Python Introduction II

Numpy, Scipy, and Matplotlib

NumPy Users Guide, http://docs.scipy.org/doc/numpy/user/

NumPy Reference, http://docs.scipy.org/doc/numpy/reference/

SciPy Reference, http://docs.scipy.org/doc/scipy/reference/index.html

SciPy Tutorial, http://docs.scipy.org/doc/scipy/reference/tutorial/

Matplotlib Users Guide, http://matplotlib.sourceforge.net/

further references:

http://www.c3se.chalmers.se/index.php/Python_and_High_Performance_Computing_2012#Lecture_3.2C_Numpy

http://matplotlib.org/resources/index.html

http://www.python-kurs.eu/numpy.php

http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf

Numpy

```
In [1]: import numpy as np
```

- arrays/vectors
- N-dimensional matrices
- fast (operations performed on arrays performed by compiled code)
- linear algebra, Fourier transforms

```
In [11]: help(np.dtype)

In [12]: b = np.array([1, 3.])

In [13]: b
Out[13]: array([ 1.,  3.])            # upcasting

In [14]: print b.dtype
float64

In [15]: c = np.array([1.3, 123.32 + 1.j], dtype=np.complex)

In [16]: c
Out[16]: array([   1.30+0.j,  123.32+1.j])

In [17]: print c.dtype
complex128
```

types:

- int8, int16, int32,int64, int
- uint8, uint16, uint32, uint64
- float16, float32, float64, float
- complex64, complex128

operations are defined per element:

```
In [3]: a = np.array([1,2,3])

In [4]: b = np.array([4,5,6])

In [5]: c = a*b + 3

In [6]: c
Out[6]: array([ 7, 13, 21])
```

shape:

```
In [20]: a = np.array([[1,2,3], [4,5,6]])

In [21]: a
Out[21]:
array([[1, 2, 3],
       [4, 5, 6]])

In [22]: a.shape
Out[22]: (2, 3)
```

Numpy

further functions on arrays:

```
In [23]: a.flatten()
Out[23]: array([1, 2, 3, 4, 5, 6])

In [24]: a.ravel()           # no copy is made
Out[24]: array([1, 2, 3, 4, 5, 6])

In [25]: a
Out[25]:
array([[1, 2, 3],
       [4, 5, 6]])

In [26]: a.reshape(3,2)
Out[26]:
array([[1, 2],
       [3, 4],
       [5, 6]])

In [27]: a.swapaxes(0,1)
Out[27]:
array([[1, 4],
       [2, 5],
       [3, 6]])
```

array creation: arange, linspace and meshgrid, ogrid

```
In [34]: a = np.arange(1,10,2)        # integer values; start, stop, increment

In [35]: a
Out[35]: array([1, 3, 5, 7, 9])


In [49]: b= np.linspace(-1,1,3) # any type, endpoint inclusive; start, stop, nr of points

In [50]: print b
[-1.  0.  1.]
```

Numpy

```
In [43]: X,Y = np.meshgrid(b,b)       # create a 2D point array from 1D vectors

In [44]: print (X,Y)
(array([[-1.,  0.,  1.],
       [-1.,  0.,  1.],
       [-1.,  0.,  1.]]), array([[-1., -1., -1.],
       [ 0.,  0.,  0.],
       [ 1.,  1.,  1.]]))

In [45]: print X
[[-1.  0.  1.]
 [-1.  0.  1.]
 [-1.  0.  1.]]

In [46]: print X[0],Y[0]
[-1.  0.  1.] [-1. -1. -1.]

In [47]: print X[0][0],Y[0][0]
-1.0 -1.0

In [48]: print X[1][1],Y[1][1]
0.0 0.0
```

```
In [71]: X1, Y1 = np.ogrid[-1:1:3j, 4:6:3j] # complex number means inclusive

In [72]: print X1    # note different output than in meshgrid
[[-1.]
 [ 0.]
 [ 1.]]

In [73]: print Y1    # note different output than in meshgrid
[[ 4.  5.  6.]]
```

```
In [74]: X1.shape, Y1.shape
Out[74]: ((3, 1), (1, 3))
```

Numpy

array creation: zeros(shape), zeros_like(vec), ones(shape), ones_like(vec), eye(dim), empty(shape), empty_like(arr)

```
In [81]: a = np.zeros((2,4))

In [82]: a
Out[82]:
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])

In [83]: b = np.ones_like(a)

In [84]: b
Out[84]:
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
```

```
In [86]: c = np.eye(3)

In [87]: c
Out[87]:
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

```
In [88]: d = np.empty(400)

In [89]: d.shape
Out[89]: (400,)
```

Numpy

Indexing

```
In [90]: a = np.arange(4,16)

In [91]: a
Out[91]: array([ 4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])

In [92]: a[2]
Out[92]: 6

In [93]: a[:2]
Out[93]: array([4, 5])

In [94]: a[:-2]
Out[94]: array([ 4,  5,  6,  7,  8,  9, 10, 11, 12, 13])

In [95]: a[2:4]
Out[95]: array([6, 7])

In [96]: a[-2:]
Out[96]: array([14, 15])

In [101]: a[::-4]
Out[101]: array([15, 11,  7])

In [102]: a[::-1]
Out[102]: array([15, 14, 13, 12, 11, 10,  9,  8,  7,  6,  5,  4])

In [103]: a[::-3]
Out[103]: array([15, 12,  9,  6])
```

Numpy

Indexing

```
In [105]: a = np.random.random(9)

In [106]: a
Out[106]:
array([ 0.50770486,  0.77773751,  0.53964056,  0.39399036,  0.15385062,
        0.58156581,  0.12979955,  0.69665401,  0.01364413])

