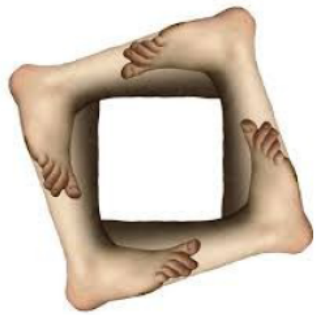
An introduction to python programming

Cecilia Jarne

cecilia.jarne@unq.edu.ar



About my Sci-Programming Working group

Pablo Alcain, Rodrigo Lugones, Graciela Molina, and I

WTPC
workshop en técnicas de programación científica

What is python?

Multi-paradigm language:
Structured, object oriented & functional styles are all supported.

- Paradigms is not enforced by language
- Clean syntax
- Highly extensible: small core, large standard lib

`https://www.python.org`

WTPC workshop en técnicas de programación científica

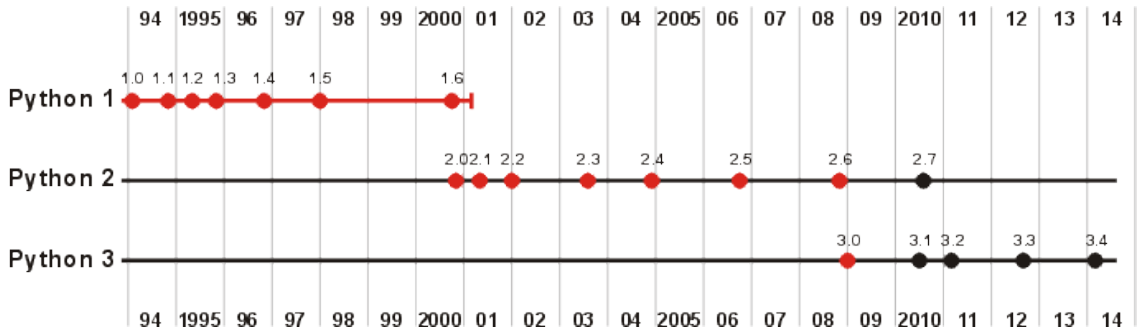
Why shoud I learn python if I know ***?

- Easy to learn
- Big community of developers
- Huge library
- Excellent science support!!
- Quick development turnaround
- Plot with python libraries is nice and easy
- Open source license

WTPC
workshop en técnicas de programación científica

A little of History

- Main development started 1989
- Author Guido van Rossum (if you want to follow him @gvanrossum)



On the design decisions

Zen of Python, by Tim Peters

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- If the implementation is hard to explain, it's a bad idea.

Use of the Interpreter

To start:

- Install the Python interpreter on your computer.
  (Linux distributions also frequently include Python and it is readily upgraded.)
- Call interpreter:

```
python
```

- It has an interative mode to introduce instructions one by one and see the result.

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

WTPC workshop en técnicas de programación científica

Basic ideas

- In a statically typed language, every variable name is bound both
  - to a type (at compile time, by means of a data declaration) to an object.
  - to an object.
- In a dynamically typed language, every variable name is (unless it is null) bound only to an object.

```
1 x = 1
2 x = "text" # dynamic typing :)
```

Data types

A list of the most common data types:

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

`https://docs.python.org/2/library/types.html`

WTPC workshop en técnicas de programación científica

Data types

A list of the most common data types:

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

`https://docs.python.org/2/library/types.html`

WTPC workshop en técnicas de programación científica

Python Syntax

In C or C++: use of ; at the end of statements

```
if(a>b)
    foo();
    bar();
baz();
```

In python: indentation level of your statements is significant! Last statement is executed out of the conditional

```
if(a>b):
    foo()
    bar()
baz()
```

WTPC workshop en técnicas de programación científica

Python Syntax

Control flow

```
for i in list:
    baz(i)
```

```
if a>b:
    foo()
elif b!=c:
    bar()
else:
    baz()
```

```
while a>b:
    foo()
    bar()
```

```
pass
```

```
break
continue
```

