Plotting Data

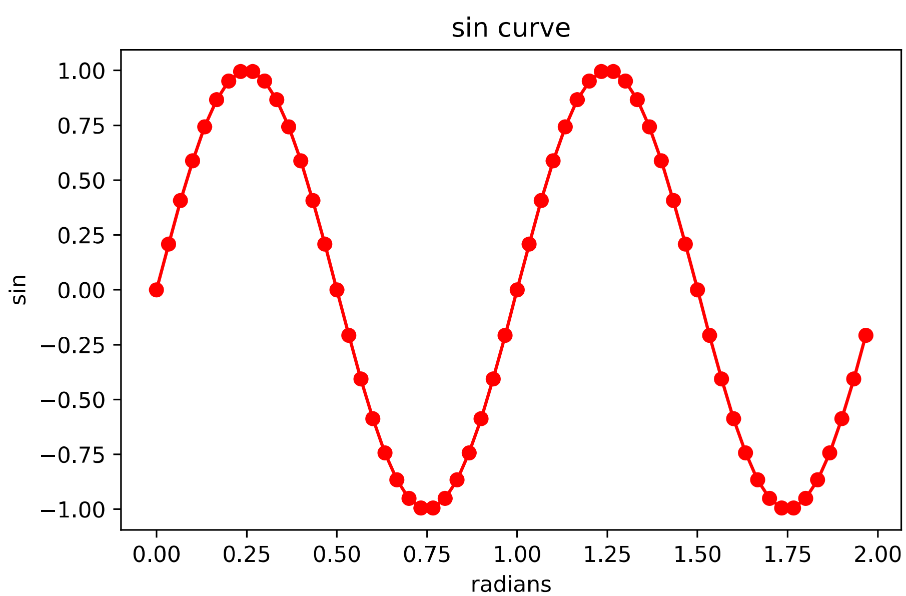
The `Matplotlib` module is based on the plotting package provided by matlab™. This module can be used to create simple data plots to plots ready for journal publication. A `Matplotlib` tutorial is found [here.](#) The faq are [here](#).

The packages web site is https://matplotlib.org/.

A simple example is:

sin_plot.py

```
import matplotlib.pyplot as plt
import math

# compute the sin function for 0 to 4 PI
t = [ t/30.0 for t in range( 60 ) ]
s = [ math.sin( x * math.pi * 2) for x in t ]

plt.figure(1, figsize=(6.0,4.0), dpi=100) # figsize is in inches
plt.subplot(111) # or subplot(1,1,1), optional
plt.xlabel('radians')
plt.ylabel('sin')
plt.title('sin curve')

# plot the sin curve
# r : red,
# - : lines between markers
# o : markers are circles
plt.plot( t, s, "r-o")

plt.tight_layout()

plt.savefig('sin_plot.png') # save the 600x400 image as a PNG file
plt.savefig('sin_plot.ps') # save in postscript
plt.savefig('sin_plot.svg') # save in svg

# alternatively the figure can be shown on the screen with
#plt.show()
```



Matplotlib terms

figure
: the output plot, it can contain subplots.

axis
: a rectangular region containing the plotted data, a `subplot` creates an axis.

xlabel
: the text that appears along the x-axis

ylabel
: the text that appears along the y-axis

title
: the title for the axis.

xlim
: the limits for the x-axis from low to high.

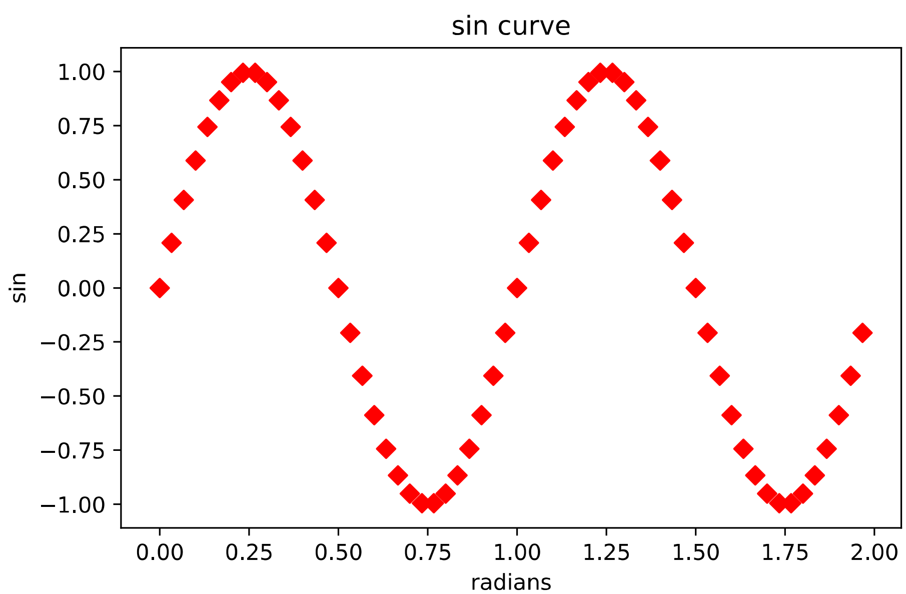
ylim
: the limits for the y-axis from low to high.