In [107]: a
Out[107]:
array([ 0.50770486,  0.77773751,  0.53964056,  0.39399036,  0.15385062,
        0.58156581,  0.12979955,  0.69665401,  0.01364413])

In [108]: index = np.arange(2,10,3)

In [109]: a[index]
Out[109]: array([ 0.53964056,  0.58156581,  0.01364413])

In [110]: index
Out[110]: array([2, 5, 8])

In [111]: mask = a > 0.3

In [112]: mask
Out[112]: array([ True,  True,  True,  True, False,  True, False,  True, False], dtype=bool)

In [113]: a[mask]
Out[113]:
array([ 0.50770486,  0.77773751,  0.53964056,  0.39399036,  0.58156581,
        0.69665401])
```

Numpy

Indexing

```
In [118]: a
Out[118]:
array([ 0.50770486,  0.77773751,  0.53964056,  0.39399036,  0.15385062,
        0.58156581,  0.12979955,  0.69665401,  0.01364413])

In [119]: b = a.reshape(3,3)

In [120]: b
Out[120]:
array([[ 0.50770486,  0.77773751,  0.53964056],
       [ 0.39399036,  0.15385062,  0.58156581],
       [ 0.12979955,  0.69665401,  0.01364413]])

In [121]: b[:,1]
Out[121]: array([ 0.77773751,  0.15385062,  0.69665401])

In [122]: b[:1, 1:]
Out[122]: array([[ 0.77773751,  0.53964056]])


In [125]: b[:1, :1]
Out[125]: array([[ 0.50770486]])

In [126]: b[:2, :2]
Out[126]:
array([[ 0.50770486,  0.77773751],
       [ 0.39399036,  0.15385062]])


In [128]: b[1:3, 1:3]
Out[128]:
array([[ 0.15385062,  0.58156581],
       [ 0.69665401,  0.01364413]])

In [129]: b[::2, ::2]
Out[129]:
array([[ 0.50770486,  0.53964056],
       [ 0.12979955,  0.01364413]])
```

Numpy

Search and sort:where and argsort

```
In [139]: x = np.linspace(0, 2*np.pi, 101)

In [140]: y = np.cos(x)

In [141]: idx = np.where(y>0)         # where(boolarray, truearray, falsearray)

In [142]: x = x[idx]

In [143]: x/np.pi
Out[143]:
array([ 0.  ,  0.02,  0.04,  0.06,  0.08,  0.1 ,  0.12,  0.14,  0.16,
        0.18,  0.2 ,  0.22,  0.24,  0.26,  0.28,  0.3 ,  0.32,  0.34,
        0.36,  0.38,  0.4 ,  0.42,  0.44,  0.46,  0.48,  1.52,  1.54,
        1.56,  1.58,  1.6 ,  1.62,  1.64,  1.66,  1.68,  1.7 ,  1.72,
        1.74,  1.76,  1.78,  1.8 ,  1.82,  1.84,  1.86,  1.88,  1.9 ,
        1.92,  1.94,  1.96,  1.98,  2.  ])
```

```
In [144]: a
Out[144]:
array([ 0.50770486,  0.77773751,  0.53964056,  0.39399036,  0.15385062,
        0.58156581,  0.12979955,  0.69665401,  0.01364413])

In [145]: idx = np.argsort(a)

In [146]: a[idx]
Out[146]:
array([ 0.01364413,  0.12979955,  0.15385062,  0.39399036,  0.50770486,
        0.53964056,  0.58156581,  0.69665401,  0.77773751])

In [147]: idx
Out[147]: array([8, 6, 4, 3, 0, 2, 5, 7, 1])
```

Numpy

broadcasting:

operations are usually done elementwise, but this is relaxed when

- dimension is 1
- trailing dimensions match
- empty dimensions with np.newaxis

```
In [154]: A = np.ones((10,12))

In [155]: A
Out[155]:
array([[ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.]])
```

```
In [158]: B = np.ones((12,3))*2

In [159]: B
Out[159]:
array([[ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.],
       [ 2.,  2.,  2.]])
```

```
In [160]: C = A[:,:, np.newaxis] * B[np.newaxis, :, :]

In [162]: C.shape
Out[162]: (10, 12, 3)

In [183]: A[1,:] + B[:,1]
Out[183]: array([ 3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.,  3.])
```

Numpy

all standard mathematical functions (exp, log, cos, …)

max, min, mean, median,…

sum, prod, …

dot, cross, convolve, …

```
In [193]: b = np.arange(9)

In [195]: b = b.reshape(3,3)

In [196]: b
Out[196]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])

In [197]: np.sum(b, axis=0)
Out[197]: array([ 9, 12, 15])

In [198]: np.sum(b, axis=1)
Out[198]: array([ 3, 12, 21])

In [199]: np.prod(b, axis=0)
Out[199]: array([ 0, 28, 80])
```

```
In [220]: a = np.arange(-1,2)

In [221]: b = np.arange(4,7)

In [222]: a
Out[222]: array([-1,  0,  1])

In [223]: b
Out[223]: array([4, 5, 6])

In [224]: np.dot(a,b)
Out[224]: 2

In [225]: np.cross(a,b)
Out[225]: array([-5, 10, -5])
```

Numpy

Matrix operations cover most of lapack

```
In [226]: A = np.random.random((3,3))

In [227]: A
Out[227]:
array([[ 0.03395889,  0.49275271,  0.97400033],
       [ 0.23546018,  0.52048433,  0.57272703],
       [ 0.46077877,  0.1053315 ,  0.0434009 ]])

In [228]: A_inv = np.linalg.inv(A)

In [229]: A_inv
Out[229]:
array([[ 0.44025539, -0.94740172,  2.6219194 ],
       [-2.95957303,  5.21871601, -2.44867185],
       [ 2.50861006, -2.60714873,  1.14738381]])
```

