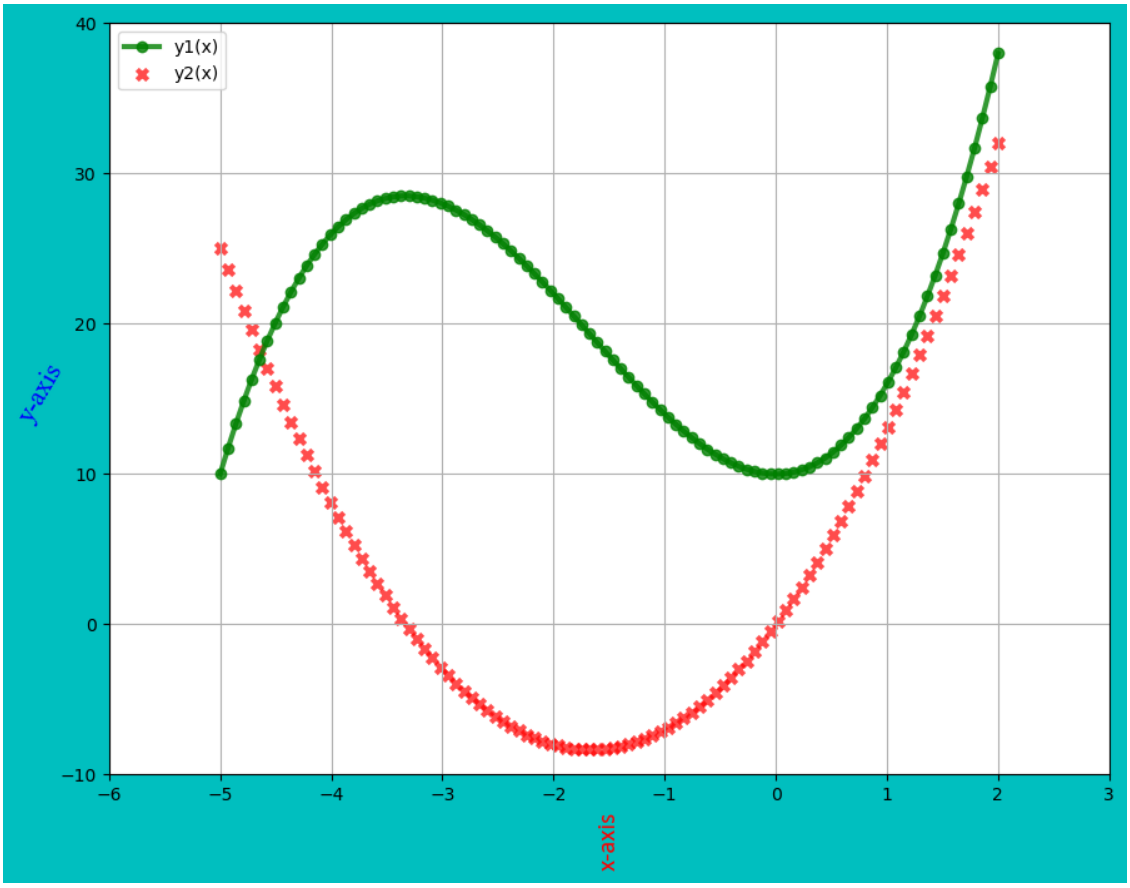
matplotlib-object-oriented

June 16, 2024

[3]:
```
'''A graph in Matplotlib is structured in terms of a Figure instance and one or␣
 ↪more
Axes instances within the figure. The Figure instance provides a canvas area␣
 ↪for drawing,
and the Axes instances provide coordinate systems that are assigned to fixed␣
 ↪regions of
the total figure canvas.'''
from numpy import *
from matplotlib.pyplot import *
#plt.style.use('Stylename') #Common stylenames: seaborn-colorblind','grayscale'␣
 ↪(for printing),'tableau-colorblind10','Solarize_Light2','bmh'
```

