Numerical Computing in Python

A Guide for Matlab Users

B. Blais

Department of Science and Technology
Bryant University

Faculty Development Seminar - Spring 2007

Abstract

Matlab is a commercial program used extensively in the scientific and business communities. There are many reasons why it is very popular, including its interactive structure, clean syntax, and ability to interface with fast compiled languages, like C. It also has many routines for signal and image processing, optimization, and visualization.

Python is a modern language used extensively by Google and NASA, as well as many others. Like Matlab, it also has an interactive structure, clean syntax, and the ability to interface with fast compiled languages, like C. There are modules in Python for doing numerical work and visualization, and thus one can make a Python-based computational environment with much the same feel as Matlab. Python is also free, is far more versatile, and can be used in many more applications than Matlab, including robotics, web frameworks, text processing, and others. It is particularly good as a first language, and I have found it personally very useful in my classes.

This Faculty Development Seminar uses a “how-to” approach to setting up Python as a computational environment, geared towards current users of Matlab or similar environments. It explores specific applications of numerical computing, and highlights the power of using Python both in research and in teaching. The seminar will explore my own experiences of the past year, converting from a die-hard Matlab fan to a Python enthusiast.

Outline

Where am I Coming From?

- 1980-1988: The BASIC Years
- 1989-1993: The Pascal Years (with a little Fortran)
- 1994-1996: The C/C++ Years
- 1995-2006: The Matlab Years (with C for cmex)
- 2003-2006: The Disenchantment Years
- 2006-present: The Python Year(s)

Where am I Coming From?

- 1980-1988: The BASIC Years
- 1989-1993: The Pascal Years (with a little Fortran)
- 1994-1996: The C/C++ Years
- 1995-2006: The Matlab Years (with C for cmex)
- 2003-2006: The Disenchantment Years
- 2006-present: The Python Year(s)

Where am I Coming From?

- 1980-1988: The BASIC Years
- 1989-1993: The Pascal Years (with a little Fortran)
- 1994-1996: The C/C++ Years
- 1995-2006: The Matlab Years (with C for cmex)
- 2003-2006: The Disenchantment Years
- 2006-present: The Python Year(s)

Where am I Coming From?

- 1980-1988: The BASIC Years
- 1989-1993: The Pascal Years (with a little Fortran)
- 1994-1996: The C/C++ Years
- 1995-2006: The Matlab Years (with C for cmex)
- 2003-2006: The Disenchantment Years
- 2006-present: The Python Year(s)

Where am I Coming From?

- 1980-1988: The BASIC Years
- 1989-1993: The Pascal Years (with a little Fortran)
- 1994-1996: The C/C++ Years
- 1995-2006: The Matlab Years (with C for cmex)
- 2003-2006: The Disenchantment Years
- 2006-present: The Python Year(s)

Where am I Coming From?

- 1980-1988: The BASIC Years
- 1989-1993: The Pascal Years (with a little Fortran)
- 1994-1996: The C/C++ Years
- 1995-2006: The Matlab Years (with C for cmex)
- 2003-2006: The Disenchantment Years
- 2006-present: The Python Year(s)

Where am I Coming From?

- 1980-1988: The BASIC Years
- 1989-1993: The Pascal Years (with a little Fortran)
- 1994-1996: The C/C++ Years
- 1995-2006: The Matlab Years (with C for cmex)
- 2003-2006: The Disenchantment Years
- 2006-present: The Python Year(s)

What is Python?

- Flexible, powerful language
- Multiple programming paradigms
- Easy, clean syntax
- Large community of support
- “Batteries included”
- Free as in “free beer”
- Free as in “free speech”

Non-Numerical Projects I've Done with Python

- Making Aemilia's and Aoife's web-page

  `web.bryant.edu/~bblais/gallery`

- Curriculum Committee web-page
- Football Statistics
- Student Picture Game
- Posting Grades
- Robot Simulator
- Robot Programming Language

Numerical Projects I've Done with Python

- Neural Simulators
  - *Plasticity*: rate-based
  - *Splash*: spike-based
- Physics Projects
  - Simulating falling objects
  - Simulating flipping coins
  - Signal processing for SETI
- AI and Robotics Projects
  - Analysis of finance data
  - Voice recognition
- Mechanisms of the Mind Projects:
  - Supervised and Unsupervised Learning
  - Associative Networks
- Bayesian Statistical Inference Notes

