Python and Statistics for Matlab Users

Session 3

Nicolas Barrier

OT-Med Labex

19 septembre 2016

Introduction

Graphical representations are achieved using the Matplotlib library.

```
import pylab as plt
import matplotlib as mp
```

Many examples are provided in the gallery

Table of Contents

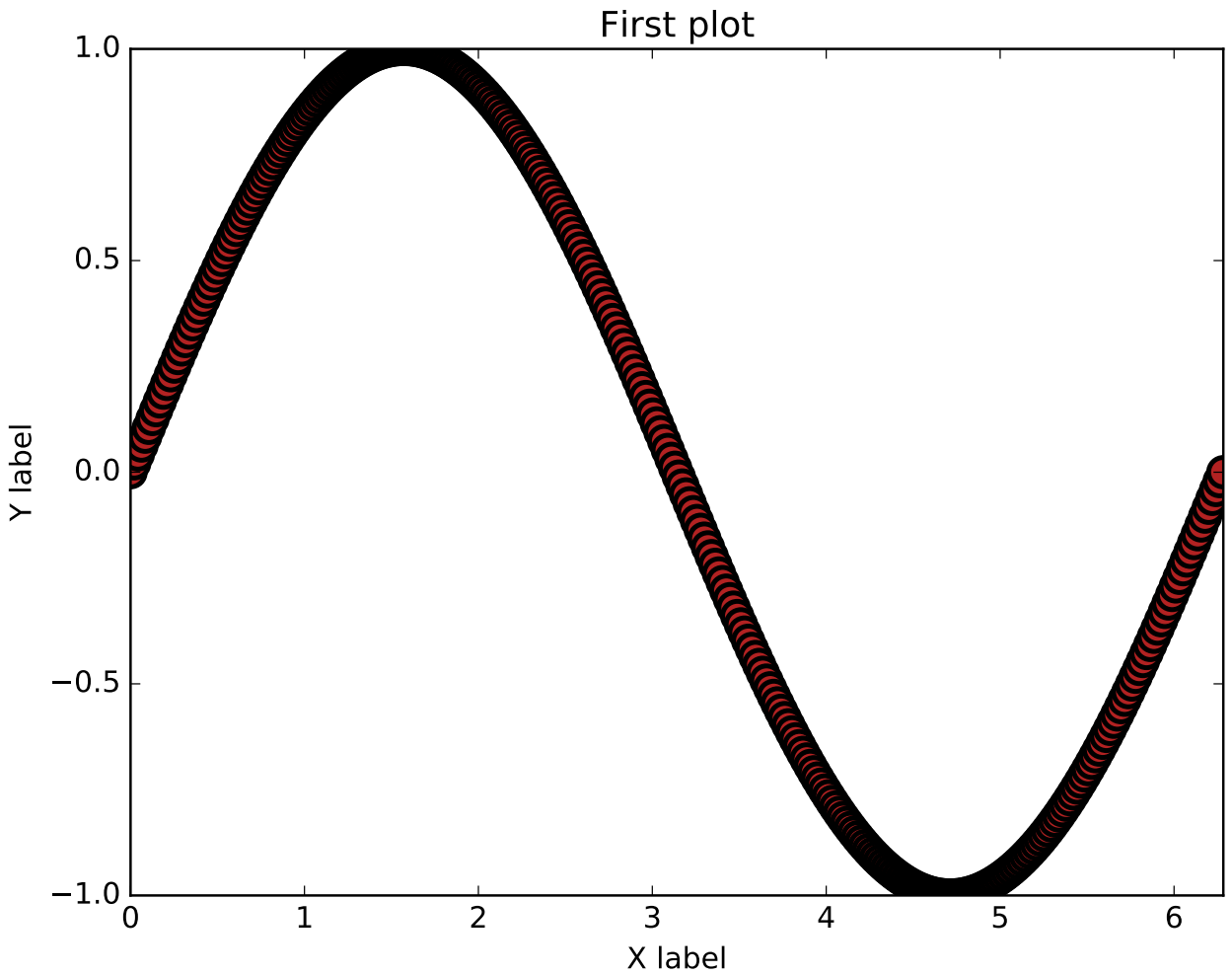
Data for XY plots

```
import pylab as plt
import matplotlib as mp
import numpy as np

x = np.linspace(0, 2*np.pi, 30)
y = np.sin(x)
z = np.cos(x)
t = np.tan(x)
```

Data for XY plots

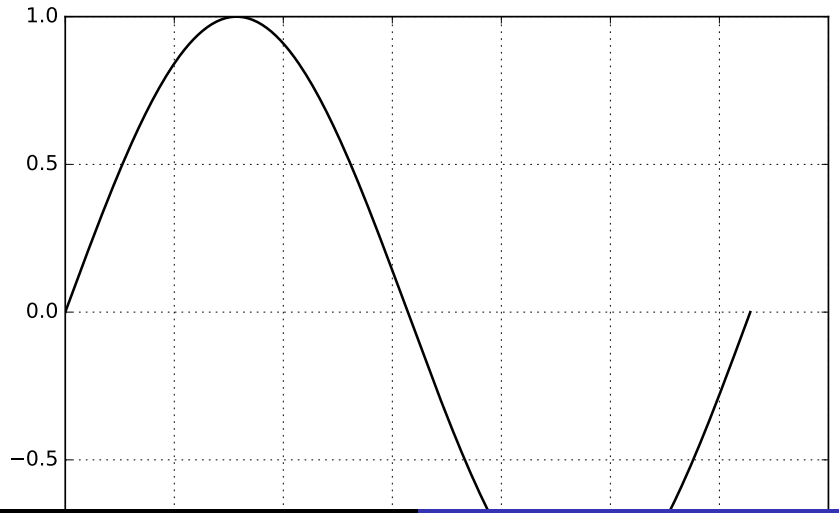
```
fig = plt.figure(figsize=(10, 4)) #initialize figure
ax = plt.gca() # initialize axes (here, not necessary)

plt.plot(x, y, color='Plum', linestyle='-', linewidth=2,
     marker='o', markeredgewidth=2, markerfacecolor='FireBrick',
     markeredgecolor='black', markersize=12)

plt.title('First plot')
plt.xlabel('X label')
plt.ylabel('Y label')

plt.xlim(x.min(), x.max()) # set the limits of the x-axis
plt.ylim(y.min(), y.max()) # set the limits of the y-axis

plt.savefig('figs/xy.pdf', bbox_inches='tight') # save the figure
# tight: remove the white spaces around the figure
# plt.show() display the figure but freezes the pgm
```

