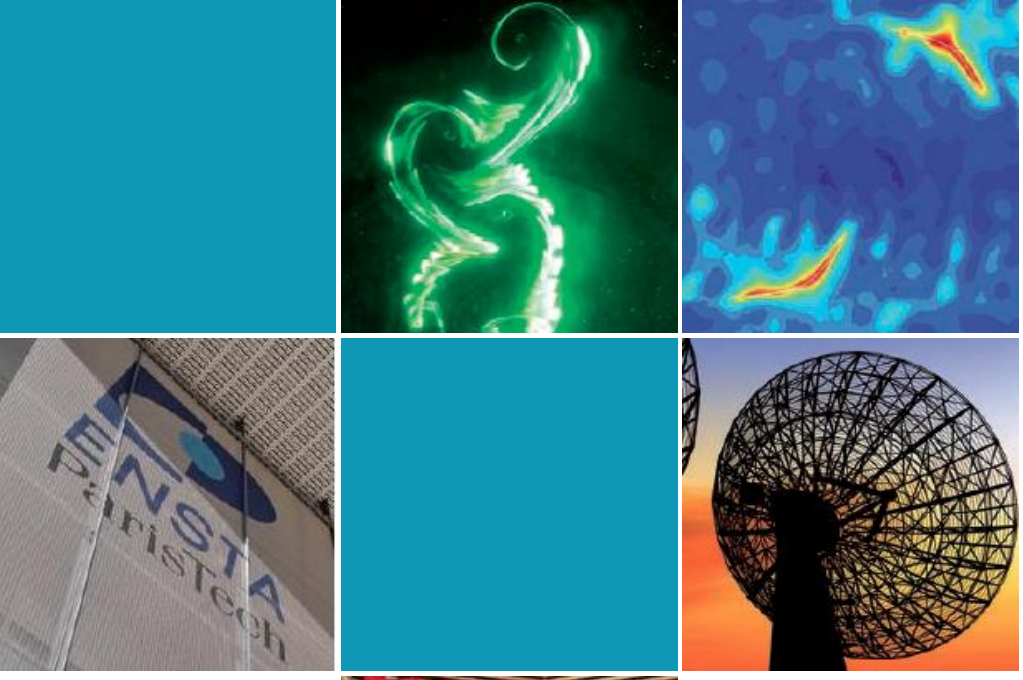


Une école
en mouvement

MO101: Python for Engineering

Vladimir Paun
ENSTA ParisTech

École Nationale Supérie
de **Techniques Avanc**

ENS
ParisTe
univer
PARIS-SAC

Introduction to Python

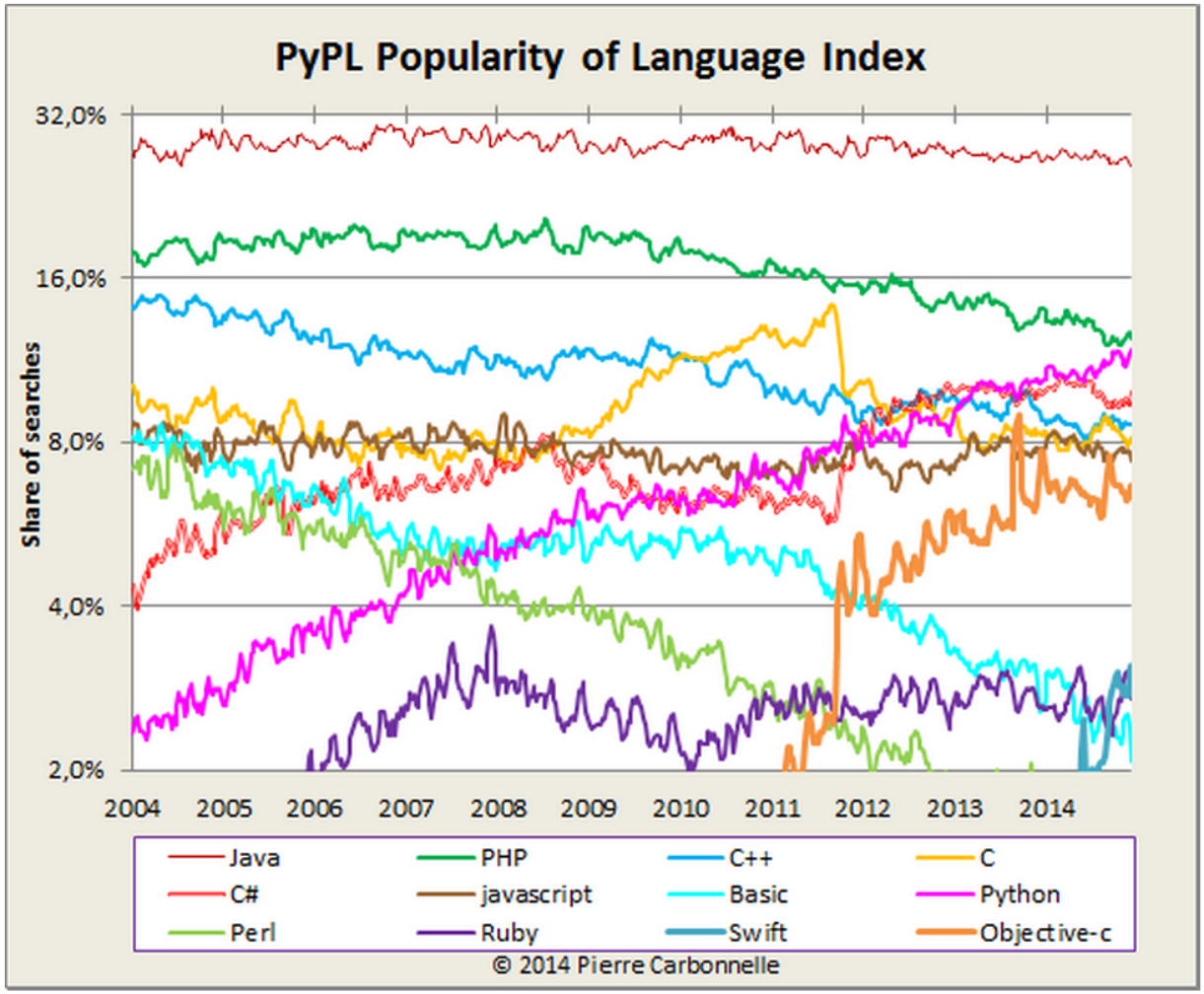
- Python itself is an *official programming language*.
- The general python includes the **programming language** and **interpreter**
- Has a (rich) standard library
- Need additional packages to plot, to do scientific computing
- Can use additional packages to do "everything"

Python in use

A general purpose programming language used extensively by:

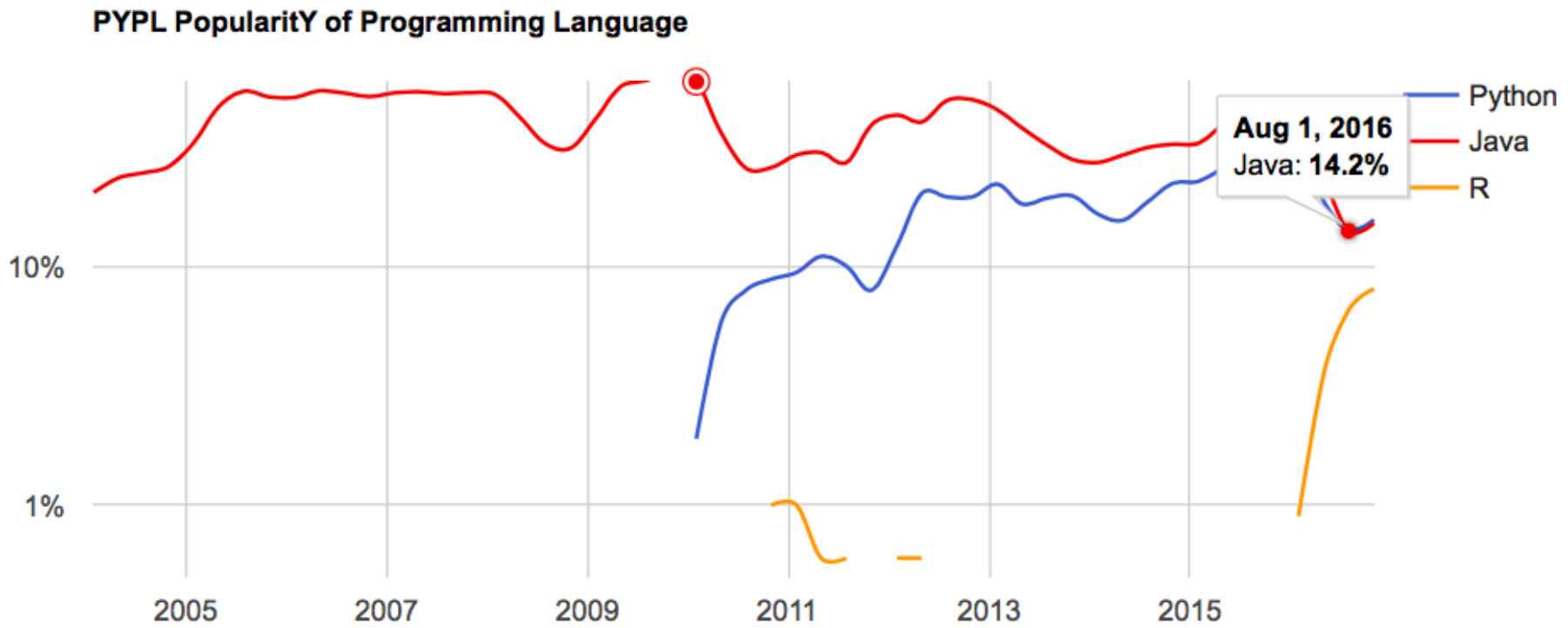
- Tech firms (YouTube, Dropbox, Reddit, etc., etc.)
- Hedge funds and finance industry
- Gov't agencies (NASA, CERN, etc.)
- Academia

Python popularity



Python popularity - France

In **France**, *Python* is the most popular language, *R* grew the most in the last 5 years (8.1%) and *Java* lost the most (–24.3%)



Scientific Programming and Python

Rapid adoption by the scientific community

- Artifical intelligence
- engineering
- computational biology
- chemistry
- physics, etc., etc.

Introduction to Python

MATLAB integrates in a single environments more than the standard Python distribution.

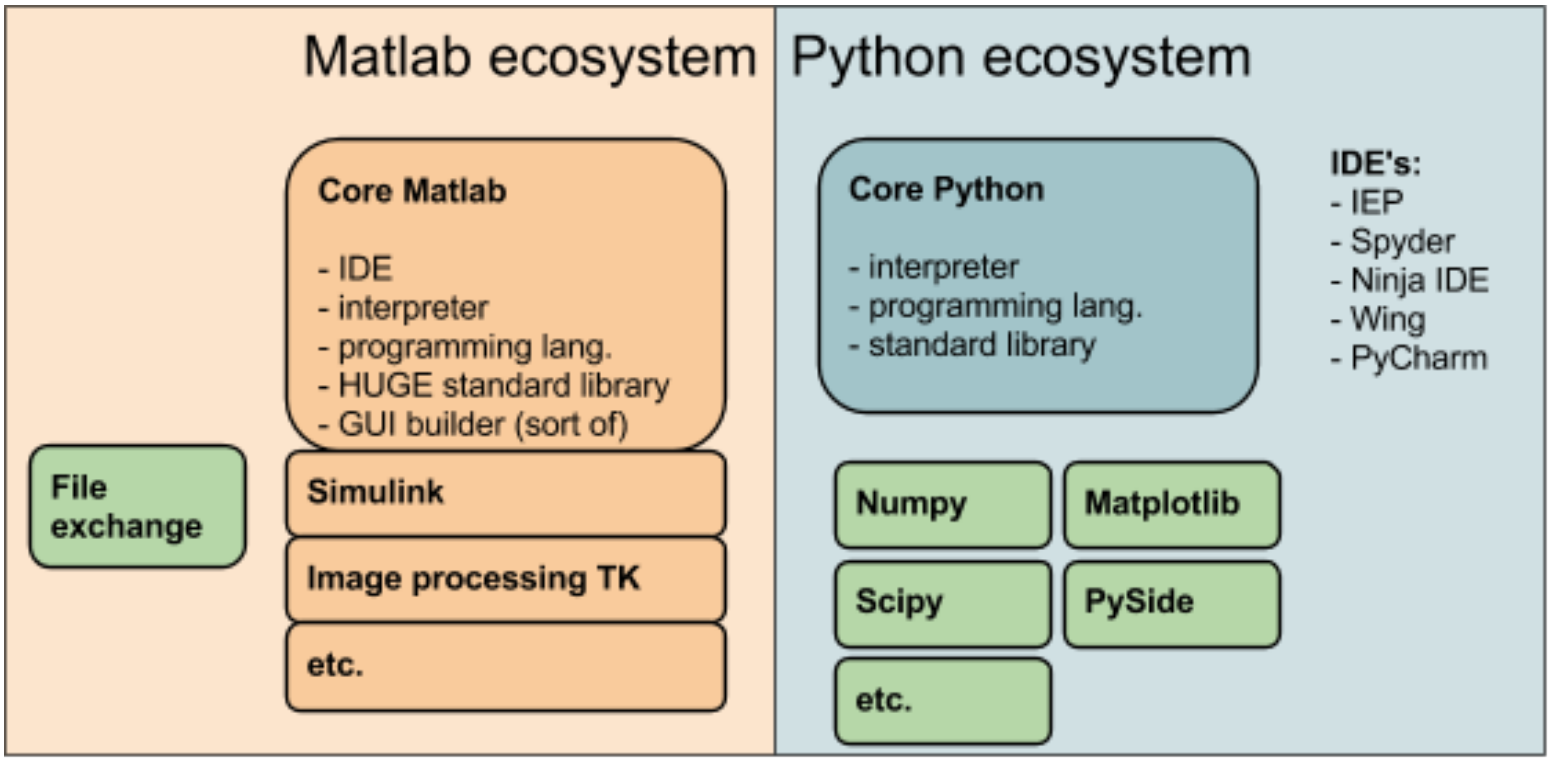


Figure: Matlab vs Python

Advantages and Disadvantages

MATLAB – Advantages

- Great IDE – Matlab Desktop
- Can do a lot with plotting
- Usually you can get access if you are at a university
- Lots of online support
- Dynamic language

MATLAB – Disadvantages

- Expensive (thanks Scilab)
- Closed source (Licensed)

Advantages and Disadvantages

Python - Advantages

- Free
- Open source
- Easy to read
- Powerful language
- Many great available libraries/packages

Python - Disadvantages

- Not as nicely packaged (for beginners)
- Have to choose an IDE (also an advantage - see PyCharm)
- Traditionally less used for non-developer scientists (but easy to learn)
- Have to import libraries/packages (don't we always?)

Important Python Packages

Many packages are created and modified by the community.

These packages allow you to do nearly everything Matlab:

Important Python Packages

Many packages are created and modified by the community.

These packages allow you to do nearly everything Matlab:

- **NumPy** – Matlab core
  - basic data types
  - simple array processing operations

Important Python Packages

Many packages are created and modified by the community.

These packages allow you to do nearly everything Matlab:

- **NumPy** – Matlab core
  - basic data types
  - simple array processing operations
- **SciPy** – Matlab Toolboxes
  - built on top of NumPy
  - provides additional functionality

Important Python Packages

Many packages are created and modified by the community.

