW(X) - plot image X to current axes, resampling to scale to axes
            size (X may be numarray/Numeric array or PIL image)
IMSHOW(X, **kwargs) - Use keyword args to control image scaling,
colormapping etc. See below for details
Display the image in X to current axes.  X may be a float array, a
UInt8 array or a PIL image. If X is an array, X can have the following
shapes:
    MxN    : luminance (grayscale, float array only)
    MxNx3  : RGB (float or UInt8 array)
    MxNx4  : RGBA (float or UInt8 array)
The value for each component of MxNx3 and MxNx4 float arrays should be
in the range 0.0 to 1.0; MxN float arrays may be normalised.
A matplotlib.image.AxesImage instance is returned
The following kwargs are allowed:
  * cmap is a cm colormap instance, eg cm.jet.  If None, default to rc
    image.cmap value (Ignored when X has RGB(A) information)
  * aspect is one of: auto, equal, or a number.  If None, default to rc
    image.aspect value
  * interpolation is one of:
    'nearest', 'bilinear', 'bicubic', 'spline16', 'spline36',
    'hanning', 'hamming', 'hermite', 'kaiser', 'quadric',
    'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc',
    'lanczos', 'blackman'
    if interpolation is None, default to rc
    image.interpolation.  See also th the filternorm and
    filterrad parameters
  * norm is a matplotlib.colors.Normalize instance; default is
    normalization().  This scales luminance -> 0-1 (only used for an
    MxN float array).
  * vmin and vmax are used to scale a luminance image to 0-1.  If
    either is None, the min and max of the luminance values will be
    used.  Note if you pass a norm instance, the settings for vmin and
    vmax will be ignored.
  * alpha = 1.0 : the alpha blending value
  * origin is 'upper' or 'lower', to place the [0,0]
    index of the array in the upper left or lower left corner of
    the axes.  If None, default to rc image.origin
  * extent is (left, right, bottom, top) data values of the
    axes.  The default assigns zero-based row, column indices
    to the x, y centers of the pixels.
  * shape is for raw buffer images
  * filternorm is a parameter for the antigrain image resize
    filter.  From the antigrain documentation, if normalize=1,
    the filter normalizes integer values and corrects the
    rounding errors. It doesn't do anything with the source
    floating point values, it corrects only integers according
    to the rule of 1.0 which means that any sum of pixel
    weights must be equal to 1.0.  So, the filter function
    must produce a graph of the proper shape.
 * filterrad: the filter radius for filters that have a radius
   parameter, ie when interpolation is one of: 'sinc',
   'lanczos' or 'blackman'
Additional kwargs are matplotlib.artist properties
```

ioff()

turn interactive mode off

ion()

turn interactive mode on

ishold()

Return the hold status of the current axes

isinteractive()

Return the interactive status

jet()

set the default colormap to jet and apply to current image if any. See help(colormaps) for more information

legend(**args, **kwargs*)

```
LEGEND(*args, **kwargs)
Place a legend on the current axes at location loc.  Labels are a
sequence of strings and loc can be a string or an integer specifying
the legend location
USAGE:
  Make a legend with existing lines
  >>> legend()
  legend by itself will try and build a legend using the label
  property of the lines/patches/collections.  You can set the label of
  a line by doing plot(x, y, label='my data') or line.set_label('my
  data'). If label is set to '_nolegend_', the item will not be shown
  in legend.
    # automatically generate the legend from labels
    legend( ('label1', 'label2', 'label3') )
    # Make a legend for a list of lines and labels
    legend( (line1, line2, line3), ('label1', 'label2', 'label3') )
    # Make a legend at a given location, using a location argument
    # legend( LABELS, LOC )  or
    # legend( LINES, LABELS, LOC )
    legend( ('label1', 'label2', 'label3'), loc='upper left')
    legend( (line1, line2, line3),  ('label1', 'label2', 'label3'), loc=2)
The location codes are
  'best' : 0,
  'upper right'  : 1, (default)
  'upper left'   : 2,
  'lower left'   : 3,
  'lower right'  : 4,
  'right'        : 5,
  'center left'  : 6,
  'center right' : 7,
  'lower center' : 8,
  'upper center' : 9,
  'center'       : 10,
If none of these are suitable, loc can be a 2-tuple giving x,y
in axes coords, ie,
  loc = 0, 1 is left top
  loc = 0.5, 0.5 is center, center
and so on.  The following kwargs are supported:
isaxes=True           # whether this is an axes legend
numpoints = 4         # the number of points in the legend line
prop = FontProperties(size='smaller')  # the font property
pad = 0.2             # the fractional whitespace inside the legend border
markerscale = 0.6     # the relative size of legend markers vs. original
shadow                # if True, draw a shadow behind legend
labelsep = 0.005     # the vertical space between the legend entries
handlelen = 0.05     # the length of the legend lines
handletextsep = 0.02 # the space between the legend line and legend text
axespad = 0.02       # the border between the axes and legend edge
```

loglog(**args, **kwargs*)

```
LOGLOG(*args, **kwargs)
Make a loglog plot with log scaling on the a and y axis.  The args
to semilog x are the same as the args to plot.  See help plot for
more info.
Optional keyword args supported are any of the kwargs
supported by plot or set_xscale or set_yscale.  Notable, for
log scaling:
  * basex: base of the x logarithm
  * subsx: the location of the minor ticks; None defaults to
    autosubs, which depend on the number of decades in the
    plot; see set_xscale for details
  * basey: base of the y logarithm
  * subsy: the location of the minor yticks; None defaults to
    autosubs, which depend on the number of decades in the
    plot; see set_yscale for details
The remaining valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

matshow(*A*, *fignum*=None, ***kw*)

```
Display an array as a matrix in a new figure window.

The origin is set at the upper left hand corner and rows (first dimension
of the array) are displayed horizontally.  The aspect ratio of the figure
window is that of the array, unless this would make an excessively
short or narrow figure.

Tick labels for the xaxis are placed on top.

With one exception, keyword arguments are passed to
imshow().

Special keyword argument which is NOT passed to imshow():

  - fignum(None): by default, matshow() creates a new figure window with
  automatic numbering.  If fignum is given as an integer, the created
  figure will use this figure number.  Because of how matshow() tries to
  set the figure aspect ratio to be the one of the array, if you provide
  the number of an already existing figure, strange things may happen.

  if fignum is False or 0, a new figure window will NOT be created.

Example usage:

def samplemat(dims):
    aa = zeros(dims)
    for i in range(min(dims)):
        aa[i,i] = i
    return aa

dimlist = [(12,12),(128,64),(64,512),(2048,256)]

for d in dimlist:
    im = matshow(samplemat(d))
show()
```

over(*func*, **args*, ***kwargs*)

Call func(*args, **kwargs) with hold(True) and then restore the hold state

pcolor(**args*, ***kwargs*)

```
pcolor(*args, **kwargs): pseudocolor plot of a 2-D array
Function signatures
  pcolor(C, **kwargs)
  pcolor(X, Y, C, **kwargs)
C is the array of color values
X and Y, if given, specify the (x,y) coordinates of the colored
quadrilaterals; the quadrilateral for C[i,j] has corners at
(X[i,j],Y[i,j]), (X[i,j+1],Y[i,j+1]), (X[i+1,j],Y[i+1,j]),
(X[i+1,j+1],Y[i+1,j+1]).  Ideally the dimensions of X and Y
should be one greater than those of C; if the dimensions are the
same, then the last row and column of C will be ignored.
Note that the the column index corresponds to the x-coordinate,
and the row index corresponds to y; for details, see
the "Grid Orientation" section below.
If either or both of X and Y are 1-D arrays or column vectors,
they will be expanded as needed into the appropriate 2-D arrays,
making a rectangular grid.
X,Y and C may be masked arrays.  If either C[i,j], or one
of the vertices surrounding C[i,j] (X or Y at [i,j],[i+1,j],
[i,j+1],[i=1,j+1]) is masked, nothing is plotted.
Optional keyword args are shown with their defaults below (you must
use kwargs for these):
  * cmap = cm.jet : a cm Colormap instance from matplotlib.cm.
    defaults to cm.jet
  * norm = Normalize() : matplotlib.colors.Normalize instance
    is used to scale luminance data to 0,1.
  * vmin=None and vmax=None : vmin and vmax are used in conjunction
    with norm to normalize luminance data.  If either are None, the
    min and max of the color array C is used.  If you pass a norm
    instance, vmin and vmax will be None
  * shading = 'flat' : or 'faceted'.  If 'faceted', a black grid is
    drawn around each rectangle; if 'flat', edges are not drawn
  * alpha=1.0 : the alpha blending value
Return value is a matplotlib.collections.PatchCollection
object
Grid Orientation
    The orientation follows the Matlab(TM) convention: an
    array C with shape (nrows, ncolumns) is plotted with
    the column number as X and the row number as Y, increasing
    up; hence it is plotted the way the array would be printed,
    except that the Y axis is reversed.  That is, C is taken
    as C(y,x).
    Similarly for meshgrid:
        x = arange(5)
        y = arange(3)
        X, Y = meshgrid(x,y)
    is equivalent to
        X = array([[0, 1, 2, 3, 4],
                   [0, 1, 2, 3, 4],
                   [0, 1, 2, 3, 4]])
        Y = array([[0, 0, 0, 0, 0],
                   [1, 1, 1, 1, 1],
                   [2, 2, 2, 2, 2]])
    so if you have
        C = rand( len(x), len(y))
```

pcolor_classic(**args*, ***kwargs*)

pcolor_classic is no longer available; please use pcolor, which is a drop-in replacement.

pcolormesh(**args*, ***kwargs*)

```
PCOLORMESH(*args, **kwargs)
Function signatures
  PCOLORMESH(C) - make a pseudocolor plot of matrix C
  PCOLORMESH(X, Y, C) - a pseudo color plot of C on the matrices X and Y
  PCOLORMESH(C, **kwargs) - Use keyword args to control colormapping and
                        scaling; see below
C may be a masked array, but X and Y may not.  Masked array support
is implemented via cmap and norm; in contrast, pcolor simply does
not draw quadrilaterals with masked colors or vertices.
Optional keyword args are shown with their defaults below (you must
use kwargs for these):
  * cmap = cm.jet : a cm Colormap instance from matplotlib.cm.
    defaults to cm.jet
  * norm = Normalize() : matplotlib.colors.Normalize instance
    is used to scale luminance data to 0,1.  Instantiate it
    with clip=False if C is a masked array.
  * vmin=None and vmax=None : vmin and vmax are used in conjunction
    with norm to normalize luminance data.  If either are None, the
    min and max of the color array C is used.
  * shading = 'flat' : or 'faceted'.  If 'faceted', a black grid is
    drawn around each rectangle; if 'flat', edge colors are same as
    face colors
  * alpha=1.0 : the alpha blending value
Return value is a matplotlib.collections.PatchCollection
object
See pcolor for an explantion of the grid orientation and the
expansion of 1-D X and/or Y to 2-D arrays.
kwargs can be used to control the QuadMesh polygon collection properties:
        alpha: float
        animated: [True | False]
        array: unknown
        axes: an axes instance
        clim: a length 2 sequence of floats
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cmap: a colormap
        color: matplotlib color arg or sequence of rgba tuples
        colorbar: unknown
        edgecolor: matplotlib color arg or sequence of rgba tuples
        facecolor: matplotlib color arg or sequence of rgba tuples
        figure: a matplotlib.figure.Figure instance
        label: any string
        linewidth: float or sequence of floats
        lod: [True | False]
        norm: unknown
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

pie(**args*, ***kwargs*)

```
PIE(x, explode=None, labels=None,
    colors=('b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'),
    autopct=None, pctdistance=0.6, shadow=False)
Make a pie chart of array x.  The fractional area of each wedge is
given by x/sum(x).  If sum(x)<=1, then the values of x give the
fractional area directly and the array will not be normalized.
  - explode, if not None, is a len(x) array which specifies the
    fraction of the radius to offset that wedge.
  - colors is a sequence of matplotlib color args that the pie chart
    will cycle.
  - labels, if not None, is a len(x) list of labels.
  - autopct, if not None, is a string or function used to label the
    wedges with their numeric value.  The label will be placed inside
    the wedge.  If it is a format string, the label will be fmt%pct.
    If it is a function, it will be called
  - pctdistance is the ratio between the center of each pie slice
    and the start of the text generated by autopct.  Ignored if autopct
    is None; default is 0.6.
  - shadow, if True, will draw a shadow beneath the pie.
The pie chart will probably look best if the figure and axes are
square.  Eg,
  figure(figsize=(8,8))
  ax = axes([0.1, 0.1, 0.8, 0.8])
Return value:
  If autopct is None, return a list of (patches, texts), where patches
  is a sequence of matplotlib.patches.Wedge instances and texts is a
  list of the label Text instnaces
  If autopct is not None, return (patches, texts, autotexts), where
  patches and texts are as above, and autotexts is a list of text
  instances for the numeric labels

Addition kwargs: hold = [True|False] overrides default hold state
```

pink()

set the default colormap to pink and apply to current image if any. See help(colormaps) for more information

plot(**args*, ***kwargs*)

```
PLOT(*args, **kwargs)
Plot lines and/or markers to the Axes.  *args is a variable length
argument, allowing for multiple x,y pairs with an optional format
string.  For example, each of the following is legal
    plot(x,y)            # plot x and y using the default line style and color
    plot(x,y, ’bo’)      # plot x and y using blue circle markers
    plot(y)              # plot y using x as index array 0..N-1
    plot(y, ’r+’)        # ditto, but with red plusses
If x and/or y is 2-Dimensional, then the corresponding columns
will be plotted.
An arbitrary number of x, y, fmt groups can be specified, as in
a.plot(x1, y1, ’g^’, x2, y2, ’g-’)
Return value is a list of lines that were added.
The following line styles are supported:
    -     : solid line
    --    : dashed line
    -.    : dash-dot line
    :     : dotted line
    .     : points
    ,     : pixels
    o     : circle symbols
    ^     : triangle up symbols
    v     : triangle down symbols
    <     : triangle left symbols
    >     : triangle right symbols
    s     : square symbols
    +     : plus symbols
    x     : cross symbols
    D     : diamond symbols
    d     : thin diamond symbols
    1     : tripod down symbols
    2     : tripod up symbols
    3     : tripod left symbols
    4     : tripod right symbols
    h     : hexagon symbols
    H     : rotated hexagon symbols
    p     : pentagon symbols
    |     : vertical line symbols
    _     : horizontal line symbols
    steps : use gnuplot style ’steps’ # kwarg only
The following color abbreviations are supported
    b  : blue
    g  : green
    r  : red
    c  : cyan
    m  : magenta
    y  : yellow
    k  : black
    w  : white
In addition, you can specify colors in many weird and
wonderful ways, including full names ’green’, hex strings
’#008000’, RGB or RGBA tuples (0,1,0,1) or grayscale
intensities as a string ’0.8’.
Line styles and colors are combined in a single format string, as in
’bo’ for blue circles.
The **kwargs can be used to set line properties (any property that has
```

plot_date(**args*, ***kwargs*)

```
PLOT_DATE(x, y, fmt='bo', tz=None, xdate=True, ydate=False, **kwargs)
Similar to the plot() command, except the x or y (or both) data
is considered to be dates, and the axis is labeled accordingly.
x or y (or both) can be a sequence of dates represented as
float days since 0001-01-01 UTC.
fmt is a plot format string.
tz is the time zone to use in labelling dates.  Defaults to rc value.
If xdate is True, the x-axis will be labeled with dates.
If ydate is True, the y-axis will be labeled with dates.
Note if you are using custom date tickers and formatters, it
may be necessary to set the formatters/locators after the call
to plot_date since plot_date will set the default tick locator
to AutoDateLocator (if the tick locator is not already set to
a DateLocator instance) and the default tick formatter to
AutoDateFormatter (if the tick formatter is not already set to
a DateFormatter instance).
Valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number
See matplotlib.dates for helper functions date2num, num2date
and drange for help on creating the required floating point dates
```



```
Addition kwargs: hold = [True|False] overrides default hold state
```

plotting()

Plotting commands axes - Create a new axes axis - Set or return the current axis limits bar - make a bar chart boxplot - make a box and whiskers chart cla - clear current axes clabel - label a contour plot clf - clear a figure window close - close a figure window colorbar - add a colorbar to the current figure cohere - make a plot of coherence contour - make a contour plot contourf - make a filled contour plot csd - make a plot of cross spectral density draw - force a redraw of the current figure errorbar - make an errorbar graph figlegend - add a legend to the figure figimage - add an image to the figure, w/o resampling figtext - add text in figure coords figure - create or change active figure fill - make filled polygons gca - return the current axes gcf - return the current figure gci - get the current image, or None get - get a handle graphics property hist - make a histogram hold - set the hold state on current axes legend - add a legend to the axes loglog - a log log plot imread - load image file into array imshow - plot image data matshow - display a matrix in a new figure preserving aspect pcolor - make a pseudocolor plot plot - make a line plot psd - make a plot of power spectral density quiver - make a direction field (arrows) plot rc - control the default params savefig - save the current figure scatter - make a scatter plot set - set a handle graphics property semilogx - log x axis semilogy - log y axis show - show the figures specgram - a spectrogram plot stem - make a stem plot subplot - make a subplot (numrows, numcols, axesnum) table - add a table to the axes text - add some text at location x,y to the current axes title - add a title to the current axes xlabel - add an xlabel to the current axes ylabel - add a ylabel to the current axes autumn - set the default colormap to autumn bone - set the default colormap to bone cool - set the default colormap to cool copper - set the default colormap to copper flag - set the default colormap to flag gray - set the default colormap to gray hot - set the default colormap to hot hsv - set the default colormap to hsv jet - set the default colormap to jet pink - set the default colormap to pink prism - set the default colormap to prism spring - set the default colormap to spring summer - set the default colormap to summer winter - set the default colormap to winter spectral - set the default colormap to spectral

polar(**args*, ***kwargs*)

POLAR(theta, r)
Make a polar plot. Multiple theta, r arguments are supported, with format strings, as in plot.

prism()

set the default colormap to prism and apply to current image if any. See help(colormaps) for more information

psd(**args*, ***kwargs*)

```
PSD(x, NFFT=256, Fs=2, detrend=detrend_none,
    window=window_hanning, noverlap=0, **kwargs)
The power spectral density by Welches average periodogram method.  The
vector x is divided into NFFT length segments.  Each segment is
detrended by function detrend and windowed by function window.
noperlap gives the length of the overlap between segments.  The
absolute(fft(segment))**2 of each segment are averaged to compute Pxx,
with a scaling to correct for power loss due to windowing.  Fs is the
sampling frequency.
    NFFT is the length of the fft segment; must be a power of 2
    Fs is the sampling frequency.
    detrend - the function applied to each segment before fft-ing,
      designed to remove the mean or linear trend.  Unlike in matlab,
      where the detrend parameter is a vector, in matplotlib is it a
      function.  The mlab module defines detrend_none, detrend_mean,
      detrend_linear, but you can use a custom function as well.
    window - the function used to window the segments.  window is a
      function, unlike in matlab(TM) where it is a vector.  mlab defines
      window_none, window_hanning, but you can use a custom function
      as well.
    noverlap gives the length of the overlap between segments.
Returns the tuple Pxx, freqs
For plotting, the power is plotted as 10*log10(pxx)) for decibels,
though pxx itself is returned
Refs:
  Bendat & Piersol -- Random Data: Analysis and Measurement
  Procedures, John Wiley & Sons (1986)
kwargs control the Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
```

quiver(**args*, ***kwargs*)

```
Plot a 2-D field of arrows.

Function signatures:

    quiver(U, V, **kw)
    quiver(U, V, C, **kw)
    quiver(X, Y, U, V, **kw)
    quiver(X, Y, U, V, C, **kw)

Arguments:

    X, Y give the x and y coordinates of the arrow locations
        (default is tail of arrow; see 'pivot' kwarg)
    U, V give the x and y components of the arrow vectors
    C is an optional array used to map colors to the arrows

    All arguments may be 1-D or 2-D arrays or sequences.
    If X and Y are absent, they will be generated as a uniform grid.
    If U and V are 2-D arrays but X and Y are 1-D, and if
        len(X) and len(Y) match the column and row dimensions
        of U, then X and Y will be expanded with meshgrid.

