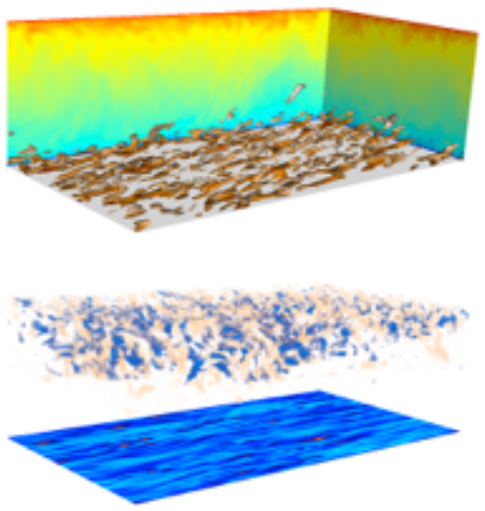
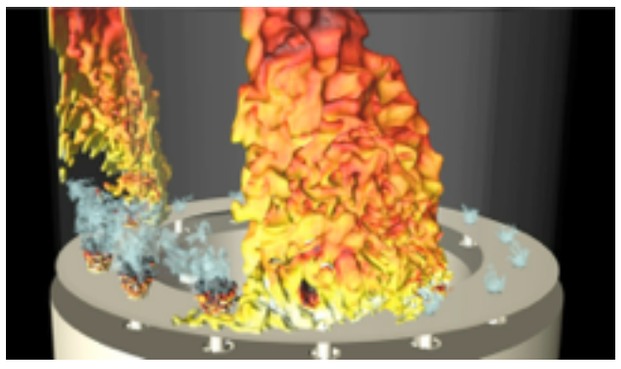
Numerical Methods in Engineering Applications

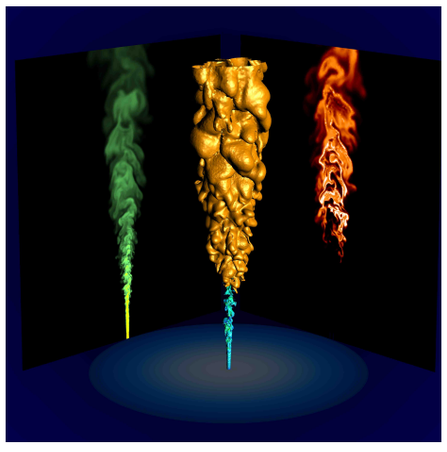
Workshop #01-1

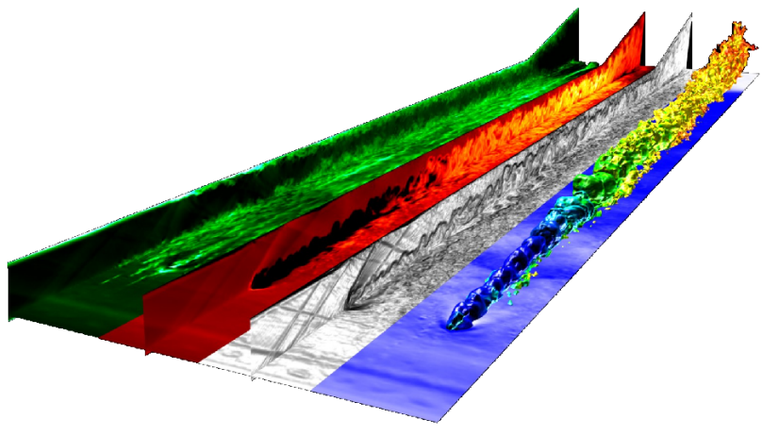
Introduction to Python

ronan.vicquelin@centralesupelec.fr
aymeric.vie@centralesupelec.fr
nicolas.dumont@centralesupelec.fr
leo.cunha@centralesupelec.fr









Objectives of Workshop #1

- **Python**
  - Functions and Modules
  - Control statements IF, FOR and WHILE
  - Libraries
  - Plotting in 2D and 3D
  - Input and Output

- **Miscellaneous algorithms**
  - Basic quadrature rules
  - Root-Finding algorithms

Why Python?

performance

Machine

Assembly

```
0x52ac7c:  movl   %eax, -20(%ebp)
0x52ac7f:  movl   $0, (%edi,%eax)
0x52ac86:  testl  %esi, %esi
0x52ac88:  je     0x52ad21
    [UINavigationController _updateScroll
    toViewController:] + 425
0x52ac8e:  movl   7306542(%ebx), %eax
0x52ac94:  movl   (%edi,%eax), %eax
0x52ac97:  movl   %eax, -24(%ebp)
0x52ac9a:  movl   7212558(%ebx), %eax
0x52aca0:  movl   %eax, 4(%esp)
0x52aca4:  movl   %esi, (%esp)
0x52aca7:  calll  0x9bff06
    objc_msgSend
0x52acac:  movl   %eax, -28(%ebp)
0x52acaf:  movl   %edx, -32(%ebp)
0x52acb2:  movl   7211062(%ebx), %eax
0x52acb8:  movl   %eax, 4(%esp)
```