Example Python Code

```
from math import sin,pi

def sinc(x):
    '''Compute the sinc function:
        sin(pi*x)/(pi*x)'''

    try:
        val = (x*pi)
        return sin(val)/val
    except ZeroDivisionError:
        return 1.0


input=[0,0.1,0.5,1.0] # list of input values
output=[sinc(x) for x in input]

print output
```

Packages for a Useful Computational Environment

- Minimum

python	-	the base language
numpy	-	array class, numerical routines
scipy	-	higher level scientific routines (depends on numpy)
matplotlib	-	visualization
ipython	-	a more flexible python shell

Packages for a Useful Computational Environment

- Minimum

python	-	the base language
numpy	-	array class, numerical routines
scipy	-	higher level scientific routines (depends on numpy)
matplotlib	-	visualization
ipython	-	a more flexible python shell

Packages for a Useful Computational Environment

- Minimum

python	-	the base language
numpy	-	array class, numerical routines
scipy	-	higher level scientific routines (depends on numpy)
matplotlib	-	visualization
ipython	-	a more flexible python shell

Packages for a Useful Computational Environment

- Minimum

python	-	the base language
numpy	-	array class, numerical routines
scipy	-	higher level scientific routines (depends on numpy)
matplotlib	-	visualization
ipython	-	a more flexible python shell

Packages for a Useful Computational Environment

- Minimum

python	-	the base language
numpy	-	array class, numerical routines
scipy	-	higher level scientific routines (depends on numpy)
matplotlib	-	visualization
ipython	-	a more flexible python shell

Packages for a Useful Computational Environment

- Useful

pyrex	-	writing fast compiled extensions (like cmex, but way better)
wxpython	-	GUI library
pywin32	-	Windows COM Interface
BeautifulSoup	-	HTML Parser
xlrd, pyXLWriter	-	Reading/Writing Excel Spreadsheets

Packages for a Useful Computational Environment

- Useful

pyrex	-	writing fast compiled extensions (like cmex, but way better)
wxpython	-	GUI library
pywin32	-	Windows COM Interface
BeautifulSoup	-	HTML Parser
xlrd, pyXLWriter	-	Reading/Writing Excel Spreadsheets

Packages for a Useful Computational Environment

- Useful

pyrex	-	writing fast compiled extensions (like cmex, but way better)
wxpython	-	GUI library
pywin32	-	Windows COM Interface
BeautifulSoup	-	HTML Parser
xlrd, pyXLWriter	-	Reading/Writing Excel Spreadsheets

Packages for a Useful Computational Environment

- Useful

pyrex	-	writing fast compiled extensions (like cmex, but way better)
wxpython	-	GUI library
pywin32	-	Windows COM Interface
BeautifulSoup	-	HTML Parser
xlrd, pyXLWriter	-	Reading/Writing Excel Spreadsheets

Packages for a Useful Computational Environment

- Useful

pyrex	-	writing fast compiled extensions (like cmex, but way better)
wxpython	-	GUI library
pywin32	-	Windows COM Interface
BeautifulSoup	-	HTML Parser
xlrd, pyXLWriter	-	Reading/Writing Excel Spreadsheets

Functions: Data types and Files

Matlab Code: f2c.m and c2f.m

```
function c=f2c(f)
  c=(f-32)*(100/180);

function f=c2f(c)
  f=(180/100)*c+32;
```

Python Code: convert.py

```
def f2c(f):
    return (f-32)*(100.0/180.0)

def c2f(c):
    return (180.0/100.0)*c+32
```

Interactive Environment

Running the Matlab Code

```
>> a=f2c(212)

a =

   100

>> b=c2f(-40)

b =

   -40
```

Interactive Environment

Running the Python Code: Namespaces

```
In [1]:from convert import *

In [2]:a=f2c(212)

In [3]:a
Out[3]:100.0

In [4]:b=c2f(-40)

In [5]:b
Out[5]:-40.0
```

Interactive Environment

Object-oriented or Procedural

```
In [7]:import convert

In [8]:a=convert.f2c(212)

In [9]:a
Out[9]:100.0

In [10]:dir(convert)
Out[10]:['__builtins__', '__doc__',
         '__file__', '__name__',
         'c2f', 'f2c']
```