Scatter Plot

A scatter plot is done with:

sin_scatter.py

```
import matplotlib.pyplot as plt
import math

# compute the sin function for 0 to 4 PI
t = [ t/30.0 for t in range( 60 ) ]
s = [ math.sin( x * math.pi * 2) for x in t ]

#plt.figure(1, figsize=(6.0,4.0), dpi=100) # figsize is in inches
#plt.subplot(111) # or subplot(1,1,1), this is optional
plt.xlabel('radians')
plt.ylabel('sin')
plt.title('scatter sin curve')

# a scatter plot
plt.scatter(t, s, c='r', marker='D')
# or # plt.scatter(x=t, y=s)

plt.tight_layout()

plt.savefig('sin_scatter.svg') # save in svg

#plt.show()
```

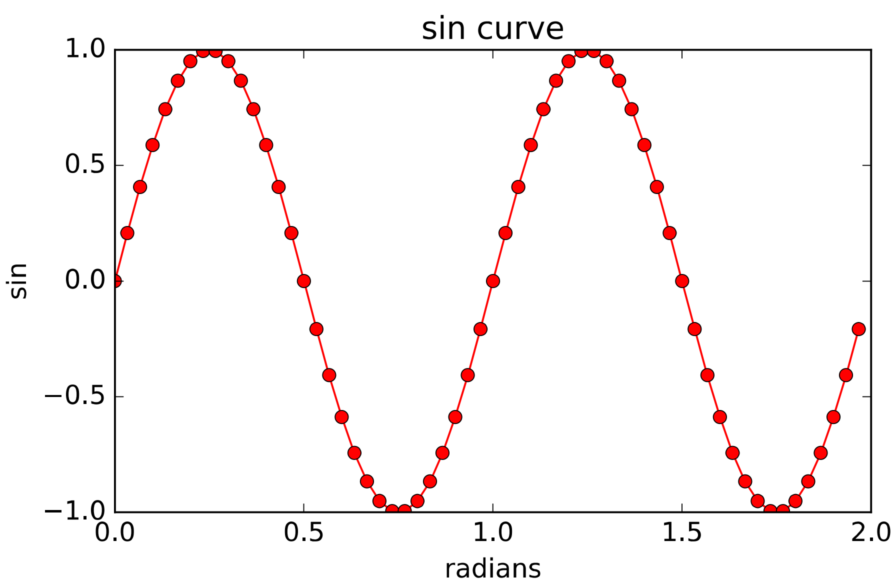


Plotting Data Using OO

The previous example use python code to mimic the matlab interface. Plotting can be done using the classes defined by this module in a more object-oriented fashion.

sin_plot_oo.py

```
# switch between backends
#from matplotlib.backends.backend_ps import FigureCanvasPS as FigureCanvas
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
import math

# compute the sin function for 0 to 4 PI
t = [ t/30.0 for t in range( 60 ) ]
s = [ math.sin( x * math.pi * 2) for x in t ]

fig = Figure( figsize=(6.0,4.0), dpi=100) # figsize is in inches
canvas = FigureCanvas(fig)
ax = fig.add_subplot(111)
ax.set_xlabel('radians')
ax.set_ylabel('sin')
ax.set_title('sin curve')
ax.plot( t, s, "r-o") # plot the sin curve

fig.tight_layout()
fig.savefig('sin_plot_oo.svg')
```



Symbols, Colours, Line Styles

In addition to multiple subplots, each data plot can use different colour, line styles, and markers in the plot.

Subplots are arranged in a grid. A subplot is created/referenced with:

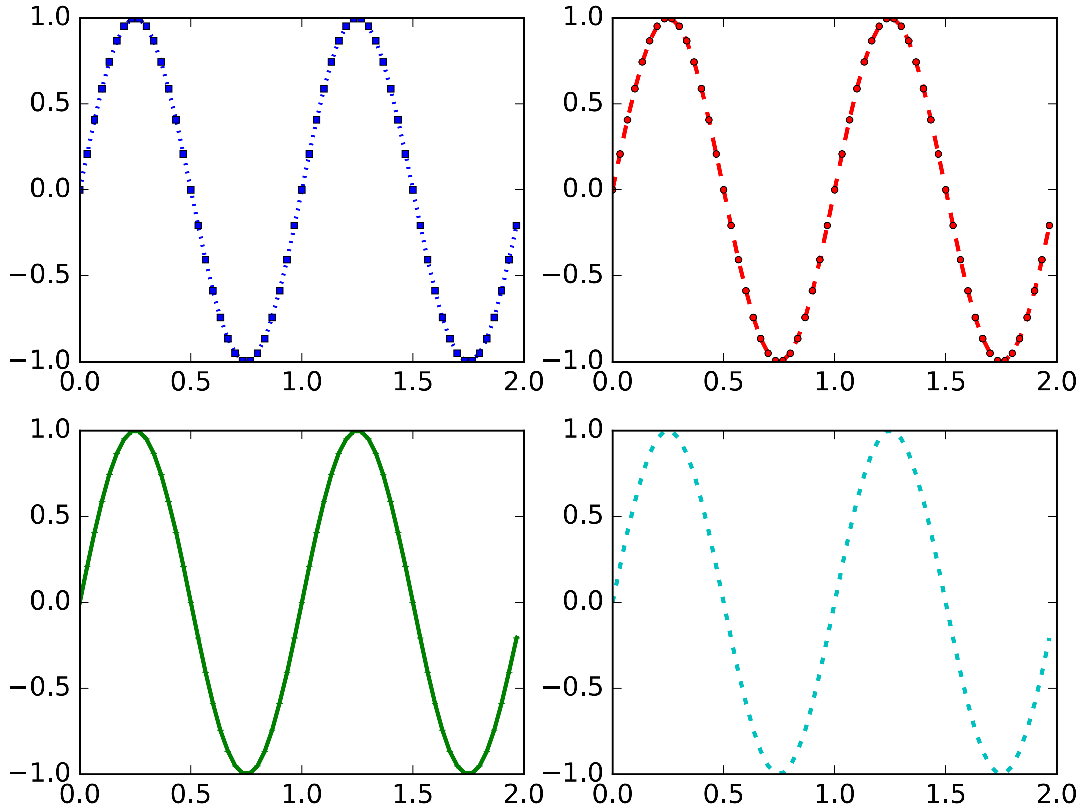
```
subplot(nrows, ncols, plot_number)
```

`plot_number` identifies a particular subplot in the nrows by ncols matrix of subplots. 1 is mat[0][0], 2 is mat[0][1], ..., N is [nrows-1][ncols-1].

