Python for Scientific Computing

Ivan Lima

Woods Hole Oceanographic Institution
Dept. of Marine Chemistry & Goechemistry
(**ivan@whoi.edu**)

November 14, 2006

Outline

What is Python?

Python is an elegant and robust programming language that combines the power and flexibility of traditional compiled languages with the ease-of-use of simpler scripting and interpreted languages.

- High level
- Interpreted
- Scalable
- Extensible
- Portable
- Easy to learn, read and maintain
- Robust
- Object Oriented
- Versatile

Why Python?

1. Free & open source
2. Available on a wide variety of platforms
3. Better support for arrays with more than 2 dimensions
4. Better support for wrapping FORTRAN 77/90/95 & C/C++ code
5. Far better memory management
6. Much wider library support for non-numerical work
7. Far more options for a full featured GUI
8. Easier to create stand-alone applications on any platform.
9. Truly Object Oriented

Numeric

- Original array module
- Very successful
- Large code base
- Still around
- Limited

Numarray

- Created at the STSCI
- Very powerful & lots of new features
- Fast for large arrays but slow for small arrays
- Incompatible with existing code base (Numeric/SciPy)
- Slow acceptance
- Fragmented community

NumPy

- Best of both Numeric & Numarray
- Very powerful & flexible
- Core array object for SciPy
- Include migration tools for Numeric
- Numeric development has ceased
- Numarray development moving to NumPy

Python/NumPy Example

```
>>> import numpy
>>> a = numpy.arange(12)
>>> print a
[ 0  1  2  3  4  5  6  7  8  9 10 11]
>>> print a.size, a.ndim, a.shape
12 1 (12,)
>>> a = numpy.reshape(a,(3,4))
>>> print a
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
>>> print a.size, a.ndim, a.shape
12 2 (3, 4)
>>> print a.min(), a.max(), a.mean(), a.std()
0 11 5.5 3.45205252953
>>> print a[:,0]
[0 4 8]
>>> print a[2,:]
[ 8  9 10 11]
>>> b = numpy.arange(3)
>>> c = numpy.arange(4)
>>> b * c
ValueError: shape mismatch: objects cannot
be broadcast to a single shape
>>> print b, c
[0 1 2] [0 1 2 3]
```

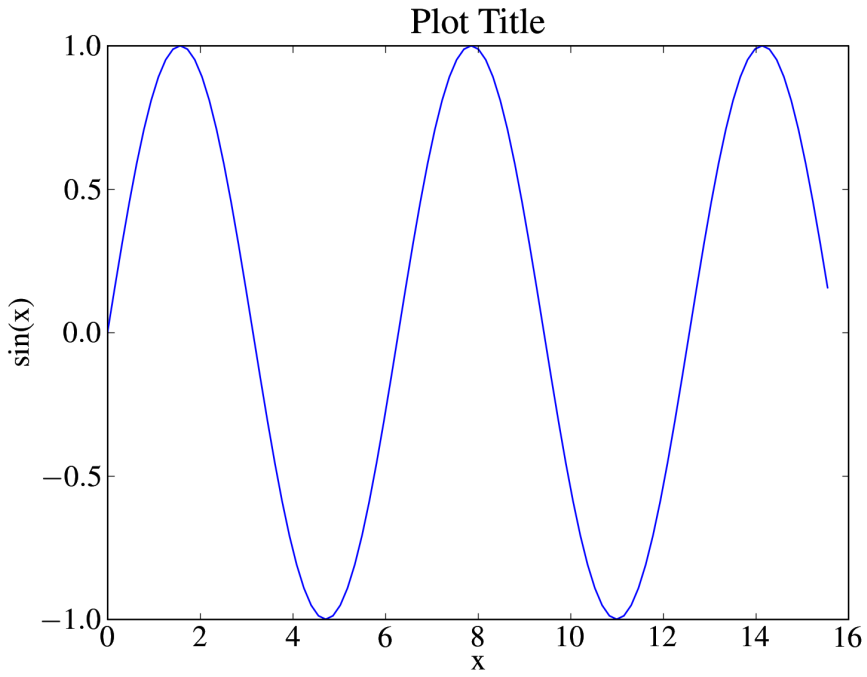
```
>>> print numpy.reshape(b,(3,1))
[[0]
 [1]
 [2]]
>>> print numpy.reshape(b,(3,1)) * c
[[0 0 0 0]
 [0 1 2 3]
 [0 2 4 6]]
>>> print b[:,numpy.newaxis] * c
[[0 0 0 0]
 [0 1 2 3]
 [0 2 4 6]]
>>> print a.shape, c.shape
(3, 4) (4,)
>>> print a * c
[[ 0  1  4  9]
 [ 0  5 12 21]
 [ 0  9 20 33]]
>>> print a.shape, b.shape
(3, 4) (3,)
>>> print a * b[:,numpy.newaxis]
[[ 0  0  0  0]
 [ 4  5  6  7]
 [16 18 20 22]]
>>> print a.shape, b[:,numpy.newaxis].shape
(3, 4) (3,1)
```

