Data Analysis With Python

Arnaud Beck

Laboratoire Leprince-Ringuet, École Polytechnique, CNRS/IN2P3

Space Science Training Week

Outline

Why come to Python ?

Should I use low-level,compiled language or an interpreted language ?
Commercial or open source ?

	C/C++	Matlab	Python
Easy and flexible		X	X
Performances	X		
Free and available on any system	X		X

C++ g++ used what fraction? **used** how many times more?

Benchmark	Time	Memory	Code
mandelbrot	1/121		±
n-body	1/97		±
spectral-norm	1/80		2×
fannkuch-redux	1/73		2×
fasta	1/54		2×
k-nucleotide	1/17	1/3	2×
binary-trees	1/15	1/3	±
reverse-complement †	1/7	1/3	7×
regex-dna	1/3	±	±
pidigits	±		3×

Why stick to Python ?

Python is distinguished by its large and active scientific computing community.
There are people developing “libraries” for virtually anything.

Glue to other languages

Libraries to interface other languages (C/C++/Fortran)...
...with the same performances ! !
Critical part of codes are written in a lower level language.

Parallelization

- MPI
- OpenMP
- GPU

Data management and visualization

- IO data in any format (HDF5, VTK, ...)
- Data management dedicated libraries (scipy, pandas)
- Direct visualization or interfaces with other softwares (Paraview, Mayavi)

Outline

Getting Python for data analysis

Basic Python distribution

Available on any Linux or Mac OS.

Critical for data analysis

Modules : Scipy, Matplotlib

Application specific

Modules : mpi4py, VTK, pytable, etc.

It is possible to install fully pre-built scientific Python environment : “Enthought Python Distribution” or “Python(x,y)” for Windows.

Running Python

Interactive mode in a Python shell

```
[arnaud@beck ~]$     python
Python 2.7.3 (default, Jul 24 2012, 10:05:38)
[GCC 4.7.0 20120507 (Red Hat 4.7.0-5)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "Hello world"
>>> print a
Hello world
>>>
```

Use of a script

```
[arnaud@beck ~]$ more hello_world.py
a = "Hello world"
print a
[arnaud@beck ~]$ python hello_world.py
Hello world
[arnaud@beck ~]$
```

Turn your python script into a unix script

```
[arnaud@beck ~]$ more hello_world.py
#!/usr/bin/env python
a = "Hello world"
print a
[arnaud@beck ~]$ ./hello_world.py
Hello world
[arnaud@beck ~]$
```

You can compile scripts into binary .pyc files. Mostly for developers.

IPython : a convenient and comfortable Python shell

Interesting features

- Command history
- Any Xterm command accessible via ’ !’
- Commands auto-completion
- Quick help through the use of “ ?”
- Inline and interactive graphics
- Timing and profiling tools
- Many many more ...

Best tool for exploring, debugging or work interactively. Have a look !

IPython example


```
[arnaud@beck ~]$ ipython
Python 2.7.3 (default, Jul 24 2012, 10:05:38)
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]: a = [1,2,3,4]

In [2]: a?
Type:       list
Base Class: <type 'list'>
String Form:[1, 2, 3, 4]
Namespace:  Interactive
Length:     4
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items

In [3]: a.
a.append   a.count    a.extend   a.index    a.insert   a.pop      a.remove   a.reverse  a.sort

In [3]: a.count?
Type:       builtin_function_or_method
Base Class: <type 'builtin_function_or_method'>
String Form:<built-in method count of list object at 0x2161a28>
Namespace:  Interactive
Docstring:  L.count(value) -> integer -- return number of occurrences of value

In [4]: a.count(3)
Out[4]: 1
```

Outline

Python is an object oriented language

"In Python, we do things with stuff !"

things = operations

stuff = objects

Type	Example
Numbers	`128, 3.14, 4+5j`
Strings	`'Rony', "Giovanni's"`
Lists	`[1,"string",2.45]`
Tuples	`(1,"string",2.45)`

Strings, Lists and Tuples are sequences.

Strings and Tuples are "immutable".

Numbers


```
In [22]: a = 122; type(a)
Out[22]: int

In [23]: a = 122.2; type(a)
Out[23]: float

In [24]: (a+2)*4
Out[24]: 496.8

In [25]: 4/3
Out[25]: 1

In [26]: 4./3
Out[26]: 1.3333333333333333

In [27]: 4**5
Out[27]: 1024

In [28]: 2**100
Out[28]: 1267650600228229401496703205376L
```