Low-level
C, C++
Fortran

```
program HelloWorld
  write (*,*) 'Hello, world!'
end program HelloWorld
```

High-level
Matlab
Python

ease of use and interactivity

- **Free**
- **High-level language**
  - High interactivity with datas
- **Widely used in the scientific community**
  - Many scientific libraries available
- **Modules**

Developing Environment for python

Other modules/ packages/frameworks…

IDE's
Spyder, Eclipse

Editor
Gedit, Notepad, Xcode, …

Interactive interpreter
ipython

Base interpreter
python

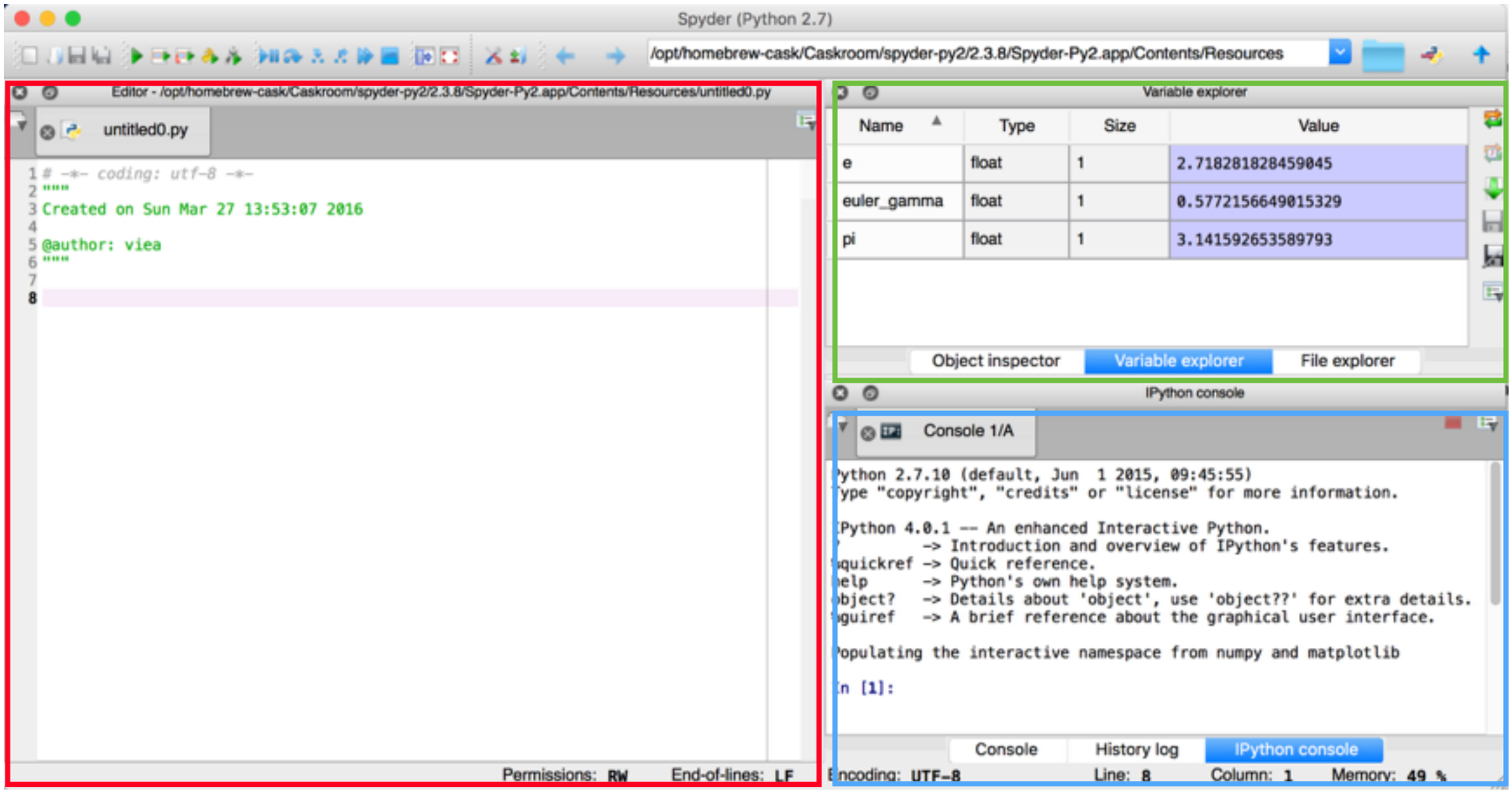
Scientific libraries
numpy,scipy, matplotlib…

