Scientific Computing (Phys 2109/Ast 3100H)
I. Scientfic Software Development

SciNet HPC Consortium

University of Toronto

Winter 2013

Part I

Introduction to Software Development

Lecture 3

Intro to Python

Python

- Flexible, mature (20yo) scripting-style programming language
- Ubiquitous
- Huge standard library, massive number of third party modules
- Much slower than C/Fortran or even IDL/MATLAB

http://www.python.org

IPython

- Enhanced interactive Python shell
- `--pylab`: automatically loads lots of good math, plotting stuff.
- If you write Python scripts, have to load these yourself.
- IPython notebook: Mathematica/Maple-like IPython environment in browser

  $ **`ipython notebook --pylab inline`**

```
mercury2$ ipython --pylab
Python 2.7.3 (default, Apr 20 2012, 2
Type "copyright", "credits" or "licen

IPython 0.12.1 -- An enhanced Interac
?         -> Introduction and overvie
%quickref -> Quick reference.
help      -> Python's own help system
object?   -> Details about 'object',

Welcome to pylab, a matplotlib-based
For more information, type 'help(pyla

In [1]: █
```

http://ipython.org

Basic Python

- Variables
- Like most scripting languages, dont have to declare.
- Very handy for quick stuff, but has real drawbacks
- Math works the way you'd expect

```
In [1]: x = 2

In [2]: y = 3

In [3]: print x+y
5

In [4]: print x*y
6

In [5]: print y/x
1
```

Numpy, Arrays

- Python has lists [] but not "real" arrays
- Arrays are supplied by numpy, automatically included by pylab
- Numpy is the backbone of most scientific computing done in Python.

```
In [6]: z = array([1.,2.,3.,4.,5.])

In [7]: print z
[ 1.  2.  3.  4.  5.]

In [8]: print x*z
[  2.   4.   6.   8.  10.]

In [9]: z2d = array([ [1.,2.,3.],
   ...:              [4.,5.,6.]] )

In [10]: print z2d
[[ 1.  2.  3.]
 [ 4.  5.  6.]]

In [11]: print y*z2d
[[  3.   6.   9.]
 [ 12.  15.  18.]]
```

Numpy, SciPy

- Numpy provides basic N-dimensional array data structure, “fast” operations on that structure.
- Some low level math libraries
- SciPy has higher-level routines - linear algebra, fftpack, sparse matrix stuff, optimization modules, etc.

SciPy.org

http://www.scipy.org/SciPy

Python Loops

- For loops are more like foreach
- Each item in list
- If want a C-like for loop, use xrange (generates list 0..N-1)
- Note indentation: indentation is important in Python!

  (what happens with for element in z2d?)

```
In [12]: for element in z:
   ....:     print element
   ....:
1.0
2.0
3.0
4.0
5.0

In [13]: for name in ['Frank','Tina',
   ....: 'Sam','Kim']:
   ....:     print name
   ....:
Frank
Tina
Sam
Kim

In [14]: for i in xrange(10):
   ....:     print i,
   ....:
0 1 2 3 4 5 6 7 8 9
```

Python Functions

- Can also define functions
- 'def' keyword

```
In [15]: def squareNum(x):
   ....:     return x*x
   ....:

In [16]: print squareNum(4)
16

In [17]: print squareNum(7.3)
53.29

In [18]: print squareNum('no strings')
```

If/Else

- Control flow
- Same : syntax, same punctuation significance
- Functions needn't return a value

```
In [22]: def evenOrOdd(n):
   ....:     if n % 2 == 0:
   ....:         print "even."
   ....:     else:
   ....:         print "odd"
   ....:

In [23]: evenOrOdd(17)
odd

In [24]: evenOrOdd(18)
even.
```

Writing Python Files

- Can write functions in a file, import them in ipython
- specify them with filename.functionname
- Code not in functions will be run at import time.

```
mercury2 $ cat > myRoutines.py
def myFunction(x, y):
        '''This returns square of sum o
        return x**2+y**2
```

```
In [32]: import myRoutines

In [33]: myRoutines.myFunction?
Type:         function
Base Class:   <type 'function'>
String Form:<function myFunction at 0x2
Namespace:    Interactive
File:         /home/rzon/myRoutines.py
Definition:   myRoutines.myFunction(x, y)
Docstring:    This returns square of sum

In [35]: a = myRoutines.myFunction(1,2)

In [36]: print a
5
```