Matplotlib/Basemap

- Emulates Matlab syntax (pylab interface)
- Good for interactive sessions
- Object Oriented
- Excellent 2-D graphics & some 3-D capability
- Map toolkit (Basemap)
- Works with Numeric, Numarray & NumPy

Matplotlib Example: Simple Plot

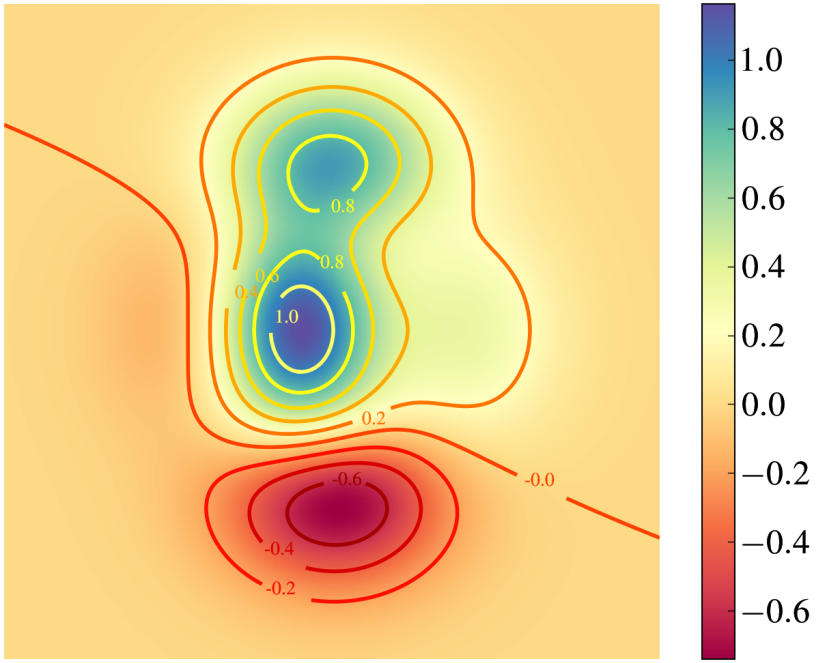
```
>>> import numpy as N
>>> import pylab as pl
>>> x = N.arange(0,1,0.01) * 5 * N.pi
>>> pl.plot(x,N.sin(x))
>>> pl.xlabel('x')
>>> pl.ylabel('sin(x)')
>>> pl.title('Plot Title')
```



Matplotlib Example: 2-D Plot

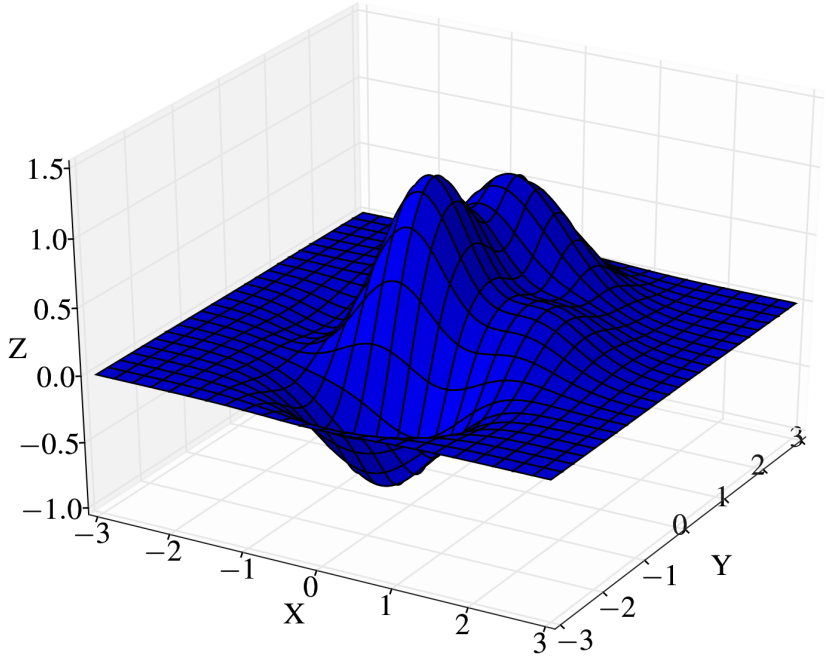
```
>>> import numpy as N
>>> import pylab as pl
>>> def z_func(x,y):
...     return (1-x+x**3+y**5)*N.exp(-(x**2+y**2))
...
>>> x    = N.arange(-3.0,3.0,0.025)
>>> y    = N.arange(-3.0,3.0,0.025)
>>> X,Y  = pl.meshgrid(x, y)
>>> Z    = z_func(X, Y)
>>> im   = pl.imshow(Z,interpolation='bilinear',/
...     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('$z=(1-x+x^3+y^5) e^-(x^2+y^2)$')
```