These packages allow you to do nearly everything Matlab:

- **NumPy** – Matlab core
  - basic data types
  - simple array processing operations
- **SciPy** – Matlab Toolboxes
  - built on top of NumPy
  - provides additional functionality
- **Matplotlib** – graphing
  - 2D and 3D figures

Important Python Packages

Many packages are created and modified by the community.

These packages allow you to do nearly everything Matlab:

- **NumPy** – Matlab core
  - basic data types
  - simple array processing operations
- **SciPy** – Matlab Toolboxes
  - built on top of NumPy
  - provides additional functionality
- **Matplotlib** – graphing
  - 2D and 3D figures
- **Ipython** – like the desktop environment

NumPy

NumPy - Wikipedia:

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

NumPy

NumPy - Wikipedia:

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

- At the core of the NumPy package, is the **ndarray** object which encapsulates n-dimensional arrays of homogeneous data.

NumPy

NumPy - Wikipedia:

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

- At the core of the NumPy package, is the **ndarray** object which encapsulates n-dimensional arrays of homogeneous data.
- Many operations performed using **ndarray** objects execute in compiled code for performance

NumPy

NumPy - Wikipedia:

NumPy is an extension to the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

- At the core of the NumPy package, is the **ndarray** object which encapsulates n-dimensional arrays of homogeneous data.
- Many operations performed using **ndarray** objects execute in compiled code for performance
- The standard mathematical and scientific packages in Python use **NumPy** arrays

NumPy - example

NumPy Example: Mean and standard dev of an array

```
import numpy as np
a = np.random.randn(10000)
print(a.mean())
    # 0.0020109779347995344
print(a.std())
    # 1.009575844793006
```

ndarrays - creation

Example of **ndarrays** - creation

```
import numpy as np

normal_arr = [[1.2, 2.3], [-3.1, 4.77]]
ndarr = np.array(normal_arr)

ndarr.shape # (2, 2)
```

ndarrays - creation

Example of **ndarrays** - creation

```
import numpy as np

identity10 = np.eye(10)
ones4x2 = np.ones((4, 2))
```

ndarrays - Element access

Example of **ndarrays** element access

```
import numpy as np

A = np.ones(4)
A[0, 0] += 2
A12 = A[1, 2]

first_row = A[0,:]
last_col = A[:,-1]
```

Array creation

Several ways to create arrays...

```
import numpy as np # lists
arr = np.array([[1, 2, 3], [4, 5, 6]])

#In [1]: arr
#Out[1]:
#array([[1, 2, 3],
#       [4, 5, 6]])

# sequences
np.arange(0, 10, 0.1)
np.linspace(0, 2 * np.pi, 100)

# zeros & ones
np.zeros((5, 5))
np.ones((5, 5))

# random
np.random.random(size=(3, 4))
np.random.normal(loc=10., scale=3., size=(3, 4, 5))
```

Array IO

Several ways to create arrays...

```
import numpy as np

# create an array , write to file , read from file
arr = np.array([[1, 2, 3], [4, 5, 6]])

# save to a text file
# creates a space delimited file by default

np.savetxt(fname='array_out.txt', X=arr)
# load text file

loaded_arr = np.loadtxt(fname='array_out.txt')

np.all(arr==loaded_arr) #True
```

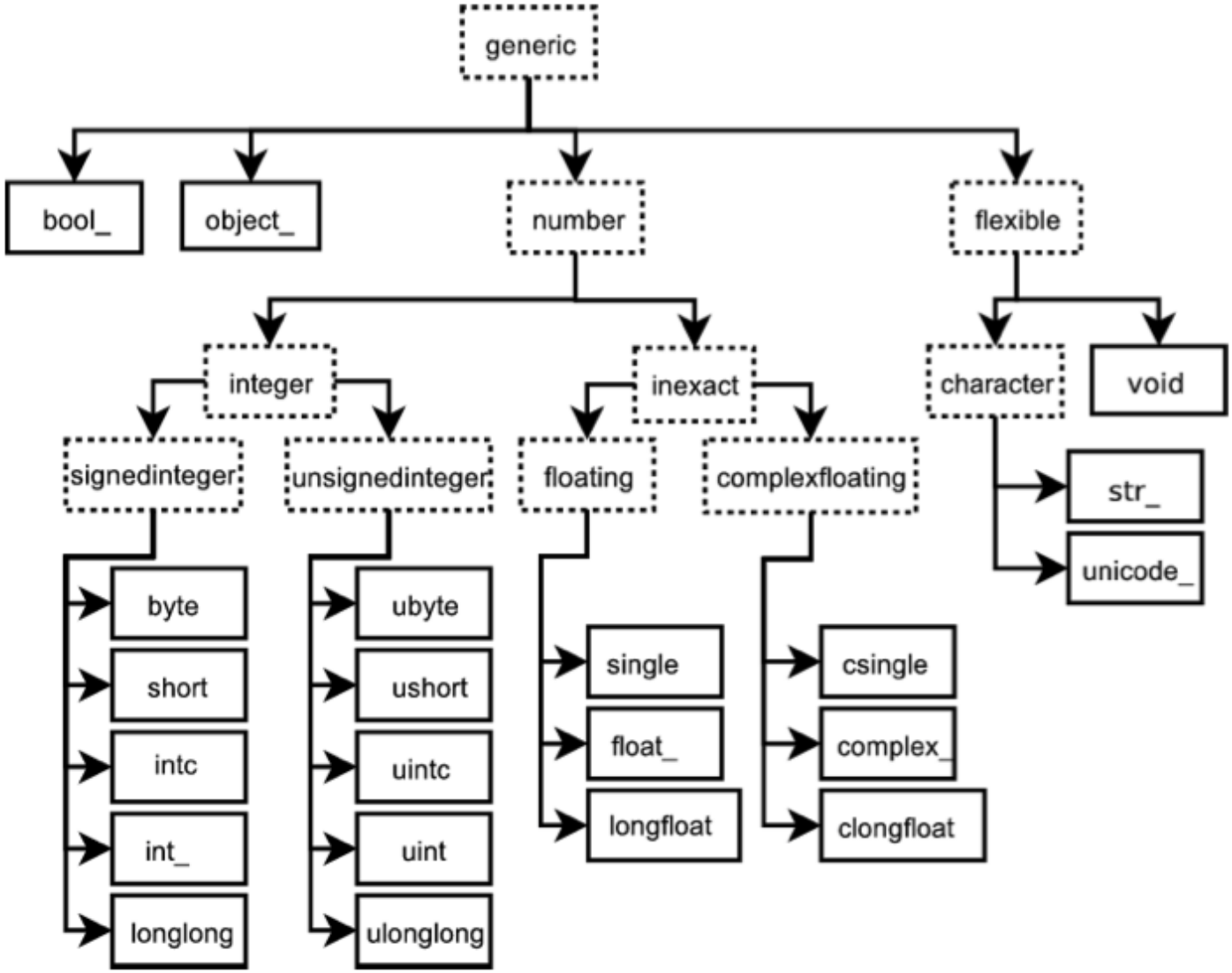
Other options control *data types*, *delimiters*, *comments*, *headers*, etc.
See documentation, especially "See Also".

Array Attributes

Arrays are objects and so have attributes and methods.

```
import numpy as np

arr = np.arange(10).reshape((2, 5))

arr.ndim   #  2 number of dimensions
arr.shape  #  (2, 5) shape of the array
arr.size   #  10 number of elements
arr.T      #  transpose
arr.dtype  #  data type of elements in the array
```

Built-in Types



Array Operations & **ufuncs**

Default behavior is element-wise

```
import numpy as np
arr1 = np.arange(10).reshape((2, 5))
arr2 = np.random.random((2, 5))

# elementwise for basic and boolean operations
# +,    ,    , /,        , np.log, <, >=,==
# arrays are upcast, resulting in float or boolean arrays
arr1 + arr2 # elementwise sum
arr1 * arr2 # elementwise multiplication

# operations in place
arr1 += arr2

# matrix product
np.dot(arr1, arr2)

# similarly numpy ufuncs operate elementwise
np.sin(arr1)
np.sqrt(arr1)
```

And many others. Explore in documentation...