Multidimensional Arrays

- Some special arrays: identity matrix of size n x n, or arbitrary shape array of zeros
- Can pass nested list to 'array'

```
In [42]: eye(5)
Out[42]:
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])

In [43]: zeros([2,4,3])
Out[43]:
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.]]])

In [44]: array([[1.,3.],[-3.,2.]])
Out[44]:
array([[ 1.,  3.],
       [-3.,  2.]])
```

Multidimensional Arrays

- Python lists and numpy arrays are zero based.
- You can select out particular rows and columns.

```
In [52]: z = zeros([4,3])

In [53]: z[2,1] = 1

In [54]: print z
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  0.]]

In [55]: print z[:,1]
[ 0.  0.  1.  0.]
```

Python Array Slicing

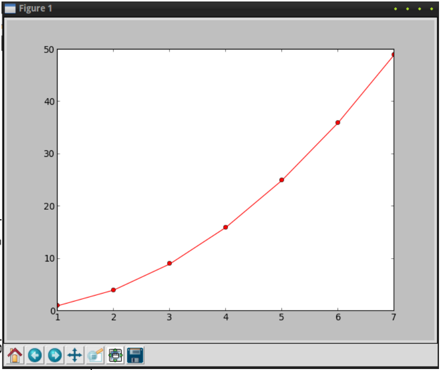
- Like in Fortran and MATLAB, but:
- ':' selects the entire range in that dimension
- start:end selects from start to before end
- start:end:stride

```
In [56]: a = ['a','b','c','d','e','f','g']

In [57]: a[1]
Out[57]: 'b'

In [58]: a[2]
Out[58]: 'c'

In [59]: a[3]
Out[59]: 'd'

In [60]: a[:]
Out[60]: ['a', 'b', 'c', 'd', 'e', 'f', 'g']

In [61]: a[1:3]
Out[61]: ['b', 'c']

In [62]: a[1:6:2]
Out[62]: ['b', 'd', 'f']
```

Visualization with matplotlib

Basic Plotting with Matplotlib

- http://matplotlib.org
- gallery of example with source code
- MATLAB-like

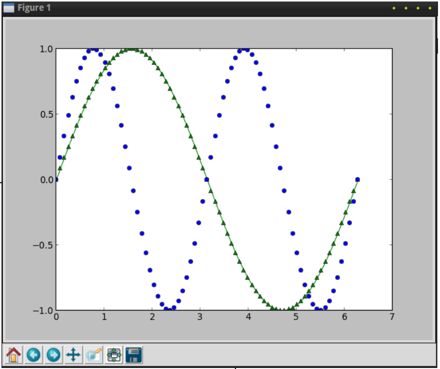



```
In [30]: x = array([1.,2.,3.,4.,5.,6.,7.])

In [31]: y = x*x

In [32]: plot(x,y)
Out[32]: [<matplotlib.lines.Line2D at 0x45

In [33]: clf()

In [34]: plot(x,y,'ro-')
Out[34]: [<matplotlib.lines.Line2D at 0x46e4690>]

In [35]: █
```

SciNet compute • calcul CANADA

Basic Plotting with Matplotlib

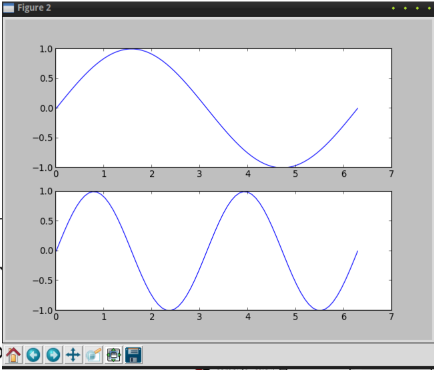
- `linspace(start,end,npnts)`
- pi, e defined
- by default, overplot




```
In [34]: clf()

In [35]: x = linspace(0,2*pi,75)

In [36]: y = sin(x)

In [37]: z = sin(2*x)

In [38]: plot(x, y, 'g^-')
Out[38]: [<matplotlib.lines.Line2D at 0x3192890>]

In [39]: plot(x, z, 'bo')
Out[39]: [<matplotlib.lines.Line2D at 0x2ac10d0>]

In [40]: 
```