Data for XY plots



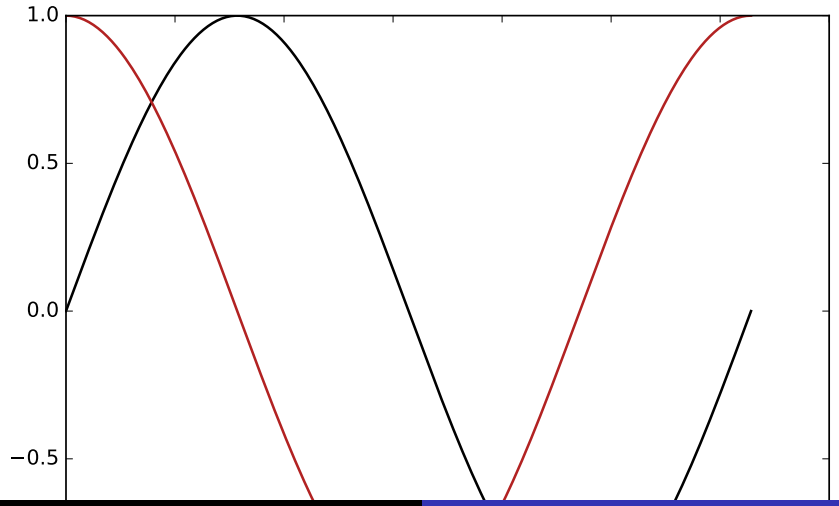
Setting ticks and ticklabels

```
ax = plt.gca() # initialize axes
plt.plot(x, y)

# location of the ticks and ticklabels
xticks = np.arange(0, 2*np.pi, np.pi/2)
xticklabels = ["$0$", r"$\frac{\pi}{2}$",
               "$\pi$", r"$\frac{3\pi}{2}$"]
ax.set_xticks(xticks)
ax.set_xticklabels(xticklabels, rotation=45,
                   ha='right', fontsize=20)
```



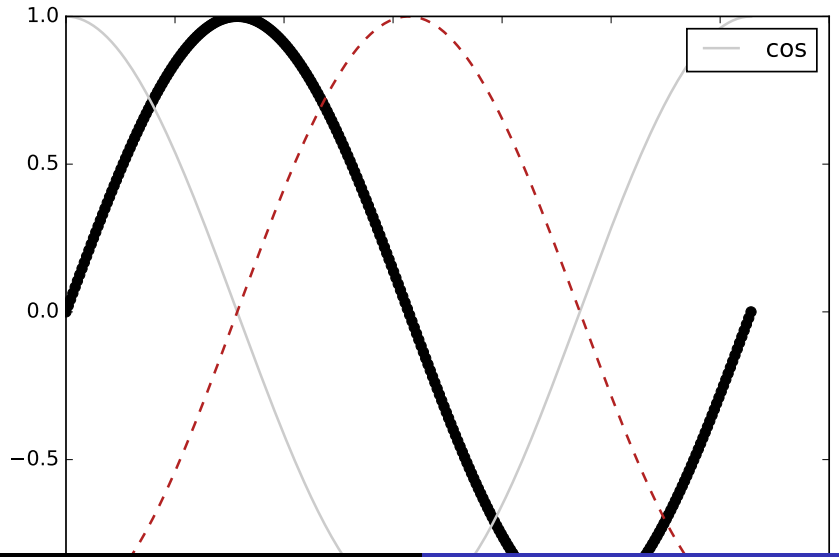
Multiline plot + legend

```
l0 = plt.plot(x, y, label='sin')
l1 = plt.plot(x, z, label='cos')

# set the legend font size
prop = mp.font_manager.FontProperties(size=15)

# add the legend. loc=0 => best location
leg = plt.legend(prop=prop, loc=0, ncol=1)
# equivalent to leg = plt.legend([l0[0], l1[0]],['sin', 'cos'])
```



Mulitline plot + separated legend

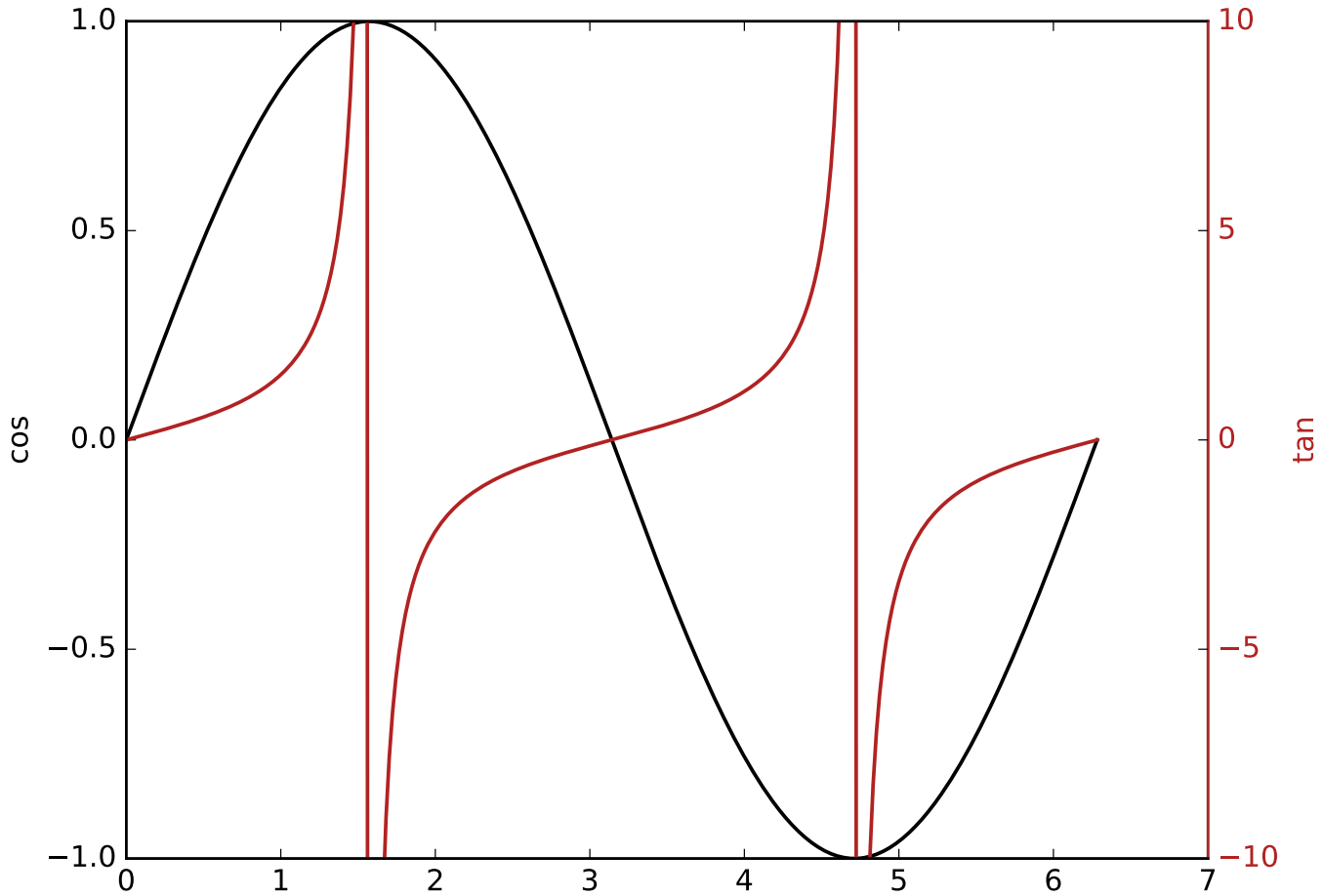
```
l0 = plt.plot(x, y)
l1 = plt.plot(x, z)

leg1 = plt.legend([l0[0]], ['sin'], loc=2)
ax.add_artist(leg1) # add 1st legend to the axes

leg2 = plt.legend([l1[0]], ['cos'], loc=1)
# second legend is added automatically
```