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



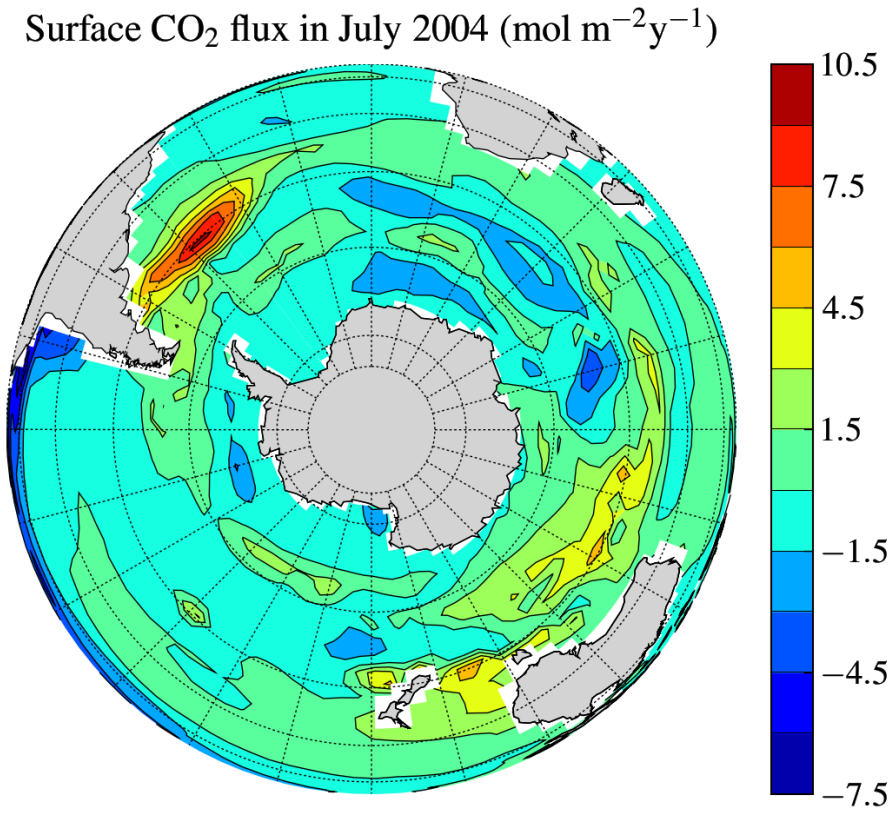
Matplotlib Example: 3-D Plot

```
>>> 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')
```



Matplotlib Example: Map

```
>>> import numpy as N
>>> import pylab as pl
>>> import netCDF4
>>> from numpy import ma as MA
>>> from matplotlib.toolkits.basemap import Basemap
>>> fpin = netCDF4.Dataset('ccsm_840-12.nc','r')
>>> tlat      = fpin.variables['TLAT'][:]
>>> tlon      = fpin.variables['TLONG'][:]
>>> fgco2     = fpin.variables['FG_CO2'][:]
>>> fillvalue = fpin.variables['FG_CO2']._FillValue
>>> fpin.close()
>>> fgco2 = MA.masked_values(fgco2,fillvalue)
>>> fgco2 = fgco2*1.e-2*1.e-3*60.*60.*24.*365.
>>> map = Basemap(projection='ortho',lat_0=-90,/
...    lon_0=0,resolution='c')
>>> x,y = map(tlon,tlat)
>>> ax  = pl.axes([0.1,0.1,0.7,0.7],axisbg='white')
>>> CS  = map.contourf(x,y,fgco2,15,cmap=pl.cm.jet)
>>> map.drawcoastlines(linewidth=0.5)
>>> map.drawmeridians(N.arange(-180,180,15))
>>> map.drawparallels(N.arange(-90,90,15))
>>> map.fillcontinents(color='lightgrey')
>>> pl.title('Surface CO$_2$ flux in July 2004 /
...    (mol m$^-2$y$^-1$)')
>>> l,b,w,h = ax.get_position()
>>> cax     = pl.axes([l+w+0.025, b, 0.025, h])
>>> pl.colorbar(CS,cax=cax,drawedges=True)
```