Array Slicing and Indexing

Similar to lists but a few new ways to select

```
import numpy as np

arr = np.arange(20).reshape((4, 5))

# slicing (like lists for each dimension)
arr[0:4, 3:5] # all rows and last two columns
arr[:4, 3:5] # equivalent     can leave off start if 0
arr[:, 3:] # equivalent     can leave off end if size of axis
arr[slice(None), slice(3, None)] # equivalent     can use slice()

# integer indices
arr[[1, 2], :] # rows one and two, all columns
arr[np.array([1, 2]), :] # equivalent

# boolean indices
arr[[False, True, True, False], :] # equivalent
arr[np.array([False, True, True, False]), :] # equivalent
```

And many others. Explore in documentation...

Array Broadcasting & Vectorization

Broadcasting allows us to operate on arrays of different shapes by 'copying' smaller arrays when possible. This allows us to write more efficient and readable code (with fewer for loops).

```
import numpy as np

# multiplication by a scalar
arr = np.random.random((4, 5))
arr * 5  # multiply each element of the array by 5

# scales the first column by 0.
# scales the second column by 1.
# etc.
arr * np.arange(5)
```

NumPy

Linear Algebra Solvers

```
import numpy as np

list_matrix = [[1, 3, 4], [2, 3, 5], [5, 7, 9]]
A = np.array(list_matrix)
b = np.array([4, 4, 4])

# Solve for Ax = b
x = np.linalg.solve(A, b)
```

What is SciPy?

SciPy is a library of algorithms and mathematical tools built to work with NumPy arrays.

- statistics - **scipy.stats**
- optimization - **scipy.optimize**
- sparse matrices - **scipy.sparse**
- signal processing - **scipy.signal**
- etc.

Example: KS-test

Question: do two data samples come from the same distribution?

```
import scipy.stats as stats

# generate two data samples from different distributions
samp1 = stats.norm.rvs(loc=0., scale=1., size=100)
samp2 = stats.norm.rvs(loc=2., scale=1., size=100)

# perform ks test : null hypothesis is distributions are the same
D, pval = stats .ks_2samp(samp1, samp2) # D=.58, pval=1.34 e 15

# reject the null

# generate two data samples from the same distribution
samp1 = stats.norm.rvs(loc=0., scale=1., size=100)
samp2 = stats.norm.rvs(loc=0., scale=1., size=100)

# perform ks test
D, pval = stats.ks_2samp(samp1, samp2) # D=.09, pval=.79
  # fail to reject the null
```

SciPy - example

SciPy Example: Calculate

$$\int_{-2}^{2} \phi(z)dz$$

where $\phi \sim N(0,1)$

```
from scipy.stats import norm
from scipy.integrate import quad
phi = norm()
value, error = quad(phi.pdf, -2, 2)
print(value)
    # 0.9544997361036417
```

What is Matplotlib

What it is? [matplotlib.org]

`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 (à la MATLAB or Mathematica),
- web application servers,
- six graphical user interface toolkits.

Matplotlib purpose

matplotlib is the standard Python plotting library and is very useful for data analysis.
matplotlib can be used in:

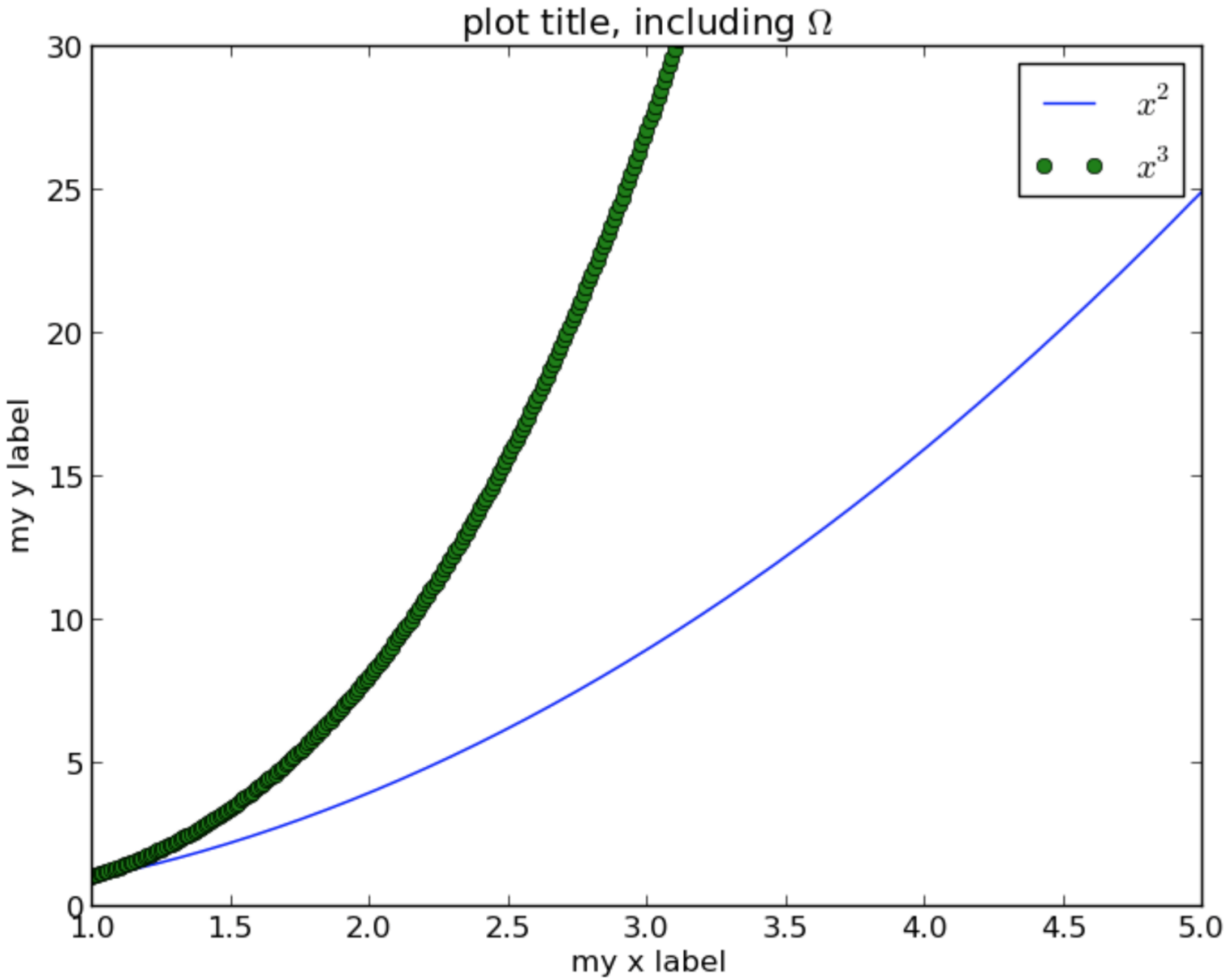
- create histograms - `matplotlib.pyplot.hist`,
- power spectra - `matplotlib.mlab.psd`,
- bar charts - `matplotlib.pyplot.bar`,
- errorcharts,
- scatterplots,
- etc.