multiple_sin_plot.py

```
import matplotlib.pyplot as plt
import math

# compute the sin function for 0 to 4 PI
t = [ t/30.0 for t in range( 60 ) ]
s = [ math.sin( x * math.pi * 2) for x in t ]

plt.figure(1, figsize=(8.0,6.0), dpi=100) # figsize is in inches

# two by two grid
```

```
for index,style in enumerate( ["b:s", "r--o", "g-+", "c-." ] ) :
    plt.subplot(2,2, index+1 )
    plt.plot( t, s, style, markersize=3.0, linewidth=2.0 )

plt.savefig('multiple_sin_plot.svg')
```



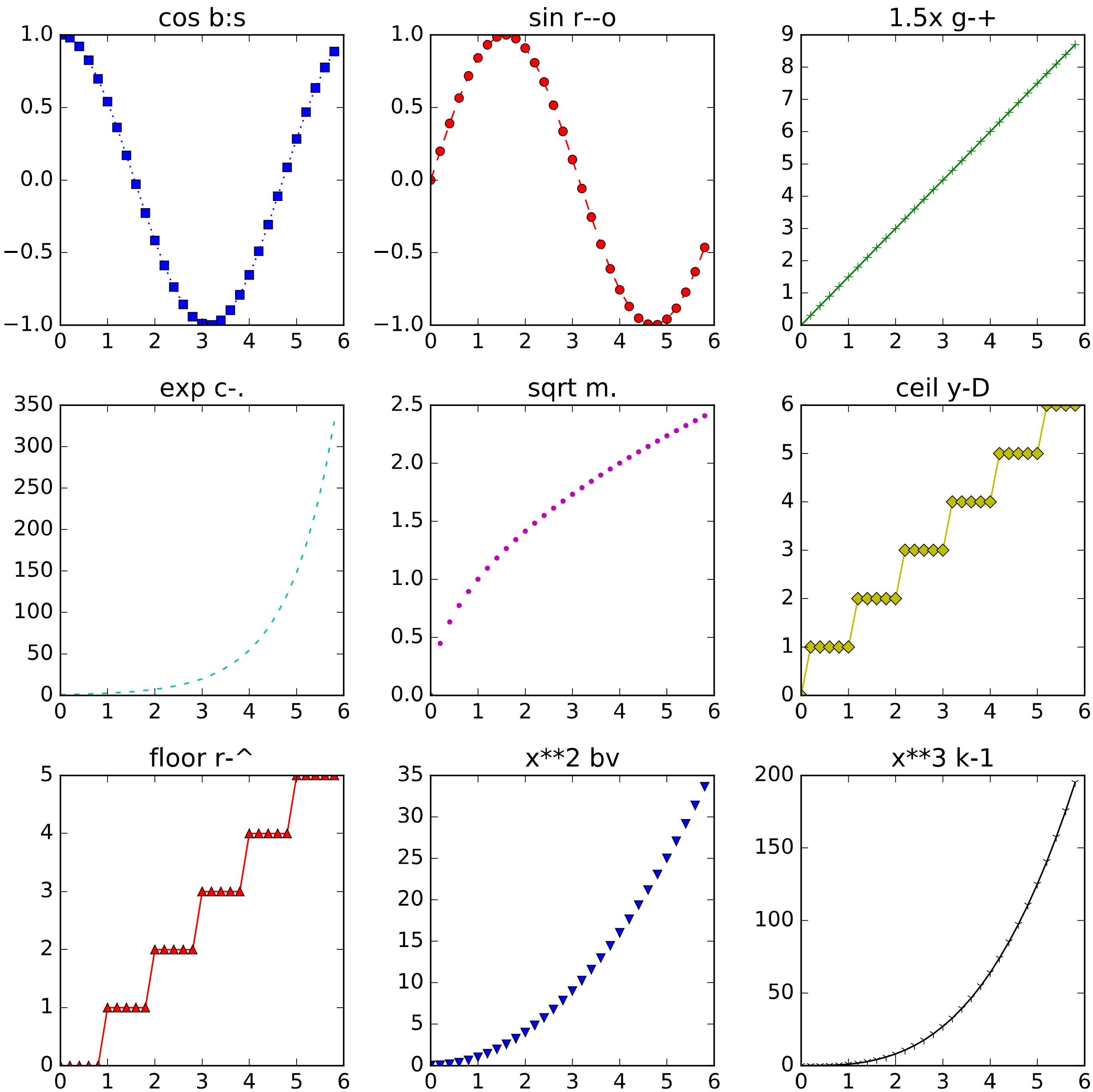
More Symbols, Colours, Line Styles

The following example uses the OO form of matplotlib to demonstrate multiple markers, line styles, and drawing colours.

multiple_fn_plot_oo.py

```
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
import math

t = [ t/5.0 for t in range( 30 ) ]

fig = Figure( figsize=(9.0,9.0), dpi=100)
canvas = FigureCanvas(fig)

# an assortment of styles
styles = [ "b:s", "r--o", "g-+",
           "c-.", "m.", "y-D",
           "r-^", "bv", "k-1" ]

functions = [ math.cos, math.sin, lambda x : 1.5*x,
              math.exp, math.sqrt, math.ceil,
              math.floor, lambda x : x**2, lambda x : x**3 ]

names = [ "cos", "sin", "1.5x",
          "exp", "sqrt", "ceil",
          "floor", "x**2", "x**3" ]

names = [ n + " " + styles[i] for i,n in enumerate(names) ]

plots_info = enumerate( zip(styles,functions,names) )
```

```
for index,(style,fn,name) in plots_info :
    ax = fig.add_subplot( 3, 3, index+1 )
    ax.set_title( name )
    f = [ fn( x ) for x in t ]
    ax.plot( t, f, style, markersize=5.0 )

fig.tight_layout()
fig.savefig('multiple_fn_plot_oo.svg')
```



