AstroInformatics

Introduzione a Python

Stefano Cavuoti
Massimo Brescia

Python

- General purpose interpreted programming language
- Widely used by scientists and programmers of all stripes
- Supported by many 3rd-party libraries (currently 21,054 on the main python package website)
- Free!

Snape

Why is it well-suited to science?

- Numpy
  - Numerical library for python
  - Written in C, wrapped by python
  - Fast
- Scipy
  - Built on top of numpy (i.e. Also fast!)
  - Common maths, science, engineering routines
- Matplotlib
  - Hugely flexible plotting library
  - Similar syntax to Matlab
  - Produces publication-quality output
- Pyfits
  - Handle fits files
- Scikit-learn
  - Several machine learning methods
- …

What is Python not?

- An integrated graphical environment like Matlab (although there are tools which put it in one – e.g. Spyder)
- Specifically designed for scientists/mathematicians (but the 3rd-party libraries for plotting/numerical work are some of the best around)
- High performance (but it is very easy to wrap C/Fortran libraries in Python code)

Who uses Python?

- Netflix
- Yahoo Maps/Groups
- Google
- NASA
- ESRI
- Linux distros
- Multiplayer.it
- CIA
- Civilization 4

Who uses Python?

- Netflix
- Yahoo Maps/Groups
- Google
- NASA
- ESRI
- Linux distros
- Multiplayer.it
- CIA
- Civilization 4
- **Stefano Cavuoti**

Something cool that Python can do.

- It can do everything
  - Fast mathematical operations
  - Easy file manipulation
  - Format conversion
  - Plotting
  - Scripting
  - Command line

Something cool that Python can do.

- It can do everything
  - Fast mathematical operations
  - Easy file manipulation
  - Format conversion
  - Plotting
  - Scripting
  - Command line

- OK, not everything
  - ~~Write papers for you~~

How can I get Python?

- Windows – Python(x,y) [www.pythonxy.com] This is a scientific/engineering oriented distribution of python. It includes everything you need to get started

- Linux – it's already there! Unless you're running a very unusual distro (in which case you probably already know what you're doing).

- Mac – it's already there on OS X, but it's old. Get a more up-to-date one [www.python.org]

How can I learn Python?

- The official python tutorial: http://docs.python.org/tutorial/
- Software Carpentry: http://software-carpentry.org/
- Dive into Python: http://www.diveintopython.net/
- Learn Python the Hard Way: http://learnpythonthehardway.org/
- A Byte of Python: http://www.ibiblio.org/g2swap/byteofpython/read/
- Pensare da informatico: http://www.python.it/doc/Howtothink/HowToThink_ITA.pdf

Scientific Python?

- Extra features required:
  - fast, multidimensional arrays
  - libraries of reliable, tested scientific functions
  - plotting tools
- NumPy is at the core of nearly every scientific Python application or module since it provides a fast N-d array datatype that can be manipulated in a vectorized form.

Hereafter, slides are related to further examples of using Python

Arrays – Numerical Python (Numpy)

- Lists ok for storing small amounts of one-dimensional data

```
>>> a = [1,3,5,7,9]
>>> print(a[2:4])
[5, 7]
>>> b = [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
>>> print(b[0])
[1, 3, 5, 7, 9]
>>> print(b[1][2:4])
[6, 8]
```

```
>>> a = [1,3,5,7,9]
>>> b = [3,5,6,7,9]
>>> c = a + b
>>> print c
[1, 3, 5, 7, 9, 3, 5, 6, 7, 9]
```

- But, can’t use directly with arithmetical operators (+, -, *, /, …)
- Need efficient arrays with arithmetic and better multidimensional tools
- **Numpy**

```
>>> import numpy
```

- Similar to lists, but much more capable, except fixed size

Numpy – N-dimensional Array manipulations

The fundamental library needed for scientific computing with Python is called NumPy. This Open Source library contains:

- a powerful N-dimensional array object
- advanced array slicing methods (to select array elements)
- convenient array reshaping methods

and it even contains 3 libraries with numerical routines:

- basic linear algebra functions
- basic Fourier transforms
- sophisticated random number capabilities

NumPy can be extended with C-code for functions where performance is highly time critical. In addition, tools are provided for integrating existing Fortran code. NumPy is a hybrid of the older NumArray and Numeric packages, and is meant to replace them both.

Numpy – Creating arrays

- There are a number of ways to initialize new numpy arrays, for example from
  - a Python list or tuples
  - using functions that are dedicated to generating numpy arrays, such as arange, linspace, etc.
  - reading data from files

Numpy – Creating vectors

- From lists
  - numpy.array

```
# as vectors from lists
>>> a = numpy.array([1,3,5,7,9])
>>> b = numpy.array([3,5,6,7,9])
>>> c = a + b
>>> print c
[4, 8, 11, 14, 18]

>>> type(c)
(<type 'numpy.ndarray'>)

>>> c.shape
(5,)
```

Numpy – Creating matrices

```
>>> l = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list
>>> a = numpy.array(l)  # convert a list to an array
>>>print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> a.shape
(3, 3)
>>> print(a.dtype)  # get type of an array
int64

# or directly as matrix
>>> M = array([[1, 2], [3, 4]])
>>> M.shape
(2,2)
>>> M.dtype
dtype('int64')
```

Numpy – Creating matrices

```
>>> l = [[1, 2, 3], [3, 6, 9], [2, 4, 6]]  # create a list
>>> a = numpy.array(l)  # convert a list to an array
>>>print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> a.shape
(3, 3)
>>> print(a.dtype)  # ge
int64

# or directly as matrix
>>> M = array([[1, 2], [
>>> M.shape
(2,2)
>>> M.dtype
dtype('int64')
```

```
#only one type
>>> M[0,0] = "hello"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for long() with base 10: 'hello'

>>> M = numpy.array([[1, 2], [3, 4]], dtype=complex)
>>> M
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

Numpy – Matrices use

```
>>> print(a)
[[1 2 3]
 [3 6 9]
 [2 4 6]]
>>> print(a[0])  # this is just like a list of lists
[1 2 3]
>>> print(a[1, 2])  # arrays can be given comma separated indices
9
>>> print(a[1, 1:3])  # and slices
[6 9]
>>> print(a[:,1])
[2 6 4]
>>> a[1, 2] = 7
>>> print(a)
[[1 2 3]
 [3 6 7]
 [2 4 6]]
>>> a[:, 0] = [0, 9, 8]
>>> print(a)
[[0 2 3]
 [9 6 7]
 [8 4 6]]
```

Numpy – Creating arrays

- Generation functions

```
>>> x = arange(0, 10, 1) # arguments: start, stop, step
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> numpy.linspace(0, 10, 25)
array([  0.        ,   0.41666667,   0.83333333,   1.25      ,
         1.66666667,   2.08333333,   2.5       ,   2.91666667,
         3.33333333,   3.75      ,   4.16666667,   4.58333333,
         5.        ,   5.41666667,   5.83333333,   6.25      ,
         6.66666667,   7.08333333,   7.5       ,   7.91666667,
         8.33333333,   8.75      ,   9.16666667,   9.58333333,  10.        ])
>>> numpy.logspace(0, 10, 10, base=numpy.e)
array([  1.00000000e+00,   3.03773178e+00,   9.22781435e+00,
         2.80316249e+01,   8.51525577e+01,   2.58670631e+02,
         7.85771994e+02,   2.38696456e+03,   7.25095809e+03,
         2.20264658e+04])
```

Numpy – Creating arrays

```
# a diagonal matrix
>>> numpy.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])

>>> b = numpy.zeros(5)
>>> print(b)
[ 0.  0.  0.  0.  0.]
>>> b.dtype
dtype('float64')
>>> n = 1000
>>> my_int_array = numpy.zeros(n, dtype=numpy.int)
>>> my_int_array.dtype
dtype('int32')

>>> c = numpy.ones((3,3))
>>> c
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Numpy – array creation and use

```
>>> d = numpy.arange(5)  # just like range()
>>> print(d)
[0 1 2 3 4]

>>> d[1] = 9.7
>>> print(d)  # arrays keep their type even if elements changed
[0 9 2 3 4]

>>> print(d*0.4)  # operations create a new array, with new type
[ 0.   3.6  0.8  1.2  1.6]

>>> d = numpy.arange(5, dtype=numpy.float)
>>> print(d)
[ 0.  1.  2.  3.  4.]

>>> numpy.arange(3, 7, 0.5)  # arbitrary start, stop and step
array([ 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. , 6.5])
```

Numpy – array creation and use

```
>>> x, y = numpy.mgrid[0:5, 0:5] # similar to meshgrid in MATLAB
>>> x
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])
# random data
>>> numpy.random.rand(5,5)
array([[ 0.51531133,  0.74085206,  0.99570623,  0.97064334,  0.5819413 ],
       [ 0.2105685 ,  0.86289893,  0.13404438,  0.77967281,  0.78480563],
       [ 0.62687607,  0.51112285,  0.18374991,  0.2582663 ,  0.58475672],
       [ 0.72768256,  0.08885194,  0.69519174,  0.16049876,  0.34557215],
       [ 0.93724333,  0.17407127,  0.1237831 ,  0.96840203,  0.52790012]])
```

Numpy – Creating arrays

- File I/O

```
>>> os.system('head DeBilt.txt')
"Stn", "Datum", "Tg", "qTg", "Tn", "qTn", "Tx", "qTx"
001, 19010101,   -49, 00,   -68, 00,   -22, 40
001, 19010102,   -21, 00,   -36, 30,   -13, 30
001, 19010103,   -28, 00,   -79, 30,    -5, 20
001, 19010104,   -64, 00,   -91, 20,   -10, 00
001, 19010105,   -59, 00,   -84, 30,   -18, 00
001, 19010106,   -99, 00,  -115, 30,   -78, 30
001, 19010107,   -91, 00,  -122, 00,   -66, 00
001, 19010108,   -49, 00,   -94, 00,    -6, 00
001, 19010109,    11, 00,   -27, 40,    42, 00
0

>>> data = numpy.genfromtxt('DeBilt.txt', delimiter=',', skip_header=1)
>>> data.shape
(25568, 8)
```

Numpy – Creating arrays

- File I/O

```
>>> os.system('head DeBilt.txt')
"Stn", "Datum", "Tg", "qTg", "Tn", "qTn", "Tx", "qTx"
001, 19010101,   -49, 00,   -68, 00,   -22, 40
0
0
0
0
0
0
0
0
0

>
>
(25568, 8)
```

```
>>> numpy.savetxt('datasaved.txt', data)
>>> os.system('head datasaved.txt')
1.000000000000000000e+00 1.901010100000000000e+07 -4.900000000000000000e+01
0.000000000000000000e+00 -6.800000000000000000e+01 0.000000000000000000e+00 -
2.200000000000000000e+01 4.000000000000000000e+01
1.000000000000000000e+00 1.901010200000000000e+07 -2.100000000000000000e+01
0.000000000000000000e+00 -3.600000000000000000e+01 3.000000000000000000e+01 -
1.300000000000000000e+01 3.000000000000000000e+01
1.000000000000000000e+00 1.901010300000000000e+07 -2.800000000000000000e+01
0.000000000000000000e+00 -7.900000000000000000e+01 3.000000000000000000e+01 -
5.000000000000000000e+00 2.000000000000000000e+01
```

Numpy – Creating arrays

```
>>> M = numpy.random.rand(3,3)
>>> M
array([[ 0.84188778,  0.70928643,  0.87321035],
       [ 0.81885553,  0.92208501,  0.873464  ],
       [ 0.27111984,  0.82213106,  0.55987325]])
>>>
>>> numpy.save('saved-matrix.npy', M)
>>> numpy.load('saved-matrix.npy')
array([[ 0.84188778,  0.70928643,  0.87321035],
       [ 0.81885553,  0.92208501,  0.873464  ],
       [ 0.27111984,  0.82213106,  0.55987325]])
>>>
>>> os.system('head saved-matrix.npy')
NUMPYF{'descr': '<f8', 'fortran_order': False, 'shape': (3, 3), }
Ï<
£¾ðê?sy²æ?$÷ÒVñë?Ù4ê?%dn¸í?Ã[Äjóë?Ä,ZÑ?Ç
ÎåNê?ó7L{êá?0
>>>
```

Numpy - ndarray

- NumPy's main object is the homogeneous multidimensional array called `ndarray`.
  - This is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Typical examples of multidimensional arrays include vectors, matrices, images and spreadsheets.
  - Dimensions usually called axes, number of axes is the rank

[7, 5, -1]	An array of rank 1 i.e. It has 1 axis of length 3
[[1.5, 0.2, -3.7] , [0.1, 1.7, 2.9]]	An array of rank 2 i.e. It has 2 axes, the first length 3, the second of length 3 (a matrix with 2 rows and 3 columns