Standard library

Anaconda or Winpython distributions

Spyder Integrated Development Environment

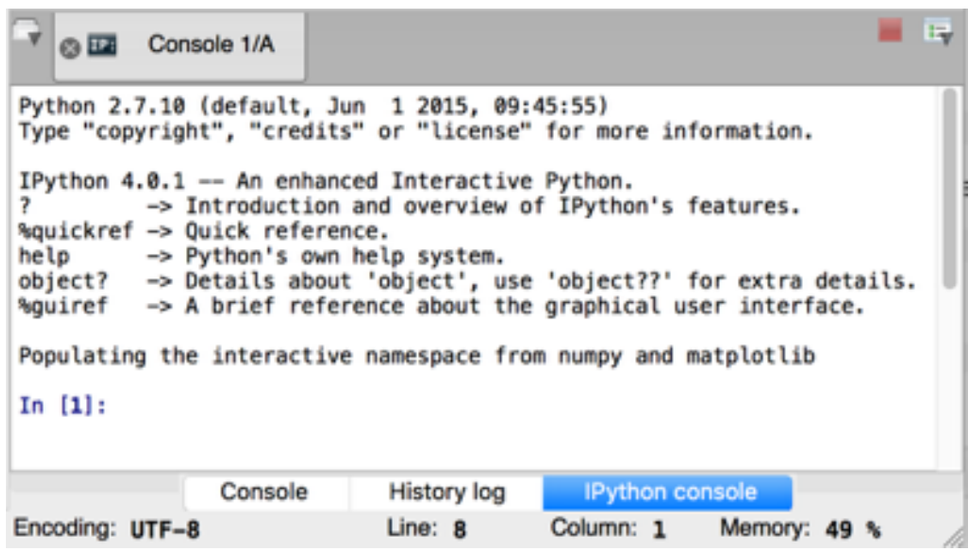
- **"Matlab-like"**
  - Editor
  - Command windows (python or Ipython)
  - Variable explorer



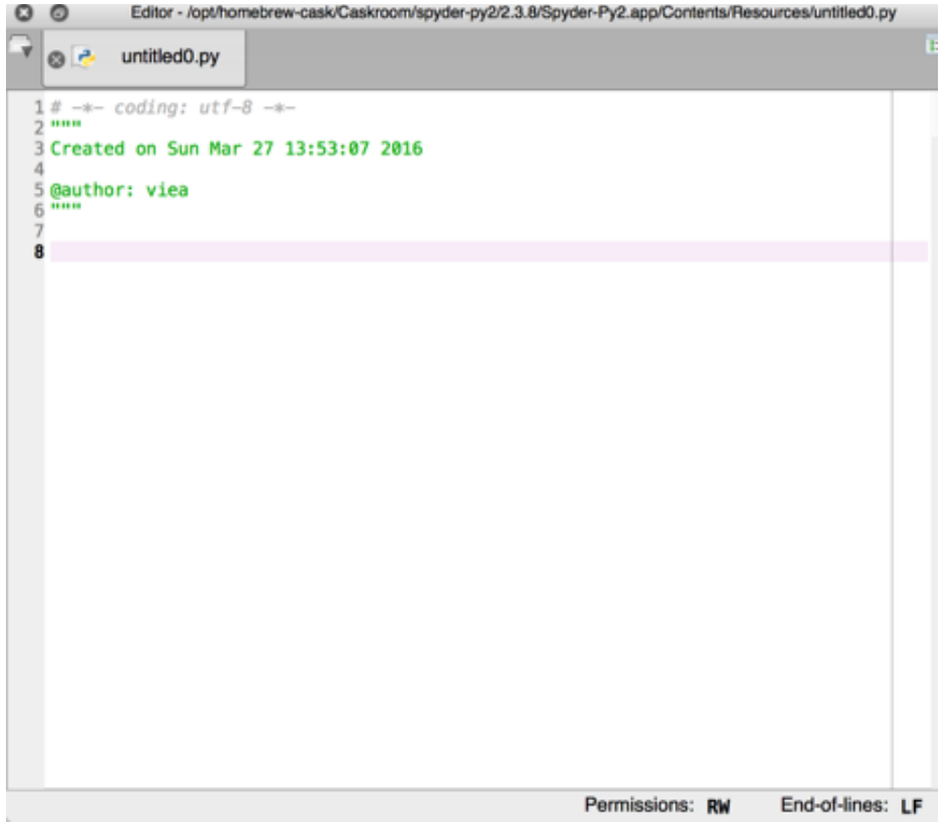
Execution of Python's code

- **Inline commands**

- **Script execution**
  - script.py files
  - blocks are identified using indentation



- **Functions & Modules**
  - functions can be defined using def argument with proper block indentation
  - Stored in the main script or in an external module



- **Comments**
  - Comments begin by #
  - multiline comment are enclose between triple quotes ”’

Basic operations in Python

- **Arithmetic operations**
  - sum +,substration -
  - multiplication *, division /
  - exponent **

```
In  [1]:  5+4
Out [1]:  9
In  [2]:  5**2
Out [2]:  25
```

