Introduction to Python (Part II)

Python for Computational Science

September 7th, 2010

Outline

- Review of Python
- Python for science - 3 powerful Python modules:
  - Numpy / Scipy - numerical computation
  - Matplotlib - plotting
  - Pytables - data storage
  - (All installed on SciNet)

Quick review of Python

- Interpreted language with an interactive shell
  - ipython = enhanced shell
- Everything is an object
- Variables
- Data structures
  - lists, tuples
  - dictionaries
  - sets
- Functions
- Classes and Objects

```
$ python  ## Run the python interpreter

Python 2.4.4 (#1, Oct 18 2006, 10:34:39)

[GCC 4.0.1 (Apple Computer, Inc. build 5341)] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>> a = 6       ## set a variable in this interpreter session
>>> a           ## entering an expression prints its value
6
>>> a + 2
8
>>> a = 'hi'    ## a can hold a string just as well
>>> a
'hi'
>>> len(a)      ## call the len() function on a string
2
>>> foo(a)      ## try something that doesn't work
Traceback (most recent call last):
  File "", line 1, in ?
NameError: name 'foo' is not defined
>>> ctrl-d             ## type ctrl-d to exit (ctrl-z on Windows)
```

```
#!/usr/bin/python

# import modules used here -- sys is a very standard one
import sys

# Gather our code in a main() function
def main():
  print 'Hello there', sys.argv[1]
  # Command line args are in sys.argv[1], sys.argv[2] ..
  # sys.argv[0] is the script name itself and can be ignored

# Standard boilerplate to call the main() function to begin
# the program.
if __name__ == '__main__':
  main()
```

Importing a module

import sys
import sys as s
from sys import *

Lists

```
# iterate over a list
squares = [1, 4, 9 , 16]
sum = 0
for num in squares:
    sum+=num

print sum


# check for an element in a list
list = ['larry', 'curly', 'moe']
if 'curly' in list:
    print 'yay'
else:
    print 'no'


#generate a sequence of numbers
for i in range(100):
    print i

```

List operations

slicing

```
#list slicing
list = ['a', 'b', 'c', 'd']
print list[1:-1]  #['b', 'c']
list[0:2] = 'z'
print list
```

list methods

- list.append(item)
- list.insert(index, elem)
- list.extend(list2)
- list.remove(elem)
- list.sort()
- list.reverse()
- list.pop()

Dictionary

- Series of key, value pairs
- keys are unique - strings, numbers, and tuples
- values can be any type

```
## Can build up a dict by starting with the the empty dict {}
 ## and storing key/value pairs into the dict like this:
 ## dict[key] = value-for-that-key
 dict = {}
 dict['a'] = 'alpha'
 dict['g'] = 'gamma'
 dict['o'] = 'omega'

 print dict  ## {'a': 'alpha', 'o': 'omega', 'g': 'gamma'}

 print dict['a']     ## Simple lookup, returns 'alpha'
 dict['a'] = 6       ## Put new key/value into dict
 'a' in dict         ## True
 ## print dict['z']                  ## Throws KeyError
 if 'z' in dict: print dict['z']     ## Avoid KeyError
 print dict.get('z')  ## None (instead of KeyError)
```

Dictionary

```
for key in dict:
        print key

print dict.keys()   ## ['a','o','g']
print dict.values() ## ['alpha', 'omega', 'gamma']
print dict.items()  ## [('a', 'alpha'),('o', 'omega'), ('g', 'gamma'')]

del dict['a']
print dict          ## {'o': 'omega', 'g': 'gamma'}
```

Files

- Read files line-by-line

```
f = open("file.txt")
for line in f:
    split_line = line.split(' ')
    ...
```

- csv module (`import csv`)
  - reads and writes all kinds of delimited text files
- pyparsing module (`import pyparsing`)
  - define a “grammar” to parse any kind of file
  - ex: parse CHARMM topology files using a grammar
- A few XML parsing modules (expat, sax, etree)
- There is usually a module that can parse your file!