[4]:
```
#Simple Plot:
# For interactive Mode use below patch of code:
'''import matplotlib as mpl
mpl.use('TkAgg')'''
'''As introduced in the previous section, the Figure object is used in␣
 ↪Matplotlib to
represent a graph. In addition to providing a canvas on which, for example, Axes
instances can be placed, the Figure object also provides methods for performing␣
 ↪actions
on figures, and it has several attributes that can be used to configure the␣
 ↪properties of a figure'''
'''A Figure object can be created using the function plt.figure, which takes␣
 ↪several
optional keyword arguments for setting figure properties. In particular, it␣
 ↪accepts the
figsize keyword argument, which should be assigned to a tuple on the form␣
 ↪(width,
height), specifying the width and height of the figure canvas in inches. It can␣
 ↪also
be useful to specify the color of the figure canvas by setting the facecolor␣
 ↪keyword
argument'''
fig=figure(figsize=(8,6), facecolor='c')
'''Once a Figure is created, we can use the add_axes method to create a new
```

```
Axes instance and assign it to a region on the figure canvas. The add_axes␣
 ↪takes one
mandatory argument: a list containing the coordinates of the lower-left corner␣
 ↪and
the width and height of the Axes in the figure canvas coordinate system, on the␣
 ↪format
(left, bottom, width, height), Here, (left,bottom) represent coordinates where␣
 ↪left=bottom
corner of axes lines and are in inches'''
ax = fig.add_axes((0, 0, 1, 1),facecolor="w")
'''After axes creation we plot using some ploting function which takes some␣
 ↪arguments for
plot description and line properties specification i.e␣
 ↪lw=linewidth,ls=linestyle,
alpha=transparency (0 to 1 i.e 0% to 100%), marker=Each data point, whether or␣
 ↪not it
is connected with adjacent data points etc.'''
x = linspace(-5, 2, 100)
y1 = x**3 + 5*x**2 + 10
y2 = 3*x**2 + 10*x
ax.plot(x, y1, color="g",lw=3,ls='-',alpha=0.8,label="y1(x)",marker='o')
ax.scatter(x, y2, color="r",lw=3,alpha=0.7,label="y2(x)",marker='x')
'''We can use the set_xlim and set_ylim methods of the Axes object to set axis␣
 ↪ranges explicitly other than default'''
ax.set_xlim(-6,3 )
ax.set_ylim(-10, 40)
'''Below is label/legend of axes. We can also format text of labels/titles.'''
ax.set_xlabel("x-axis", color='r', fontsize=15, family='Calibri', rotation=90)
ax.set_ylabel("y-axis", color='b', fontsize=15, family='Times New Roman',␣
 ↪rotation=60)
ax.grid()
ax.legend(loc=2) #loc=1 (upper right corner), loc=4 (lower right corner)
'''To save this figure as pdf (or you can specify format) The resolution of the␣
 ↪generated image can be set with the dpi argument.
DPI stands for "dots per inch,"  and since the figure size is specified in␣
 ↪inches using the figsize argument, multiplying
these numbers gives the output image size in pixels. For example, with␣
 ↪figsize=(8, 6) and dpi=100, the size of the generated image is
800x600 pixels'''
fig.savefig("plot.pdf", dpi=100, facecolor="w",transparent=True)
#We can later change figsize by this argument: fig.
 ↪set_size_inches([width,Height])
#Note that fig,ax=plt.subplots() is simple and reliable method instead of above␣
 ↪for general plotting
```



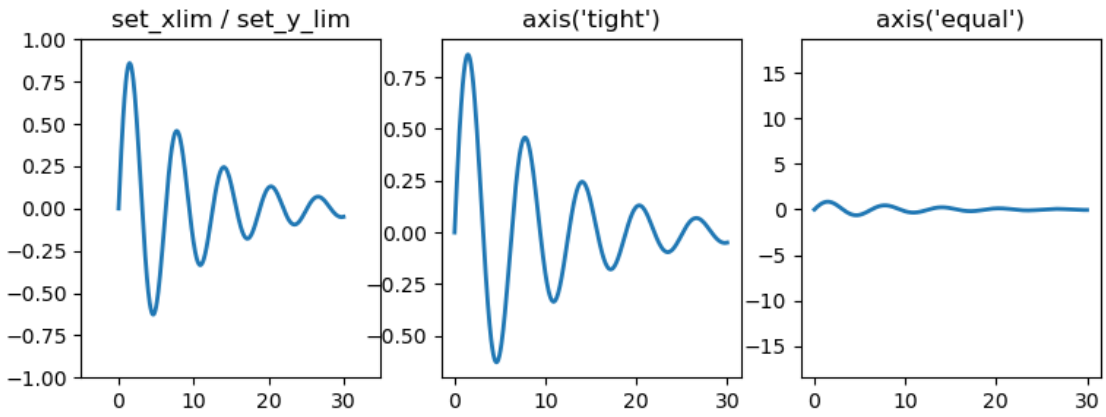
[4]:
```
'''A summary of commonly used 2D plot functions is shown in attached'''
'''A summary of Basic Line Properties and Their Corresponding Argument Names for
Use with the Matplotlib Plotting Methods is attached'''
'''Summary of Selected Font Properties and the Corresponding Keyword
Arguments is attached''
symbols: - , --, -., , . , , , o , ^ , v , < , > , s , + , x , D , d , 1 ,
2 , 3 , 4 , h , H , p ,| , _, colors b, g, r, c, m, y, k, w'''
```

