Introduction to Matplotlib

Basic plotting with python in matlab-style

introduction

- „make easy things easy and hard things possible:“
  - create simple plots with just a few commands
  - “emulate” MATLABs plotting capabilities

- matplotlib is conceptually divided into three parts
  - ***Pylab interface :*** MATLAB like plotting
  - ***Matplotlib API :*** abstract interface
  - ***Backends :*** managing the output

- available at (including many examples)

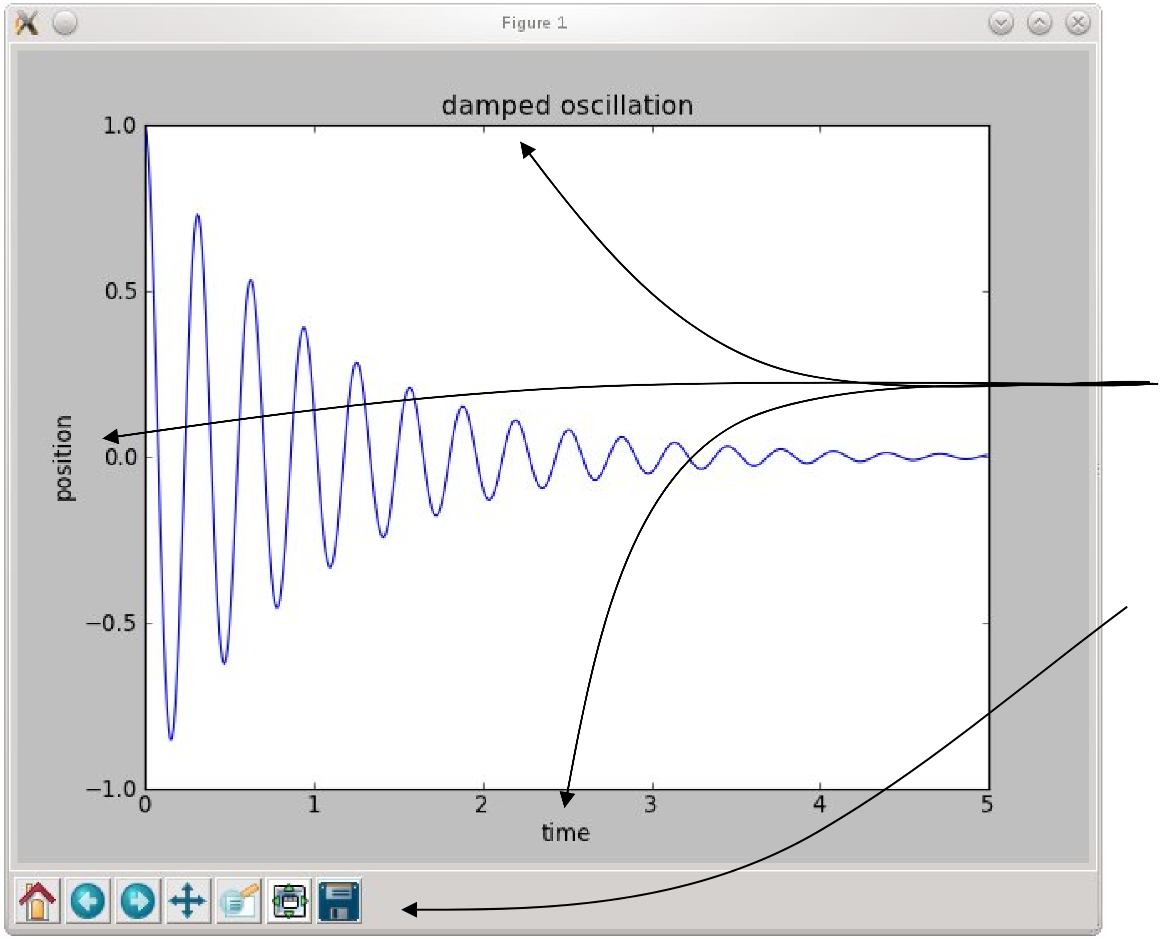
  http://matplotlib.sourceforge.net/

Basic 2D - plotting

- Matlab like example:

```
import numpy as np              # import numpy
import pylab as pl              # import pylab interface



times = np.arange ( 0, 5, 0.01 )            # define x-vector
fun  = lambda x : np.cos (20 *x) * np.exp (- pl.absolute(x) )
                          # define some function fun (x)

pl.plot ( times, fun(times) )   # plot fun (t) vs. t
pl.xlabel ('time' )             # creating x-label
pl.ylabel ('position')          # creating y-label

pl.title ( 'damped oscillation')            # setting the title
pl.show()                                    # show the plot
```

Basic 2D - plotting



line plot represents data

title and labels

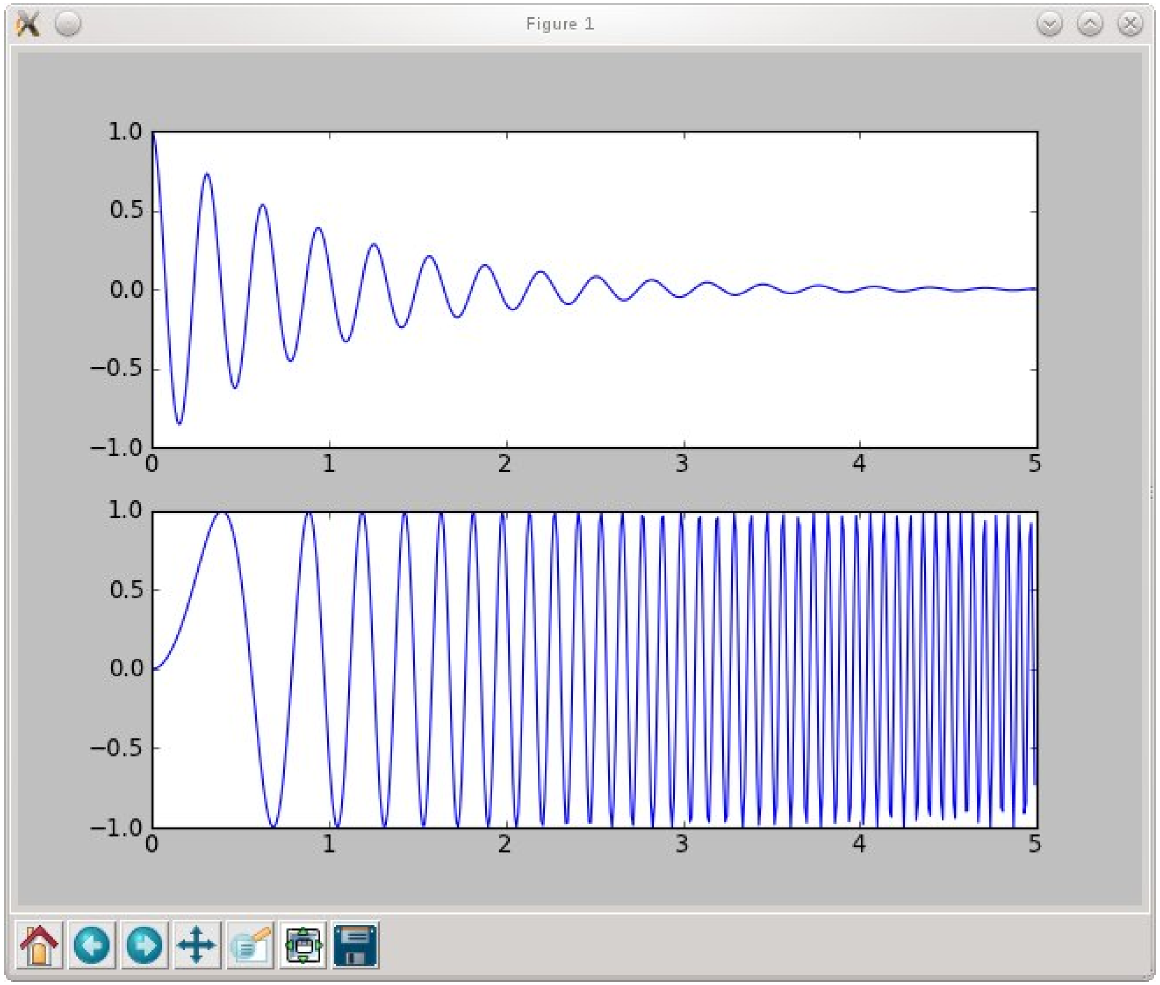
toolbar for zooming, saving/exporting etc.

appearance depend on backend

Subplots

```
import numpy as np                 # import numpy
import pylab as pl                 # import pylab interface

times = np.arange ( 0, 5, 0.01 )            # define x-vector

fun   = lambda x : np.cos (20 *x) * np.exp (- pl.absolute(x) )
fun2  = lambda x : np.sin (10 *x**2)     # define two functions


pl.subplot (2,1,1)                 # choose a subplot ( rows, colums, idx)
pl.plot ( times, fun(times) )      # plot fun(t)

pl.subplot (2,1,2)                 # choose a subplot ( rows, colums, idx)
pl.plot ( times, fun2(times) )     # plot fun2(t)

pl.show()
```

Subplots



subplot (2,1,1) :