Numpy – ndarray attributes

- **ndarray.ndim**
  - the number of axes (dimensions) of the array i.e. the rank.
- **ndarray.shape**
  - the dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the rank, or number of dimensions, ndim.
- **ndarray.size**
  - the total number of elements of the array, equal to the product of the elements of shape.
- **ndarray.dtype**
  - an object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. NumPy provides many, for example bool_, character, int_, int8, int16, int32, int64, float_, float8, float16, float32, float64, complex_, complex64, object_.
- **ndarray.itemsize**
  - the size in bytes of each element of the array. E.g. for elements of type float64, itemsize is 8 (=64/8), while complex32 has itemsize 4 (=32/8) (equivalent to ndarray.dtype.itemsize).
- **ndarray.data**
  - the buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

Numpy – array creation and use

Two ndarrays are mutable and may be views to the same memory:

```
>>> x = np.array([1,2,3,4])
>>> y = x
>>> x is y
True
>>> id(x), id(y)
(139814289111920, 139814289111920)
>>> x[0] = 9
>>> y
array([9, 2, 3, 4])

>>> x[0] = 1
>>> z = x[:]
>>> x is z
False
>>> id(x), id(z)
(139814289111920, 139814289112080)
>>> x[0] = 8
>>> z
array([8, 2, 3, 4])
```

```
>>> x = np.array([1,2,3,4])
>>> y = x.copy()
>>> x is y
False
>>> id(x), id(y)
(139814289111920, 139814289111840)
>>> x[0] = 9
>>> x
array([9, 2, 3, 4])
>>> y
array([1, 2, 3, 4])
```

Numpy – Creating arrays

```
>>> a = numpy.arange(4.0)
>>> b = a * 23.4
>>> c = b/(a+1)
>>> c += 10
>>> print c
[ 10.    21.7   25.6   27.55]

>>> arr = numpy.arange(100, 200)
>>> select = [5, 25, 50, 75, -5]
>>> print(arr[select])  # can use integer lists as indices
[105, 125, 150, 175, 195]

>>> arr = numpy.arange(10, 20 )
>>> div_by_3 = arr%3 == 0  # comparison produces boolean array
>>> print(div_by_3)
[ False False  True False False  True False False  True False]
>>> print(arr[div_by_3])  # can use boolean lists as indices
[12 15 18]

>>> arr = numpy.arange(10, 20) . reshape((2,5))
[[10 11 12 13 14]
 [15 16 17 18 19]]
```

Numpy – array methods

```
>>> arr.sum()
145
>>> arr.mean()
14.5
>>> arr.std()
2.8722813232690143
>>> arr.max()
19
>>> arr.min()
10
>>> div_by_3.all()
False
>>> div_by_3.any()
True
>>> div_by_3.sum()
3
>>> div_by_3.nonzero()
(array([2, 5, 8]),)
```

Numpy – array methods - sorting

```
>>> arr = numpy.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> arr.sort()  # acts on array itself
>>> print(arr)
[ 1.2  1.8  2.3  4.5  5.5  6.7]

>>> x = numpy.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> numpy.sort(x)
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])

>>> print(x)
[ 4.5  2.3  6.7  1.2  1.8  5.5]

>>> s = x.argsort()
>>> s
array([3, 4, 1, 0, 5, 2])
>>> x[s]
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])
>>> y[s]
array([ 6.2,  7.8,  2.3,  1.5,  8.5,  4.7])
```

Numpy – array functions

- Most array methods have equivalent functions

```
>>> arr.sum()
45
>>> numpy.sum(arr)
45
```

- Ufuncs provide many element-by-element math, trig., etc. operations
  - e.g., add(x1, x2), absolute(x), log10(x), sin(x), logical_and(x1, x2)

- See http://numpy.scipy.org

Numpy – array operations

```
>>> a = array([[1.0, 2.0], [4.0, 3.0]])
>>> print a
[[ 1. 2.]
 [ 3. 4.]]

>>> a.transpose()
array([[ 1., 3.],
       [ 2., 4.]])

>>> inv(a)
array([[-2. , 1. ],
       [ 1.5, -0.5]])

>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"

>>> u
array([[ 1., 0.],
       [ 0., 1.]])

>>> j = array([[0.0, -1.0], [1.0, 0.0]])

>>> dot (j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
```

Numpy – statistics

In addition to the mean, var, and std functions, NumPy supplies several other methods for returning statistical features of arrays. The median can be found:

```
>>> a = np.array([1, 4, 3, 8, 9, 2, 3], float)
>>> np.median(a)
3.0
```

The correlation coefficient for multiple variables observed at multiple instances can be found for arrays of the form [[x1, x2, …], [y1, y2, …], [z1, z2, …], …] where x, y, z are different observables and the numbers indicate the observation times:

```
>>> a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
>>> c = np.corrcoef(a)
>>> c
array([[ 1.        ,  0.72870505],
       [ 0.72870505,  1.        ]])
```

Here the return array c[i,j] gives the correlation coefficient for the ith and jth observables. Similarly, the covariance for data can be found::

```
>>> np.cov(a)
array([[ 0.91666667,  2.08333333],
       [ 2.08333333,  8.91666667]])
```

Using arrays wisely

- Array operations are implemented in C or Fortran
- Optimised algorithms - i.e. fast!
- Python loops (i.e. for i in a:…) are much slower
- Prefer array operations over loops, especially when speed important
- Also produces shorter code, often more readable

Numpy – arrays, matrices

For **two dimensional** arrays NumPy defined a special matrix class in module matrix. Objects are created either with matrix() or mat() or converted from an array with method asmatrix().

```
>>> import numpy
>>> m = numpy.mat([[1,2],[3,4]])
or
>>> a = numpy.array([[1,2],[3,4]])
>>> m = numpy.mat(a)
or
>>> a = numpy.array([[1,2],[3,4]])
>>> m = numpy.asmatrix(a)
```

Note that the statement m = mat(a) creates a copy of array 'a'.
Changing values in 'a' will not affect 'm'.
On the other hand, method m = asmatrix(a) returns a new reference to the same data.
Changing values in 'a' will affect matrix 'm'.

Numpy – matrices

Array and matrix operations may be quite different!

```
>>> a = array([[1,2],[3,4]])
>>> m = mat(a) # convert 2-d array to matrix
>>> m = matrix([[1, 2], [3, 4]])
>>> a[0]             # result is 1-dimensional
array([1, 2])
>>> m[0]             # result is 2-dimensional
matrix([[1, 2]])
>>> a*a              # element-by-element multiplication
array([[ 1, 4], [ 9, 16]])
>>> m*m              # (algebraic) matrix multiplication
matrix([[ 7, 10], [15, 22]])
>>> a**3             # element-wise power
array([[ 1, 8], [27, 64]])
>>> m**3             # matrix multiplication m*m*m
matrix([[ 37, 54], [ 81, 118]])
>>> m.T              # transpose of the matrix
matrix([[1, 3], [2, 4]])
>>> m.H              # conjugate transpose (differs from .T for complex matrices)
matrix([[1, 3], [2, 4]])
>>> m.I              # inverse matrix
matrix([[-2. , 1. ], [ 1.5, -0.5]])
```

Numpy – matrices

- Operator *, dot(), and multiply():
  - For array, '*' **means element-wise multiplication**, and the dot() function is used for matrix multiplication.
  - For matrix, '*'**means matrix multiplication**, and the multiply() function is used for element-wise multiplication.
- Handling of vectors (rank-1 arrays)
  - For array, the vector shapes 1xN, Nx1, and N are all different things. Operations like A[:,1] return a rank-1 array of shape N, not a rank-2 of shape Nx1. Transpose on a rank-1 array does nothing.
  - For matrix, rank-1 arrays are always upgraded to 1xN or Nx1 matrices (row or column vectors). A[:,1] returns a rank-2 matrix of shape Nx1.
- Handling of higher-rank arrays (rank > 2)
  - array objects can have rank > 2.
  - matrix objects always have exactly rank 2.
- Convenience attributes
  - array has a .T attribute, which returns the transpose of the data.
  - matrix also has .H, .I, and .A attributes, which return the conjugate transpose, inverse, and asarray() of the matrix, respectively.
- Convenience constructor
  - The array constructor takes (nested) Python sequences as initializers. As in array([[1,2,3],[4,5,6]]).
  - The matrix constructor additionally takes a convenient string initializer. As in matrix("[1 2 3; 4 5 6]")

Numpy – array mathematics

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a – b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])

>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

Numpy – array mathematics

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a – b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])

>>> a = np.array([[1, 2], [3, 4],
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)

>>> a * a
array([[  1.,   4.],
       [  9.,  16.],
       [ 25.,  36.]])
>>> b * b
array([ 1.,  9.])
>>> a * b
array([[ -1.,   6.],
       [ -3.,  12.],
       [ -5.,  18.]])
>>>
```

Numpy – array mathematics

```
>>> A = np.array([[n+m*10 for n in range(5)] for m in range(5)])
>>> v1 = arange(0, 5)
>>> A
array([[ 0, 1, 2, 3, 4],
[10, 11, 12, 13, 14],
[20, 21, 22, 23, 24],
[30, 31, 32, 33, 34],
[40, 41, 42, 43, 44]])
>>> v1
array([0, 1, 2, 3, 4])
>>> np.dot(A,A)
array([[ 300,  310,  320,  330,  340],
       [1300, 1360, 1420, 1480, 1540],
       [2300, 2410, 2520, 2630, 2740],
       [3300, 3460, 3620, 3780, 3940],
       [4300, 4510, 4720, 4930, 5140]])
>>>
>>> np.dot(A,v1)
array([ 30, 130, 230, 330, 430])
>>> np.dot(v1,v1)
30
>>>
```

Numpy – array mathematics

```
>>> A = np.array([
>>> v1 = arange(0,
>>> A
array([[ 0,  1,  2,
 [10, 11, 12, 13, 1
 [20, 21, 22, 23, 2
 [30, 31, 32, 33, 3
 [40, 41, 42, 43, 4
>>> v1
array([0, 1, 2, 3,
>>> np.dot(A,A)
array([[ 300,   310
       [1300, 1360
       [2300, 2410
       [3300, 3460
       [4300, 4510
>>>
>>> np.dot(A,v1)
array([ 30, 130, 2
>>> np.dot(v1,v1)
30
>>>
```

```
Alternatively, we can cast the array objects to the type matrix. This changes
the behavior of the standard arithmetic operators +, -, * to use matrix
algebra.
>>> M = np.matrix(A)
>>> v = np.matrix(v1).T
>>> v
matrix([[0],
        [1],
        [2],
        [3],
        [4]])
>>> M*v
matrix([[ 30],
        [130],
        [230],
        [330],
        [430]])
>>> v.T * v   # inner product
matrix([[30]])
# standard matrix algebra applies
>>> v + M*v
matrix([[ 30],
        [131],
        [232],
        [333],
        [434]])
```

Plotting - matplotlib

- User friendly, but powerful, plotting capabilites for python
- http://matplotlib.sourceforge.net/

- Once installed (default at Observatory)

```
>>> import pylab
```

- Settings can be customised by editing ~/.matplotlib/matplotlibrc
  - default font, colours, layout, etc.
- Helpful website
  - many examples

Pyplot and pylab

Pylab is a module in matplotlib that gets installed alongside matplotlib; and matplotlib.pyplot is a module in matplotlib.

- **Pyplot** provides the state-machine interface to the underlying plotting library in matplotlib. This means that figures and axes are implicitly and automatically created to achieve the desired plot. Setting a title will then automatically set that title to the current axes object.
- **Pylab** combines the pyplot functionality (for plotting) with the numpy functionality (for mathematics and for working with arrays) in a single namespace, For example, one can call the sin and cos functions just like you could in MATLAB, as well as having all the features of pyplot.
- The pyplot interface is generally preferred for non-interactive plotting (i.e., scripting). The pylab interface is convenient for interactive calculations and plotting, as it minimizes typing. Note that this is what you get if you use the ipython shell with the --pylab option, which imports everything from pylab and makes plotting fully interactive.

Pyplot and pylab

```
$ python
Python 2.7.3 (default, Aug  9 2012, 17:23:57)
[GCC 4.7.1 20120720 (Red Hat 4.7.1-5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import matplotlib.pyplot as plt
>>> plt.plot([1,2,3,4])
[<matplotlib.lines.Line2D object at 0x1fe53d0>]
>>> plt.ylabel('some numbers')
<matplotlib.text.Texy object at 0x1d6ad90>
>>> plt.show()
```

```
$ ipython --pylab
Python 2.7.3 (default, Aug  9 2012, 17:23:57)
Type "copyright", "credits" or "license" for more information.

IPython 0.13.2 -- 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.