[4]:
```
"Summary of Selected Font Properties and the Corresponding Keyword\nArguments is
attached''\nsymbols: - , --, -., , . , , , o , ^ , v , < , > , s , + , x , D , d
, 1 ,  \n2 , 3 , 4 , h , H , p ,| , _, colors b, g, r, c, m, y, k, w"
```

[5]:
```
#Subplot:
x = linspace(0, 30, 500)
y = sin(x) * np.exp(-x/10)
fig, axes = subplots(1, 3, figsize=(9, 3)) #IT TAKES AN ARG sharey=True which␣
 ↪makes all subplots y-axis same
axes[0].plot(x, y, lw=2)
axes[0].set_xlim(-5, 35)
axes[0].set_ylim(-1, 1)
```

```
axes[0].set_title("set_xlim / set_y_lim")
axes[1].plot(x, y, lw=2)
'''An alternative to set_xlim and set_ylim is the axis method, which, accepts␣
 ↪the string argument 'tight',
for a coordinate range that tightly fit the lines it contains, and 'equal', for␣
 ↪a coordinate range where one
unit length along each axis corresponds to the same number of pixels'''
axes[1].axis('tight')
axes[1].set_title("axis('tight')")
axes[2].plot(x, y, lw=2)
axes[2].axis('equal')
axes[2].set_title("axis('equal')")
```

[5]: Text(0.5, 1.0, "axis('equal')")

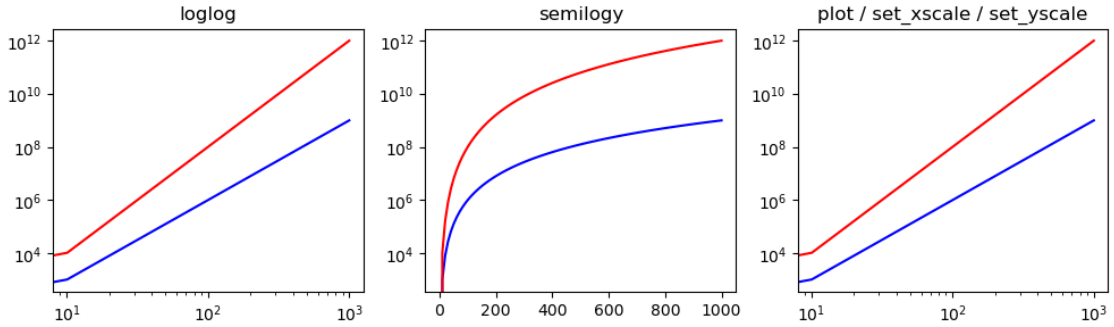


[6]:
```
'''For very basic customizations of  Axis Ticks, Tick Labels, and Grids etc.
see Chap 04 of book: Numerical Python Scientific Computing and Data Science
Applications with Numpy, SciPy and Matplotlib, Second Edition, Robert␣
 ↪Johansson'''
```

[6]: 'For very basic customizations of Axis Ticks, Tick Labels, and Grids etc. \nsee Chap 04 of book: Numerical Python Scientific Computing and Data Science \nApplications with Numpy, SciPy and Matplotlib, Second Edition, Robert Johansson'