- 2 columns, 1 row
- choose first subplot

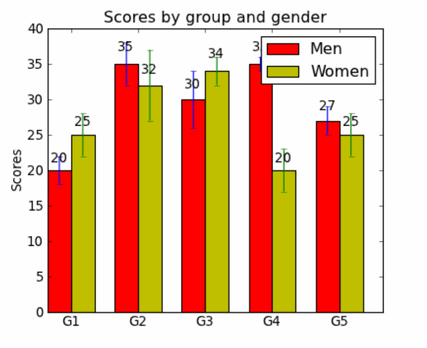
! Indexing starts with 1

subplot (2,1,2) :

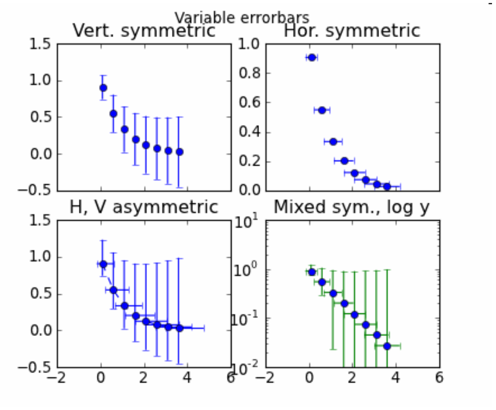
- 2 columns, 1 row
- choose second subplot

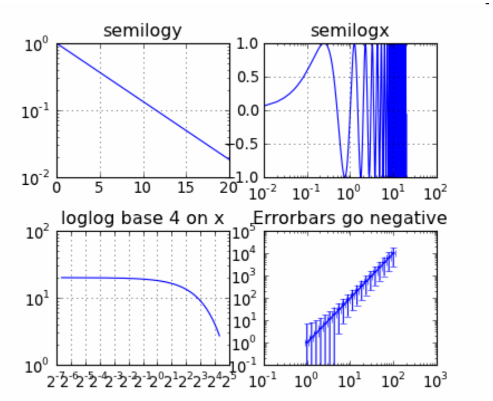
Other basic plotting commands

- pl.bar () # box plot



- pl.errorbar() # plot with errorbars
- pl.loglog() # logarithmically scaled axis
- pl.semilogx () # x-axis logarithmically scaled
- pl.semilogy () # y-axis logarithmically scaled

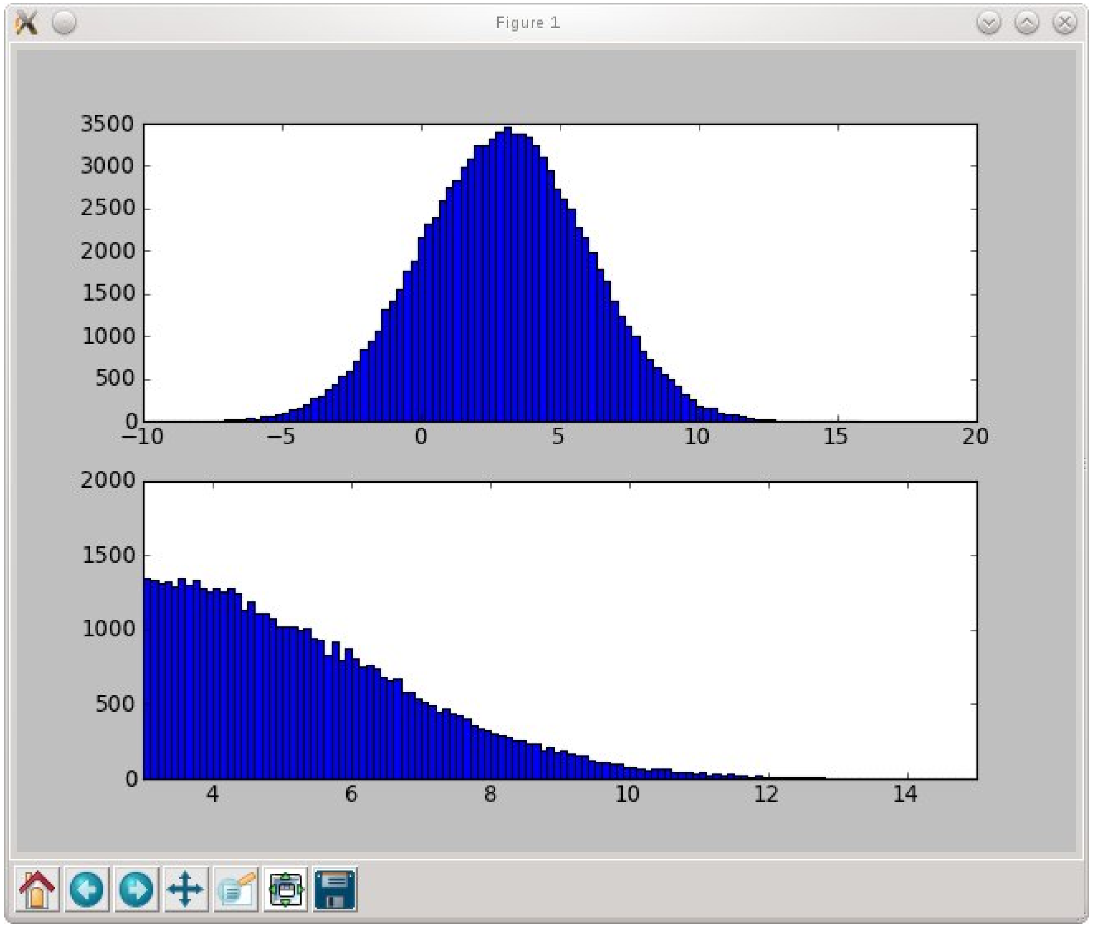




Histograms

```
import numpy as np              # import numpy
import pylab as pl              # import pylab interface

data = 3. + 3. * np.random.randn (100000)
          # generate normally distributed randonnumbers

pl.subplot (2,1,1)
pl.hist (data, 100)    # make histogram with 100 bins

pl.subplot (2,1,2)
pl.hist ( data, bins = np.arange(3, 25, 0.1) )
                       # make histogram with given bins

pl.axis ( (3, 15,0,2000 ))        # specify axis (x1,x2,y1,y2)

pl.show()
```

Histograms



(automatic) histogram with 100 bins

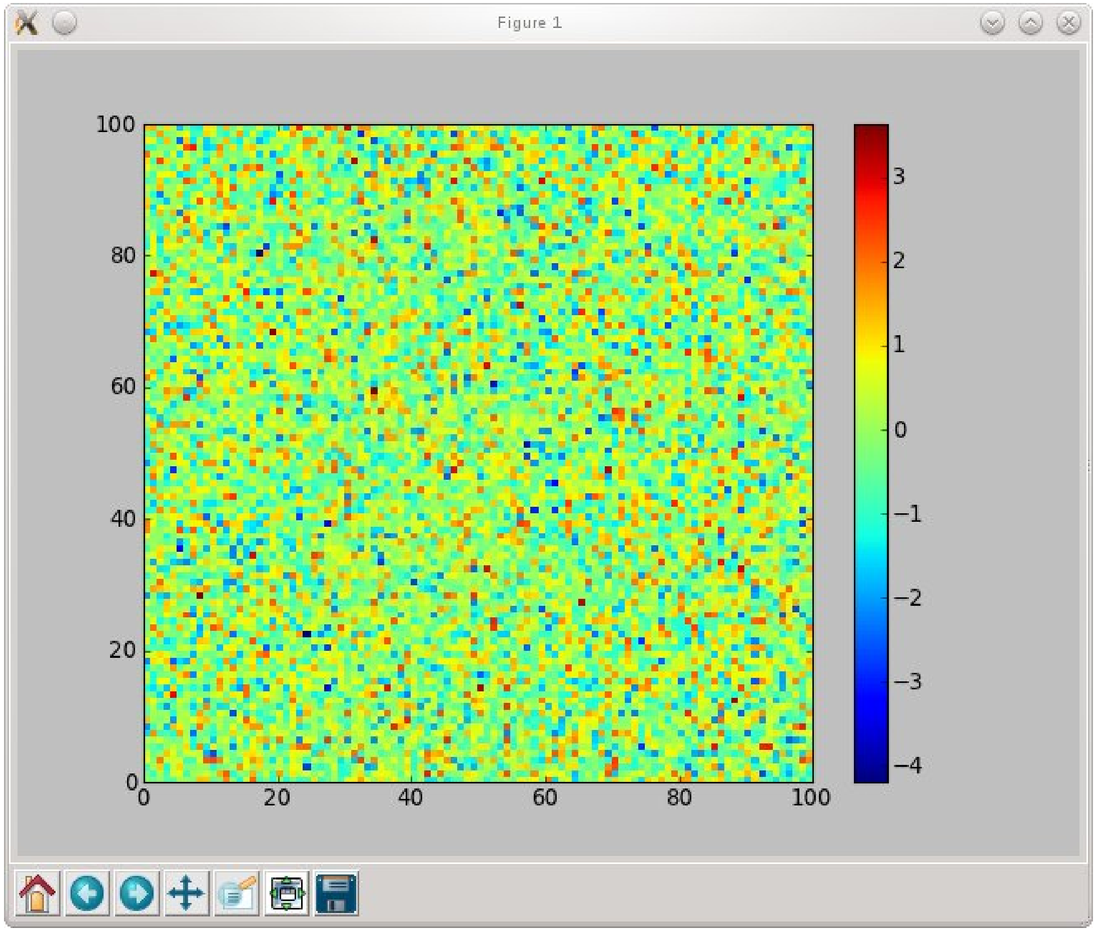
histogram for data between 3. and 25. with binsize 0.1

axis set to (3,15,0,2000)