Keyword arguments (default given first):

  * units = 'width' | 'height' | 'dots' | 'inches' | 'x' | 'y'
            arrow units; the arrow dimensions *except for length*
            are in multiples of this unit.
  * scale = None | float
            data units per arrow unit, e.g. m/s per plot width;
            a smaller scale parameter makes the arrow longer.
            If None, a simple autoscaling algorithm is used, based
            on the average vector length and the number of vectors.

    Arrow dimensions and scales can be in any of several units:

    'width' or 'height': the width or height of the axes
    'dots' or 'inches':  pixels or inches, based on the figure dpi
    'x' or 'y': X or Y data units

    In all cases the arrow aspect ratio is 1, so that if U==V the angle
    of the arrow on the plot is 45 degrees CCW from the X-axis.

    The arrows scale differently depending on the units, however.
    For 'x' or 'y', the arrows get larger as one zooms in; for other
    units, the arrow size is independent of the zoom state.  For
    'width or 'height', the arrow size increases with the width and
    height of the axes, respectively, when the the window is resized;
    for 'dots' or 'inches', resizing does not change the arrows.
```



```
  * width = ?       shaft width in arrow units; default depends on
                        choice of units, above, and number of vectors;
                        a typical starting value is about
                        0.005 times the width of the plot.
  * headwidth = 3    head width as multiple of shaft width
  * headlength = 5   head length as multiple of shaft width
```

quiver2(**args*, ***kwargs*)

```
Plot a 2-D field of arrows.

Function signatures:

    quiver(U, V, **kw)
    quiver(U, V, C, **kw)
    quiver(X, Y, U, V, **kw)
    quiver(X, Y, U, V, C, **kw)

Arguments:

    X, Y give the x and y coordinates of the arrow locations
        (default is tail of arrow; see 'pivot' kwarg)
    U, V give the x and y components of the arrow vectors
    C is an optional array used to map colors to the arrows

    All arguments may be 1-D or 2-D arrays or sequences.
    If X and Y are absent, they will be generated as a uniform grid.
    If U and V are 2-D arrays but X and Y are 1-D, and if
        len(X) and len(Y) match the column and row dimensions
        of U, then X and Y will be expanded with meshgrid.

Keyword arguments (default given first):

  * units = 'width' | 'height' | 'dots' | 'inches' | 'x' | 'y'
            arrow units; the arrow dimensions *except for length*
            are in multiples of this unit.
  * scale = None | float
            data units per arrow unit, e.g. m/s per plot width;
            a smaller scale parameter makes the arrow longer.
            If None, a simple autoscaling algorithm is used, based
            on the average vector length and the number of vectors.

    Arrow dimensions and scales can be in any of several units:

    'width' or 'height': the width or height of the axes
    'dots' or 'inches':  pixels or inches, based on the figure dpi
    'x' or 'y': X or Y data units

    In all cases the arrow aspect ratio is 1, so that if U==V the angle
    of the arrow on the plot is 45 degrees CCW from the X-axis.

    The arrows scale differently depending on the units, however.
    For 'x' or 'y', the arrows get larger as one zooms in; for other
    units, the arrow size is independent of the zoom state.  For
    'width or 'height', the arrow size increases with the width and
    height of the axes, respectively, when the the window is resized;
    for 'dots' or 'inches', resizing does not change the arrows.
```



```
  * width = ?       shaft width in arrow units; default depends on
                        choice of units, above, and number of vectors;
                        a typical starting value is about
                        0.005 times the width of the plot.
  * headwidth = 3    head width as multiple of shaft width
  * headlength = 5   head length as multiple of shaft width
```

quiverkey(**args*, ***kwargs*)

```
Add a key to a quiver plot.

Function signature:
    quiverkey(Q, X, Y, U, label, **kw)

Arguments:
    Q is the Quiver instance returned by a call to quiver.
    X, Y give the location of the key; additional explanation follows.
    U is the length of the key
    label is a string with the length and units of the key

Keyword arguments (default given first):
  * coordinates = 'axes' | 'figure' | 'data' | 'inches'
        Coordinate system and units for X, Y: 'axes' and 'figure'
        are normalized coordinate systems with 0,0 in the lower
        left and 1,1 in the upper right; 'data' are the axes
        data coordinates (used for the locations of the vectors
        in the quiver plot itself); 'inches' is position in the
        figure in inches, with 0,0 at the lower left corner.
  * color overrides face and edge colors from Q.
  * labelpos = 'N' | 'S' | 'E' | 'W'
        Position the label above, below, to the right, to the left
        of the arrow, respectively.
  * labelsep = 0.1 inches distance between the arrow and the label
  * labelcolor (defaults to default Text color)
  * fontproperties is a dictionary with keyword arguments accepted
        by the FontProperties initializer: family, style, variant,
        size, weight

    Any additional keyword arguments are used to override vector
    properties taken from Q.

    The positioning of the key depends on X, Y, coordinates, and
    labelpos.  If labelpos is 'N' or 'S', X,Y give the position
    of the middle of the key arrow.  If labelpos is 'E', X,Y
    positions the head, and if labelpos is 'W', X,Y positions the
    tail; in either of these two cases, X,Y is somewhere in the middle
    of the arrow+label key object.

Addition kwargs: hold = [True|False] overrides default hold state
```

raise_msg_to_str(*msg*)

msg is a return arg from a raise. Join with new lines

rc(**args*, ***kwargs*)

```
Set the current rc params.  Group is the grouping for the rc, eg
for lines.linewidth the group is 'lines', for axes.facecolor, the
group is 'axes', and so on.  Group may also be a list or tuple
of group names, eg ('xtick','ytick').  kwargs is a list of
attribute name/value pairs, eg
  rc('lines', linewidth=2, color='r')
sets the current rc params and is equivalent to
  rcParams['lines.linewidth'] = 2
  rcParams['lines.color'] = 'r'
The following aliases are available to save typing for interactive
users
    'lw'  : 'linewidth'
    'ls'  : 'linestyle'
    'c'   : 'color'
    'fc'  : 'facecolor'
    'ec'  : 'edgecolor'
    'mew' : 'markeredgewidth'
    'aa'  : 'antialiased'
Thus you could abbreviate the above rc command as
      rc('lines', lw=2, c='r')
Note you can use python's kwargs dictionary facility to store
dictionaries of default parameters.  Eg, you can customize the
font rc as follows
  font = {'family' : 'monospace',
          'weight' : 'bold',
          'size'   : 'larger',
         }
  rc('font', **font)  # pass in the font dict as kwargs
This enables you to easily switch between several configurations.
Use rcdefaults to restore the default rc params after changes.
```

rcdefaults()

Restore the default rc params - the ones that were created at matplotlib load time

rgrids(**args*, ***kwargs*)

```
Set/Get the radial locations of the gridlines and ticklabels

With no args, simply return lines, labels where lines is an
array of radial gridlines (Line2D instances) and labels is an
array of tick labels (Text instances).

  lines, labels = rgrids()

With arguments, the syntax is

  lines, labels = RGRIDS(radii, labels=None, angle=22.5, **kwargs)

The labels will appear at radial distances radii at angle

  labels, if not None, is a len(radii) list of strings of the
  labels to use at each angle.

  if labels is None, the self.rformatter will be used

Return value is a list of lines, labels where the lines are
matplotlib.Line2D instances and the labels are matplotlib.Text
instances.  Note that on input the labels argument is a list of
strings, and on output it is a list of Text instances

Examples
  # set the locations of the radial gridlines and labels
  lines, labels = rgrids( (0.25, 0.5, 1.0) )

  # set the locations and labels of the radial gridlines and labels
  lines, labels = rgrids( (0.25, 0.5, 1.0), ('Tom', 'Dick', 'Harry' )
```

savefig(**args*, ***kwargs*)

```
SAVEFIG(fname, dpi=None, facecolor='w', edgecolor='w',
orientation='portrait', papertype=None, format=None):
Save the current figure.
fname - the filename to save the current figure to.  The
        output formats supported depend on the backend being
        used.  and are deduced by the extension to fname.
        Possibilities are eps, jpeg, pdf, png, ps, svg.  fname
        can also be a file or file-like object - cairo backend
        only.  dpi - is the resolution in dots per inch.  If
        None it will default to the value savefig.dpi in the
        matplotlibrc file
facecolor and edgecolor are the colors of the figure rectangle
orientation is either 'landscape' or 'portrait' - not supported on
all backends; currently only on postscript output
papertype is is one of 'letter', 'legal', 'executive', 'ledger', 'a0'
through 'a10', or 'b0' through 'b10' - only supported for postscript
output
format - one of 'pdf', 'png', 'ps', 'svg'. It is used to specify the
         output when fname is a file or file-like object - cairo
         backend only.
```

scatter(**args*, ***kwargs*)

```
SCATTER(x, y, s=20, c='b', marker='o', cmap=None, norm=None,
    vmin=None, vmax=None, alpha=1.0, linewidths=None,
    faceted=True, **kwargs)
Supported function signatures:
    SCATTER(x, y, **kwargs)
    SCATTER(x, y, s, **kwargs)
    SCATTER(x, y, s, c, **kwargs)
Make a scatter plot of x versus y, where x, y are 1-D sequences
of the same length, N.
Arguments s and c can also be given as kwargs; this is encouraged
for readability.
    s is a size in points^2.  It is a scalar
      or an array of the same length as x and y.
    c is a color and can be a single color format string,
      or a sequence of color specifications of length N,
      or a sequence of N numbers to be mapped to colors
      using the cmap and norm specified via kwargs (see below).
      Note that c should not be a single numeric RGB or RGBA
      sequence because that is indistinguishable from an array
      of values to be colormapped. c can be a 2-D array in which
      the rows are RGB or RGBA, however.
The marker can be one of
    's' : square
    'o' : circle
    '^' : triangle up
    '>' : triangle right
    'v' : triangle down
    '<' : triangle left
    'd' : diamond
    'p' : pentagram
    'h' : hexagon
    '8' : octagon
If marker is None and verts is not None, verts is a sequence
of (x,y) vertices for a custom scatter symbol.
s is a size argument in points squared.
Any or all of x, y, s, and c may be masked arrays, in which
case all masks will be combined and only unmasked points
will be plotted.
Other keyword args; the color mapping and normalization arguments will
on be used if c is an array of floats
  * cmap = cm.jet : a colors.Colormap instance from matplotlib.cm.
    defaults to rc image.cmap
  * norm = Normalize() : matplotlib.colors.Normalize instance
    is used to scale luminance data to 0,1.
  * vmin=None and vmax=None : vmin and vmax are used in conjunction
    with norm to normalize luminance data.  If either are None, the
    min and max of the color array C is used.  Note if you pass a norm
    instance, your settings for vmin and vmax will be ignored
  * alpha =1.0 : the alpha value for the patches
  * linewidths, if None, defaults to (lines.linewidth,).  Note
    that this is a tuple, and if you set the linewidths
    argument you must set it as a sequence of floats, as
    required by RegularPolyCollection -- see
    matplotlib.collections.RegularPolyCollection for details
 * faceted: if True, will use the default edgecolor for the
   markers.  If False, will set the edgecolors to be the same
```

scatter_classic(**args*, ***kwargs*)

scatter_classic is no longer available; please use scatter. To help in porting, for comparison to the scatter docstring, here is the scatter_classic docstring: SCATTER_CLASSIC(x, y, s=None, c='b') Make a scatter plot of x versus y. s is a size (in data coords) and can be either a scalar or an array of the same length as x or y. c is a color and can be a single color format string or an length(x) array of intensities which will be mapped by the colormap jet. If size is None a default size will be used

sci(*im*)

Set the current image (the target of colormap commands like jet, hot or clim)

semilogx(**args*, ***kwargs*)

```
SEMILOGX(*args, **kwargs)
Make a semilog plot with log scaling on the x axis.  The args to
semilog x are the same as the args to plot.  See help plot for more
info.
Optional keyword args supported are any of the kwargs supported by
plot or set_xscale.  Notable, for log scaling:
    * basex: base of the logarithm
    * subsx: the location of the minor ticks; None defaults to
      autosubs, which depend on the number of decades in the
      plot; see set_xscale for details
The remaining valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

semilogy(**args*, ***kwargs*)

```
SEMILOGY(*args, **kwargs):
Make a semilog plot with log scaling on the y axis.  The args to
semilogy are the same as the args to plot.  See help plot for more
info.
Optional keyword args supported are any of the kwargs supported by
plot or set_yscale.  Notable, for log scaling:
    * basey: base of the logarithm
    * subsy: a sequence of the location of the minor ticks;
      None defaults to autosubs, which depend on the number of
      decades in the plot; see set_yscale for details
The remaining valid kwargs are Line2D properties:
        alpha: float
        animated: [True | False]
        antialiased or aa: [True | False]
        axes: unknown
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color or c: any matplotlib color
        dash_capstyle: ['butt' | 'round' | 'projecting']
        dash_joinstyle: ['miter' | 'round' | 'bevel']
        dashes: sequence of on/off ink in points
        data: (array xdata, array ydata)
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle or ls: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' | ' ' | '' ]
        linewidth or lw: float value in points
        lod: [True | False]
        marker: [ '+' | ',' | '.' | '1' | '2' | '3' | '4'
        markeredgecolor or mec: any matplotlib color
        markeredgewidth or mew: float value in points
        markerfacecolor or mfc: any matplotlib color
        markersize or ms: float
        picker: [None|float|boolean|callable]
        solid_capstyle: ['butt' | 'round' |  'projecting']
        solid_joinstyle: ['miter' | 'round' | 'bevel']
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        xdata: array
        ydata: array
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

setp(**args*, ***kwargs*)

matplotlib supports the use of setp ("set property") and getp to set and get object properties, as well as to do introspection on the object For example, to set the linestyle of a line to be dashed, you can do

```
>>> line, = plot([1,2,3])
>>> setp(line, linestyle='--')
```

If you want to know the valid types of arguments, you can provide the name of the property you want to set without a value

```
>>> setp(line, 'linestyle')
    linestyle: [ '-' | '--' | '-.' | ':' | 'steps' | 'None' ]
```

If you want to see all the properties that can be set, and their possible values, you can do

```
>>> setp(line)
    ... long output listing omitted'
```

setp operates on a single instance or a list of instances. If you are in query mode introspecting the possible values, only the first instance in the sequence is used. When actually setting values, all the instances will be set. Eg, suppose you have a list of two lines, the following will make both lines thicker and red

```
>>> x = arange(0,1.0,0.01)
>>> y1 = sin(2*pi*x)
>>> y2 = sin(4*pi*x)
>>> lines = plot(x, y1, x, y2)
>>> setp(lines, linewidth=2, color='r')
```

setp works with the matlab(TM) style string/value pairs or with python kwargs. For example, the following are equivalent

```
>>> setp(lines, 'linewidth', 2, 'color', r')  # matlab style
>>> setp(lines, linewidth=2, color='r')       # python style
```

specgram(**args, **kwargs*)

```
SPECGRAM(x, NFFT=256, Fs=2, detrend=detrend_none,
         window=window_hanning, noverlap=128,
         cmap=None, xextent=None)
Compute a spectrogram of data in x.  Data are split into NFFT length
segements and the PSD of each section is computed.  The windowing
function window is applied to each segment, and the amount of overlap
of each segment is specified with noverlap.
    * cmap is a colormap; if None use default determined by rc
    * xextent is the image extent in the xaxes xextent=xmin, xmax -
      default 0, max(bins), 0, max(freqs) where bins is the return
      value from matplotlib.matplotlib.mlab.specgram
    * See help(psd) for information on the other keyword arguments.
Return value is (Pxx, freqs, bins, im), where
    bins are the time points the spectrogram is calculated over
    freqs is an array of frequencies
    Pxx is a len(times) x len(freqs) array of power
    im is a matplotlib.image.AxesImage.
Note: If x is real (i.e. non-complex) only the positive spectrum is
shown.  If x is complex both positive and negative parts of the
spectrum are shown.

Addition kwargs: hold = [True|False] overrides default hold state
```

spectral()

set the default colormap to spectral and apply to current image if any. See help(colormaps) for more information

spring()

set the default colormap to spring and apply to current image if any. See help(colormaps) for more information

spy(**args*, ***kwargs*)

```
spy(Z) plots the sparsity pattern of the 2-D array Z
If precision is None, any non-zero value will be plotted;
else, values of absolute(Z)>precision will be plotted.
The array will be plotted as it would be printed, with
the first index (row) increasing down and the second
index (column) increasing to the right.
By default aspect is 'equal' so that each array element
occupies a square space; set the aspect kwarg to 'auto'
to allow the plot to fill the plot box, or to any scalar
number to specify the aspect ratio of an array element
directly.
Two plotting styles are available: image or marker. Both
are available for full arrays, but only the marker style
works for scipy.sparse.spmatrix instances.
If marker and markersize are None, an image will be
returned and any remaining kwargs are passed to imshow;
else, a Line2D object will be returned with the value
of marker determining the marker type, and any remaining
kwargs passed to the axes plot method.
If marker and markersize are None, useful kwargs include:
    cmap
    alpha
See documentation for imshow() for details.
For controlling colors, e.g. cyan background and red marks, use:
    cmap = matplotlib.colors.ListedColormap(['c','r'])
If marker or markersize is not None, useful kwargs include:
    marker
    markersize
    color
See documentation for plot() for details.
Useful values for marker include:
    's'  square (default)
    'o'  circle
    '.'  point
    ','  pixel

Addition kwargs: hold = [True|False] overrides default hold state
```

stem(**args*, ***kwargs*)

STEM(x, y, linefmt='b-', markerfmt='bo', basefmt='r-') A stem plot plots vertical lines (using linefmt) at each x location from the baseline to y, and places a marker there using markerfmt. A horizontal line at 0 is is plotted using basefmt Return value is (markerline, stemlines, baseline) . See http://www.mathworks.com/access/helpdesk/help/techdoc/ref/stem.html for details and examples/stem_plot.py for a demo.
Addition kwargs: hold = [True|False] overrides default hold state

subplot(**args*, ***kwargs*)

```
Create a subplot command, creating axes with

  subplot(numRows, numCols, plotNum)

where plotNum=1 is the first plot number and increasing plotNums
fill rows first.  max(plotNum)==numRows*numCols

You can leave out the commas if numRows<=numCols<=plotNum<10, as
in

  subplot(211)    # 2 rows, 1 column, first (upper) plot

subplot(111) is the default axis

The background color of the subplot can be specified via keyword
argument 'axisbg', which takes a color string or gdk.Color as value, as in

subplot(211, axisbg='y')

See help(axes) for additional information on axes and subplot
keyword arguments.