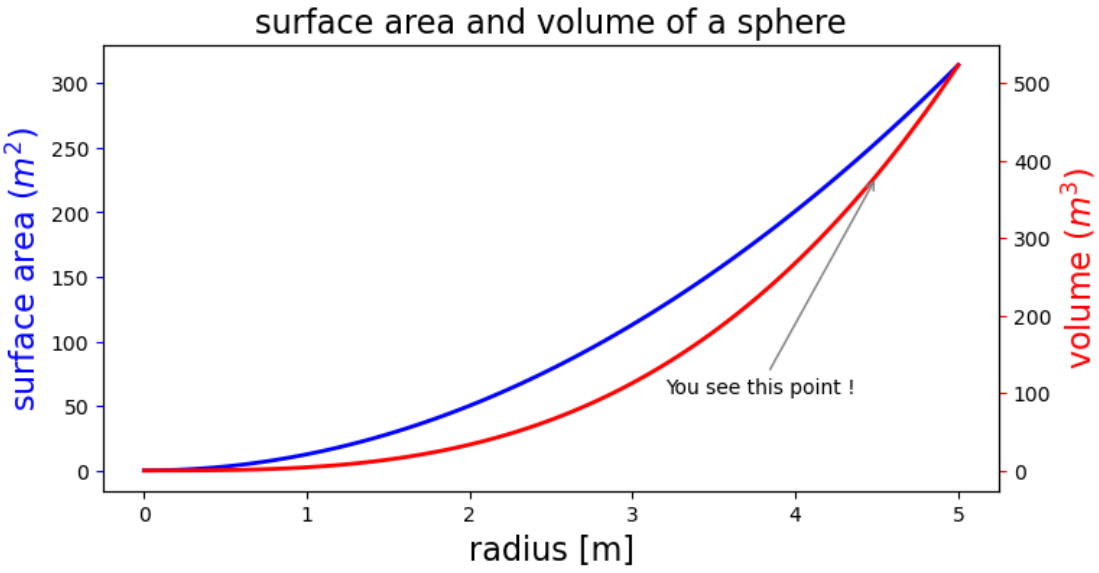
[7]:
```
#Log Plots (i.e Power of 10):
fig, axes = subplots(1, 3, figsize=(12, 3))
x = linspace(0, 1e3, 100)
y1, y2 = x**3, x**4
axes[0].set_title('loglog')
axes[0].loglog(x, y1, 'b', x, y2, 'r')
axes[1].set_title('semilogy')
```

```
axes[1].semilogy(x, y1, 'b', x, y2, 'r')
axes[2].set_title('plot / set_xscale / set_yscale')
axes[2].plot(x, y1, 'b', x, y2, 'r')
axes[2].set_xscale('log')
axes[2].set_yscale('log')
```



[20]:
```
#Twin Axes:
fig, ax1 = subplots(figsize=(8, 4))
r = linspace(0, 5, 100)
a = 4 * np.pi * r ** 2 # area
v = (4 * np.pi / 3) * r ** 3 # volume
ax1.set_title("surface area and volume of a sphere", fontsize=16)
ax1.set_xlabel("radius [m]", fontsize=16)
ax1.plot(r, a, lw=2, color="blue")
ax1.set_ylabel(r"surface area ($m^2$)", fontsize=16, color="blue")
ax1.tick_params('y',color='b') #To modify yticks
ax2 = ax1.twinx()
ax2.plot(r, v, lw=2, color="red")
ax2.set_ylabel(r"volume ($m^3$)", fontsize=16, color="red")
ax2.tick_params('y',color='r')
ax2.annotate("You see this point !",xy=(4.5,380),xytext=(3.
↪2,100),arrowprops={"arrowstyle":"->", "color":"gray"}) #here xytext␣
↪specifies loc. of annotation and xy what to annotate etc.
```

[20]:
```
Text(3.2, 100, 'You see this point !')
```



[ ]:
```
#Spines:
#Normal Plot:
x = np.linspace(-10, 10, 500)
y = np.sin(x) / x
fig, ax = subplots(figsize=(8, 4))
ax.plot(x, y, linewidth=2)
```

[ ]:
```
#Spine Plot:
x = np.linspace(-10, 10, 500)
y = np.sin(x) / x
fig, ax = subplots(figsize=(8, 4))
ax.plot(x, y, linewidth=2)
ax.plot(x, y, linewidth=2)
# remove top and right spines
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
# move bottom and left spine to x = 0 and y = 0
ax.set_xticks([-10, -5, 5, 10])
ax.set_yticks([0.5, 1]) #Specify location of tics
ax.spines['bottom'].set_position(('data', 0))
ax.spines['left'].set_position(('data', 0))
```