Basic Matrix Plotting

```
import numpy as np              # import numpy
import pylab as pl              # import pylab interface

data = np.random.randn (100,100)
                # generate random data

pl.pcolor (data)       # plot data
pl.colorbar()          # show a colorbar

pl.show()
```

Basic Matrix Plotting



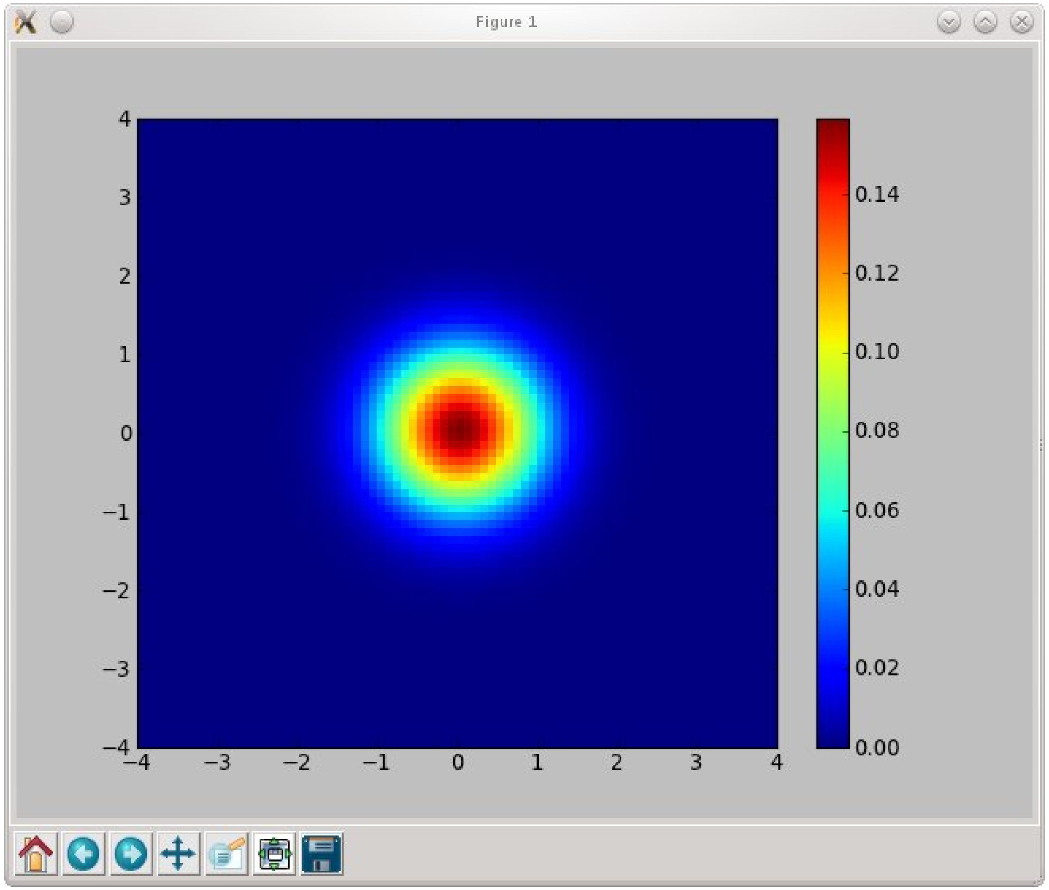
dimensions of matrix used as coordinates

entries are translated to a colorcode

2D - Functions

```
import numpy as np               # import numpy
import pylab as pl               # import pylab interface

x = np.arange ( -4, 4.01, 0.1)   # x-values
y = np.arange ( -4, 4.01, 0.1)   # y-values

X,Y = np.meshgrid(x,y)           # Create a meshgrid !
Z = np.zeros ( X.shape )         # Matrix for function values
Z = 1./2/np.pi * np.exp ( - (X**2 + Y**2) )

pl.pcolor(X,Y,Z)                 # plot function
pl.axis ( (-4,4,-4,4) )          # set axis
pl.colorbar()                    # show colorbar

pl.show()
```

2D - Functions



$$X = \begin{pmatrix} x_1 & x_1 & x_1 & \dots \\ x_2 & x_2 & x_2 & \dots \\ \dots & \dots & \dots & \end{pmatrix}$$

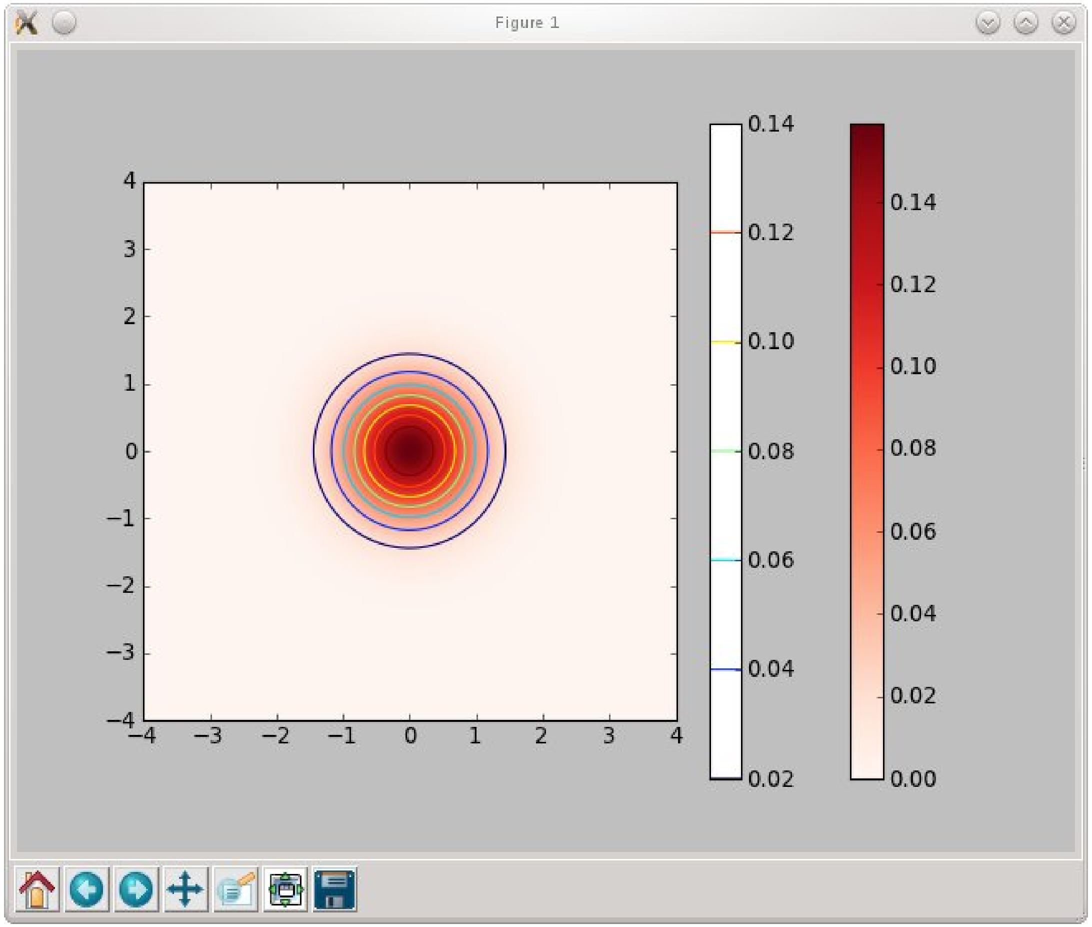
$$Y = \begin{pmatrix} y_1 & y_2 & y_3 & \dots \\ y_1 & y_2 & y_3 & \dots \\ \dots & \dots & \dots & \end{pmatrix}$$

$$Z = \begin{pmatrix} f(x_1, y_1) & f(x_1, y_2) & \dots \\ f(x_2, y_1) & f(x_2, y_2) & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Contour Plots

```
import numpy as np              # import numpy
import pylab as pl              # import pylab interface

x = np.arange ( -4, 4.01, 0.01)         # x-values
y = np.arange ( -4, 4.01, 0.01)         # y-values

X,Y = np.meshgrid(x,y)          # Create a meshgrid !
Z = np.zeros ( X.shape )        # Matrix for function values
Z = 1./2/np.pi * np.exp ( - (X**2 + Y**2) )

pl.imshow(Z, extent=(-4,4,-4,4), cmap = pl.cm.Reds  )
pl.colorbar()

pl.contour(X,Y,Z) # Creates a contour plot
pl.colorbar()

pl.show()
```