Welcome to pylab, a matplotlib-based Python environment [backen: GTKAgg].
For more information, type ‘help(pylab)’

In [1]: plot([1,2,3,4])
```

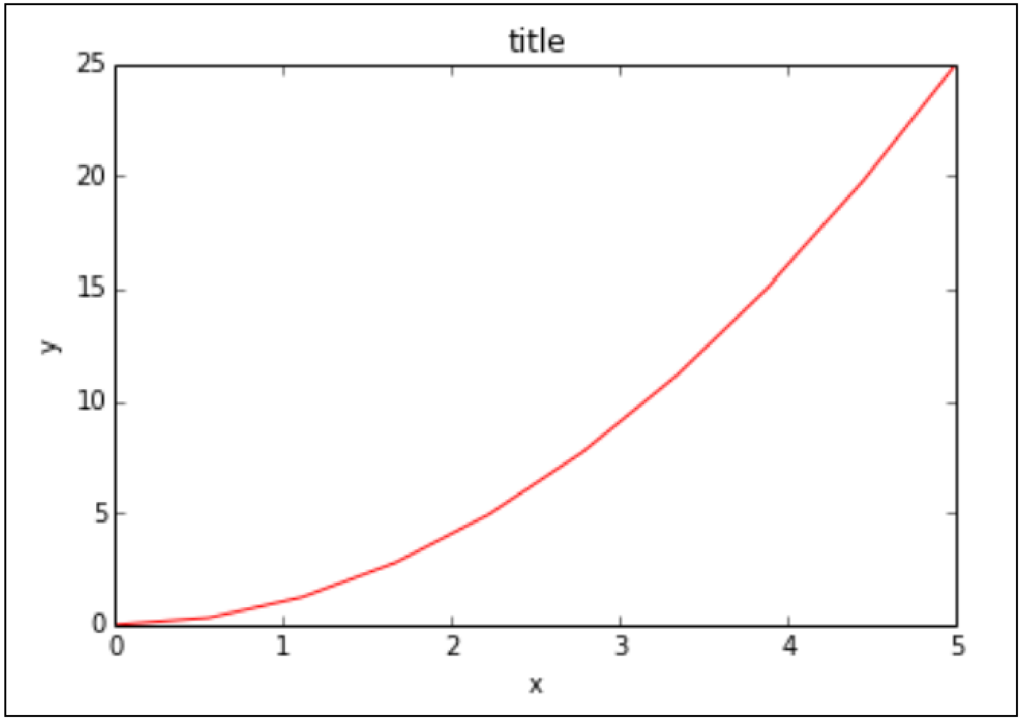
Matplotlib.pyplot example

Function	Description	Function	Description	Function	Description
acorr	plot the autocorrelation function	gca	return the current axes	gca	return the current axes
annotate	annotate something in the figure	gcf	return the current figure	gcf	return the current figure
arrow	add an arrow to the axes	gci	get the current image, or None	gci	get the current image, or None
axes	create a new axes	getp	get a graphics property	getp	get a graphics property
axhline	draw a horizontal line across axes	grid	set whether gridding is on	grid	set whether gridding is on
axvline	draw a vertical line across axes	hexbin	make a 2D hexagonal binning plot	hexbin	make a 2D hexagonal binning plot
axhspan	draw a horizontal bar across axes	hist	make a histogram	hist	make a histogram
axvspan	draw a vertical bar across axes	hold	set the axes hold state	hold	set the axes hold state
axis	set or return the current axis limits	ioff	turn interaction mode off	ioff	turn interaction mode off
barbs	a (wind) barb plot	ion	turn interaction mode on	ion	turn interaction mode on
bar	make a bar chart	isinteractive	return True if interaction mode is on	isinteractive	return True if interaction mode is on
barh	a horizontal bar chart	imread	load image file into array	imread	load image file into array
broken_barh	a set of horizontal bars with gaps	imsave	save array as an image file	imsave	save array as an image file
box	set the axes frame on/off state	imshow	plot image data	imshow	plot image data
boxplot	make a box and whisker plot	ishold	return the hold state of the current ax	ishold	return the hold state of the current axes
cla	clear current axes	legend	make an axes legend	legend	make an axes legend
clabel	label a contour plot	locator_params	adjust parameters used in locating ax ticks	locator_params	adjust parameters used in locating axis ticks
clf	clear a figure window				
clim	adjust the color limits of the current image	loglog	a log log plot	loglog	a log log plot
close	close a figure window	matshow	display a matrix in a new figure prese aspect	matshow	display a matrix in a new figure preserving aspect
colorbar	add a colorbar to the current figure				
cohere	make a plot of coherence	margins	set margins used in autoscaling	margins	set margins used in autoscaling
contour	make a contour plot	pcolor	make a pseudocolor plot	pcolor	make a pseudocolor plot
contourf	make a filled contour plot	pcolormesh	make a pseudocolor plot using a quadrilateral mesh	pcolormesh	make a pseudocolor plot using a quadrilateral mesh
csd	make a plot of cross spectral density				
delaxes	delete an axes from the current figure	pie	make a pie chart	pie	make a pie chart
draw	Force a redraw of the current figure	plot	make a line plot	plot	make a line plot
errorbar	make an errorbar graph	plot_date	plot dates	plot_date	plot dates
figlegend	make legend on the figure rather than the axes	plotfile	plot column data from an ASCII tab/space/comma delimited file	plotfile	plot column data from an ASCII tab/space/comma delimited file
figimage	make a figure image	pie	pie charts	pie	pie charts
figtext	add text in figure coords	polar	make a polar plot on a PolarAxes	polar	make a polar plot on a PolarAxes
figure	create or change active figure	psd	make a plot of power spectral densit	psd	make a plot of power spectral density
fill	make filled polygons	quiver	make a direction field (arrows) plot	quiver	make a direction field (arrows) plot
fill_between	make filled polygons between two curves	rc	control the default params	rc	control the default params

Matplotlib.pyplot example

The easiest way to get started with plotting using matplotlib is often to use the MATLAB-like API provided by matplotlib.

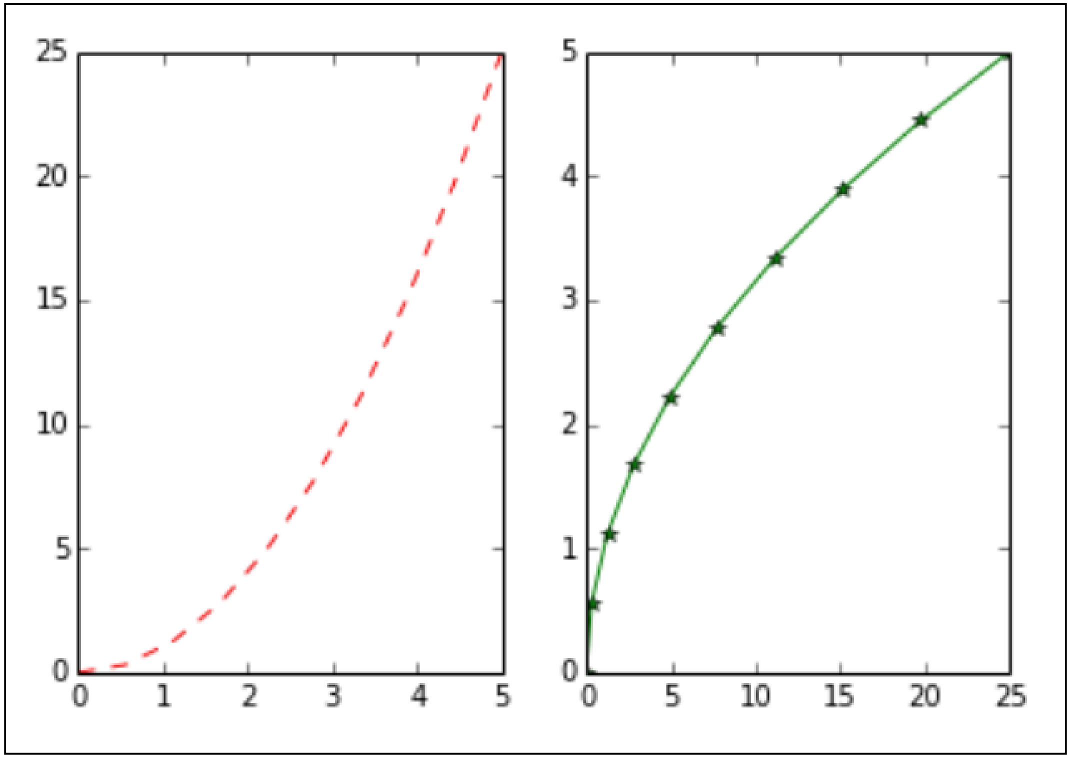
```
>>> import matplotlib.pylab as plt
>>> x = plt.linspace(0, 5, 10)
>>> y = x ** 2

>>> figure()
>>> plot(x, y, 'r')
>>> xlabel('x')
>>> ylabel('y')
>>> title('title')
>>> show()
```



Matplotlib.pyplot example

Most of the plotting related functions in MATLAB are covered by the pylab module. For example, subplot and color/symbol selection

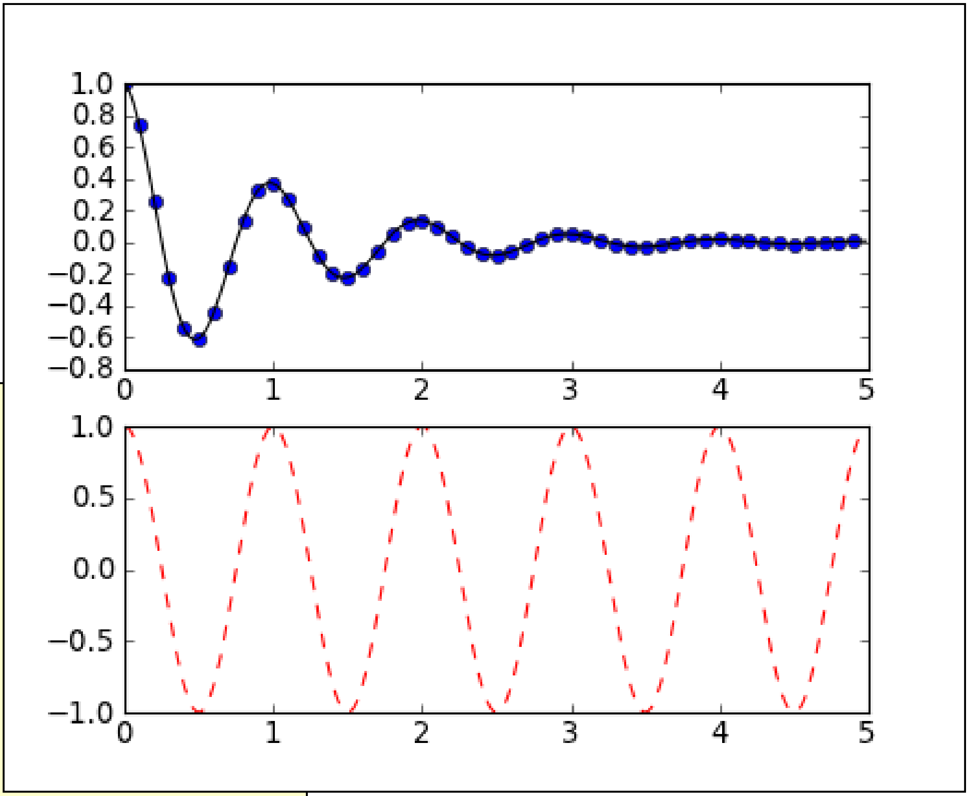
```
>>> plt.subplot(1,2,1)
>>> plt.plot(x, y, 'r--')
>>> plt.subplot(1,2,2)
>>> plt.plot(y, x, 'g*-');
```



Matplotlib.pyplot example

Working with multiple figures and axes. The subplot() command specifies numrows, numcols, fignum, where fignum ranges from 1 to numrows*numcols.

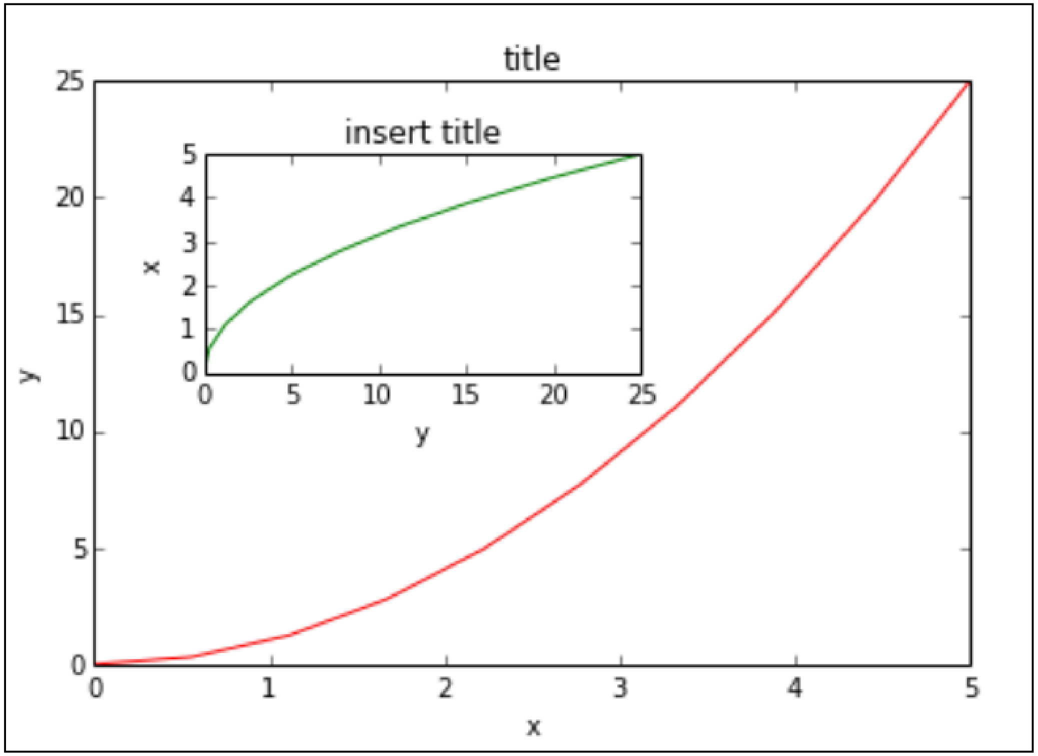
```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
   return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```



