SAND2016-5089PE

Exceptional service in the national interest

Sandia National Laboratories

A Crash Course in Python for Scientists

Rick Muller

Center for Computing Research

Sandia National Laboratories

Available as an IPython notebook:

http://nbviewer.jupyter.org/gist/rpmuller/5920182

CCR Center for Computing Research

U.S. DEPARTMENT OF ENERGY | National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

The story behind the class

Sandia National Laboratories

- When I was a grad student, we wrote Fortran
- When I returned, few computational materials scientists were coding
- How can we lower the bar to computational exploration?

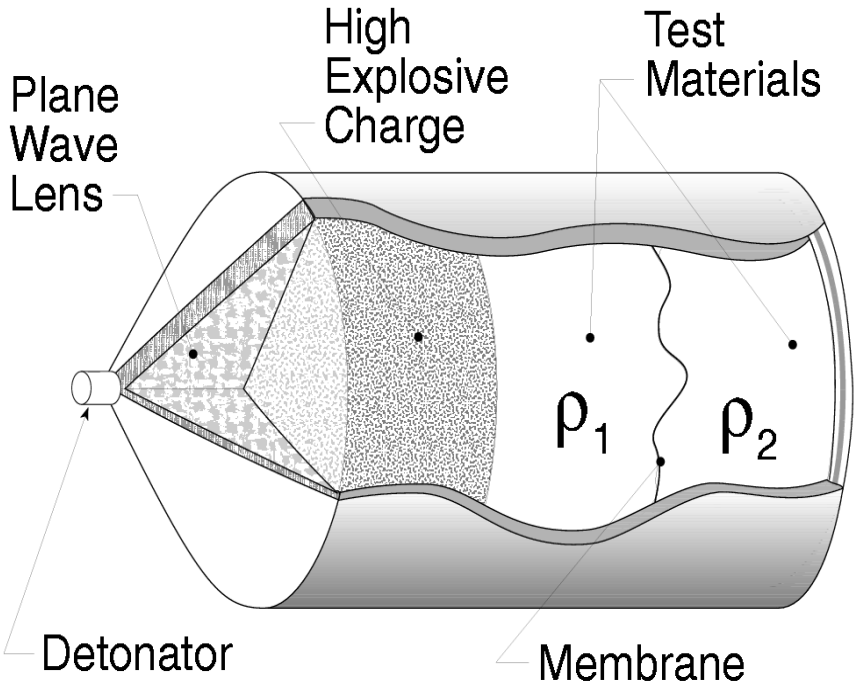
```
c neogvb code:
if ( n_core .gt. 0 ) then
   iham = iham + 1
   do mo = 1, n_core
      ij = 1
      do i = 1, nbf
         call daxpy ( i, trans(i,mo),
                        trans(1,mo), 1,
                        density(ij,iham), 1 )
         ij = ij + i
      enddo
   enddo
endif
```

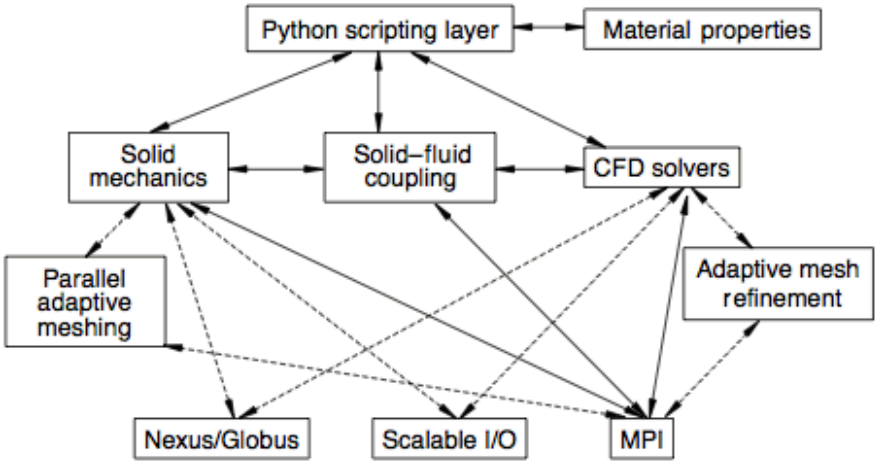
```
# pyquante2 code:
d = np.dot(c[:,:nc],c[:,:nc].T)
if nopen > 0:
    d += 0.5*np.dot(c[:,nc:(nc+no)],
                    c[:,nc:(nc+no)].T)
```

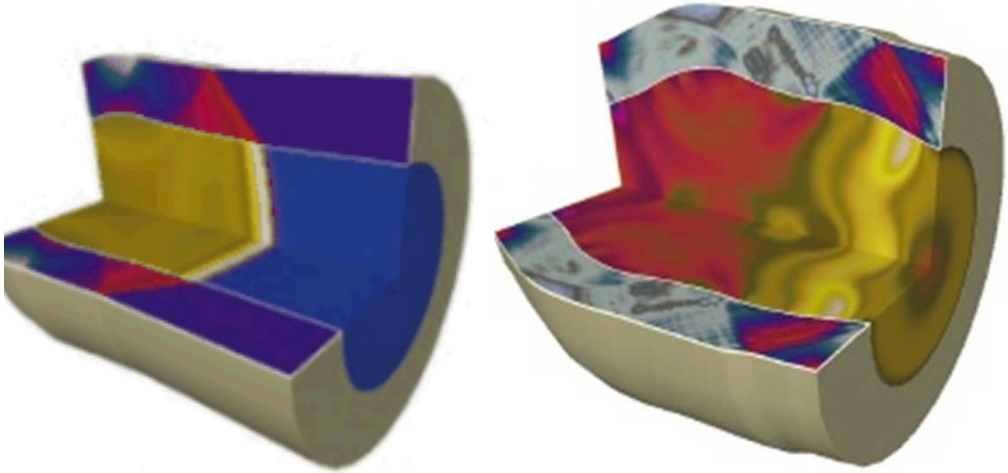
Caltech Virtual Test Facility (1998-2010)

Sandia National Laboratories

- ASCI/ASAP team to simulate shock physics
- Integrate explosives with mechanics using Python
- Could we use similar ideas in my group?