Contour Plots

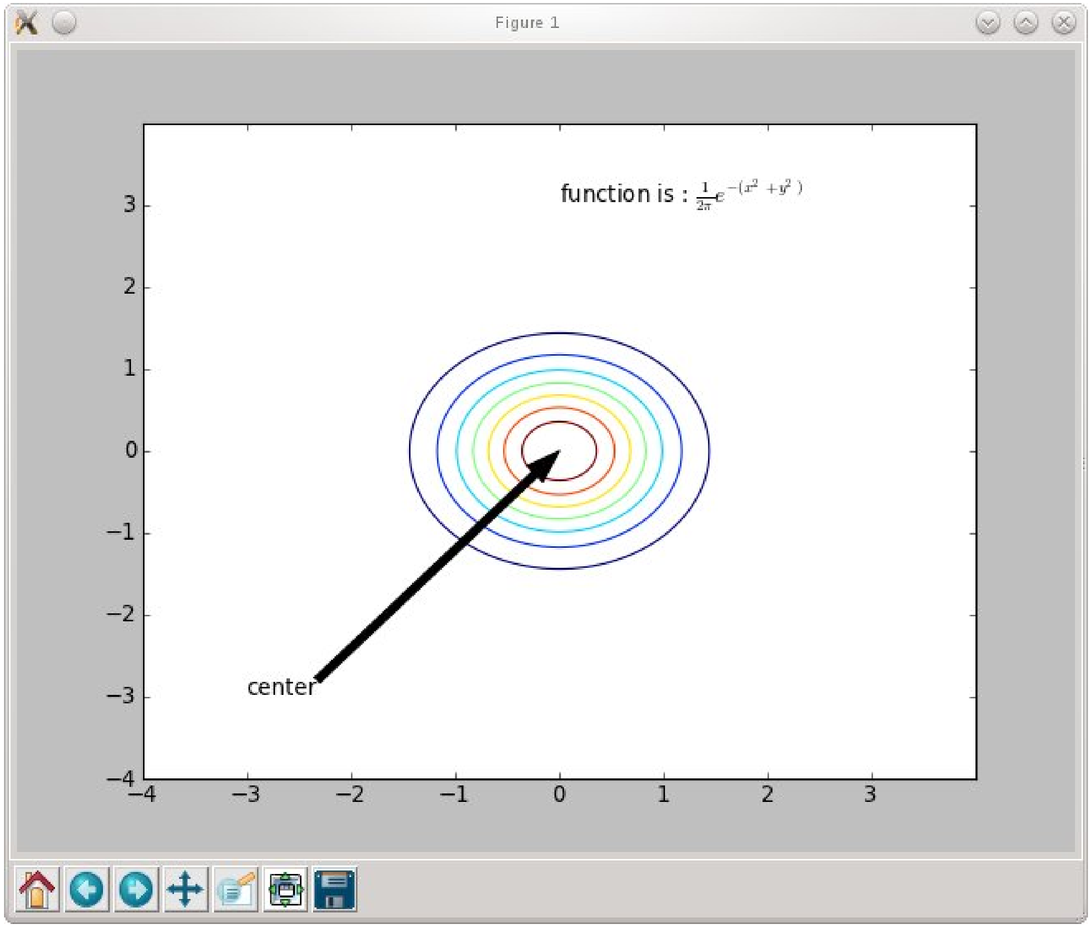


Working with text

- Include text with text() or annotate()
  - you can use TeX (translated by matplotlib itself)
  - you can use real LaTeX (matplotlib.rc (‘text’, usetex=‘true’))

```
import numpy as np                # import numpy
import pylab as pl                # import pylab interface
…
pl.contour(X,Y,Z)                 # make a contour plot

pl.text (0,3, 'function is : '+ r'$\frac{1}{2\pi} e^{- (x^2 + y^2) } $' )
            # (x,y, text);   r‘….‘ indicates rawtext
pl.annotate ( 'center', xy = (0,0), xytext = (-3,-3), arrowprops =
{'facecolor':'black'} )
            # xy <= where the arrow ends
            # xytext <= position of the text

pl.show()
```

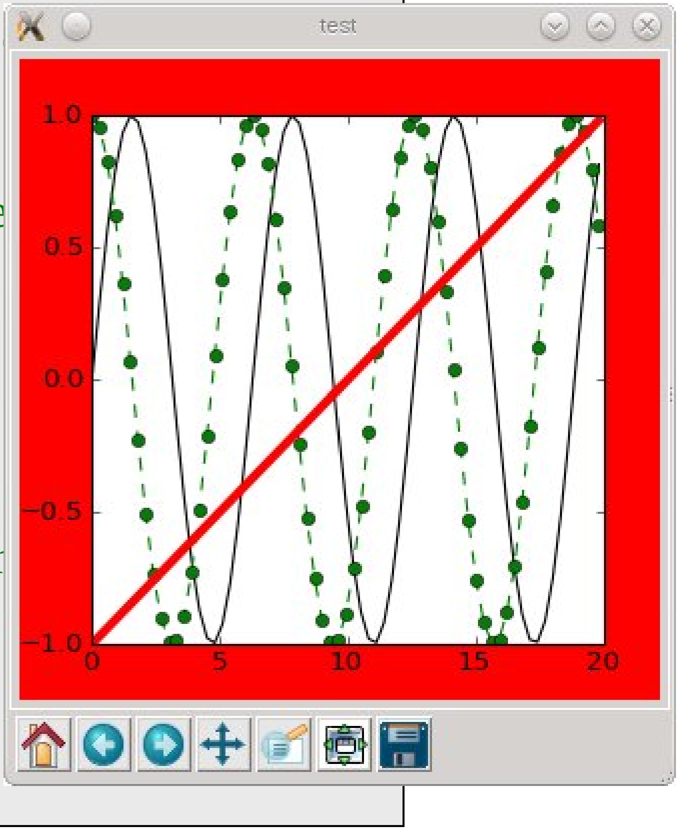
Working with text



Formatting the figures (keywords)

- Using pylab, properties of plots can be set by keywords:

```
import numpy as np              # import numpy
import pylab as pl              # import pylab interfac

pl.figure( “test“ , figsize = (4,4), facecolor = 'r')
          # create figure with title test, 4x5 inches, re

x = np.arange ( 0, 20, 0.3 )    # x – values

# for basic properties: using formatstring
pl.plot (x, np.sin(x), 'k' )    # black line
pl.plot (x, np.cos(x), 'go--' ) # green dotted line with

# using keywords
pl.plot (x, x / 10. - 1, color = 'red', linewidth = 4)
pl.show()
```



For details: **help(command)** or **http://matplotlib.sourceforge.net/**

Using the API

- When creating figures, subplots etc. an object reference is returned that can be used for manipulations:

```
import numpy as np                # import numpy
import pylab as pl                # import pylab interface

fig1 = pl.figure()                          # reurns the figure object
graph = fig1.add_subplot(111)               # returns an axes object
line = graph.plot ( np.arange(10), np.arange(10)**2 )
                                            # returns a list of lines object
xlabel = graph.set_xlabel ( 'x values' )    # returns a text object

# modify the objects
xlabel.set_color('red')           # set  the color of xlabel
line[0].set_linewidth(10)         # set the width of the first line
graph.set_xlim(0,15)              # set the extent of x-axis

pl.show()
```