- **Data types and assignments**
  - Numeric types (int, float, complex)
  - Sequences (str, byte, list, tuples)

```
In  [1]:  x=5
In  [2]:  x
Out [2]:  5
```

- **List and tuples**
  - Container for multiple objects of various types
  - Tuples are immutable lists

```
In  [1]:  list=[5,4,3,'char']
In  [2]:  list
Out [2]:  [5, 4, 3, 'char']
In  [3]:  list[0]
Out [3]:  5
```

Control statements

- **“if” and “while” statements**
  - if, elif, else
  - blocks identified by indentation
  - statements terminate with :

```
if x == 5:
    print ‘X=5’
elif x == 4:
    print ‘X=4’
else:
    print ‘something else’
```

- **“for” statement**
  - based on a given sequence
  - iterate using the sequence order
  - function range can generate the adequate sequence

```
for x in range(5,10):
    x=x+1
```

How to use modules

- **import module**
  - load definitions and statements inside module
  - definitions in module are called using "module.definition"

module.py

```
def definition():
    print 'do stuff'
```

script.py

```
import module


module.definition()
```

- **from module import ***
  - load definitions and inside module
  - but do not require "module." at call

```
from module import *


definition()
```

- **import module as md**
  - rename the module prefix as md

```
import module as md


md.definition()
```

Scientific computing in Python

- **Unlike Matlab, many basic operations are initially missing**
  - cosinus, sinus, array, matrix algebra, plotting,

- **Use of scientific libraries**
  - Enable high level functionality for scientific computing

- **Math**
  - cos, sin, tan, …

- **Numpy**
  - fast, compact, multi-dimensional array facility
- **SciPy**
  - integration, ODE solvers, optimisation, parallel computing
- **Matplotlib**
  - Advanced plotting

Scientific computing in Python

- **Unlike Matlab, many basic operations are initially missing**
  - cosinus, sinus, array, matrix algebra, plotting,

- **Use of scientific libraries**
  - Enable high level f… scientific c…
  - …
  - …, …
- **Numpy**
  - fast, compact, multi-dimensional array facility
- **SciPy**
  - integration, ODE solvers, optimisation, parallel computing
- **Matplotlib**
  - Advanced plotting

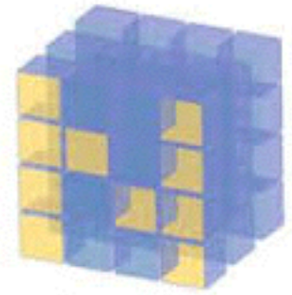
All loaded with
from pylab import *

SciPy.org Sponsored By ENTHOUGHT

NumPy

matplotlib

Numpy: multi-dimensional arrays



- **Why not using list or tuples?**
  - Numpy arrays are homogeneous data sets
  - Array content occupies a contiguous memory space
  - Arithmetic and vectorized operations can be performed on it

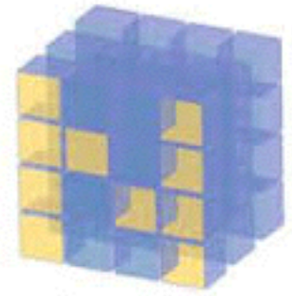
- **Vectors**
  - np.array(object,dtype)
  - object= list of the vector content
  - dtype= data type

- **Multi dimensional arrays**
  - np.ndarray(shape, dtype, order)
  - shape= dimension in each direction
  - type = data type
  - order = memory ordering (fortran or C style)

- **Access to data by position between brackets**
  x[0], x[-1], y[1,2], y[0,:], y[:,0], y[:,:]

```
import numpy as np

>>> x=np.array([1,2,3,4])
>>> x
array([1,2,3,4])
>>> x+1
array([2,3,4,5])

>>>y=np.ndarray([2,2],int)
>>>y
array([[0,0],
       [0,0]])
>>>y[0,0]=1
>>>y
array([[1,0],
       [0,0]])
```

Numpy: multi-dimensional arrays



- **Size of array**
  - np.shape(array)
    - return size in each dimension
  - np.size(array)
    - return full dimension

- **Array constructors**
  - x = np.zeros(n)
    y = np.zeros((n1,n2))
    - Vector of size n and n1xn2 matrix filled with zeros
  - x = np.arange(start,stop,step)
    - exclude end point