```
In [231]: np.dot(A, A_inv)
Out[231]:
array([[  1.00000000e+00,   0.00000000e+00,  -1.38777878e-17],
       [  2.22044605e-16,   1.00000000e+00,  -2.22044605e-16],
       [  1.38777878e-17,  -1.38777878e-17,   1.00000000e+00]])

In [232]: np.mat(A) * np.mat(A_inv)
Out[232]:
matrix([[  1.00000000e+00,   0.00000000e+00,  -1.38777878e-17],
        [  2.22044605e-16,   1.00000000e+00,  -2.22044605e-16],
        [  1.38777878e-17,  -1.38777878e-17,   1.00000000e+00]])

In [233]: A * A_inv
Out[233]:
array([[ 0.01495058, -0.46683476,  2.55375037],
       [-0.6968616 ,  2.71625991, -1.40242054],
       [ 1.15591426, -0.2746149 ,  0.04979749]])

In [234]: A.T
Out[234]:
array([[ 0.03395889,  0.23546018,  0.46077877],
       [ 0.49275271,  0.52048433,  0.1053315 ],
       [ 0.97400033,  0.57272703,  0.0434009 ]])
```

```
In [241]: np.linalg.eig(A)
Out[241]:
(array([-0.60371055,  1.06870072,  0.13285395]),
 array([[-0.7969552 ,  0.65654684,  0.28150824],
        [-0.13323116,  0.66120214, -0.84440718],
        [ 0.589162  ,  0.36298481,  0.45577366]]))

In [242]: np.linalg.eigvals(A)
Out[242]: array([-0.60371055,  1.06870072,  0.13285395])

In [243]: np.linalg.svd(A)
Out[243]:
(array([[-0.79962645, -0.34538097, -0.49123266],
        [-0.58737293,  0.27976269,  0.7594247 ],
        [-0.12486227,  0.89579284, -0.42657308]]),
 array([ 1.34677899,  0.49165126,  0.12945113]),
 array([[-0.16557375, -0.52932845, -0.8321038 ],
        [ 0.9496702 ,  0.14192966, -0.27925345],
        [-0.26591701,  0.83646122, -0.47918763]]))
```

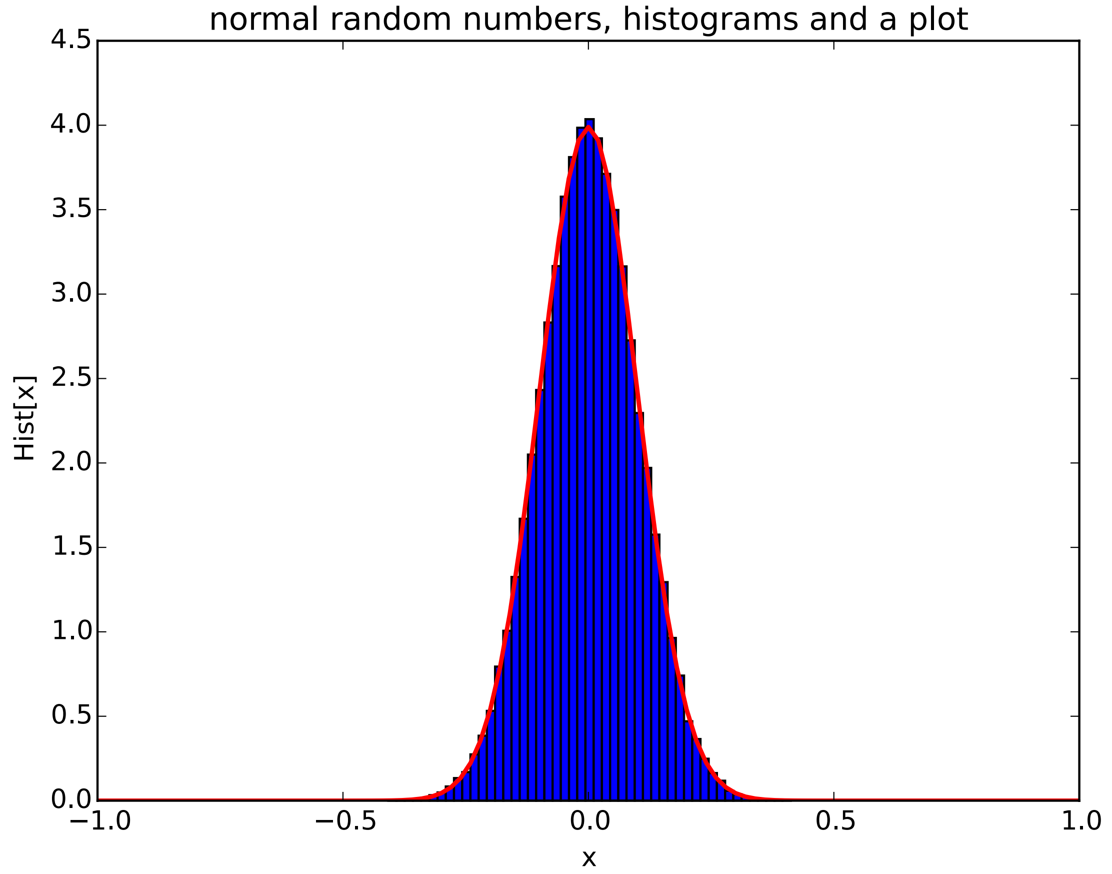
```
In [256]: Q,R = np.linalg.qr(A)

In [257]: np.dot(Q,R)
Out[257]:
array([[ 0.03395889,  0.49275271,  0.97400033],
       [ 0.23546018,  0.52048433,  0.57272703],
       [ 0.46077877,  0.1053315 ,  0.0434009 ]])