New subplots that overlap old will delete the old axes.  If you do
not want this behavior, use fig.add_subplot or the axes command.  Eg

  from pylab import *
  plot([1,2,3])  # implicitly creates subplot(111)
  subplot(211)   # overlaps, subplot(111) is killed
  plot(rand(12), rand(12))
```

subplot_tool(*targetfig*=`None`)

Launch a subplot tool window for targetfig (default gcf)
A matplotlib.widgets.SubplotTool instance is returned

subplots_adjust(**args*, ***kwargs*)

```
subplots_adjust(left=None, bottom=None, right=None, top=None,
                wspace=None, hspace=None)

Tune the subplot layout via the figure.SubplotParams mechanism.
The parameter meanings (and suggested defaults) are

  left  = 0.125  # the left side of the subplots of the figure
  right = 0.9    # the right side of the subplots of the figure
  bottom = 0.1   # the bottom of the subplots of the figure
  top = 0.9      # the top of the subplots of the figure
  wspace = 0.2   # the amount of width reserved for blank space between subplots
  hspace = 0.2   # the amount of height reserved for white space between subplots

The actual defaults are controlled by the rc file
```

summer()

set the default colormap to summer and apply to current image if any. See help(colormaps) for more information

switch_backend(*newbackend*)

Swtich the default backend to newbackend. This feature is EXPERIMENTAL, and is only expected to work switching to an image backend. Eg, if you have a bunch of PS scripts that you want to run from an interactive ipython session, yuo may want to switch to the PS backend before running them to avoid having a bunch of GUI windows popup. If you try to interactively switch from one GUI backend to another, you will explode.
Calling this command will close all open windows.

table(**args*, ***kwargs*)

```
TABLE(cellText=None, cellColours=None,
      cellLoc='right', colWidths=None,
      rowLabels=None, rowColours=None, rowLoc='left',
      colLabels=None, colColours=None, colLoc='center',
      loc='bottom', bbox=None):
Add a table to the current axes.  Returns a table instance.  For
finer grained control over tables, use the Table class and add it
to the axes with add_table.
Thanks to John Gill for providing the class and table.
kwargs control the Table properties:
        alpha: float
        animated: [True | False]
        axes: an axes instance
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        figure: a matplotlib.figure.Figure instance
        fontsize: a float in points
        label: any string
        lod: [True | False]
        picker: [None|float|boolean|callable]
        transform: a matplotlib.transform transformation instance
        visible: [True | False]
        zorder: any number
```

text(**args, **kwargs*)

```
TEXT(x, y, s, fontdict=None, **kwargs)
Add text in string s to axis at location x,y (data coords)
  fontdict is a dictionary to override the default text properties.
  If fontdict is None, the defaults are determined by your rc
  parameters.
  withdash=True will create a TextWithDash instance instead
  of a Text instance.
Individual keyword arguments can be used to override any given
parameter
    text(x, y, s, fontsize=12)
The default transform specifies that text is in data coords,
alternatively, you can specify text in axis coords (0,0 lower left and
1,1 upper right).  The example below places text in the center of the
axes
    text(0.5, 0.5,’matplotlib’,
         horizontalalignment=’center’,
         verticalalignment=’center’,
         transform = ax.transAxes,
    )
You can put a rectangular box around the text instance (eg to
set a background color) by using the keyword bbox.  bbox is a
dictionary of matplotlib.patches.Rectangle properties (see help
for Rectangle for a list of these).  For example
 text(x, y, s, bbox=dict(facecolor=’red’, alpha=0.5))
Valid kwargs are Text properties
        alpha: float
        animated: [True | False]
        axes: an axes instance
        backgroundcolor: any matplotlib color
        bbox: rectangle prop dict plus key ’pad’ which is a pad in points
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        color: any matplotlib color
        family: [ ’serif’ | ’sans-serif’ | ’cursive’ | ’fantasy’ | ’monospace’ ]
        figure: a matplotlib.figure.Figure instance
        fontproperties: a matplotlib.font_manager.FontProperties instance
        horizontalalignment or ha: [ ’center’ | ’right’ | ’left’ ]
        label: any string
        lod: [True | False]
        multialignment: [’left’ | ’right’ | ’center’ ]
        name or fontname: string eg, [’Sans’ | ’Courier’ | ’Helvetica’ ...]
        picker: [None|float|boolean|callable]
        position: (x,y)
        rotation: [ angle in degrees ’vertical’ | ’horizontal’
        size or fontsize: [ size in points | relative size eg ’smaller’, ’x-large’ ]
        style or fontstyle: [ ’normal’ | ’italic’ | ’oblique’]
        text: string or anything printable with ’%s’ conversion
        transform: a matplotlib.transform transformation instance
        variant: [ ’normal’ | ’small-caps’ ]
        verticalalignment or va: [ ’center’ | ’top’ | ’bottom’ ]
        visible: [True | False]
        weight or fontweight: [ ’normal’ | ’bold’ | ’heavy’ | ’light’ | ’ultrabold’ | ’ultralight’]
        x: float
        y: float
        zorder: any number
```

thetagrids(**args*, ***kwargs*)

```
Set/Get the theta locations of the gridlines and ticklabels

If no arguments are passed, return lines, labels where lines is an
array of radial gridlines (Line2D instances) and labels is an
array of tick labels (Text instances).

  lines, labels = thetagrids()

Otherwise the syntax is

  lines, labels = THETAGRIDS(angles, labels=None, fmt='%d', frac = 1.1)

set the angles at which to place the theta grids (these gridlines
are equal along the theta dimension).  angles is in degrees

  labels, if not None, is a len(angles) list of strings of the
  labels to use at each angle.

  if labels is None, the labels with be fmt%angle

  frac is the fraction of the polar axes radius at which to place
  the label (1 is the edge).Eg 1.05 isd outside the axes and 0.95
  is inside the axes

Return value is a list of lines, labels where the lines are
matplotlib.Line2D instances and the labels are matplotlib.Text
instances.  Note that on input the labels argument is a list of
strings, and on output it is a list of Text instances

Examples:

  # set the locations of the radial gridlines and labels
  lines, labels = thetagrids( range(45,360,90) )

  # set the locations and labels of the radial gridlines and labels
  lines, labels = thetagrids( range(45,360,90), ('NE', 'NW', 'SW','SE') )
```

title(*s*, **args*, ***kwargs*)

```
Set the title of the current axis to s

Default font override is:
  override = {
    'fontsize'            : 'medium',
    'verticalalignment'   : 'bottom',
    'horizontalalignment' : 'center'
  }

See the text docstring for information of how override and the
optional args work
```

twinx(*ax*=`None`)

Make a second axes overlay ax (or the current axes if ax is None) sharing the xaxis. The ticks for ax2 will be placed on the right, and the ax2 instance is returned. See examples/two_scales.py

twiny(*ax*=`None`)

Make a second axes overlay ax (or the current axes if ax is None) sharing the yaxis. The ticks for ax2 will be placed on the top, and the ax2 instance is returned.

vlines(**args*, ***kwargs*)

```
VLINES(x, ymin, ymax, color='k')
Plot vertical lines at each x from ymin to ymax.  ymin or ymax can be
scalars or len(x) numpy arrays.  If they are scalars, then the
respective values are constant, else the heights of the lines are
determined by ymin and ymax
colors is a line collections color args, either a single color
or a len(x) list of colors
linestyle is one of solid|dashed|dashdot|dotted
Returns the LineCollection that was added
kwargs are LineCollection properties:
        alpha: float or sequence of floats
        animated: [True | False]
        array: unknown
        axes: an axes instance
        clim: a length 2 sequence of floats
        clip_box: a matplotlib.transform.Bbox instance
        clip_on: [True | False]
        clip_path: an agg.path_storage instance
        cmap: a colormap
        color: matplotlib color arg or sequence of rgba tuples
        colorbar: unknown
        figure: a matplotlib.figure.Figure instance
        label: any string
        linestyle: ['solid' | 'dashed', 'dashdot', 'dotted' |  (offset, on-off-dash-seq) ]
        linewidth: float or sequence of floats
        lod: [True | False]
        norm: unknown
        picker: [None|float|boolean|callable]
        segments: unknown
        transform: a matplotlib.transform transformation instance
        verts: unknown
        visible: [True | False]
        zorder: any number

Addition kwargs: hold = [True|False] overrides default hold state
```

winter()

set the default colormap to winter and apply to current image if any. See help(colormaps) for more information

xcorr(**args*, ***kwargs*)

XCORR(x, y, normed=False, detrend=detrend_none, usevlines=False, **kwargs): Plot the cross correlation between x and y. If normed=True, normalize the data but the cross correlation at 0-th lag. x and y are detrended by the detrend callable (default no normalization. x and y must be equal length data are plotted as plot(lags, c, **kwargs) return value is lags, c, line where lags are a length 2*maxlags+1 lag vector, c is the 2*maxlags+1 auto correlation vector, and line is a Line2D instance returned by plot. The default linestyle is None and the default marker is 'o', though these can be overridden with keyword args. The cross correlation is performed with numerix cross_correlate with mode=2. If usevlines is True, Axes.vlines rather than Axes.plot is used to draw vertical lines from the origin to the acorr. Otherwise the plotstyle is determined by the kwargs, which are Line2D properties. If usevlines, the return value is lags, c, linecol, b where linecol is the LineCollection and b is the x-axis if usevlines=True, kwargs are passed onto Axes.vlines if usevlines=False, kwargs are passed onto Axes.plot maxlags is a positive integer detailing the number of lags to show. The default value of None will return all (2*len(x)-1) lags. See the respective function for documentation on valid kwargs
Addition kwargs: hold = [True|False] overrides default hold state

xlabel(*s*, **args*, ***kwargs*)

```
Set the x axis label of the current axis to s

Default override is

  override = {
      'fontsize'            : 'small',
      'verticalalignment'   : 'top',
      'horizontalalignment' : 'center'
      }

See the text docstring for information of how override and
the optional args work
```

xlim(**args*, ***kwargs*)

```
Set/Get the xlimits of the current axes

xmin, xmax = xlim()   : return the current xlim
xlim( (xmin, xmax) )  : set the xlim to xmin, xmax
xlim( xmin, xmax )    : set the xlim to xmin, xmax

If you do not specify args, you can pass the xmin and xmax as
kwargs, eg

  xlim(xmax=3) # adjust the max leaving min unchanged
  xlim(xmin=1) # adjust the min leaving max unchanged

The new axis limits are returned as a length 2 tuple
```

xticks(**args*, ***kwargs*)

Set/Get the xlimits of the current ticklocs, labels
return locs, labels where locs is an array of tick locations and # labels is an array of tick labels. locs, labels = xticks()
set the locations of the xticks xticks(arange(6))
set the locations and labels of the xticks xticks(arange(5), ('Tom', 'Dick', 'Harry', 'Sally', 'Sue'))
The keyword args, if any, are text properties; see text for more information on text properties.

ylabel(*s*, **args*, ***kwargs*)

```
Set the y axis label of the current axis to s

Defaults override is

    override = {
       'fontsize'            : 'small',
       'verticalalignment'   : 'center',
       'horizontalalignment' : 'right',
       'rotation'='vertical' : }

See the text docstring for information of how override and the
optional args work
```

ylim(**args*, ***kwargs*)

```
Set/Get the ylimits of the current axes

ymin, ymax = ylim()   : return the current ylim
ylim( (ymin, ymax) )  : set the ylim to ymin, ymax
ylim( ymin, ymax )    : set the ylim to ymin, ymax

If you do not specify args, you can pass the ymin and ymax as
kwargs, eg

  ylim(ymax=3) # adjust the max leaving min unchanged
  ylim(ymin=1) # adjust the min leaving max unchanged

The new axis limits are returned as a length 2 tuple
```

yticks(**args*, ***kwargs*)

Set/Get the ylimits of the current ticklocs, labels
return locs, labels where locs is an array of tick locations and # labels is an array of tick labels. locs, labels = yticks()
set the locations of the yticks yticks(arange(6))
set the locations and labels of the yticks yticks(arange(5), ('Tom', 'Dick', 'Harry', 'Sally', 'Sue'))
The keyword args, if any, are text properties; see text for more information on text properties.

32 Module matplotlib.pyparsing

pyparsing module - Classes and methods to define and execute parsing grammars

The pyparsing module is an alternative approach to creating and executing simple grammars, vs. the traditional lex/yacc approach, or the use of regular expressions. With pyparsing, you don't need to learn a new syntax for defining grammars or matching expressions - the parsing module provides a library of classes that you use to construct the grammar directly in Python.

Here is a program to parse "Hello, World!" (or any greeting of the form "<salutation>, <addressee>!"):

```
from pyparsing import Word, alphas

# define grammar of a greeting
greet = Word( alphas ) + "," + Word( alphas ) + "!"

hello = "Hello, World!"
print hello, "->", greet.parseString( hello )
```

The program outputs the following:

```
Hello, World! -> ['Hello', ',', 'World', '!']
```

The Python representation of the grammar is quite readable, owing to the self-explanatory class names, and the use of '+', '|' and '^' operators.

The parsed results returned from parseString() can be accessed as a nested list, a dictionary, or an object with named attributes.

The pyparsing module handles some of the problems that are typically vexing when writing text parsers:

- extra or missing whitespace (the above program will also handle "Hello,World!", "Hello , World !", etc.)
- quoted strings
- embedded comments

32.1 Functions

_expanded(*p*)

col(*loc*, *strg*)

Returns current column within a string, counting newlines as line separators The first column is number 1.

delimitedList(*expr*, *delim*=`','`, *combine*=`False`)

Helper to define a delimited list of expressions - the delimiter defaults to ','. By default, the list elements and delimiters can have intervening whitespace, and comments, but this can be overridden by passing 'combine=True' in the constructor. If combine is set to True, the matching tokens are returned as a single token string, with the delimiters included; otherwise, the matching tokens are returned as a list of tokens, with the delimiters suppressed.

dictOf(*key*, *value*)

Helper to easily and clearly define a dictionary by specifying the respective patterns for the key and value. Takes care of defining the Dict, ZeroOrMore, and Group tokens in the proper order. The key pattern can include delimiting markers or punctuation, as long as they are suppressed, thereby leaving the significant key text. The value pattern can include named results, so that the Dict results can include named token fields.

downcaseTokens(*s*, *l*, *t*)

Helper parse action to convert tokens to lower case.

line(*loc*, *strg*)

Returns the line of text containing loc within a string, counting newlines as line separators The first line is number 1.

lineno(*loc*, *strg*)

Returns current line number within a string, counting newlines as line separators The first line is number 1.

makeHTMLTags(*tagStr*)

Helper to construct opening and closing tag expressions for HTML, given a tag name

makeXMLTags(*tagStr*)

Helper to construct opening and closing tag expressions for XML, given a tag name

nullDebugAction(**args*)

'Do-nothing' debug action, to suppress debugging output during parsing.

oneOf(*strs*, *caseless*=`False`, *useRegex*=`True`)

Helper to quickly define a set of alternative Literals, and makes sure to do longest-first testing when there is a conflict, regardless of the input order, but returns a MatchFirst for best performance.

removeQuotes(*s*, *l*, *t*)

Helper parse action for removing quotation marks from parsed quoted strings. To use, add this parse action to quoted string using:

```
quotedString.setParseAction( removeQuotes )
```

replaceWith(*replStr*)

Helper method for common parse actions that simply return a literal value. Especially useful when used with transformString().

srange(*s*)

Helper to easily define string ranges for use in Word construction. Borrows syntax from regexp '[]' string range definitions:

```
  srange("[0-9]")   -> "0123456789"
  srange("[a-z]")   -> "abcdefghijklmnopqrstuvwxyz"
  srange("[a-z$_]") -> "abcdefghijklmnopqrstuvwxyz$_"
```

The input string must be enclosed in []'s, and the returned string is the expanded character set joined into a single string. The values enclosed in the []'s may be:

```
  a single character
  an escaped character with a leading backslash (such as \- or \])
  an escaped hex character with a leading '\0x' (\0x21, which is a '!' character)
  an escaped octal character with a leading '\0' (\041, which is a '!' character)
  a range of any of the above, separated by a dash ('a-z', etc.)
  any combination of the above ('aeiouy', 'a-zA-Z0-9_$', etc.)
```

upcaseTokens(*s*, *l*, *t*)

Helper parse action to convert tokens to upper case.

32.2 Class And

__builtin__.object

matplotlib.pyparsing.ParserElement

matplotlib.pyparsing.ParseExpression

And

Requires all given ParseExpressions to be found in the given order. Expressions may be separated by whitespace. May be constructed using the '+' operator.

32.2.1 Methods

__init__(*self*, *exprs*, *savelist*=`True`)
Overrides: matplotlib.pyparsing.ParseExpression.__init__

__iadd__(*self*, *other*)

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseExpression.__str__

checkRecursion(*self*, *parseElementList*)
Overrides: matplotlib.pyparsing.ParserElement.checkRecursion

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseExpression: __getitem__, append, ignore, leaveWhitespace, setResultsName, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.2.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.2.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.3 Class CaselessKeyword

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
matplotlib.pyparsing.Keyword
CaselessKeyword

32.3.1 Methods

__init__(*self*, *matchString*, *identChars*=`'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123...`)
Overrides: matplotlib.pyparsing.Keyword.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.Keyword.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from Keyword: copy
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.3.2 Static Methods

Inherited from Keyword: setDefaultKeywordChars
Inherited from ParserElement: setDefaultWhitespaceChars

32.3.3 Class Variables

Name	Description
Inherited from Keyword: DEFAULT_KEYWORD_CHARS *(p. 462)*	
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.4 Class CaselessLiteral

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
matplotlib.pyparsing.Literal
CaselessLiteral

Token to match a specified string, ignoring case of letters. Note: the matched results will always be in the case of the given match string, NOT the case of the input text.

32.4.1 Methods

__init__(*self*, *matchString*)
Overrides: matplotlib.pyparsing.Literal.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.Literal.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.4.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.4.3 Class Variables

Name	Description
Inherited from Literal: __slotnames__ *(p. 465)*	
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.5 Class CharsNotIn

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
CharsNotIn

Token for matching words composed of characters *not* in a given set. Defined with string containing all disallowed characters, and an optional minimum, maximum, and/or exact length.

32.5.1 Methods

__init__(*self*, *notChars*, *min*=`1`, *max*=`0`, *exact*=`0`)
Overrides: matplotlib.pyparsing.Token.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParserElement.__str__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.5.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.5.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.6 Class Combine

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
matplotlib.pyparsing.TokenConverter
Combine

Converter to concatenate all matching tokens to a single string. By default, the matching patterns must also be contiguous in the input string; this can be disabled by specifying 'adjacent=False' in the constructor.

32.6.1 Methods

__init__(*self*, *expr*, *joinString*=`''`, *adjacent*=`True`)
Overrides: matplotlib.pyparsing.TokenConverter.__init__

ignore(*self*, *other*)

Define expression to be ignored (e.g., comments) while doing pattern matching; may be called repeatedly, to define multiple comment or other ignorable patterns.

Overrides: matplotlib.pyparsing.ParseElementEnhance.ignore extit(inherited documentation)

postParse(*self*, *instring*, *loc*, *tokenlist*)
Overrides: matplotlib.pyparsing.ParserElement.postParse

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, leaveWhitespace, parseImpl, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.6.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.6.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.7 Class Dict

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
matplotlib.pyparsing.TokenConverter
Dict

Converter to return a repetitive expression as a list, but also as a dictionary. Each element can also be referenced using the first token in the expression as its key. Useful for tabular report scraping when the first column can be used as a item key.

32.7.1 Methods

__init__(*self*, *exprs*)
Overrides: matplotlib.pyparsing.TokenConverter.__init__

postParse(*self*, *instring*, *loc*, *tokenlist*)
Overrides: matplotlib.pyparsing.ParserElement.postParse

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, ignore, leaveWhitespace, parseImpl, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.7.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.7.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.8 Class Each

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseExpression
Each

Requires all given ParseExpressions to be found, but in any order. Expressions may be separated by whitespace. May be constructed using the '&' operator.

32.8.1 Methods

__init__(*self*, *exprs*, *savelist*=`True`)
Overrides: matplotlib.pyparsing.ParseExpression.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseExpression.__str__

checkRecursion(*self*, *parseElementList*)
Overrides: matplotlib.pyparsing.ParserElement.checkRecursion

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseExpression: __getitem__, append, ignore, leaveWhitespace, setResultsName, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.8.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.8.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.9 Class Empty

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
Empty

An empty token, will always match.

32.9.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.Token.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseImpl, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.9.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.9.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.10 Class FollowedBy

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
FollowedBy

Lookahead matching of the given parse expression. FollowedBy does *not* advance the parsing position within the input string, it only verifies that the specified parse expression matches at the current position. FollowedBy always returns a null token list.

32.10.1 Methods

__init__(*self*, *expr*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, ignore, leaveWhitespace, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.10.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.10.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.11 Class Forward

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
Forward

Known Subclasses: _ForwardNoRecurse

Forward declaration of an expression to be defined later - used for recursive grammars, such as algebraic infix notation. When the expression is known, it is assigned to the Forward variable using the '<<' operator.

32.11.1 Methods

__init__(*self*, *other*=`None`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__init__

__lshift__(*self*, *other*)

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__str__

leaveWhitespace(*self*)

Disables the skipping of whitespace before matching the characters in the ParserElement's defined pattern. This is normally only used internally by the pyparsing module, but may be needed in some whitespace-sensitive grammars.

Overrides: matplotlib.pyparsing.ParseElementEnhance.leaveWhitespace extit(inherited documentation)

streamline(*self*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.streamline

validate(*self*, *validateTrace*=`[]`)

Check defined expressions for valid structure, check for infinite recursive definitions.

Overrides: matplotlib.pyparsing.ParseElementEnhance.validate extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: checkRecursion, ignore, parseImpl
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.11.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.11.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.12 Class GoToColumn

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
matplotlib.pyparsing.PositionToken
GoToColumn

Token to advance to a specific column of input text; useful for tabular report scraping.

32.12.1 Methods

__init__(*self*, *colno*)
Overrides: matplotlib.pyparsing.PositionToken.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

preParse(*self*, *instring*, *loc*)
Overrides: matplotlib.pyparsing.ParserElement.preParse

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.12.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.12.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.13 Class Group

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
matplotlib.pyparsing.TokenConverter
Group

Converter to return the matched tokens as a list - useful for returning tokens of ZeroOrMore and OneOrMore expressions.

32.13.1 Methods

__init__(*self*, *expr*)
Overrides: matplotlib.pyparsing.TokenConverter.__init__

postParse(*self*, *instring*, *loc*, *tokenlist*)
Overrides: matplotlib.pyparsing.ParserElement.postParse

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, ignore, leaveWhitespace, parseImpl, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.13.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.13.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.14 Class Keyword