Line plot example

Adding multiple lines and a legend

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 1000)
y1 = np.power(x, 2)
y2 = np.power(x, 3)

plt.plot(x, y1, 'b-', x, y2, 'go')
plt.xlim((1, 5))
plt.ylim((0, 30))
plt.xlabel('my x label')
plt.ylabel('my y label')
plt.title('plot title , including $\Omega$')
plt.legend(('$x^2$', '$x^3$'))

plt.savefig('line plot plus2.png')
```

Line plot example - result

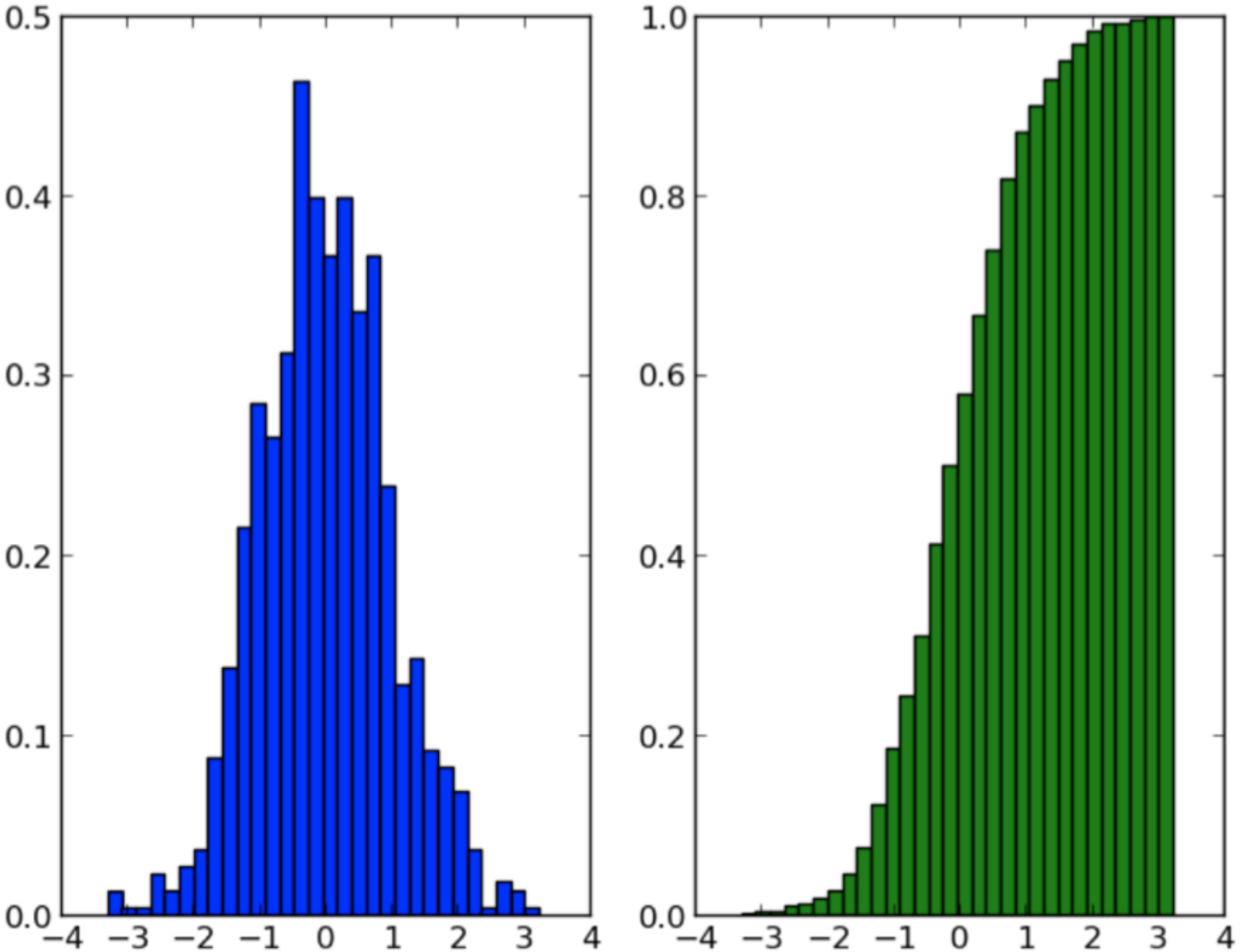


Histogram

Adding multiple lines and a legend

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

data = np.random.randn(1000)

# histogram (pdf)
plt.subplot(1, 2, 1)
plt.hist(data, bins=30, normed=True, facecolor='b')

# empirical cdf
plt.subplot(1, 2, 2)
plt.hist(data, bins=30, normed=True, color='g',
         cumulative=True)

plt.savefig('histogram.png')
```

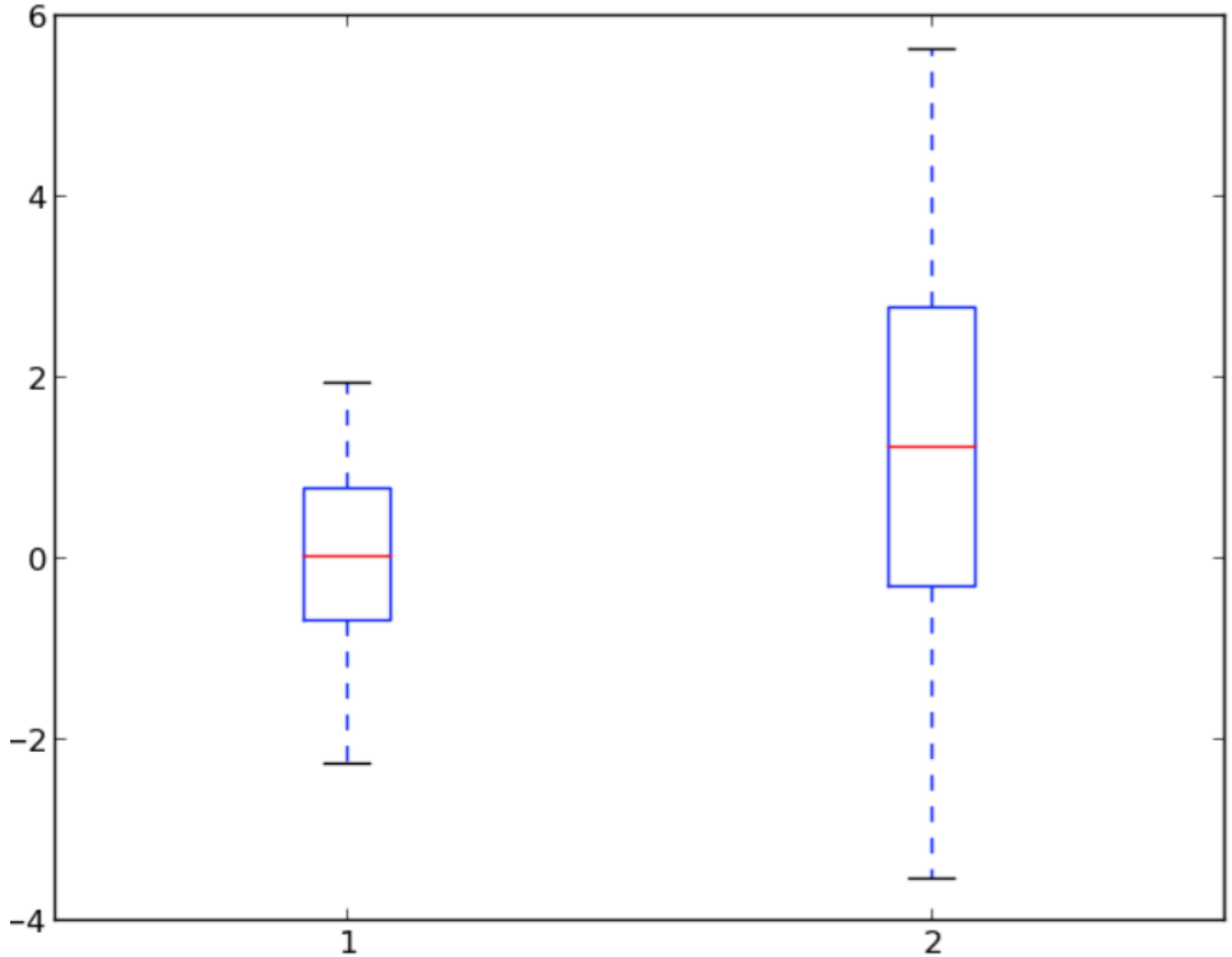
Histogram



Box Plot

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

samp1 = np.random.normal(loc=0., scale=1., size=100)
samp2 = np.random.normal(loc=1., scale=2., size=100)