Shared axes

```
ax1 = plt.gca() # initialise first axes
plt.plot(x, y, 'k')
ax1.set_ylabel('cos', color='k')

# create axes that share the x-axis of ax1
ax2 = ax1.twinx()  # ax1.twiny() for shared y-ax
plt.plot(x, t, color='FireBrick')
ax2.set_ylim(-10, 10)

# change the color of ylabel, yticklabels and axes spine
plt.setp(ax2.get_yticklabels(), color='FireBrick')
ax2.set_ylabel('tan', color='FireBrick')
ax2.spines['right'].set_color('FireBrick')
```

Shared axes



Filled plots

```
# fill positive values
plt.fill_between(x, 0, y, color='r', where=y>0,
                 interpolate=True, edgecolor='k')
# fill negative values
plt.fill_between(x, 0, y, color='b', where=y<0,
                 interpolate=True, edgecolor='k')
```

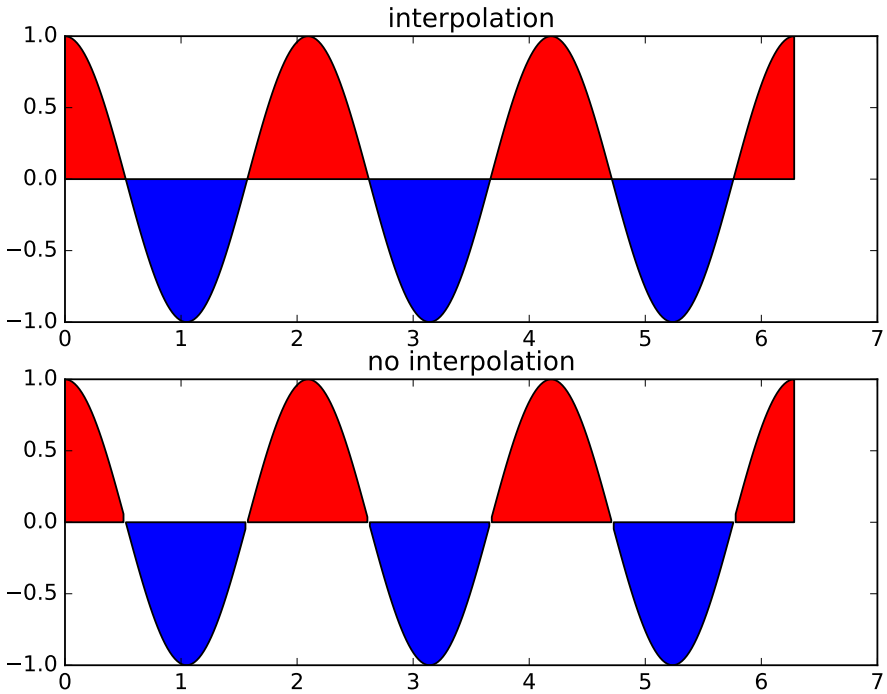


Table of Contents

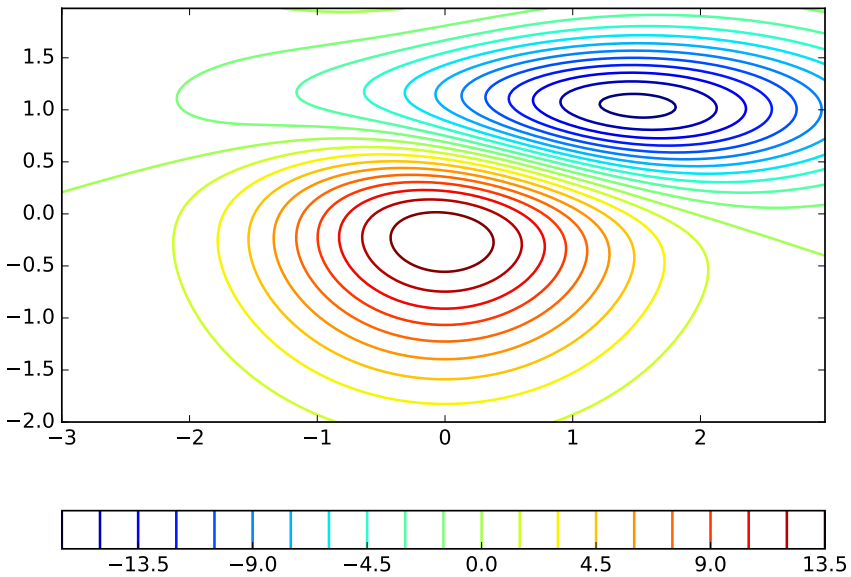
Data for contour plots

Adapted from Matplotlib contour demo :

```
import pylab as plt
import matplotlib as mp
import numpy as np

delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
xx, yy = np.meshgrid(x, y)
zz1 = mlab.bivariate_normal(xx, yy, 1.0, 1.0, 0.0, 0.0)
zz2 = mlab.bivariate_normal(xx, yy, 1.5, 0.5, 1, 1)
# difference of Gaussians
zz = 100.0 * (zz1 - zz2)
```