Matplotlib.pyplot example

Matplotlib can be used in object oriented approach which is particularly useful when we deal with subplots

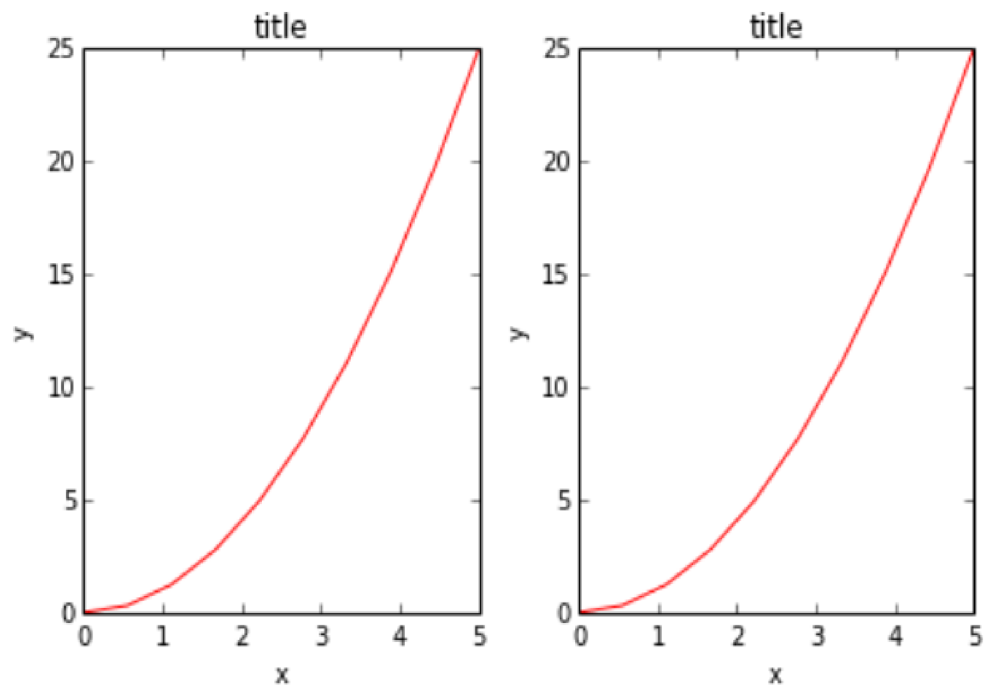
```
fig = plt.figure()
axes1 = fig.add_axes([0.1, 0.1,
0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5,
0.4, 0.3]) # inset axes
# main figure
axes1.plot(x, y, 'r')
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('title')
# insert
axes2.plot(y, x, 'g')
axes2.set_xlabel('y')
axes2.set_ylabel('x')
axes2.set_title('insert title');
```



Matplotlib.pyplot example

With multiple plots on one screen sometimes the labels are getting in the way. Solve this with tight_layout

```
fig, axes = plt.subplots(nrows=1,
   ncols=2)
for ax in axes:
ax.plot(x, y, ’r’)
ax.set_xlabel(’x’)
ax.set_ylabel(’y’)
ax.set_title(’title’)
fig.tight_layout()
```



Matplotlib.pyplot example

Labels and legends and titles

These figure decrations are essential for presenting your data to others (in papers and in oral presentations).

```
>>> ax.set_title("title");
>>> ax.set_xlabel("x")
>>> ax.set_ylabel("y");
>>> ax.legend(["curve1", "curve2", "curve3"]);

>>> ax.plot(x, x**2, label="curve1")
>>> ax.plot(x, x**3, label="curve2")
>>> ax.legend()

>>> ax.legend(loc=0) # let matplotlib decide
>>> ax.legend(loc=1) # upper right corner
>>> ax.legend(loc=2) # upper left corner
>>> ax.legend(loc=3) # lower left corner
>>> ax.legend(loc=4) # lower right corner
```

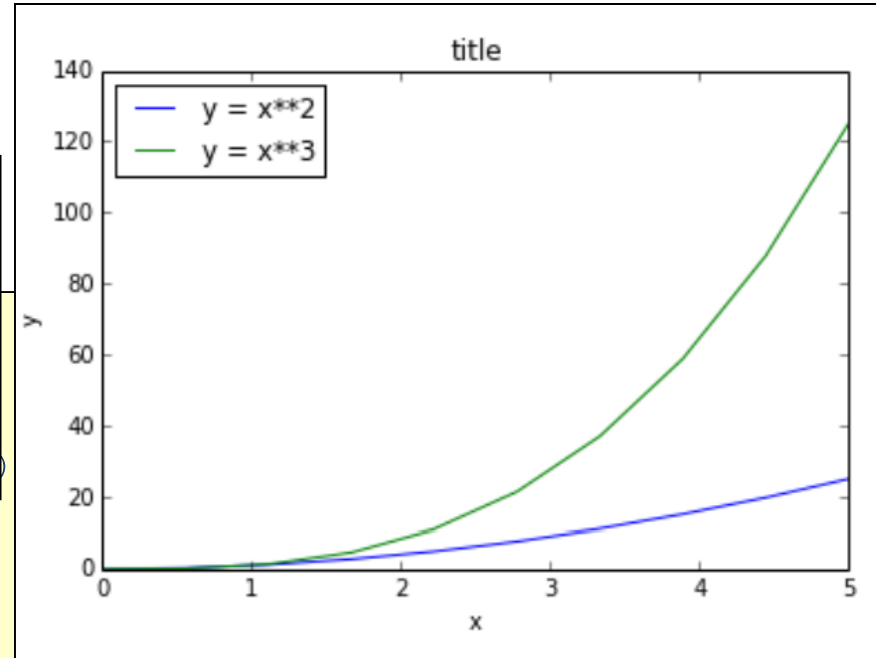
Matplotlib.pyplot example

Labels and legends and titles

These figure decrations are essential

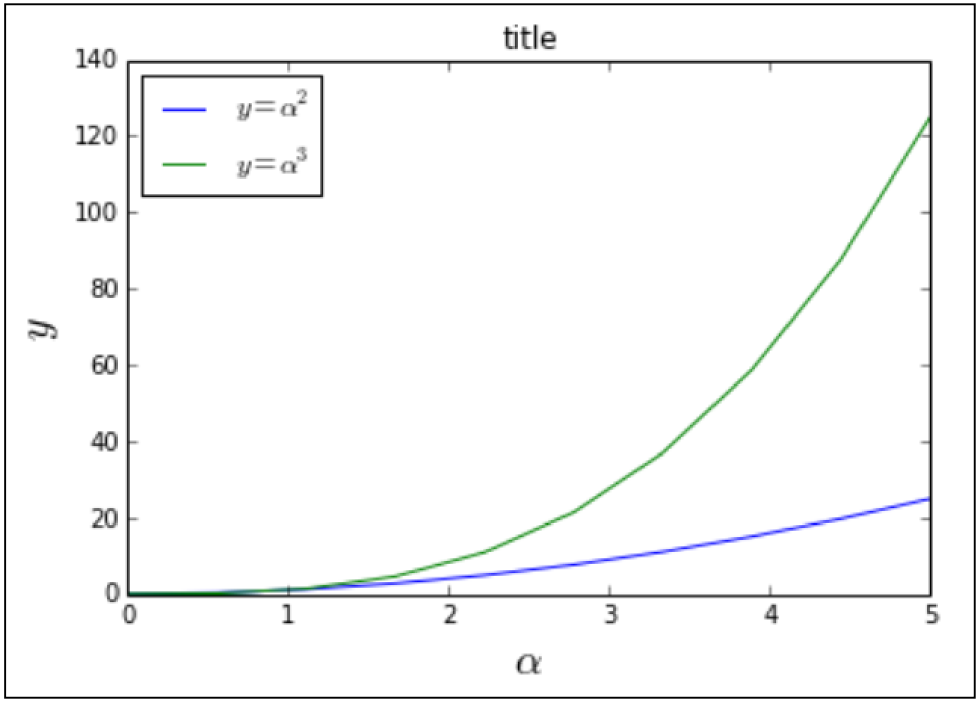
```
>>> fig, ax = plt.subplots()
>>> ax.plot(x, x**2, label="y = x**2")
>>> ax.plot(x, x**3, label="y = x**3")
>>> ax.legend(loc=2); # upper left corner
>>> ax.set_xlabel('x')
>>> ax.set_ylabel('y')
>>> ax.set_title('title');
```

```
>>> ax.plot(x, x**2, label="curve1")
>>> ax.plot(x, x**3, label="curve2")
>>> ax.legend()

>>> ax.legend(loc=0) # let matplotlib decide
>>> ax.legend(loc=1) # upper right corner
>>> ax.legend(loc=2) # upper left corner
>>> ax.legend(loc=3) # lower left corner
>>> ax.legend(loc=4) # lower right corner
```



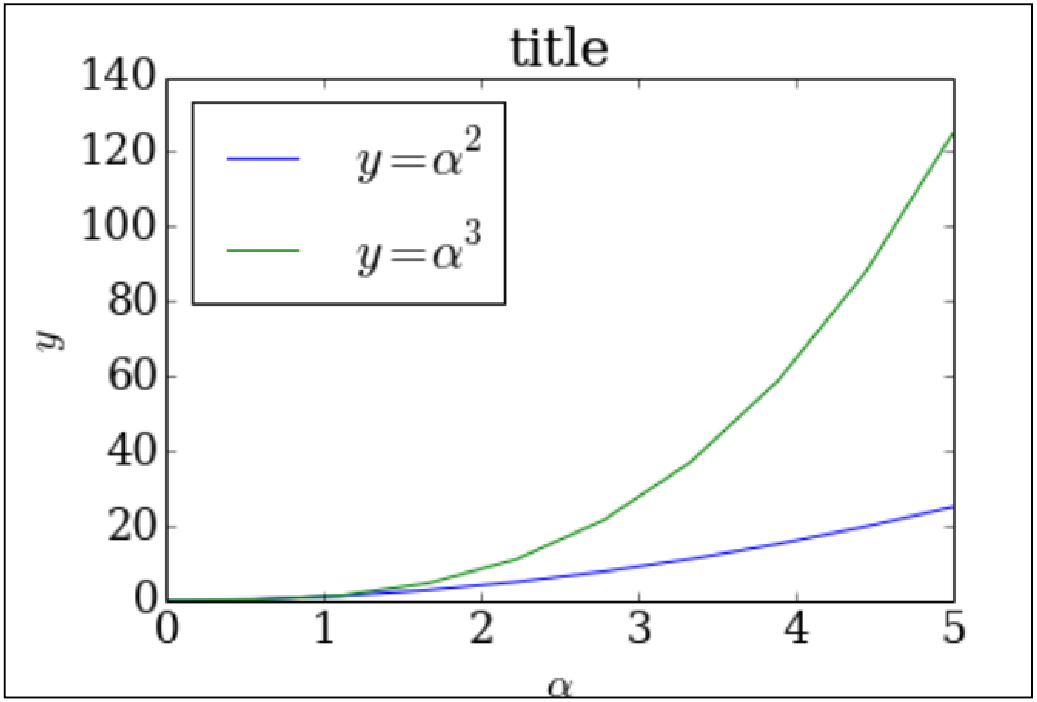
Matplotlib.pyplot example

Changing the font size and family and using LaTeX formatted tekst.




```
>>> fig, ax = plt.subplots()
>>> ax.plot(x, x**2, label=r"$y = \alpha^2$")
>>> ax.plot(x, x**3, label=r"$y = \alpha^3$")
>>> ax.legend(loc=2) # upper left corner
>>> ax.set_xlabel(r'$\alpha$', fontsize=18)
>>> ax.set_ylabel(r'$y$', fontsize=18)
>>> ax.set_title('title');
```

Matplotlib.pyplot example



Changing the font size and family and using LaTeX formatted tekst.