plt.boxplot((samp1, samp2))
plt.savefig('boxplot.png')
```

Box Plot - result



Scatter Plot Matrix

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

from pandas.tools.plotting import scatter_matrix
from pandas import DataFrame

df = DataFrame(np.random.normal(loc=0.,
                                scale=1.,
                                size=(1000, 5)),
               columns=['a', 'b', 'c', 'd', 'e'])
scatter_matrix(df, alpha=0.4, diagonal='kde')

plt.savefig('scattermatrix.png')
```

Scatter Plot Matrix - result

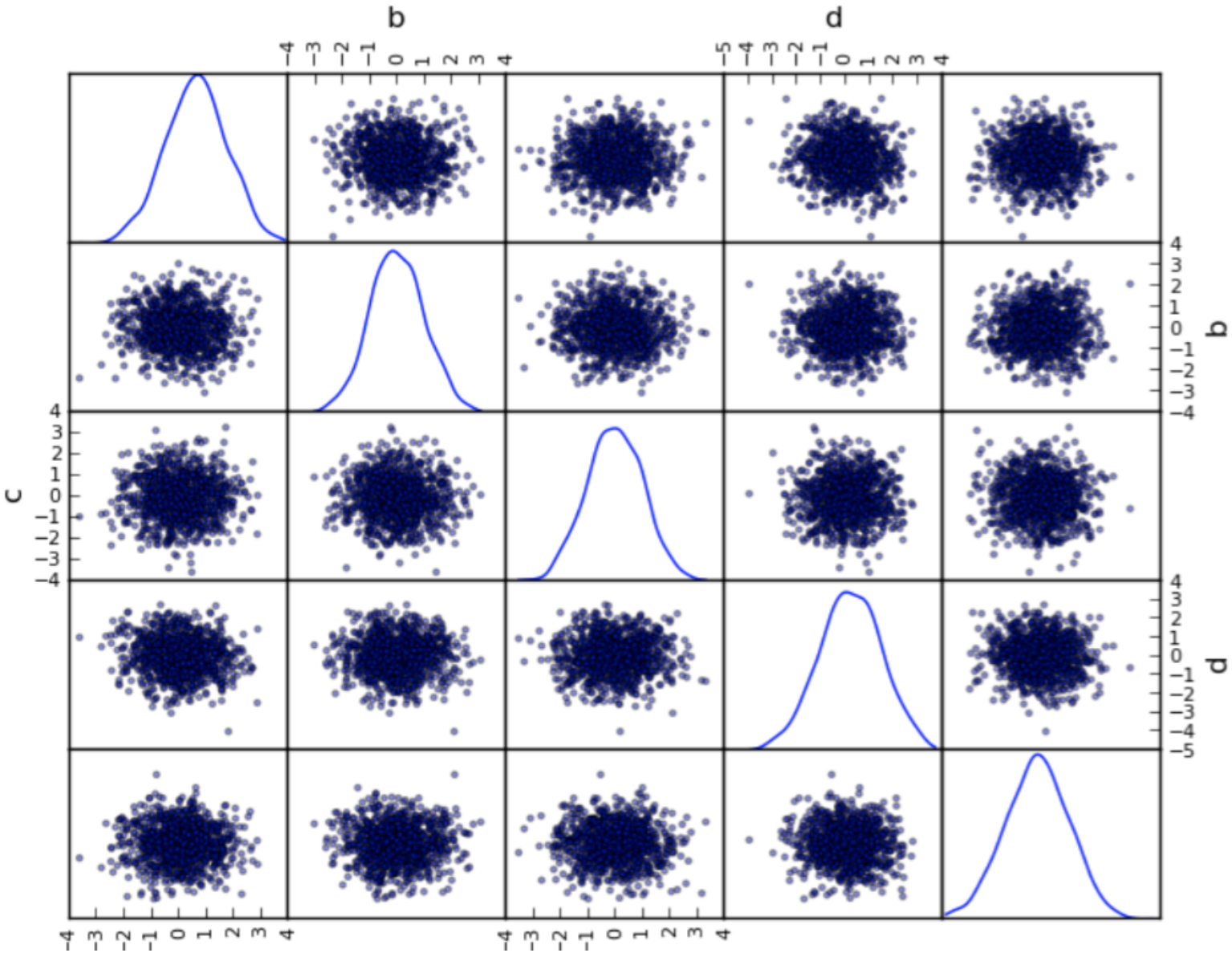
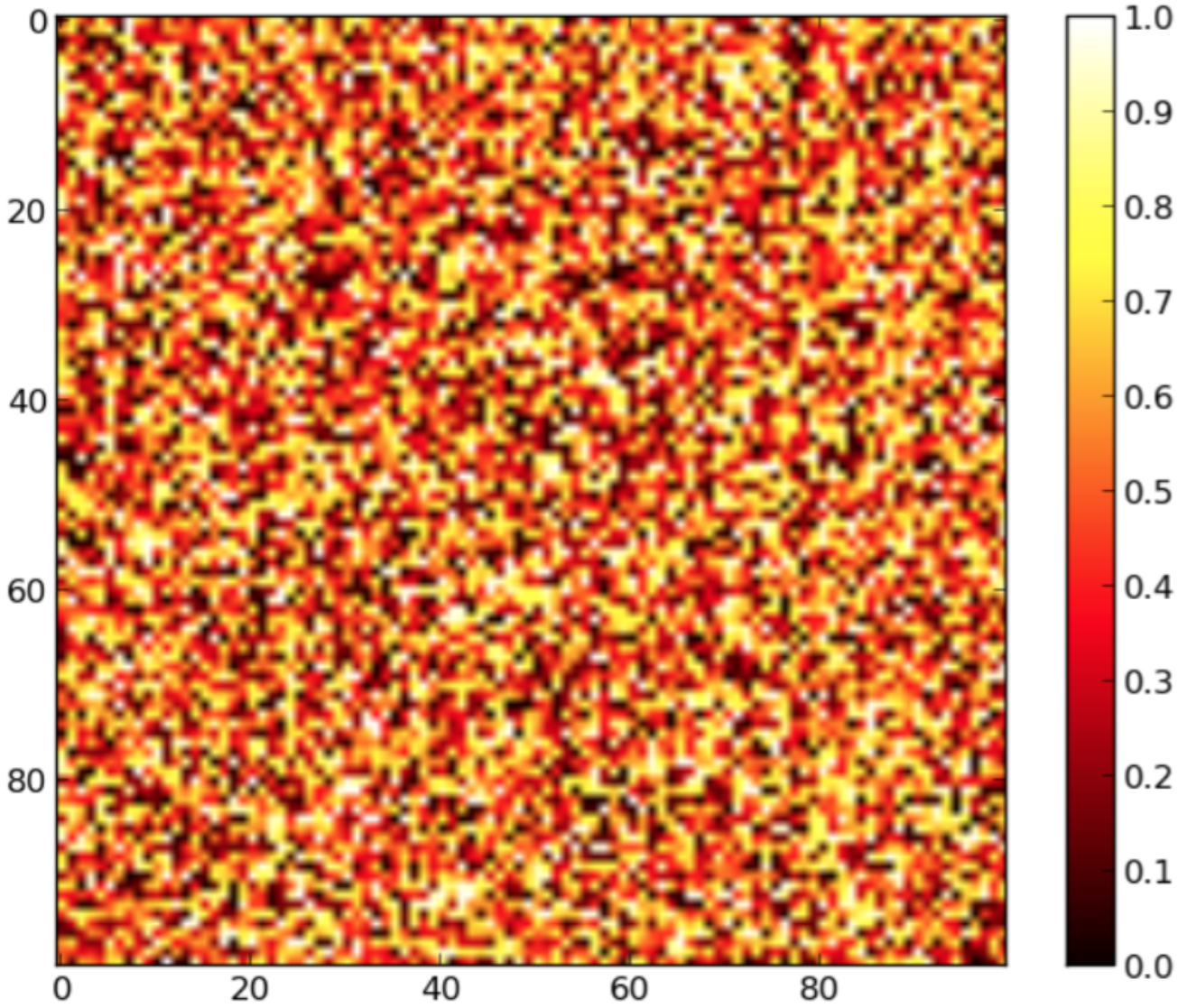