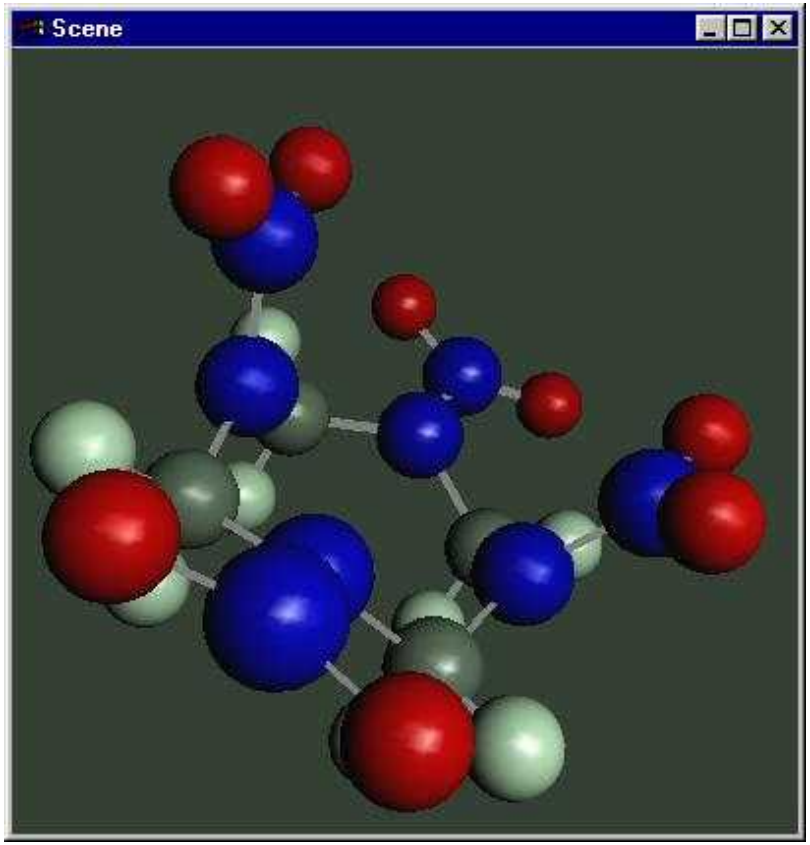


Aivazis et al., Comp. in Sci & Eng., 42 (2000).

Python Short Course

Sandia National Laboratories

Lecture 1: Python Overview
Lecture 2: Numerical Python
Lecture 3: Practical Python
Lecture 4: Object Oriented Programming
Lecture 5: Extending Python with C
Lecture 6: Python GUIs with Tkinter
Lecture 7: 3D Graphics with PyOpenGL



- Spring 2000, Beckman Institute, Caltech
  - Hence my fascination with drop-shadows!
- 2002, one-day course, Lectures 1, 2, 6, 7
- Currently IPython Notebook:
  - http://nbviewer.jupyter.org/gist/rpmuller/5920182
  - Part of an effort to move our experimental condensed matter physics framework to Python
- Also taught to 9- and 10-year olds!

Today's Talk: Why Python for Scientific Exploration

Sandia National Laboratories

- Not the original intended audience
  - Try to touch on language basics
  - Couple of advanced topics that may interest veterans

- Easy to Learn: Computer Programming For Everyone
- Readability Counts: Code is Conversation
- Code Ecosystem: Stand on the Shoulders of Giants
- Speed & Parallelization: Premature Optimization…

Today's Talk: Why Python?

Sandia National Laboratories

- Easy to Learn: Computer Programming For Everyone
- Readability Counts: Code is Conversation
- Code Ecosystem: Stand on the Shoulders of Giants
- Speed & Parallelization: Premature Optimization…

Computer Programming for Everyone

Sandia National Laboratories

- Python is an easy to learn
  - You don't need to be a computer scientist to program. Anyone can learn to program in Python. I've taught it to 9-10 year olds. I've taught it to experimental physicists.
  - You can learn (much of) Python in a day.
  - Python tutorial: http://docs.python.org/2/tutorial
  - This class: http://nbviewer.jupyter.org/gist/rpmuller/5920182
- At the same time, Python is Powerful
  - You don't need to be a computer scientist to get all of the cool stuff that computer scientists do all day
  - Easy to use data structures, containers, syntax
  - Multiple programming styles: procedural, object-oriented, functional
  - Lots of bells and whistles included in stdlib: web server, multiprocessing, math, json, databases, XML, regex

Today's Talk: Why Python?

Sandia National Laboratories

- Easy to Learn: Computer Programming For Everyone
- Readability Counts: Code is Conversation
- Batteries Included: Stand on the Shoulders of Giants
- Speed & Parallelization: Premature Optimization…

Code is conversation

Sandia National Laboratories

- Code is conversation
  - Your ideas aren't any good if you can't share them
- Python produces readable code
  - Whitespace sensitive yields standard look and feel
  - Simple syntax makes understanding other people's ideas easy. Code follows concepts.

Solving a quantum HO in Python

$$V(x) = \frac{1}{2}m\omega^2 x^2$$

```
x = linspace(-3,3)
m = 1.0
ohm = 1.0
T = (-0.5/m)*Laplacian(x)
V = 0.5*(ohm**2)*(x**2)
H =  T + diag(V)
E,U = eigh(H)
```