Python Syntax

Function definition:

```
1 def function(x,y,z):
2     x=3*y
3     return x+y-z
```

Also functions can be pass as values

```
1 def func(x, p1,p2):
2     return p1*(x)+p2
```

Python Syntax example

In C indentation is optional:

```c
int factorial(int x)
{
if (x == 0)
return 1;
else
return x * factorial(x - 1);
}
```

In python is mandatory:

```python
def factorial(x):
    if x == 0:
        return 1
    else:
        return x * factorial(x - 1)
```

WTPC Workshop en técnicas de programación científica

Python Boolean and operators

Math operators:

Name	Function	symbol
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20

Boolean:

Operation	Result
x or y	if x is false, then y, else x
x and y	if x is false, then x, else y
not x	if x is false, then True, else False

Strings

It is possible to use ' or " on string definition

```
1  a= "I am Cecilia"
2  b= 'I am from Argentina'
```

To print on screen we use the print function:

```
print ('---All rigth!!!----')
```

Strings examples

An example:

```
'I ate %d %s today' %(12,'apples')
```

other example: I ate today.format(12,'apples')

structures

- List

```
1 a=[1,'apple',1.2]
```

- Tuple

```
1 a=(1,'apple',1.2)
```

- Dict

```
1 a={'name':'Giovanni', 'age':42}
```

- Set

```
1 a={1,'apple',1.2}
```

Lists

- Very useful structure on python
- Index and slice access

```
a[0] a[1] a[2:5] a[2:10:2]
```

- Comprehension list

```
[x**2 for x in range(1,11)]
```

List Methods

```
list.append(obj)
```

Appends object obj to list

```
list.count(obj)
```

Returns count of how many times obj occurs in list

```
list.extend(seq)
```

Appends the contents of seq to list

```
list.index(obj)
```

Returns the lowest index in list that obj appears

```
list.insert(index, obj)
```

Inserts object obj into list at offset index

WTPC workshop en técnicas de programación científica

List Methods

```
list.pop(obj=list[-1])
```

Removes and returns last object or obj from list

```
list.remove(obj)
```

Removes object obj from list

```
list.reverse()
```

Reverses objects of list in place

Main reason to use python in scientific programing

Three libraries to work with:

- NumPy:
  `http://www.numpy.org/`
- SciPy:
  `http://www.scipy.org/`
- MatPlotLib:
  `http://matplotlib.org/`

All are open source!

WTPC workshop en técnicas de programación científica

Main reason to use python in scientific programing

- NumPy provides functionality to create, delete, manage and operate on large arrays of type "raw" data (like Fortran and C/C++ arrays)
- SciPy extends NumPy with a collection of useful algorithms like minimization, Fourier transforms, regression and many other applied mathematical techniques
- Both packages are add-on packages (not part of the Python standard library) containing Python code and compiled with (fftpack, BLAS)
- MatPlotLib is a nice library to plot

WTPC workshop en técnicas de programación científica

Numpy

NumPy is the fundamental package for scientific computing in Python.

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities
- Operations on matrices and vectors in NumPy are very efficient because they are linked to compiled in BLAS/LAPACK code

WTPC workshop en técnicas de programación científica

Scipy

One of the core packages that make up the SciPy stack. SciPy is built on top of NumPy and implements many specialized scientific computation tools:

- Manly user-friendly
- Efficient numerical routines such as routines for numerical integration and optimization.

  - Clustering,
  - Fourier transforms
  - numerical integration, interpolations
  - data I/O, LAPACK
  - sparse matrices, linear solvers, optimization
  - signal processing
  - statistical functions

WTPC workshop en técnicas de programación científica

Matplotlib

Is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Can be used in:

- Python scripts
- The Python and IPython shell
- The jupyter notebook
- Web application servers
- Graphical user interface toolkits

How to start?