```
__builtin__.object
  └─ matplotlib.pyparsing.ParserElement
       └─ matplotlib.pyparsing.Token
            └─ Keyword
```

Known Subclasses: CaselessKeyword

Token to exactly match a specified string as a keyword, that is, it must be immediately followed by a non-keyword character. Compare with Literal:

```
Literal("if") will match the leading 'if' in 'ifAndOnlyIf'.
Keyword("if") will not; it will only match the leading 'if in 'if x=1', or 'if(y==2)'
```

Accepts two optional constructor arguments in addition to the keyword string: identChars is a string of characters that would be valid identifier characters, defaulting to all alphanumerics + "_" and "$"; caseless allows case-insensitive matching, default is False.

32.14.1 Methods

__init__(*self*, *matchString*, *identChars*=`'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123...`, *caseless*=`False`)
Overrides: matplotlib.pyparsing.Token.__init__

copy(*self*)

Make a copy of this ParseElement. Useful for defining different parse actions for the same parsing pattern, using copies of the original parse element.

Overrides: matplotlib.pyparsing.ParserElement.copy extit(inherited documentation)

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.14.2 Static Methods

setDefaultKeywordChars(*chars*)

Overrides the default Keyword chars

Inherited from ParserElement: setDefaultWhitespaceChars

32.14.3 Class Variables

Name	Description
DEFAULT_KEYWORD_CHARS	**Value:** `'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_$'` *(type=`str`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.15 Class LineEnd

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
matplotlib.pyparsing.PositionToken
LineEnd

Matches if current position is at the end of a line within the parse string

32.15.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.PositionToken.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.15.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.15.3 Class Variables

Name	Description
__slotnames__	**Value:** [] *(type=list)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.16 Class LineStart

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
matplotlib.pyparsing.PositionToken
LineStart

Matches if current position is at the beginning of a line within the parse string

32.16.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.PositionToken.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=True)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

preParse(*self*, *instring*, *loc*)
Overrides: matplotlib.pyparsing.ParserElement.preParse

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.16.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.16.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.17 Class Literal

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
Literal

Known Subclasses: CaselessLiteral

Token to exactly match a specified string.

32.17.1 Methods

__init__(*self*, *matchString*)
Overrides: matplotlib.pyparsing.Token.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.17.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.17.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.18 Class MatchFirst

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseExpression
MatchFirst

Requires that at least one ParseExpression is found. If two expressions match, the first one listed is the one that will match. May be constructed using the '|' operator.

32.18.1 Methods

__init__(*self*, *exprs*, *savelist*=`False`)
Overrides: matplotlib.pyparsing.ParseExpression.__init__

__ior__(*self*, *other*)

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseExpression.__str__

checkRecursion(*self*, *parseElementList*)
Overrides: matplotlib.pyparsing.ParserElement.checkRecursion

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseExpression: __getitem__, append, ignore, leaveWhitespace, setResultsName, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.18.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.18.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.19 Class NoMatch

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
NoMatch

A token that will never match.

32.19.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.Token.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.19.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.19.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.20 Class NotAny

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
NotAny

Lookahead to disallow matching with the given parse expression. NotAny does *not* advance the parsing position within the input string, it only verifies that the specified parse expression does *not* match at the current position. Also, NotAny does *not* skip over leading whitespace. NotAny always returns a null token list. May be constructed using the '~' operator.

32.20.1 Methods

__init__(*self*, *expr*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__str__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: checkRecursion, ignore, leaveWhitespace, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.20.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.20.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.21 Class OneOrMore

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
OneOrMore

Repetition of one or more of the given expression.

32.21.1 Methods

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__str__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.parseImpl

setResultsName(*self*, *name*, *listAllMatches*=`False`)

Define name for referencing matching tokens as a nested attribute of the returned parse results. NOTE: this returns a *copy* of the original ParseElement object; this is so that the client can define a basic element, such as an integer, and reference it in multiple places with different names.

Overrides: matplotlib.pyparsing.ParserElement.setResultsName extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __init__, checkRecursion, ignore, leaveWhitespace, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.21.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.21.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.22 Class Optional

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
Optional

Optional matching of the given expression. A default return string can also be specified, if the optional expression is not found.

32.22.1 Methods

__init__(*self*, *exprs*, *default*=`None`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__str__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: checkRecursion, ignore, leaveWhitespace, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.22.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.22.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.23 Class Or

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseExpression
Or

Requires that at least one ParseExpression is found. If two expressions match, the expression that matches the longest string will be used. May be constructed using the '^' operator.

32.23.1 Methods

__init__(*self*, *exprs*, *savelist*=`False`)
Overrides: matplotlib.pyparsing.ParseExpression.__init__

__ixor__(*self*, *other*)

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseExpression.__str__

checkRecursion(*self*, *parseElementList*)
Overrides: matplotlib.pyparsing.ParserElement.checkRecursion

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseExpression: __getitem__, append, ignore, leaveWhitespace, setResultsName, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.23.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.23.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.24 Class ParseBaseException

__builtin__.object
exceptions.BaseException
exceptions.Exception
ParseBaseException

Known Subclasses: ParseException, ParseFatalException

base exception class for all parsing runtime exceptions

32.24.1 Methods

__init__(*self*, *pstr*, *loc*, *msg*, *elem*=None)
Overrides: exceptions.Exception.__init__

__getattr__(*self*, *aname*)

supported attributes by name are:

- lineno - returns the line number of the exception text
- col - returns the column number of the exception text
- line - returns the line containing the exception text

__repr__(*self*)
Overrides: exceptions.BaseException.__repr__

__str__(*self*)
Overrides: exceptions.BaseException.__str__

---

markInputline(*self*, *markerString*='>!<')

Extracts the exception line from the input string, and marks the location of the exception with a special symbol.

Inherited from object: __hash__, __reduce_ex__
Inherited from BaseException: __delattr__, __getattribute__, __getitem__, __getslice__, __reduce__, __setattr__, __setstate__
Inherited from Exception: __new__

32.24.2 Class Variables

Name	Description
__slots__	**Value:** `('loc', 'msg', 'pstr', 'parserElement')` *(type=tuple)*
loc	**Value:** `<member 'loc' of 'ParseBaseException' objects>` *(type=member_descriptor)*
msg	**Value:** `<member 'msg' of 'ParseBaseException' objects>` *(type=member_descriptor)*
parserElement	**Value:** `<member 'parserElement' of 'ParseBaseException'-objects>` *(type=member_descriptor)*
pstr	**Value:** `<member 'pstr' of 'ParseBaseException' objects>` *(type=member_descriptor)*
Inherited from BaseException: args *(p. ??)*, message *(p. ??)*	

32.25 Class ParseElementEnhance

__builtin__.object
matplotlib.pyparsing.ParserElement
ParseElementEnhance

Known Subclasses: FollowedBy, Forward, NotAny, OneOrMore, Optional, SkipTo, TokenConverter, ZeroOrMore

Abstract subclass of ParserElement, for combining and post-processing parsed tokens.

32.25.1 Methods

__init__(*self*, *expr*, *savelist*=`False`)
Overrides: matplotlib.pyparsing.ParserElement.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParserElement.__str__

checkRecursion(*self*, *parseElementList*)
Overrides: matplotlib.pyparsing.ParserElement.checkRecursion

ignore(*self*, *other*)

Define expression to be ignored (e.g., comments) while doing pattern matching; may be called repeatedly, to define multiple comment or other ignorable patterns.

Overrides: matplotlib.pyparsing.ParserElement.ignore extit(inherited documentation)

leaveWhitespace(*self*)

Disables the skipping of whitespace before matching the characters in the ParserElement's defined pattern. This is normally only used internally by the pyparsing module, but may be needed in some whitespace-sensitive grammars.

Overrides: matplotlib.pyparsing.ParserElement.leaveWhitespace extit(inherited documentation)

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

streamline(*self*)
Overrides: matplotlib.pyparsing.ParserElement.streamline

validate(*self*, *validateTrace*=`[]`)

Check defined expressions for valid structure, check for infinite recursive definitions.

Overrides: matplotlib.pyparsing.ParserElement.validate extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.25.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.25.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.26 Class ParseException

__builtin__.object
exceptions.BaseException
exceptions.Exception
matplotlib.pyparsing.ParseBaseException
ParseException

exception thrown when parse expressions don't match class

32.26.1 Methods

Inherited from object: __hash__, __reduce_ex__
Inherited from BaseException: __delattr__, __getattribute__, __getitem__, __getslice__, __reduce__, __setattr__, __setstate__
Inherited from Exception: __new__
Inherited from ParseBaseException: __init__, __getattr__, __repr__, __str__, markInputline

32.26.2 Class Variables

Name	Description
Inherited from BaseException: args *(p. ??)*, message *(p. ??)*	
Inherited from ParseBaseException: __slots__ *(p. 471)*, loc *(p. 471)*, msg *(p. 471)*, parserElement *(p. 471)*, pstr *(p. 471)*	

32.27 Class ParseExpression

__builtin__.object
matplotlib.pyparsing.ParserElement
ParseExpression

Known Subclasses: And, Each, MatchFirst, Or

Abstract subclass of ParserElement, for combining and post-processing parsed tokens.

32.27.1 Methods

__init__(*self*, *exprs*, *savelist*=`False`)
Overrides: matplotlib.pyparsing.ParserElement.__init__

__getitem__(*self*, *i*)

__str__(*self*)
Overrides: matplotlib.pyparsing.ParserElement.__str__

append(*self*, *other*)

ignore(*self*, *other*)

Define expression to be ignored (e.g., comments) while doing pattern matching; may be called repeatedly, to define multiple comment or other ignorable patterns.

Overrides: matplotlib.pyparsing.ParserElement.ignore extit(inherited documentation)

leaveWhitespace(*self*)

Extends leaveWhitespace defined in base class, and also invokes leaveWhitespace on all contained expressions.

Overrides: matplotlib.pyparsing.ParserElement.leaveWhitespace

setResultsName(*self*, *name*, *listAllMatches*=`False`)

Define name for referencing matching tokens as a nested attribute of the returned parse results. NOTE: this returns a *copy* of the original ParseElement object; this is so that the client can define a basic element, such as an integer, and reference it in multiple places with different names.

Overrides: matplotlib.pyparsing.ParserElement.setResultsName extit(inherited documentation)

streamline(*self*)
Overrides: matplotlib.pyparsing.ParserElement.streamline

validate(*self*, *validateTrace*=[])

Check defined expressions for valid structure, check for infinite recursive definitions.

Overrides: matplotlib.pyparsing.ParserElement.validate extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, checkRecursion, copy, parse, parseFile, parseImpl, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.27.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.27.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.28 Class ParseFatalException

__builtin__.object
exceptions.BaseException
exceptions.Exception
matplotlib.pyparsing.ParseBaseException
ParseFatalException

user-throwable exception thrown when inconsistent parse content is found; stops all parsing immediately

32.28.1 Methods

Inherited from object: __hash__, __reduce_ex__
Inherited from BaseException: __delattr__, __getattribute__, __getitem__, __getslice__, __reduce__, __setattr__, __setstate__
Inherited from Exception: __new__
Inherited from ParseBaseException: __init__, __getattr__, __repr__, __str__, markInputline

32.28.2 Class Variables

Name	Description
Inherited from BaseException: args *(p. ??)*, message *(p. ??)*	
Inherited from ParseBaseException: __slots__ *(p. 471)*, loc *(p. 471)*, msg *(p. 471)*, parserElement *(p. 471)*, pstr *(p. 471)*	

32.29 Class ParserElement

__builtin__.object
└ **ParserElement**

Known Subclasses: ParseElementEnhance, ParseExpression, Token

Abstract base level parser element class.

32.29.1 Methods

__init__(*self*, *savelist*=`False`)
Overrides: __builtin__.object.__init__

__add__(*self*, *other*)

Implementation of + operator - returns And

__and__(*self*, *other*)

Implementation of & operator - returns Each

__invert__(*self*)

Implementation of ~ operator - returns NotAny

__or__(*self*, *other*)

Implementation of | operator - returns MatchFirst

__radd__(*self*, *other*)

Implementation of += operator

__rand__(*self*, *other*)

Implementation of right-& operator

__repr__(*self*)
Overrides: __builtin__.object.__repr__

__ror__(*self*, *other*)

Implementation of |= operator

__rxor__(*self*, *other*)

Implementation of ˆ= operator

__str__(*self*)
Overrides: __builtin__.object.__str__

__xor__(*self*, *other*)

Implementation of ˆ operator - returns Or

checkRecursion(*self*, *parseElementList*)

copy(*self*)

Make a copy of this ParseElement. Useful for defining different parse actions for the same parsing pattern, using copies of the original parse element.

ignore(*self*, *other*)

Define expression to be ignored (e.g., comments) while doing pattern matching; may be called repeatedly, to define multiple comment or other ignorable patterns.

leaveWhitespace(*self*)

Disables the skipping of whitespace before matching the characters in the ParserElement's defined pattern. This is normally only used internally by the pyparsing module, but may be needed in some whitespace-sensitive grammars.

parse(*self*, *instring*, *loc*, *doActions*=`True`, *callPreParse*=`True`)

parseFile(*self*, *file_or_filename*)

Execute the parse expression on the given file or filename. If a filename is specified (instead of a file object), the entire file is opened, read, and closed before parsing.

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)

parseString(*self*, *instring*)

Execute the parse expression with the given string. This is the main interface to the client code, once the complete expression has been built.

parseWithTabs(*self*)

Overrides default behavior to expand <TAB>s to spaces before parsing the input string. Must be called before parseString when the input grammar contains elements that match <TAB> characters.

postParse(*self*, *instring*, *loc*, *tokenlist*)

preParse(*self*, *instring*, *loc*)

scanString(*self*, *instring*)

Scan the input string for expression matches. Each match will return the matching tokens, start location, and end location.

setDebug(*self*, *flag*=`True`)

Enable display of debugging messages while doing pattern matching.

setDebugActions(*self*, *startAction*, *successAction*, *exceptionAction*)

Enable display of debugging messages while doing pattern matching.

setName(*self*, *name*)

Define name for this expression, for use in debugging.

setParseAction(*self*, *fn*)

Define action to perform when successfully matching parse element definition. Parse action fn is a callable method with the arguments (s, loc, toks) where:

- s = the original string being parsed
- loc = the location of the matching substring
- toks = a list of the matched tokens, packaged as a ParseResults object

If the function fn modifies the tokens, it can return them as the return value from fn, and the modified list of tokens will replace the original. Otherwise, fn does not need to return any value.

setResultsName(*self*, *name*, *listAllMatches*=`False`)

Define name for referencing matching tokens as a nested attribute of the returned parse results. NOTE: this returns a *copy* of the original ParseElement object; this is so that the client can define a basic element, such as an integer, and reference it in multiple places with different names.

setWhitespaceChars(*self*, *chars*)

Overrides the default whitespace chars

skipIgnorables(*self*, *instring*, *loc*)

streamline(*self*)

suppress(*self*)

Suppresses the output of this ParseElement; useful to keep punctuation from cluttering up returned output.

transformString(*self*, *instring*)

Extension to scanString, to modify matching text with modified tokens that may be returned from a parse action. To use transformString, define a grammar and attach a parse action to it that modifies the returned token list. Invoking transformString() on a target string will then scan for matches, and replace the matched text patterns according to the logic in the parse action. transformString() returns the resulting transformed string.

tryParse(*self*, *instring*, *loc*)

validate(*self*, *validateTrace*=`[]`)

Check defined expressions for valid structure, check for infinite recursive definitions.

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__

32.29.2 Static Methods

setDefaultWhitespaceChars(*chars*)

Overrides the default whitespace chars

32.29.3 Class Variables

Name	Description
DEFAULT_WHITE_CHARS	**Value:** `' \n\t\r'` *(type=`str`)*

32.30 Class ParseResults

__builtin__.object
└─ **ParseResults**

Structured parse results, to provide multiple means of access to the parsed data:

- as a list (len(results))
- by list index (results[0], results[1], etc.)
- by attribute (results.<resultsName>)

32.30.1 Methods

__init__(*self*, *toklist*, *name*=`None`, *asList*=`True`, *modal*=`True`)
Overrides: __builtin__.object.__init__

__contains__(*self*, *k*)

__delitem__(*self*, *i*)

__getattr__(*self*, *name*)

__getitem__(*self*, *i*)

__iadd__(*self*, *other*)

__iter__(*self*)

__len__(*self*)

__repr__(*self*)
Overrides: __builtin__.object.__repr__

__setitem__(*self*, *k*, *v*)

__str__(*self*)
Overrides: __builtin__.object.__str__

asDict(*self*)

Returns the named parse results as dictionary.

asList(*self*)

Returns the parse results as a nested list of matching tokens, all converted to strings.

asXML(*self*, *doctag*=`None`, *namedItemsOnly*=`False`, *indent*=`''`, *formatted*=`True`)

Returns the parse results as XML. Tags are created for tokens and lists that have defined results names.

copy(*self*)

Returns a new copy of a ParseResults object.

getName(*self*)

Returns the results name for this token expression.

items(*self*)

Returns all named result keys and values as a list of tuples.

keys(*self*)

Returns all named result keys.

values(*self*)

Returns all named result values.

Inherited from object: __delattr__, __getattribute__, __hash__, __reduce__, __reduce_ex__, __setattr__

32.30.2 Static Methods

__new__(*cls*, *toklist*, *name*=`None`, *asList*=`True`, *modal*=`True`)
Overrides: __builtin__.object.__new__

32.30.3 Class Variables

Name	Description
__slots__	**Value:** `('__toklist', '__tokdict', '__doinit', '__name', -'__parent', '__modal')` *(type=***tuple***)*

32.31 Class PositionToken

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
PositionToken

Known Subclasses: GoToColumn, LineEnd, LineStart, StringEnd, StringStart

32.31.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.Token.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__,

__rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseImpl, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.31.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.31.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.32 Class RecursiveGrammarException

__builtin__.object
exceptions.BaseException
exceptions.Exception
RecursiveGrammarException

exception thrown by validate() if the grammar could be improperly recursive

32.32.1 Methods

__init__(*self*, *parseElementList*)
Overrides: exceptions.Exception.__init__

__str__(*self*)
Overrides: exceptions.BaseException.__str__

Inherited from object: __hash__, __reduce_ex__
Inherited from BaseException: __delattr__, __getattribute__, __getitem__, __getslice__, __reduce__, __repr__, __setattr__, __setstate__
Inherited from Exception: __new__

32.32.2 Class Variables

Name	Description
Inherited from BaseException: args *(p. ??)*, message *(p. ??)*	

32.33 Class Regex