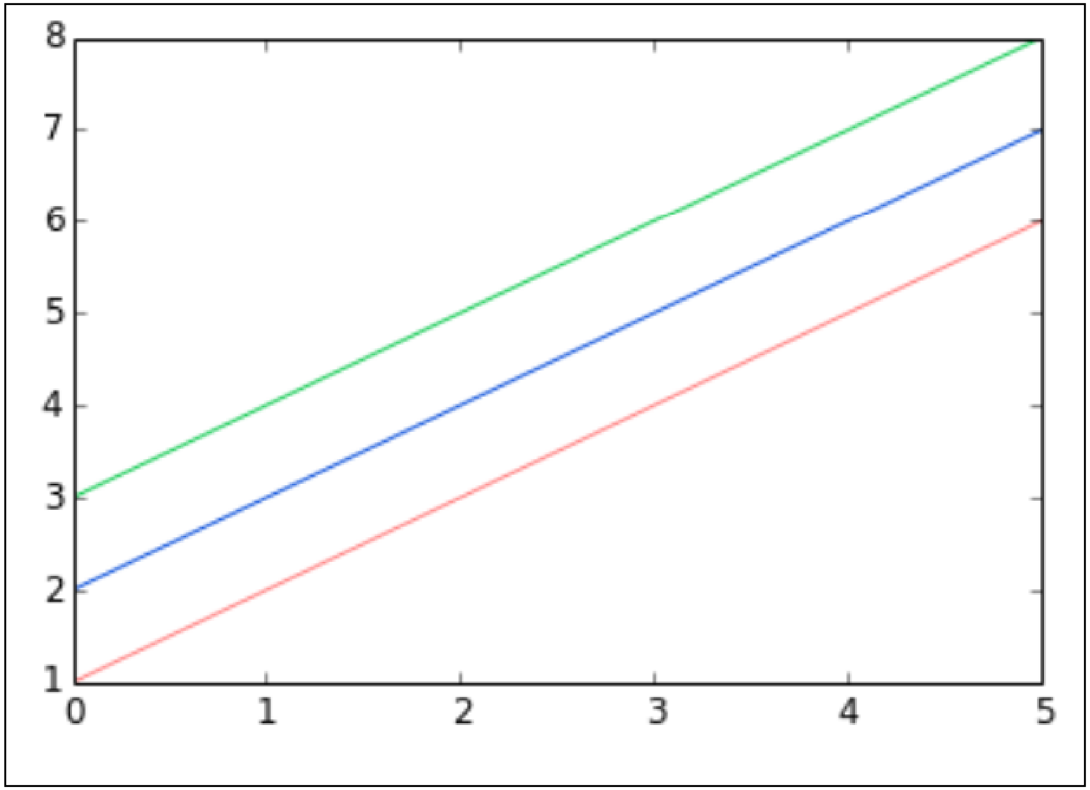
```
matplotlib.rcParams.update({'font.size': 18, 'font.family': 'serif'})
```

```
>>> fig, ax = plt.subplots()
>>> ax.plot(x, x**2, label=r"$y = \alpha^2$")
>>> ax.plot(x, x**3, label=r"$y = \alpha^3$")
>>> ax.legend(loc=2) # upper left corner
>>> ax.set_xlabel(r'$\alpha$', fontsize=18)
>>> ax.set_ylabel(r'$y$', fontsize=18)
>>> ax.set_title('title');
```

Matplotlib.pyplot example

You have full control over colors, linewidths and linetypes.

For colors one can use simple syntax like ‘b’ for blue, ‘g’ for green, etc. or RGB hex codes with transparency alpha code:




```
fig, ax = plt.subplots()
ax.plot(x, x+1, color="red", alpha=0.5) # half-transparant red
ax.plot(x, x+2, color="#1155dd") # RGB hex code for a bluish color
ax.plot(x, x+3, color="#15cc55") # RGB hex code for a greenish color
```

Matplotlib.pyplot example

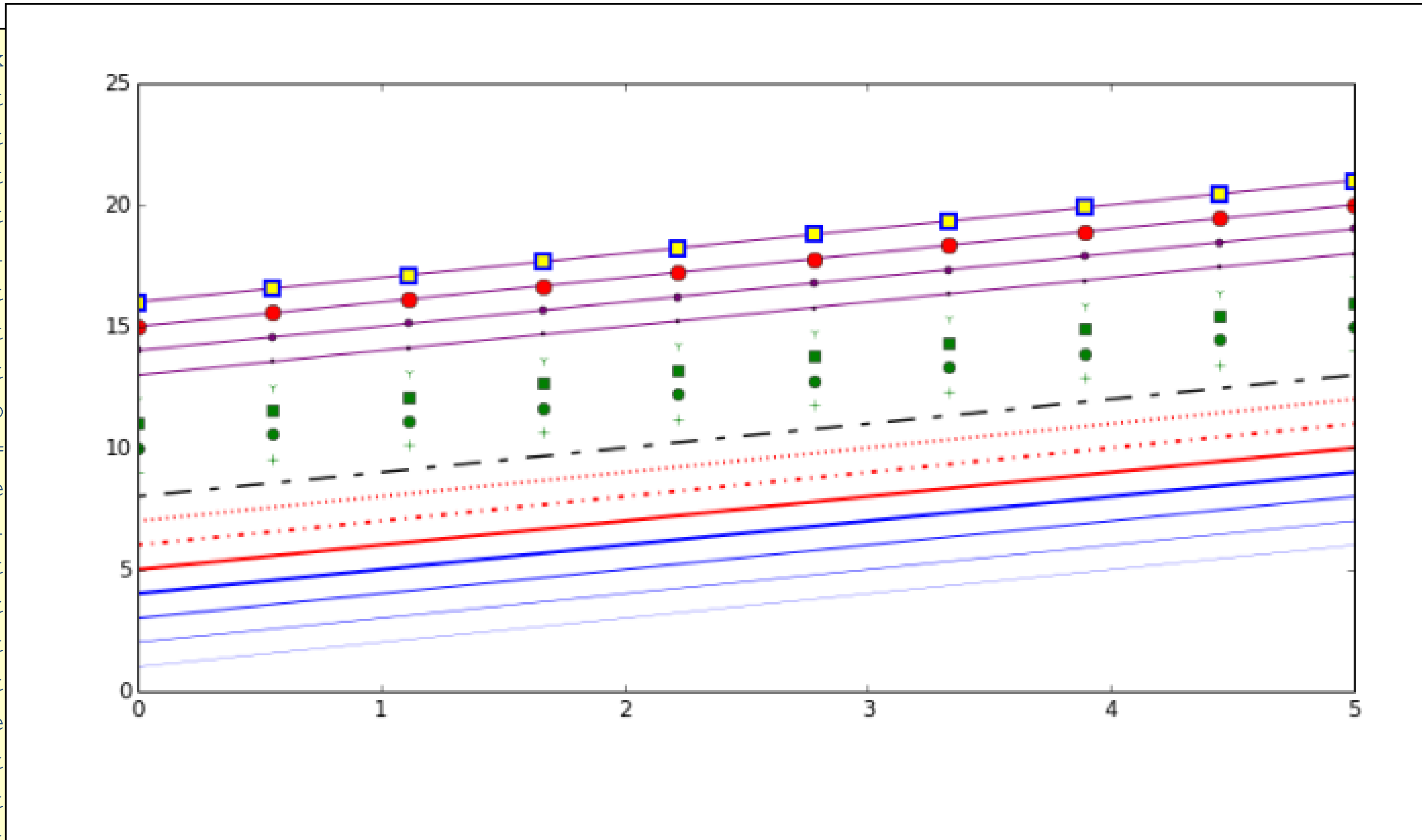
Line and marker styles are coded:

```
fig, ax = plt.subplots(figsize=(12,6))
ax.plot(x, x+1, color="blue", linewidth=0.25)
ax.plot(x, x+2, color="blue", linewidth=0.50)
ax.plot(x, x+3, color="blue", linewidth=1.00)
ax.plot(x, x+4, color="blue", linewidth=2.00)
# possible linestype options ‘-‘, ‘{’, ‘-.’, ‘:’, ‘steps’
ax.plot(x, x+5, color="red", lw=2, linestyle=’-’)
ax.plot(x, x+6, color="red", lw=2, ls=’-.’)
ax.plot(x, x+7, color="red", lw=2, ls=’:’)
# custom dash
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10]) # format: line length, space length, ...
# possible marker symbols: marker = ’+’, ’o’, ’*’, ’s’, ’,’, ’.’, ’1’, ’2’, ’3’, ’4’, ...
ax.plot(x, x+ 9, color="green", lw=2, ls=’*’, marker=’+’)
ax.plot(x, x+10, color="green", lw=2, ls=’*’, marker=’o’)
ax.plot(x, x+11, color="green", lw=2, ls=’*’, marker=’s’)
ax.plot(x, x+12, color="green", lw=2, ls=’*’, marker=’1’)
# marker size and color
ax.plot(x, x+13, color="purple", lw=1, ls=’-’, marker=’o’, markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls=’-’, marker=’o’, markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls=’-’, marker=’o’, markersize=8, markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls=’-’, marker=’s’, markersize=8,
markerfacecolor="yellow", markeredgewidth=2, markeredgecolor="blue");
```

Matplotlib.pyplot example

Line and marker styles are coded:

```
fig, ax
ax.plot
ax.plot
ax.plot
ax.plot
# possi
ax.plot
ax.plot
ax.plot
# custo
line, =
line.se
# possi
ax.plot
ax.plot
ax.plot
ax.plot
# marke
ax.plot
ax.plot
ax.plot
ax.plot(x, x+16, color="purple", lw=1, ls=’-’, marker=’s’, markersize=8,
markerfacecolor="yellow", markeredgewidth=2, markeredgecolor="blue");
```



Matplotlib.pyplot example

Sometimes it is useful to have dual x or y axes in a figure; for example, when plotting curves with different

units together. Matplotlib supports this with the twinx and twiny functions:

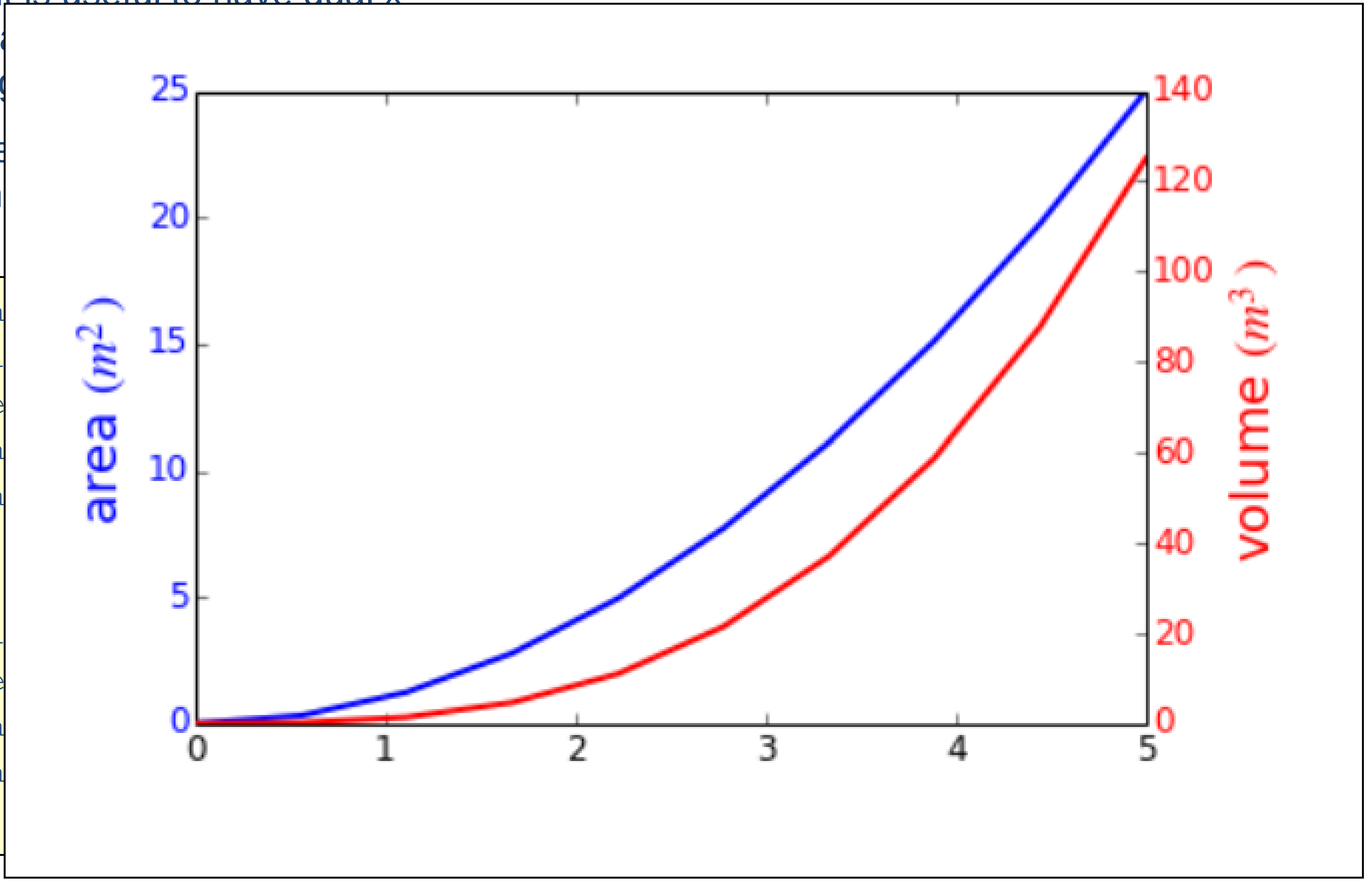
```
>>> fig, ax1 = plt.subplots()
>>> ax1.plot(x, x**2, lw=2, color="blue")
>>> ax1.set_ylabel(r"area $(m^2)$", fontsize=18, color="blue")
>>> for label in ax1.get_yticklabels():
...     label.set_color("blue")
...
>>> ax2 = ax1.twinx()
>>> ax2.plot(x, x**3, lw=2, color="red")
>>> ax2.set_ylabel(r"volume $(m^3)$", fontsize=18, color="red")
>>> for label in ax2.get_yticklabels():
...     label.set_color("red")
```

Matplotlib.pyplot example

Sometimes it is useful to have dual x
or y axes in a
when plotting

units togethe
with the twin

```
>>> fig, a
>>> ax1.pl
>>> ax1.se
>>> for la
...      la
...
>>> ax2 =
>>> ax2.pl
>>> ax2.se
>>> for la
...      la
```

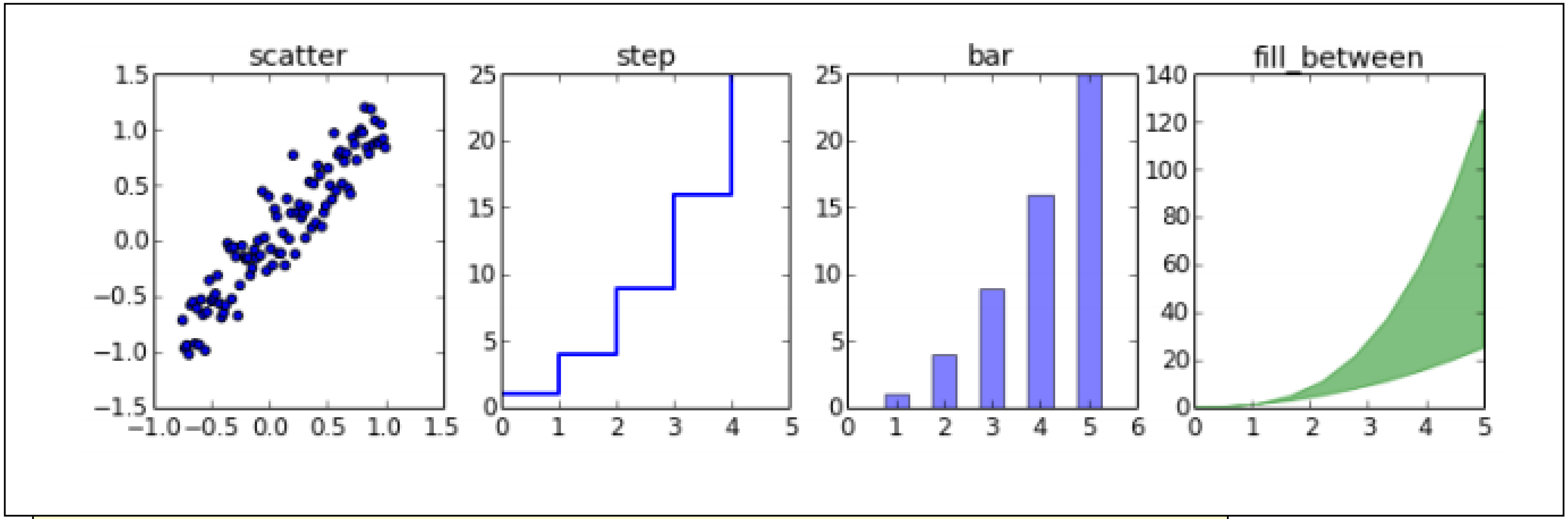


Matplotlib.pyplot example

Other 2D plt styles

```
>>> n = array([0,1,2,3,4,5])
>>> fig, axes = plt.subplots(1, 4, figsize=(12,3))
>>> axes[0].scatter(xx, xx + 0.25*randn(len(xx)))
>>> axes[0].set_title("scatter")

>>> axes[1].step(n, n**2, lw=2)
>>> axes[1].set_title("step")

>>> axes[2].bar(n, n**2, align="center", width=0.5, alpha=0.5)
>>> axes[2].set_title("bar")

>>> axes[3].fill_between(x, x**2, x**3, color="green", alpha=0.5)
>>> axes[3].set_title("fill_between")
```

Matplotlib.pyplot example



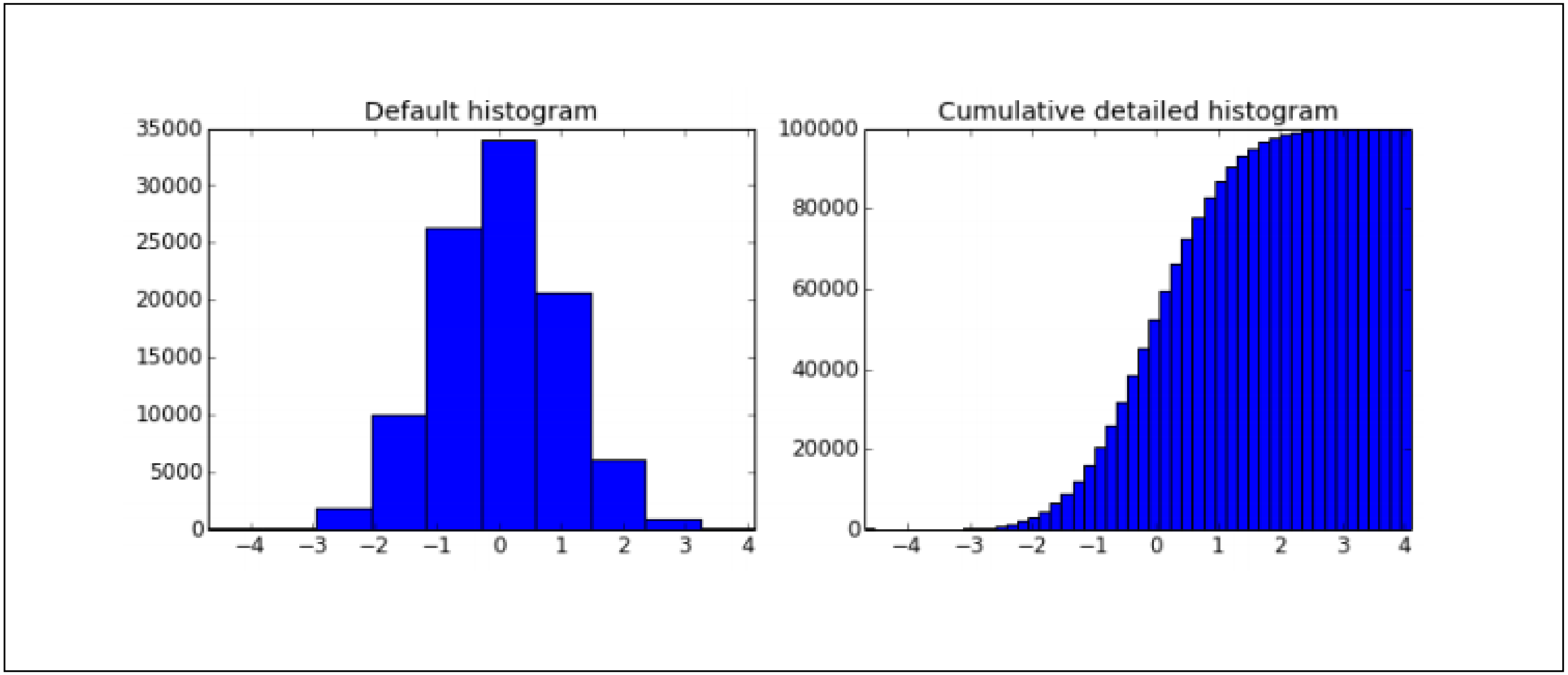

```
>>> axes[2].set_title("bar")

>>> axes[3].fill_between(x, x**2, x**3, color="green", alpha=0.5)
>>> axes[3].set_title("fill_between")
```

Matplotlib.pyplot example

Histograms are also a very useful visualisation tool

```
>>> n = np.random.randn(100000)
>>> fig, axes = plt.subplots(1, 2, figsize=(12,4))
>>> axes[0].hist(n)
>>> axes[0].set_title("Default histogram")
>>> axes[0].set_xlim((min(n), max(n)))
>>> axes[1].hist(n, cumulative=True, bins=50)
>>> axes[1].set_title("Cumulative detailed histogram")
>>> axes[1].set_xlim((min(n), max(n)));
```