Interactive Environment

Object-oriented or Procedural

```
In [1]:from Temperature import *
In [2]:t=Temperature(f=32)
In [3]:print t.c
0.0

In [4]:print t.k
273.15

In [5]:t.c=-40

In [6]:t.k
Out[6]:233.14999999999998

In [7]:t.f
Out[7]:-40.0

In [8]:t.k=350

In [9]:t.c
Out[9]:76.850000000000023

In [10]:t.f
Out[10]:170.33000000000004
```

Interactive Environment

Object-oriented or Procedural

```
class Temperature(object):
    coefficients = {'c': (1.0, 0.0, -273.15), 'f': (1.8, -273.15, 32.0)}
    def __init__(self, **kwargs):
        try:
            name, value = kwargs.popitem( )
        except KeyError:
            name, value = 'k', 0
        setattr(self, name, float(value))
    def __getattr__(self, name):
        try:
            eq = self.coefficients[name]
        except KeyError:
            raise AttributeError, name
        return (self.k + eq[1]) * eq[0] + eq[2]
    def __setattr__(self, name, value):
        if name in self.coefficients:
            # name is c or f  -- compute and set k
            eq = self.coefficients[name]
            self.k = (value - eq[2]) / eq[0] - eq[1]
        elif name == 'k':
            object.__setattr__(self, name, value)
        else:
            raise AttributeError, name
```

Interactive Environment

Object-oriented or Procedural

```
In [1]:from Temperature import *
In [2]:t=Temperature(f=32)
In [3]:print t.c
0.0

In [4]:print t.k
273.15

In [5]:t.c=-40

In [6]:t.k
Out[6]:233.14999999999998

In [7]:t.f
Out[7]:-40.0

In [8]:t.k=350

In [9]:t.c
Out[9]:76.850000000000023

In [10]:t.f
Out[10]:170.33000000000004
```

Getting Help

In Matlab, help is contained in the file

```
>> help fft

 FFT Discrete Fourier transform.
    FFT(X) is the discrete Fourier transform (DFT) of vector X.  For
    matrices, the FFT operation is applied to each column. For N-D
    arrays, the FFT operation operates on the first non-singleton
    dimension.

...
```

Getting Help

In Python, help is contained in the object

```
In [14]:help(fft)
Help on function fft in module numpy.fft.fftpack:

fft(a, n=None, axis=-1)
    fft(a, n=None, axis=-1)

    Return the n point discrete Fourier transform of a. n defaults to
    the length of a. If n is larger than the length of a, then a will
    be zero-padded to make up the difference.  If n is smaller than
    the length of a, only the first n items in a will be used.

...
```

. . . and everything is an object: lists, arrays, functions, integers, etc. . .

Getting Help

In Python, help is contained in the object

```
In [14]:help(fft)
Help on function fft in module numpy.fft.fftpack:

fft(a, n=None, axis=-1)
    fft(a, n=None, axis=-1)

    Return the n point discrete Fourier transform of a. n defaults to
    the length of a. If n is larger than the length of a, then a will
    be zero-padded to make up the difference.  If n is smaller than
    the length of a, only the first n items in a will be used.

...
```

. . . and everything is an object: lists, arrays, functions, integers, etc. . .

Getting Help

Namespace is important for getting Help

```
In [19]:import scipy

In [20]:help(scipy)


Help on package scipy:

NAME
    scipy

FILE
    /usr/local/lib/python2.5/site-packages/scipy/__init__.py

DESCRIPTION
    SciPy --- A scientific computing package for Python
    ===================================================
...
    Available subpackages
    ---------------------
    ndimage          --- n-dimensional image package [*]
    stats            --- Statistical Functions [*]
    signal           --- Signal Processing Tools [*]
...
```

Help

In Python, all assignments are name assignments

```
In [15]:d=fft  # assign a new name

In [16]:help(d)
Help on function fft in module numpy.fft.fftpack:

fft(a, n=None, axis=-1)
    fft(a, n=None, axis=-1)

    Return the n point discrete Fourier transform of a. n defaults to
    the length of a. If n is larger than the length of a, then a will
    be zero-padded to make up the difference.  If n is smaller than
    the length of a, only the first n items in a will be used.
```

. . . so all parameters are pass by reference. We're all adults here.