Symbols, Colours, Line Styles Descriptions

Matplotlib defines the following drawing specifications:

symbols

. : 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

lines

- : solid line, -- : dashed line, -. : dash-dot line, : : dotted line

colours

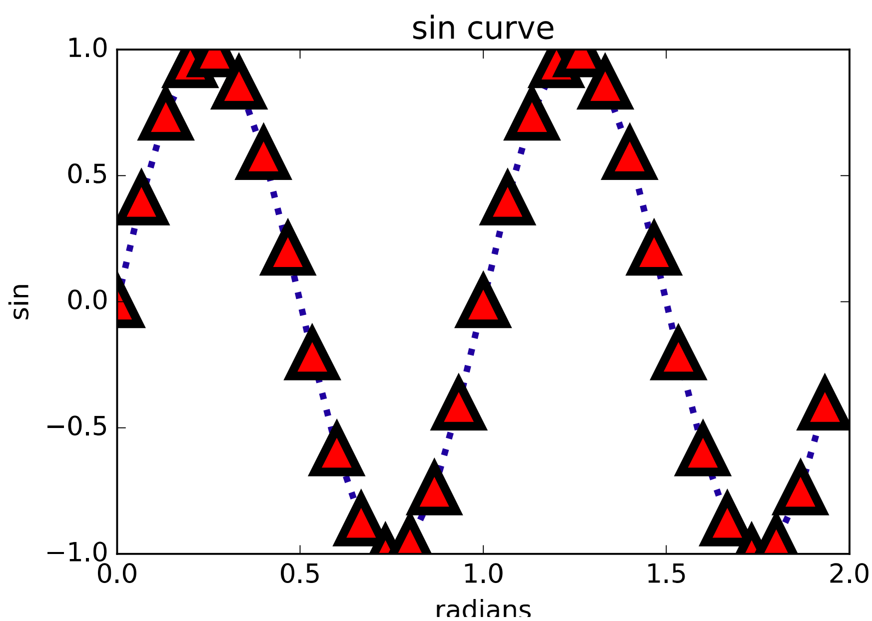
b : blue, g : green, r : red, c : cyan, m : magenta, y : yellow, k : black, w : white

Line properties and setting them

More rendering properties for lines and markers can be set with `setp`.

sin_plot_prop.py

```
import matplotlib.pyplot as plt
import math

# compute the sin function for 0 to 4 PI
t = [ 2*t/30.0 for t in range( 30 ) ]
s = [ math.sin( x * math.pi * 2) for x in t ]

plt.figure(1, figsize=(6.0,4.0), dpi=100) # figsize is in inches
plt.subplot(111)
plt.xlabel('radians')
plt.ylabel('sin')
plt.title('sin curve')
t = plt.plot( t, s )
# setp is set properties
plt.setp( t, color="#2000A0", linestyle="-.", linewidth=3.0)
plt.setp( t, marker="^", markersize=20.0,
         markeredgewidth=4.0,
         markeredgecolor="k", markerfacecolor="r" )

# linestyle or ls, linewidth or lw
# markeredgecolor or mec, markeredgewidth or mw
# markerfacecolor or mfc, markersize or ms
# linewidth and markersize are in points

plt.savefig('sin_plot_prop.svg')
```



Line properties

alpha
: The opacity of the line, varys from 0 to 1.

antialiased
: True for antialiasing, False for off.

color or c
: Color of line.

linestyle or ls
: One of `--` `:` `-.` `-` line styles.

linewidth or lw
: Width of line in points

marker
: Type of marker.

markeredgewidth
: Marker width in points.

markeredgecolor
: Color of marker edge.

markerfacecolor
: Color of marker body.

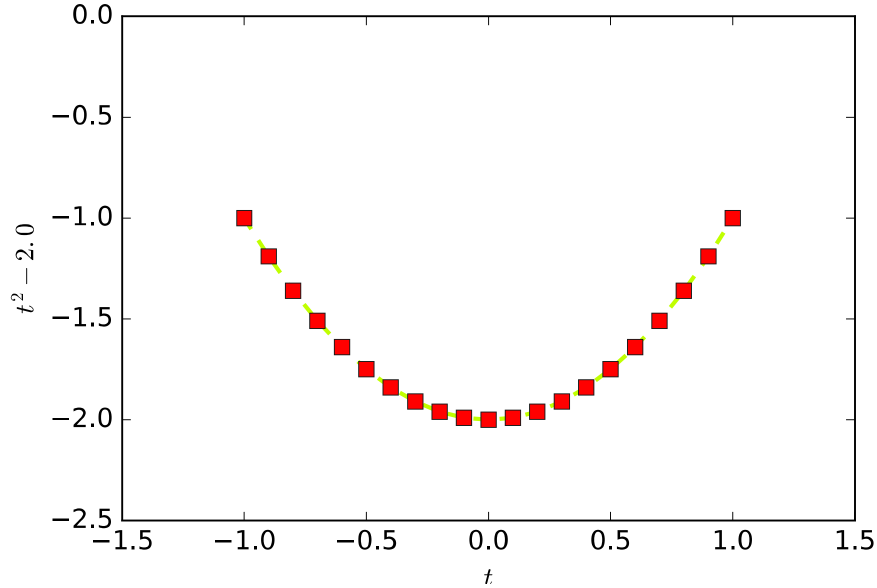
markersize or ms
: Size of marker in points.