  - x = np.linspace(start,stop,npoints)
    - include end point

  - x, y = np.mgrid[x1:x2:dx,y1:y2:dy]
    - equivalent to meshgrid in matlab

  - Others: ones, identity,

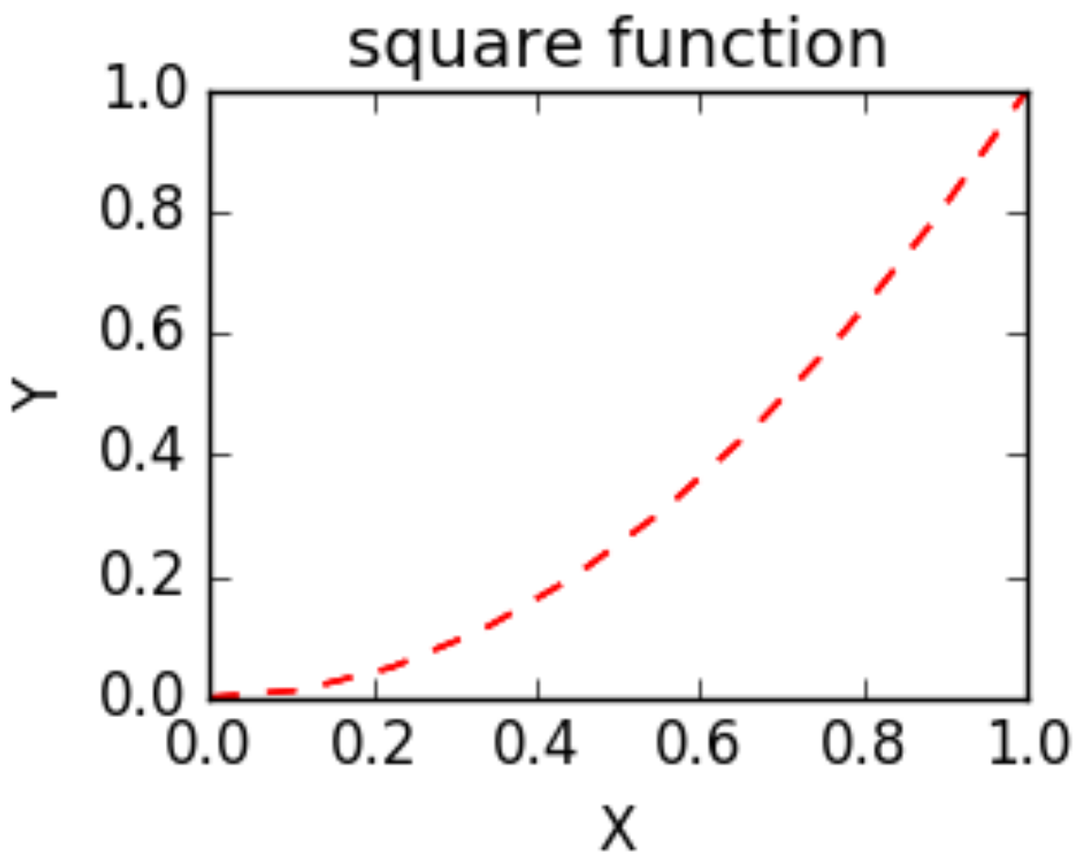
```
import numpy as np

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

>>>y=np.linspace(0.,1.,3)
>>>y
array([[0, 0.5, 1])

>>>x, y =np.mgrid[0:2,0:2]
>>>x
array([[0,0],
       [1,1]])
>>>y
array([[0,1],
       [0,1]])
```

Matplotlib: Advanced plotting

- **Matplotlib capacities**
  - 2D and 3D plot
  - surface and volume rendering
  - figure and plot/subplot handling

```
x = linspace(0,1,10)
y = x**2
plot(x,y)
xlabel('X')
ylabel('Y')
title(' square function')
```

- **Matlab-like implementation**
  - in Ipython, type %pylab
  - 2D line plots
    - plot(x,y)
    - xlabel,ylabel for axis labels
    - title for plot title



Matplotlib: Advanced plotting

- **Matplotlib object-oriented**
  - generate objects for each element
  - act on object properties
  - from matplotlib.pyplot import plt

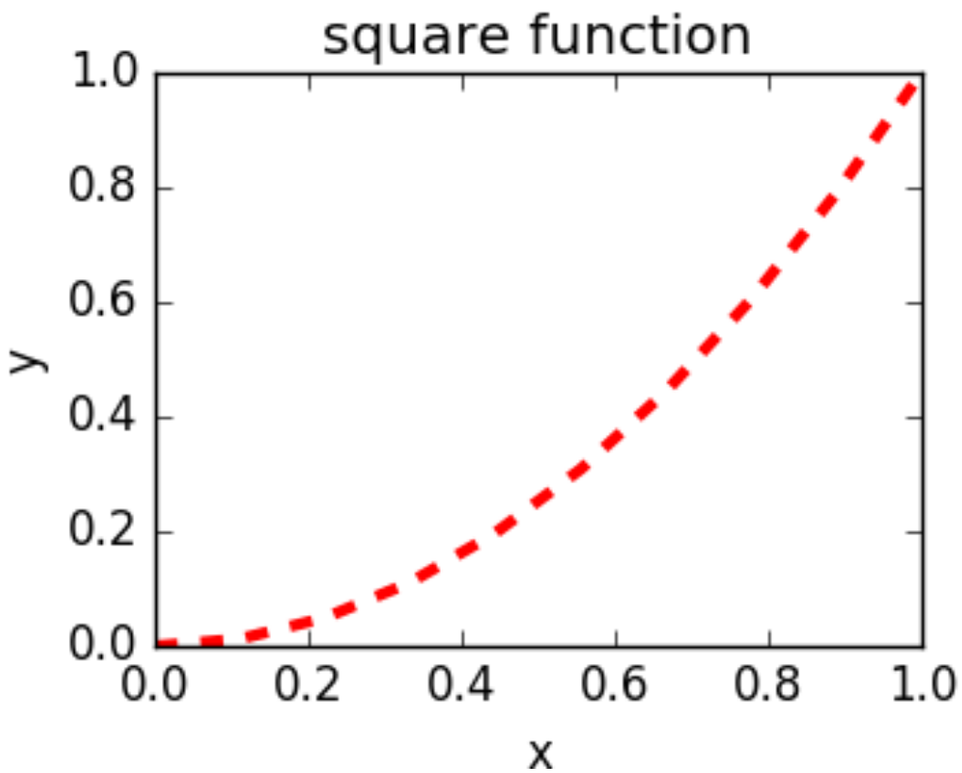
- **Figure object**
  - fig=plt.figure(property=value)