Help

In Python, all assignments are name assignments

```
In [15]:d=fft  # assign a new name

In [16]:help(d)
Help on function fft in module numpy.fft.fftpack:

fft(a, n=None, axis=-1)
    fft(a, n=None, axis=-1)

    Return the n point discrete Fourier transform of a. n defaults to
    the length of a. If n is larger than the length of a, then a will
    be zero-padded to make up the difference.  If n is smaller than
    the length of a, only the first n items in a will be used.
```

. . . so all parameters are pass by reference. We're all adults here.

Some Useful Help Tips in IPython

- Tab completion, especially for a methods list

```
In [4]:numpy.d<TAB>
numpy.delete     numpy.diagflat   numpy.digitize   numpy.dot        numpy.dstack
numpy.deprecate  numpy.diagonal   numpy.disp       numpy.double     numpy.dtype
numpy.diag       numpy.diff       numpy.divide     numpy.dsplit
```

- '?' notation for help

```
In [4]:numpy.diag?
Type:           function
Base Class:     <type 'function'>
String Form:    <function diag at 0xb618e8b4>
Namespace:      Interactive
File:           /usr/local/lib/python2.5/site-packages/numpy/lib/twodim_base.py
Definition:     numpy.diag(v, k=0)
Docstring:
    returns a copy of the the k-th diagonal if v is a 2-d array
    or returns a 2-d array with v as the k-th diagonal if v is a
    1-d array.
```

Some Useful Help Tips in IPython

- Tab completion, especially for a methods list

```
In [4]:numpy.d<TAB>
numpy.delete     numpy.diagflat   numpy.digitize   numpy.dot        numpy.dstack
numpy.deprecate  numpy.diagonal   numpy.disp       numpy.double     numpy.dtype
numpy.diag       numpy.diff       numpy.divide     numpy.dsplit
```

- '?' notation for help

```
In [4]:numpy.diag?
Type:           function
Base Class:     <type 'function'>
String Form:    <function diag at 0xb618e8b4>
Namespace:      Interactive
File:           /usr/local/lib/python2.5/site-packages/numpy/lib/twodim_base.py
Definition:     numpy.diag(v, k=0)
Docstring:
    returns a copy of the the k-th diagonal if v is a 2-d array
    or returns a 2-d array with v as the k-th diagonal if v is a
    1-d array.
```

Zen

```
In [23]: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!
```

Golden Ratio

Matlab

```
function goldfract(n)
%GOLDFRACT Golden ratio continued
%  fraction.
% GOLDFRACT(n) displays n terms.
p = '1';
for k = 1:n
    p = ['1+1/(' p ')'];
end
p
p = 1;
q = 1;
for k = 1:n
    s = p;
    p = p + q;
    q = s;
end

p = sprintf('%d/%d',p,q)
format long
p = eval(p)
format short
err = (1+sqrt(5))/2 - p
```

Python

```
def goldfract(N):
    """GOLDFRACT(N)
    Golden ratio continued fraction
    Displays N terms."""
    p = '1.0'
    for k in range(N):
        p = '1.0+1.0/(' +p+ ')'
    print p

    p = 1
    q = 1
    for k in range(N):
        s = p
        p = p + q
        q = s

    print '%d/%d' % (p,q)
    p='%f/%f' % (p,q)  # use floats
    p=eval(p)
    print "%.20f" % p
    err = (1+sqrt(5))/2 - p
    print err
```

Fibonacci

Matlab

```
function f = fibonacci(n)
% FIBONACCI Fibonacci sequence
% f = FIBONACCI(n) generates the
%  first n Fibonacci numbers.


f = zeros(n,1);
f(1) = 1;
f(2) = 2;
for k = 3:n
  f(k) = f(k-1) + f(k-2);
end
```

Python

```
def fibonacci(n):
    """FIBONACCI Fibonacci sequence
    """
    from numpy import zeros

    f=zeros(n)
    f[0] = 1
    f[1] = 2
    for k in range(2,n):
        f[k]=f[k-1]+f[k-2]

    return f
```