In [258]: Q
Out[258]:
array([[-0.06548601, -0.7475938 , -0.66091988],
       [-0.45405928, -0.56747224,  0.68688093],
       [-0.88856162,  0.3450779 , -0.30229041]])

In [259]: R
Out[259]:
array([[-0.51856704, -0.36219268, -0.3623998 ],
       [ 0.        , -0.62739171, -1.03818661],
       [ 0.        ,  0.        , -0.26346059]])
```

Numpy

Also has an FFT and a random number (Mersenne Twister) generator

```
In [266]: help(np.random.random)
```

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

mu,sigma = 0., 0.1
s = np.random.normal(mu, sigma, 100000)

xp = np.linspace(-1,1,101)
yp = np.exp(-xp*xp/0.02)/np.sqrt(2 * np.pi * 0.01)

print xp
print yp


plt.hist(s, bins=50,normed=1, facecolor='blue', alpha=0.5)
plt.plot(xp,yp, 'r-', linewidth=2)
plt.xlabel('x')
plt.ylabel('Hist[x]')
plt.title('normal random numbers, histograms and a plot')
plt.savefig('py_intro2.eps', dpi=1000)
plt.show()
```



Polynomials: roots, polyfit, polyval, polyder, polyint (see the documentation)

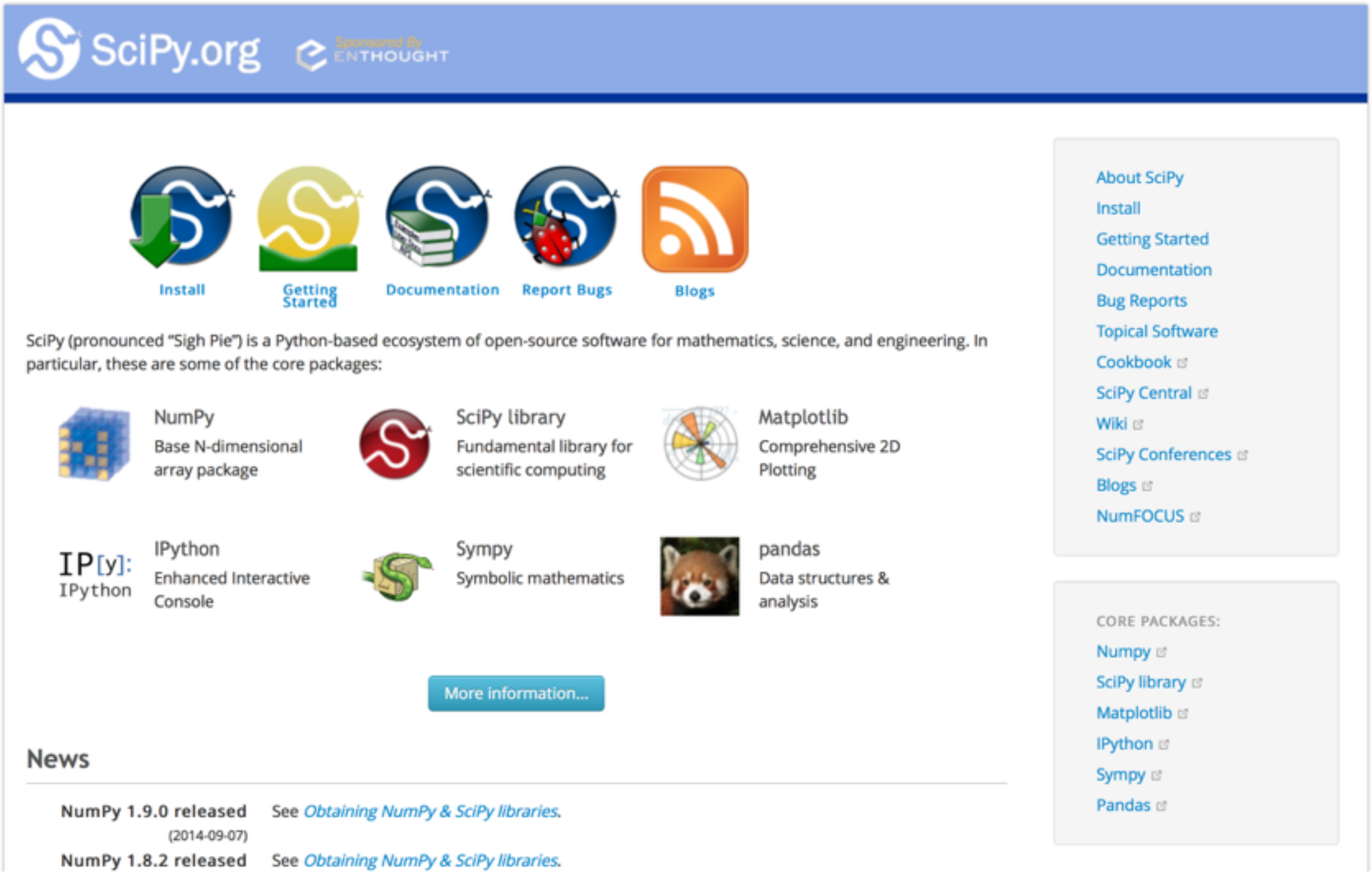
Numpy

Fast coding:

Tensor product: $C_{ij} = A_{ik} B_{kj}$

```
import time
import numpy as np

def do_not_use_me(a,b,c):
        for i in np.arange(a.shape[0]):
                for k in np.arange(a.shape[1]):
                        for m in np.arange(b.shape[1]):
                                c[i,m] += a[i,k] * b[k,m]


A = np.random.random((100,120))
B = np.random.random((120,30))
C = np.zeros((100,30))

t0 = time.clock()
# nested for loops
do_not_use_me(A,B,C)
t1 = time.clock()
#print C

# broadcasting and np.sum
t2 = time.clock()
Cbis = np.sum(A[:,:,np.newaxis] * B[np.newaxis,:,:], axis=1)
t3 = time.clock()
#print Cbis

# np.tensordot
t4 = time.clock()
Ctres = np.tensordot(A,B, axes=([1],[0]))
t5 = time.clock()
#print Ctres

# matrix product using np.mat
t6 = time.clock()
Cquad = np.array(np.mat(A) * np.mat(B) )
t7 = time.clock()
#print Cquad


print 'for loops    :', t1-t0
print 'broadcast    :', t3-t2
print 'np.tensordot :', t5-t4
print 'np.mat       :', t7-t6
```

Scipy


SciPy.org
Sponsored By ENTHOUGHT
Install
Getting Started
Documentation
Report Bugs
Blogs
SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:
NumPy
Base N-dimensional array package
SciPy library
Fundamental library for scientific computing
Matplotlib
Comprehensive 2D Plotting
IP[y]: IPython
IPython
Enhanced Interactive Console
Sympy
Symbolic mathematics
pandas
Data structures & analysis
More information...
News
NumPy 1.9.0 released (2014-09-07)
See Obtaining NumPy & SciPy libraries.
NumPy 1.8.2 released
See Obtaining NumPy & SciPy libraries.
About SciPy
Install
Getting Started
Documentation
Bug Reports
Topical Software
Cookbook
SciPy Central
Wiki
SciPy Conferences
Blogs
NumFOCUS
CORE PACKAGES:
Numpy
SciPy library
Matplotlib
IPython
Sympy
Pandas

Scipy

Scipy.org | Docs

index | modules | next

SciPy

Release:	0.14.0
Date:	May 11, 2014

SciPy (pronounced "Sigh Pie") is open-source software for mathematics, science, and engineering.

- SciPy Tutorial
  - Introduction
  - Basic functions
  - Special functions (**`scipy.special`**)
  - Integration (**`scipy.integrate`**)
  - Optimization (**`scipy.optimize`**)
  - Interpolation (**`scipy.interpolate`**)
  - Fourier Transforms (**`scipy.fftpack`**)
  - Signal Processing (**`scipy.signal`**)
  - Linear Algebra (**`scipy.linalg`**)
  - Sparse Eigenvalue Problems with ARPACK
  - Compressed Sparse Graph Routines (**`scipy.sparse.csgraph`**)
  - Spatial data structures and algorithms (**`scipy.spatial`**)
  - Statistics (**`scipy.stats`**)
  - Multidimensional image processing (**`scipy.ndimage`**)
  - File IO (**`scipy.io`**)
  - Weave (**`scipy.weave`**)
- Contributing to SciPy
- API - importing from Scipy
- Release Notes

Table Of Contents

Next topic

SciPy Tutorial

Scipy

Special functions (scipy.special)

The main feature of the scipy.special package is the definition of numerous special functions of mathematical physics. Available functions include airy, elliptic, bessel, gamma, beta, hypergeometric, parabolic cylinder, mathieu, spheroidal wave, struve, and kelvin. There are also some low-level stats functions that are not intended for general use as an easier interface to these functions is provided by the `stats` module. Most of these functions can take array arguments and return array results following the same broadcasting rules as other math functions in Numerical Python. Many of these functions also accept complex numbers as input. For a complete list of the available functions with a one-line description type `>>> help(special).` Each function also has its own documentation accessible using help. If you don't see a function you need, consider writing it and contributing it to the library. You can write the function in either C, Fortran, or Python. Look in the source code of the library for examples of each of these kinds of functions.

Table Of Contents

Previous topic

Basic functions

Next topic

Integration (`scipy.integrate`)

```
>>> from scipy import special
>>> help(special)
```

Scipy

Scipy.org | Docs | SciPy v0.14.0 Reference Guide | SciPy Tutorial

index | modules | next | previous

Integration (scipy.integrate)

The scipy.integrate sub-package provides several integration techniques including an ordinary differential equation integrator. An overview of the module is provided by the help command:

```
>>> help(integrate)
 Methods for Integrating Functions given function object.

   quad          -- General purpose integration.
   dblquad       -- General purpose double integration.
   tplquad       -- General purpose triple integration.
   fixed_quad    -- Integrate func(x) using Gaussian quadrature of order n.
   quadrature    -- Integrate with given tolerance using Gaussian quadrature.
   romberg       -- Integrate func using Romberg integration.

 Methods for Integrating Functions given fixed samples.

   trapz         -- Use trapezoidal rule to compute integral from samples.
   cumtrapz      -- Use trapezoidal rule to cumulatively compute integral.
   simps         -- Use Simpson's rule to compute integral from samples.
   romb          -- Use Romberg Integration to compute integral from
                    (2**k + 1) evenly-spaced samples.

   See the special module's orthogonal polynomials (special) for Gaussian
      quadrature roots and weights for other weighting factors and regions.

 Interface to numerical integrators of ODE systems.

   odeint        -- General integration of ordinary differential equations.
   ode           -- Integrate ODE using VODE and ZVODE routines.
```

Table Of Contents

Previous topic

Special functions (**scipy.special**)

Next topic

Optimization (**scipy.optimize**)

Scipy

Scipy.org | Docs | SciPy v0.14.0 Reference Guide | SciPy Tutorial

Linear Algebra (scipy.linalg)

When SciPy is built using the optimized ATLAS LAPACK and BLAS libraries, it has very fast linear algebra capabilities. If you dig deep enough, all of the raw lapack and blas libraries are available for your use for even more speed. In this section, some easier-to-use interfaces to these routines are described.