```
__builtin__.object
    |
    matplotlib.pyparsing.ParserElement
        |
        matplotlib.pyparsing.Token
            |
            Regex
```

Token for matching strings that match a given regular expression. Defined with string specifying the regular expression in a form recognized by the inbuilt Python re module.

32.33.1 Methods

__init__(*self*, *pattern*, *flags*=`0`)

The parameters pattern and flags are passed to the re.compile() function as-is. See the Python re module for an explanation of the acceptable patterns and flags.

Overrides: matplotlib.pyparsing.Token.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParserElement.__str__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.33.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.33.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.34 Class SkipTo

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
SkipTo

Token for skipping over all undefined text until the matched expression is found. If include is set to true, the matched expression is also consumed. The ignore argument is used to define grammars (typically quoted strings and comments) that might contain false matches.

32.34.1 Methods

__init__(*self*, *other*, *include*=`False`, *ignore*=`None`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, ignore, leaveWhitespace, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.34.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.34.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.35 Class StringEnd

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
matplotlib.pyparsing.PositionToken
StringEnd

Matches if current position is at the end of the parse string

32.35.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.PositionToken.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.35.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.35.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.36 Class StringStart

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
matplotlib.pyparsing.PositionToken
StringStart

Matches if current position is at the beginning of the parse string

32.36.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.PositionToken.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.36.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.36.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.37 Class Suppress

```
__builtin__.object
   matplotlib.pyparsing.ParserElement
      matplotlib.pyparsing.ParseElementEnhance
         matplotlib.pyparsing.TokenConverter
            Suppress
```

Converter for ignoring the results of a parsed expression.

32.37.1 Methods

postParse(*self*, *instring*, *loc*, *tokenlist*)
Overrides: matplotlib.pyparsing.ParserElement.postParse

suppress(*self*)

Suppresses the output of this ParseElement; useful to keep punctuation from cluttering up returned output.

Overrides: matplotlib.pyparsing.ParserElement.suppress extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, ignore, leaveWhitespace, parseImpl, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, transformString, tryParse
Inherited from TokenConverter: __init__

32.37.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.37.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.38 Class Token

```
__builtin__.object ─┐
matplotlib.pyparsing.ParserElement ─┐
                                    Token
```

Known Subclasses: CharsNotIn, Empty, Keyword, Literal, NoMatch, PositionToken, Regex, White, Word

Abstract ParserElement subclass, for defining atomic matching patterns.

32.38.1 Methods

__init__(*self*)
Overrides: matplotlib.pyparsing.ParserElement.__init__

setName(*self*, *name*)

Define name for this expression, for use in debugging.

Overrides: matplotlib.pyparsing.ParserElement.setName extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseImpl, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate

32.38.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.38.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.39 Class TokenConverter

```
__builtin__.object ─┐
matplotlib.pyparsing.ParserElement ─┐
matplotlib.pyparsing.ParseElementEnhance ─┐
                                          TokenConverter
```

Known Subclasses: Combine, Dict, Group, Suppress, Upcase

Abstract subclass of ParseExpression, for converting parsed results.

32.39.1 Methods

__init__(*self*, *expr*, *savelist*=`False`) Overrides: matplotlib.pyparsing.ParseElementEnhance.__init__

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, ignore, leaveWhitespace, parseImpl, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.39.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.39.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.40 Class Upcase

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.ParseElementEnhance
matplotlib.pyparsing.TokenConverter
Upcase

Converter to upper case all matching tokens.

32.40.1 Methods

__init__(*self*, **args*) Overrides: matplotlib.pyparsing.TokenConverter.__init__

postParse(*self*, *instring*, *loc*, *tokenlist*)
Overrides: matplotlib.pyparsing.ParserElement.postParse

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: __str__, checkRecursion, ignore, leaveWhitespace, parseImpl, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.40.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.40.3 Class Variables

Name	Description
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.41 Class White

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
White

Special matching class for matching whitespace. Normally, whitespace is ignored by pyparsing grammars. This class is included when some whitespace structures are significant. Define with a string containing the whitespace characters to be matched; default is " \t\n". Also takes optional min, max, and exact arguments, as defined for the Word class.

32.41.1 Methods

__init__(*self*, *ws*=`' \t\r\n'`, *min*=`1`, *max*=`0`, *exact*=`0`)
Overrides: matplotlib.pyparsing.Token.__init__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__,

__rxor__, __str__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.41.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.41.3 Class Variables

Name	Description
whiteStrs	**Value:** `{'\t': '<TAB>', ' ': '<SPC>', '\n': '<LF>', '\r': '<CR>', '\x0c': '<FF>'}` *(type=`dict`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.42 Class Word

__builtin__.object
matplotlib.pyparsing.ParserElement
matplotlib.pyparsing.Token
Word

Token for matching words composed of allowed character sets. Defined with string containing all allowed initial characters, an optional string containing allowed body characters (if omitted, defaults to the initial character set), and an optional minimum, maximum, and/or exact length.

32.42.1 Methods

__init__(*self*, *initChars*, *bodyChars*=`None`, *min*=`1`, *max*=`0`, *exact*=`0`)
Overrides: matplotlib.pyparsing.Token.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParserElement.__str__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParserElement.parseImpl

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__,

__rxor__, __xor__, checkRecursion, copy, ignore, leaveWhitespace, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setParseAction, setResultsName, setWhitespaceChars, skipIgnorables, streamline, suppress, transformString, tryParse, validate
Inherited from Token: setName

32.42.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.42.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=list)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

32.43 Class ZeroOrMore

```
__builtin__.object
  matplotlib.pyparsing.ParserElement
    matplotlib.pyparsing.ParseElementEnhance
      ZeroOrMore
```

Optional repetition of zero or more of the given expression.

32.43.1 Methods

__init__(*self*, *expr*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__init__

__str__(*self*)
Overrides: matplotlib.pyparsing.ParseElementEnhance.__str__

parseImpl(*self*, *instring*, *loc*, *doActions*=`True`)
Overrides: matplotlib.pyparsing.ParseElementEnhance.parseImpl

setResultsName(*self*, *name*, *listAllMatches*=`False`)

Define name for referencing matching tokens as a nested attribute of the returned parse results. NOTE: this returns a *copy* of the original ParseElement object; this is so that the client can define a basic element, such as an integer, and reference it in multiple places with different names.

Overrides: matplotlib.pyparsing.ParserElement.setResultsName extit(inherited documentation)

Inherited from object: __delattr__, __getattribute__, __hash__, __new__, __reduce__, __reduce_ex__, __setattr__
Inherited from ParseElementEnhance: checkRecursion, ignore, leaveWhitespace, streamline, validate
Inherited from ParserElement: __add__, __and__, __invert__, __or__, __radd__, __rand__, __repr__, __ror__, __rxor__, __xor__, copy, parse, parseFile, parseString, parseWithTabs, postParse, preParse, scanString, setDebug, setDebugActions, setName, setParseAction, setWhitespaceChars, skipIgnorables, suppress, transformString, tryParse

32.43.2 Static Methods

Inherited from ParserElement: setDefaultWhitespaceChars

32.43.3 Class Variables

Name	Description
__slotnames__	**Value:** `[]` *(type=`list`)*
Inherited from ParserElement: DEFAULT_WHITE_CHARS *(p. 477)*	

33 Module matplotlib.quiver

Support for plotting fields of arrows.

Presently this contains a single class, Quiver, but it might make sense to consolidate other arrow plotting here.

This will also become a home for things such as standard deviation ellipses, which can and will be derived very easily from the Quiver code.

33.1 Class Quiver

matplotlib.artist.Artist

matplotlib.collections.Collection

matplotlib.cm.ScalarMappable

matplotlib.collections.PatchCollection

matplotlib.collections.PolyCollection

Quiver

Specialized PolyCollection for arrows.

The only API method is set_UVC(), which can be used to change the size, orientation, and color of the arrows; their locations are fixed when the class is instantiated. Possibly this method will be useful in animations.

Much of the work in this class is done in the draw() method so that as much information as possible is available about the plot. In subsequent draw() calls, recalculation is limited to things that might have changed, so there should be no performance penalty from putting the calculations in the draw() method.

33.1.1 Methods

__init__(*self, ax, *args, **kw*)

```
The constructor takes one required argument, an Axes
        instance, followed by the args and kwargs described
        by the following pylab interface documentation:

Plot a 2-D field of arrows.

Function signatures:

    quiver(U, V, **kw)
    quiver(U, V, C, **kw)
    quiver(X, Y, U, V, **kw)
    quiver(X, Y, U, V, C, **kw)

Arguments:

    X, Y give the x and y coordinates of the arrow locations
        (default is tail of arrow; see 'pivot' kwarg)
    U, V give the x and y components of the arrow vectors
    C is an optional array used to map colors to the arrows

    All arguments may be 1-D or 2-D arrays or sequences.
    If X and Y are absent, they will be generated as a uniform grid.
    If U and V are 2-D arrays but X and Y are 1-D, and if
        len(X) and len(Y) match the column and row dimensions
        of U, then X and Y will be expanded with meshgrid.

Keyword arguments (default given first):

  * units = 'width' | 'height' | 'dots' | 'inches' | 'x' | 'y'
            arrow units; the arrow dimensions *except for length*
            are in multiples of this unit.
  * scale = None | float
            data units per arrow unit, e.g. m/s per plot width;
            a smaller scale parameter makes the arrow longer.
            If None, a simple autoscaling algorithm is used, based
            on the average vector length and the number of vectors.

    Arrow dimensions and scales can be in any of several units:

    'width' or 'height': the width or height of the axes
    'dots' or 'inches':  pixels or inches, based on the figure dpi
    'x' or 'y': X or Y data units

    In all cases the arrow aspect ratio is 1, so that if U==V the angle
    of the arrow on the plot is 45 degrees CCW from the X-axis.

    The arrows scale differently depending on the units, however.
    For 'x' or 'y', the arrows get larger as one zooms in; for other
    units, the arrow size is independent of the zoom state.  For
    'width or 'height', the arrow size increases with the width and
    height of the axes, respectively, when the the window is resized;
    for 'dots' or 'inches', resizing does not change the arrows.
```

draw(*self*, *renderer*)
Overrides: matplotlib.collections.PolyCollection.draw

set_UVC(*self*, *U*, *V*, *C*=None)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from
Inherited from ScalarMappable: add_observer, autoscale, autoscale_None, changed, get_array, get_clim, notify, set_array, set_clim, set_cmap, set_colorbar, set_norm, to_rgba
Inherited from PatchCollection: get_transformed_patches, get_transoffset, pick, set_alpha, set_color, set_edgecolor, set_facecolor, set_linewidth, update_scalarmappable
Inherited from PolyCollection: get_verts, set_verts

33.1.2 Class Variables

Name	Description
quiver_doc	**Value:** `"\nPlot a 2-D field of arrows.\n\nFunction signatures:\n\n    quiver(U, V, **...` *(type=str)*
Inherited from Artist: aname *(p. 92)*	
Inherited from PatchCollection: zorder *(p. 230)*	

33.2 Class QuiverKey

matplotlib.artist.Artist
└ **QuiverKey**

Labelled arrow for use as a quiver plot scale key.

33.2.1 Methods

__init__(*self*, *Q*, *X*, *Y*, *U*, *label*, ***kw*)

```
Add a key to a quiver plot.

Function signature:
    quiverkey(Q, X, Y, U, label, **kw)

Arguments:
    Q is the Quiver instance returned by a call to quiver.
    X, Y give the location of the key; additional explanation follows.
    U is the length of the key
    label is a string with the length and units of the key

Keyword arguments (default given first):
  * coordinates = 'axes' | 'figure' | 'data' | 'inches'
        Coordinate system and units for X, Y: 'axes' and 'figure'
        are normalized coordinate systems with 0,0 in the lower
        left and 1,1 in the upper right; 'data' are the axes
        data coordinates (used for the locations of the vectors
        in the quiver plot itself); 'inches' is position in the
        figure in inches, with 0,0 at the lower left corner.
  * color overrides face and edge colors from Q.
  * labelpos = 'N' | 'S' | 'E' | 'W'
        Position the label above, below, to the right, to the left
        of the arrow, respectively.
  * labelsep = 0.1 inches distance between the arrow and the label
  * labelcolor (defaults to default Text color)
  * fontproperties is a dictionary with keyword arguments accepted
        by the FontProperties initializer: family, style, variant,
        size, weight

    Any additional keyword arguments are used to override vector
    properties taken from Q.

    The positioning of the key depends on X, Y, coordinates, and
    labelpos.  If labelpos is 'N' or 'S', X,Y give the position
    of the middle of the key arrow.  If labelpos is 'E', X,Y
    positions the head, and if labelpos is 'W', X,Y positions the
    tail; in either of these two cases, X,Y is somewhere in the middle
    of the arrow+label key object.
```

Overrides: matplotlib.artist.Artist.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pick, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

33.2.2 Class Variables

Name	Description
halign	**Value:** `{'S': 'center', 'E': 'left', 'W': 'right', 'N':-'center'}` *(type=`dict`)*
pivot	**Value:** `{'S': 'mid', 'E': 'tip', 'W': 'tail', 'N': 'mid'}` *(type=`dict`)*
quiverkey_doc	**Value:** `"\nAdd a key to a quiver plot.\n\nFunction sign-ature:\n    quiverkey(Q, X, Y,...` *(type=`str`)*
valign	**Value:** `{'S': 'top', 'E': 'center', 'W': 'center', 'N':-'bottom'}` *(type=`dict`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

34 Module matplotlib.table

Place a table below the x-axis at location loc.

The table consists of a grid of cells.

The grid need not be rectangular and can have holes.

Cells are added by specifying their row and column.

For the purposes of positioning the cell at (0, 0) is assumed to be at the top left and the cell at (max_row, max_col) is assumed to be at bottom right.

You can add additional cells outside this range to have convenient ways of positioning more interesting grids.

Author : John Gill <jng@europe.renre.com> Copyright : 2004 John Gill and John Hunter License : matplotlib license

34.1 Functions

table(*ax*, *cellText*=None, *cellColours*=None, *cellLoc*='right', *colWidths*=None, *rowLabels*=None, *rowColours*=None, *rowLoc*='left', *colLabels*=None, *colColours*=None, *colLoc*='center', *loc*='bottom', *bbox*=None)

```
TABLE(cellText=None, cellColours=None,
      cellLoc='right', colWidths=None,
      rowLabels=None, rowColours=None, rowLoc='left',
      colLabels=None, colColours=None, colLoc='center',
      loc='bottom', bbox=None)

Factory function to generate a Table instance.

Thanks to John Gill for providing the class and table.
```

34.2 Class Cell

matplotlib.artist.Artist

matplotlib.patches.Patch

matplotlib.patches.Rectangle

Cell

A cell is a Rectangle with some associated text.

34.2.1 Methods

__init__(*self*, *xy*, *width*, *height*, *edgecolor*=`'k'`, *facecolor*=`'w'`, *fill*=`True`, *text*=`''`, *loc*=`None`)
Overrides: matplotlib.patches.Rectangle.__init__

auto_set_font_size(*self*, *renderer*)

Shrink font size until text fits.

draw(*self*, *renderer*)
Overrides: matplotlib.patches.Patch.draw

get_fontsize(*self*)

Return the cell fontsize

get_required_width(*self*, *renderer*)

Get width required for this cell.

get_text(*self*)

Return the cell Text intance

get_text_bounds(*self*, *renderer*)

Get text bounds in axes co-ordinates.

set_figure(*self*, *fig*)

Set the figure instance the artist belong to
ACCEPTS: a matplotlib.figure.Figure instance

Overrides: matplotlib.artist.Artist.set_figure extit(inherited documentation)

set_fontsize(*self*, *size*)

set_text_props(*self*, ***kwargs*)

update the text properties with kwargs

set_transform(*self*, *trans*)
Overrides: matplotlib.artist.Artist.set_transform

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_label, set_lod, set_picker, set_visible, set_zorder, update
Inherited from Patch: get_aa, get_antialiased, get_ec, get_edgecolor, get_facecolor, get_fc, get_fill, get_hatch,

get_linewidth, get_lw, get_window_extent, pick, set_antialiased, set_ec, set_edgecolor, set_facecolor, set_fc, set_fill, set_hatch, set_linewidth, set_lw, update_from
Inherited from Rectangle: get_height, get_verts, get_width, get_x, get_y, set_bounds, set_height, set_width, set_x, set_y

34.2.2 Class Variables

Name	Description
PAD	**Value:** `0.10000000000000001` *(type=`float`)*
Inherited from Artist: aname *(p. 92)*	
Inherited from Patch: zorder *(p. 360)*	

34.3 Class Table

matplotlib.artist.Artist
└ **Table**

Create a table of cells.

Table can have (optional) row and column headers.

Each entry in the table can be either text or patches.

Column widths and row heights for the table can be specified.

Return value is a sequence of text, line and patch instances that make up the table

34.3.1 Methods

__init__(*self*, *ax*, *loc*=`None`, *bbox*=`None`)
Overrides: matplotlib.artist.Artist.__init__

add_cell(*self*, *row*, *col*, **args*, ***kwargs*)

Add a cell to the table.

auto_set_column_width(*self*, *col*)

auto_set_font_size(*self*, *value*=`True`)

Automatically set font size.

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_celld(*self*)

return a dict of cells in the table

get_child_artists(*self*)

Return the Artists cintained by the table

get_window_extent(*self*, *renderer*)

Return the bounding box of the table in window coords

scale(*self*, *xscale*, *yscale*)

Scale column widths by xscale and row heights by yscale.

set_fontsize(*self*, *size*)

Set the fontsize of the cell text
ACCEPTS: a float in points

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pick, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update, update_from

34.3.2 Class Variables

Name	Description
AXESPAD	**Value:** `0.02` *(type=`float`)*
codes	**Value:** `{'right': 14, 'center': 9, 'bottom': 17, 'lower-left': 3, 'center right': 6, ...` *(type=`dict`)*
FONTSIZE	**Value:** `10` *(type=`int`)*
Inherited from Artist: aname *(p. 92)*, zorder *(p. 92)*	

35 Module matplotlib.texmanager

```
This module supports embedded TeX expressions in matplotlib via dvipng
and dvips for the raster and postscript backends.  The tex and
dvipng/dvips information is cached in ~/.matplotlib/tex.cache for reuse between
sessions

Requirements:

  tex

  *Agg backends: dvipng

  PS backend: latex w/ psfrag, dvips, and Ghostscript 8.51
  (older versions do not work properly)

Backends:

  Only supported on *Agg and PS backends currently


For raster output, you can get RGBA numerix arrays from TeX expressions
as follows

  texmanager = TexManager()
  s = r'\TeX\ is Number $\displaystyle\sum_{n=1}^\infty
rac{-e^{i\pi}}{2^n}$!'
  Z = self.texmanager.get_rgba(s, size=12, dpi=80, rgb=(1,0,0))