Strings

Ordered collection (or sequence) of characters


```
In [2]: s = "spam"; c = " a lot !"; s + c
Out[2]: 'spam a lot !'

In [3]: s*3
Out[3]: 'spamspamspam'

In [4]: s[0] = c
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/home/arnaud/<ipython-input-4-27903bb729b1> in <module>()
----> 1 s[0] = c

TypeError: 'str' object does not support item assignment

In [5]: s.replace("pa","a")
Out[5]: 'sam'

In [6]: "42" + 1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/home/arnaud/<ipython-input-6-eb0154d479ea> in <module>()
----> 1 "42" + 1

TypeError: cannot concatenate 'str' and 'int' objects

In [7]: int("42") + 1
Out[7]: 43

In [8]: "42" + repr(1)
Out[8]: '421'
```

String Methods


```
In [12]: "42.12   58.4  1.3e-3".split()
Out[12]: ['42.12', '58.4', '1.3e-3']

In [13]: "       hello    ".strip()
Out[13]: 'hello'

In [14]: "42".rjust(6,"0")
Out[14]: '000042'

In [15]: " or ".join("123456")
Out[15]: '1 or 2 or 3 or 4 or 5 or 6'

In [16]: "This is the end".endswith("the end")
Out[16]: True
```

Lists
Sequence of any objects


```
In [53]: list = [2,45,28,9]; complement = [3,"ert"]; list + complement
Out[53]: [2, 45, 28, 9, 3, 'ert']

In [54]: list * 2
Out[54]: [2, 45, 28, 9, 2, 45, 28, 9]

In [55]: list.append(3);list
Out[55]: [2, 45, 28, 9, 3]

In [56]: list.extend([234,457,"ola"]); list
Out[56]: [2, 45, 28, 9, 3, 234, 457, 'ola']

In [57]: list.append([234,457,"ola"]); list
Out[57]: [2, 45, 28, 9, 3, 234, 457, 'ola', [234, 457, 'ola']]

In [58]: list.sort();list
Out[58]: [2, 3, 9, 28, 45, 234, 457, [234, 457, 'ola'], 'ola']

In [59]: list.pop(3);list #removes 3rd element
Out[59]: [2, 3, 9, 45, 234, 457, [234, 457, 'ola'], 'ola']

In [60]: list.remove(3);list #removes element 3
Out[60]: [2, 9, 45, 234, 457, [234, 457, 'ola'], 'ola']

In [61]: len(list) #Works for any sequence
Out[61]: 7

In [62]: range(6) #Generates a list
Out[62]: [0, 1, 2, 3, 4, 5]
```

Slices
Manipulating sequences


```
In [74]: list = range(10); list
Out[74]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [75]: list[0]
Out[75]: 0

In [76]: list[-1]
Out[76]: 9

In [77]: list[1:]
Out[77]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [78]: list[1:5]
Out[78]: [1, 2, 3, 4]

In [79]: list[1:-3]
Out[79]: [1, 2, 3, 4, 5, 6]

In [80]: list[1:8:2]
Out[80]: [1, 3, 5, 7]

In [81]: list[1:8:2][:2]
Out[81]: [1, 3]

In [82]: s="spam"

In [83]: s=s[2:]+"erica"; s
Out[83]: 'america'
```

Importing modules

Modules define new object types and operations.

```
In [38]: import math

In [39]: math.pi
Out[39]: 3.141592653589793

In [40]: import random

In [41]: random.random()
Out[41]: 0.6015750142337749

In [42]: random.choice([1,2,3,4,"bingo",6,7,8,9.3])
Out[42]: 8
```

The large and growing Python users community provides an increasing number of modules that already do what you need.

Outline

The Scipy module

Scipy is a collection of powerful , high level functions for mathematics and data management. It is based on the numpy.ndarray object type and vectorized operations. The operations are optimized and coded in C to deliver high performances.

```
In [134]: a=scipy.arange(10000000)

In [135]: %time for i in range(len(a)):a[i]=a[i]**2
CPU times: user 8.26 s, sys: 0.07 s, total: 8.33 s
Wall time: 8.31 s

In [136]: %time a = a**2
CPU times: user 0.02 s, sys: 0.00 s, total: 0.02 s
Wall time: 0.03 s
```

If you are using a for loop, you are probably doing something wrong !

Creating an ndarray


```
In [140]: scipy.arange(10)
Out[140]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [141]: scipy.zeros((5,5))
Out[141]:
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.]])

In [142]: scipy.ones(5)
Out[142]: array([ 1.,  1.,  1.,  1.,  1.])

In [143]: scipy.linspace(0,3.1415,10)
Out[143]:
array([ 0.        ,  0.34905556,  0.69811111,  1.04716667,  1.39622222,
        1.74527778,  2.09433333,  2.44338889,  2.79244444,  3.1415    ])

In [144]: scipy.fromstring("0.  1.  2.32  1.45",dtype=float,sep=" ")
Out[144]: array([ 0.  ,  1.  ,  2.32,  1.45])

In [145]: list = range(5);scipy.array(list)
Out[145]: array([0, 1, 2, 3, 4])
```

Manipulating ndarrays

- Slicing is still the basis of array manipulation.
- Reshape –> Change number and size of dimensions of the array.
- Sort –> Quite self explanatory.
- Delete, insert, append –> Remove or add parts of the array.
- Squeeze, flatten, ravel –> More ways to control dimensionality of the array.
- Transpose,swapaxes, rollaxis –> More ways to arange the dimensions as you want

These functions are important because a well aranged data is a quickly processed data.