It is necesary to import libraries:

```
1 import numpy
2 import scipy
3 import matplotlib.pyplot
```

There are different way to import libraries:

```
1 import numpy
2 import numpy as np
3 from numpy import *
```

I use:

```
1 import numpy as np
2 import scipy
3 import matplotlib.pyplot as pp
```

WTPC Workshop en técnicas de programación científica

Things from Numpy

NumPy functionality:

- Polynomial mathematics
- Statistical computations
- Pseudo random number generators
- Discrete Fourier transforms
- Size / shape / type testing of arrays

Fundamental things from Numpy: np.array

There are 5 general mechanisms for creating arrays:

- Conversion from other Python structures (e.g., lists, tuples)
- Intrinsic numpy array array creation objects (e.g., arange, ones, zeros, etc.)
- Reading arrays from disk, either from standard or custom formats
- Creating arrays from raw bytes through the use of strings or buffers
- Use of special library functions (e.g., random)

WTPC workshop en técnicas de programación científica

Numpy array examples

(>>> means input)

```
mport numpy as np
>>> x = np.array([2, 3, 1, 0])
>>> print x
[2 3 1 0]

>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> np.arange(2, 10, dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8., 9.])

>>> np.arange(2, 3, 0.1)
array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9])

>>> np.linspace(1., 4., 6)
array([ 1. ,  1.6,  2.2,  2.8,  3.4,  4. ])
```

WTPC Workshop en técnicas de programación científica

Changing array shape

```
>>> a = np.zeros((5, 2))
>>> print a
[[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
```

reshape:

```
>>> b = a.reshape((2, 5))
>>> print b
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

Array transpose and Fill:

Some routine examples:

```
>>> x = np.array([1.,2.,3.,4.])
>>> x
array([ 1.,  2.,  3.,  4.])
>>> x.T
array([ 1.,  2.,  3.,  4.])
```

```
>>> a = np.array([1, 2])
>>> a.fill(0)
>>> a
array([0, 0])
>>> a = np.empty(2)
>>> a.fill(1)
>>> a
array([ 1.,  1.])
```

WTPC workshop en técnicas de programación científica

Some elemental operations with Numpy:

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array

>>>print(x + y)
>>>print(np.add(x, y))
 [[ 6.0  8.0]
 [10.0 12.0]]

# Elementwise difference; both produce the array

>>>print(x - y)
>>>print(np.subtract(x, y))
 [[-4.0 -4.0]
 [-4.0 -4.0]]
```

WTPC Workshop en técnicas de programación científica

Some elemental operations with Numpy:

```
# Elementwise product; both produce the array
>>>print(x * y)
>>>print(np.multiply(x, y))
[[ 5.0 12.0]
[21.0 32.0]]

# Elementwise division; both produce the array
>>>print(x / y)
>>>print(np.divide(x, y))
[[ 0.2         0.33333333]
[ 0.42857143  0.5       ]]

# Elementwise square root; produces the array
>>>print(np.sqrt(x))
[[ 1.          1.41421356]
[ 1.73205081  2.        ]]
```

More operations with arrays

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array

print(x.dot(y))
print(np.dot(x, y))
```

WTPC
Workshop en técnicas de programación científica

On array creation

```
a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"
b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"
c = np.full((2,2), 7) # Create a constant array
print(c)              # Prints "[[ 7.  7.]
                      #          [ 7.  7.]]"
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"
e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #              [ 0.68744134  0.87236687]]"
```

How to create a data set, plot and save in txt

```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open('tabla.txt', 'w')
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ('Vec: ',Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color='r',label='Sin vs x')
pp.xlabel('Time (s)')
pp.savefig("fig_01.jpg",dpi=200)
```

How to create a data set, plot and save in txt