- **Axes object**
  - add axes object to figure object
  - axes=fig.add_axes([lower,bottom,width, height])

- **Plot object**
  - add plot object to axes object
  - pl=axes.plot(x,y,'r')
  - plt.setp(pl,'property',value)

- **Save Figure**
  - fig.savefig("toto.pdf", format="pdf")

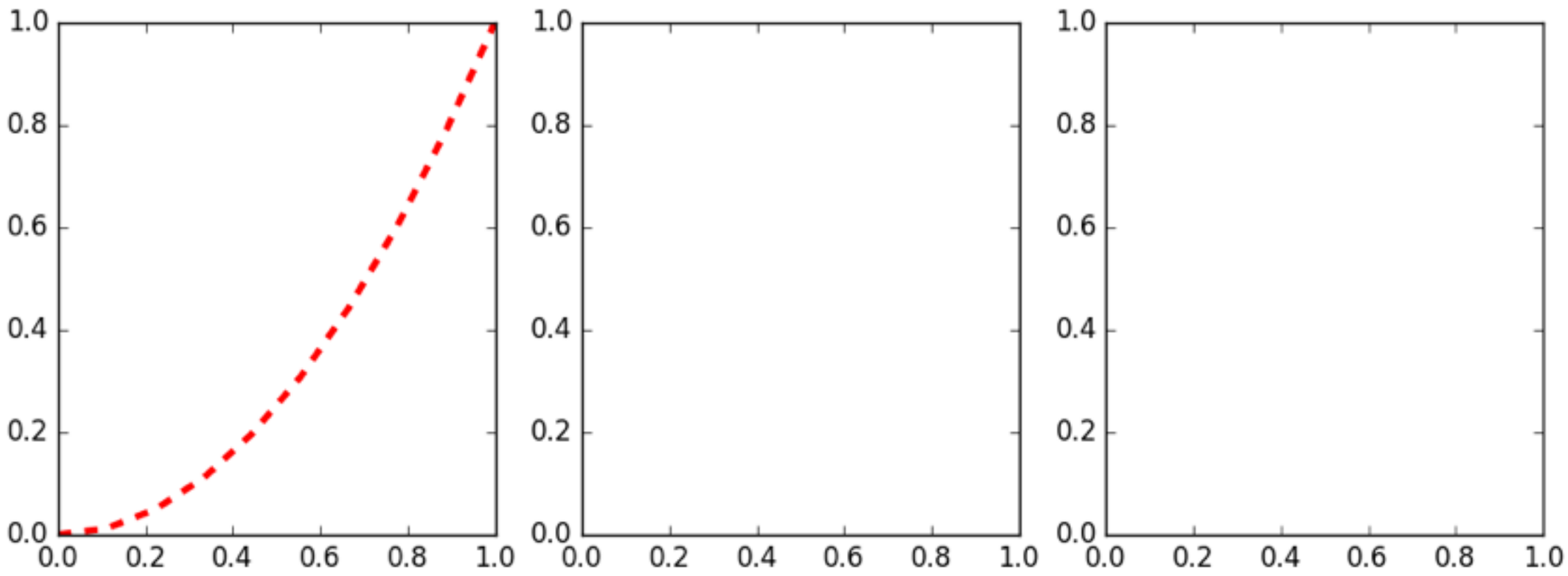
```
import matplotlib as plt
#figure
fig=plt.figure(figsize=(4, 3))
#axes
ax=fig.add_axes([0.15,0.15,0.7,0.7])
#plot
pl=axes.plot(x,y,'r—')
plt.setp(pl,'linewidth',3)
```



Matplotlib: Advanced plotting

- **Subplots**
  - fig, axes=subplots(nrow,ncol,figsize)
  - axes' length is nrow*ncol
    OR
  - fig= plt.figure()
    axes = fig.add_subplot(nrow,ncol,index_fig)