All of these linear algebra routines expect an object that can be converted into a 2-dimensional array. The output of these routines is also a two-dimensional array.

scipy.linalg vs numpy.linalg¶

`scipy.linalg` contains all the functions in `numpy.linalg`. plus some other more advanced ones not contained in `numpy.linalg`

Another advantage of using `scipy.linalg` over `numpy.linalg` is that it is always compiled with BLAS/LAPACK support, while for numpy this is optional. Therefore, the scipy version might be faster depending on how numpy was installed.

Therefore, unless you don't want to add `scipy` as a dependency to your `numpy` program, use `scipy.linalg` instead of `numpy.linalg`

Scipy

index | modules | next | previous

Table Of Contents

Previous topic

Signal Processing (**`scipy.signal`**)

Next topic

Sparse Eigenvalue Problems with ARPACK

QR decomposition

The QR decomposition (sometimes called a polar decomposition) works for any $M \times N$ array and finds an $M \times M$ unitary matrix $\mathbf{Q}$ and an $M \times N$ upper-trapezoidal matrix $\mathbf{R}$ such that

$$\mathbf{A} = \mathbf{QR}.$$

Notice that if the SVD of $\mathbf{A}$ is known then the QR decomposition can be found

$$\mathbf{A} = \mathbf{U\Sigma V}^H = \mathbf{QR}$$

implies that $\mathbf{Q} = \mathbf{U}$ and $\mathbf{R} = \mathbf{\Sigma V}^H$. Note, however, that in SciPy independent algorithms are used to find QR and SVD decompositions. The command for QR decomposition is linalg.qr .

Matplotlib

matplotlib

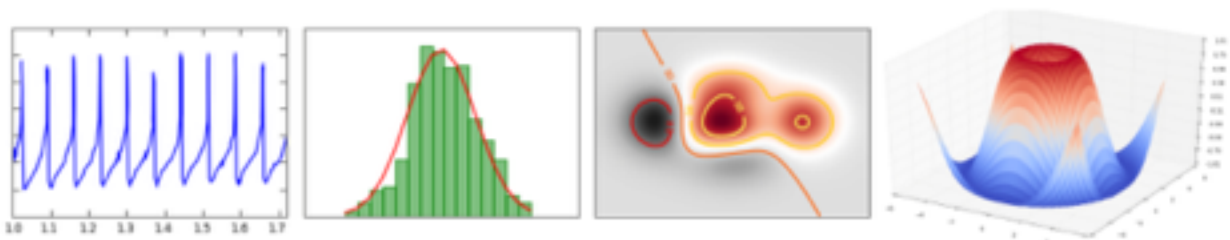
home | examples | gallery | pyplot | docs »

modules | index

Fork me on GitHub

Introduction

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB®* or Mathematica®†), web application servers, and six graphical user interface toolkits.



matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory

For simple plotting the `pyplot` interface provides a MATLAB-like interface, particularly when combined with `IPython`. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Download

Visit the matplotlib downloads page.

Documentation

This is the documentation for matplotlib version 1.4.1.

Other versions are available:

- 1.4.1 Latest stable version.
- 1.4.0 Previous stable version.
- 1.3.1 Older stable version.

Trying to learn how to do a particular kind of plot? Check out the gallery, examples, or the list of plotting commands.

John Hunter (1968-2012)

On August 28 2012, John D. Hunter, the creator of matplotlib, died from complications arising from cancer treatment, after a brief but intense battle with this terrible illness. John is survived by his wife Miriam, his three daughters Rahel, Ava and Clara, his sisters Layne and Mary, and his mother Sarah.

If you have benefited from John's many contributions, please say thanks in the way that would matter most to him. Please consider making a donation to the John Hunter Memorial Fund.