Controlling the style of a single plot

The style of the line and markers produced by the `plot` command can be modified with the lines object returned from `plot` command. An example is:

line_styles.py

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

# compute the sin function for 0 to 4 PI
t = np.arange( -1.0, 1.1, 0.1 )
y = t**2 - 2.0

plt.figure(1, figsize=(6.0,4.0), dpi=100) # figsize is in inches
plt.xlabel('$t$')         # use tex formatting
plt.ylabel('$t^2 - 2.0$') # use tex formatting
lines = plt.plot( t, y )
line = lines[0]
plt.ylim(-2.5, 0.0 )
plt.xlim( -1.5, 1.5 )

line.set_linewidth( 2.0 )
line.set_color( '#C0FF00' )
line.set_linestyle( '--' )
line.set_marker('s')
line.set_markerfacecolor('red')
line.set_markeredgecolor( '0.1' ) # 10% grey
line.set_markersize( 7 ) # in pts (1/72 inch)

# other properties are: alpha, antialiased, label, data_clipping

plt.savefig('line_styles.svg')
#plt.show()
```

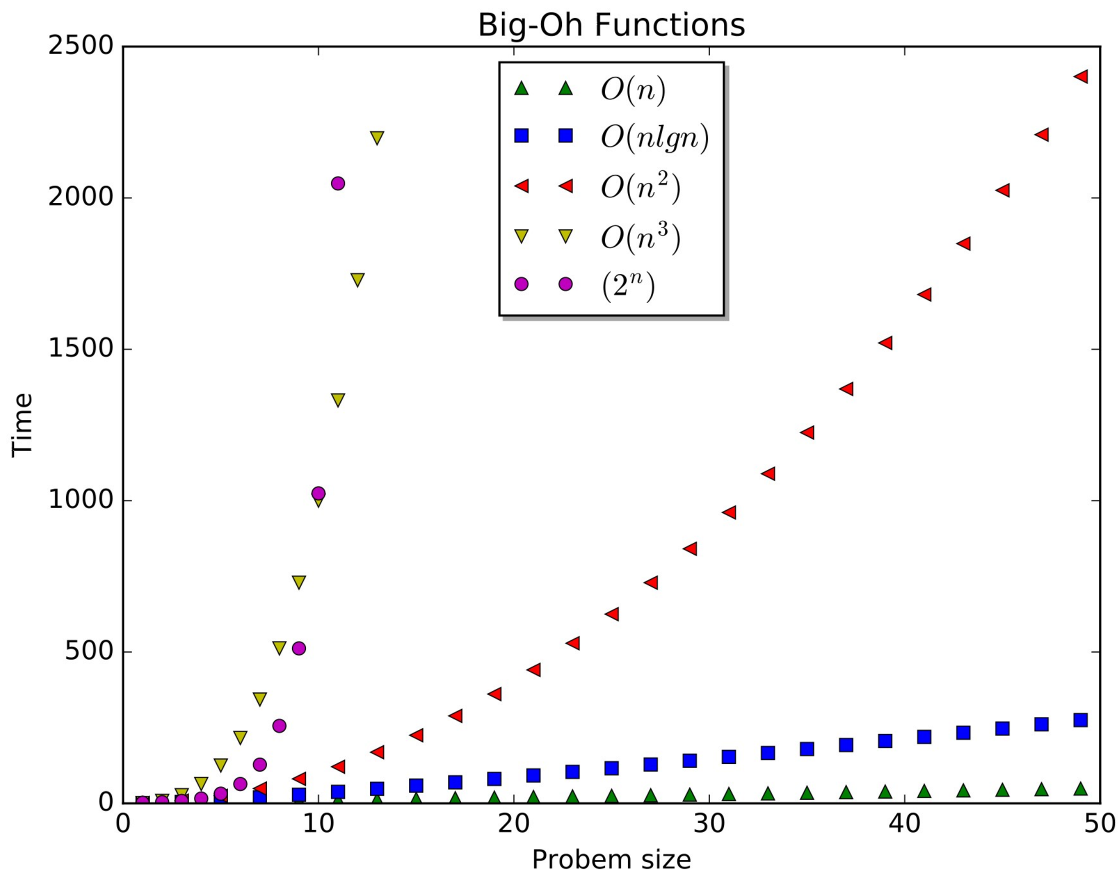


Creating a Plot of Time Complexity Functions

Time complexity is a measure of the amount of effort required by an algorithm. Plotting the common time complexity function shows the relative time required for each time complexity. A python program to create this graph is:

tc.py

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

def nlgn( x ) :
    return  x*np.log(x)/math.log(2)

t = np.arange(1., 50., 2.)
# limit the range of O(n**3) and  O(2**n) to fit with
# the other time complexity functions
t3 = np.arange(1., 15., 1.)
pt = np.arange(1., 12., 1.)

plt.xlabel('Probem size')
plt.ylabel('Time')
plt.title('Big-Oh Functions')

plt.plot(t, t, 'g^')             # O(n)
plt.plot( t, nlgn(t) , 'bs')     # O(n lg n)
plt.plot( t, t*t, 'r<' )         # O(n**2)
plt.plot( t3, t3**3, 'yv' ) # O(n**3)
plt.plot( pt, np.exp2(pt), 'mo' )  # O(2**n)
plt.ylim( 0, 2500 ) # limits the y-axis from 0 to 2500

# uses TEX markup
l = ['$O(n)$', '$O(n lg n)$','$O(n^2)$', '$O(n^3)$', '$(2^n)$' ]
plt.legend( l, shadow=True, loc='upper center')

plt.savefig('tc.svg')
```



Vector functions from matplotlib

Matplotlib uses numpy to provide functions that operate on vectors. for example:

```
>>> import numpy as np
>>> import math
>>>
>>> t = np.arange(1.0, 8.0, 1.0) # works on floats
>>> t
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.])
>>> type(t[0])
<class 'numpy.float64'>
>>> 2*t + 1
array([  3.,   5.,   7.,   9.,  11.,  13.,  15.])
>>> t + t + 1
array([  3.,   5.,   7.,   9.,  11.,  13.,  15.])
>>> 2.0**10.0
1024.0
>>> 2.0**t
array([   2.,    4.,    8.,   16.,   32.,   64.,  128.])
>>> x = _
>>> np.log(x) / math.log(2) # log 2 (x) = ln(x) / ln(2)
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.])
```

These functions that accept arrays (vectors) as arguments enable the writing of compact code fragments without the use of loops.