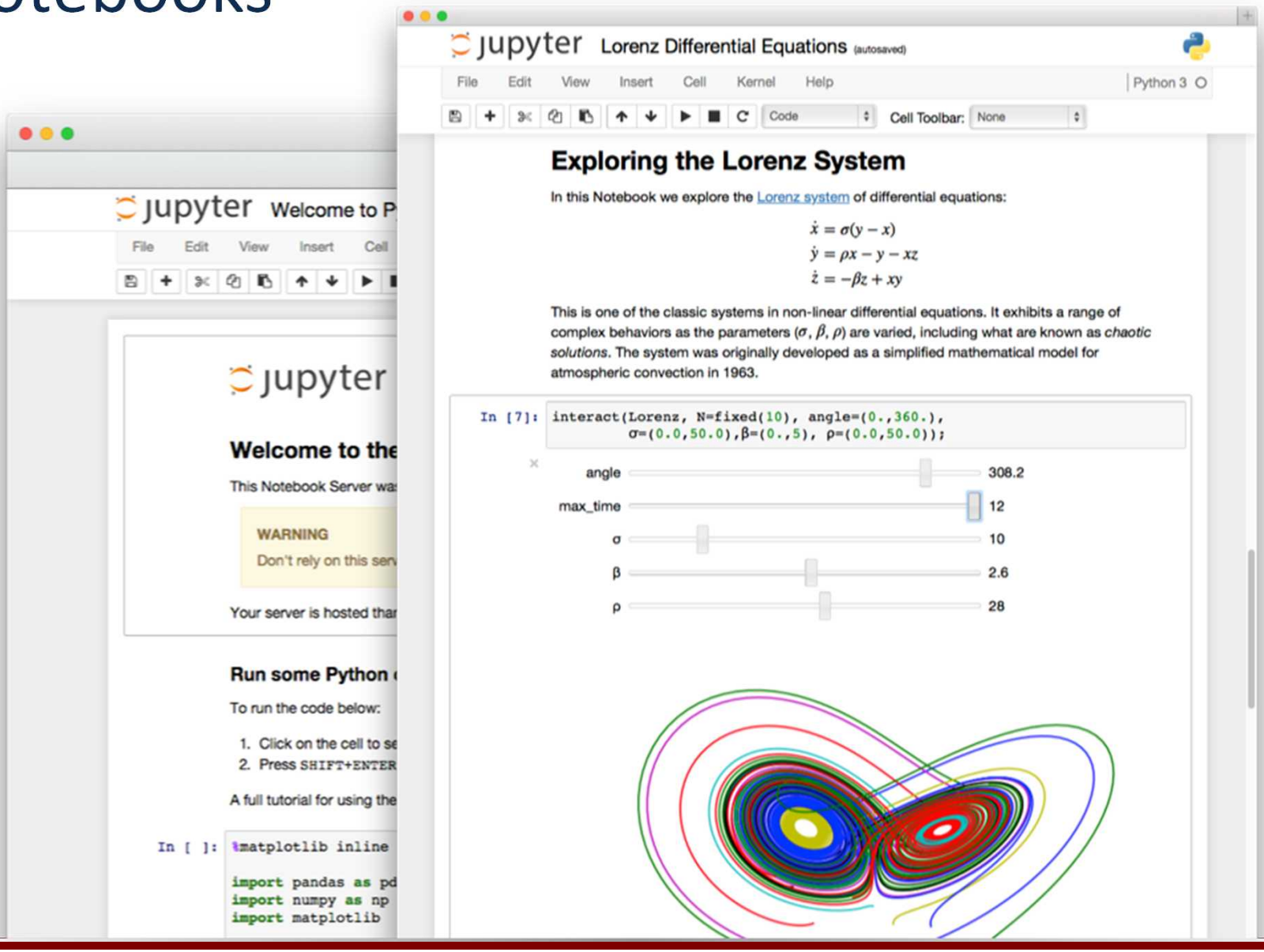
$$-\frac{\hbar^2}{2m}\frac{\partial^2\psi(x)}{\partial x^2} + V(x)\psi(x) = E\psi(x)$$

$$y'' = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

```
def Laplacian(x):
    h = x[1]-x[0]
    n = len(x)
    M = -2*identity(n,'d')
    for i in range(1,n):
        M[i,i-1] = M[i-1,i] = 1
    return M/h**2
```

IPython/Jupyter makes readable notebooks

Sandia National Laboratories



A Crash Course in Python for Scientists

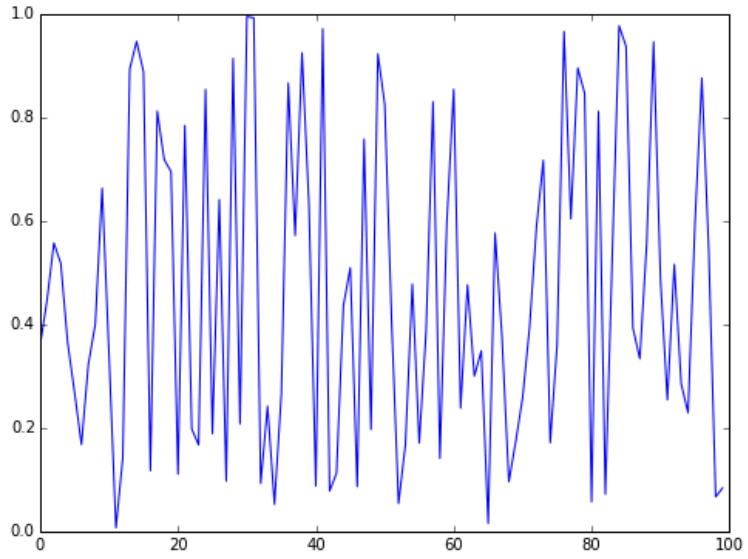
Rick Muller, Sandia National Laboratories

Monte Carlo, random numbers, and computing π

Many methods in scientific computing rely on Monte Carlo integration, where a sequence of (pseudo) random numbers are used to approximate the integral of a function. Python has good random number generators in the standard library. The **random()** function gives pseudorandom numbers uniformly distributed between 0 and 1:

In [129]:
```
from random import random
rands = []
for i in range(100):
    rands.append(random())
plot(rands)
```

Out[129]:
```
[<matplotlib.lines.Line2D at 0x10f9bf210>]
```



Today's Talk: Why Python?

