# lines

## matplotlib.lines

This module contains all the 2D line class which can draw with a variety of line styles, markers and colors.

*class* matplotlib.lines.Line2D(*xdata*, *ydata*, *linewidth=None*, *linestyle=None*, *color=None*, *marker=None*, *markersize=None*, *markeredgewidth=None*, *markeredgecolor=None*, *markerfacecolor=None*, *markerfacecoloralt='none'*, *fillstyle=None*, *antialiased=None*, *dash_capstyle=None*, *solid_capstyle=None*, *dash_joinstyle=None*, *solid_joinstyle=None*, *pickradius=5*, *drawstyle=None*, *markevery=None*, ***kwargs*)

Bases: matplotlib.artist.Artist

A line - the line can have both a solid linestyle connecting all the vertices, and a marker at each vertex. Additionally, the drawing of the solid line is influenced by the drawstyle, e.g., one can create “stepped” lines in various styles.

Create a Line2D instance with *x* and *y* data in sequences *xdata*, *ydata*.

The kwargs are Line2D properties:

| Property | Description |
|---|---|
| agg_filter | unknown |
| alpha | float (0.0 transparent through 1.0 opaque) |
| animated | [True \| False] |
| antialiased or aa | [True \| False] |
| axes | an Axes instance |
| clip_box | a matplotlib.transforms.Bbox instance |

| Property | Description |
|---|---|
| clip_on | [True \| False] |
| clip_path | [ (Path, Transform) \| Patch \| None ] |
| color or c | any matplotlib color |
| contains | a callable function |
| dash_capstyle | ['butt' \| 'round' \| 'projecting'] |
| dash_joinstyle | ['miter' \| 'round' \| 'bevel'] |
| dashes | sequence of on/off ink in points |
| drawstyle | ['default' \| 'steps' \| 'steps-pre' \| 'steps-mid' \| 'steps-post'] |
| figure | a matplotlib.figure.Figure instance |
| fillstyle | ['full' \| 'left' \| 'right' \| 'bottom' \| 'top' \| 'none'] |
| gid | an id string |
| label | string or anything printable with '%s' conversion. |
| linestyle or ls | ['solid' \| 'dashed', 'dashdot', 'dotted' \| (offset, on-off-dash-seq) \| '-' \| '--' \| '-.' \| ':' \| 'None' \| ' ' \| ''] |
| linewidth or lw | float value in points |
| marker | A valid marker style |
| markeredgecolor or mec | any matplotlib color |
| markeredgewidth or mew | float value in points |
| markerfacecolor or mfc | any matplotlib color |
| markerfacecoloralt or mfcalt | any matplotlib color |
| markersize or ms | float |
| markevery | [None \| int \| length-2 tuple of int \| slice \| list/array of int \| float \| length-2 tuple of float] |
| path_effects | unknown |
| picker | float distance in points or callable pick function fn(artist, event) |
| pickradius | float distance in points |
| rasterized | [True \| False \| None] |
| sketch_params | unknown |

| Property | Description |
|---|---|
| snap | unknown |
| solid_capstyle | [‘butt’ \| ‘round’ \| ‘projecting’] |
| solid_joinstyle | [‘miter’ \| ‘round’ \| ‘bevel’] |
| transform | a matplotlib.transforms.Transform instance |
| url | a url string |
| visible | [True \| False] |
| xdata | 1D array |
| ydata | 1D array |
| zorder | any number |

See set_linestyle() for a decription of the line styles, set_marker() for a description of the markers, and set_drawstyle() for a description of the draw styles.

---

axes

The Axes instance the artist resides in, or *None*.

---

contains(*mouseevent*)

Test whether the mouse event occurred on the line. The pick radius determines the precision of the location test (usually within five points of the value). Use get_pickradius() or set_pickradius() to view or modify it.

Returns *True* if any values are within the radius along with {'ind': pointlist}, where *pointlist* is the set of points within the radius.

TODO: sort returned indices by distance

---

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

draw the Line with renderer unless visibility is False

---

drawStyleKeys = *['default', 'steps-mid', 'steps-pre', 'steps-post', 'steps']*

drawStyles = *{'default': '_draw_lines', 'steps-mid': '_draw_steps_mid', 'steps-pre': '_draw_steps_pre', 'steps-post': '_draw_steps_post', 'steps': '_draw_steps_pre'}*

---

fillStyles = *('full', 'left', 'right', 'bottom', 'top', 'none')*

---

filled_markers = *('o', 'v', '^', '<', '>', '8', 's', 'p', '*', 'h', 'H', 'D', 'd', 'P', 'X')*

---

get_aa()

alias for get_antialiased

---

get_antialiased()

---

get_c()

alias for get_color

---

get_color()

---

get_dash_capstyle()

Get the cap style for dashed linestyles

---

get_dash_joinstyle()

Get the join style for dashed linestyles

---

get_data(*orig=True*)

Return the xdata, ydata.