```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open('tabla.txt', 'w')
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ('Vec: ',Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color='r',label='Sin vs x')
pp.xlabel('Time (s)')
pp.savefig("fig_01.jpg",dpi=200)
```

How to create a data set, plot and save in txt

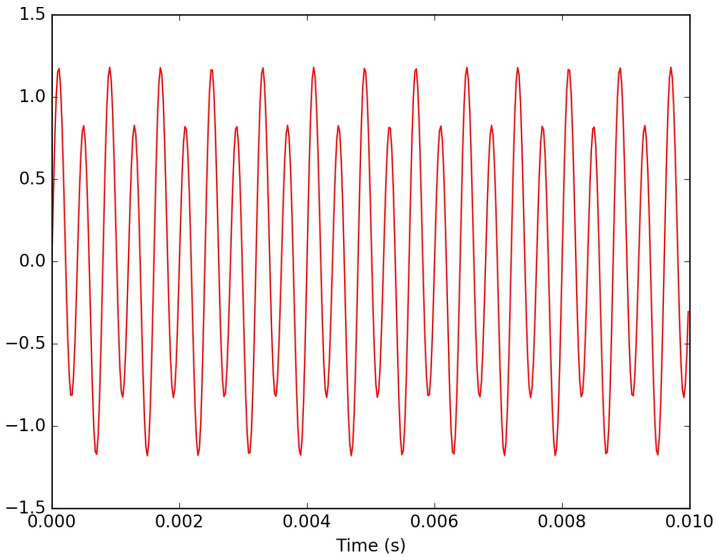
```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open('tabla.txt', 'w')
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ('Vec: ',Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color='r',label='Sin vs x')
pp.xlabel('Time (s)')
pp.savefig("fig_01.jpg",dpi=200)
```

How to create a data set, plot and save in txt

```
from math import *
import numpy as np
import matplotlib.pyplot as pp
#File creation
f_out_max = open('tabla.txt', 'w')
#data generation
x    = np.arange(441)
Sin1 = 1*np.sin(2*pi*(25/441.0)*x)
Sin2 = 0.25*np.sin(2*pi*((25./2)/441.0)*x)
Sin  = Sin1+Sin2
Vec  = np.c_[x,Sin]
#printing
print ('Vec: ',Vec.shape)
#save in .txt file
np.savetxt(f_out_max,Vec,fmt='%f',delimiter='\t',header="x #f(x)")
f_out_max.close()
#plot figure
pp.figure()
pp.plot(x*1./44100.,Sin, color='r',label='Sin vs x')
pp.xlabel('Time (s)')
pp.savefig("fig_01.jpg",dpi=200)
```

We obtain:

A basic figure and table in a txt file:



# x	#f(x)
0.000000	0.000000
1.000000	0.392994
2.000000	0.740813
3.000000	1.003819
4.000000	1.152764
5.000000	1.172342
6.000000	1.062998
7.000000	0.840786
8.000000	0.535279
9.000000	0.185801
10.000000	-0.163538
11.000000	-0.469389
12.000000	-0.694586
13.000000	-0.812779
14.000000	-0.811673
15.000000	-0.694499
16.000000	-0.479506
17.000000	-0.197559
18.000000	0.111860 ...

WTPC workshop en técnicas de programación científica

How to open a table in .txt or cvs

A very easy way is using numpy:

```
1  import numpy as np
2
3  file_name_you_want        = np.loadtxt(fname,delimiter=" ")
4
5  print "First column element: ",file_name_you_want[0]
6  #to get the full column:
7
8  Transpose_your_file       = file_name_you_want.T
9
10 print "First column: ", Transpose_your_file[0]
```

How to perform a simple Fit and nicely plot it

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

How to perform a simple Fit and nicely plot it

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

How to perform a simple Fit and nicely plot it

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

How to perform a simple Fit and nicely plot it