Extracting information from your data


```
In [168]: a=scipy.rand(10)

In [169]: a[a>0.5]
Out[169]: array([ 0.51838586,  0.80146882,  0.82736855,  0.63072527])

In [170]: w = scipy.where(scipy.sqrt(a)<0.5); print w
(array([1, 2, 3, 8]),)

In [171]: arg = a.argsort();print arg
[2 1 8 3 7 5 0 9 4 6]

In [172]: a[w]
Out[172]: array([ 0.11826273,  0.09938495,  0.21119653,  0.20446567])

In [173]: a[arg]
Out[173]:
array([ 0.09938495,  0.11826273,  0.20446567,  0.21119653,  0.25749345,
        0.43061162,  0.51838586,  0.63072527,  0.80146882,  0.82736855])
```

- Intersection (convenient for filtering)
- Histograms (perfect for distribution functions)
- Convolution
- Integration
- Interpolation
- Name it ...

Outline

Reading data

The whole game is to fit your data in a `ndarray`.

```
data = scipy.fromfile("file",dtype='float32',count=-1,sep=" ")
```

Works with raw binary files and ASCII files but not very flexible.

```
data = scipy.loadtxt("file",skiprows=0,delimiter=",")
```

More flexible but works only with text files.

The `file` object

The `file` object is a basic python type. It is created by

```
fid = open("filename","r")
```

"r" for read, "w" for write.

- fid.readline() –> reads a line in a `string`
- fid.readlines() –> reads all line in a `list` of `strings`
- fid.tell() –> returns the file's current position (in byte)
- fid.seek(n) –> goes to position n
- fid.read() –> reads all file in a `string`
- fid.close()

Manipulating a file


```
[arnaud@beck 600TW_light_bestcase0]$ head -n 3 final_density_full.csv
"phaseqxqyqz","Points:0","Points:1","Points:2"
0.0000000000e+00,1.1230000000e+05,-9.4500000000e+02,0.0000000000e+00
0.0000000000e+00,1.1230025781e+05,-9.4500000000e+02,0.0000000000e+00
[arnaud@beck 600TW_light_bestcase0]$ ipython
Python 2.7.3 (default, Jul 24 2012, 10:05:38)
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]: fid = open("final_density_full.csv",'r'); fid.readline()
Out[1]: '"phaseqxqyqz","Points:0","Points:1","Points:2"\n'

In [2]: import scipy; fid.tell()
Out[2]: 47

In [3]: data = scipy.fromfile(fid,count=-1,dtype=float,sep=","); data
Out[3]: array([      0.,  112300.,    -945.,       0.])

In [4]: fid.seek(47); data = fid.read().replace("\n",",")

In [5]: data = scipy.fromstring(data,dtype=float,sep=","); scipy.shape(data)
Out[5]: (6832468,)

In [6]: data=data.reshape(-1,4); data[46,:]
Out[6]:
array([  1.38599993e-04,   1.12311930e+05,  -9.45000000e+02,
         0.00000000e+00])
```

Quick words about reading HDF5 files

Reading HDF5 files is module dependant. You can use either “tables” or “h5py” for instance.

These modules coexist well with `Scipy` and load data directly into `ndarray`.

`tables` example

```
In [18]: import scipy

In [19]: import tables

In [20]: h5_file = tables.openFile("proc5.hdf", mode = "r")

In [21]: data = h5_file.root.
h5_file.root.energy      h5_file.root.moments     h5_file.root.topology
h5_file.root.fields      h5_file.root.potentials

In [21]: data = h5_file.root.moments.
h5_file.root.moments.phi         h5_file.root.moments.species_0
h5_file.root.moments.rho         h5_file.root.moments.species_1

In [21]: data = h5_file.root.moments.species_0.Jx.cycle_0.read()

In [22]: type(data)
Out[22]: numpy.ndarray

In [23]: scipy.shape(data)
Out[23]: (21, 41, 1)
```

Writing data

- `scipy.save("file",ndarray)` and `scipy.load("file")` in order to use the binary scipy format to store arrays.
- `ndarray.tofile()` in order to store an array in a text file or raw binary.
- `fileobject.write("any_string")` to write a string in a text file.
- The `h5py` and `tables` modules are used to write HDF5 files.