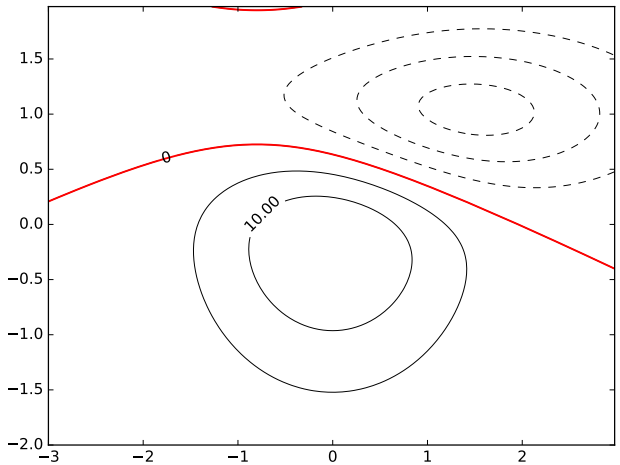
Colored contour lines

```
# draw 21 contours
cs = plt.contour(xx, yy, zz, 21, cmap=plt.cm.jet)

# add the colorbar
cb = plt.colorbar(cs)
```



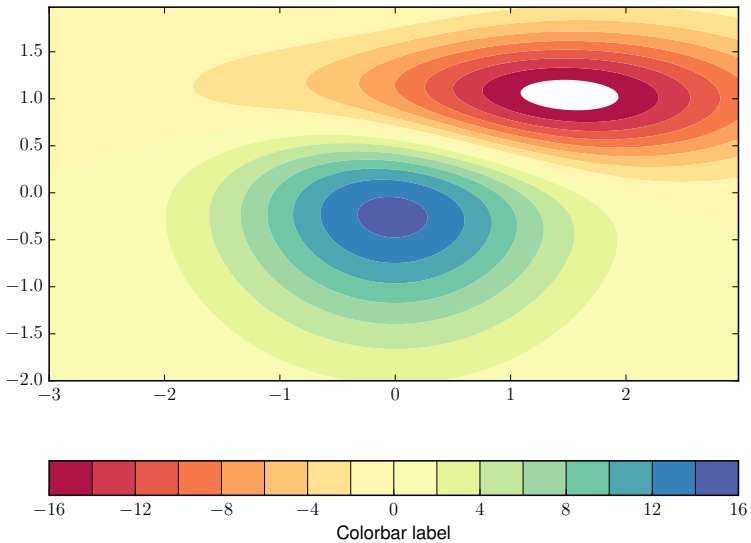
Black contour lines

```
# draw user defined levels with a single color
cs = plt.contour(xx, yy, zz, levels=np.arange(-20, 25, 5),
                 linewidths=0.5, colors='k') # note the plural!

# add the contour labels
plt.clabel(cs, [-20, -10, 10, 20], fmt="%.2d", manual=False)
# if manual=True, you place your labels where you want

# superimpose the 0 contour
plt.contour(xx, yy, zz, levels=[0], colors='r')
```



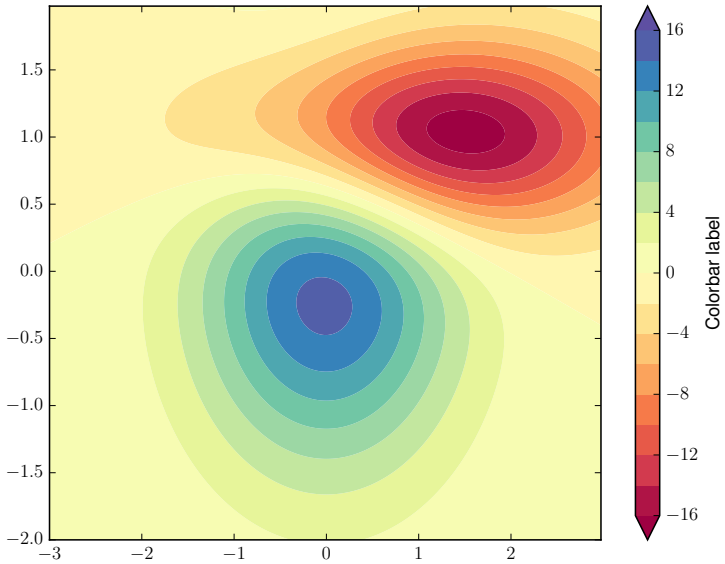
Filled contours

```
cs = plt.contourf(xx, yy, zz,
                  levels = np.arange(-16, 18, 2),
                  cmap=plt.cm.get_cmap('Spectral'))

# add the colorbar
cb = plt.colorbar(cs,orientation='horizontal', drawedges=True)
cb.set_ticks(cs.levels[::2])
cb.set_label('Colorbar label')
```



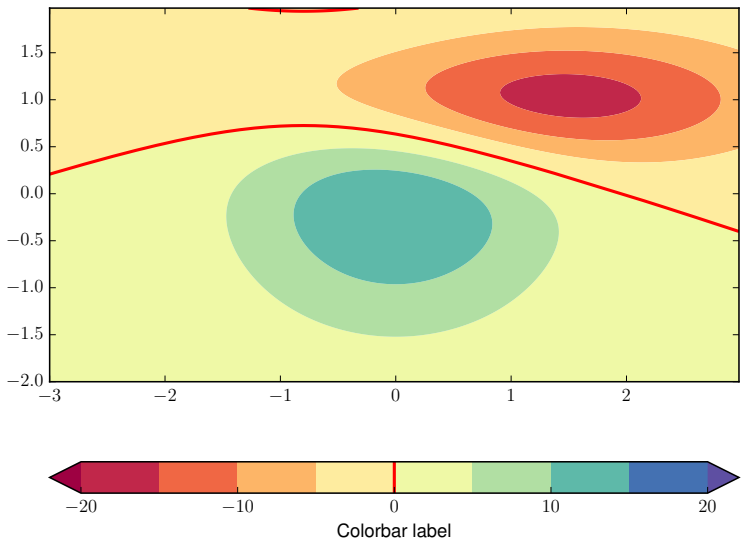
Filled contours

```
cs = plt.contourf(xx, yy, zz,
                  levels = np.arange(-16, 18, 2),
                  cmap=plt.cm.get_cmap('Spectral'),
                  extend='both') # saturates cmap

cb = plt.colorbar(cs,orientation='horizontal', drawedges=True)
cb.set_ticks(cs.levels[::2])
cb.set_label('Colorbar label')
```