Built-in Python modules

- os (file operations)
- sys (system operations)
- re (regular expressions, “like perl!”)
- math (math stuff, ceil, floor, exp, log, etc...)
- optparse (command line argument parsing)
- logging (easy logging for your scripts or applications)
- random (random number generation)
- datetime (date and time!)
- ... many more

Tons of other modules!

- numpy and scipy
- matplotlib
- pytables

Why Python for Science

- Rapid development
  - Easily write testable, readable code
- Write less code: modules to do anything you need
- Well-documented, with lots of examples for learning
- Glue together existing applications
- Replace long bash scripts

Python for science - 3 powerful Python modules:

- Numpy / Scipy - numerical computation
- Matplotlib - plotting
- Pytables - data storage

Numpy

- Python numerical package
- Homogeneous numerical multidimensional arrays
- Written in C
- Makes computing and plotting numerical data organized as vectors, matrices easy
- Built-in functions
    - statistical - histogram
    - linear algebra - dot, cross, determinants

Numpy array creation

```
from numpy import *

a = array([1,2,3])
b = array((10,11,12))

>>> print a+b
array([11,13,15])

>>> print a.dtype
dtype('<i4')

a = array([1,2,3], dtype=float)

t = arange(0, 2*pi, 0.1)
sinvalues = sin(t)
```

Numpy 2D array

```
# 2D arrays
>>> b = arange(12).reshape(3,4)
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>>
>>> b.sum(axis=0)                            # sum of each column
array([12, 15, 18, 21])
>>>
>>> b.min(axis=1)                            # min of each row
array([0, 4, 8])
>>>
>>> b.cumsum(axis=1)                         # cumulative sum along the rows
array([[ 0,  1,  3,  6],
       [ 4,  9, 15, 22],
       [ 8, 17, 27, 38]])
```

- data_array = numpy.genfromtxt(datafile, comments="#", delimiter=" ", names="a,b", skip_header=3, skip_footer=5, usecols=(0,-1), missing_values={0:"N/A", 'b':" "}, filling_values={0:0, 'b':0})
- numpy.savetxt('data.txt', data_array, fmt='%0.3f')

Numpy array slicing

```
t = arange(0, 2*pi, 0.1)
sinvalues = sin(t)

# slicing
>>>t[:]                # get all t-values
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,
        3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,
        4.4,  4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,
        5.5,  5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2])

>>>t[2:4]              # get sub-array with the elements at the indexes 2,3
array([  0.2,  0.3,  0.4 ])


>>>t[0:6:2]            # every even-indexed value up to but excluding 6
array([ 0. ,  0.2,  0.4])
```

Numpy 2D array slicing

```
#2D slicing
>>> t_mat = t.reshape(t.size/9, 9)
>>> t_mat
array([[ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8],
       [ 0.9,  1. ,  1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7],
       [ 1.8,  1.9,  2. ,  2.1,  2.2,  2.3,  2.4,  2.5,  2.6],
       [ 2.7,  2.8,  2.9,  3. ,  3.1,  3.2,  3.3,  3.4,  3.5],
       [ 3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,  4.4],
       [ 4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3],
       [ 5.4,  5.5,  5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2]])
>>> t_mat[0]
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8])

>>> t_mat[:,0:2]
array([[ 0. ,  0.1],
       [ 0.9,  1. ],
       [ 1.8,  1.9],
       [ 2.7,  2.8],
       [ 3.6,  3.7],
       [ 4.5,  4.6],
       [ 5.4,  5.5]])

>>> t_mat[0:3, 0:5]
array([[ 0. ,  0.1,  0.2,  0.3,  0.4],
       [ 0.9,  1. ,  1.1,  1.2,  1.3],
       [ 1.8,  1.9,  2. ,  2.1,  2.2]])
```

Numpy operations

- A+B, A-B, *c**A
- numpy.linalg
- dot(a,b)
- inner(a,b)
- Lots of high level functions
  - Decompositions
  - Matrix eigenvalues
  - Basic fourier analysis