Sandia National Laboratories

- Easy to Learn: Computer Programming For Everyone
- Readability Counts: Code is Conversation
- Code Ecosystem: Stand on the Shoulders of Giants
- Speed & Parallelization: Premature Optimization…

Python stdlib: Batteries Included

Sandia National Laboratories

- Broad standard library includes
  - Web servers
  - String manipulation functions for parsing text, including unicode
  - Math and statistics
  - Databases, json, XML.
- Large number of data structures
  - Linked lists
  - Dictionaries
  - Binary Trees
  - Sets
  - Queues
  - Many others

Survey of other modules

Sandia National Laboratories

- **Numpy** for multidimensional arrays and linear algebra
- **Matplotlib** for publication quality plotting and graphics
- **Scipy** a large selection of numerical routines that uses Numpy
- **Sympy** for symbolic manipulation of math/physics
- **Pandas** for data science
- **Mayavi** for 3D graphics (also Pyglet for OpenGL)
- **Scikit-learn** for machine learning
- **pymc** Markov chain Monte Carlo
- **Pyquante** for quantum chemistry (mine)
- **GMPY2** interface to the Gnu multiprecision library
- **NetworkX** for networks and graph theory
- **PyTrilinos** interface to Sandia parallel Trilinos toolkit
- **mpi4py** interface to MPI
- **PySpark** interface to Apache Spark Map-reduce framework

Numpy Overview

Sandia National Laboratories

- Matlab-like array module for numerics
  - Construct arrays
  - Slicing
  - Array operations
  - Linear algebra
- Fast, easy to use
  - Written in C, linked to optimized BLAS.

```
x = linspace(-3,3)
m = 1.0
ohm = 1.0
T = (-0.5/m)*Laplacian(x)
V = 0.5*(ohm**2)*(x**2)
H =  T + diag(V)
E,U = eigh(H)
```

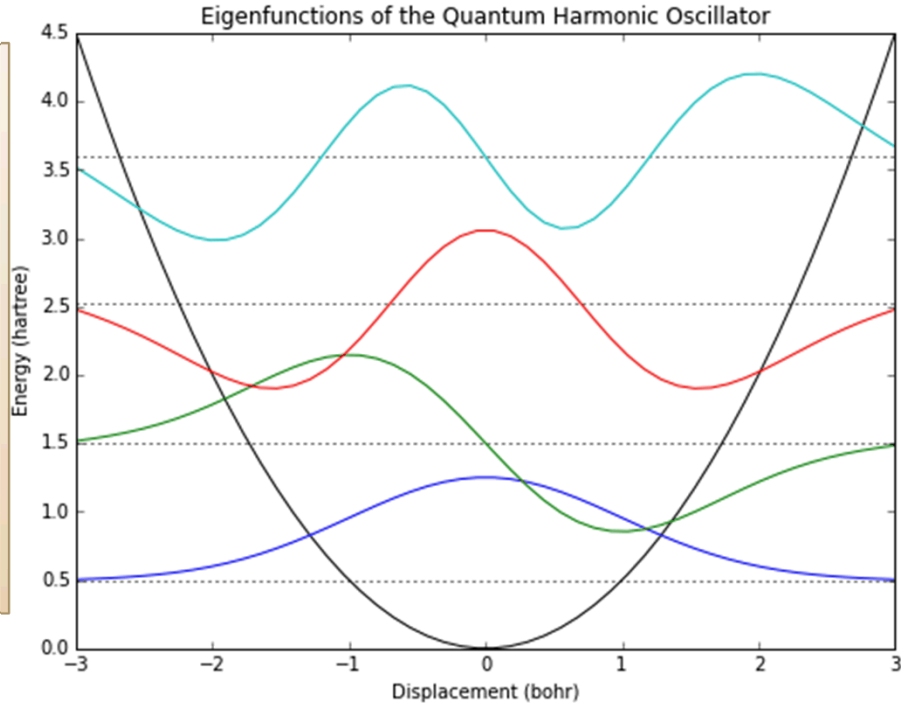
```
def Laplacian(x):
    h = x[1]-x[0]
    n = len(x)
    M = -2*identity(n,'d')
    for i in range(1,n):
        M[i,i-1] = M[i-1,i] = 1
    return M/h**2
```

Plotting solutions using Matplotlib

Sandia National Laboratories

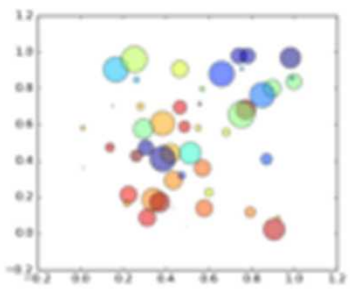
- There's also a matplotlib module for easy plotting
  - Essentially copied the Matlab API

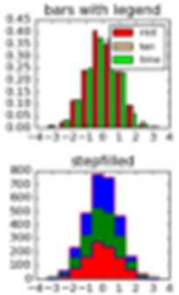
```
# Plot the Harmonic potential
plot(x,V,color='k')
for i in range(4):
    # Plot the energy level:
    axhline(y=E[i],color='k',ls=":")
    # And the eigenfunction
    plot(x,-U[:,i]/sqrt(h)+E[i])
title("Eigenfunctions of the HO")
xlabel("Displacement (bohr)")
ylabel("Energy (hartree)")
```