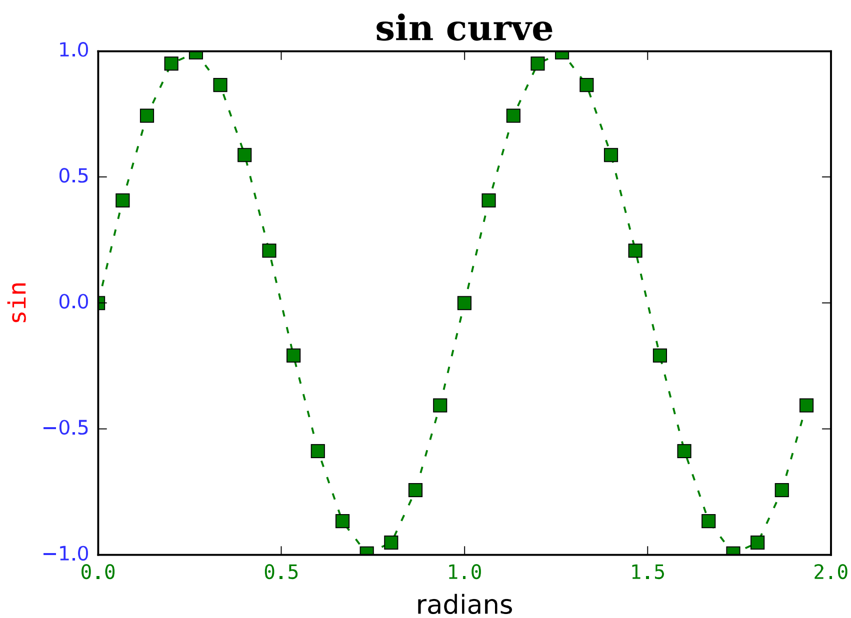
Changing Font Properties

The fonts used for the text in the figure can be controlled by:

sin_plot_font.py

```
import matplotlib.pyplot as plt
```

```
import math

# compute the sin function for 0 to 4 PI
t = [ 2*t/30.0 for t in range( 30 ) ]
s = [ math.sin( x * math.pi * 2) for x in t ]

plt.figure(1, figsize=(6.0,4.0), dpi=100) # figsize is in inches
ax = plt.subplot(111)
# get the lables so that the font properties can be set
labels = plt.getp(ax, 'xticklabels')
plt.setp(labels, color='g', fontsize=9, fontname="monospace" )
labels = plt.getp(ax, 'yticklabels')
plt.setp(labels, color='#3030FF', fontsize=9, fontname="sans" )

# set font propeties with arguments
plt.xlabel('radians', fontname="sans", fontsize=12)
# alternatively, setp can be used
yl = plt.ylabel('sin')
plt.setp( yl, color="red", fontname="monospace", fontsize=11)
plt.title('sin curve', fontname="serif", fontsize=16, fontweight="bold")
plt.plot( t, s, "g-.s" )

plt.savefig('sin_plot_font.svg')
```



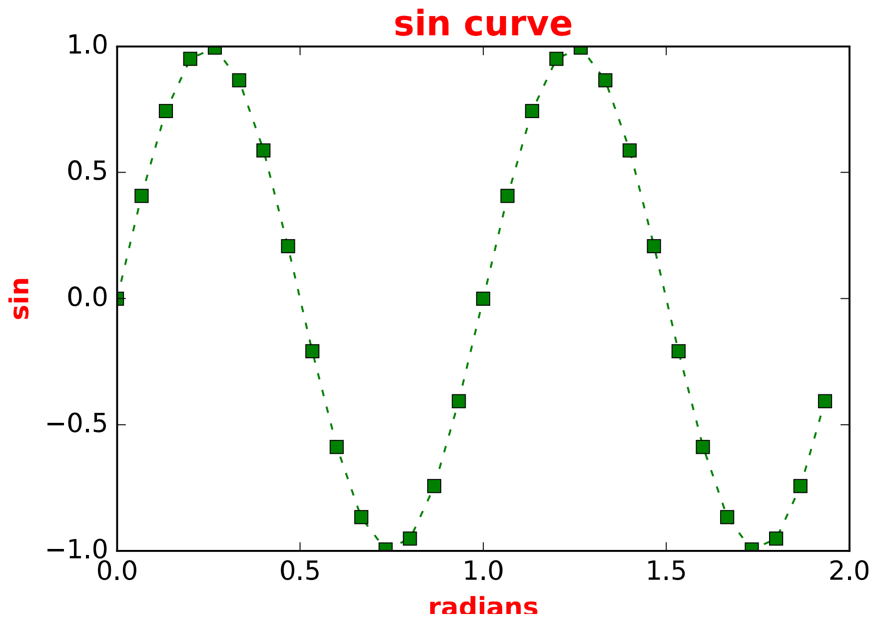
Font Properties with Dictionaries

Common font properties can be set with a dictionary.

sin_plot_font_dict.py

```
import matplotlib.pyplot as plt
import math

# compute the sin function for 0 to 4 PI
t = [ 2*t/30.0 for t in range( 30 ) ]
s = [ math.sin( x * math.pi * 2) for x in t ]

# set font properties with dictionary
font = {'fontname'   : 'sans-serif',
        'color'      : 'r',
        'fontweight' : 'bold',
        'fontsize'   : 12}

plt.figure(1, figsize=(6.0,4.0), dpi=100 ) # figsize is in inches

plt.xlabel('radians', font )
plt.ylabel('sin', font)
# modify one of the font properties
plt.title('sin curve', font, fontsize=16)
```

```
plt.plot( t, s, "g-.s" )

plt.savefig('sin_plot_font_dict.svg')
```



Font Properties

alpha
: The opacity of the text, varys from 0 to 1.

color
: Color of the text.

fontangle
: italic or normal or oblique

fontname
: monospace, sans, sans serif, sans-serif, serif, ...

fontsize
: Font size is specified in points.