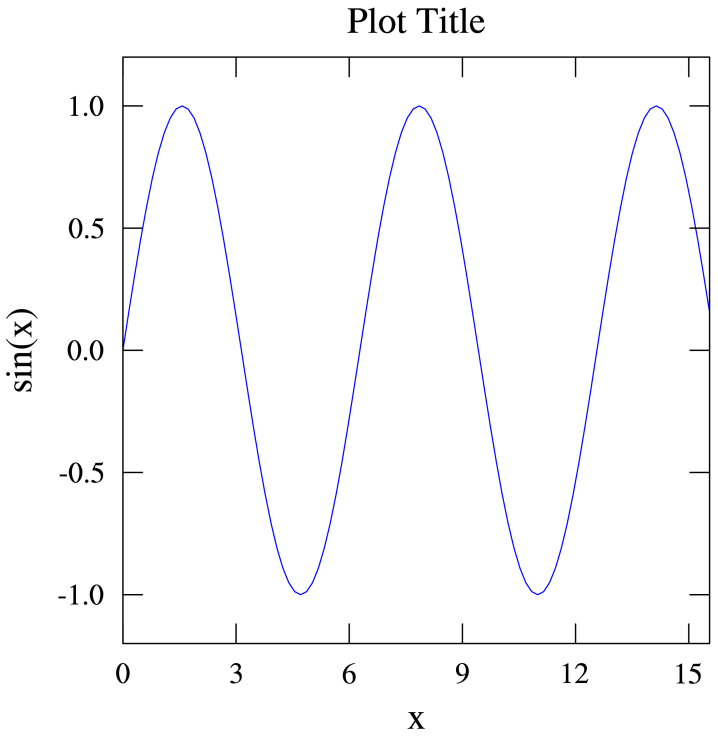


PyNGL

- Python interface to NCAR's NCL
- Geared towards geophysical/geospatial data
- Steeper learning curve
- Not very good for interactive sessions
- Better for maps
- Handles wide variety of spatial grids (curvilinear, irregular)
- Excellent 2-D graphics & no 3-D capability
- Works with Numeric & NumPy
- PyNIO read/write netCDF, GRIB 1, HDF 4, HDFEOS 2, and CCM history files

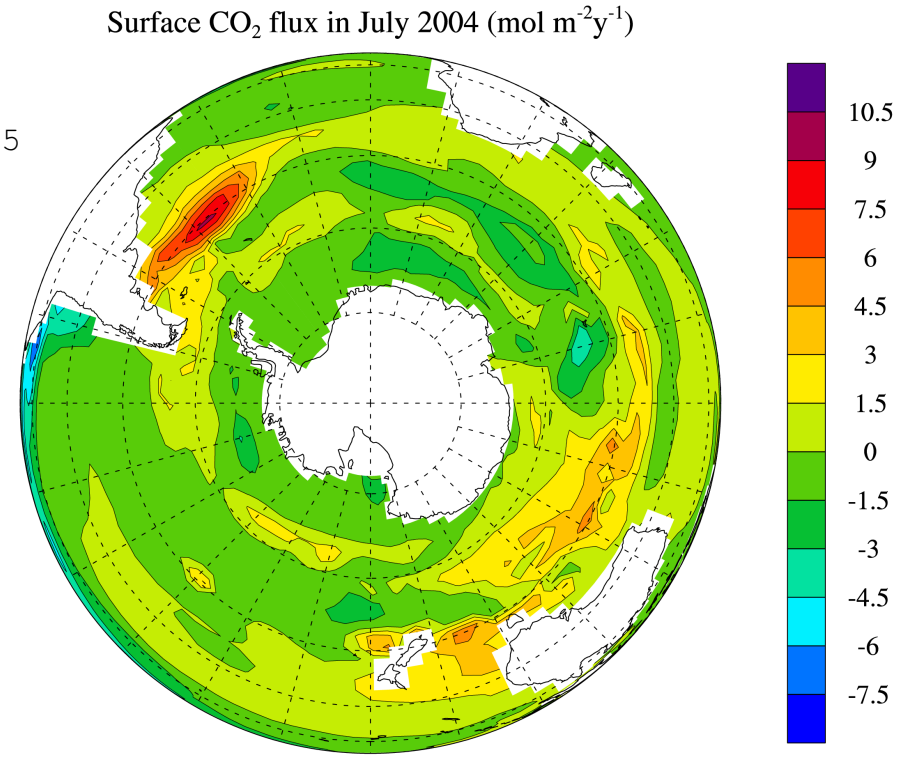
PyNGL Example: Simple Plot

```
>>> import numpy as N
>>> import PyNGL_numpy.Ngl as Ngl
>>> x = N.arange(0,1,0.01) * 5 * N.pi
>>> pres = Ngl.Resources()
>>> pres.tiXAxisString       = 'x'
>>> pres.tiYAxisString       = 'sin(x)'
>>> pres.tiMainString        = 'Plot Title'
>>> pres.trXMaxF             = x[-1]
>>> pres.xyLineColors        = ['blue']
>>> pres.nglPaperOrientation = 'portrait'
>>> wres = Ngl.Resources()
>>> wks  = Ngl.open_wks('ps','example_06',wres)
>>> plot = Ngl.xy(wks,x,N.sin(x),pres)
```