[ ]:
```
#Inset:
'''In matplotlib, an inset refers to a smaller plot that is embedded within a␣
 ↪larger plot.
This is often used to show a magnified or detailed view of a specific region of␣
 ↪interest within the main plot.'''
```

```
x = linspace(0, 10, 100)
y1 = sin(x)
y2 = cos(x)
fig=figure(figsize=(5,5))
ax1=fig.add_axes((0,0,1,1))
ax1.plot(x, y1, label='sin(x)')
ax1.plot(x, y2, label='cos(x)')
ax1.legend()
# Create an inset axis object
ax_inset = fig.add_axes([0.325,0.8, 0.2, 0.15])  # [left, bottom, width, height]
ax_inset.plot(x, y1, label='sin(x)')
ax_inset.plot(x, y2, label='cos(x)')
ax_inset.legend()
ax_inset.set_title('Inset Plot')
ax_inset.set_xlabel('X-axis')
ax_inset.set_ylabel('Y-axis')
show()
```

[ ]:
```
'''Contour plots (sometimes called Level Plots) are a way to show a
three-dimensional surface on a two-dimensional plane. It graphs two
predictor variables X Y on the y-axis and a response variable Z as
contours. These contours are sometimes called the z-slices
or the iso-response values. A contour plot is appropriate if you
want to see how alue Z changes as a function of two inputs X and Y,
such that Z = f(X,Y). A contour line or isoline of a function of two
variables is a curve along which the function has a constant value.
The independent variables x and y are usually restricted to a regular
grid called meshgrid. The numpy.meshgrid creates a rectangular grid
out of an array of x values and an array of y values.
Matplotlib API contains contour() and contourf() functions that draw
contour lines and filled contours, respectively. Both functions need
three parameters x,y and z.'''
```

[ ]:
```
#Contour Plot:
xlist = linspace(-3.0, 3.0, 100)
ylist = linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = sqrt(X**2 + Y**2)
fig,ax=subplots(figsize=(5,5))
ax.contour(X,Y,Z,40)
ax.set_xlabel('xlist')
ax.set_ylabel('ylist')
ax.set_title('Contour Plot')
show()
```

[ ]:
```
'''Even though Matplotlib was initially designed with only two-dimensional
plotting in mind,some three-dimensional plotting utilities were built on
```

```
top of Matplotlib's two-dimensional display in later versions, to provide
a set of tools for three-dimensional data visualization.
Three-dimensional plots are enabled by importing the mplot3d toolkit,
included with the Matplotlib package.
Furthermore for 3D Plotting wee have to create objects figure and
axes(with 3d Projection) for object oriented working'''
```

[ ]:
```
#3D line Plot:
from mpl_toolkits import mplot3d
from numpy import *
from matplotlib.pyplot import *
fig=figure(figsize=(5,5),facecolor='c')
ax = fig.add_axes((0.5,0.5,1,1),projection='3d')
z = linspace(0, 1, 100)
x = z * np.sin(20 * z)
y = z * np.cos(20 * z)
ax.plot3D(x, y, z,'g+')
ax.set_title('3D line plot')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
show()
```

[ ]:
```
#3D Contour Plot:
def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))
x = linspace(-6, 6, 30)
y = linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = figure(figsize=(12,12), facecolor='w')
ax = fig.add_axes((0,0,0.5,0.5),projection='3d', facecolor='None') # Projection␣
 ↪keyword is important for 3d plots
ax.contour3D(X, Y, Z, 50, cmap='viridis') #cmap represents color maps
ax.set_xlabel('x', fontsize=20)
ax.set_ylabel('y', fontsize=20)
ax.set_zlabel('z', fontsize=20)
ax.set_title('3D contour',fontsize=20)
savefig("3D Contourplot.pdf", facecolor="w",transparent=True, dpi=200)
show()
```

[ ]:
```
'''Plots such as contour3d, SurfacePlot reqquire meshgrid for plotting'''
```

[ ]:
```
#3D Surface Plot:
def f(x, y):
    return np.sin(np.sqrt(x**2  + y**2 ))
x = np.linspace(-6, 6, 30)
```

```
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
fig = figure(figsize=(15,15), facecolor='w')
ax = fig.add_axes((0,0.5,1,1),projection='3d', facecolor='None')
ax.plot_surface(X, Y, Z, cmap='rainbow') #cmap represents color maps
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('3D Surface')
show()
```

[ ]:
```
#See CheatSheet by DataCamp: https://images.datacamp.com/image/upload/
↪v1676360378/Marketing/Blog/Matplotlib_Cheat_Sheet.pdf
```

Table 4-2. *Summary of Selected Font Properties and the Corresponding Keyword Arguments*

Argument	Description
`fontsize`	The size of the font, in points.
`family or fontname`	The font type.
`backgroundcolor`	Color specification for the background color of the text label.
`color`	Color specification for the font color.
`alpha`	Transparency of the font color.
`rotation`	Rotation angle of the text label.