VTK script

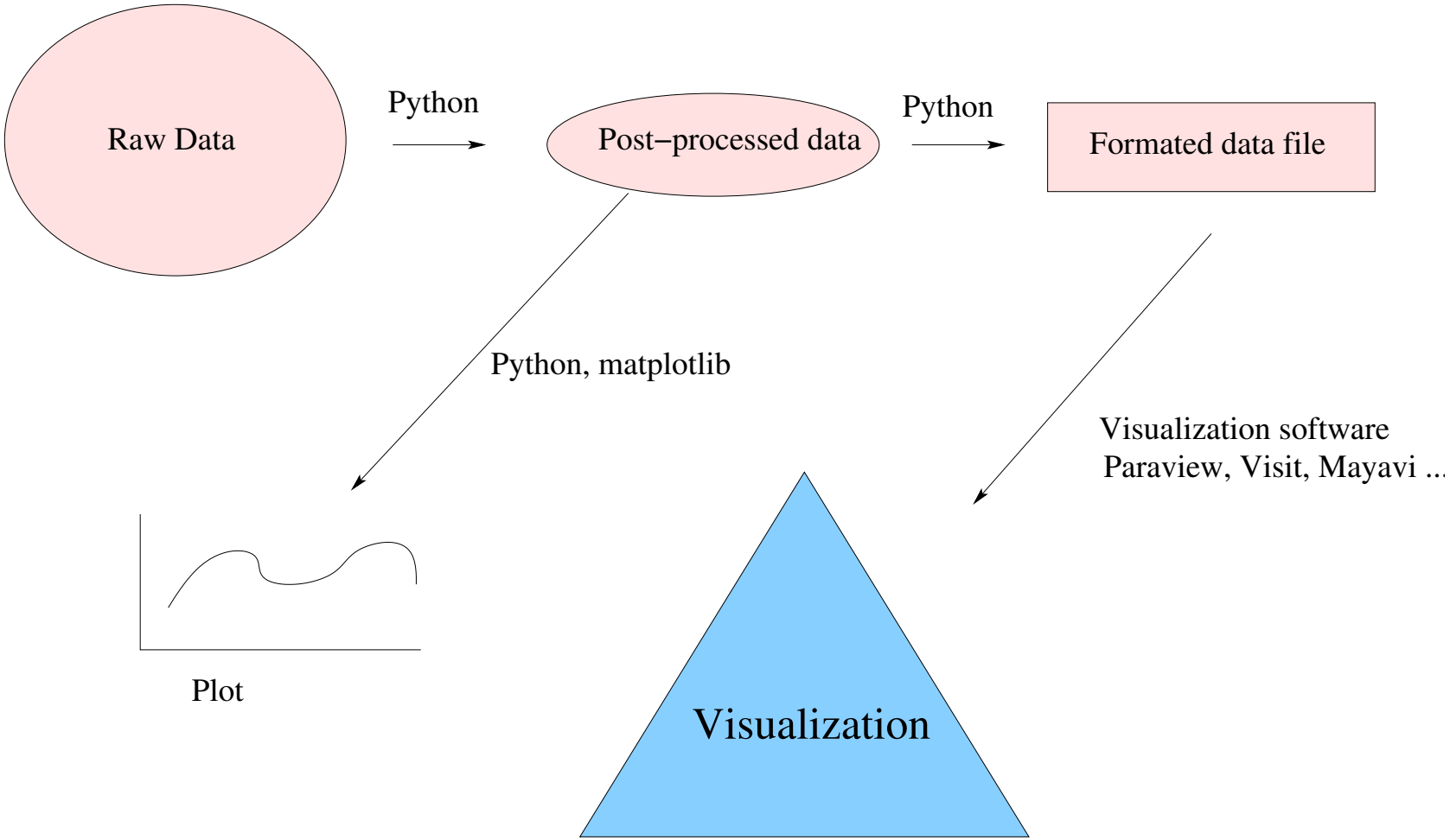
```
img = vtk.vtkImageData()
img.SetOrigin(xmin,ymin,zmin)
img.SetDimensions(nx,ny,nz)
img.SetExtent(0,nx-1,0,ny-1,0,nz-1)
img.SetSpacing(dx_local*reducefactor,dy_local,dz_local)

numpoints = nx*ny*nz
vtk_datachamp = vtk.vtkFloatArray()
vtk_datachamp.SetNumberOfTuples(numpoints)
vtk_datachamp.SetNumberOfComponents(1)
vtk_datachamp.SetVoidArray(champ, numpoints, 1)
vtk_datachamp.SetName(namedata)
img.GetPointData().SetScalars(vtk_datachamp)

writer =vtk.vtkXMLPImageDataWriter()
writer.SetFileName(data_dir+'/'+cas+'/'+ifile+".pvti")
writer.SetNumberOfPieces(numberOfPieces);
writer.SetEndPiece(numberOfPieces-1);
writer.SetStartPiece(0);
writer.SetInput(img)
writer.Write()
```