Scipy

- science and math algorithms built on top of numpy for Python
- import scipy

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
maxentropy	Maximum entropy methods
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions
weave	C/C++ integration

Python for science - 3 powerful Python modules:

- Numpy / Scipy - numerical computation
- ➡ Matplotlib - plotting
- Pytables - data storage

Matplotlib

- Library to generate plots from data
- Like gnuplot
- Can output to various file formats:
  - svg, eps, png
- ipython -pylab : interactive plotting

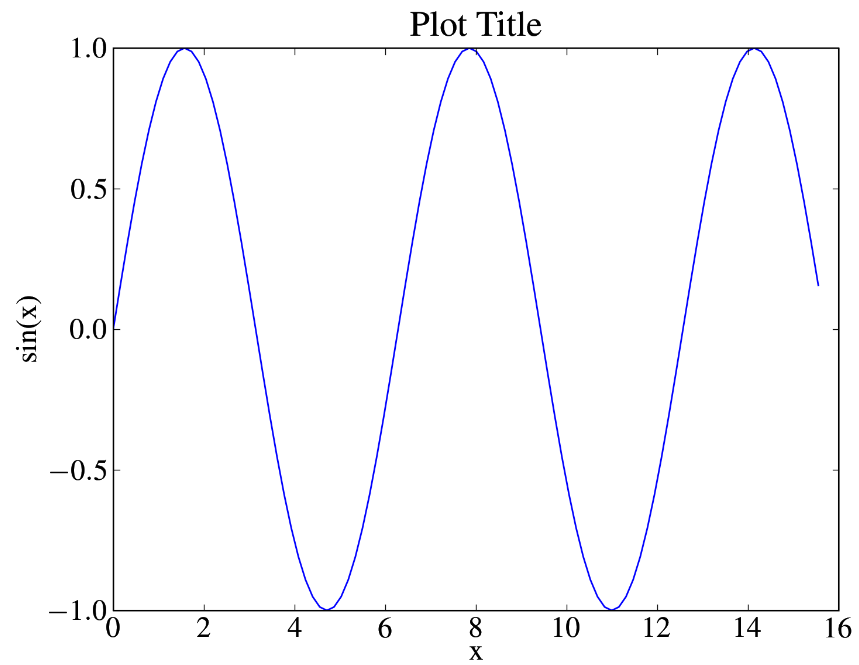
Using matplotlib

- ipython -pylab = pyplot + numpy functions
  - plot(x,y)
  - plotfile('data.dat', (0,1))
- Customizing the figure
  - Figure object
  - Each line of code modifying a figure object
    - add title, grid, labels, etc
- savefig('a.png')

Demo

Matplotlib Example: Simple Plot

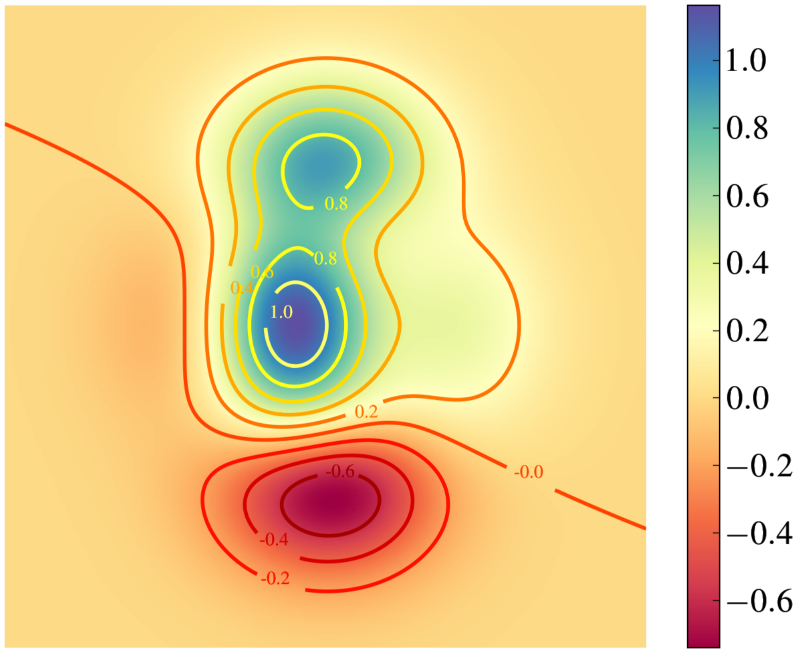
```
>>> import numpy as N
>>> import pylab as pl
>>> x = N.arange(0,1,0.01) * 5 * N.pi
>>> pl.plot(x,N.sin(x))
>>> pl.xlabel('x')
>>> pl.ylabel('sin(x)')
>>> pl.title('Plot Title')
```