Table 4-1. *Basic Line Properties and Their Corresponding Argument Names for Use with the Matplotlib Plotting Methods*

Argument	Example Values	Description
`color`	A color specification can be a string with a color name, such as "red," "blue," etc., or a RGB color code on the form "#aabbcc."	A color specification.
`alpha`	Float number between 0.0 (completely transparent) and 1.0 (completely opaque).	The amount of transparency.
`linewidth, lw`	Float number.	The width of a line.
`linestyle, ls`	"-" – solid "--" – dashed ":" – dotted ".-" – dash-dotted	The style of the line, i.e., whether the line is to be drawn as a solid line or if it should be, for example, dotted or dashed.
`marker`	+, o, * = cross, circle, star s = square . = small dot 1, 2, 3, 4, ... = triangle-shaped symbols with different angles.	Each data point, whether or not it is connected with adjacent data points, can be represented with a marker symbol as specified with this argument.
`markersize`	Float number.	The marker size.
`markerfacecolor`	Color specification (see in the preceding text).	The fill color for the marker.
`markeredgewidth`	Float number.	The line width of the marker edge.
`markeredgecolor`	Color specification (see above).	The marker edge color.

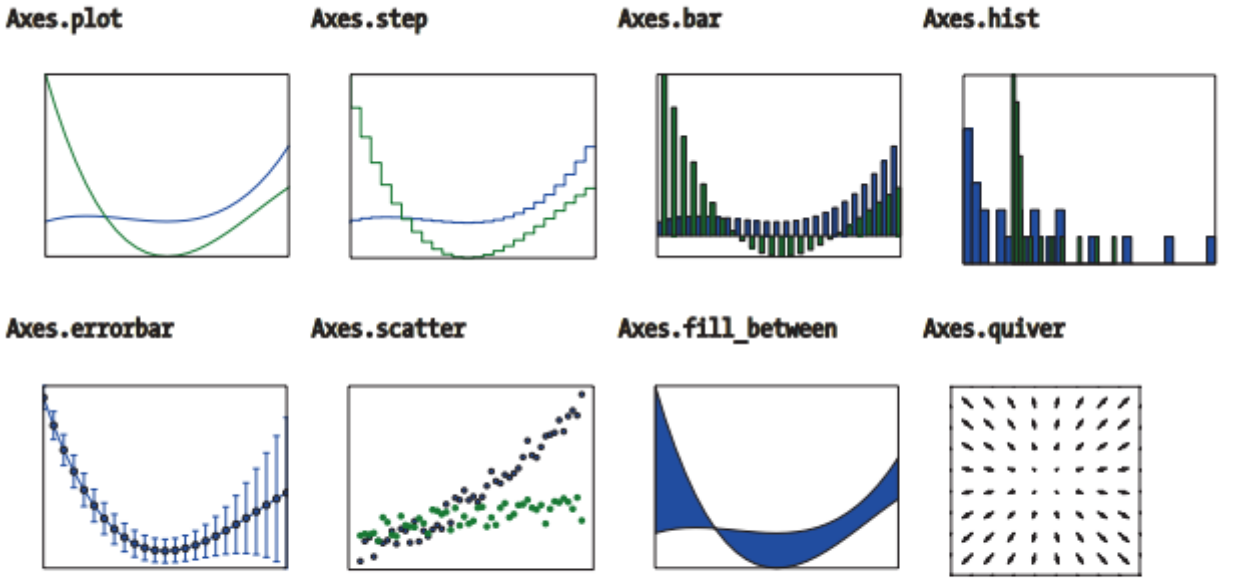


Figure 4-6. *Overview of selected 2D graph types. The name of the* `Axes` *method for generating each type of graph is shown together with the corresponding graph.*