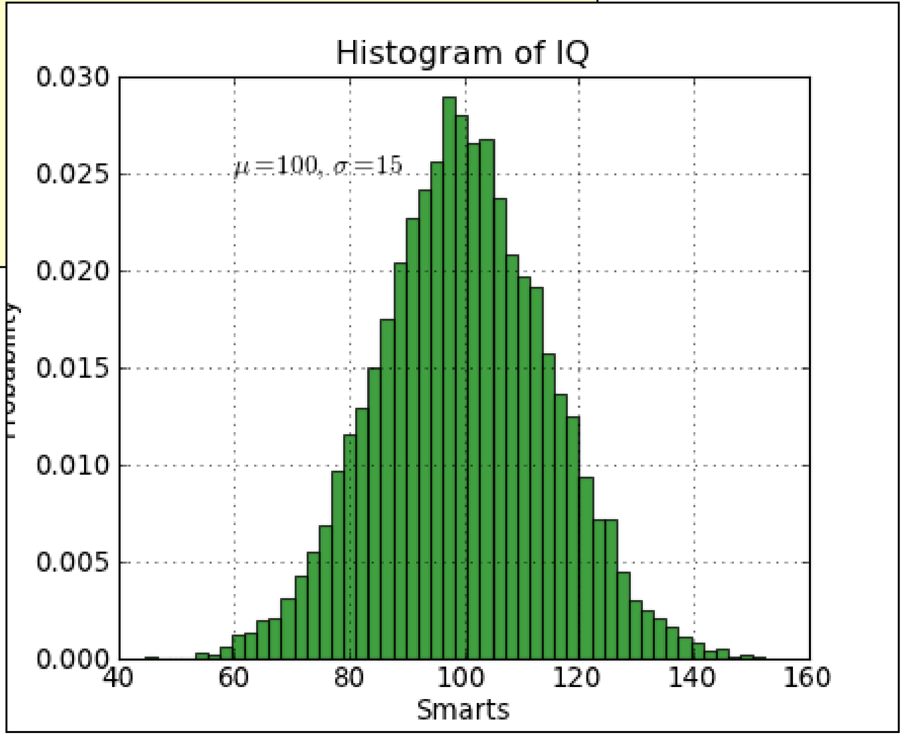
Matplotlib.pyplot example



Matplotlib.pyplot example

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

mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
```

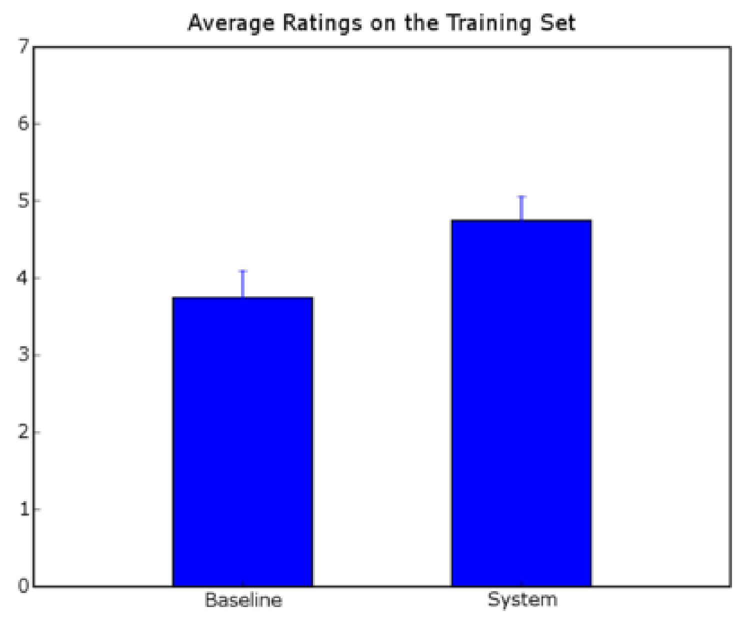
Working with text.
The text() command can be used to add text in an arbitrary location, and the xlabel(), ylabel() and title() are used to add text in the indicated locations.



Matplotlib.pyplot example

Bar chart.

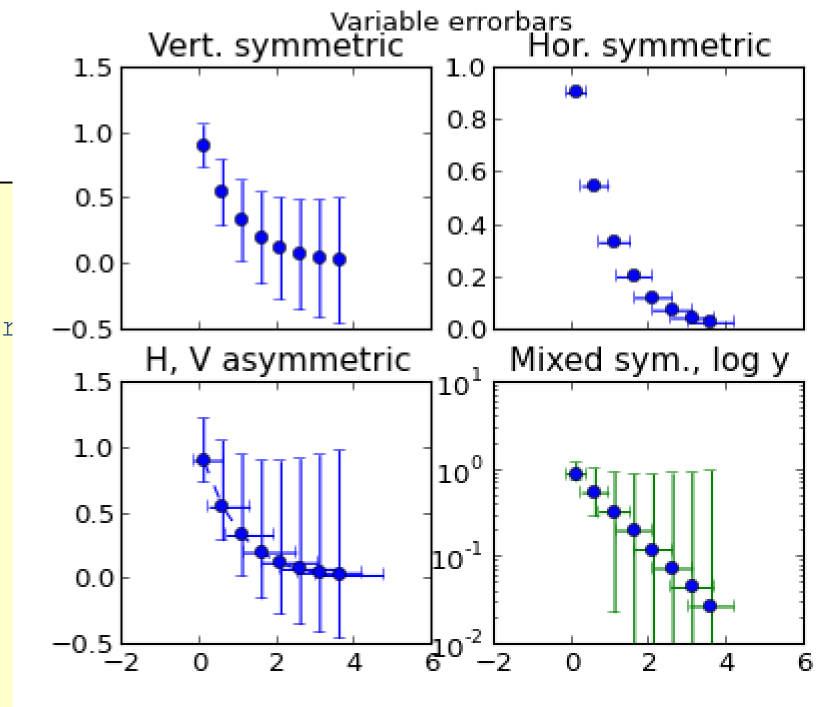
```
import numpy.numarray as na

from pylab import *

labels = ["Baseline", "System"]
data =   [3.75               , 4.75]
error =  [0.3497             , 0.3108]

xlocations = na.array(range(len(data)))+0.5
width = 0.5
bar(xlocations, data, yerr=error, width=width)
yticks(range(0, 8))
xticks(xlocations+ width/2, labels)
xlim(0, xlocations[-1]+width*2)
title("Average Ratings on the Training Set")

show()
```



Matplotlib.pyplot example

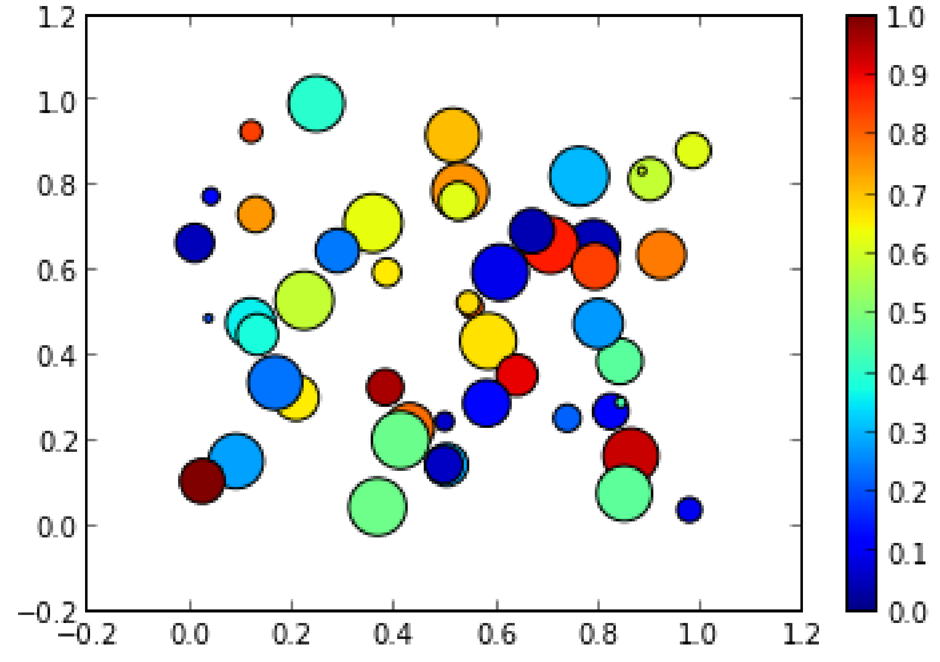
Error bars.

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

fig, axs = plt.subplots(nrows=2, ncols=2, sharex=Tr
ax = axs[0,0]
ax.errorbar(x, y, yerr=yerr, fmt='o')
ax.set_title('Vert. symmetric')
# With 4 subplots, reduce the number
# of axis ticks to avoid crowding.
ax.locator_params(nbins=4)
ax = axs[0,1]
ax.errorbar(x, y, xerr=xerr, fmt='o')
ax.set_title('Hor. symmetric')
ax = axs[1,0]
ax.errorbar(x, y, yerr=[yerr, 2*yerr], xerr=[xerr, 2*xerr], fmt='--o')
ax.set_title('H, V asymmetric')
ax = axs[1,1]
ax.set_yscale('log')
# Here we have to be careful to keep all y values positive:
ylower = np.maximum(1e-2, y - yerr)
yerr_lower = y - ylower
ax.errorbar(x, y, yerr=[yerr_lower, 2*yerr], xerr=xerr,
                            fmt='o', ecolor='g')
ax.set_title('Mixed sym., log y')
fig.suptitle('Variable errorbars')
plt.show()
```



Matplotlib.pyplot example

- Scatter plots




```
import numpy as np
import pylab as plt

x = np.random.random(50)
y = np.random.random(50)
c = np.random.random(50)   # color of points
s = 500 * np.random.random(50)   # size of points

fig, ax = plt.subplots()
im = ax.scatter(x, y, c=c, s=s, cmap=plt.cm.jet)

# Add a colorbar
fig.colorbar(im, ax=ax)

# set the color limits - not necessary here, but good to know how.
im.set_clim(0.0, 1.0)
```

Matplotlib.pyplot example

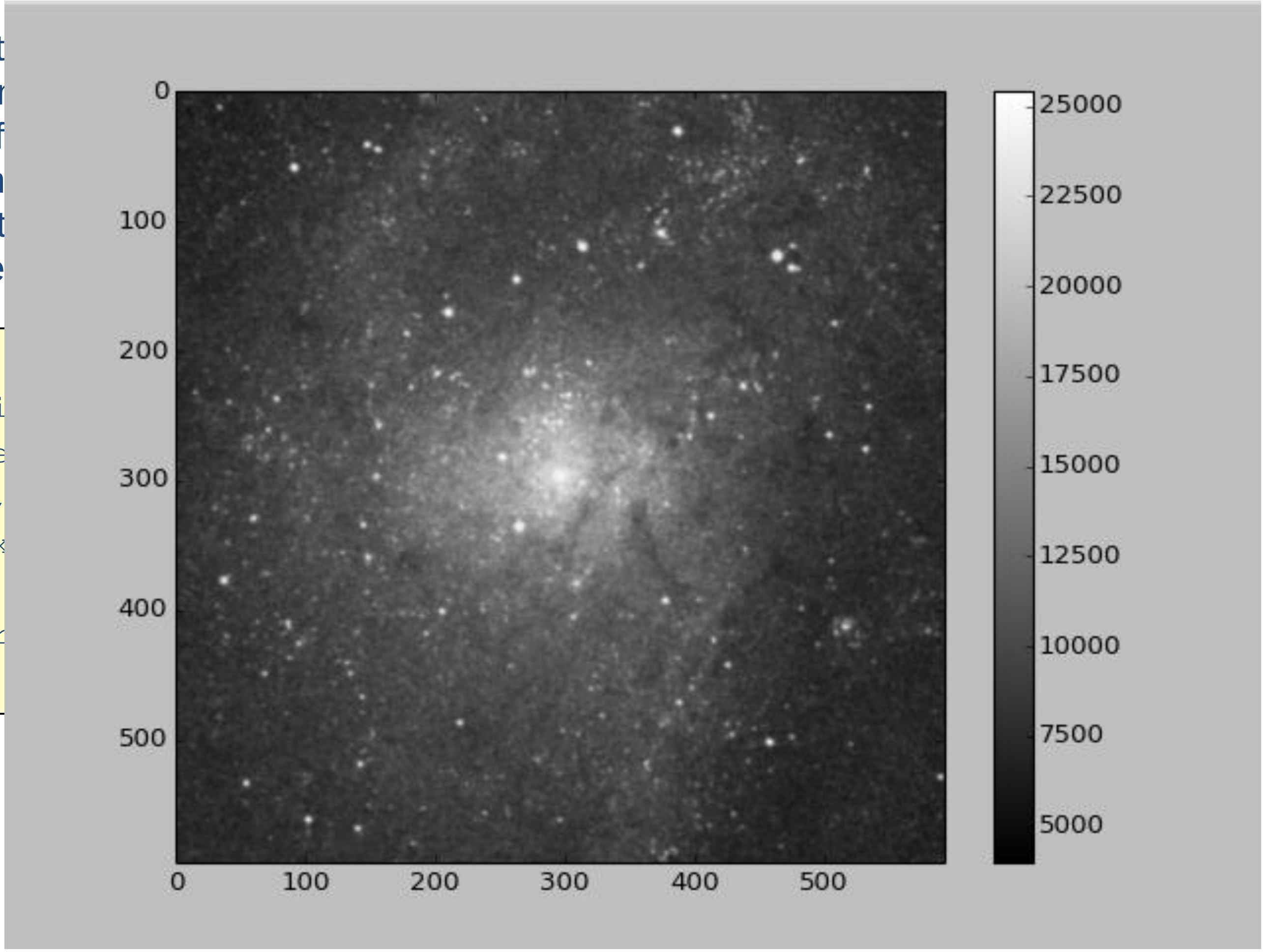
Colormaps and contour figures are useful for plotting functions of two variables. In most of these functions we will use a colormap to encode one dimension of the data. There are a number of predefined colormaps.

```
>>> import pyfits
>>> import matplotlib.pyplot as plt
>>> data = pyfits.getdata('m33.fits')
>>> plt.imshow(data, cmap='gray')
<matplotlib.image.AxesImage object at 0x7f6c09b1a250>
>>> plt.colorbar()
<matplotlib.colorbar.Colorbar instance at 0x7f6c09aabb90>
>>> plt.show()
```

Matplotlib.pyplot example

Colormaps and cont
useful for plotting fu
variables. In most of
we will use a colorm
dimension of the dat
number of predefine