PyNGL Example: Map

```
(import modules, read data)
>>> mres = Ngl.Resources()
>>> mres.sfXArray                = tlon[:]
>>> mres.sfYArray                = tlat[:]
>>> mres.sfMissingValueV         = fillvalue
>>> mres.cnLevelSelectionMode    = 'ExplicitLevels'
>>> mres.cnLevels                = N.arange(-7.5,11.5
>>> mres.cnLineThicknessF        = 0.5
>>> mres.pmLabelBarDisplayMode   = 'Always'
>>> mres.pmTickMarkDisplayMode   = 'Never'
>>> mres.mpProjection            = 'Orthographic'
>>> mres.mpMinLonF               = -180
>>> mres.mpMaxLonF               = 180
>>> mres.mpMinLatF               = -90
>>> mres.mpMaxLatF               = 90
>>> mres.mpCenterLonF            = 0
>>> mres.mpCenterLatF            = -90
>>> mres.mpGridSpacingF          = 15
>>> mres.mpPerimOn               = False
>>> mres.tiMainString            = 'Surface /
...    CO~B~2~N~ flux in July 2004 /
...    (mol m~S~- 2~N~y~S~-1~N~)'
>>> wres = Ngl.Resources()
>>> wres.wkColorMap = 'BlAqGrYeOrReVi200'
>>> wks = Ngl.open_wks('ps','example_07',wres)
>>> map = Ngl.contour_map(wks,MA.filled(fgco2,fillvalue),mres)
```



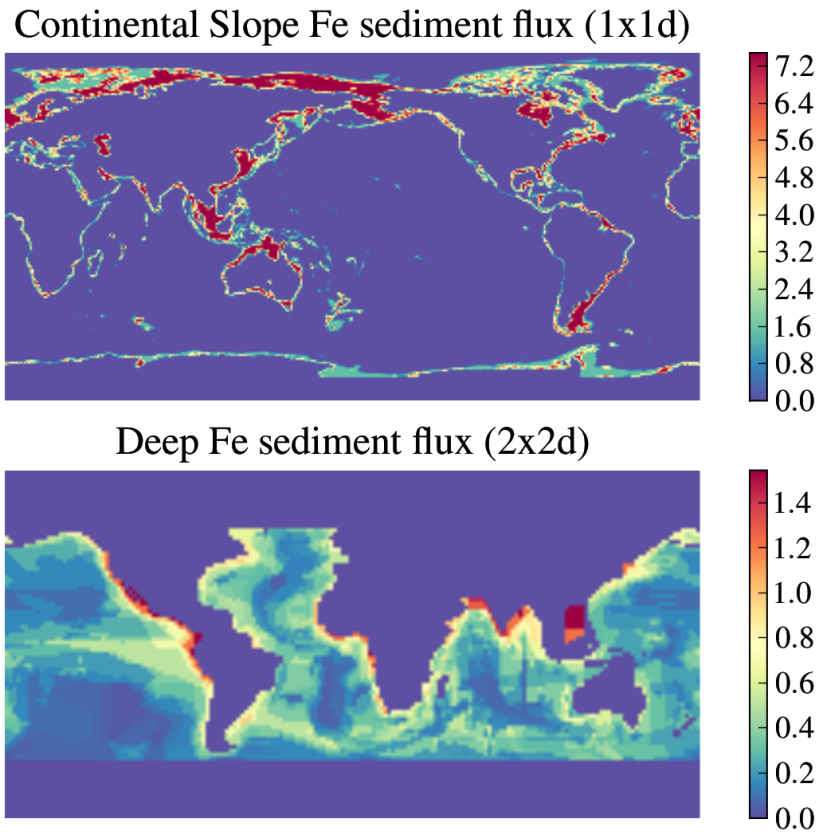
SciPy

Based on NumPy

- Input-Output
- Integration
- Optimization
- Interpolation
- Signal Processing/Filtering
- Statistics
- Fourier Transforms
- Linear Algebra