fontweight
: normal or bold or light

horizontalalignment
: left or center or right

rotation
: horizontal or vertical

verticalalignment
: bottom or center or top

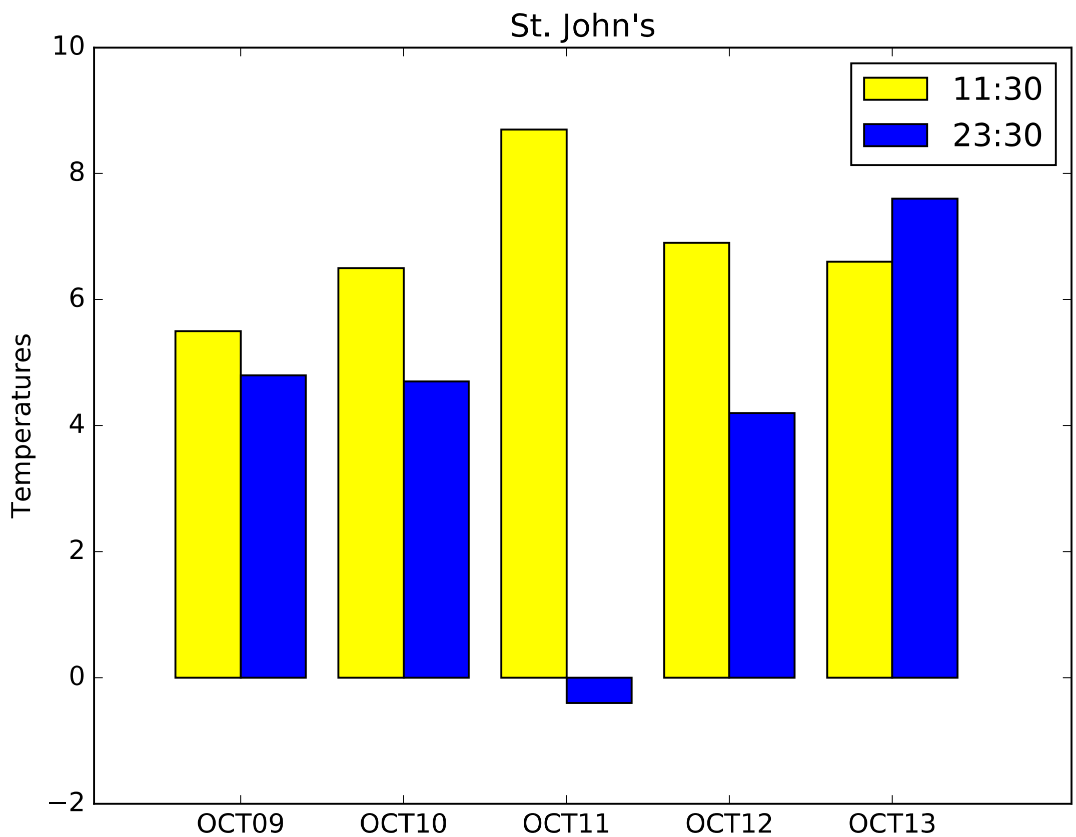
Temperature Bar Charts

Bar charts can be produced by:

stj_bar.py

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

# temperatures at St. John's
# october 9th, 2007 to october 13th, 2007
stjTemp11_30 = [ 5.5, 6.5, 8.7, 6.9, 6.6 ]
stjTemp23_30 = [ 4.8, 4.7, -0.4, 4.2, 7.6 ]
N = len(stjTemp11_30)

index = np.arange(N)
width = 0.40        # the width of the bars
plt.bar(index, stjTemp11_30, width, color='yellow', label='11:30')
```

```
plt.bar(index+width, stjTemp23_30, width, color='blue', label='23:30')
plt.xlim(-0.5,N+0.5) # leave space

plt.ylabel('Temperatures')
plt.title("St. John's")
plt.xticks(index+width, ['OCT%02d'%d for d in range(9,14)])

plt.legend()

plt.savefig('stj_bar.svg')
```



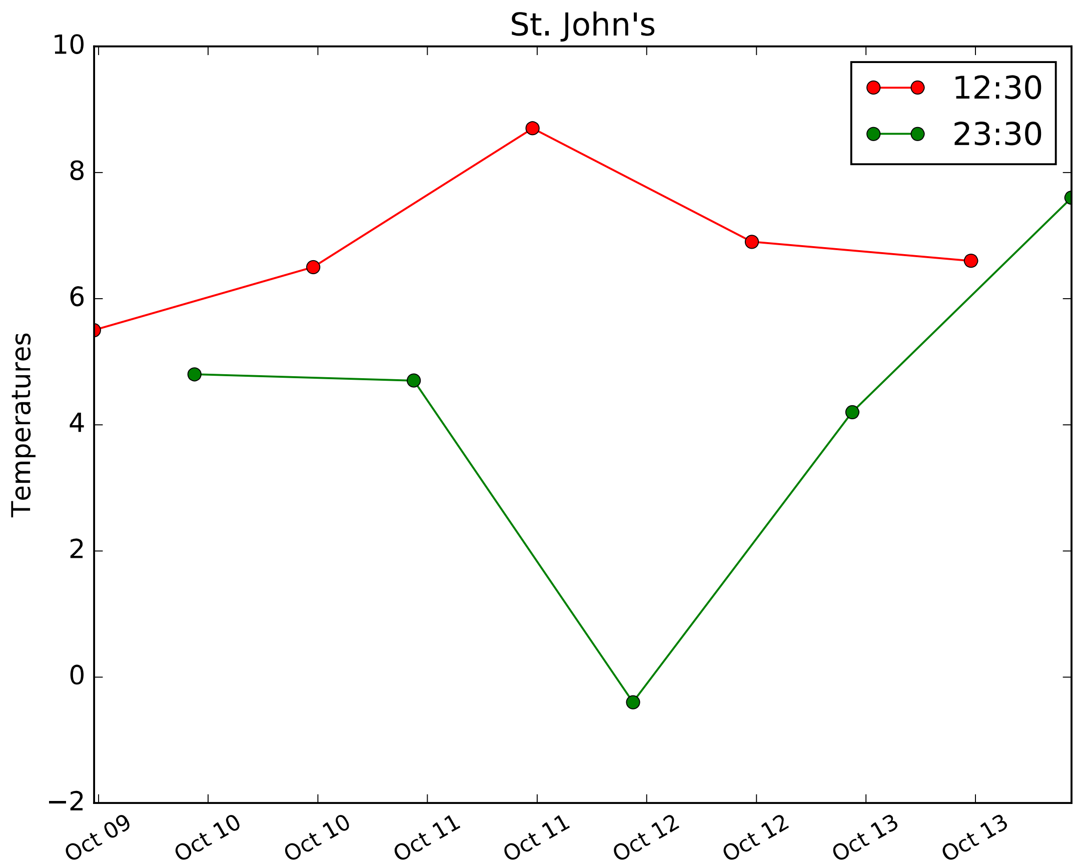
Temperature Line Charts with plot_date

The temperature can be plotted with markers and lines.

stj_line.py