Plotting Multiple Figures

- Use the `subplot` command.
- First two arguments are layout: number of rows and columns
- Last argument sets current plot




```
In [42]: figure()
Out[42]: <matplotlib.figure.Figur

In [43]: subplot(2,1,1)
Out[43]: <matplotlib.axes.AxesSub

In [44]: plot(x,y)
Out[44]: [<matplotlib.lines.Line2D at 0x3648810>]

In [45]: subplot(2,1,2)
Out[45]: <matplotlib.axes.AxesSubplot at 0x364e8d0>

In [46]: plot(x,z)
Out[46]: [<matplotlib.lines.Line2D at 0x381e650>]
```

Two-Dimensional Plotting

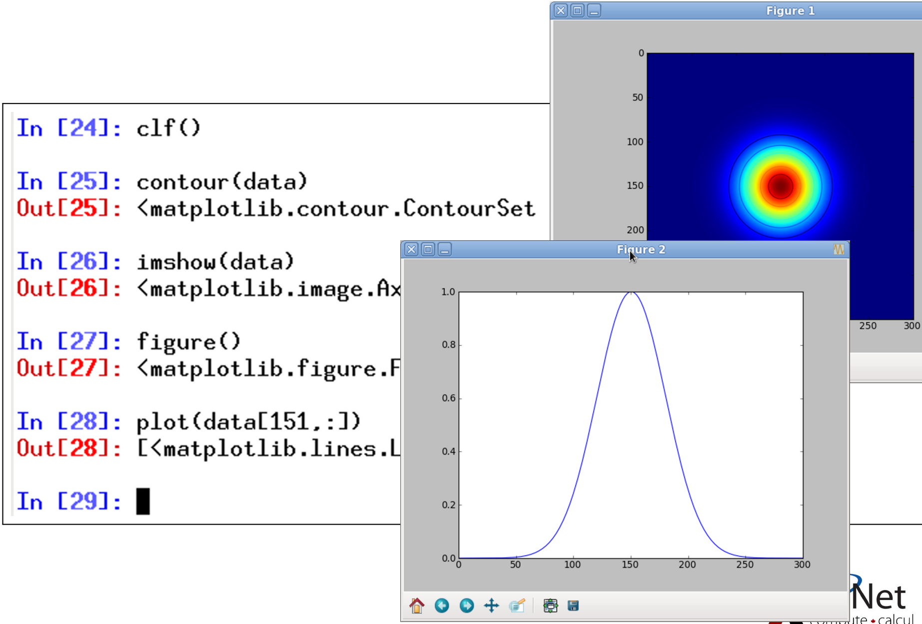
- First, let's load some 2d data
- Import your data from HW1
- mgrid - generate x,y coordinates

```
In [54]: data = genfromtxt("data.txt")

In [55]: shape(data)
Out[55]: (301, 301)

In [56]: #or generate

In [57]: x,y = mgrid[0:301,0:301]

In [58]: x=x-150

In [59]: y=y-150

In [60]: gauss=exp(-(x**2+y**2)/(2*30.**2))

In [61]: 
```

If you haven't finished HW1 yet:

```
mercury$cat>create_gausian.py
import math
f = open("data.txt","w")
dim = 301
for i in xrange(dim):
   for j in xrange(dim):
       x = i - dim/2.
       y = j - dim/2.
       z = math.exp(-(x**2+y**2)/(2*30.**2))
    f.write(str(z) + " ")
f.write("\n")
f.close()
mercury$python create_gausian.py
mercury$
```