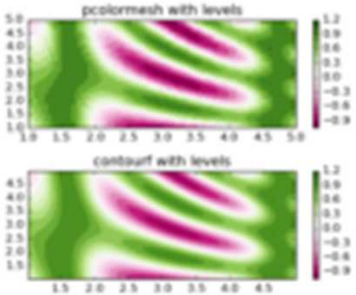


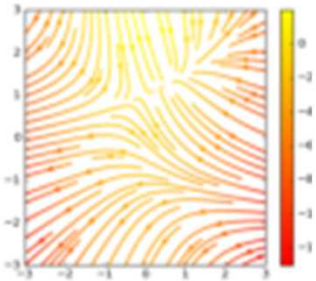
Matplotlib

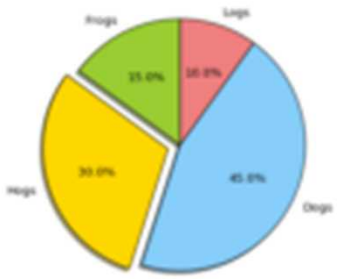
Sandia National Laboratories

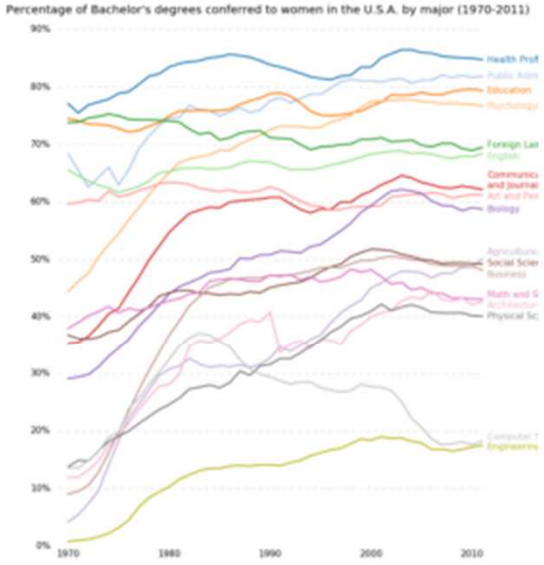


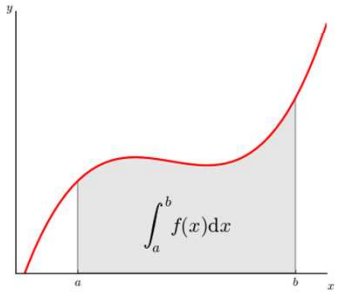


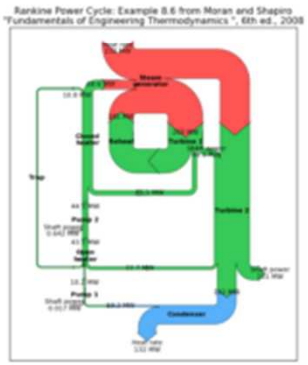


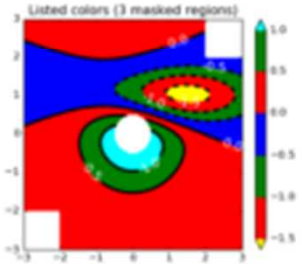


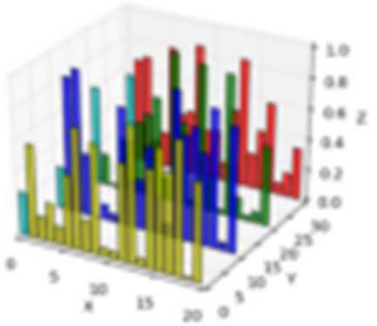


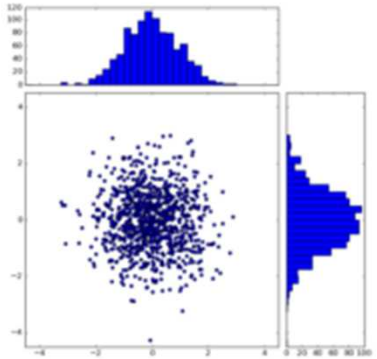


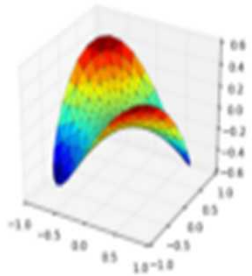












Scipy Overview

Sandia National Laboratories

- Scipy is a suite of Fortran/C codes numeric codes that have been integrated using Numpy arrays.
  - Integration
  - Optimization
  - Interpolation
  - Sparse matrices and eigenvalues
  - Graph theory
  - Spatial decompositions
  - Image processing

Optimization

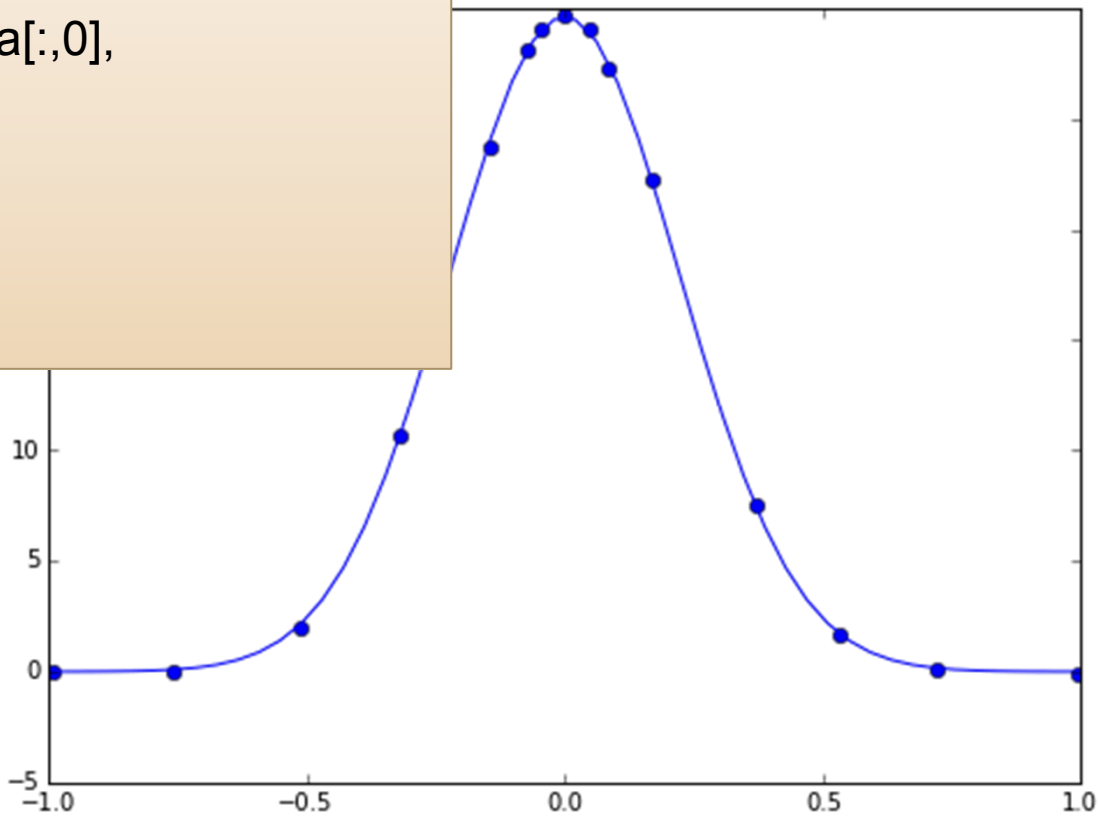
- Scipy.optimize has a very nice interface to optimization

```
from scipy.optimize import curve_fit

def gauss(x,A,a): return A*exp(a*x**)

params,conv = curve_fit(gauss,data[:,0],
  data[:,1])
x = linspace(-1,1)
plot(data[:,0],data[:,1],'bo')
A,a = params
plot(x,gauss(x,A,a),'b-')
```