Outline

Visualization workflow



Matplotlib : the `figure` object

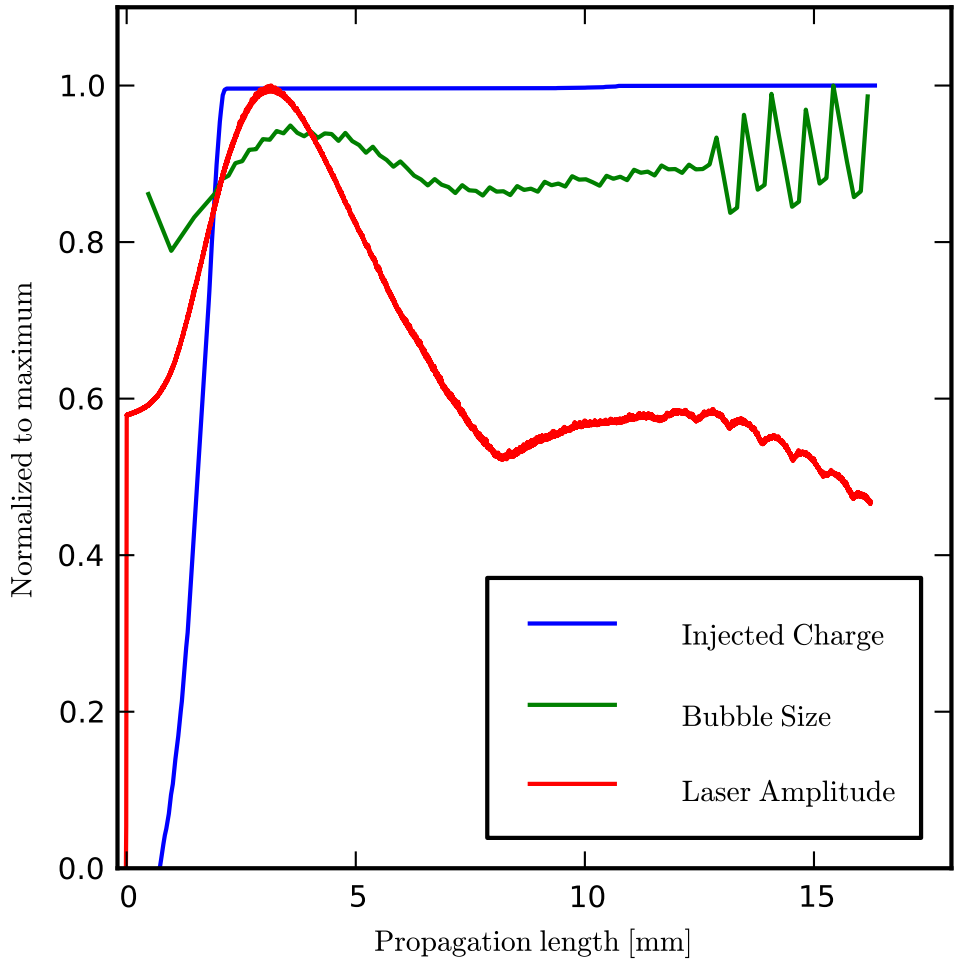

```
fig = figure([options])
```

Options include :

- Size in inches
- Dpi
- Face and edge colors
- Frame layout

Operations include :

- Title and axis labels
  `fig.xlabel("string")`
- Axis ticks and extent
  `fig.ticks(ndarray)`
- Display a colorbar
  `fig.colorbar()`
- Display a legend
  `fig.legend()`
- Save figure (png or eps)
  `fig.savefig()`