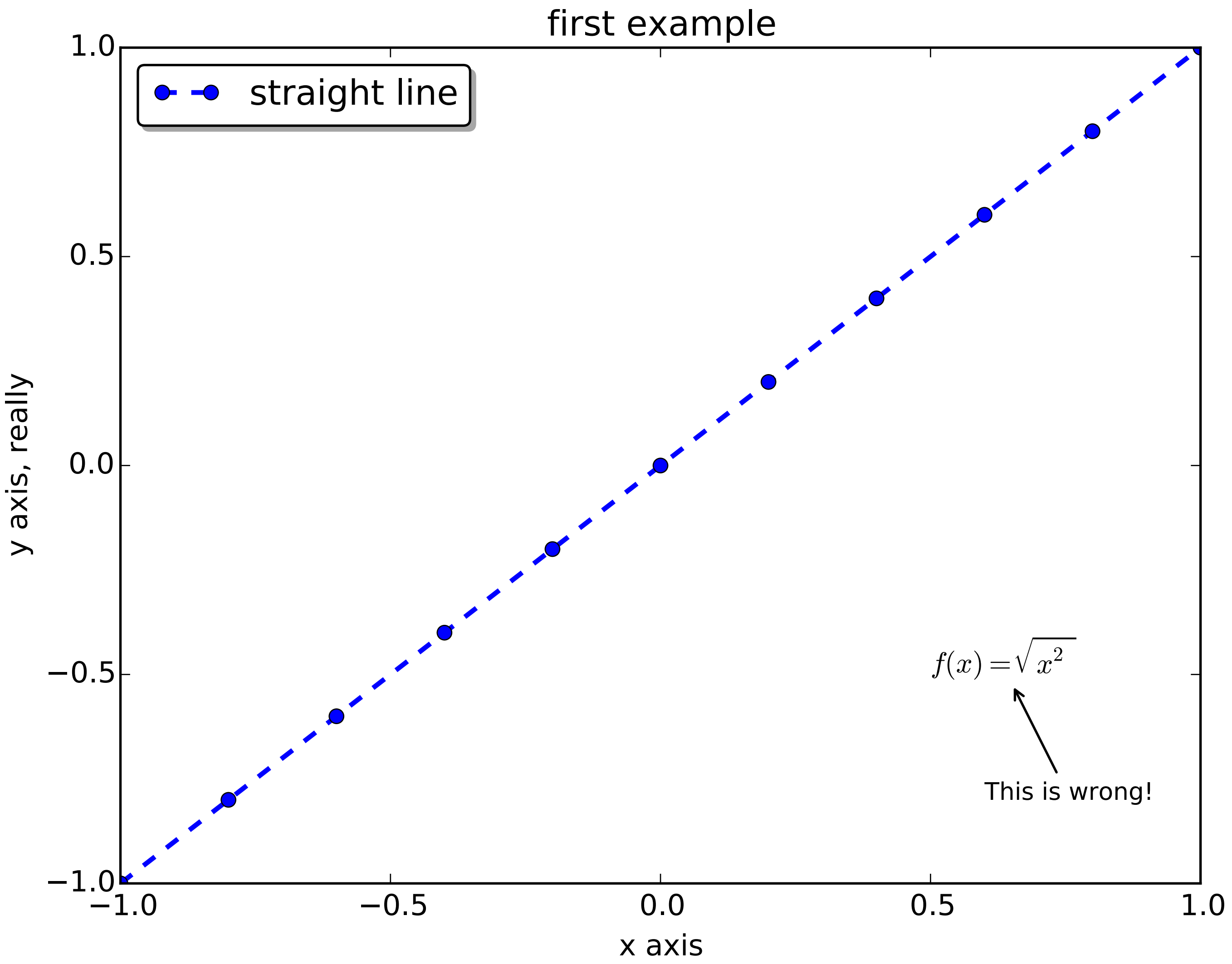
Quick search

Go

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

Matplotlib

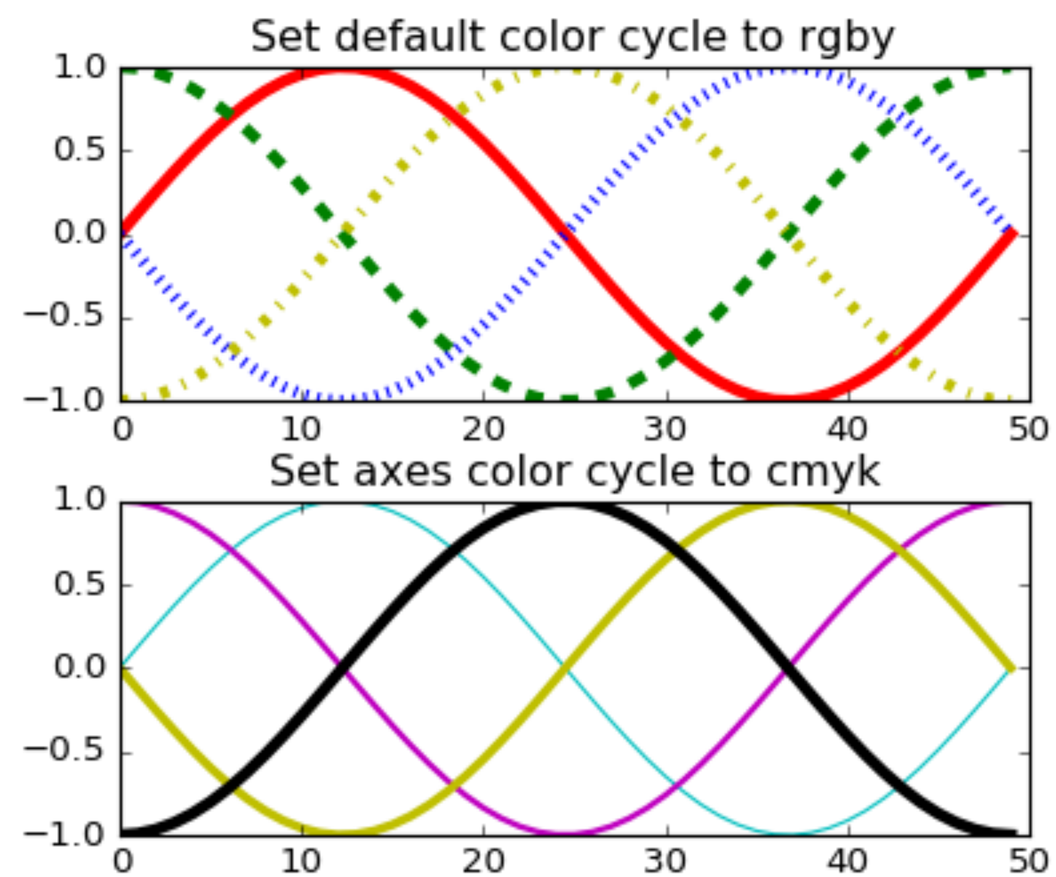
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1,11)