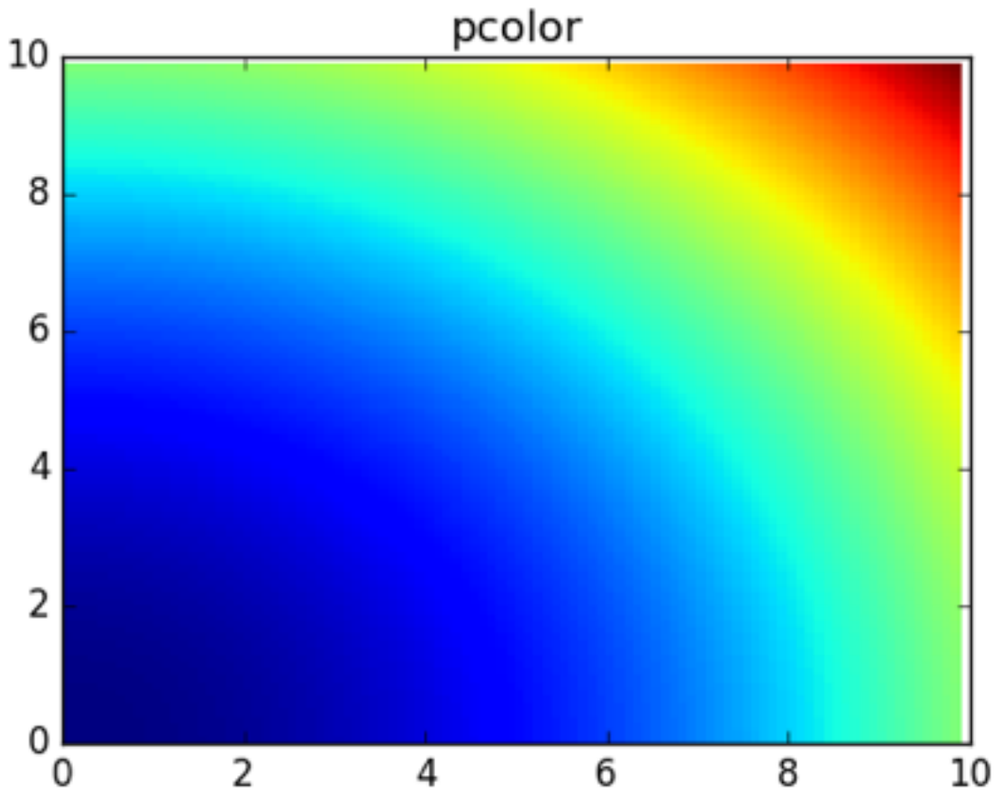
```
#figure and axes
fig,axes=subplots(1,3,figsize=(12,4))
#plot
pl=axes[0].plot(x,y,'r—')
plt.setp(pl,'linewidth',3)
```