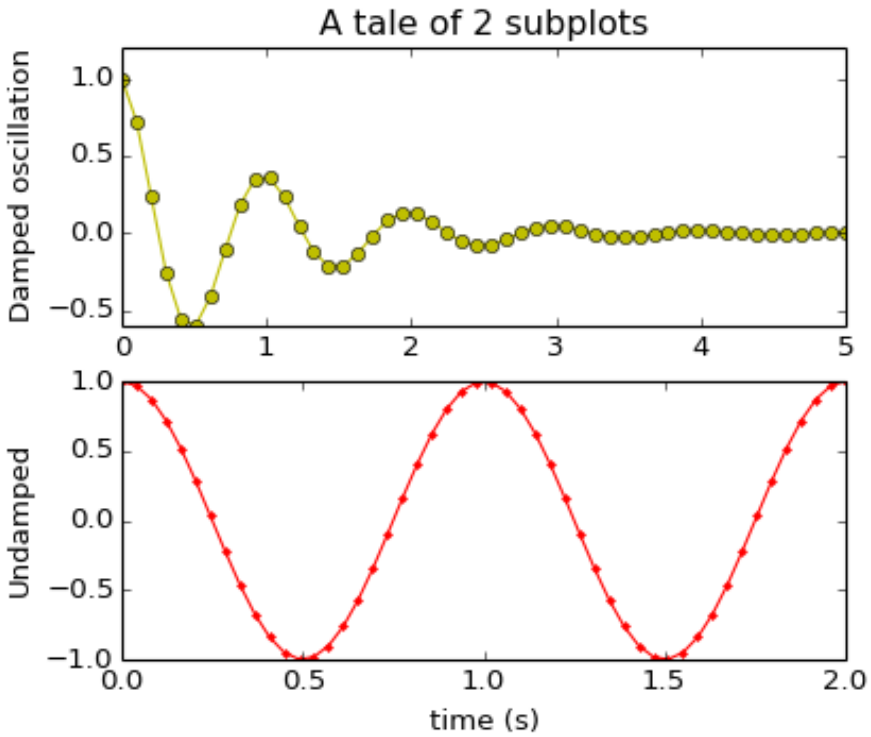


Matplotlib : Simple plots


```
plot(x,y,[options])
```

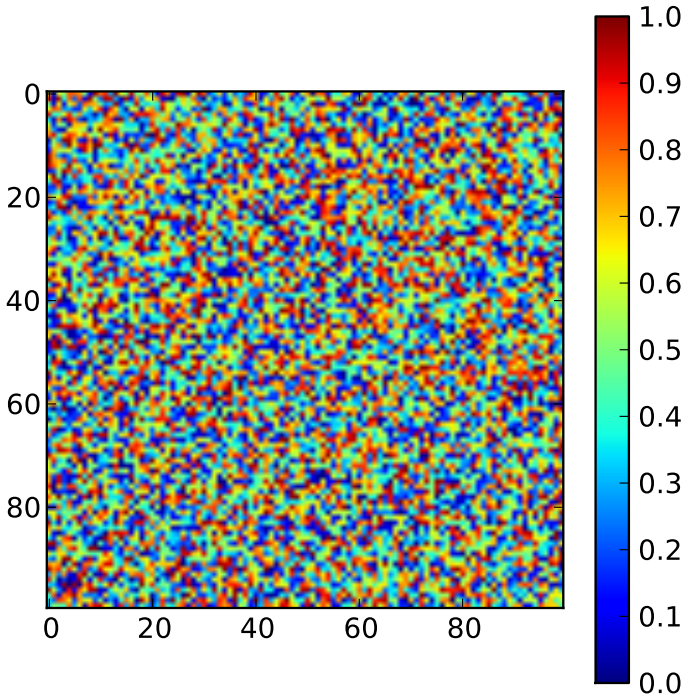
If x is omitted, default is `x=range(len(y))`.

All typical options are here : lines (style, color, width ...), markers (size, shape, colors ...), labels for legend, antialiasing, transparency, many more ...