```
1  import numpy as np
2  import matplotlib.pyplot as pp
3  from scipy.optimize import curve_fit
4
5  def fitFunc(t, a, b, c):
6      return a*np.exp(-b*t) + c
7
8  t                          = np.linspace(0,4,50)
9  temp                       = fitFunc(t, 2.5, 1.3, 0.5)
10 noisy                      = temp + 0.25*np.random.normal(size=len(temp))
11 fitParams, fitCovariances = curve_fit(fitFunc, t, noisy)
12
13 pp.figure(figsize=(12, 6))
14 pp.ylabel('Temperature (C)', fontsize = 16)
15 pp.xlabel('time (s)', fontsize = 16)
16 pp.xlim(0,4.1)
17 pp.errorbar(t, noisy, fmt = 'ro', yerr = 0.2)
18 sigma = [fitCovariances[0,0], fitCovariances[1,1], fitCovariances[2,2] ]
19 pp.plot(t, fitFunc(t, fitParams[0], fitParams[1], fitParams[2]))
20 pp.plot(t, fitFunc(t, fitParams[0] + sigma[0], fitParams[1] - sigma[1], fitParams[2] + sigma[2]))
21 pp.plot(t, fitFunc(t, fitParams[0] - sigma[0], fitParams[1] + sigma[1], fitParams[2] - sigma[2]))
22 pp.savefig('dataFitted.pdf', bbox_inches=0, dpi=600)
23 pp.show()
```

WTPC Workshop en técnicas de programación científica

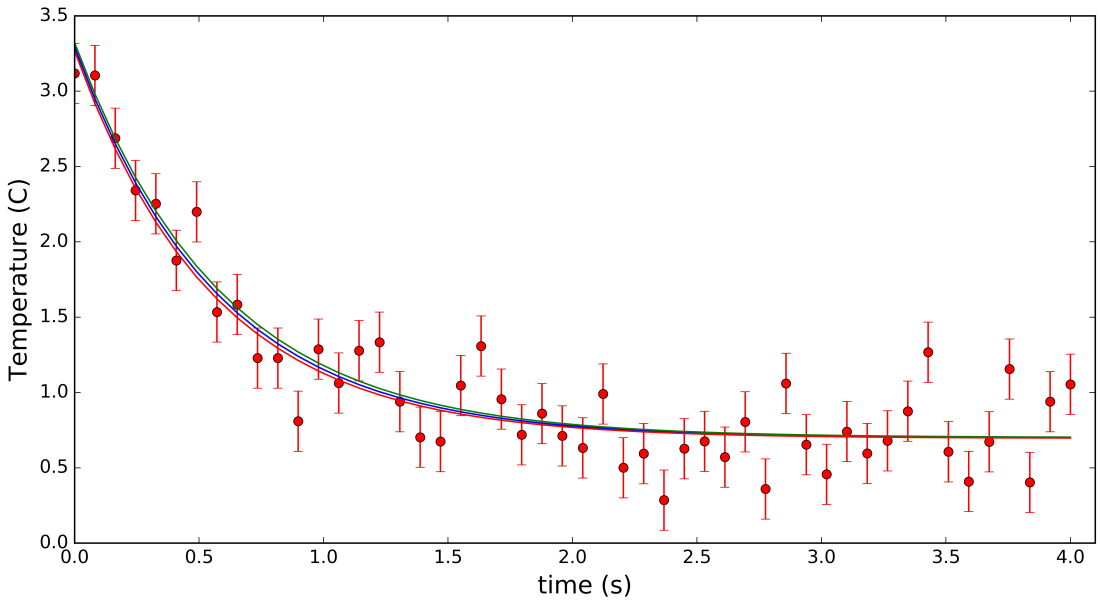
How to perform a simple Fit

We obtain the fit parameters:

```
[ 2.595658    1.74438726  0.69809511]
```

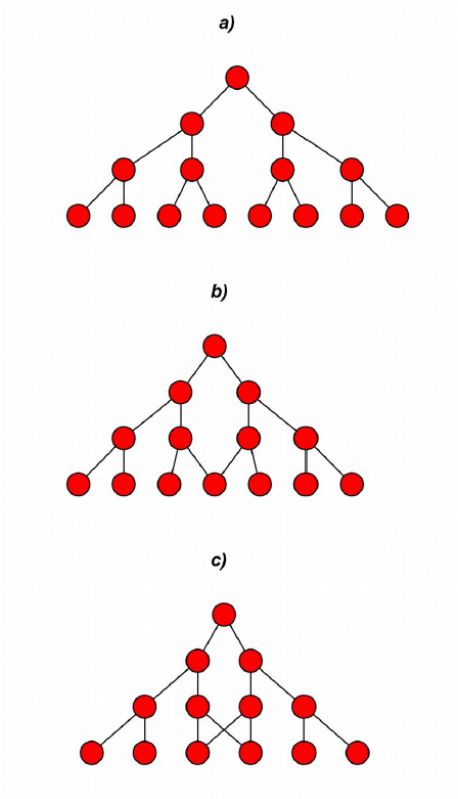
The covariance matrix:

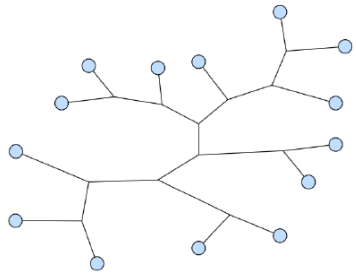
```
[[ 0.02506636  0.01490486 -0.00068609]
 [ 0.01490486  0.04178044  0.00641246]
 [-0.00068609  0.00641246  0.00257799]]
```



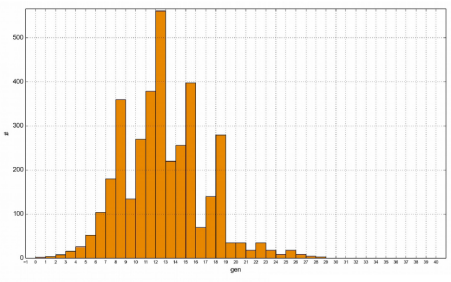
Other more advance kind of plots

```
1 import networkx as nx
2 import pygraphviz
```



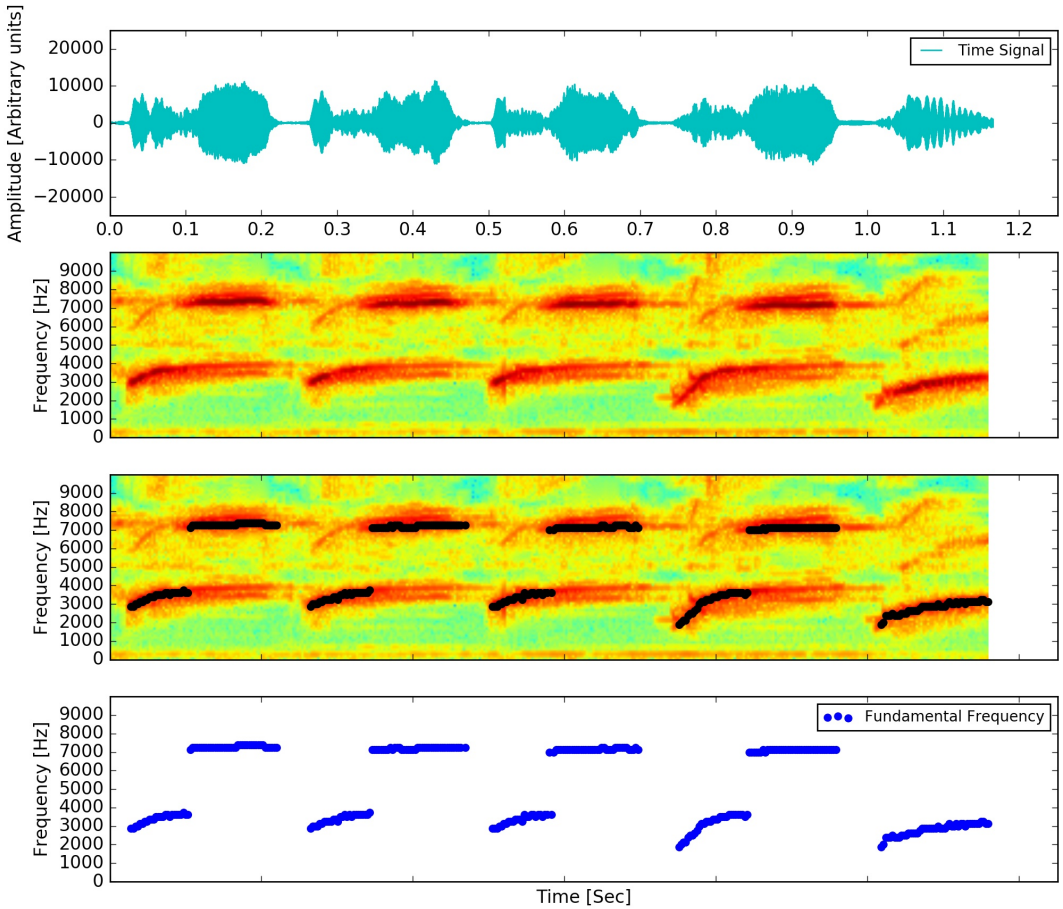