Matplotlib Example: 2-D Plot

```
>>> import numpy as N
>>> import pylab as pl
>>> def z_func(x,y):
...     return (1-x+x**3+y**5)*N.exp(-(x**2+y**2))
...
>>> x    = N.arange(-3.0,3.0,0.025)
>>> y    = N.arange(-3.0,3.0,0.025)
>>> X,Y  = pl.meshgrid(x, y)
>>> Z    = z_func(X, Y)
>>> im   = pl.imshow(Z,interpolation='bilinear',/
...     cmap=pl.cm.Spectral)
>>> cset = pl.contour(Z,N.arange(-1.2,1.6,0.2),/
...     linewidths=2,cmap=pl.cm.hot)
>>> pl.clabel(cset,inline=True,fmt='%1.1f',/
...     fontsize=10)
>>> pl.colorbar(im)
>>> pl.axis('off')
>>> pl.title('$z=(1-x+x^3+y^5) e^-(x^2+y^2)$')
```

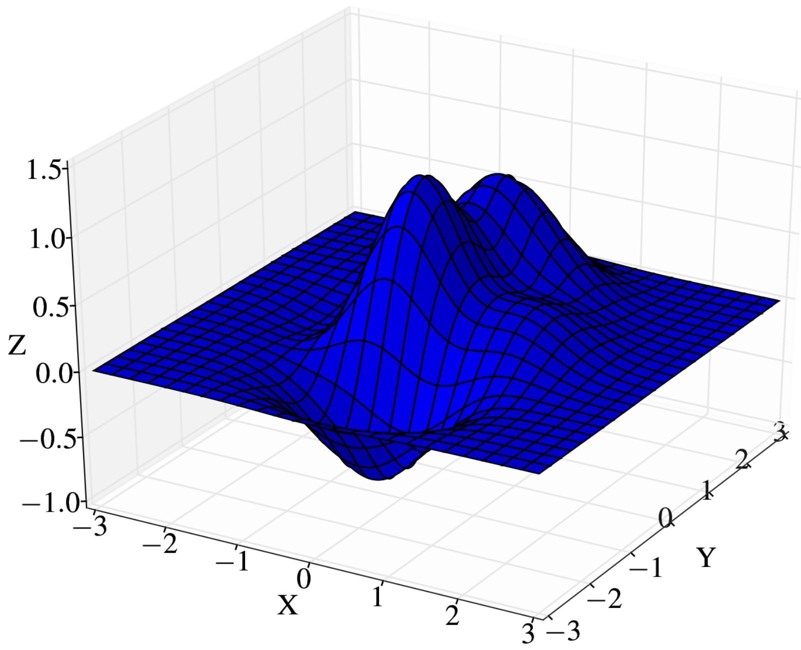
$$z = (1 - x + x^3 + y^5)\, e^{-(x^2+y^2)}$$



Wednesday, October 6, 2010

Matplotlib Example: 3-D Plot

```
>>> import numpy as N
>>> import pylab as pl
>>> from matplotlib import axes3d
>>> from myFunctions import z_func
>>> x   = N.arange(-3.0,3.0,0.025)
>>> y   = N.arange(-3.0,3.0,0.025)
>>> X,Y = pl.meshgrid(x, y)
>>> Z   = z_func(X, Y)
>>> fig = pl.figure()
>>> ax  = axes3d.Axes3D(fig)
>>> ax.plot_surface(X,Y,Z)
>>> ax.set_xlabel('X')
>>> ax.set_ylabel('Y')
>>> ax.set_zlabel('Z')
```