Fibonacci

Python: Array

```
def fibonacci(n):
    """FIBONACCI Fibonacci sequence
    """
    from numpy import zeros

    f=zeros(n)
    f[0] = 1
    f[1] = 2
    for k in range(2,n):
        f[k]=f[k-1]+f[k-2]

    return f
```

Python: List

```
def fibonacci2(n):
    """FIBONACCI Fibonacci sequence
    """

    f=[1,2] # use a list
    for k in range(2,n):
        f.append(f[k-1]+f[k-2])

    return f
```

- Lists are a little like cell arrays, but more flexible

Financial Data

Get the Data

```
import scipy
import os
import urllib
import datetime

# get the data
start=[1,1,2007]
end=[4,25,2007]

fname='my_yahoo_data.csv'

if not os.path.exists(fname):
    url='http://ichart.finance.yahoo.com/table.csv?s=%%5EIXIC&d=%d&e=%d&f=%d&g=d&a=%d&b=%d&c=%
    print url

    f = urllib.urlopen(url)

    k=open(fname,"wt")
    st=f.read()
    k.write(st)
    k.close()
    f.close()
```

Financial Data

```
Date,Open,High,Low,Close,Volume,Adj Close
2007-04-25,2533.54,2551.39,2523.84,2547.89,2644120000,2547.89
2007-04-24,2528.39,2529.48,2509.26,2524.54,2220610000,2524.54
2007-04-23,2525.77,2531.40,2518.47,2523.67,1928530000,2523.67
...
```

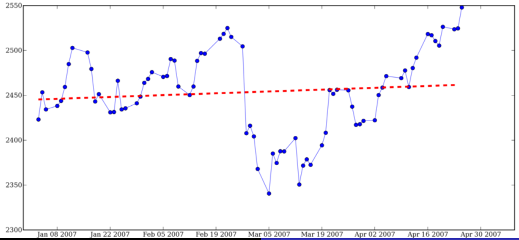
Parse the Data

```
# read the data
count=0
vals=[]; dates=[]
for line in open(fname):
    count=count+1

    if count==1: # skip the first line
        continue

    val=float(line.split(',')[-1])
    vals.append(val)  # last value

    date=line.split(',')[0].split('-')
    dint=[int(x) for x in date]  # convert to ints
    dateval=datetime.date(dint[0],dint[1],dint[2]).toordinal()
    dates.append(dateval)  # first value
```

Financial Data

Plot the Data

```
clf()
# plot the data
plot_date(dates,vals,'-o')

p=scipy.polyfit(dates,vals,1)
x=arange(min(dates),max(dates),1)
y=p[0]*x+p[1]
plot(x,y,'r--',linewidth=3)
```



Rosenbrock Function of N Variables

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$$

Minimum at $x_0 = x_1 = \ldots = 1$

Perform the Optimization

```
from scipy.optimize import fmin
def rosen(x): # The Rosenbrock function
        return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

x0 = [1.3, 0.7, 0.8, 1.9, 1.2]
xopt = fmin(rosen, x0) # Nelder-Mead simplex algorithm
```

```
Optimization terminated successfully.
         Current function value: 0.000066
         Iterations: 141
         Function evaluations: 243
[ 0.99910115  0.99820923  0.99646346  0.99297555  0.98600385]
```

Rosenbrock Function of N Variables

$$f(\boldsymbol{x}) = \sum_{i=1}^{N-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$$

Minimum at $x_0 = x_1 = \ldots = 1$

Perform the Optimization

```
from scipy.optimize import fmin
def rosen(x): # The Rosenbrock function
        return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

x0 = [1.3, 0.7, 0.8, 1.9, 1.2]
xopt = fmin(rosen, x0) # Nelder-Mead simplex algorithm
```

```
Optimization terminated successfully.
         Current function value: 0.000066
         Iterations: 141
         Function evaluations: 243
[ 0.99910115  0.99820923  0.99646346  0.99297555  0.98600385]
```