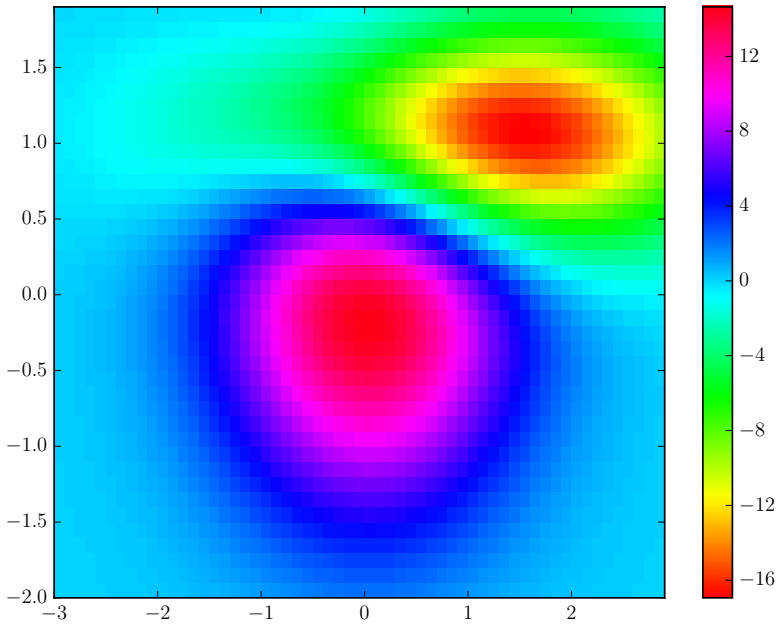
Filled contours

```
# overlay a line contour plot. draws 1 out of 2 levels
cl = plt.contour(xx, yy, zz,
                 levels=cs.levels[::2],
                 colors='m', linewidths=0.5)

# add these lines on the colorbar
cb.add_lines(cl)
```



Pcolor

```
cs = plt.pcolor(xx, yy, zz,
                cmap=plt.cm.get_cmap('Spectral_r'),
                edgecolors='none')

# set the pcolor limits
cs.set_clim(-15, 15)
```



Pcolor with masked data

```
# masking the data
zz = np.ma.masked_where((zz<=10) & (zz>=5), zz)

ax = plt.gca()
ax.set_axis_bgcolor('lightgray') # set background color
cs = plt.pcolor(xx, yy, zz, cmap=plt.cm.get_cmap('Spectral_r'))
```

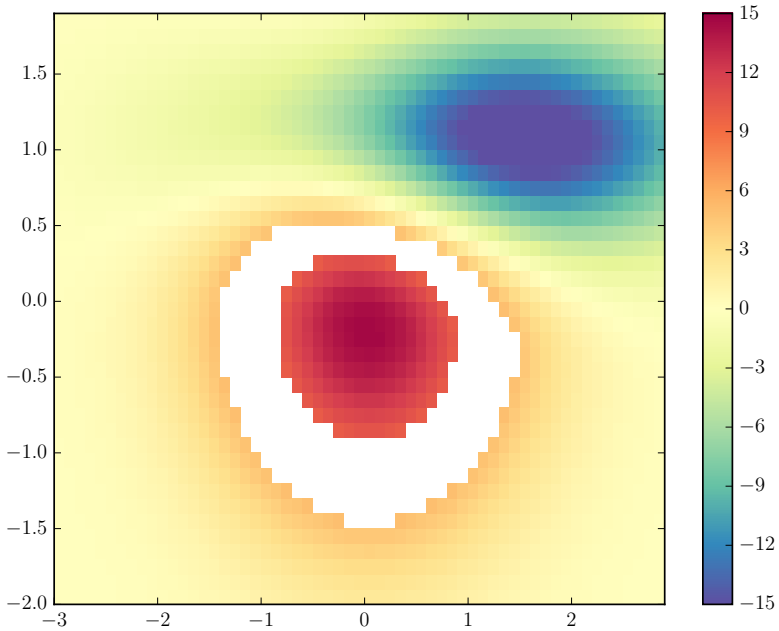


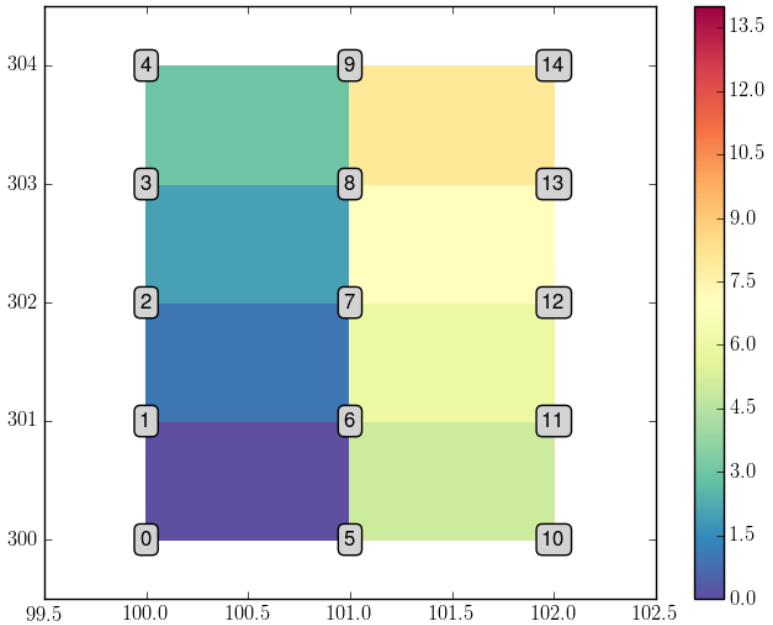
Table of Contents

Data for imshow plots

```
# data: 3 by 5 array of int from 0 to 14
data = np.arange(0, 15)
data = np.reshape(data, (3, 5))

# coordinates
x = np.arange(3) + 100
y = np.arange(5) + 300
```