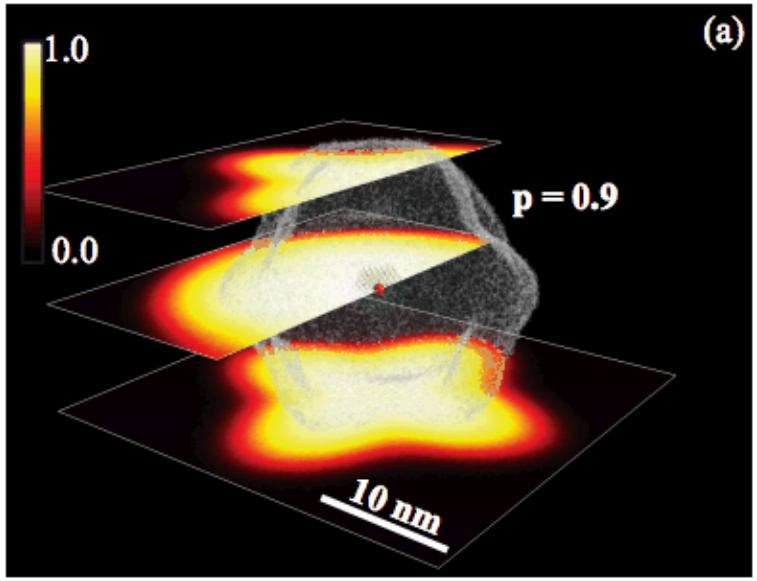
Do you really want to rewrite Levenberg-Marquardt? Or Fletcher-Powell?



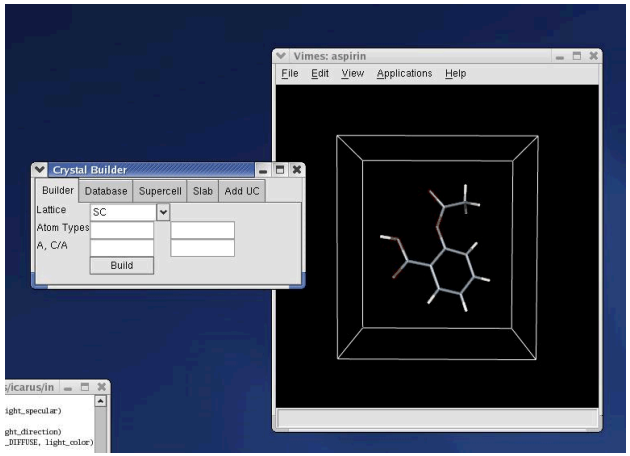
Visualization, Graphics, GUIs

Sandia National Laboratories

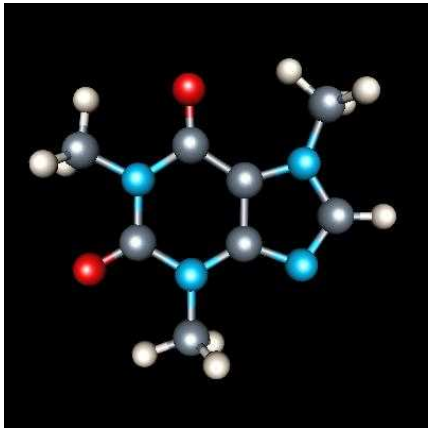
MayaVi

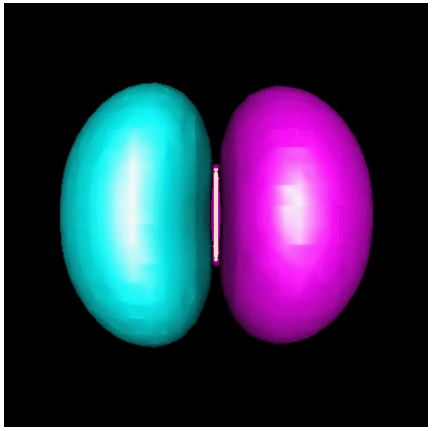


GUI Toolkits



OpenGL





Today's Talk: Why Python?

Sandia National Laboratories

- Easy to Learn: Computer Programming For Everyone
- Readability Counts: Code is Conversation
- Code Ecosystem: Stand on the Shoulders of Giants
- Speed & Parallelization: Premature Optimization…

Premature Optimization

Sandia National Laboratories

- Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.
  - Donald Knuth, “Structured Programming with Go To Statements,” 1974

- Python can be 5-10 times slower than compiled C/Fortran. However:
  - You can easily find out what parts are slow (profiling)
  - You can speed these parts using JIT compilers
  - You can use C/Fortran for these parts, either using a compiled library (Numpy/Scipy) or by writing them in C, Fortran, Cython
  - You can parallelize the code using multiprocessing or mpi4py.

Profiling: Find out what is slow

Sandia National Laboratories

- Python profiling makes it easy to figure out where code is slow

```
import cProfile,pstats
from pyquante2 import rhf,c6h6,basisset

def main():
    bfs = basisset(c6h6,'sto-3g')
    solver = rhf(hmx,bfs)
    solver.converge()
    return

if __name__ == '__main__':
    cProfile.run('main()','profile.dat')
    prof = pstats.Stats('profile.dat')
    prof.strip_dirs().sort_stats('time').print_stats(15)
```

```
 ncalls  tottime  percall  cumtime  percall filename:lineno(function)
  222800   80.894    0.000   81.199    0.000 {pyquante2.ctwo.ERI_hgp}
       1    1.354    1.354   83.423   83.423 integrals.py:90(__init__)
   18648    0.900    0.000    1.684    0.000 {pyquante2.cone.V}
  320832    0.550    0.000    0.845    0.000 cgbf.py:44(__getitem__)
  891200    0.305    0.000    0.305    0.000 cgbf.py:54(cne_list)
  222801    0.288    0.000    0.293    0.000 integrals.py:140(iiterator)
```

Writing/linking code in C or Fortran

Sandia National Laboratories

- You can write routines in C, and wrap them in python, also using C.
- New methods make this much easier
  - Load/prototype from DLL library
  - Cython/f2py/SWIG interface generators

```
#include "Python.h"

static PyObject *py_cdistance(PyObject *self,
                              Pyobject *args){
    double xi,yi,zi,xj,yj,zj,dist;
    PyArg_ParseTuple(args,"dddddd",&xi,&yi,&zi,
                                   &xj,&yj,&zj);
    dist = c_dist_function(xi,yi,zi,xj,yj,zj);
    return Py_BuildValue("d",dist);
}
```

```
from ctypes import*

mydll = cdll.LoadLibrary(”dist.dll")
dist = mydll.dist(0.,0.,0.,1.,1.,1.)
```