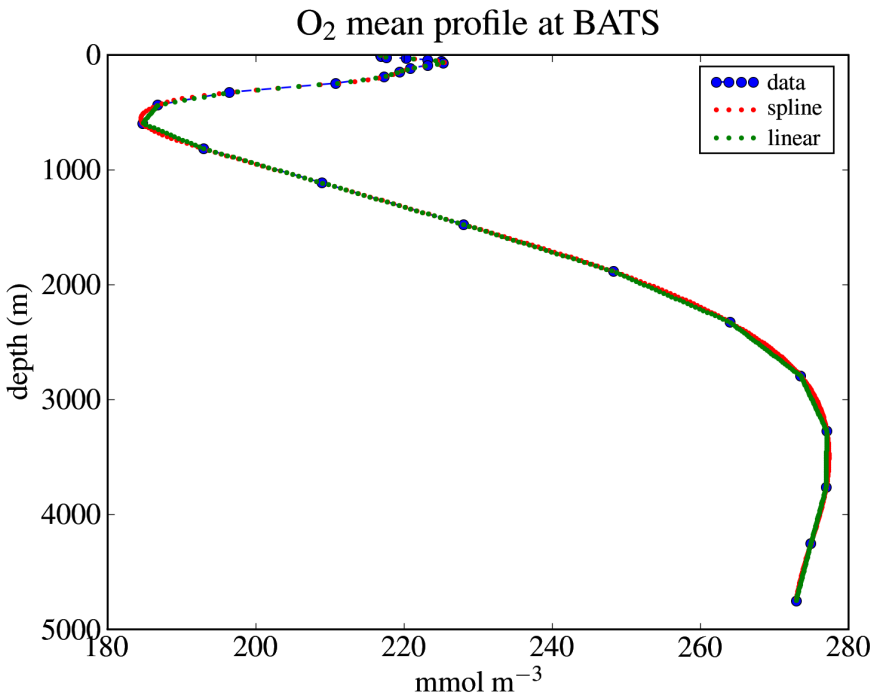
SciPy Example: IO

```
>>> import numpy as N
>>> import pylab as pl
>>> from scipy.io import fread, bswap, read_array
>>> fbin = open('slopeFeflux_1x1d.bin','r')
>>> feflux_slope = fread(fbin,180*360,'f')
>>> fbin.close()
>>> feflux_slope = bswap(feflux_slope)
>>> feflux_slope = N.reshape(feflux_slope,/
...     (180,360))
>>> pl.subplot(2,1,1)
>>> pl.imshow(feflux_slope,cmap=pl.cm.Spectral_r)
>>> pl.colorbar()
>>> pl.title('Continental Slope Fe sediment flux /
...      (1x1d)' ,fontsize=18)
>>> pl.axis('off')
>>> feflux_deep = read_array('deepFeflux_2x2d.txt')
>>> pl.subplot(2,1,2)
>>> pl.imshow(feflux_deep,cmap=pl.cm.Spectral_r)
>>> pl.colorbar()
>>> pl.title('Deep Fe sediment flux (2x2d)',/
...     fontsize=18)
>>> pl.axis('off')
```