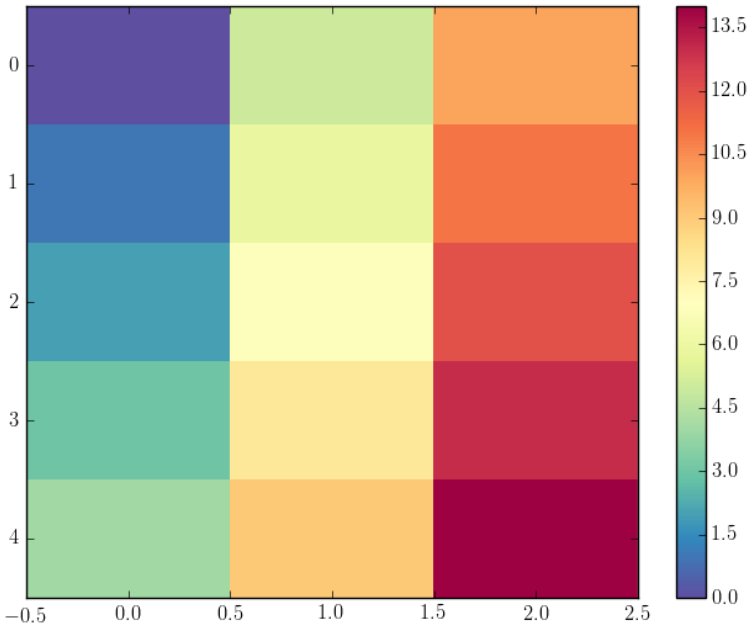
With a pcolor

```
cs = plt.pcolormesh(x, y, data.T)
# data are interpolated
```



Imshow

```
cs = ax.imshow(data.T, interpolation="none")
# axes are now in pixels, and lower values are on top
```

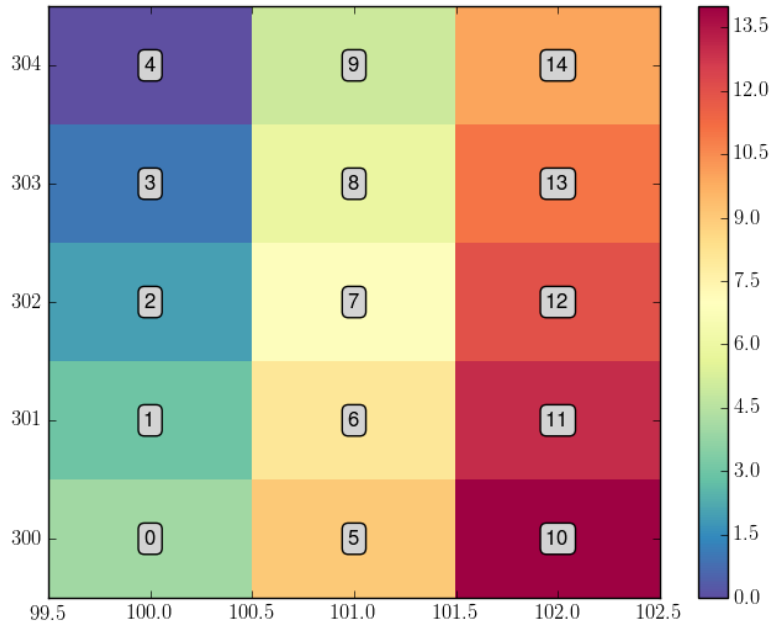


Warning !

The `interpolation="none"` does not work when saving `pdf` files.

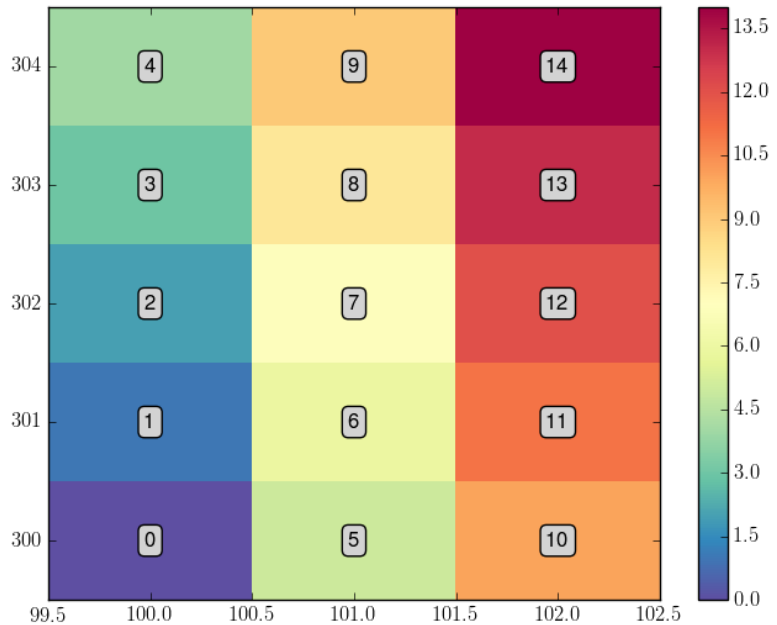
Imshow

```
# plot extent: left, right, bottom, top
extent = [x.min()-0.5, x.max()+0.5, y.min()-0.5, y.max()+0.5]

# use of the extent option
cs = ax.imshow(data.T, interpolation="none", extent=extent)
# axes are ok, but lower values are still on top
```



Imshow

```
# changing the image origin from defaut upper to lower
cs = ax.imshow(data.T, interpolation="none",
               extent=extent, origin="lower")
```



Warning !

Use `imshow` only with *regular* grids !