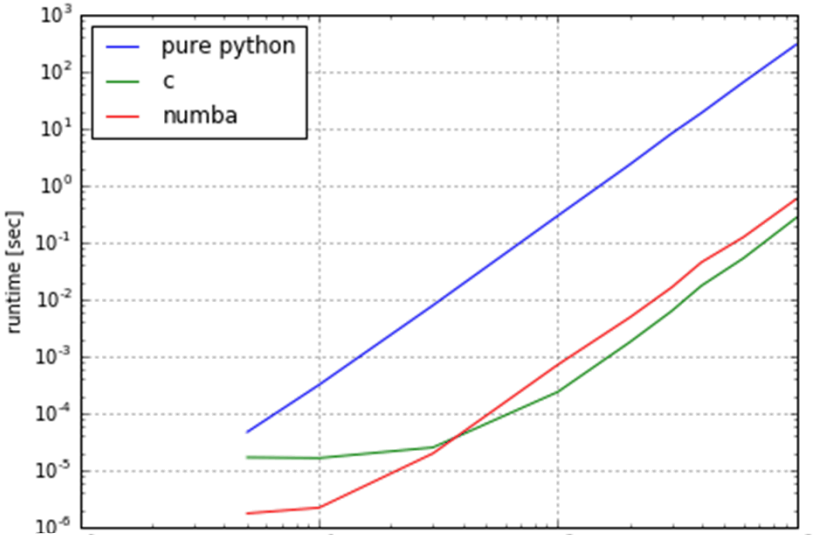
Other methods for speed

- Cython: A Python-like language with explicit types
- PyPy: Optimized Python written in Python
- Numba:
  - JIT compiler
  - Compiles down to LLVM IR

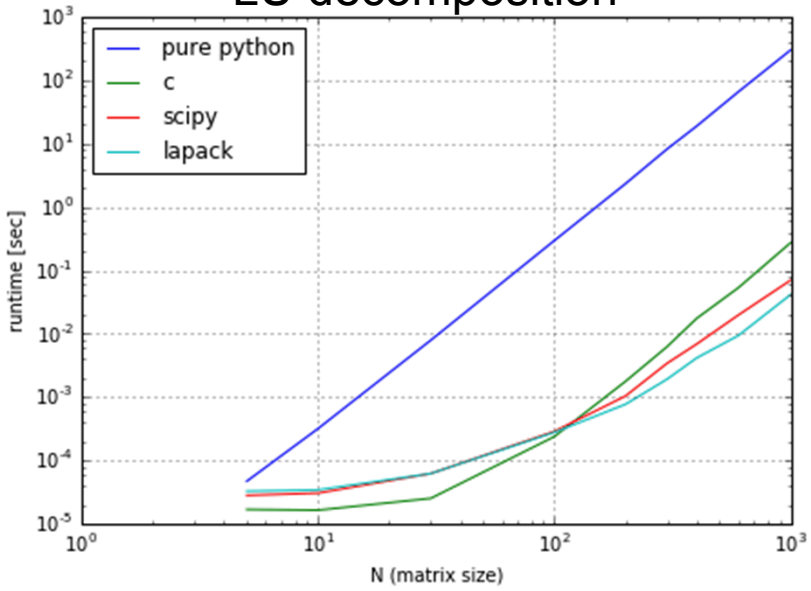
```
from numba import jit
from numpy.random import randn

@jit
def f(x):
    return x*x

if __name__ == '__main__':
    f(randn(1000))
```



LU decomposition



A Speed Comparison Of C, Julia, Python, Numba, and Cython on LU Factorization, Jean Francois Puget, IBM Developer Blog

Threads, processing, parallel programming

Sandia National Laboratories

- Global Interpreter Lock (GIL) makes threaded programming hard
  - Necessary for interpreted language
  - Many different solutions proposed, none have stuck
- Multiprocessing module has many of these paradigms
- MPI frameworks available
  - mpi4py
  - PyTrilinos
- What I really want is something that has Chapel-like abstractions for parallel programming
  - Not totally unlike Trilinos's maps

```
from deco import concurrent, synchronized

@concurrent
def slow(index):
    time.sleep(5)

@synchronized
def run():
    for index in list('123'):
        slow(index)
run()
```

```
const MessageSpace = {1..numMessages}
                          dmapped Cyclic(startIdx=1);

forall msg in MessageSpace do
  writeln("Hello, world! (from iteration ", msg, " of ",
          numMessages, " owned by locale ",
here.id,
          " of ", numLocales, ")");
```

Today's Talk: Why Python?

Sandia National Laboratories

- Easy to Learn: Computer Programming For Everyone
- Readability Counts: Code is Conversation
- Code Ecosystem: Stand on the Shoulders of Giants
- Speed & Parallelization: Premature Optimization…

Conclusion

Sandia National Laboratories

```
>>> import this
```

The 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.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Backup

Sandia National Laboratories

Data structures: Lists (skip)

Sandia National Laboratories

- Keep a group of similar items together. Built-in functions allow access to size and other features:

```
>>> days_of_the_week = ["Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"]
>>> len(days_of_the_week)
7
```

- Access members of the list using indexing. Like C, start from 0.

```
>>> days_of_the_week[2]
'Tuesday'
```

- Convenience function for lists of numbers

```
>>> range(5)
[0, 1, 2, 3, 4, 5]
```

- Other data types: tuples, dictionaries, …

Iteration, indentation, blocks (skip)

Sandia National Laboratories

- **for** iterates over lists (or iterators)

```
for day in days_of_the_week:
    statement = “Today is “ + day
    print statement
```

- This is where the **range** function is useful for flow control:

```
for i in range(20):
    print “the square of “,i,” is “,i*i
```

- Python is whitespace sensitive. It uses indentation to define blocks.