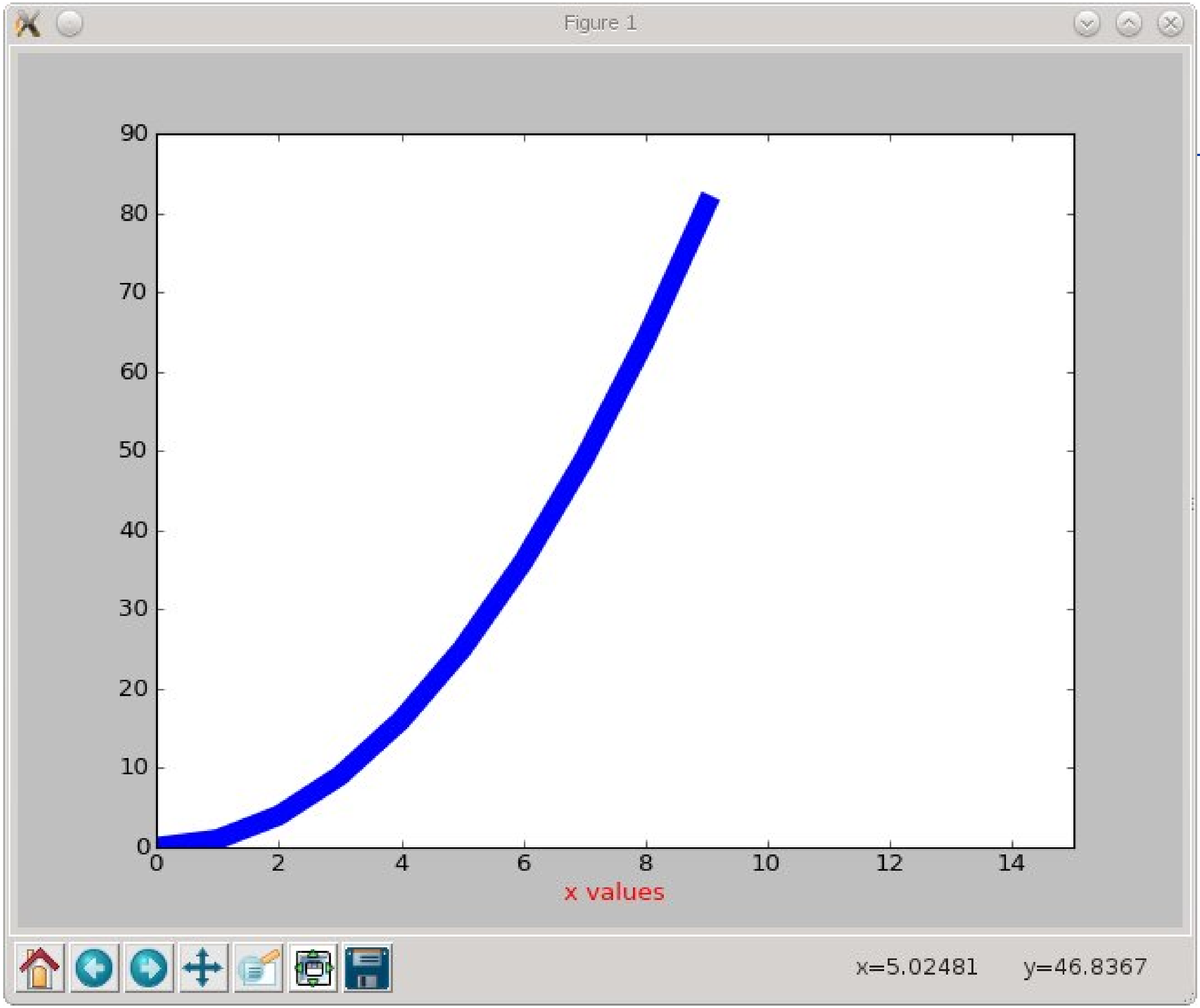


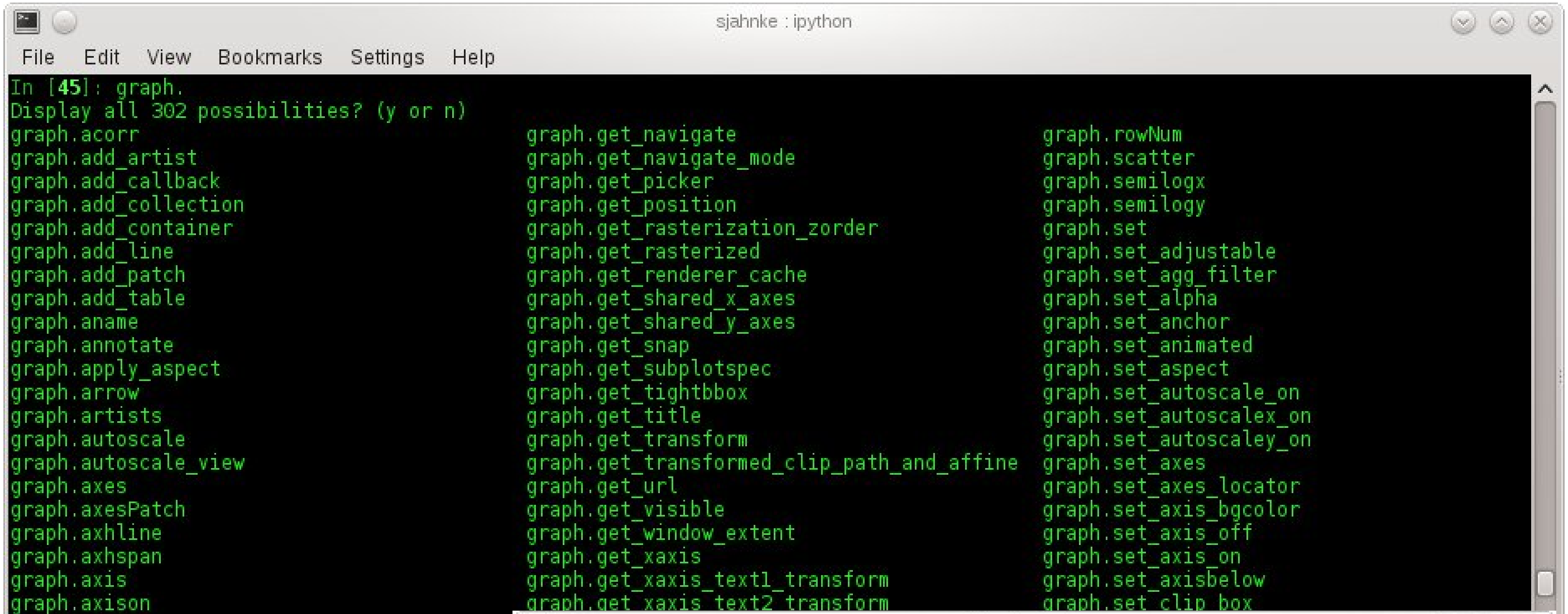
Figure 1
90
80
70
60
50
40
30
20
10
0
0
2
4
6
8
10
12
14
x values
x=5.02481
y=46.8367

pylab.getp(), pylab.setp()

Once you have the objects, you can manipulate them via

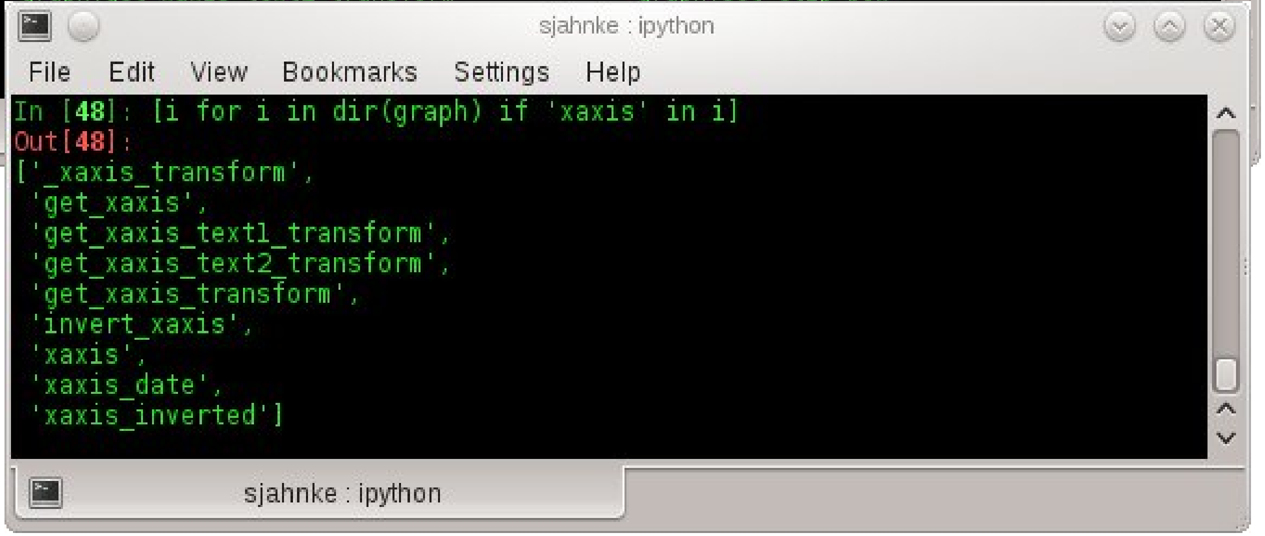
- the methods provided by the class:
  - there are thousands of **object.get_xxx** and **object.set_xxx** methods

    use the <tab> or look at **http://matplotlib.sourceforge.net/api**



```
In [45]: graph.
Display all 302 possibilities? (y or n)
graph.acorr                                 graph.get_navigate                          graph.rowNum
graph.add_artist                            graph.get_navigate_mode                     graph.scatter
graph.add_callback                          graph.get_picker                            graph.semilogx
graph.add_collection                        graph.get_position                          graph.semilogy
graph.add_container                         graph.get_rasterization_zorder              graph.set
graph.add_line                              graph.get_rasterized                        graph.set_adjustable
graph.add_patch                             graph.get_renderer_cache                    graph.set_agg_filter
graph.add_table                             graph.get_shared_x_axes                     graph.set_alpha
graph.aname                                 graph.get_shared_y_axes                     graph.set_anchor
graph.annotate                              graph.get_snap                              graph.set_animated
graph.apply_aspect                          graph.get_subplotspec                       graph.set_aspect
graph.arrow                                 graph.get_tightbbox                         graph.set_autoscale_on
graph.artists                               graph.get_title                             graph.set_autoscalex_on
graph.autoscale                             graph.get_transform                         graph.set_autoscaley_on
graph.autoscale_view                        graph.get_transformed_clip_path_and_affine  graph.set_axes
graph.axes                                  graph.get_url                               graph.set_axes_locator
graph.axesPatch                             graph.get_visible                           graph.set_axis_bgcolor
graph.axhline                               graph.get_window_extent                     graph.set_axis_off
graph.axhspan                               graph.get_xaxis                             graph.set_axis_on
graph.axis                                  graph.get_xaxis_text1_transform             graph.set_axisbelow
graph.axison                                graph.get_xaxis_text2_transform             graph.set_clip_box
graph.axvline
graph.axvspan
graph.bar
```

“search” through methods:



```
In [48]: [i for i in dir(graph) if 'xaxis' in i]
Out[48]:
['_xaxis_transform',
 'get_xaxis',
 'get_xaxis_text1_transform',
 'get_xaxis_text2_transform',
 'get_xaxis_transform',
 'invert_xaxis',
 'xaxis',
 'xaxis_date',
 'xaxis_inverted']
```