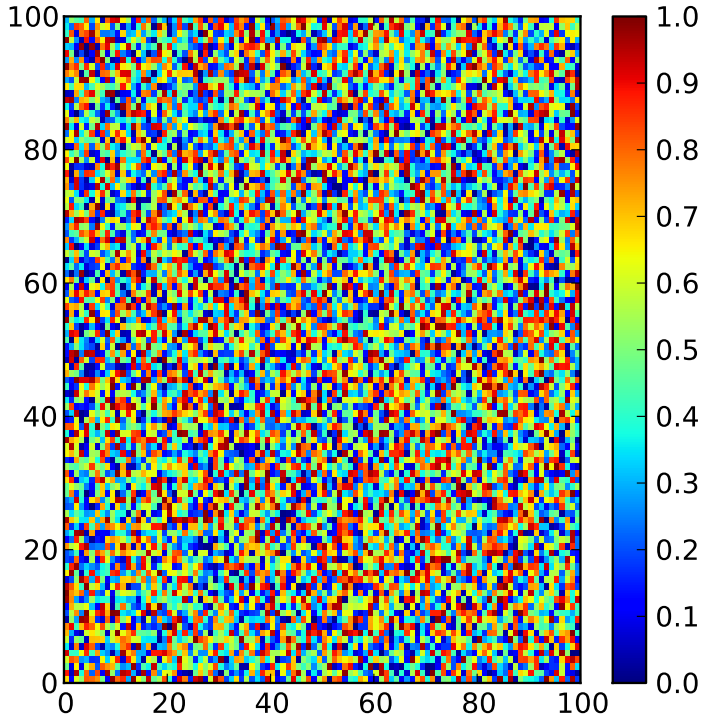


Matplotlib 2D plots : imshow and pcolor

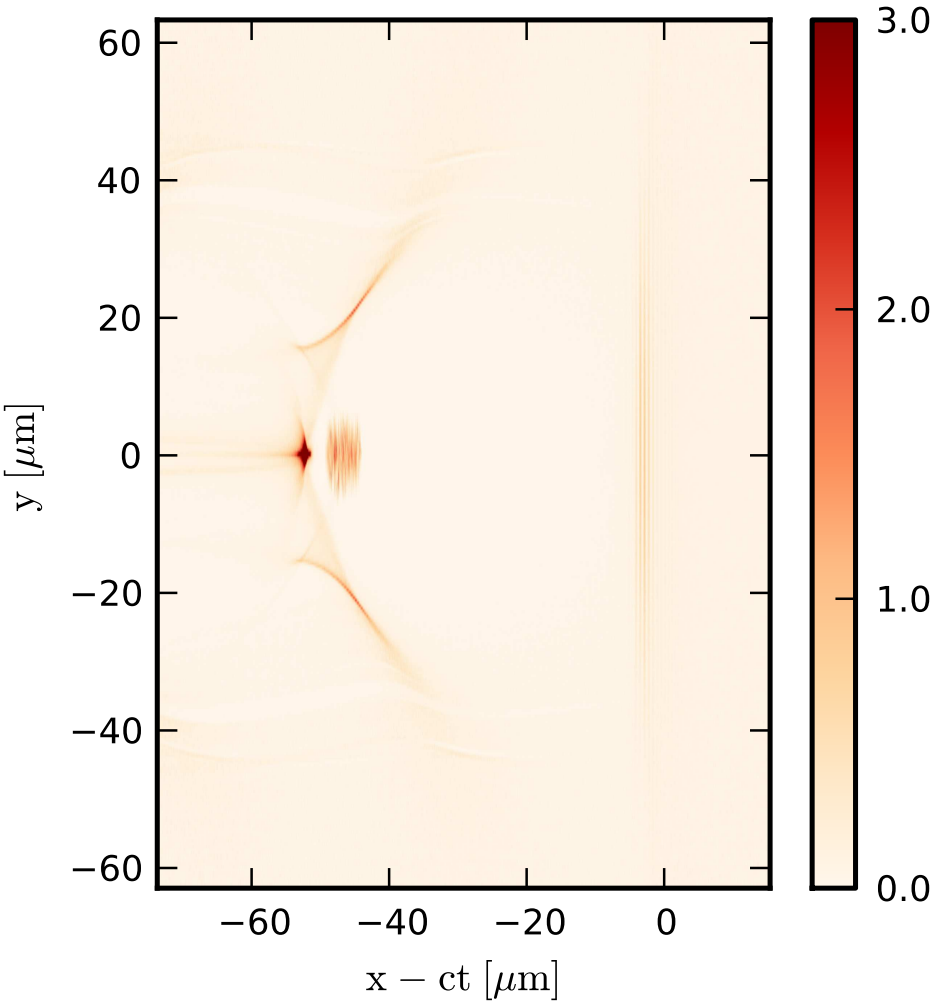

```
2Dar = rand((100,100))
imshow(2Dar,[options])
```



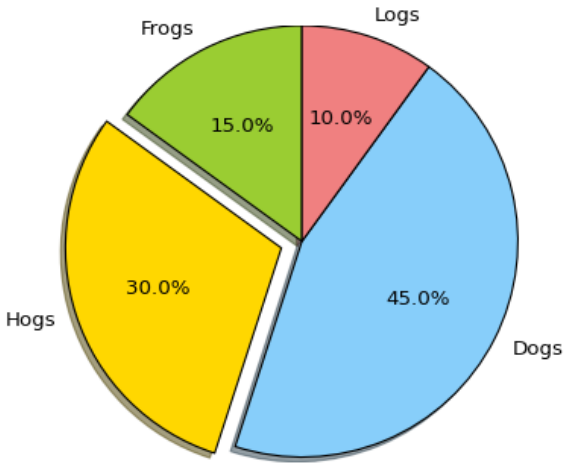

```
2Dar = rand((100,100))
pcolor(2Dar,[options])
```