```
1 import matplotlib.pyplot as pp
2 pp.hist([vector,bins=bins)
```



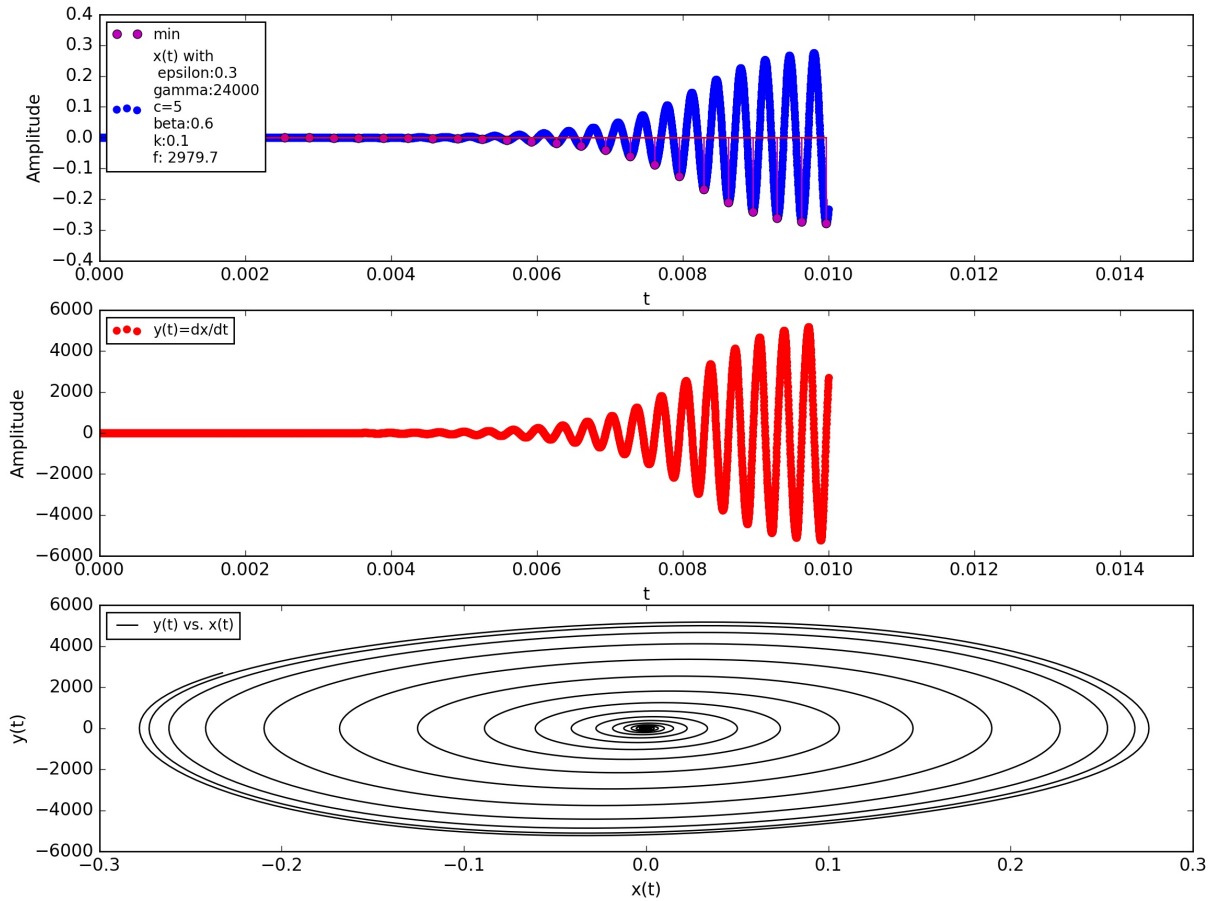
Fourier transform and Spectrograms