To enable tex rendering of all text in your matplotlib figure, set
text.usetex in your matplotlibrc file (http://matplotlib.sf.net/matplotlibrc)
or include these two lines in your script:
from matplotlib import rc
rc('text', usetex=True)
```

35.1 Functions

get_dvipng_version()

35.2 Class TexManager

Convert strings to dvi files using TeX, caching the results to a working dir

35.2.1 Methods

__init__(*self*)

get_basefile(*self*, *tex*, *fontsize*, *dpi*=None)

get_font_config(*self*)

get_font_preamble(*self*)

get_ps_bbox(*self*, *tex*, *fontsize*)

get_rgba(*self*, *tex*, *fontsize*=None, *dpi*=None, *rgb*=(0, 0, 0))

Return tex string as an rgba array

get_shell_cmd(*self*, **args*)

On windows, changing directories can be complicated by the presence of multiple drives. get_shell_cmd deals with this issue.

make_dvi(*self*, *tex*, *fontsize*, *force*=0)

make_png(*self*, *tex*, *fontsize*, *dpi*, *force*=0)

make_ps(*self*, *tex*, *fontsize*, *force*=0)

make_tex(*self*, *tex*, *fontsize*)

35.2.2 Class Variables

Name	Description
arrayd	**Value:** {} *(type=dict)*
configdir	**Value:** '/home/jdhunter/.matplotlib' *(type=str)*
cursive	**Value:** ('pzc', '\\usepackage{chancery}') *(type=tuple)*
dvipngVersion	**Value:** '1.9' *(type=str)*
font_family	**Value:** 'serif' *(type=str)*
font_info	**Value:** {'computer modern sans serif': ('cmss', ''), 'b-ookman': ('pbk', '\\renewcomma... *(type=dict)*
monospace	**Value:** ('cmtt', '') *(type=tuple)*
oldcache	**Value:** '/home/jdhunter/.tex.cache' *(type=str)*
oldpath	**Value:** '/home/jdhunter' *(type=str)*
postscriptd	**Value:** {} *(type=dict)*
pscnt	**Value:** 0 *(type=int)*

continued on next page

Name	Description
sans_serif	**Value:** `('cmss', '')` *(type=`tuple`)*
serif	**Value:** `('cmr', '')` *(type=`tuple`)*
texcache	**Value:** `'/home/jdhunter/.matplotlib/tex.cache'` *(type=`str`)*

36 Module matplotlib.text

Figure and Axes text

36.1 Functions

get_rotation(*rotation*)

return the text angle as float

scanner(*s*)

Split a string into mathtext and non-mathtext parts. mathtext is surrounded by $ symbols. quoted \$ are ignored
All slash quotes dollar signs are ignored
The number of unquoted dollar signs must be even
Return value is a list of (substring, inmath) tuples

36.2 Class Annotation

matplotlib.artist.Artist
matplotlib.text.Text
Annotation

A Text class to make annotating things in the figure: Figure, Axes, Point, Rectangle, etc... easier

36.2.1 Methods

__init__(*self*, *s*, *xy*, *xycoords*='data', *xytext*=None, *textcoords*=None, *arrowprops*=None, ***kwargs*)

```
Annotate the x,y point xy with text s at x,y location xytext
(xytext if None defaults to xy and textcoords if None defaults
to xycoords).
arrowprops, if not None, is a dictionary of line properties
(see matplotlib.lines.Line2D) for the arrow that connects
annotation to the point.   Valid keys are
  - width : the width of the arrow in points
  - frac  : the fraction of the arrow length occupied by the head
  - headwidth : the width of the base of the arrow head in points
  - shrink: often times it is convenient to have the arrowtip
    and base a bit away from the text and point being
    annotated.  If d is the distance between the text and
    annotated point, shrink will shorten the arrow so the tip
    and base are shink percent of the distance d away from the
    endpoints.  ie, shrink=0.05 is 5%
  - any key for matplotlib.patches.polygon
xycoords and textcoords are a string that indicates the
coordinates of xy and xytext.
   'figure points'   : points from the lower left corner of the figure
   'figure pixels'   : pixels from the lower left corner of the figure
   'axes points'     : points from lower left corner of axes
   'axes pixels'     : pixels from lower left corner of axes
   'axes fraction'   : 0,1 is lower left of axes and 1,1 is upper right
   'data'            : use the coordinate system of the object being annotated (default)
   'polar'           : you can specify theta, r for the annotation, even
                       in cartesian plots.  Note that if you
                       are using a polar axes, you do not need
                       to specify polar for the coordinate
                       system since that is the native"data" coordinate system.
If a points or pixels option is specified, values will be
added to the left, bottom and if negative, values will be
subtracted from the top, right.  Eg,
  # 10 points to the right of the left border of the axes and
  # 5 points below the top border
  xy=(10,-5), xycoords='axes points'
Additional kwargs are Text properties:
              alpha: float
            animated: [True | False]
            backgroundcolor: any matplotlib color
            bbox: rectangle prop dict plus key 'pad' which is a pad in points
            clip_box: a matplotlib.transform.Bbox instance
            clip_on: [True | False]
            color: any matplotlib color
            family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
            figure: a matplotlib.figure.Figure instance
            fontproperties: a matplotlib.font_manager.FontProperties instance
            horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
            label: any string
            lod: [True | False]
            multialignment: ['left' | 'right' | 'center' ]
            name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
            position: (x,y)
            rotation: [ angle in degrees 'vertical' | 'horizontal'
```

draw(*self*, *renderer*)
Overrides: matplotlib.text.Text.draw

set_clip_box(*self*, *clipbox*)

Set the artist's clip Bbox
ACCEPTS: a matplotlib.transform.Bbox instance

Overrides: matplotlib.artist.Artist.set_clip_box

update_positions(*self*, *renderer*)

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update
Inherited from Text: get_color, get_font_properties, get_fontname, get_fontsize, get_fontstyle, get_fontweight, get_ha, get_horizontalalignment, get_name, get_position, get_prop_tup, get_rotation, get_rotation_matrix, get_size, get_style, get_text, get_va, get_verticalalignment, get_weight, get_window_extent, is_math_text, pick, set_backgroundcolor, set_bbox, set_color, set_family, set_fontname, set_fontproperties, set_fontsize, set_fontstyle, set_fontweight, set_ha, set_horizontalalignment, set_ma, set_multialignment, set_name, set_position, set_rotation, set_size, set_style, set_text, set_va, set_variant, set_verticalalignment, set_weight, set_x, set_y, update_from

36.2.2 Class Variables

Name	Description
Inherited from Artist: aname *(p. 92)*	
Inherited from Text: zorder *(p. 511)*	

36.3 Class Text

matplotlib.artist.Artist
    Text

Known Subclasses: Annotation, oText3D, TextWithDash

Handle storing and drawing of text in window or data coordinates

36.3.1 Methods

__init__(*self*, *x*=0, *y*=0, *text*='', *color*=None, *verticalalignment*='bottom', *horizontalalignment*='left', *multialignment*=None, *fontproperties*=None, *rotation*=None, ***kwargs*)

```
Create a Text instance at x,y with string text.  Valid kwargs are
    alpha: float
animated: [True | False]
backgroundcolor: any matplotlib color
bbox: rectangle prop dict plus key 'pad' which is a pad in points
clip_box: a matplotlib.transform.Bbox instance
clip_on: [True | False]
color: any matplotlib color
family: [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
figure: a matplotlib.figure.Figure instance
fontproperties: a matplotlib.font_manager.FontProperties instance
horizontalalignment or ha: [ 'center' | 'right' | 'left' ]
label: any string
lod: [True | False]
multialignment: ['left' | 'right' | 'center' ]
name or fontname: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]
position: (x,y)
rotation: [ angle in degrees 'vertical' | 'horizontal'
size or fontsize: [ size in points | relative size eg 'smaller', 'x-large' ]
style or fontstyle: [ 'normal' | 'italic' | 'oblique']
text: string
transform: a matplotlib.transform transformation instance
variant: [ 'normal' | 'small-caps' ]
verticalalignment or va: [ 'center' | 'top' | 'bottom' ]
visible: [True | False]
weight or fontweight: [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']
x: float
y: float
zorder: any number
```

Overrides: matplotlib.artist.Artist.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.artist.Artist.draw

get_color(*self*)

Return the color of the text

get_font_properties(*self*)

Return the font object

get_fontname(*self*)

alias for get_name

get_fontsize(*self*)

alias for get_size

get_fontstyle(*self*)

alias for get_style

get_fontweight(*self*)

alias for get_weight

get_ha(*self*)

alias for get_horizontalalignment

get_horizontalalignment(*self*)

Return the horizontal alignment as string

get_name(*self*)

Return the font name as string

get_position(*self*)

Return x, y as tuple

get_prop_tup(*self*)

Return a hashable tuple of properties
Not intended to be human readable, but useful for backends who want to cache derived information about text (eg layouts) and need to know if the text has changed

get_rotation(*self*)

return the text angle as float

get_rotation_matrix(*self*, *x0*, *y0*)

get_size(*self*)

Return the font size as integer

get_style(*self*)

Return the font style as string

get_text(*self*)

Get the text as string

get_va(*self*)

alias for getverticalalignment

get_verticalalignment(*self*)

Return the vertical alignment as string

get_weight(*self*)

Get the font weight as string

get_window_extent(*self*, *renderer*=None)

is_math_text(*self*)

pick(*self*, *mouseevent*)

if the mouse click is inside the vertices defining the bounding box of the text, fire off a backend_bases.PickEvent

Overrides: matplotlib.artist.Artist.pick

set_backgroundcolor(*self*, *color*)

Set the background color of the text by updating the bbox (see set_bbox for more info)
ACCEPTS: any matplotlib color

set_bbox(*self*, *rectprops*)

```
Draw a bounding box around self.  rect props are any settable
properties for a rectangle, eg facecolor='red', alpha=0.5.

  t.set_bbox(dict(facecolor='red', alpha=0.5))

ACCEPTS: rectangle prop dict plus key 'pad' which is a pad in points
```

set_color(*self*, *color*)

Set the foreground color of the text
ACCEPTS: any matplotlib color

set_family(*self*, *fontname*)

Set the font family
ACCEPTS: ['serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace']

set_fontname(*self*, *fontname*)

alias for set_name

set_fontproperties(*self*, *fp*)

Set the font properties that control the text
ACCEPTS: a matplotlib.font_manager.FontProperties instance

set_fontsize(*self*, *fontsize*)

alias for set_size

set_fontstyle(*self*, *fontstyle*)

alias for set_style

set_fontweight(*self*, *weight*)

alias for set_weight

set_ha(*self*, *align*)

alias for set_horizontalalignment

set_horizontalalignment(*self*, *align*)

Set the horizontal alignment to one of
ACCEPTS: ['center' | 'right' | 'left']

set_ma(*self*, *align*)

alias for set_verticalalignment

set_multialignment(*self*, *align*)

Set the alignment for multiple lines layout. The layout of the bounding box of all the lines is determined bu the horizontalalignment and verticalalignment properties, but the multiline text within that box can be
ACCEPTS: ['left' | 'right' | 'center']

set_name(*self*, *fontname*)

Set the font name,
ACCEPTS: string eg, ['Sans' | 'Courier' | 'Helvetica' ...]

set_position(*self*, *xy*)

Set the xy position of the text
ACCEPTS: (x,y)

set_rotation(*self*, *s*)

Set the rotation of the text
ACCEPTS: [angle in degrees 'vertical' | 'horizontal'

set_size(*self*, *fontsize*)

Set the font size, eg, 8, 10, 12, 14...
ACCEPTS: [size in points | relative size eg 'smaller', 'x-large']

set_style(*self*, *fontstyle*)

Set the font style
ACCEPTS: ['normal' | 'italic' | 'oblique']

set_text(*self*, *s*)

Set the text string s
ACCEPTS: string or anything printable with '%s' conversion

set_va(*self*, *align*)

alias for set_verticalalignment

set_variant(*self*, *variant*)

Set the font variant, eg,
ACCEPTS: ['normal' | 'small-caps']

set_verticalalignment(*self*, *align*)

Set the vertical alignment
ACCEPTS: ['center' | 'top' | 'bottom']

set_weight(*self*, *weight*)

Set the font weight
ACCEPTS: ['normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight']

set_x(*self*, *x*)

Set the x position of the text
ACCEPTS: float

set_y(*self*, *y*)

Set the y position of the text
ACCEPTS: float

update_from(*self*, *other*)

Copy properties from other to self

Overrides: matplotlib.artist.Artist.update_from

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_figure, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_figure, set_label, set_lod, set_picker, set_transform, set_visible, set_zorder, update

36.3.2 Class Variables

Name	Description
zorder	**Value:** `3` *(type=`int`)*
Inherited from Artist: aname *(p. 92)*	

36.4 Class TextWithDash

```
matplotlib.artist.Artist
        |
        matplotlib.text.Text
                |
                TextWithDash
```

This is basically a Text with a dash (drawn with a Line2D) before/after it. It is intended to be a drop-in replacement for Text, and should behave identically to Text when dashlength=0.0.

The dash always comes between the point specified by set_position() and the text. When a dash exists, the text alignment arguments (horizontalalignment, verticalalignment) are ignored.

dashlength is the length of the dash in canvas units. (default=0.0).

dashdirection is one of 0 or 1, where 0 draws the dash after the text and 1 before. (default=0).

dashrotation specifies the rotation of the dash, and should generally stay None. In this case self.get_dashrotation() returns self.get_rotation(). (I.e., the dash takes its rotation from the text's rotation). Because the text center is projected onto the dash, major deviations in the rotation cause what may be considered visually unappealing results. (default=None).

dashpad is a padding length to add (or subtract) space between the text and the dash, in canvas units. (default=3).

dashpush "pushes" the dash and text away from the point specified by set_position() by the amount in canvas units. (default=0)

NOTE: The alignment of the two objects is based on the bbox of the Text, as obtained by get_window_extent(). This, in turn, appears to depend on the font metrics as given by the rendering backend. Hence the quality of the "centering" of the label text with respect to the dash varies depending on the backend used.

NOTE2: I'm not sure that I got the get_window_extent() right, or whether that's sufficient for providing the object bbox.

36.4.1 Methods

__init__(*self*, *x*=`0`, *y*=`0`, *text*=`''`, *color*=`None`, *verticalalignment*=`'center'`, *horizontalalignment*=`'center'`, *multialignment*=`None`, *fontproperties*=`None`, *rotation*=`None`, *dashlength*=`0.0`, *dashdirection*=`0`, *dashrotation*=`None`, *dashpad*=`3`, *dashpush*=`0`, *xaxis*=`True`)
Overrides: matplotlib.text.Text.__init__

draw(*self*, *renderer*)
Overrides: matplotlib.text.Text.draw

get_dashdirection(*self*)

get_dashlength(*self*)

get_dashpad(*self*)

get_dashpush(*self*)

get_dashrotation(*self*)

get_figure(*self*)

return the figure instance

Overrides: matplotlib.artist.Artist.get_figure

get_position(*self*)

Return x, y as tuple

Overrides: matplotlib.text.Text.get_position

get_window_extent(*self*, *renderer*=`None`)
Overrides: matplotlib.text.Text.get_window_extent

set_dashdirection(*self*, *dd*)

Set the direction of the dash following the text. 1 is before the text and 0 is after. The default is 0, which is what you'd want for the typical case of ticks below and on the left of the figure.
ACCEPTS: int

set_dashlength(*self*, *dl*)

Set the length of the dash.
ACCEPTS: float

set_dashpad(*self*, *dp*)

Set the "pad" of the TextWithDash, which is the extra spacing between the dash and the text, in canvas units.
ACCEPTS: float

set_dashpush(*self*, *dp*)

Set the "push" of the TextWithDash, which is the extra spacing between the beginning of the dash and the specified position.
ACCEPTS: float

set_dashrotation(*self*, *dr*)

Set the rotation of the dash.
ACCEPTS: float

set_figure(*self*, *fig*)

Set the figure instance the artist belong to.
ACCEPTS: a matplotlib.figure.Figure instance

Overrides: matplotlib.artist.Artist.set_figure

set_position(*self*, *xy*)

Set the xy position of the TextWithDash.
ACCEPTS: (x,y)

Overrides: matplotlib.text.Text.set_position

set_transform(*self*, *t*)

Set the Transformation instance used by this artist.
ACCEPTS: a matplotlib.transform transformation instance

Overrides: matplotlib.artist.Artist.set_transform

set_x(*self*, *x*)

Set the x position of the TextWithDash.
ACCEPTS: float

Overrides: matplotlib.text.Text.set_x

set_y(*self*, *y*)

Set the y position of the TextWithDash.
ACCEPTS: float

Overrides: matplotlib.text.Text.set_y

update_coords(*self*, *renderer*)

Computes the actual x,y coordinates for text based on the input x,y and the dashlength. Since the rotation is with respect to the actual canvas's coordinates we need to map back and forth.

Inherited from Artist: add_callback, convert_xunits, convert_yunits, get_alpha, get_animated, get_axes, get_clip_box, get_clip_on, get_clip_path, get_label, get_picker, get_transform, get_visible, get_zorder, have_units, is_figure_set, is_transform_set, pchanged, pickable, remove_callback, set, set_alpha, set_animated, set_axes, set_clip_box, set_clip_on, set_clip_path, set_label, set_lod, set_picker, set_visible, set_zorder, update
Inherited from Text: get_color, get_font_properties, get_fontname, get_fontsize, get_fontstyle, get_fontweight, get_ha, get_horizontalalignment, get_name, get_prop_tup, get_rotation, get_rotation_matrix, get_size, get_style, get_text, get_va, get_verticalalignment, get_weight, is_math_text, pick, set_backgroundcolor, set_bbox, set_color, set_family, set_fontname, set_fontproperties, set_fontsize, set_fontstyle, set_fontweight, set_ha, set_horizontalalignment, set_ma, set_multialignment, set_name, set_rotation, set_size, set_style, set_text, set_va, set_variant, set_verticalalignment, set_weight, update_from

36.4.2 Class Variables

Name	Description
__name__	**Value:** `'TextWithDash'` *(type=str)*
Inherited from Artist: aname *(p. 92)* **Inherited from Text:** zorder *(p. 511)*	

37 Module matplotlib.ticker

```
Tick locating and formatting
============================

This module contains classes to support completely configurable tick
locating and formatting.  Although the locators know nothing about
major or minor ticks, they are used by the Axis class to support major
and minor tick locating and formatting.  Generic tick locators and
formatters are provided, as well as domain specific custom ones..


Tick locating
-------------

The Locator class is the base class for all tick locators.  The
locators handle autoscaling of the view limits based on the data
limits, and the choosing of tick locations.  A useful semi-automatic
tick locator is MultipleLocator.  You initialize this with a base, eg
10, and it picks axis limits and ticks that are multiples of your
base.

The Locator subclasses defined here are

  * NullLocator     - No ticks

  * FixedLocator    - Tick locations are fixed

  * IndexLocator    - locator for index plots (eg where x = range(len(y))

  * LinearLocator   - evenly spaced ticks from min to max

  * LogLocator      - logarithmically ticks from min to max

  * MultipleLocator - ticks and range are a multiple of base;
                      either integer or float

  * OldAutoLocator  - choose a MultipleLocator and dyamically reassign
                      it for intelligent ticking during navigation

  * MaxNLocator     - finds up to a max number of ticks at nice
                      locations

  * AutoLocator     - MaxNLocator with simple defaults. This is the
                      default tick locator for most plotting.

There are a number of locators specialized for date locations - see
the dates module
```

You can define your own locator by deriving from Locator. You must override the __call__ method, which returns a sequence of locations, and you will probably want to override the autoscale method to set the view limits from the data limits.

If you want to override the default locator, use one of the above or a custom locator and pass it to the x or y axis instance. The relevant methods are::

```
  ax.xaxis.set_major_locator( xmajorLocator )
  ax.xaxis.set_minor_locator( xminorLocator )
  ax.yaxis.set_major_locator( ymajorLocator )
  ax.yaxis.set_minor_locator( yminorLocator )
```

The default minor locator is the NullLocator, eg no minor ticks on by default.

Tick formatting

Tick formatting is controlled by classes derived from Formatter. The formatter operates on a single tick value and returns a string to the axis.

```
  * NullFormatter      - no labels on the ticks

  * FixedFormatter     - set the strings manually for the labels

  * FuncFormatter      - user defined function sets the labels

  * FormatStrFormatter - use a sprintf format string

  * ScalarFormatter    - default formatter for scalars; autopick the fmt string

  * LogFormatter       - formatter for log axes
```

You can derive your own formatter from the Formatter base class by simply overriding the __call__ method. The formatter class has access to the axis view and data limits.

To control the major and minor tick label formats, use one of the following methods::

```
  ax.xaxis.set_major_formatter( xmajorFormatter )
  ax.xaxis.set_minor_formatter( xminorFormatter )
  ax.yaxis.set_major_formatter( ymajorFormatter )
```

```
  ax.yaxis.set_minor_formatter( yminorFormatter )

See examples/major_minor_demo1.py for an example of setting major an
minor ticks.  See the matplotlib.dates module for more information and
examples of using date locators and formatters.