plt.plot(x,x,'--o', label="straight line", linewidth=2)
plt.xlabel('x axis')
plt.ylabel('y axis, really')
plt.title('first example')
plt.text(0.5,-0.5, r'$f(x)=\sqrt{x^2}$', fontsize=12)
plt.annotate('This is wrong!', xy = (0.65, -0.52), xytext = (0.6, -0.8), fontsize=10, arrowprops = dict(arrowstyle='->'))
plt.legend(loc="upper left", shadow=True, fancybox=True)
plt.savefig('plt_ex1.pdf')
plt.show()
```



Matplotlib

from official website: color maps and subplots; handles

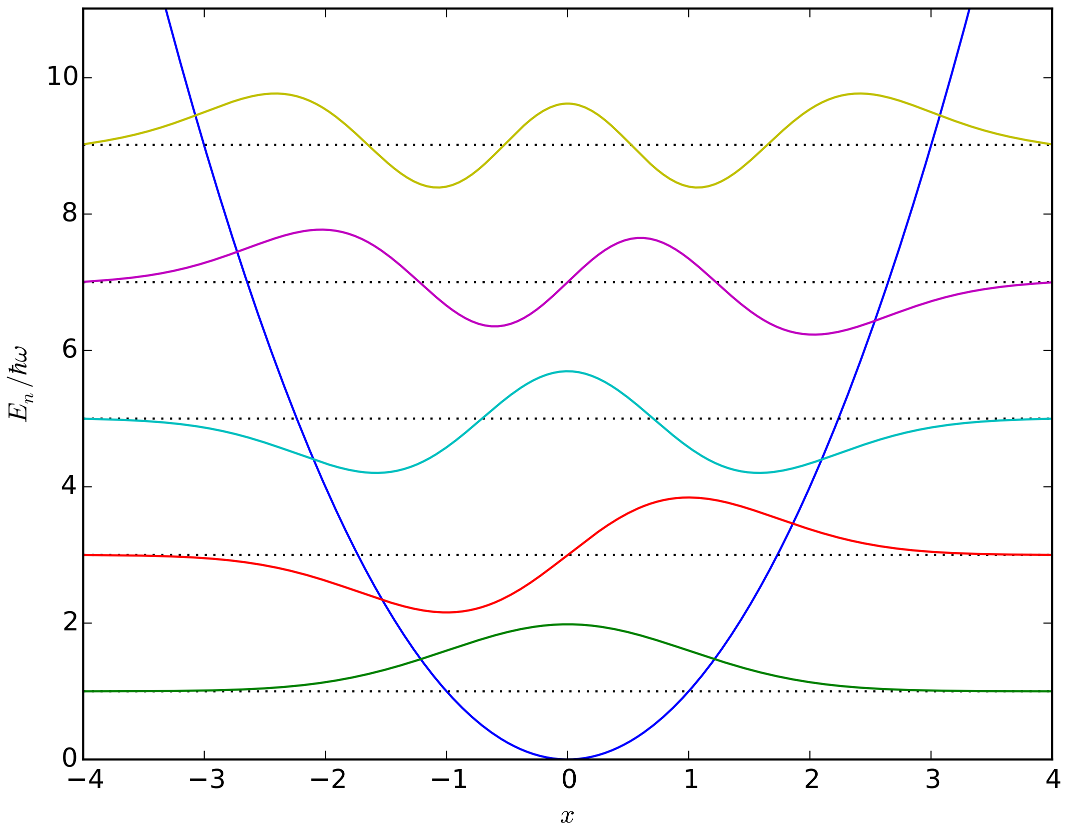
```
"""
Demo of custom property-cycle settings to control colors and such
for multi-line plots.

This example demonstrates two different APIs:

    1. Setting the default rc-parameter specifying the property cycle
       This affects all subsequent axes (but not axes already created
    2. Setting the property cycle for a specific axes. This only
       affects a single axes.
"""
from cycler import cycler
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 * np.pi)
offsets = np.linspace(0, 2*np.pi, 4, endpoint=False)
# Create array with shifted-sine curve along each column
yy = np.transpose([np.sin(x + phi) for phi in offsets])

plt.rc('lines', linewidth=4)
plt.rc('axes', prop_cycle=(cycler('color', ['r', 'g', 'b', 'y']) +
                           cycler('linestyle', ['-', '--', ':', '-.'])
fig, (ax0, ax1) = plt.subplots(nrows=2)
ax0.plot(yy)
ax0.set_title('Set default color cycle to rgby')

ax1.set_prop_cycle(cycler('color', ['c', 'm', 'y', 'k']) +
                   cycler('lw', [1, 2, 3, 4]))
ax1.plot(yy)
ax1.set_title('Set axes color cycle to cmyk')

# Tweak spacing between subplots to prevent labels from overlapping
plt.subplots_adjust(hspace=0.3)
plt.show()
```



Matplotlib

harmonic oscillator showcase (thanks to H. Strand)

```python
import numpy as np
import scipy.sparse as sparse
import scipy.sparse.linalg as splinalg
import matplotlib.pyplot as plt

N = 300
x = np.linspace(-4.0, 4.0, num=N); dx = x[1] - x[0]
d2_dx2 = - np.array([1., -2., 1.]) / dx**2
H_diags = np.zeros((3, N)) + d2_dx2[:, np.newaxis]
H_diags[1, :] += x**2 # Add potential on middle diagonal

H = sparse.spdiags(H_diags, [-1,0,1], N, N).tocsr()
eigv, eigvec = splinalg.eigsh(H, 5, which='SM', tol=1e-4, maxiter=10*N)


plt.plot(x, x**2)
for idx in xrange(eigvec.shape[-1]):
    plt.plot(x, eigv[idx]+np.zeros_like(x), ':k')
    plt.plot(x, 8*np.ravel(eigvec[:, idx]) + eigv[idx])
plt.ylim([0, eigv[-1]+2])
plt.ylabel(r'$E_n / \hbar \omega$')
plt.xlabel(r'$x$')
plt.savefig('plt_harmosc.pdf')
plt.show()
```



other possibilities: contour plots, 3D, … (see documentation)

almost endless number of possibilities

Recommended modules

(thanks to P. Kroiss)

- from __future__ import braces   # (C-style braces instead of indentation)
- import this
- import antigravity

Further references

B. Slatkin: Effective Python
http://www.effectivepython.com/

Anaconda
https://www.continuum.io/downloads

Ipython Notebook
http://ipython.org/notebook.html

PEP 8
https://www.python.org/dev/peps/pep-0008/

Hidden Python features
http://stackoverflow.com/questions/101268/hidden-features-of-python