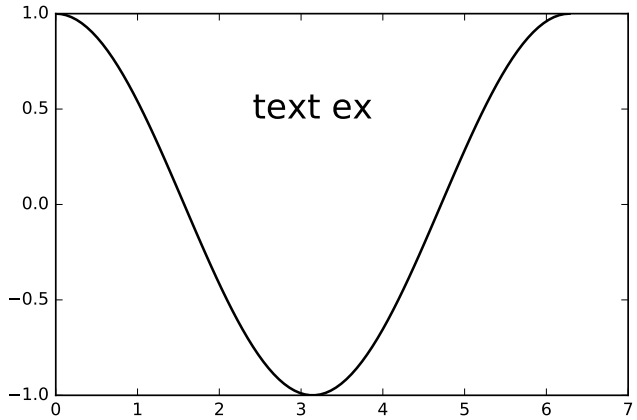
Table of Contents

Data for adding texts

```
import pylab as plt

x = np.linspace(0, 2*np.pi, 100)
y = np.cos(x)
```

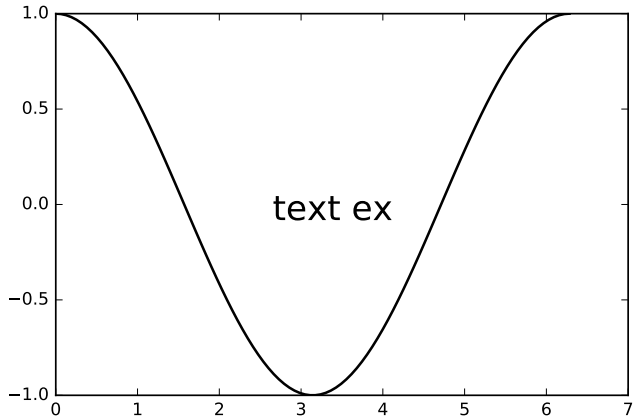
Simple text (data coords.)

```
# text in data coordinates
plt.text(np.pi, 0.5, 'text ex', fontsize=20,
         ha='center', va='center')
```



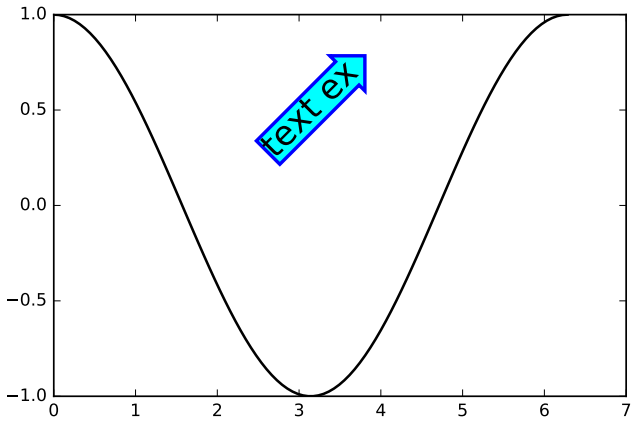
Simple text (figure coords.)

```
# text in figure coordinates (ranging from 0 to 1)
plt.figtext(0.5, 0.5, 'text ex', fontsize=20,
            ha='center', va='center')
```



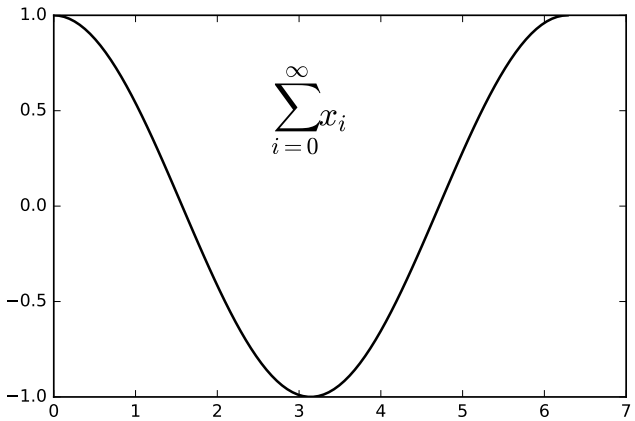
Text bounding box

```
# property for text bounding box
bbox_prop = dict(boxstyle="round, pad=0.8",
                 fc="cyan", ec="b", lw=2)

plt.text(np.pi, 0.5, 'text ex', fontsize=20,
         rotation=45, bbox=bbox_prop)
```



Math text

```
plt.text(np.pi, 0.5, r'$\sum_{i=0}^\infty x_i$', fontsize=20,
         ha='center', va='center')
# The 'r' stands for "raw string litteral".
# It prevents from doubling \ when next to t, n, etc.
# It mimics LaTex syntax.
```



Using LaTex

```
# enable the use of LaTex
plt.rcParams['text.usetex'] = True
# warning! when using LaTex, the _ char. raises an error.
# for math with LaTex, start with the \displaystyle statement
plt.text(np.pi, 0.5, r'$\displaystyle \sum_{i=0}^\infty x_i$',
         fontsize=20, ha='center', va='center')
```