Wednesday, October 6, 2010

Pytables

- A wrapper around the HDF5 file format and numpy
- HDF5
  - Hierarchial data - tables and arrays in a tree-like structure
    - Similar to working with directories and files in UNIX filesystem
      - /group/table
- Data organized as rows (records), querying is done on rows
- Declare a descriptor for values in your row vector
- Automatic conversion between numpy array objects and rows in pytable

PyTables Highlights (I)

- Designed for efficiently dealing with extremely large amounts of data.
- High level of flexibility for structuring your data:
  - Datatypes: scalars (numerical & strings), records, enumerated, time...
  - Multidimensional cells
  - Nested records
  - Variable length arrays
- Support for the complete Numeric/numarray/NumPy family.

PyTables Highlights (II)

- Transparent data compression support (Zlib, LZO, Bzip2...).
- Support of full 64-bit addressing in files, even on 32-bit platforms.
- Can handle generic HDF5 files (most of them).
- Aware of little/big endian issues (data is portable).
- It's Open Source (BSD license).

PyTables Highlights (II)

- Transparent data compression support (Zlib, LZO, Bzip2...).
- Support of full 64-bit addressing in files, even on 32-bit platforms.
- Can handle generic HDF5 files (most of them).
- Aware of little/big endian issues (data is portable).
- It's Open Source (BSD license).

Ease of Use

Natural naming

```
# access to file:/group1/table
table = file.root.group1.table
```

Support for generalized slicing

```
# step means a stride in the slice
array[idx, start:stop, :, start:stop:step]
```

Support for iterators

```
# get the values in col1 that satisfy the
# (1.3 < col3 <= 2.) condition in table
col3 = table.cols.col3
[r['col1'] for r in table.where(1.3 < col3 <= 2.)]
```

Constructing the table

- import tables

```
RGTable = {
        'temp' : Int32Col(dflt=0, pos=0),
        'replicanum' : Int32Col(dflt=0, pos=1),
        'seqnum' : Int32Col(dflt=0, pos=2),
        'time' : Int32Col(dflt=0, pos=3),
        'Rg' : Float32Col(dflt=0.0, pos=4),
        'Rgx' : Float32Col(dflt=0.0, pos=5),
        'Rgy' : Float32Col(dflt=0.0, pos=6),
        'Rgz' : Float32Col(dflt=0.0, pos=7)
}
```

- tables.openFile('analysis.h5', mode='a')
- mygroup = tables.createGroup(file.root, 'groupname')
- mytable = tables.createTable(file.root.groupname, 'tablename', RGTable)

Querying the table

```
RGTable = {
        'temp' : Int32Col(dflt=0, pos=0),
        'replicanum' : Int32Col(dflt=0, pos=1),
        'seqnum' : Int32Col(dflt=0, pos=2),
        'time' : Int32Col(dflt=0, pos=3),
        'Rg' : Float32Col(dflt=0.0, pos=4),
        'Rgx' : Float32Col(dflt=0.0, pos=5),
        'Rgy' : Float32Col(dflt=0.0, pos=6),
        'Rgz' : Float32Col(dflt=0.0, pos=7)
}
```

```
file = tables.openFile(analysisfile)
table = file.getNode('/'+group+'/'+tablename)

for T in Tlist:
    readout = table.readWhere('temp==%(T)d' % vars())
```

Gets all the rows where the temperature is T

Putting it all together

References

- Google's Python class
- Python for Scientific Computing Hans Petter langtangen third edition, Chapter 3, 4
- A Primer on Scientific Programming with Python, Hans Petter Langtangen
- http://docs.scipy.org/doc/numpy/reference
- http://www.scipy.org/Plotting_Tutorial