If *orig* is *True*, return the original data.

get_drawstyle()

get_fillstyle()

return the marker fillstyle

get_linestyle()

get_linewidth()

get_ls()

alias for get_linestyle

get_lw()

alias for get_linewidth

get_marker()

get_markeredgecolor()

get_markeredgewidth()

get_markerfacecolor()

get_markerfacecoloralt()

get_markersize()

get_markevery()

return the markevery setting

`get_mec`()

alias for get_markeredgecolor

`get_mew`()

alias for get_markeredgewidth

`get_mfc`()

alias for get_markerfacecolor

`get_mfcalt`(*alt=False*)

alias for get_markerfacecoloralt

`get_ms`()

alias for get_markersize

`get_path`()

Return the `Path` object associated with this line.

`get_pickradius`()

return the pick radius used for containment tests

`get_solid_capstyle`()

Get the cap style for solid linestyles

`get_solid_joinstyle`()

Get the join style for solid linestyles

`get_window_extent`(*renderer*)

get_xdata(*orig=True*)

Return the xdata.

If *orig* is *True*, return the original data, else the processed data.

---

get_xydata()

Return the *xy* data as a Nx2 numpy array.

---

get_ydata(*orig=True*)

Return the ydata.

If *orig* is *True*, return the original data, else the processed data.

---

is_dashed()

return True if line is dashstyle

---

```
lineStyles = {'-': '_draw_solid', '--':
'_draw_dashed', '-.': '_draw_dash_dot', ':':
'_draw_dotted', 'None': '_draw_nothing', ' ':
'_draw_nothing', '': '_draw_nothing'}
```

---

```
markers = {'.': 'point', ',': 'pixel', 'o': 'circle',
'v': 'triangle_down', '^': 'triangle_up', '<':
'triangle_left', '>': 'triangle_right', '1':
'tri_down', '2': 'tri_up', '3': 'tri_left', '4':
'tri_right', '8': 'octagon', 's': 'square', 'p':
'pentagon', '*': 'star', 'h': 'hexagon1', 'H':
'hexagon2', '+': 'plus', 'x': 'x', 'D': 'diamond',
'd': 'thin_diamond', '|': 'vline', '_': 'hline', 'P':
'plus_filled', 'X': 'x_filled', 0: 'tickleft', 1:
'tickright', 2: 'tickup', 3: 'tickdown', 4:
'caretleft', 5: 'caretright', 6: 'caretup', 7:
'caretdown', 8: 'caretleftbase', 9: 'caretrightbase',
10: 'caretupbase', 11: 'caretdownbase', 'None':
'nothing', None: 'nothing', ' ': 'nothing', '':
'nothing'}
```

recache(*always=False*)

recache_always()

set_aa(*val*)

alias for set_antialiased

set_antialiased(*b*)

True if line should be drawin with antialiased rendering

ACCEPTS: [True | False]

set_c(*val*)

alias for set_color

set_color(*color*)

Set the color of the line

ACCEPTS: any matplotlib color

set_dash_capstyle(*s*)

Set the cap style for dashed linestyles

ACCEPTS: [‘butt’ | ‘round’ | ‘projecting’]

set_dash_joinstyle(*s*)

Set the join style for dashed linestyles ACCEPTS: [‘miter’ | ‘round’ | ‘bevel’]

set_dashes(*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`(**args*)

Set the x and y data

ACCEPTS: 2D array (rows are x, y) or two 1D arrays

`set_drawstyle`(*drawstyle*)

Set the drawstyle of the plot

'default' connects the points with lines. The steps variants produce step-plots. 'steps' is equivalent to 'steps-pre' and is maintained for backward-compatibility.

ACCEPTS: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']

`set_fillstyle`(*fs*)

Set the marker fill style; 'full' means fill the whole marker. 'none' means no filling; other options are for half-filled markers.

ACCEPTS: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']

`set_linestyle`(*ls*)

Set the linestyle of the line (also accepts drawstyles, e.g., `'steps--'`)

| linestyle | description |
|---|---|
| `'-'` or `'solid'` | solid line |
| `'--'` or `'dashed'` | dashed line |
| `'-.'` or `'dashdot'` | dash-dotted line |
| `':'` or `'dotted'` | dotted line |
| `'None'` | draw nothing |
| `' '` | draw nothing |
| `''` | draw nothing |

'steps' is equivalent to 'steps-pre' and is maintained for backward-compatibility.

Alternatively a dash tuple of the following form can be provided:

```
(offset, onoffseq),
```

where `onoffseq` is an even length tuple of on and off ink in points.

ACCEPTS: [‘solid’ | ‘dashed’, ‘dashdot’, ‘dotted’ |

(offset, on-off-dash-seq) | `'-'` | `'--'` | `'-.'` | `':'` | `'None'` | `' '` | `''`]

**See also**

`set_drawstyle()`

To set the drawing style (stepping) of the plot.

**Parameters:** **ls** : { `'-'`, `'--'`, `'-.'`, `':'`} and more see description

The line style.

---

`set_linewidth(w)`

Set the line width in points

ACCEPTS: float value in points

---

`set_ls(val)`

alias for set_linestyle

---

`set_lw(val)`

alias for set_linewidth

---

`set_marker(marker)`

Set the line marker

ACCEPTS: `A valid marker style`

**Parameters:** **marker: marker style**

See `markers` for full description of possible argument

---

`set_markeredgecolor`(*ec*)

Set the marker edge color

ACCEPTS: any matplotlib color

---

`set_markeredgewidth`(*ew*)

Set the marker edge width in points

ACCEPTS: float value in points

---

`set_markerfacecolor`(*fc*)

Set the marker face color.