DEVELOPERS NOTE

If you are implementing your own class or modifying one of these, it
is critical that you use viewlim and dataInterval READ ONLY MODE so
multiple axes can share the same locator w/o side effects!
```

37.1 Class AutoLocator

matplotlib.ticker.TickHelper
  matplotlib.ticker.Locator
    matplotlib.ticker.MaxNLocator
      AutoLocator

37.1.1 Methods

__init__(*self*) Overrides: matplotlib.ticker.MaxNLocator.__init__

Inherited from Locator: pan, refresh, zoom
Inherited from MaxNLocator: __call__, autoscale, bin_boundaries
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.1.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.2 Class FixedFormatter

matplotlib.ticker.TickHelper
  matplotlib.ticker.Formatter
    FixedFormatter

Return fixed strings for tick labels

37.2.1 Methods

__init__(*self*, *seq*)

seq is a sequence of strings. For positions i<len(seq) return seq[i] regardless of x. Otherwise return ”

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.Formatter.__call__

get_offset(*self*)
Overrides: matplotlib.ticker.Formatter.get_offset

set_offset_string(*self*, *ofs*)

Inherited from Formatter: format_data, format_data_short, set_locs
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.2.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.3 Class FixedLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
FixedLocator

Tick locations are fixed. If nbins is not None, the array of possible positions will be subsampled to keep the number of ticks <= nbins +1.

37.3.1 Methods

__init__(*self*, *locs*, *nbins*=`None`)

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__

Inherited from Locator: autoscale, pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.3.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.4 Class FormatStrFormatter

matplotlib.ticker.TickHelper
  matplotlib.ticker.Formatter
    FormatStrFormatter

Use a format string to format the tick

37.4.1 Methods

__init__(*self*, *fmt*)

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.Formatter.__call__

Inherited from Formatter: format_data, format_data_short, get_offset, set_locs
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.4.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.5 Class Formatter

matplotlib.ticker.TickHelper
  Formatter

Known Subclasses: DateFormatter, FixedFormatter, FormatStrFormatter, FuncFormatter, IndexDateFormatter, LogFormatter, NullFormatter, ScalarFormatter, AutoDateFormatter, OldScalarFormatter

Convert the tick location to a string

37.5.1 Methods

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos; pos=None indicated unspecified

format_data(*self*, *value*)

format_data_short(*self*, *value*)

return a short string version

get_offset(*self*)

set_locs(*self*, *locs*)

Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.5.2 Class Variables

Name	Description
locs	**Value:** `[]` *(type=`list`)*
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.6 Class FuncFormatter

matplotlib.ticker.TickHelper
  matplotlib.ticker.Formatter
    FuncFormatter

User defined function for formatting

37.6.1 Methods

__init__(*self*, *func*)

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.Formatter.__call__

Inherited from Formatter: format_data, format_data_short, get_offset, set_locs
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.6.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.7 Class IndexLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
IndexLocator

Place a tick on every multiple of some base number of points plotted, eg on every 5th point. It is assumed that you are doing index plotting; ie the axis is 0, len(data). This is mainly useful for x ticks.

37.7.1 Methods

__init__(*self*, *base*, *offset*)

place ticks on the i-th data points where (i-offset)%base==0

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__

Inherited from Locator: autoscale, pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.7.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.8 Class LinearLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
LinearLocator

Determine the tick locations

The first time this function is called it will try to set the number of ticks to make a nice tick partitioning.

Thereafter the number of ticks will be fixed so that interactive navigation will be nice

37.8.1 Methods

__init__(*self*, *numticks*=`None`, *presets*=`None`)

Use presets to set locs based on lom. A dict mapping vmin, vmax->locs

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__

autoscale(*self*)

Try to choose the view limits intelligently

Overrides: matplotlib.ticker.Locator.autoscale

Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.8.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.9 Class Locator

matplotlib.ticker.TickHelper
Locator

Known Subclasses: DateLocator, FixedLocator, IndexLocator, LinearLocator, LogLocator, MaxNLocator, MultipleLocator, NullLocator, OldAutoLocator

Determine the tick locations;

Note, you should not use the same locator between different Axis because the locator stores references to the Axis data and view limits

37.9.1 Methods

__call__(*self*)

Return the locations of the ticks

autoscale(*self*)

autoscale the view limits

pan(*self*, *numsteps*)

Pan numticks (can be positive or negative)

refresh(*self*)

refresh internal information based on current lim

zoom(*self*, *direction*)

Zoom in/out on axis; if direction is >0 zoom in, else zoom out

Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.9.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.10 Class LogFormatter

matplotlib.ticker.TickHelper
  matplotlib.ticker.Formatter
    LogFormatter

Known Subclasses: LogFormatterExponent, LogFormatterMathtext

Format values for log axis;

if attribute decadeOnly is True, only the decades will be labelled.

37.10.1 Methods

__init__(*self*, *base*=`10.0`, *labelOnlyBase*=`True`)

base is used to locate the decade tick, which will be the only one to be labeled if labelOnlyBase is False

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.Formatter.__call__

base(*self*, *base*)

change the base for labeling - warning: should always match the base used for LogLocator

format_data(*self*, *value*)
Overrides: matplotlib.ticker.Formatter.format_data

is_decade(*self*, *x*)

label_minor(*self*, *labelOnlyBase*)

switch on/off minor ticks labeling

nearest_long(*self*, *x*)

pprint_val(*self*, *x*, *d*)

Inherited from Formatter: format_data_short, get_offset, set_locs
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.10.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.11 Class LogFormatterExponent

matplotlib.ticker.TickHelper
　matplotlib.ticker.Formatter
　　matplotlib.ticker.LogFormatter
　　　LogFormatterExponent

Format values for log axis; using exponent = log_base(value)

37.11.1 Methods

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.LogFormatter.__call__

Inherited from Formatter: format_data_short, get_offset, set_locs
Inherited from LogFormatter: __init__, base, format_data, is_decade, label_minor, nearest_long, pprint_val
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.11.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.12 Class LogFormatterMathtext

matplotlib.ticker.TickHelper
  matplotlib.ticker.Formatter
    matplotlib.ticker.LogFormatter
      LogFormatterMathtext

Format values for log axis; using exponent = log_base(value)

37.12.1 Methods

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.LogFormatter.__call__

Inherited from Formatter: format_data_short, get_offset, set_locs
Inherited from LogFormatter: __init__, base, format_data, is_decade, label_minor, nearest_long, pprint_val
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.12.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.13 Class LogLocator

matplotlib.ticker.TickHelper
  matplotlib.ticker.Locator
    LogLocator

Determine the tick locations for log axes

37.13.1 Methods

__init__(*self*, *base*=`10.0`, *subs*=`[1.0]`)

place ticks on the location= base**i*subs[j]

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__

---

autoscale(*self*)

Try to choose the view limits intelligently

Overrides: matplotlib.ticker.Locator.autoscale

---

base(*self*, *base*)

set the base of the log scaling (major tick every base**i, i interger)

---

subs(*self*, *subs*)

set the minor ticks the log scaling every base**i*subs[j]

Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.13.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.14 Class MaxNLocator

matplotlib.ticker.TickHelper
    matplotlib.ticker.Locator
        MaxNLocator

Known Subclasses: AutoLocator

Select no more than N intervals at nice locations.

37.14.1 Methods

__init__(*self*, *nbins*=`10`, *steps*=`None`, *trim*=`True`, *integer*=`False`)

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__ extit(inherited documentation)

autoscale(*self*)

autoscale the view limits

Overrides: matplotlib.ticker.Locator.autoscale extit(inherited documentation)

bin_boundaries(*self*, *vmin*, *vmax*)

Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.14.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.15 Class MultipleLocator

matplotlib.ticker.TickHelper
    matplotlib.ticker.Locator
        MultipleLocator

Set a tick on every integer that is multiple of base in the viewInterval

37.15.1 Methods

__init__(*self*, *base*=`1.0`)

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__

autoscale(*self*)

Set the view limits to the nearest multiples of base that contain the data

Overrides: matplotlib.ticker.Locator.autoscale

Inherited from Locator: pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.15.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.16 Class NullFormatter

matplotlib.ticker.TickHelper
matplotlib.ticker.Formatter
NullFormatter

Always return the empty string

37.16.1 Methods

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.Formatter.__call__

Inherited from Formatter: format_data, format_data_short, get_offset, set_locs
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.16.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	

continued on next page

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.17 Class NullLocator

matplotlib.ticker.TickHelper
matplotlib.ticker.Locator
NullLocator

No ticks

37.17.1 Methods

__call__(*self*)

Return the locations of the ticks

Overrides: matplotlib.ticker.Locator.__call__

Inherited from Locator: autoscale, pan, refresh, zoom
Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.17.2 Class Variables

Name	Description
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.18 Class ScalarFormatter

matplotlib.ticker.TickHelper
matplotlib.ticker.Formatter
ScalarFormatter

Tick location is a plain old number. If useOffset==True and the data range is much smaller than the data average, then an offset will be determined such that the tick labels are meaningful. Scientific notation is used for data < 1e-3 or data >= 1e4.

37.18.1 Methods

__init__(*self*, *useOffset*=`True`, *useMathText*=`False`)

__call__(*self*, *x*, *pos*=`None`)

Return the format for tick val x at position pos

Overrides: matplotlib.ticker.Formatter.__call__

format_data(*self*, *value*, *sign*=`False`, *mathtext*=`False`)

return a formatted string representation of a number

Overrides: matplotlib.ticker.Formatter.format_data

format_data_short(*self*, *value*)

return a short formatted string representation of a number

Overrides: matplotlib.ticker.Formatter.format_data_short

get_offset(*self*)

Return scientific notation, plus offset

Overrides: matplotlib.ticker.Formatter.get_offset

pprint_val(*self*, *x*)

set_locs(*self*, *locs*)

set the locations of the ticks

Overrides: matplotlib.ticker.Formatter.set_locs

set_powerlimits(*self*, *lims*)

Sets size thresholds for scientific notation.
e.g. xaxis.set_powerlimits((-3, 4)) sets the pre-2007 default in which scientific notation is used for numbers less than 1e-3 or greater than 1e4. See also set_scientific().

set_scientific(*self*, *b*)

True or False to turn scientific notation on or off; see also set_powerlimits()

Inherited from TickHelper: set_bounds, set_data_interval, set_view_interval, verify_intervals

37.18.2 Class Variables

Name	Description
Inherited from Formatter: locs *(p. 525)*	
Inherited from TickHelper: dataInterval *(p. 537)*, viewInterval *(p. 537)*	

37.19 Class TickHelper

Known Subclasses: Formatter, Locator

37.19.1 Methods

set_bounds(*self*, *vmin*, *vmax*)

Set dataInterval and viewInterval from numeric vmin, vmax.
This is for stand-alone use of Formatters and/or Locators that require these intervals; that is, for cases where the Intervals do not need to be updated automatically.

set_data_interval(*self*, *interval*)

set_view_interval(*self*, *interval*)

verify_intervals(*self*)

37.19.2 Class Variables

Name	Description
dataInterval	**Value:** `None` *(type=`NoneType`)*
viewInterval	**Value:** `None` *(type=`NoneType`)*

38 Module matplotlib.transforms

The transforms module is broken into two parts, a collection of classes written in the extension module _transforms to handle efficient transformation of data, and some helper functions in transforms to make it easy to instantiate and use those objects. Hence the core of this module lives in _transforms.

The transforms class is built around the idea of a LazyValue. A LazyValue is a base class that defines a method get that returns the value. The concrete derived class Value wraps a float, and simply returns the value of that float. The concrete derived class BinOp allows binary operations on LazyValues, so you can add them, multiply them, etc. When you do something like

```
  inches = Value(8)
  dpi    = Value(72)
  width  = inches * dpi
```

width is a BinOp instance (that tells you the width of the figure in pixels). Later, if the figure size in changed, ie we call

```
  inches.set(10)
```

The width variable is automatically updated because it stores a pointer to the inches variable, not the value. Since a BinOp is also a lazy value, you can define binary operations on BinOps as well, such as

```
  middle = Value(0.5) * width
```

Pairs of LazyValue instances can occur as instances of two classes:

```
    pt = Point( Value(x), Value(y)) # where x, y are numbers
         pt.x(), pt.y() return  Value(x), Value(y))

    iv = Interval( Value(x), Value(y))
         iv.contains(z) returns True if z is in the closed interval
         iv.contains_open(z): same for open interval
         iv.span() returns y-x as a float
         iv.get_bounds() returns (x,y) as a tuple of floats
         iv.set_bounds(x, y) allows input of new floats
         iv.update(seq) updates the bounds to include all elements
              in a sequence of floats
         iv.shift(s) shifts the interval by s, a float
```

The bounding box class Bbox is also heavily used, and is defined by a lower left point ll and an upper right point ur. The points ll and ur

```
are given by Point(x, y) instances, where x and y are LazyValues.  So
you can represent a point such as

  ll = Point( Value(0), Value(0)  )  # the origin
  ur = Point( width, height )        # the upper right of the figure

where width and height are defined as above, using the product of the
figure width in inches and the dpi.  This is, in face, how the Figure
bbox is defined

  bbox = Bbox(ll, ur)

A bbox basically defines an x,y coordinate system, with ll giving the
lower left of the coordinate system and ur giving the upper right.

The bbox methods are

  ll()               - return the lower left Point
  ur()               - return the upper right Point
  contains(x,y)      - return True if self contains point
  overlaps(bbox)     - return True if self overlaps bbox
  overlapsx(bbox)    - return True if self overlaps bbox in the x interval
  overlapsy(bbox)    - return True if self overlaps bbox in the y interval
  intervalx()        - return the x Interval instance
  intervaly()        - return the y interval instance
  get_bounds()       - get the left, bottom, width, height bounding tuple
  update(xys, ignore) - update the bbox to bound all the xy tuples in
      xys; if ignore is true ignore the current contents of bbox and
      just bound the tuples.  If ignore is false, bound self + tuples
  width()            - return the width of the bbox
  height()           - return the height of the bbox
  xmax()             - return the x coord of upper right
  ymax()             - return the y coord of upper right
  xmin()             - return the x coord of lower left
  ymin()             - return the y coord of lower left
  scale(sx,sy)       - scale the bbox by sx, sy
  deepcopy()         - return a deep copy of self (pointers are lost)


The basic transformation maps one bbox to another, with an optional
nonlinear transformation of one of coordinates (eg log scaling).

The base class for transformations is Transformation, and the concrete
derived classes are SeparableTransformation and Affine.  Earlier
versions of matplotlib handled transformation of x and y separately
(ie we assumed all transformations were separable) but this makes it
difficult to do rotations or polar transformations, for example.  All
artists contain their own transformation, defaulting to the identity
```

```
transform.

The signature of a separable transformation instance is

  trans = SeparableTransformation(bbox1, bbox2, funcx, funcy)

where funcx and funcy operate on x and y.  The typical linear
coordinate transformation maps one bounding box to another, with funcx
and funcy both identity.  Eg,

  transData = Transformation(viewLim, displayLim,
                             Func(IDENTITY), Func(IDENTITY))

maps the axes view limits to display limits.  If the xaxis scaling is
changed to log, one simply calls

  transData.get_funcx().set_type(LOG10)

For more general transformations including rotation, the Affine class
is provided, which is constructed with 6 LazyValue instances:
a, b, c, d, tx, ty.  These give the values of the matrix transformation

  [xo  =  |a  c| [xi  + [tx
   yo]    |b  d|  yi]    ty]

where if sx, sy are the scaling components, tx, y are the translation
components, and alpha is the rotation

   a = sx*cos(alpha);
   b = -sx*sin(alpha);
   c = sy*sin(alpha);
   d = sy*cos(alpha);

From a user perspective, the most important Tranformation methods are

All transformations
-------------------
  freeze()             - eval and freeze the lazy objects
  thaw()               - release the lazy objects

  xy_tup(xy)           - transform the tuple (x,y)
  seq_x_y(x, y)        - transform the python sequences x and y
  numerix_x_y(x, y)    - x and y are numerix 1D arrays
  numerix_xy(xy)       - xy is a numerix array of shape (N,2)
  inverse_numerix_xy(xy)- inverse of the above
  seq_xy_tups(seq)     - seq is a sequence of xy tuples or a (N,2) array
  inverse_xy_tup(xy)   - apply the inverse transformation to tuple xy
```

```
  set_offset(xy, trans) - xy is an x,y tuple and trans is a
    Transformation instance.  This will apply a post transformational
    offset of all future transformations by xt,yt = trans.xy_tup(xy[0], xy[1])

  deepcopy()            - returns a deep copy; references are lost
  shallowcopy()         - returns a shallow copy excluding the offset

Separable transformations
-------------------------

  get_bbox1() - return the input bbox
  get_bbox2() - return the output bbox
  set_bbox1() - set the input bbox
  set_bbox2() - set the output bbox
  get_funcx() - return the Func instance on x
  get_funcy() - return the Func instance on y
  set_funcx() - set the Func instance on x
  set_funcy() - set the Func instance on y


Affine transformations
----------------------

  as_vec6() - return the affine as length 6 list of Values


In general, you shouldn't need to construct your own transformations,
but should use the helper functions defined in this module.


  zero                        - return Value(0)
  one                         - return Value(1)
  origin                      - return Point(zero(), zero())
  unit_bbox                   - return the 0,0 to 1,1 bounding box
  identity_affine             - An affine identity transformation
  identity_transform          - An identity separable transformation
  translation_transform       - a pure translational affine
  scale_transform             - a pure scale affine
  scale_sep_transform         - a pure scale separable transformation
  scale_translation_transform - a scale and translate affine
  bound_vertices              - return the bbox that bounds all the xy tuples
  bbox_all                    - return the bbox that bounds all the bboxes
  lbwh_to_bbox                - build a bbox from tuple
                                left, bottom, width, height tuple

  multiply_affines            - return the affine that is the matrix product of
                                the two affines
```

```
get_bbox_transform         - return a SeparableTransformation instance that
                              transforms one bbox to another

blend_xy_sep_transform     - mix the x and y components of two separable
                              transformations into a new transformation.
                              This allows you to specify x and y in
                              different coordinate systems

transform_bbox              - apply a transformation to a bbox and return the
                              transformed bbox

inverse_transform_bbox     - apply the inverse transformation of a bbox
                              and return the inverse transformed bbox

offset_copy                 - make a copy with an offset


The units/transform_unit.py code has many examples.

A related and partly overlapping class, PBox, has been added to the
original transforms module to facilitate Axes repositioning and resizing.
At present, the differences between Bbox and PBox include:

    Bbox works with the bounding box, the coordinates of the lower-left
    and upper-right corners; PBox works with the lower-left coordinates
    and the width, height pair (left, bottom, width, height, or 'lbwh').
    Obviously, these are equivalent, but lbwh is what is used by
    Axes._position, and it is the natural specification for the types of
    manipulations for which the PBox class was made.

    Bbox uses LazyValues grouped in pairs as 'll' and 'ur' Point objects;
    PBox uses a 4-element list, subclassed from the python list.

    Bbox and PBox methods are mostly quite different, reflecting their
    different original purposes.  Similarly, the CXX implementation of
    Bbox is good for methods such as update and for lazy evaluation, but
    for PBox intended uses, involving very little calculation, pure
    python probably is adequate.