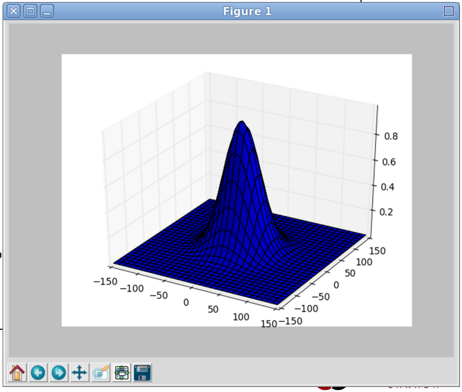
Two-Dimensional Plotting



Three-Dimensional Plotting

- Lots of very powerful things possible with matplotlib
- Once you leave the simple things, starts getting cryptic.

```
In [3]: from mpl_toolkits.mplot3d import Axes3D

In [4]: ax=gca(projection='3d')

In [5]: # older matplotlib version:

In [6]: #ax=Axes3D(figure())

In [7]: data=genfromtxt("data.txt")

In [8]: x,y=mgrid[-150:151,-150:151]

In [9]: ax.plot_surface(x,y,data)
Out[9]: <mpl_toolkits.mplot3d.art3d.Po

In [10]: draw()
```



Miscellaneous

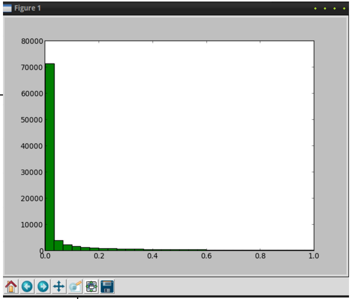
Miscellaneous: Analysis

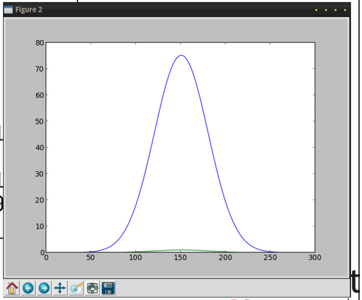
- Can get maximum
- Can get size of array
- Can create histograms
- Can select elements based on criterion
- . . .

```
In [12]: data=genfromtxt("data.txt")

In [13]: hist(data.flatten(),30);

In [14]: size(data)
Out[14]: 90601

In [15]: data.max()
Out[15]: 0.99972226079899995

In [16]: size(where(data>0.2))
Out[16]: 18224

In [17]: figure()
Out[17]: <matplotlib.figure.Figure at 0x4028a10>

In [18]: plot(sum(data,axis=1)); plot(data[151,:])
Out[18]: [<matplotlib.lines.Line2D at 0x402d890>]
```

Miscellaneous: Analysis

```
In [12]: data=genfromtxt("data.txt")

In [13]: hist(data.flatten(),30);

In [14]: size(data)
Out[14]: 90601

In [15]: data.max()
Out[15]: 0.99972226079899995

In [16]: size(where(data>0.2))
Out[16]: 18224

In [17]: figure()
Out[17]: <matplotlib.figure.Figure at 0x4028a1

In [18]: plot(sum(data,axis=1)); plot(data[151
Out[18]: [<matplotlib.lines.Line2D at 0x402d89
```





compute • calcul CANADA

Miscellaneous: Files

- Binary storage numpy array: save(z), load
- Text (Ascii) storage: loadtxt, savetxt, genfromtxt
- Won't discuss python specific pickle format
- Other python modules can use e.g. hdf5 and other binary formats
- Can open files by hand and write out explicitly

```
In [2]: a=linspace(0,1,100)

In [3]: b=sin(a)

In [4]: save('b.npy',b)

In [5]: savetxt('b.txt',b)

In [6]: quit()
```

```
mercury2 $ ls -l b.*
-rw-r--r-- 1 rzon scinet  880 Jan 22
-rw-r--r-- 1 rzon scinet 2500 Jan 22
```

```
In [1]: b=load("b.npy")

In [2]: c=loadtxt("b.txt")

In [3]: b-c
Out[3]:
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.,
```

Miscellaneous: From IPython to Python Scripts

- Python scripts best written in pure python
- Need to import modules that IPython loads by default:

```
from numpy import *
from matplotlib.pyplot import *
```

  Better practice:

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

  and prepend `np.` and `plt.` in the right places.
- Use # for comments
- Use `"""` in functions for documentation: docstring

C++ versus Python

- High performance
- Low-level programming possible
- Ubiquitous and standardized
- Useful libraries
- Modular design

- Easier to learn and understand
- High-level programming
- Interactive (IPython notebook)
- Graphics: matplotlib
- Slow performance

$\Rightarrow$ There is no 'best language' for every purpose.

Common: C++ for performance; Python as driver and post-processor