Image Plot

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

A = np.random.random((100, 100))
plt.imshow(A)
plt.hot()
plt.colorbar()

plt.savefig('imageplot.png')
```

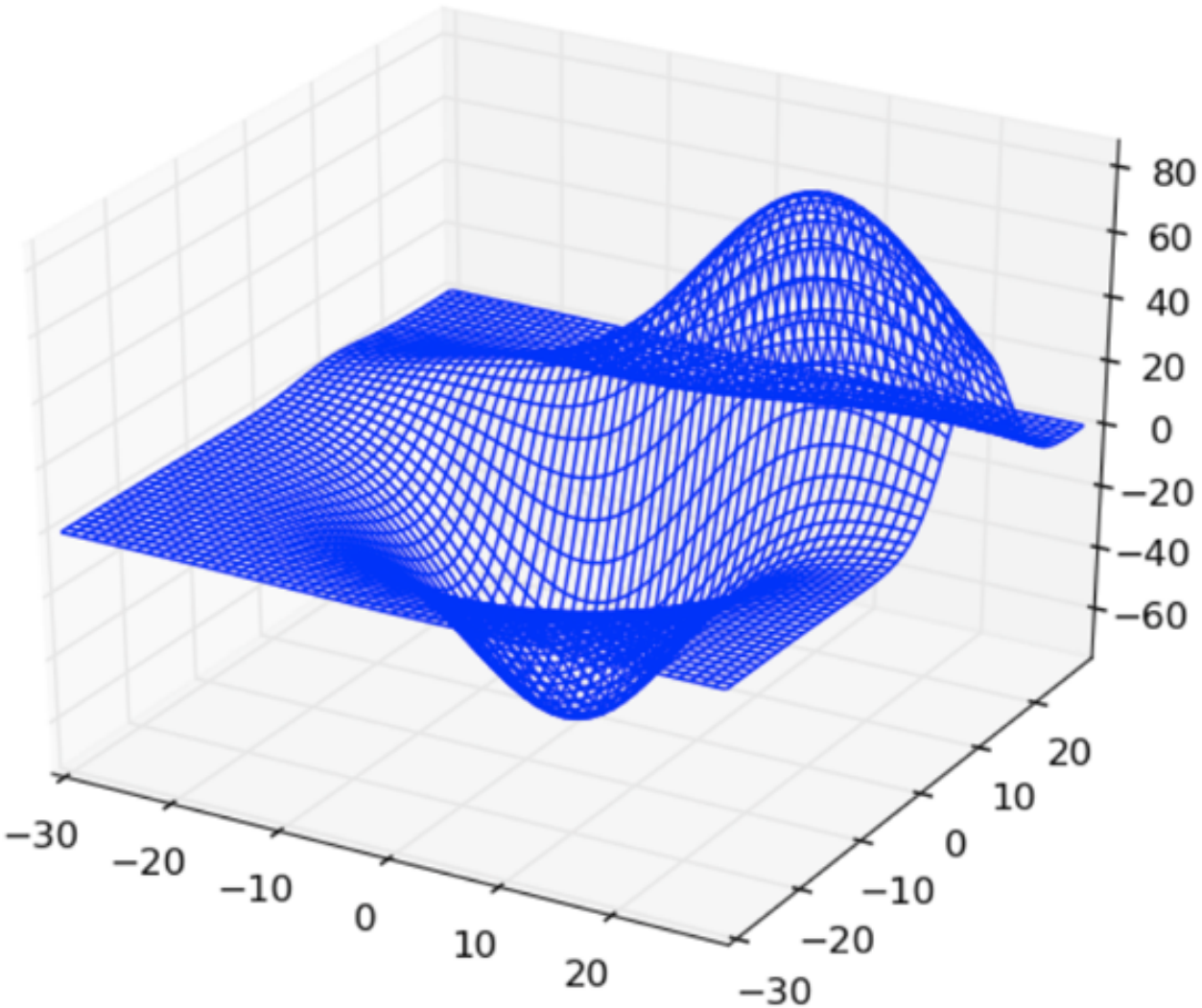
Image Plot - result



Wire Plot

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt

ax = plt.subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.1)
ax.plot_wireframe(X, Y, Z)

plt.savefig('wire.png')
```

Wire Plot - result



Curve fitting of polynomials

Let `x` and `y` be two arrays that contain data to which we like to fit a curve (*to minimise the least square deviation of the fit from the data*).
Numpy provides the routine **polyfit(x,y,n)**, similar to Matlab's polyfit function that:

- takes a list *x* of *x* – *values* for data points,
- a list *y* of *y* – *values* of the same data points
- a desired order of the polynomial that will be determined to fit the data in the least-square sense as well as possible.

Curve fitting of polynomials

```
import numpy

#demo curve fitting: xdata and ydata are input data
xdata = numpy.array([0.0, 1.0, 2.0, 3.0, 4.0, 5.0])
ydata = numpy.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])

#now do fit for cubic (order = 3) polynomial
z = numpy.polyfit(xdata , ydata , 3)

#z is an array of coefficients , highest first , i.e.
# x^3 X^2 X 0
#z=array([ 0.08703704, -0.81349206, 1.69312169, -0.03968254])

#It is convenient to use    poly1d    objects for dealing with
 polynomials:
p = numpy.poly1d(z) # creates a polynomial function p from coefficients
# and p can be evaluated for all x then.

#create plot
xs = [0.1 * i for i in range(50)]
ys = [p(x) for x in xs] # evaluate p(x) for all x in list xs

import pylab
```

Curve fitting of polynomials

```
import pylab
pylab.plot(xdata, ydata, 'o', label='data')
pylab.plot(xs, ys, label='fitted curve')
pylab.ylabel('y')
pylab.xlabel('x')
pylab.savefig('polyfit.pdf')
pylab.show()
```