```
import matplotlib.pyplot as plt
from matplotlib.dates import DateFormatter, date2num
from datetime import datetime

# temperatures at St. John's
# october 9th, 2007 to october 13th, 2007
stjTemp12_30 = [ 5.5, 6.5, 8.7, 6.9, 6.6 ]
stjTemp23_30 = [ 4.8, 4.7, -0.4, 4.2, 7.6 ]
dt1 = [ datetime(2007,10,day,12,30) for day in range(9,14) ]
# convert to seconds from epoch
dt1 = date2num( dt1 )

dt2 = [ datetime(2007,10,day,23,30) for day in range(9,14) ]
# convert to seconds from epoch
dt2 = date2num( dt2 )

ax = plt.subplot(111)
ax.xaxis.set_major_formatter( DateFormatter('%b %d') )
p1 = plt.plot_date( dt1, stjTemp12_30, "r-o", label='12:30' )
p2 = plt.plot_date( dt2, stjTemp23_30, "g-o", label='23:30' )
```

```
plt.ylabel('Temperatures')
plt.title("St. John's")
labels = ax.get_xticklabels()
plt.setp(labels, rotation=30, fontsize=10)

plt.legend()

plt.savefig('stj_line.svg')
```

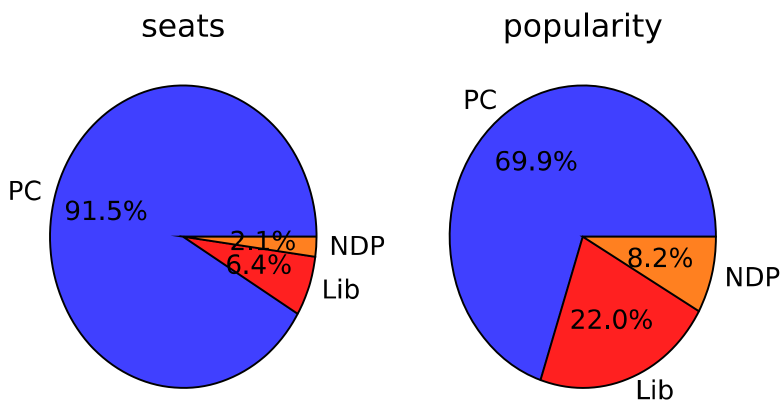


Pie charts

Matplotlib can also create pie charts.

elec.py

```
import matplotlib.pyplot as plt

plt.figure(1, figsize=(6,3), dpi=100)

labels = ['PC', 'Lib', 'NDP']
colors = [ '#4040ff', '#ff2020', '#ff8020' ]
seats = [ 43, 3, 1 ]
pop = [ 70, 22, 8.2 ]

plt.figure(1)
plt.subplot(121)
plt.pie(seats, labels=labels, colors=colors, autopct='%1.1f%%')
plt.title('seats')

plt.subplot(122)
plt.pie(pop, labels=labels, colors=colors, autopct='%1.1f%%')
plt.title('popularity')

plt.savefig('elec.svg')
```

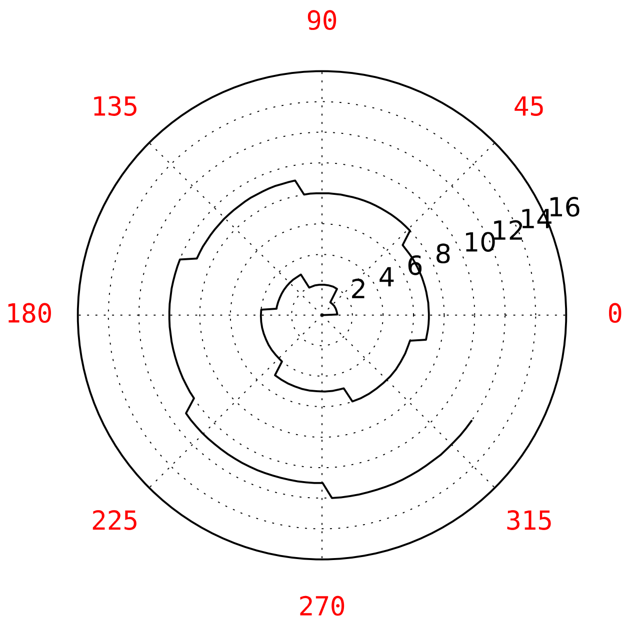


Polar charts

A simple polar chart is produced by:

polar_plot.py

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

th = np.arange(0.0, 12.0, 0.05)
r = np.ceil( th )

plt.figure(1, figsize=(4.0,4.0), dpi=100) # figsize is in inches

plt.polar( th, r, "k" )
# hack to changes radius
plt.polar( [0.1, 0.1], [14.0, 15.0] , "w")

# set the lables on the outer circle
angles = range( 0, 360, 45 )
labels = [ str(a) for a in angles ]
# frac specifies that position of the labels
lines, labs = plt.thetagrids( angles, labels, frac=1.2)
plt.setp( labs, color="red", fontname="monospace", fontsize=12)

# XXX
plt.savefig('polar_plot.svg')

#plt.show()
```



90
45
135
180
0
225
315
270
2
4
6
8
10
12
14
16