Boolean testing, if statements (skip)

Sandia National Laboratories

- Booleans and **if** control branching in programs

```
if day == “Sunday”:
    print “Sleep in”
else:
    print “Go to work”
```

- Booleans are formed using typical comparisons

```
1 == 2              # False
50 == 2*25          # True
3 < 3.14159         # True
1 == 1.0            # True
1 is 1.0            # False
1 != 0              # True
[1,2,3] < [1,2,4]   # True
```

Functions (skip)

Sandia National Laboratories

- You can of course define functions:

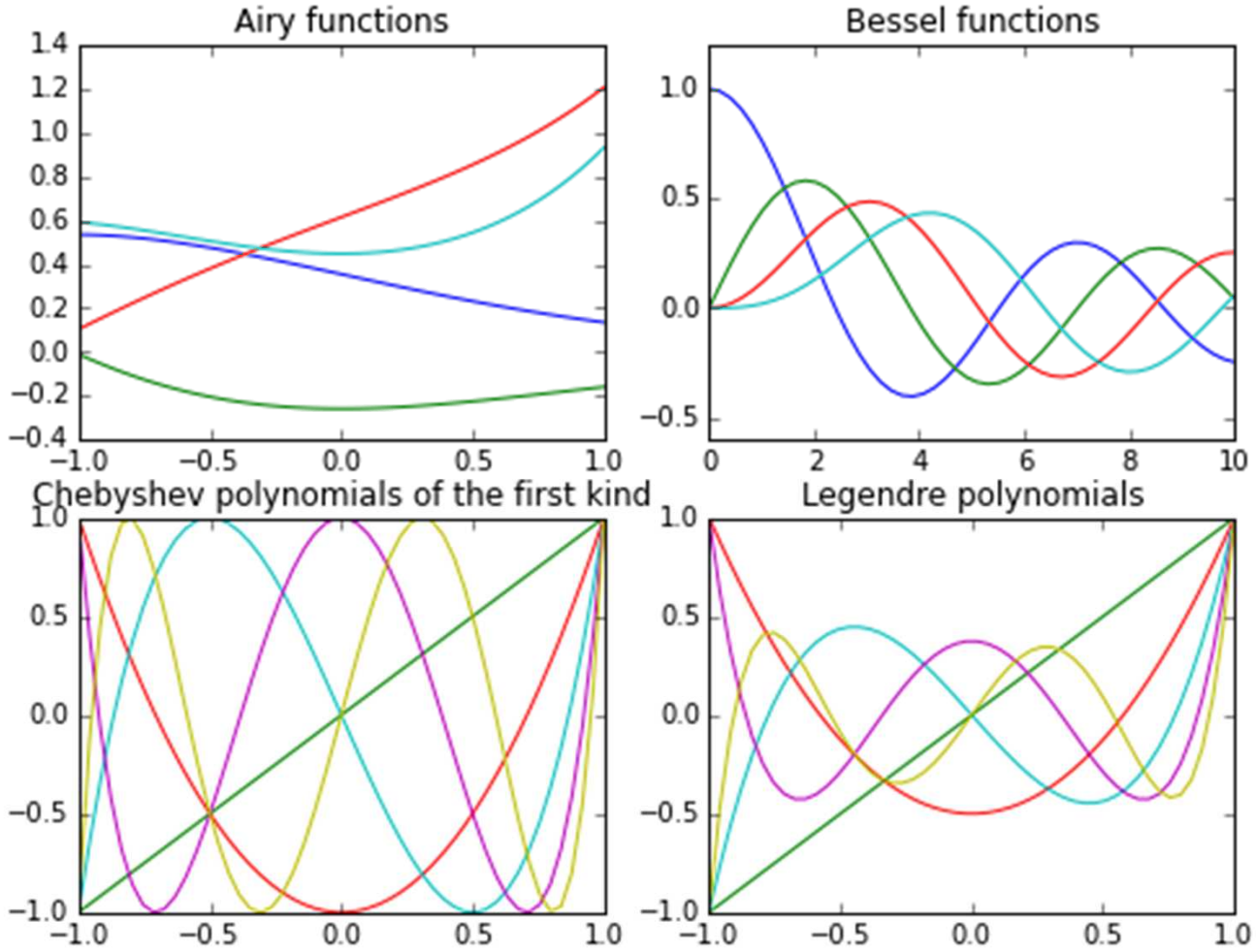
```
def fibonacci(sequence_length):
    "Return the Fibonacci sequence of length *sequence_length*"
    sequence = [0,1]
    if sequence_length < 1:
        print "Fibonacci sequence only defined for length 1 or greater"
        return
    if 0 < sequence_length < 3:
        return sequence[:sequence_length]
    for i in range(2,sequence_length):
        sequence.append(sequence[i-1]+sequence[i-2])
    return sequence

>>> fibonacci(12)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> help(fibonacci)
Help on function fibonacci in module __main__:

fibonacci(sequence_length)
    Return the Fibonacci sequence of length *sequence_length*
```

Special Functions

- Numpy has many special functions built in. Scipy has more.



Python quantum chemistry

Sandia National Laboratories

```
def rhf_simple(geo,basisname='sto3g',maxiter=25,verbose=False):
    bfs = basisset(geo,basisname)
    i1 = onee_integrals(bfs,geo)
    i2 = twoe_integrals(bfs)
    h = i1.T + i1.V
    E,U = geigh(h,i1.S)
    Enuke = geo.nuclear_repulsion()
    nocc = geo.nocc()
    Eold = Energy = 0

    for i in xrange(maxiter):
        D = dmat(U,nocc)
        Eone = trace2(h,D)
        G = i2.get_2jk(D)
        H = h+G
        Etwo = trace2(H,D)
        E,U = geigh(H,i1.S)
        Energy = Enuke+Eone+Etwo
        if np.isclose(Energy,Eold):
            break
        Eold = Energy
    return Energy,E,U
```

Iterators/generators: Don't do slow stuff (skip)

Sandia National Laboratories

- Lists are constructed in a single step. This makes nested loops very slow.

```
for i in range(10000):
    for j in range(100000):
        print i+j
```

- Iterators, generators, and the itertools module adopt Haskell-style lazy evaluation and iteration

```
for i in xrange(10000):
    for j in xrange(100000):
        print i+j
```

```
import itertools

for i,j in itertools.product(100000,repeat=2):
    print i,j
```

- In many cases, the resulting code can be extremely fast