```
1 import matplotlib.pyplot as pp
2 pp.specgram(signal, NFFT=nfft, Fs= sample_rate, noverlap=par, cmap='jet'
      )
```



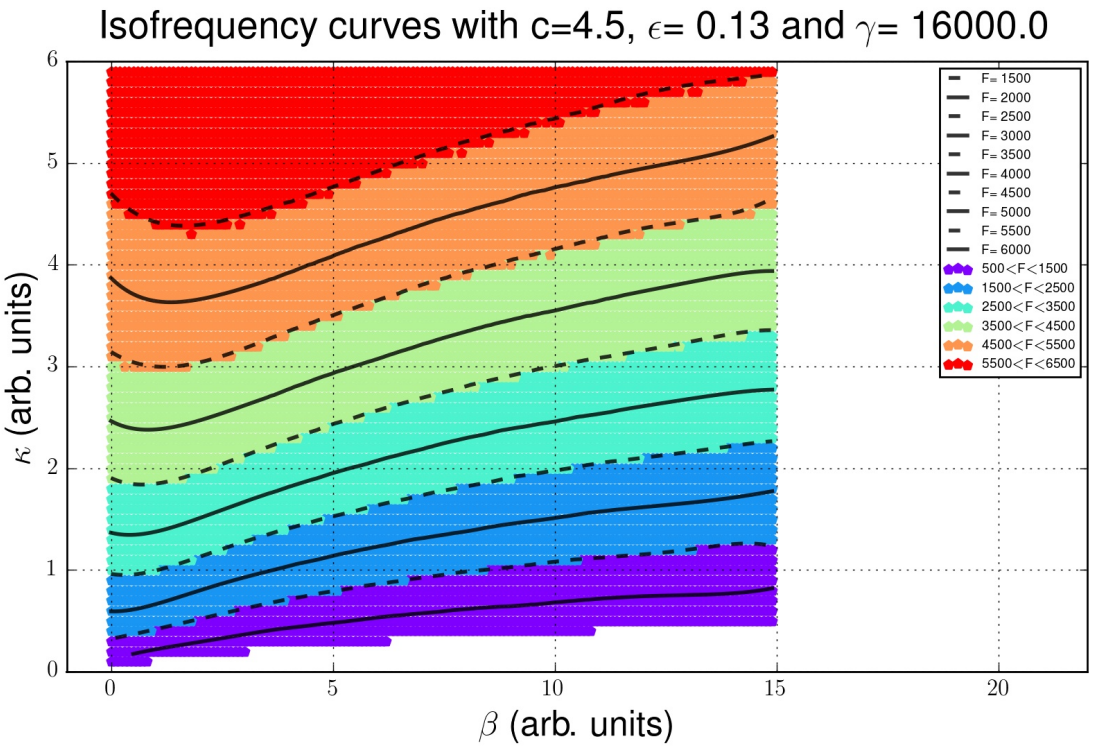
Numerical integration

```
from scipy.integrate import odeint
dy/dt = func(y, t0, ...)
sol   = odeint(func, X0, t)
```



Scatter plot of multiple numerical integrations

- Each point represent the value of the parameters in a numerical integration of the previous equation.
- Colors represent frequency regions



Conclusions

- Python is a very versatile language to write scripts or complex programs combining C, or Fortran
- Scientific libraries with a lot of examples
- Possibility of re-use and improve existing code
- A guide of style in: `https://google.github.io/styleguide/pyguide.html`
- Matplotlib examples: `http://matplotlib.org/gallery.html`

WTPC workshop en técnicas de programación científica

Additional libraries for data analysis

- Scikit learn:
  `http://scikit-learn.org`
- Pandas
  `http://pandas.pydata.org/`

Additional libraries for data analysis