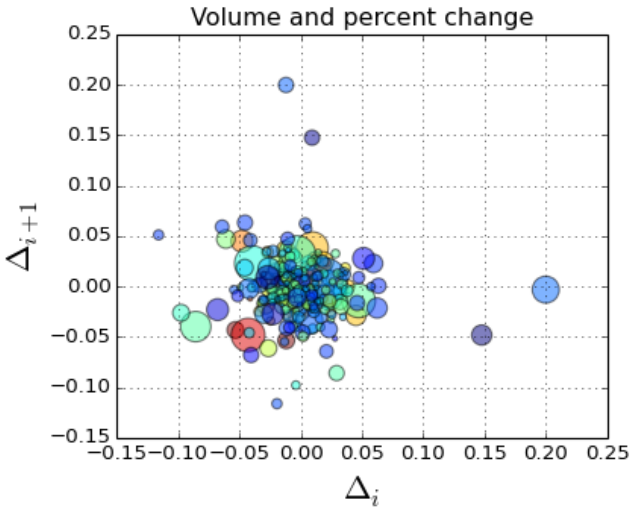


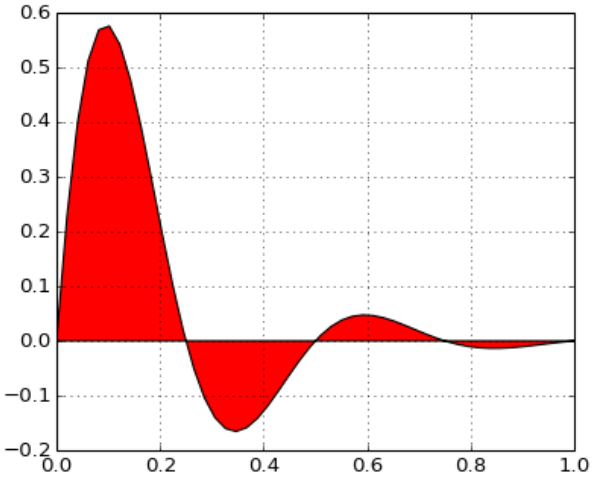
2D plots with a little bit of tuning

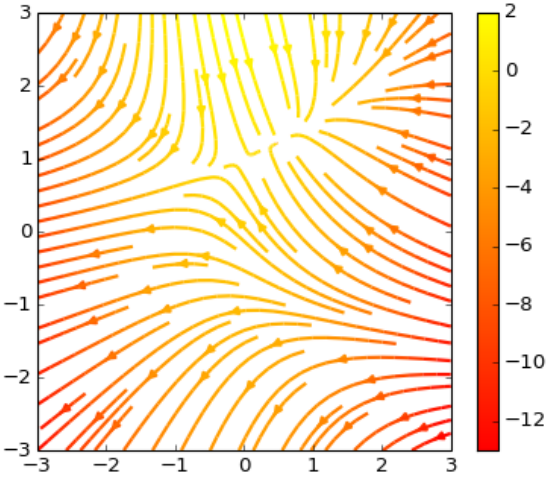


Other features of matplotlib









Matplotlib has native LaTeX rendering


```
label = r"$Math \LaTex code$"
```

Matplotlib's math rendering engine

$$W^{3\beta}_{\delta_1\rho_1\sigma_2} = U^{3\beta}_{\delta_1\rho_1} + \frac{1}{8\pi 2}\int_{\alpha_2}^{\alpha_2} d\alpha_2' \left[\frac{U^{2\beta}_{\delta_1\rho_1} - \alpha_2' U^{1\beta}_{\rho_1\sigma_2}}{U^{0\beta}_{\rho_1\sigma_2}}\right]$$

Subscripts and superscripts:

$\alpha_i > \beta_i,\ \alpha^j_{i+1} = \sin(2\pi f_j t_i) e^{-5 t_i/\tau},\ \ldots$

Fractions, binomials and stacked numbers:

$\frac{3}{4},\ \binom{3}{4},\ {}^{3}_{4},\ \left(\frac{5-\frac{1}{x}}{4}\right),\ \ldots$

Radicals:

$\sqrt{2},\ \sqrt[3]{x},\ \ldots$

Fonts:

Roman , *Italic* , Typewriter or $\mathcal{CALLIGRAPHY}$

Accents:

$\acute{a},\ \bar{a},\ \breve{a},\ \dot{a},\ \ddot{a},\ \grave{a},\ \hat{a},\ \tilde{a},\ \vec{a},\ \widehat{xyz},\ \widetilde{xyz},\ \ldots$

Greek, Hebrew:

$\alpha,\ \beta,\ \chi,\ \delta,\ \lambda,\ \mu,\ \Delta,\ \Gamma,\ \Omega,\ \Phi,\ \Pi,\ \Upsilon,\ \nabla,\ \aleph,\ \beth,\ \daleth,\ \gimel,\ \ldots$

Delimiters, functions and Symbols:

$\coprod,\ \int,\ \oint\ \prod,\ \sum,\ \log,\ \sin,\ \approx,\ \oplus,\ \star,\ \propto,\ \infty,\ \partial,\ \Re,\ \leftrightsquigarrow,\ \ldots$

The futur of visualization in Python

It is an extremely vast, active and changing domain.

New modules are emerging : Chaco, MayaVi, Bokeh, stressing interactivity and dynamic data visualizations in web browsers and in 3D.

What you saw today is extremely basic and is only a tiny part of what Python is capable of.