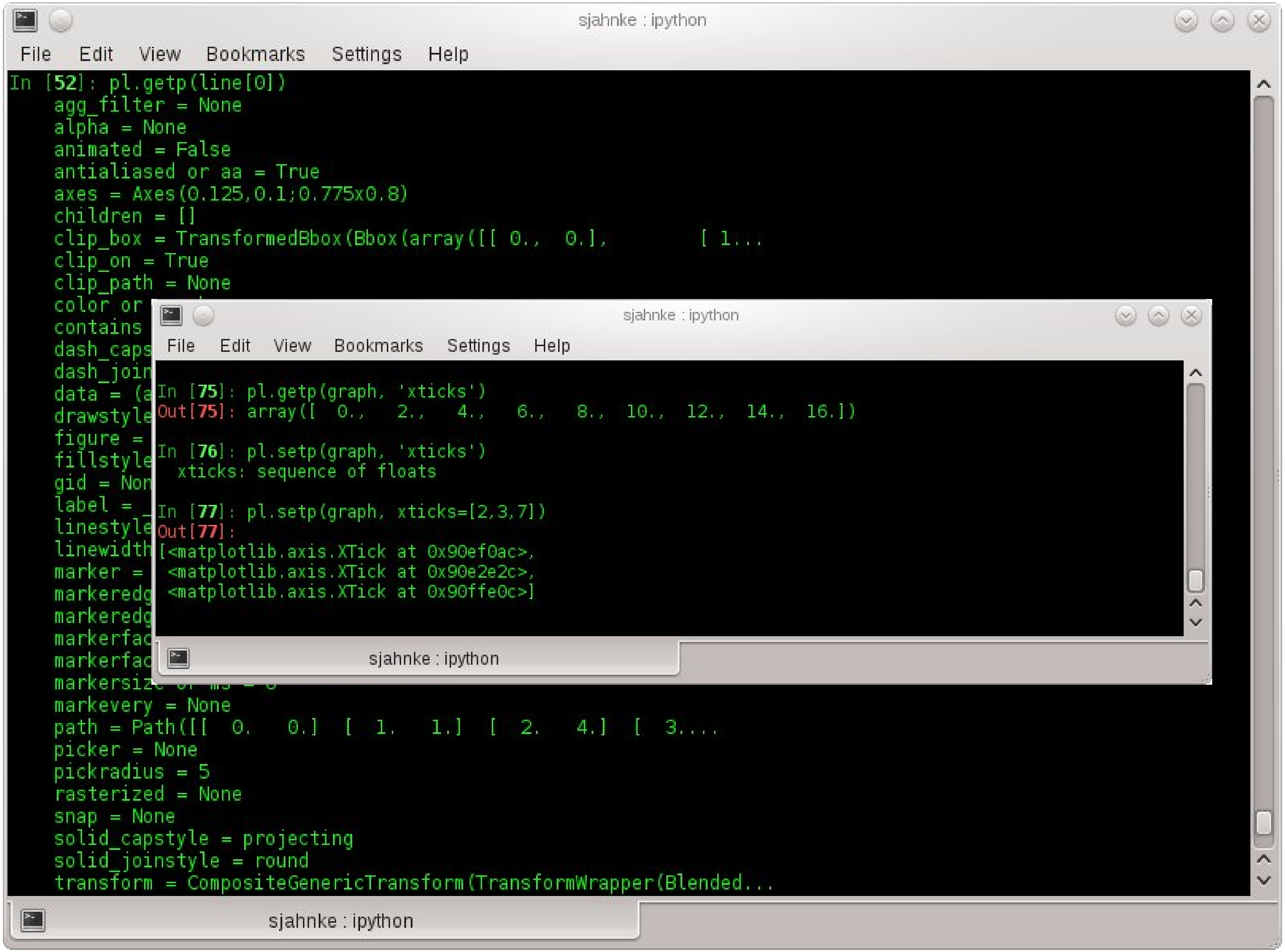
pylab.getp(), pylab.setp()

Once you have the objects, you can manipulate them via

- the methods provided by the class:
  - there are thousands of **object.get_xxx** and **object.set_xxx** methods

    use the <tab> or look at **http://matplotlib.sourceforge.net/api**

- **pylab.setp()** command:
  - **pylab.getp(obj)** returns all properties of the object
  - **pylab.getp(obj, property)** returns value of current property
  - **pylab.setp(obj, property)** returns possible values for property
  - **pylab.setp(obj, propety=value)**



```
In [52]: pl.getp(line[0])
    agg_filter = None
    alpha = None
    animated = False
    antialiased or aa = True
    axes = Axes(0.125,0.1;0.775x0.8)
    children = []
    clip_box = TransformedBbox(Bbox(array([[ 0.,  0.],        [ 1...
    clip_on = True
    clip_path = None
    color or
    contains
    dash_caps
    dash_join
    data = (a
    drawstyle
    figure =
    fillstyle
    gid = Non
    label = _
    linestyle
    linewidth
    marker =
    markeredg
    markeredg
    markerfac
    markerfac
    markersize or ms = 6
    markevery = None
    path = Path([[  0.   0.]  [  1.   1.]  [  2.   4.]  [  3....
    picker = None
    pickradius = 5
    rasterized = None
    snap = None
    solid_capstyle = projecting
    solid_joinstyle = round
    transform = CompositeGenericTransform(TransformWrapper(Blended...
```

```
In [75]: pl.getp(graph, 'xticks')
Out[75]: array([  0.,   2.,   4.,   6.,   8.,  10.,  12.,  14.,  16.])

In [76]: pl.setp(graph, 'xticks')
  xticks: sequence of floats

In [77]: pl.setp(graph, xticks=[2,3,7])
Out[77]:
[<matplotlib.axis.XTick at 0x90ef0ac>,
 <matplotlib.axis.XTick at 0x90e2e2c>,
 <matplotlib.axis.XTick at 0x90ffe0c>]
```

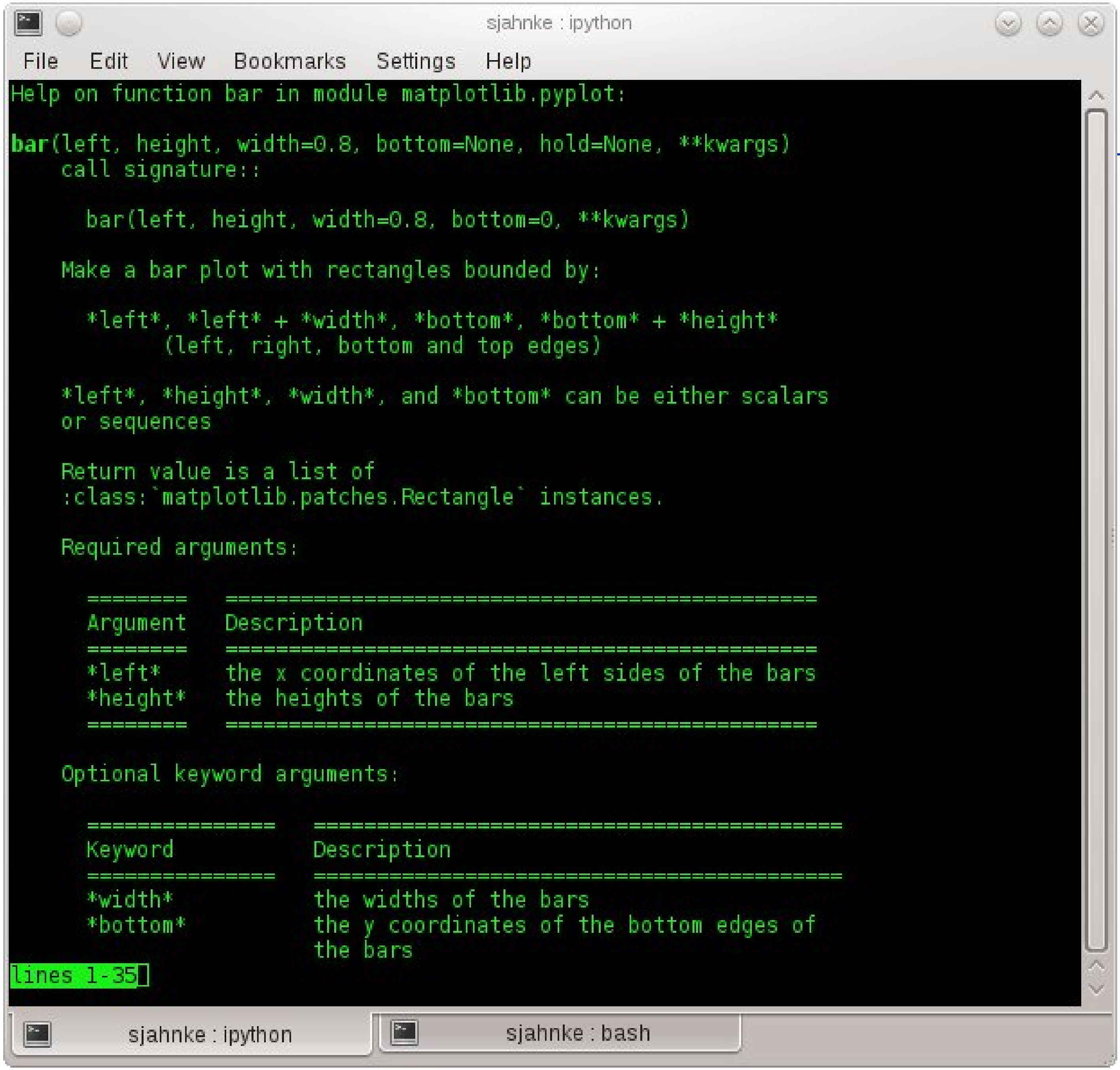
Example 1: bar plots

- The task:

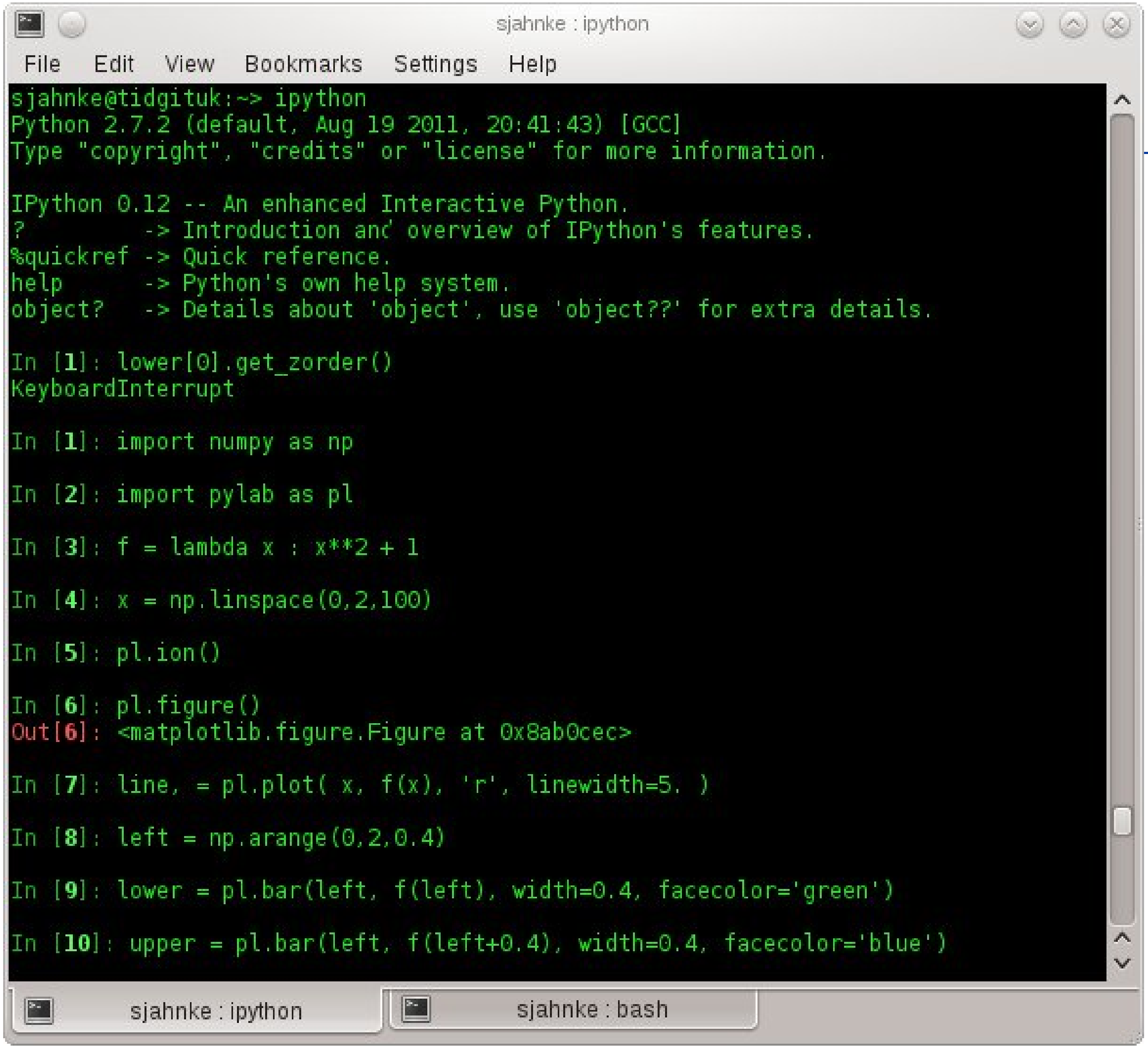
1. Plot the function $f(x) = x^2 + 1$ in the interval $[0, 2]$ in with a red thick line.

2. Make a bar plot showing the upper sum (blue) and lower sum (green) approximating the Riemann integral (width = 0.4).

Hint: You can use **pylab.ion()** to start the interactive mode. Then the results are directly displayed. With **pylab.draw()** you can update the figure after changing it.



```
Help on function bar in module matplotlib.pyplot:

bar(left, height, width=0.8, bottom=None, hold=None, **kwargs)
    call signature::

      bar(left, height, width=0.8, bottom=0, **kwargs)

    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
    :class:`matplotlib.patches.Rectangle` instances.

    Required arguments:

      ========   ===============================================
      Argument   Description
      ========   ===============================================
      *left*     the x coordinates of the left sides of the bars
      *height*   the heights of the bars
      ========   ===============================================

    Optional keyword arguments:

      ===============   ==========================================
      Keyword           Description
      ===============   ==========================================
      *width*           the widths of the bars
      *bottom*          the y coordinates of the bottom edges of
                        the bars
lines 1-35
```



```
sjahnke@tidgituk:~> ipython
Python 2.7.2 (default, Aug 19 2011, 20:41:43) [GCC]
Type "copyright", "credits" or "license" for more information.

IPython 0.12 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: lower[0].get_zorder()
KeyboardInterrupt

In [1]: import numpy as np

In [2]: import pylab as pl

In [3]: f = lambda x : x**2 + 1

In [4]: x = np.linspace(0,2,100)

In [5]: pl.ion()

In [6]: pl.figure()
Out[6]: <matplotlib.figure.Figure at 0x8ab0cec>

In [7]: line, = pl.plot( x, f(x), 'r', linewidth=5. )

In [8]: left = np.arange(0,2,0.4)

In [9]: lower = pl.bar(left, f(left), width=0.4, facecolor='green')

In [10]: upper = pl.bar(left, f(left+0.4), width=0.4, facecolor='blue')
```