ACCEPTS: any matplotlib color

---

`set_markerfacecoloralt`(*fc*)

Set the alternate marker face color.

ACCEPTS: any matplotlib color

---

`set_markersize`(*sz*)

Set the marker size in points

ACCEPTS: float

---

`set_markevery`(*every*)

Set the markevery property to subsample the plot when using markers.

e.g., if `every=5`, every 5-th marker will be plotted.

ACCEPTS: [None | int | length-2 tuple of int | slice | list/array of int | float | length-2 tuple of float]

**Parameters:** **every: None | int | length-2 tuple of int | slice | list/array of int | float | length-2 tuple of float**

Which markers to plot.

- every=None, every point will be plotted.
- every=N, every N-th marker will be plotted starting with marker 0.
- every=(start, N), every N-th marker, starting at point start, will be plotted.
- every=slice(start, end, N), every N-th marker, starting at point start, upto but not including point end, will be plotted.
- every=[i, j, m, n], only markers at points i, j, m, and n will be plotted.
- every=0.1, (i.e. a float) then markers will be spaced at approximately equal distances along the line; the distance along the line between markers is determined by multiplying the

display-coordinate distance of the axes bounding-box diagonal by the value of every.

- every=(0.5, 0.1) (i.e. a length-2 tuple of float), the same functionality as every=0.1 is exhibited but the first marker will be 0.5 multiplied by the display-cordinate-diagonal-distance along the line.

## Notes

Setting the markevery property will only show markers at actual data points. When using float arguments to set the markevery property on irregularly spaced data, the markers will likely not appear evenly spaced because the actual data points do not coincide with the theoretical spacing between markers.

When using a start offset to specify the first marker, the offset will be from the first data point which may be different from the first the visible data point if the plot is zoomed in.

If zooming in on a plot when using float arguments then the actual data points that have markers will change because the distance between markers is always determined from the display-coordinates axes-bounding-box-diagonal regardless of the actual axes data limits.

---

`set_mec(val)`

alias for set_markeredgecolor

`set_mew`(*val*)

alias for set_markeredgewidth

`set_mfc`(*val*)

alias for set_markerfacecolor

`set_mfcalt`(*val*)

alias for set_markerfacecoloralt

`set_ms`(*val*)

alias for set_markersize

`set_picker`(*p*)

Sets the event picker details for the line.

ACCEPTS: float distance in points or callable pick function `fn(artist, event)`

`set_pickradius`(*d*)

Sets the pick radius used for containment tests

ACCEPTS: float distance in points

`set_solid_capstyle`(*s*)

Set the cap style for solid linestyles

ACCEPTS: [‘butt’ | ‘round’ | ‘projecting’]

`set_solid_joinstyle`(*s*)

Set the join style for solid linestyles ACCEPTS: [‘miter’ | ‘round’ | ‘bevel’]

`set_transform`(*t*)

set the Transformation instance used by this artist

ACCEPTS: a `matplotlib.transforms.Transform` instance

---

`set_xdata(x)`

Set the data np.array for x

ACCEPTS: 1D array

---

`set_ydata(y)`

Set the data np.array for y

ACCEPTS: 1D array

---

`update_from(other)`

copy properties from other to self

---

`validCap = ('butt', 'round', 'projecting')`

---

`validJoin = ('miter', 'round', 'bevel')`

---

`zorder = 2`

*class* `matplotlib.lines.VertexSelector(line)`

Bases: `object`

Manage the callbacks to maintain a list of selected vertices for `matplotlib.lines.Line2D`. Derived classes should override `process_selected()` to do something with the picks.

Here is an example which highlights the selected verts with red circles:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as lines
```

```
class HighlightSelected(lines.VertexSelector):
    def __init__(self, line, fmt='ro', **kwargs):
        lines.VertexSelector.__init__(self, line)
        self.markers, = self.axes.plot([], [], fmt, **kw

    def process_selected(self, ind, xs, ys):
        self.markers.set_data(xs, ys)
        self.canvas.draw()

fig = plt.figure()
ax = fig.add_subplot(111)
x, y = np.random.rand(2, 30)
line, = ax.plot(x, y, 'bs-', picker=5)

selector = HighlightSelected(line)
plt.show()
```

Initialize the class with a `matplotlib.lines.Line2D` instance. The line should already be added to some `matplotlib.axes.Axes` instance and should have the picker property set.

---

`onpick`(*event*)

When the line is picked, update the set of selected indicies.

---

`process_selected`(*ind*, *xs*, *ys*)

Default “do nothing” implementation of the `process_selected()` method.

*ind* are the indices of the selected vertices. *xs* and *ys* are the coordinates of the selected vertices.

---

`matplotlib.lines.segment_hits`(*cx*, *cy*, *x*, *y*, *radius*)

Determine if any line segments are within radius of a point. Returns the list of line segments that are within that radius.