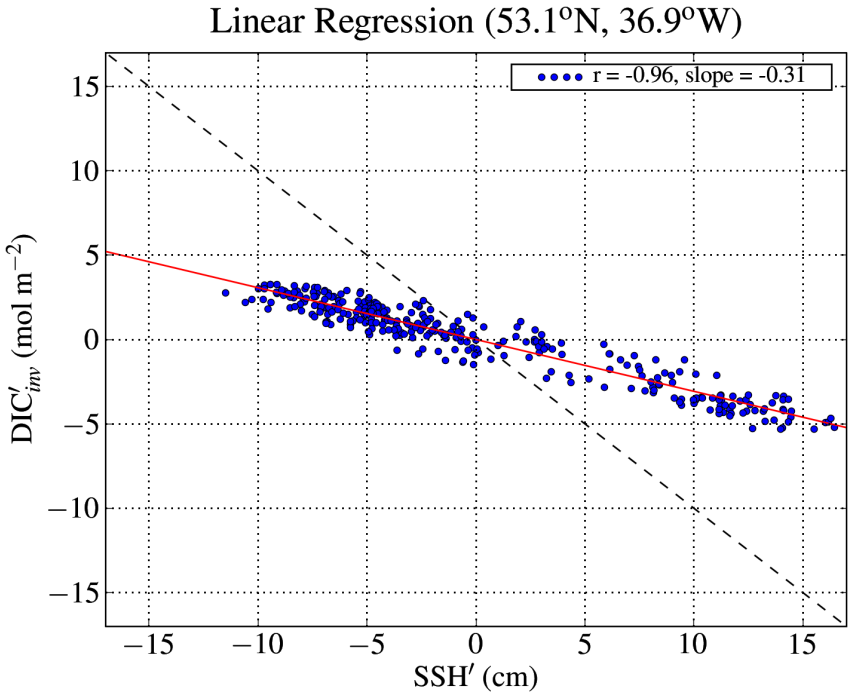
SciPy Example: Interpolation

```
>>> import numpy as N
>>> import pylab as pl
>>> import netCDF4
>>> from scipy.interpolate import interp1d, /
...     splrep, splev
>>> fpin = netCDF4.Dataset('BATSgx3v5.23.nc','r')
>>> zt   = fpin.variables['z_t'][:]
>>> o2   = fpin.variables['O2'][:]
>>> fpin.close()
>>> zt     = zt/100
>>> o2_avg = o2.mean(axis=0)
>>> pl.plot(o2_avg,-zt,'b-o')
>>> pl.xlabel('mmol m$^-3$')
>>> pl.ylabel('depth (m)')
>>> pl.title('O$_2$ mean profile at BATS',/}
...     fontsize=22)
>>> zt_new = N.arange(5,4755,25)
>>> tck    = splrep(zt,o2_avg)
>>> o2_spl = splev(zt_new,tck)
>>> pl.plot(o2_spl,-zt_new,'ro')
>>> f      = interp1d(zt,o2_avg)
>>> o2_lin = f(zt_new)
>>> pl.plot(o2_lin,-zt_new,'go')
```



SciPy Example: Linear Regression

```
>>> import numpy as N
>>> import pylab as pl
>>> import netCDF4
>>> from scipy.stats import linregress
>>> fpin      = netCDF4.Dataset('DIC_SSH_data.nc','
>>> lat       = fpin.variables['TLAT'][:]
>>> lon       = fpin.variables['TLONG'][:]
>>> anom_ssh  = fpin.variables['SSH_anom'][:]
>>> anom_idic = fpin.variables['IDIC_anom'][:]
>>> fpin.close()
>>> slope, intercept, r, prob, stderr = /
...     linregress(anom_ssh,anom_idic)
>>> print slope, intercept, r, prob, stderr
-0.307, 2.04e-05, -0.957, 2.39e-169, 0.681
>>> reg_label = 'r = %.2f, slope = %.2f'%(r,slope)
>>> lat_label = '%.1f\textsuperscript{o}N'%(lat)
>>> lon_label = '%.1f\textsuperscript{o}W'%(360-lon
>>> sshnew = N.array([50.,-51.])
>>> pl.plot(sshnew,-sshnew,'k--',label='_nolegend_'
>>> pl.xlabel('SSH$^\prime$ (cm)')
>>> pl.ylabel('DIC$_inv^\prime$ (mol m$^-2$)')
>>> pl.title('Linear Regression (%s, %s)'%(lat_label, lon_label),/
...     fontsize=22)
>>> pl.grid(True)
>>> pl.plot(anom_ssh,anom_idic,'bo',markersize=4,label=reg_label)
>>> pl.plot(sshnew,intercept + slope * sshnew,'r-',label='_nolegend_')
>>> pl.legend()
```



ScientificPython

Based on Numeric

- Input-Output (netCDF 3.x)
- Integration
- Optimization
- Interpolation
- Signal Processing/Filtering
- Statistics
- VRML
- MPI

FORTRAN 77/90/95

Designed specifically to work with NumPy/Numeric arrays

- F2PY
  - Uses NumPy
  - Easy to use
  - Handle more wrapping tasks
  - Harder to debug
- PyFort
  - Uses Numeric
  - Not as easy to use
  - More limited