In the future we may reimplement the PBox using Bbox
and transforms, or eliminate it entirely by adding its methods
and attributes to Bbox and/or putting them elsewhere in this module.
```

38.1 Functions

bbox_all(*bboxes*)
Return the Bbox that bounds all bboxes

blend_xy_sep_transform(*trans1*, *trans2*)

If trans1 and trans2 are SeparableTransformation instances, you can build a new SeparableTransformation from them by extracting the x and y bounding points and functions and recomposing a new SeparableTransformation
This function extracts all the relevant bits from trans1 and trans2 and returns the new Transformation instance. This is useful, for example, if you want to specify x in data coordinates and y in axes coordinates.

bound_vertices(*verts*)

Return the Bbox of the sequence of x,y tuples in verts

get_bbox_transform(*boxin*, *boxout*)

return the transform that maps transform one bounding box to another

get_vec6_rotation(*v*)

v is an affine vec6 a,b,c,d,tx,ty; return rotation in degrees

get_vec6_scales(*v*)

v is an affine vec6 a,b,c,d,tx,ty; return sx, sy

identity_affine()

Get an affine transformation that maps x,y -> x,y

identity_transform()

Get an affine transformation that maps x,y -> x,y

inverse_transform_bbox(*trans*, *bbox*)

inverse transform the bbox

invert_vec6(*v*)

```
v is a,b,c,d,tx,ty vec6 repr of a matrix
[ a  b  0
  c  d  0
  tx ty 1]

Return the inverse of v as a vec6
```

lbwh_to_bbox(*l*, *b*, *w*, *h*)

multiply_affines(*v1*, *v2*)

v1 and v2 are Affine instances

nonsingular(*vmin*, *vmax*, *expander*=`0.001`, *tiny*=`1.0000000000000001e-15`, *increasing*=`True`)

```
Ensure the endpoints of a range are not too close together.

"too close" means the interval is smaller than 'tiny' times
        the maximum absolute value.

If they are too close, each will be moved by the 'expander'.
If 'increasing' is True and vmin > vmax, they will be swapped.
```

offset_copy(*trans*, *fig*=`None`, *x*=`0`, *y*=`0`, *units*=`'inches'`)

```
Return a shallow copy of a transform with an added offset.
  args:
    trans is any transform
  kwargs:
    fig is the current figure; it can be None if units are 'dots'
    x, y give the offset
    units is 'inches', 'points' or 'dots'
```

one()

origin()

scale_sep_transform(*sx*, *sy*)

Return a pure scale transformation as a SeparableTransformation; sx and sy are LazyValue instances (Values or binary opertations on values)

scale_transform(*sx*, *sy*)

Return a pure scale transformation as an Affine instance; sx and sy are LazyValue instances (Values or binary opertations on values)

transform_bbox(*trans*, *bbox*)

transform the bbox to a new bbox

translation_transform(*tx*, *ty*)

return a pure tranlational transformation tx and ty are LazyValue instances (Values or binary operations on values)

unit_bbox()

Get a 0,0 -> 1,1 Bbox instance

zero()

38.2 Class PBox

```
__builtin__.object
        |
        __builtin__.list
                |
                PBox
```

A left-bottom-width-height (lbwh) specification of a bounding box, such as is used to specify the position of an Axes object within a Figure. It is a 4-element list with methods for changing the size, shape, and position relative to its container.

38.2.1 Methods

__init__(*self*, *box*, *container*=`None`, *llur*=`False`)
Overrides: __builtin__.list.__init__

anchor(*self*, *c*, *container*=`None`)

Shift to position c within its container.
c can be a sequence (cx, cy) where cx, cy range from 0 to 1, where 0 is left or bottom and 1 is right or top.
Alternatively, c can be a string: C for centered, S for bottom-center, SE for bottom-left, E for left, etc.
Optional arg container is the lbwh box within which the PBox is positioned; it defaults to the initial PBox.

as_llur(*self*)

get_container(*self*, *box*)

set_container(*self*, *box*=`None`)

shrink(*self*, *mx*, *my*)

Shrink the box by mx in the x direction and my in the y direction. The lower left corner of the box remains unchanged. Normally mx and my will be <= 1, but this is not enforced.

shrink_to_aspect(*self*, *box_aspect*, *fig_aspect*=1)

Shrink the box so that it is as large as it can be while having the desired aspect ratio, box_aspect. If the box coordinates are relative–that is, fractions of a larger box such as a figure–then the physical aspect ratio of that figure is specified with fig_aspect, so that box_aspect can also be given as a ratio of the absolute dimensions, not the relative dimensions.

splitx(*self*, **args*)

e.g., PB.splitx(f1, f2, ...)
Returns a list of new PBoxes formed by splitting the original one (PB) with vertical lines at fractional positions f1, f2, ...

splity(*self*, **args*)

e.g., PB.splity(f1, f2, ...)
Returns a list of new PBoxes formed by splitting the original one (PB) with horizontal lines at fractional positions f1, f2, ..., with y measured positive up.

Inherited from list: __add__, __contains__, __delitem__, __delslice__, __eq__, __ge__, __getattribute__, __getitem__, __getslice__, __gt__, __hash__, __iadd__, __imul__, __iter__, __le__, __len__, __lt__, __mul__, __ne__, __new__, __repr__, __reversed__, __rmul__, __setitem__, __setslice__, append, count, extend, index, insert, pop, remove, reverse, sort
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__

38.2.2 Class Variables

Name	Description
coefs	**Value:** {'C': (0.5, 0.5), 'E': (1.0, 0.5), 'SW': (0, 0)-, 'NE': (1.0, 1.0), 'N': (0.5,... *(type=dict)*

39 Module matplotlib.units

The classes here provide support for using custom classes with matplotlib, eg those that do not expose the array interface but know how to converter themselves to arrays. It also supoprts classes with units and units conversion. Use cases include converters for custom objects, eg a list of datetime objects, as well as for objects that are unit aware. We don't assume any particular units implementation, rather a units implementation must provide a ConversionInterface, and the register with the Registry converter dictionary. For example, here is a complete implementation which support plotting with native datetime objects

```
import matplotlib.units as units
import matplotlib.dates as dates
import matplotlib.ticker as ticker
import datetime

class DateConverter(units.ConversionInterface):

    def convert(value, unit):
        'convert value to a scalar or array'
        return dates.date2num(value)
    convert = staticmethod(convert)

    def axisinfo(unit):
        'return major and minor tick locators and formatters'
        if unit!='date': return None
        majloc = dates.AutoDateLocator()
        majfmt = dates.AutoDateFormatter(majloc)
        return AxisInfo(majloc=majloc,
                        majfmt=majfmt,
                        label='date')
    axisinfo = staticmethod(axisinfo)


    def default_units(x):
        'return the default unit for x or None'
        return 'date'
    default_units = staticmethod(default_units)

# finally we register our object type with a converter
units.registry[datetime.date] = DateConverter()
```

39.1 Class AxisInfo

information to support default axis labeling and tick labeling

39.1.1 Methods

__init__(*self*, *majloc*=`None`, *minloc*=`None`, *majfmt*=`None`, *minfmt*=`None`, *label*=`None`)

majloc and minloc: TickLocators for the major and minor ticks majfmt and minfmt: TickFormatters for the major and minor ticks label: the default axis label
If any of the above are None, the axis will simply use the default

39.2 Class ConversionInterface

The minimal interface for a converter to take custom instances (or sequences) and convert them to values mpl can use

39.2.1 Static Methods

axisinfo(*unit*)

return an units.AxisInfo instance for unit

convert(*obj*, *unit*)

convert obj using unit. If obj is a sequence, return the converted sequence. The ouput must be a sequence of scalars that can be used by the numerix array layer

default_units(*x*)

return the default unit for x or None

is_numlike(*x*)

The matplotlib datalim, autoscaling, locators etc work with scalars which are the units converted to floats given the current unit. The converter may be passed these floats, or arrays of them, even when units are set. Derived conversion interfaces may opt to pass plain-ol unitless numbers through the conversion interface and this is a helper function for them.

39.3 Class Registry

```
__builtin__.object
    |
    __builtin__.dict
        |
        Registry
```

register types with conversion interface

39.3.1 Methods

__init__(*self*)
Overrides: __builtin__.dict.__init__

get_converter(*self*, *x*)

get the converter interface instance for x, or None

Inherited from dict: __cmp__, __contains__, __delitem__, __eq__, __ge__, __getattribute__, __getitem__, __gt__, __hash__, __iter__, __le__, __len__, __lt__, __ne__, __new__, __repr__, __setitem__, clear, copy, get, has_key, items, iteritems, iterkeys, itervalues, keys, pop, popitem, setdefault, update, values
Inherited from object: __delattr__, __reduce__, __reduce_ex__, __setattr__, __str__
Inherited from type: fromkeys

40 Module matplotlib.widgets

GUI Neutral widgets

All of these widgets require you to predefine an Axes instance and pass that as the first arg. matplotlib doesn't try to be too smart in layout – you have to figure out how wide and tall you want your Axes to be to accommodate your widget.

40.1 Class Button

matplotlib.widgets.Widget
└ **Button**

```
A GUI neutral button

The following attributes are accesible

  ax    - the Axes the button renders into
  label - a text.Text instance
  color - the color of the button when not hovering
  hovercolor - the color of the button when hovering

Call "on_clicked" to connect to the button
```

40.1.1 Methods

__init__(*self*, *ax*, *label*, *image*=`None`, *color*=`'0.85'`, *hovercolor*=`'0.95'`)

```
ax is the Axes instance the button will be placed into

label is a string which is the button text

image if not None, is an image to place in the button -- can
  be any legal arg to imshow (array, matplotlib Image
  instance, or PIL image)

color is the color of the button when not activated

hovercolor is the color of the button when the mouse is over
  it
```

disconnect(*self*, *cid*)

remove the observer with connection id cid

on_clicked(*self*, *func*)

When the button is clicked, call this func with event
A connection id is returned which can be used to disconnect

40.1.2 Class Variables

Name	Description
Inherited from Widget: drawon *(p. 560)*, eventson *(p. 560)*	

40.2 Class CheckButtons

matplotlib.widgets.Widget
└ **CheckButtons**

```
A GUI neutral radio button

The following attributes are exposed

 ax - the Axes instance the buttons are in
 labels - a list of text.Text instances
 lines - a list of (line1, line2) tuples for the x's in the check boxes.
         These lines exist for each box, but have set_visible(False) when
         box is not checked
 rectangles - a list of patch.Rectangle instances

Connect to the CheckButtons with the on_clicked method
```

40.2.1 Methods

__init__(*self*, *ax*, *labels*, *actives*)

```
Add check buttons to axes.Axes instance ax

labels is a len(buttons) list of labels as strings

actives is a len(buttons) list of booleans indicating whether
 the button is active
```

disconnect(*self*, *cid*)

remove the observer with connection id cid

on_clicked(*self*, *func*)

When the button is clicked, call this func with button label
A connection id is returned which can be used to disconnect

40.2.2 Class Variables

Name	Description
Inherited from Widget: drawon *(p. 560)*, eventson *(p. 560)*	

40.3 Class Cursor

```
A horizontal and vertical line span the axes that and move with
the pointer.  You can turn off the hline or vline spectively with
the attributes

  horizOn =True|False: controls visibility of the horizontal line
  vertOn =True|False: controls visibility of the horizontal line

And the visibility of the cursor itself with visible attribute
```

40.3.1 Methods

__init__(*self*, *ax*, *useblit*=`False`, ***lineprops*)

Add a cursor to ax. If useblit=True, use the backend dependent blitting features for faster updates (GTKAgg only now). lineprops is a dictionary of line properties. See examples/widgets/cursor.py.

clear(*self*, *event*)

clear the cursor

onmove(*self*, *event*)

on mouse motion draw the cursor if visible

40.4 Class HorizontalSpanSelector

matplotlib.widgets.SpanSelector
└ **HorizontalSpanSelector**

40.4.1 Methods

__init__(*self*, *ax*, *onselect*, ***kwargs*)
Overrides: matplotlib.widgets.SpanSelector.__init__

Inherited from SpanSelector: ignore, onmove, press, release, update, update_background

40.5 Class Lasso

matplotlib.widgets.Widget
  Lasso

40.5.1 Methods

__init__(*self*, *ax*, *xy*, *callback*=`None`, *useblit*=`True`)

onmove(*self*, *event*)

onrelease(*self*, *event*)

40.5.2 Class Variables

Name	Description
Inherited from Widget: drawon *(p. 560)*, eventson *(p. 560)*	

40.6 Class LockDraw

some widgets, like the cursor, draw onto the canvas, and this is not desirable under all circumstaces, like when the toolbar is in zoom-to-rect mode and drawing a rectangle. The module level "lock" allows someone to grab the lock and prevent other widgets from drawing. Use matplotlib.widgets.lock(someobj) to pr

40.6.1 Methods

__init__(*self*)

__call__(*self*, *o*)

reserve the lock for o

available(*self*, *o*)

drawing is available to o

isowner(*self*, *o*)

o owns the lock

locked(*self*)

the lock is held

release(*self*, *o*)

release the lock

40.7 Class MultiCursor

Provide a vertical line cursor shared between multiple axes

from matplotlib.widgets import MultiCursor from pylab import figure, show, nx

t = nx.arange(0.0, 2.0, 0.01) s1 = nx.sin(2*nx.pi*t) s2 = nx.sin(4*nx.pi*t) fig = figure() ax1 = fig.add_subplot(211) ax1.plot(t, s1)

ax2 = fig.add_subplot(212, sharex=ax1) ax2.plot(t, s2)

multi = MultiCursor(fig.canvas, (ax1, ax2), color='r', lw=1) show()

40.7.1 Methods

__init__(*self*, *canvas*, *axes*, *useblit*=`True`, *******lineprops*)

clear(*self*, *event*)

clear the cursor

onmove(*self*, *event*)

40.8 Class RadioButtons

matplotlib.widgets.Widget
    RadioButtons

```
A GUI neutral radio button

The following attributes are exposed

 ax - the Axes instance the buttons are in
 activecolor - the color of the button when clicked
 labels - a list of text.Text instances
 circles - a list of patch.Circle instances
```

```
Connect to the RadioButtons with the on_clicked method
```

40.8.1 Methods

__init__(*self*, *ax*, *labels*, *active*=`0`, *activecolor*=`'blue'`)

Add radio buttons to axes.Axes instance ax
labels is a len(buttons) list of labels as strings
active is the index into labels for the button that is active
activecolor is the color of the button when clicked

disconnect(*self*, *cid*)

remove the observer with connection id cid

on_clicked(*self*, *func*)

When the button is clicked, call this func with button label
A connection id is returned which can be used to disconnect

40.8.2 Class Variables

Name	Description
Inherited from Widget: drawon *(p. 560)*, eventson *(p. 560)*	

40.9 Class RectangleSelector

```
Select a min/max range of the x axes for a matplotlib Axes

Example usage:

  ax = subplot(111)
  ax.plot(x,y)

  def onselect(eclick, erelease):
      'eclick and erelease are matplotlib events at press and release'
      print 'startposition : (%f,%f)'%(eclick.xdata, eclick.ydata)
      print 'endposition   : (%f,%f)'%(erelease.xdata, erelease.ydata)
      print 'used button   : ', eclick.button

  span = Selector(ax, onselect,drawtype='box')
  show()
```

40.9.1 Methods

__init__(*self*, *ax*, *onselect*, *drawtype*='box', *minspanx*=None, *minspany*=None, *useblit*=False, *lineprops*=None, *rectprops*=None)

```
Create a selector in ax.  When a selection is made, clear
the span and call onselect with

  onselect(pos_1, pos_2)

and clear the drawn box/line. There pos_i are arrays of length 2
containing the x- and y-coordinate.

If minspanx is not None then events smaller than minspanx
in x direction are ignored(it's the same for y).

The rect is drawn with rectprops; default
  rectprops = dict(facecolor='red', edgecolor = 'black',
                   alpha=0.5, fill=False)

The line is drawn with lineprops; default
  lineprops = dict(color='black', linestyle='-',
                   linewidth = 2, alpha=0.5)

Use type if you want the mouse to draw a line, a box or nothing
between click and actual position ny setting
drawtype = 'line', drawtype='box' or drawtype = 'none'.
```

ignore(*self*, *event*)

return True if event should be ignored

onmove(*self*, *event*)

on motion notify event if box/line is wanted

press(*self*, *event*)

on button press event

release(*self*, *event*)

on button release event

update(*self*)

draw using newfangled blit or oldfangled draw depending on useblit

update_background(*self*, *event*)

force an update of the background

40.10 Class Slider

matplotlib.widgets.Widget
└ **Slider**

```
A slider representing a floating point range

The following attributes are defined
  ax     : the slider axes.Axes instance
  val    : the current slider value
  vline  : a Line2D instance representing the initial value
  poly   : A patch.Polygon instance which is the slider
  valfmt : the format string for formatting the slider text
  label  : a text.Text instance, the slider label
  closedmin : whether the slider is closed on the minimum
  closedmax : whether the slider is closed on the maximum
  slidermin : another slider - if not None, this slider must be > slidermin
  slidermax : another slider - if not None, this slider must be < slidermax
  dragging : allow for mouse dragging on slider

Call on_changed to connect to the slider event
```

40.10.1 Methods

__init__(*self*, *ax*, *label*, *valmin*, *valmax*, *valinit*=0.5, *valfmt*='%1.2f', *closedmin*=True, *closedmax*=True, *slidermin*=None, *slidermax*=None, *dragging*=True)

```
Create a slider from valmin to valmax in axes ax;

valinit -  the slider initial position

label - the slider label

valfmt - used to format the slider value

closedmin and closedmax - indicate whether the slider interval is closed

slidermin and slidermax - be used to contrain the value of
  this slider to the values of other sliders.
```

disconnect(*self*, *cid*)

remove the observer with connection id cid

on_changed(*self*, *func*)

When the slider valud is changed, call this func with the new slider position
A connection id is returned which can be used to disconnect

reset(*self*)

reset the slider to the initial value if needed

set_val(*self*, *val*)

40.10.2 Class Variables

Name	Description
Inherited from Widget: drawon *(p. 560)*, eventson *(p. 560)*	

40.11 Class SpanSelector

Known Subclasses: HorizontalSpanSelector

```
Select a min/max range of the x or y axes for a matplotlib Axes

Example usage:

  ax = subplot(111)
  ax.plot(x,y)

  def onselect(vmin, vmax):
      print vmin, vmax
  span = SpanSelector(ax, onselect, 'horizontal')

  onmove_callback is an optional callback that will be called on mouse move
  with the span range
```

40.11.1 Methods

__init__(*self*, *ax*, *onselect*, *direction*, *minspan*=`None`, *useblit*=`False`, *rectprops*=`None`, *onmove_callback*=`None`)

```
Create a span selector in ax.  When a selection is made, clear
the span and call onselect with

  onselect(vmin, vmax)

and clear the span.

direction must be 'horizontal' or 'vertical'

If minspan is not None, ignore events smaller than minspan

The span rect is drawn with rectprops; default
  rectprops = dict(facecolor='red', alpha=0.5)

set the visible attribute to False if you want to turn off
the functionality of the span selector
```

ignore(*self*, *event*)

return True if event should be ignored

onmove(*self*, *event*)

on motion notify event

press(*self*, *event*)

on button press event

release(*self*, *event*)

on button release event

update(*self*)

draw using newfangled blit or oldfangled draw depending on useblit

update_background(*self*, *event*)

force an update of the background

40.12 Class SubplotTool

matplotlib.widgets.Widget
└ **SubplotTool**

A tool to adjust to subplot params of fig

40.12.1 Methods

__init__(*self*, *targetfig*, *toolfig*)

targetfig is the figure to adjust
toolfig is the figure to embed the the subplot tool into. If None, a default pylab figure will be created. If you are using this from the GUI

funcbottom(*self*, *val*)

funchspace(*self*, *val*)

funcleft(*self*, *val*)

funcright(*self*, *val*)

functop(*self*, *val*)

funcwspace(*self*, *val*)

40.12.2 Class Variables

Name	Description
Inherited from Widget: drawon *(p. 560)*, eventson *(p. 560)*	

40.13 Class Widget

Known Subclasses: Button, CheckButtons, Lasso, RadioButtons, Slider, SubplotTool

OK, I couldn't resist; abstract base class for mpl GUI neutral widgets

40.13.1 Class Variables

Name	Description
drawon	**Value:** `True` *(type=`bool`)*
eventson	**Value:** `True` *(type=`bool`)*