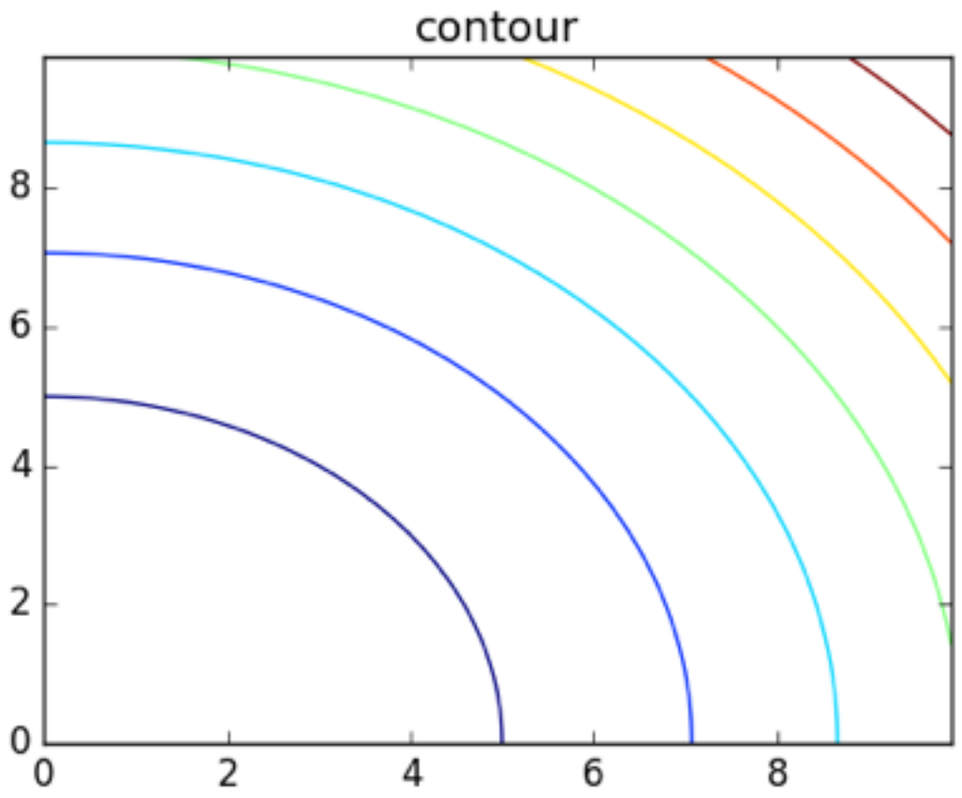


Matplotlib: Advanced plotting

- **Color and contour plots**
  - pcolor for continuous colormaps
  - contour for iso contour
  - need matrix inputs

```
x, y = mgrid[0:10:0.1,0:10:0.1]
f=x**2+y**2
#figure and axes
fig,axes=subplots(1,2,figsize=(12,4))
#plots
axes[0].pcolor(x,y,f)
axes[0].set_title('pcolor')
axes[0].contour(x,y,f)
axes[0].set_title('contour')
```

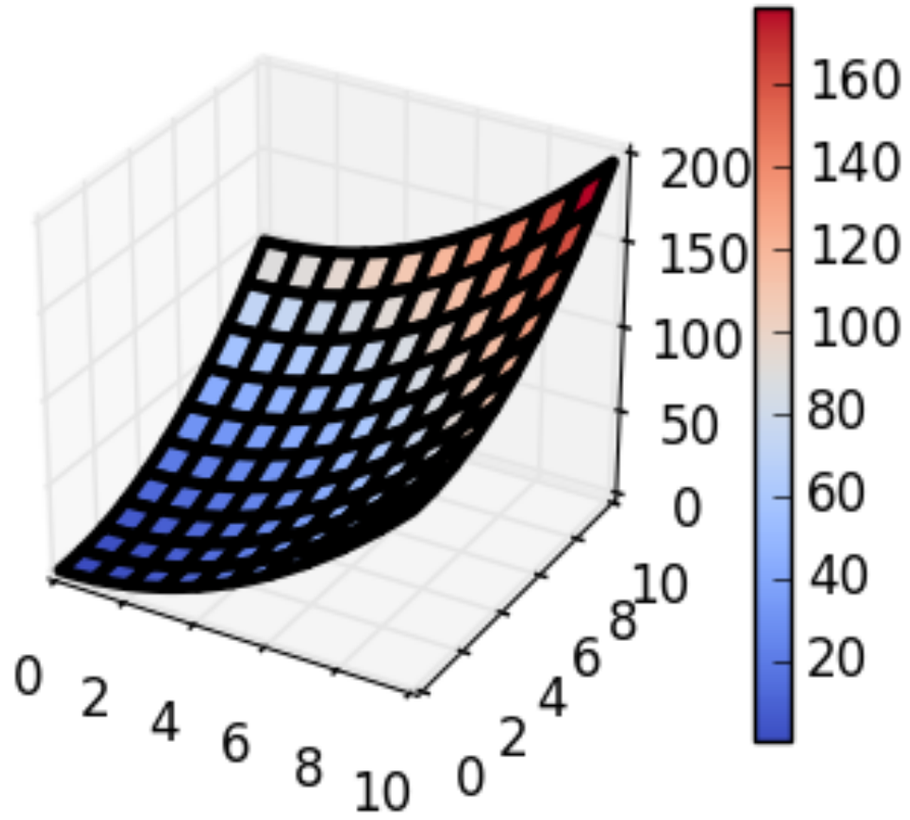




Matplotlib: Advanced plotting

- **3D figures**
  - from mpl_toolkits.mplot3d.axes3d import Axes3D
  - add projection='3d" to axes
  - need matrix inputs
  - surface plots with colormaps
  - colorbar with fig.colorbar(plot)

```
#figure
fig=plt.figure(figsize=(4,3))
#axes
axes=fig.add_subplot(1,1,1,projection='3d')
#plots
pl=axes.plot_surface(x,y,f,cmap=cm.coolwarm)
cb=fig.colorbar(pl)
```



Input and Output in Python

- **Objectives:**
  - Input files
  - Storing solutions

- **Input files**
  - create a file object with reading rights
  - open it
  - sparse every line in the file
  - eventually discard comments
  - convert into integer or float if required

```
f = file('input.txt','r')
f = open('input.txt','r')
for line in f
    line = line.split('#',1)[0]
    line = line.rstrip()
    param=float(line)
f.close()
```

- **Input/output solutions**
  - easy using numpy input/output
  - txt files
    - savetxt(filename,variable)
    - var=loadtxt(filename)
  - numpy binary files
    - save(filename,variable)
    - var=load(filename)

```
savetxt('output.txt',x)
xread=loadtxt('output.txt')

save('output.txt',x)
xbin=load('output.txt')
```

Python features

And much more …

Google

Workshop #1

- **Definition and plotting of a 1D and 2D function**
- **Computation of integral and error evaluation**
- **Solving f(x)=0**