F2PY Example

seawater.f90

```
module sw
implicit none
real, parameter :: pi = 3.14159265358979323846
contains
    subroutine press(Z,lat,zlen,p)
        ! Computes pressure in dbars
        ! from depth in m
        real, parameter :: deg2rad = pi/180.
        real :: x, c1
        integer, intent(in) :: zlen
        real, intent(in) :: lat
        real, dimension(zlen), intent(in)  :: Z
        real, dimension(zlen), intent(out) :: p
        x    = sin(abs(lat) * deg2rad)
        c1   = 5.92e-3 + x ** 2. * 5.25e-3
        p    = ((1.-c1) - sqrt(((1.-c1)**2) - &
            (8.84e-6 * Z))) / 4.42e-6
    end subroutine press
end module sw
```

```
>>> import numpy as N
>>> import seawater
>>> lat = 35.
>>> z = N.array([5.,10.,25.,50.,100.,250.,/
...     500.,1000.,2000.,3000.])
>>> p = seawater.sw.press(z,lat,len(z))
>>> for n in range(len(z)):
...     print '%9.3f %9.3f'%(z[n], p[n])
...
    5.000     5.043
   10.000    10.073
   25.000    25.190
   50.000    50.394
  100.000   100.788
  250.000   252.066
  500.000   504.414
 1000.000  1009.975
 2000.000  2024.535
 3000.000  3043.748
```

```
linux> f2py -c --fcompiler=intel --f90flags="-tpp7 -mp" -m seawater seawater.f90
```

C/C++

- SWIG
  - Simplified Wrapper & Interface Generator
  - Extension wrapper
  - Easy to use & well documented
- Pyrex
  - Python-like language for writing extension modules
- Weave (SciPy)
  - Inlining C/C++ code within Python code
  - Conversion between C/C++ types & Python objects

SWIG Example

fibonaci.c

```
#include <stdio.h>
#include <stdlib.h>
int fib(int N){
    int i, a[N];
    for (i=0 ; i<N ; ++i){
        if(i==0)
            a[i] = 0;
        else if(i==1)
            a[i] = 1;
        else
            a[i] = a[i-2] + a[i-1];
        printf("%d ",a[i]);
    }
}
```

fibonaci.i

```
%module fibonaci
%
/* Put headers and other declarations here */
extern int fib(int a);
%
extern int fib(int a);
```

```
>>> import fibonaci
>>> fibonaci.fib(8)
0 1 1 2 3 5 8 13 8
```

```
linux> swig -python fibonaci.i
linux> gcc -c -fpic fibonaci.c fibonaci_wrap.c -I/usr/include/python2.4
linux> gcc -shared  fibonaci.o fibonaci_wrap.o -o _fibonaci.so
```

IPython

- Enhanced Python shell for interactive work
- TAB-completion, aliases
- Magic commands (%)
  - %pdoc, %pdef, %psource, %pfile, *objectname*?
  - %run
  - %hist, %logstart
  - %pdb (debugger)
  - %run -p (profiler)
- Recall previous results (_, __ and ___)
- IPython + NumPy + Matplotlib (ipython -pylab)

Specialized Tools

- Image Analysis & Visualization:
  - PIL
  - VTK
- Parallel Computing:
  - Library/Tools: python-mpi, PyACTS
  - Interpreters: lampython, mpichpython, pyMPI
- Biology:
  - Modeling & Analysis of Dynamical Systems: PyDSTool, SimPy
  - Bioinformatics & Computational Molecular Biology: BioPython, PySat, PyPhy
- Molecular Modeling & Visualization: PyMOL, MDTools
- Climate Modeling: CDAT, CDMS
- GIS: Thuban, PCL
- Lots more . . . see **`http://www.python.org`**

Summary

- Python is a free & open source language that offers an environment for numerical/scientific computing equivalent (better!?) to those provided by expensive commercial software such as Matlab™ and IDL™.
- In addition to numerical/scientific computing, Python has numerous external libraries and extensions that make it well suited for a wide range of applications including: Web/CGI and GUI development, database access, HTML/XML/SGML, OS extensions and network programming, etc.

Resources

- Python ..... http://www.python.org
- NumPy ..... http://numpy.scipy.org
- SciPy ..... http://www.scipy.org
- Matplotlib ..... http://matplotlib.sourceforge.net
- PyNGL ..... http://www.pyngl.ucar.edu
- IPython ..... http://ipython.scipy.org/moin
- netCDF4 . http://code.google.com/p/netcdf4-python
- WHOI Python users ..... python-users@whoi.edu