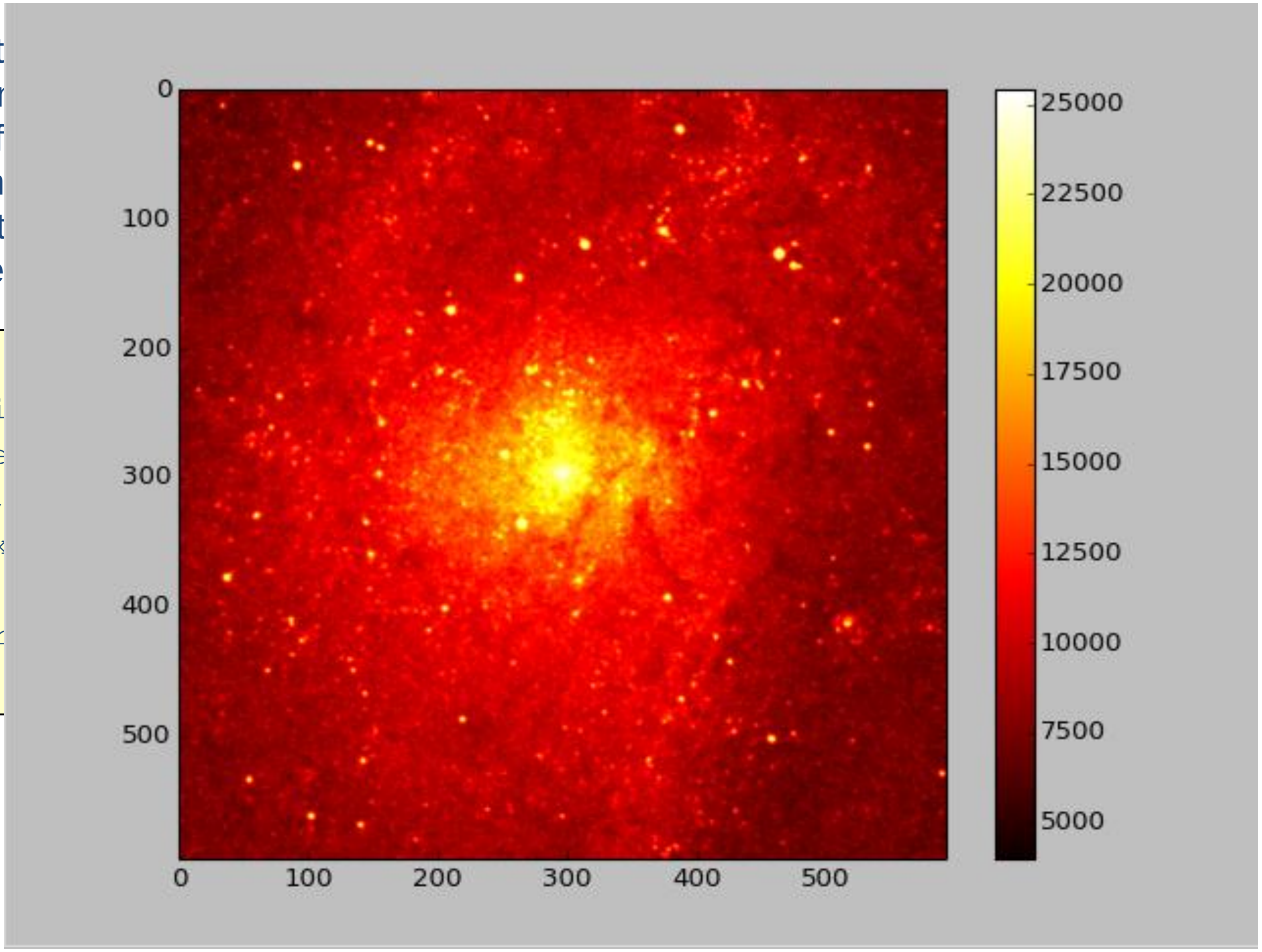
```
>>> import pyfits
>>> import matplotli
>>> data = pyfits.ge
>>> plt.imshow(data,
<matplotlib.image.Ax
>>> plt.colorbar()
<matplotlib.colorbar
>>> plt.show()
```



Matplotlib.pyplot example

Colormaps and cont
useful for plotting fu
variables. In most of
we will use a colorm
dimension of the dat
number of predefine

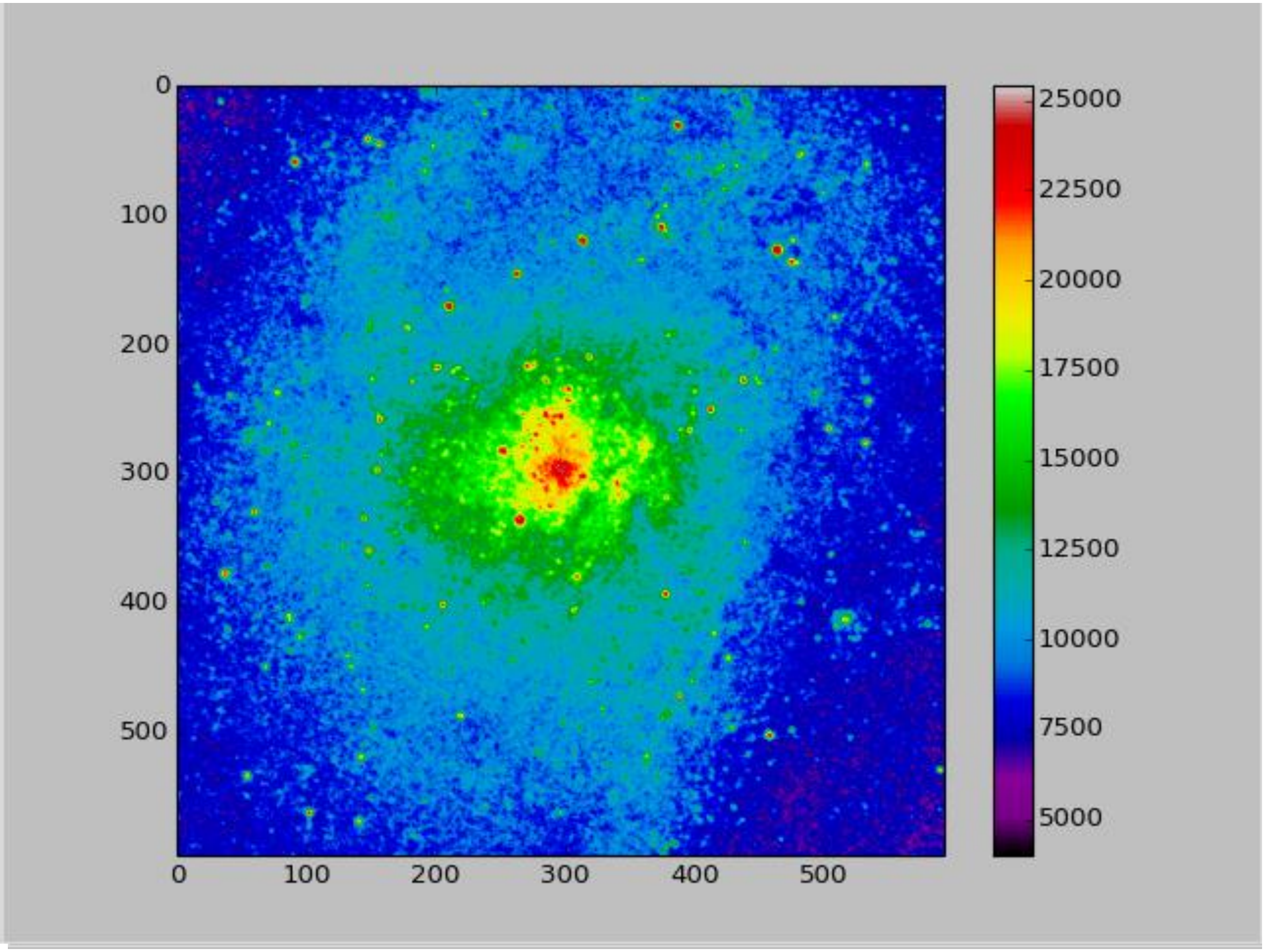
```
>>> import pyfits
>>> import matplotli
>>> data = pyfits.ge
>>> plt.imshow(data,
<matplotlib.image.Ax
>>> plt.colorbar()
<matplotlib.colorbar
>>> plt.show()
```



Matplotlib.pyplot example

Colormaps and con
useful for plotting fu
variables. In most o
we will use a colorm
dimension of the da
number of predefine

```
>>> import pyfits
>>> import matplotli
>>> data = pyfits.ge
>>> plt.imshow(data,
<matplotlib.image.Ax
>>> plt.colorbar()
<matplotlib.colorba
>>> plt.show()
```



Matplotlib.pyplot example

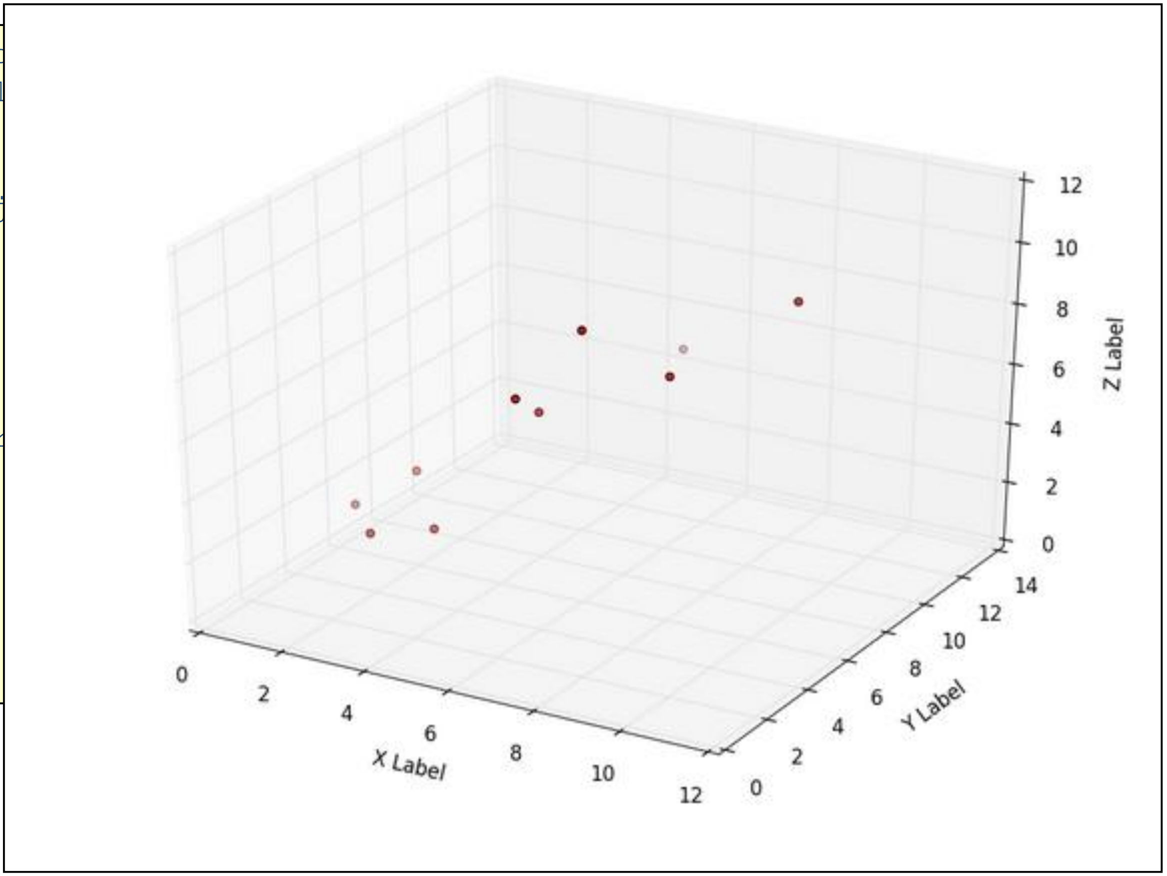
3D plots.

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x =[1,2,3,4,5,6,7,8,9,10]
y =[5,6,2,3,13,4,1,2,4,8]
z =[2,3,3,3,5,7,9,11,9,10]

ax.scatter(x, y, z, c='r', marker='o')

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```

Matplotlib.pyplot example

3D plots.

```
from mpl_toolkits.mplot3d impo
import matplotlib.pyplot as pl

fig = plt.figure()
ax = fig.add_subplot(111, proj

x =[1,2,3,4,5,6,7,8,9,10]
y =[5,6,2,3,13,4,1,2,4,8]
z =[2,3,3,3,5,7,9,11,9,10]

ax.scatter(x, y, z, c='r', mar

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```



Matplotlib.pyplot example

3D plots.

```
from numpy import *
import pylab as p
import mpl_toolkits.mplot3d.axes3d as p3

# u and v are parametric variables.
# u is an array from 0 to 2*pi, with 100 elements
u=r_[0:2*pi:100j]
# v is an array from 0 to 2*pi, with 100 elements
v=r_[0:pi:100j]
# x, y, and z are the coordinates of the points for plotting
# each is arranged in a 100x100 array
x=10*outer(cos(u),sin(v))
y=10*outer(sin(u),sin(v))
z=10*outer(ones(size(u)),cos(v)

fig=p.figure()
ax = p3.Axes3D(fig)
ax.plot_wireframe(x,y,z)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
p.show()
```