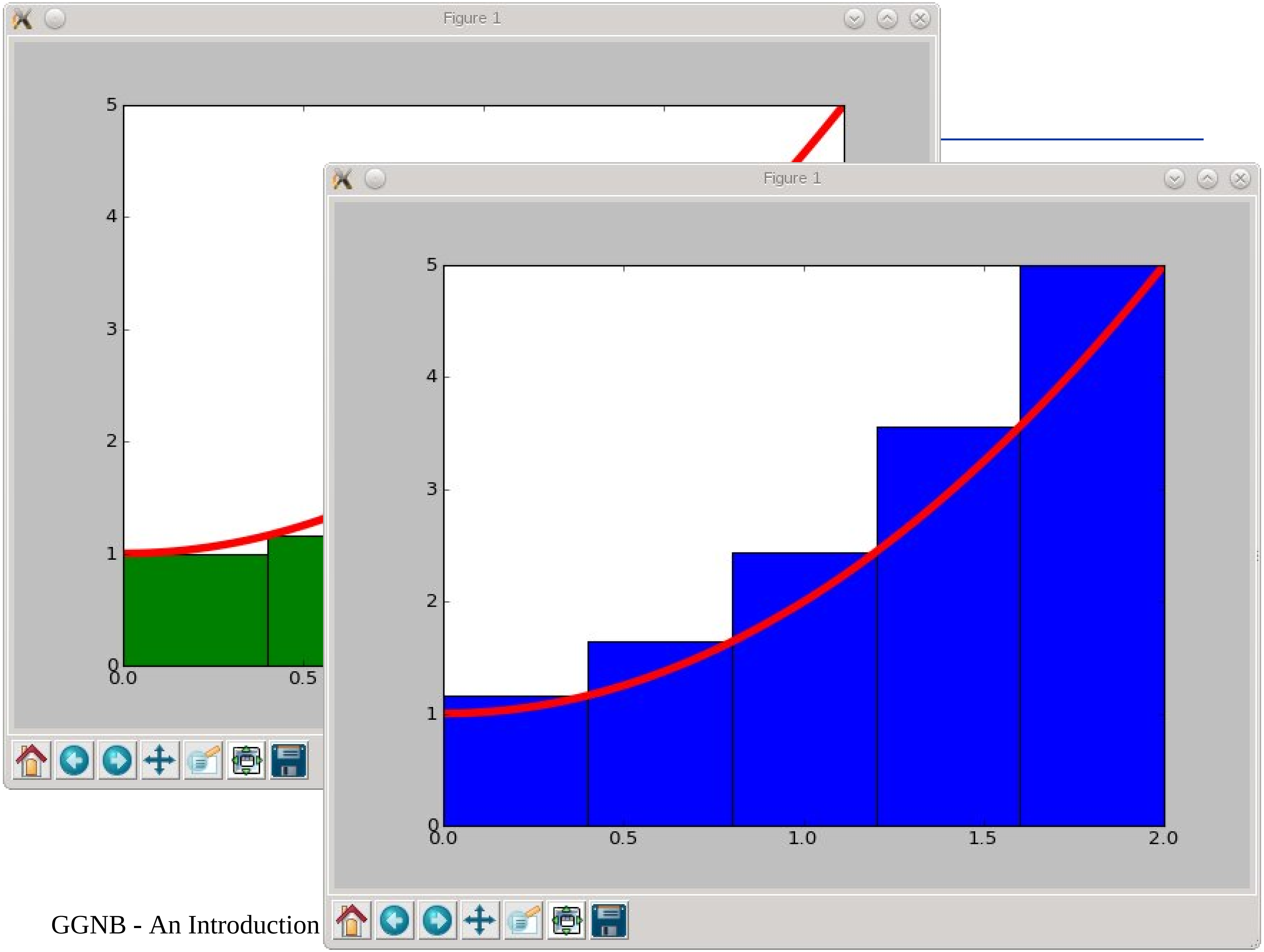
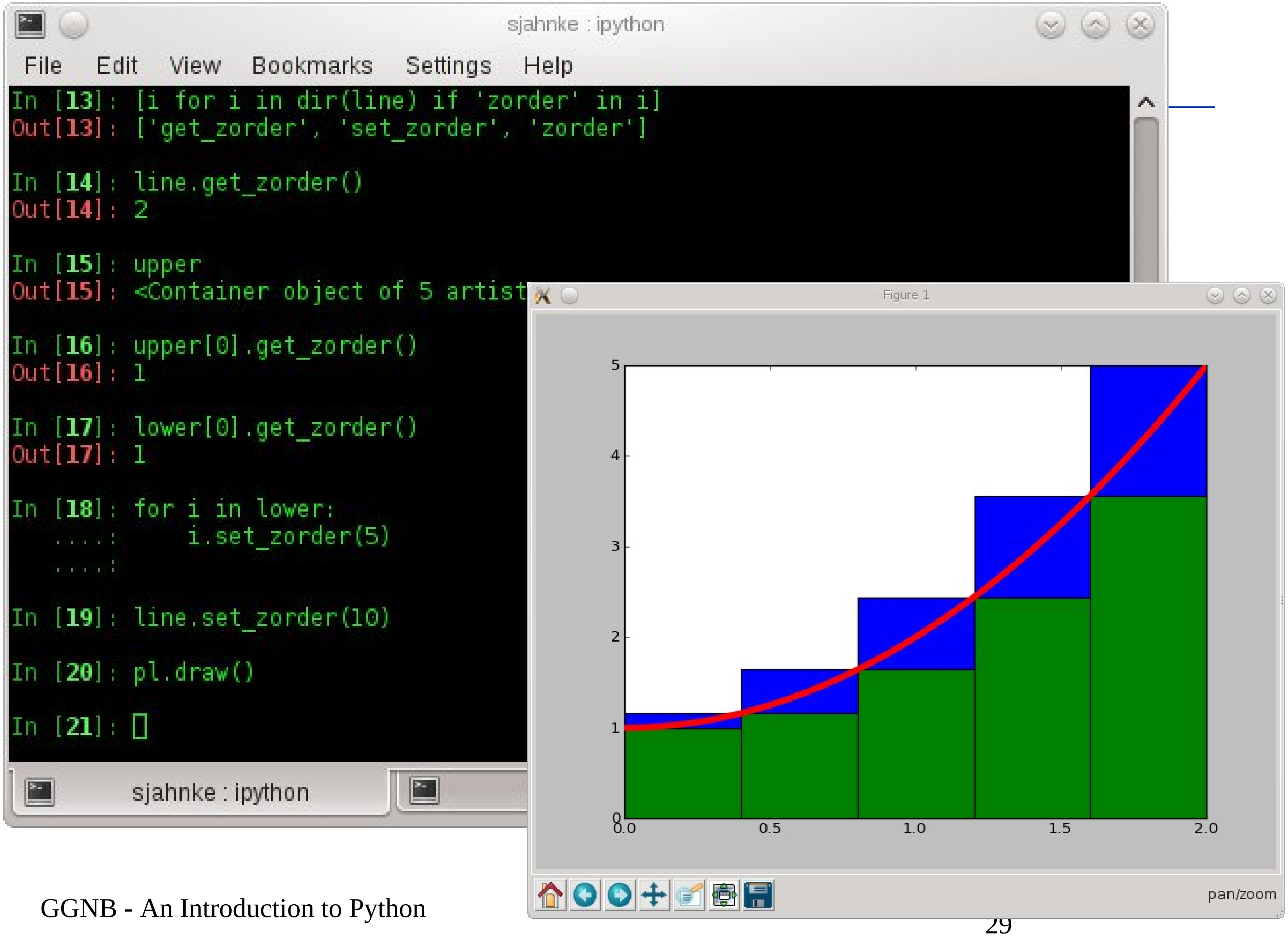
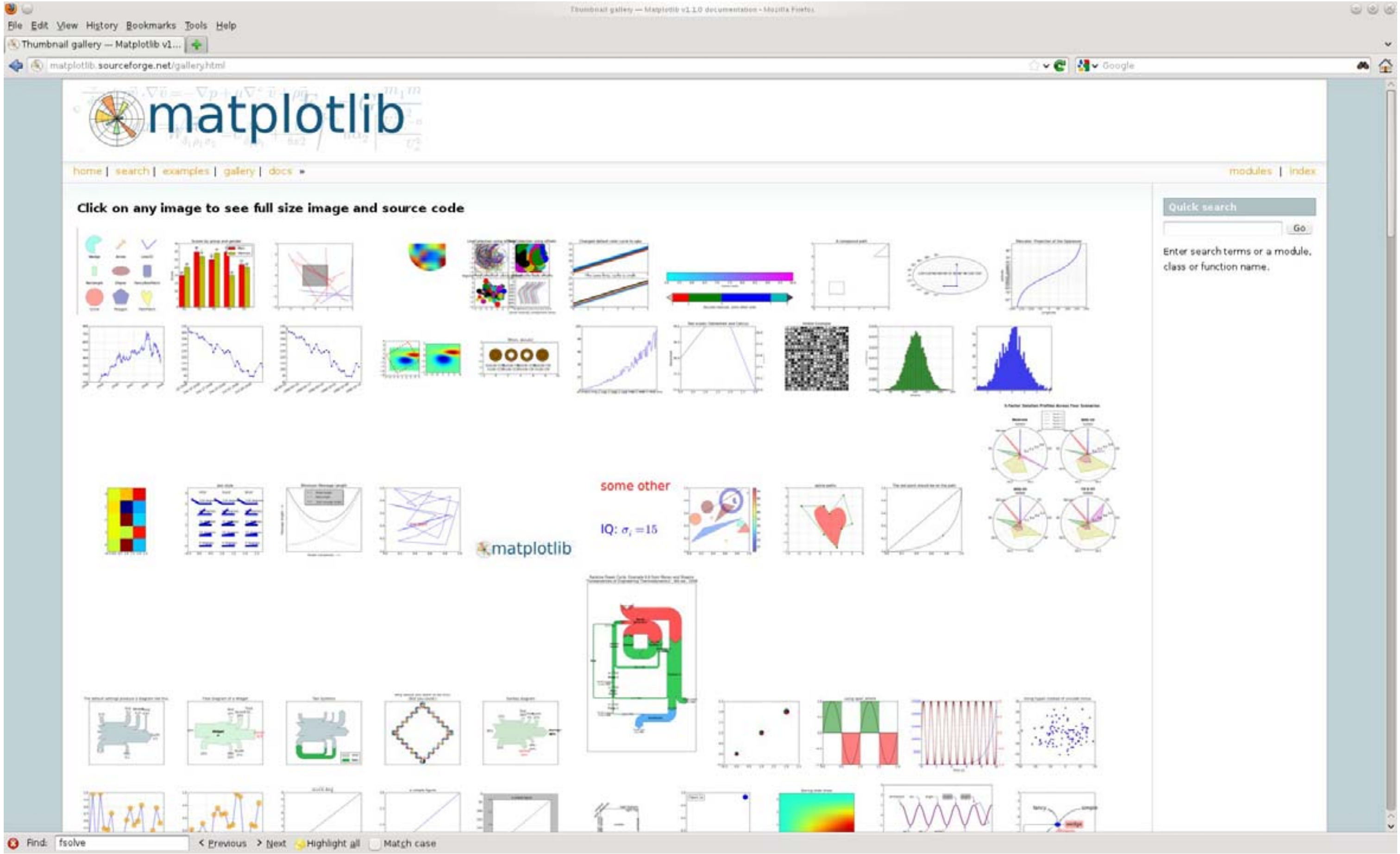


Figure 1
5
4
3
2
1
0
0.0
0.5
Figure 1
5
4
3
2
1
0
0.0
0.5
1.0
1.5
2.0



```
In [13]: [i for i in dir(line) if 'zorder' in i]
Out[13]: ['get_zorder', 'set_zorder', 'zorder']

In [14]: line.get_zorder()
Out[14]: 2

In [15]: upper
Out[15]: <Container object of 5 artist

In [16]: upper[0].get_zorder()
Out[16]: 1

In [17]: lower[0].get_zorder()
Out[17]: 1

In [18]: for i in lower:
   ....:     i.set_zorder(5)
   ....:

In [19]: line.set_zorder(10)

In [20]: pl.draw()

In [21]: 
```

The Gallery


Thumbnail gallery — Matplotlib v1.1.0 documentation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Thumbnail gallery — Matplotlib v1...
matplotlib.sourceforge.net/gallery.html
Google
matplotlib
home | search | examples | gallery | docs »
modules | index
Quick search
Go
some other
IQ: $\sigma_i = 15$
matplotlib
Find: fsolve
Previous
Next
Highlight all
Match case

Click on any image to see full size image and source code

Enter search terms or a module, class or function name.

http://matplotlib.sourceforge.net/gallery.html