Curve fitting of polynomials

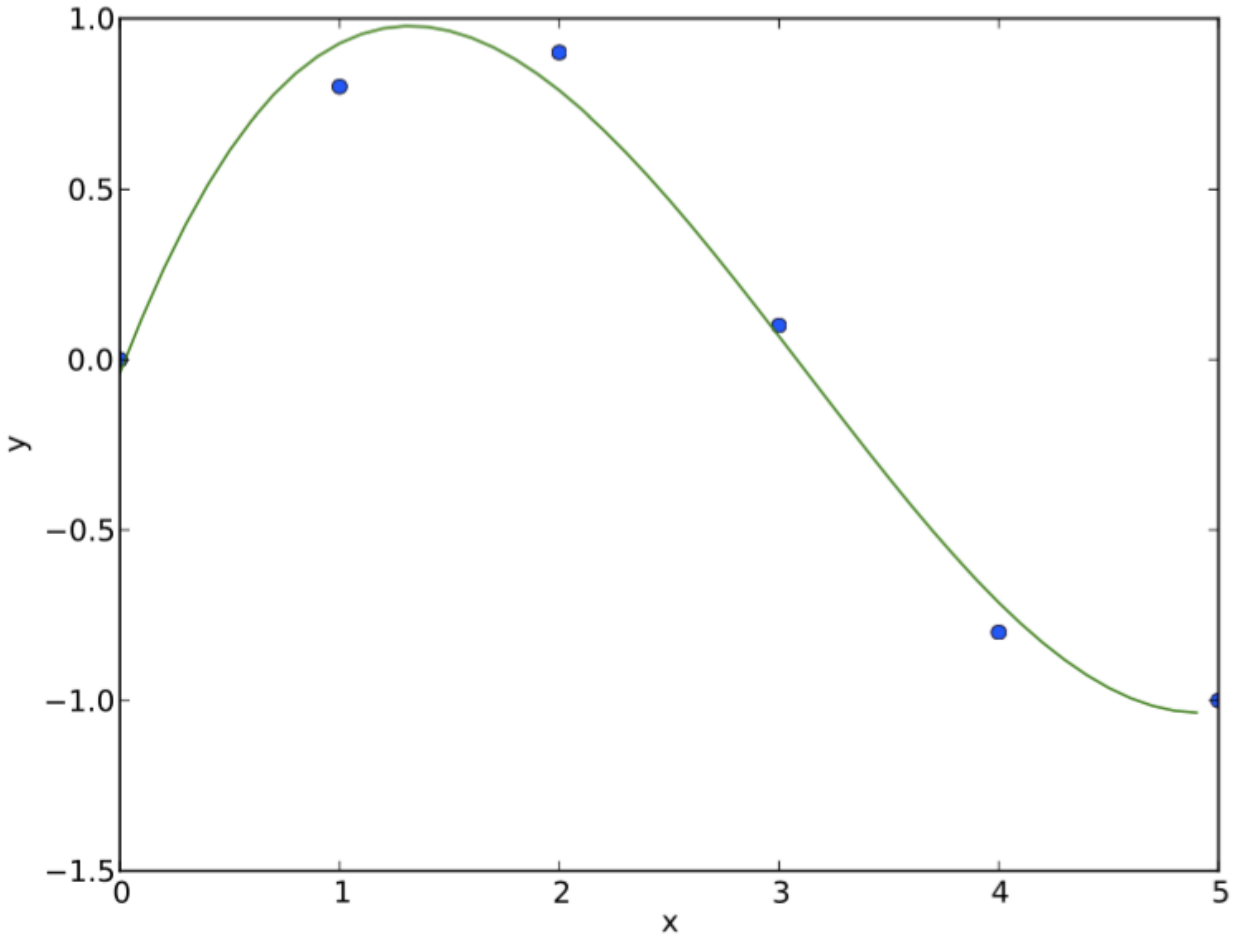


Figure: least-squares curvefitting with numpy

Other Scientific Libraries

Pandas

- statistics and data analysis

SymPy

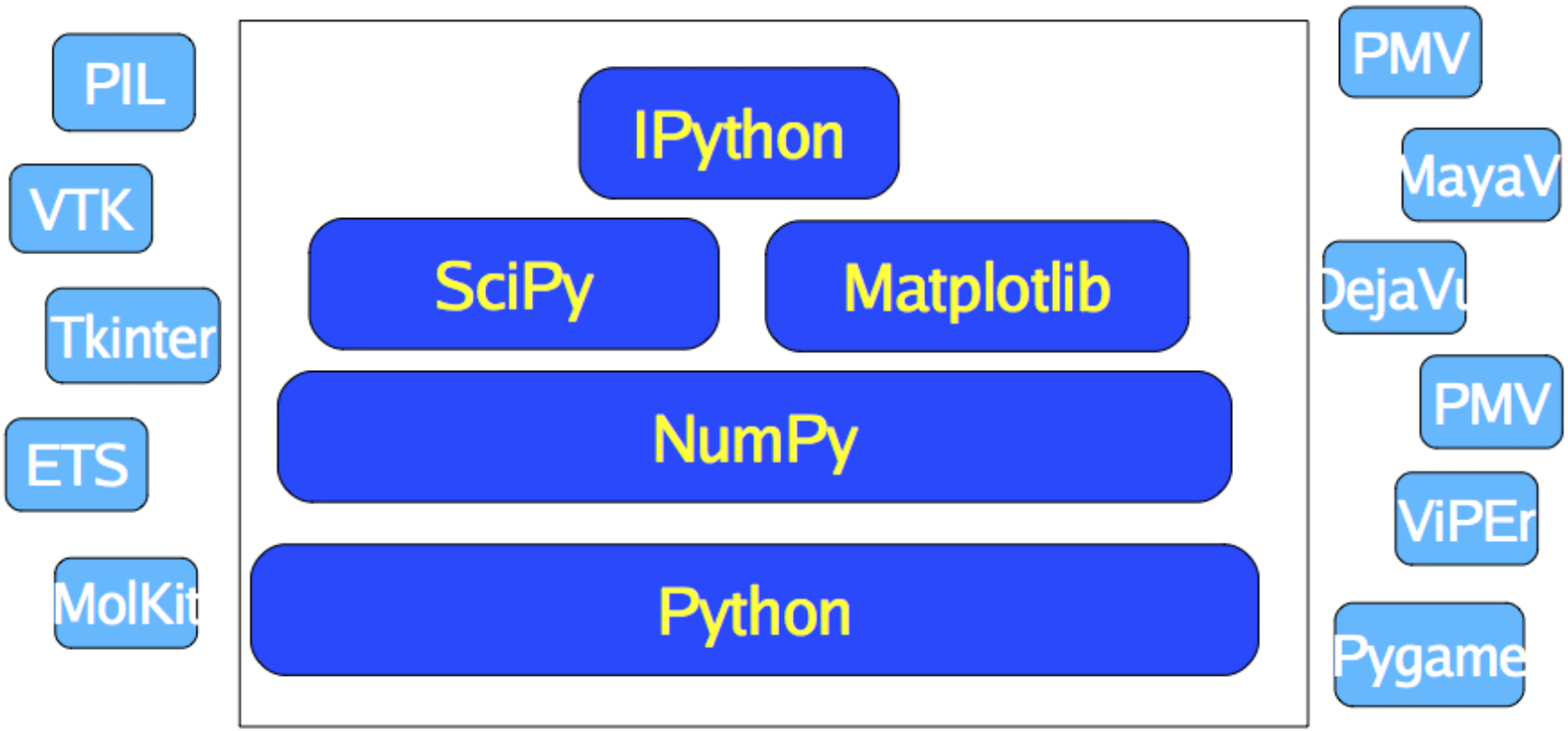
- symbolic manipulations à la Mathematica

Still more:

- `statsmodels` — statistics / econometrics
- `scikit-learn` — machine learning in Python

Other Scientific Tools

`https://www.scipy.org`



Other Scientific Tools

Also tools for

- working with graphs (as in networks)
- parallel processing, GPUs
- manipulating large data sets
- interfacing C / C++ / Fortran
- cloud computing
- database interaction
- bindings to other languages, like R and Julia
- etc.

References

- http://www.scipy-lectures.org/_downloads/ScipyLectures.pdf
- http://matplotlib.org
- http://www.scipy-lectures.org/index.html
- http://www.scipy-lectures.org/_downloads/ScipyLectures.pdf
- https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html
- http://web.stanford.edu/~schmit/cme193/

Further reading

http://www.southampton.ac.uk/~fangohr/training/python/pdfs/Python-for-Computational-Science-and-Engineering.pdf