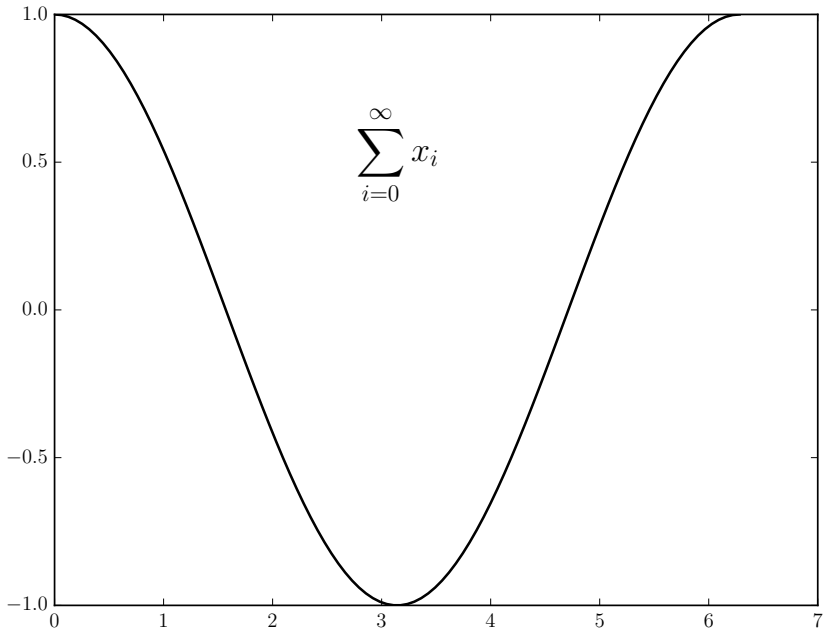
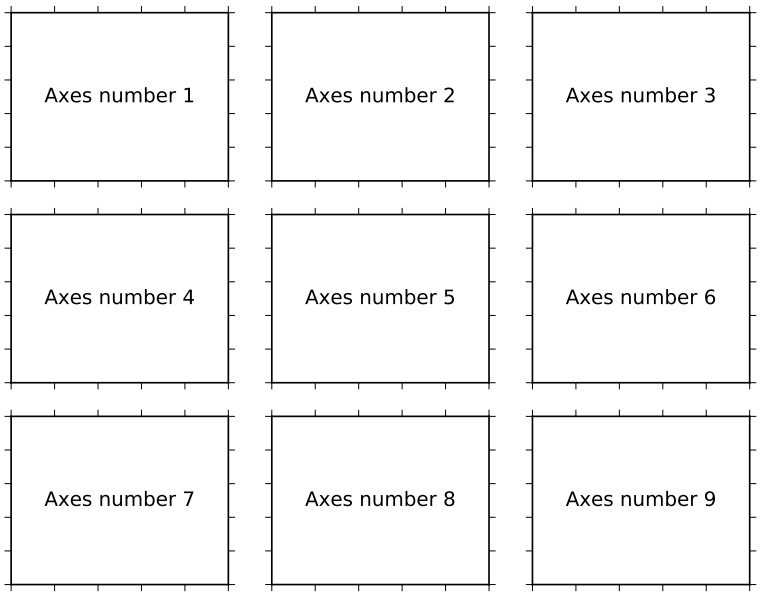


Table of Contents

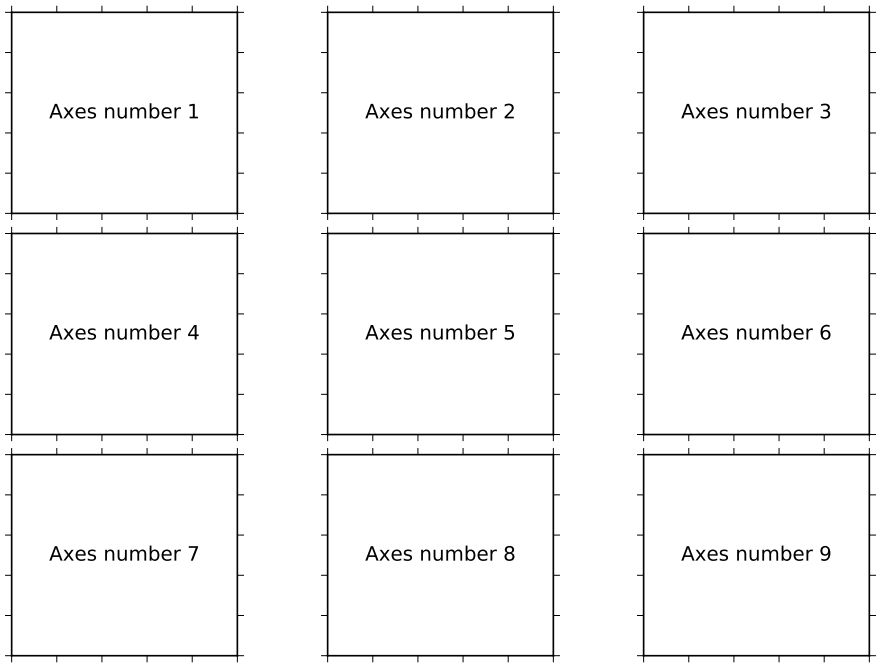
First panel plot

```
fig = plt.figure()
for p in range(1, 3*3+1): # small loop
    ax = plt.subplot(3, 3, p) # subplot of 3x3
```



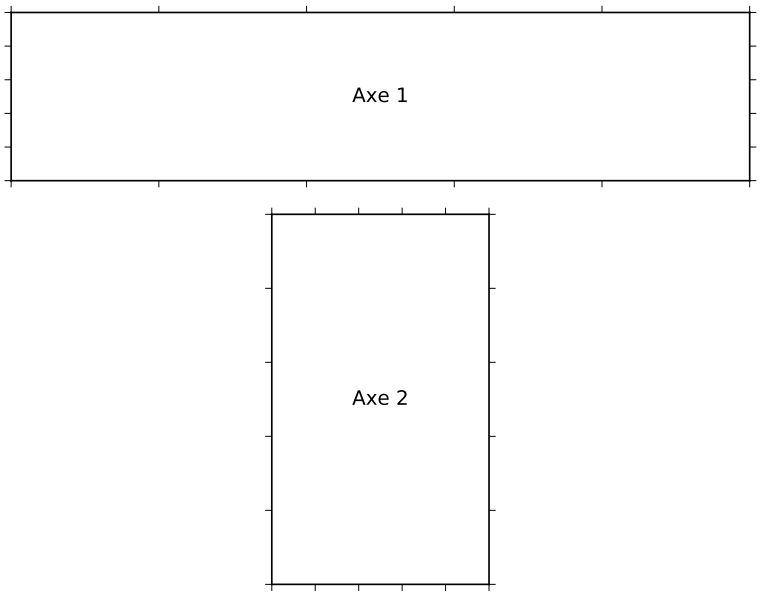
Change the spacing

```
fig = plt.figure()
plt.subplots_adjust(wspace=0.4, hspace=0.4,
                    left=0.05, right=0.95,
                    bottom=0.05, top=0.95)
for p in range(1, 3*3+1): # small loop
    ax = plt.subplot(3, 3, p) # subplot of 3x3
```



Complicated panel

```
# size of the subplot: 3 by 3
# location of the plot: 0(top), 0(left), spans 3 columns
ax1 = plt.subplot2grid((3, 3), (0, 0), colspan=3)
ax2 = plt.subplot2grid((3, 3), (1, 1), rowspan=2)
```



Hand positionning

```
fig = plt.figure()
for p in range(1, 3*3+1): # small loop
    ax = plt.subplot(3, 3, p) # subplot of 3x3
# add a new axis: arg=[left bottom width height]
ax = plt.axes([0.25, 0.25, 0.5, 0.5]) # fig. coords
```