Rosenbrock Function of N Variables

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2$$

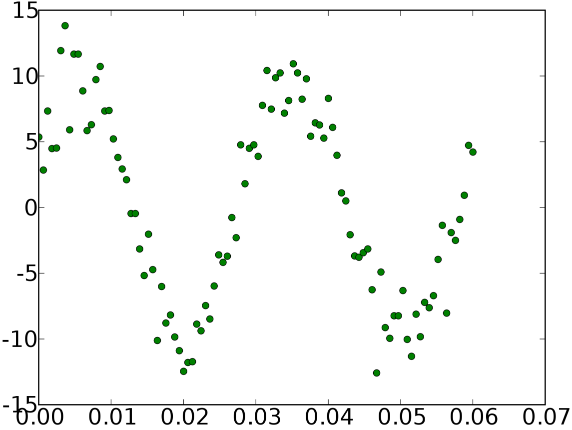
Minimum at $x_0 = x_1 = \ldots = 1$

Perform the Optimization

```
from scipy.optimize import fmin
def rosen(x): # The Rosenbrock function
        return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 + (1-x[:-1])**2.0)

x0 = [1.3, 0.7, 0.8, 1.9, 1.2]
xopt = fmin(rosen, x0) # Nelder-Mead simplex algorithm
```

```
Optimization terminated successfully.
         Current function value: 0.000066
         Iterations: 141
         Function evaluations: 243
[ 0.99910115  0.99820923  0.99646346  0.99297555  0.98600385]
```

Fitting a Sine Wave



Fitting a Sine Wave

Generate the Data

```
from pylab import *
from numpy import *
import scipy
from scipy import optimize

from bigfonts import bigfonts
bigfonts()

x=linspace(0,6e-2,100)
A,k,theta = 10, 1.0/3e-2, pi/6
y_true = A*sin(2*pi*k*x+theta)
y_meas = y_true + 2*randn(len(x))
```

Fitting a Sine Wave

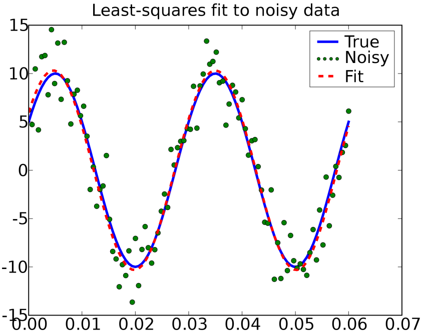
Fit the Data

```
def residuals(p, y, x):
    A,k,theta = p
    err = y-A*sin(2*pi*k*x+theta)
    return err

def peval(x, p):
    return p[0]*sin(2*pi*p[1]*x+p[2])

p0 = [20, 40, 10]
print "Initial values:",p0

plsq = optimize.leastsq(residuals, p0, args=(y_meas, x))
print "Final estimates:",plsq[0]

print "Actual values:", [A, k, theta]
```

Fitting a Sine Wave

Output from Program

```
Initial values: [20, 40, 10]
Final estimates: [-10.41111011  33.09546027  10.00631967]
Actual values: [10, 33.333333333333336, 0.52359877559829882]
```



Notable Differences in Favor of Matlab

- String as Argument when given No Parentheses

Matlab

```
help spam
```

Python

```
import spam
help(spam)
```

- Clean Syntax for Inputing Matrices

Matlab

```
a=[1 2 3 ; 4 5 6]
```

Python

```
from numpy import *
a=mat('[1 2 3 ; 4 5 6]')
a=matrix([ [1,2,3] , [4,5,6] ])
```

Notable Differences in Favor of Matlab

- String as Argument when given No Parentheses

Matlab

```
help spam
```

Python

```
import spam
help(spam)
```

- Clean Syntax for Inputing Matrices

Matlab

```
a=[1 2 3 ; 4 5 6]
```

Python

```
from numpy import *
a=mat('[1 2 3 ; 4 5 6]')
a=matrix([ [1,2,3] , [4,5,6] ])
```

Notable Differences in Favor of Matlab

- String as Argument when given No Parentheses

Matlab

```
help spam
```

Python

```
import spam
help(spam)
```

- Clean Syntax for Inputing Matrices

Matlab

```
a=[1 2 3 ; 4 5 6]
```

Python

```
from numpy import *
a=mat('[1 2 3 ; 4 5 6]')
a=matrix([ [1,2,3] , [4,5,6] ])
```

Notable Differences in Favor of Matlab

- Clean Syntax for Inputing Range

Matlab

```
a=1:10

b=linspace(1,10,10)
```

Python

```
from numpy import *
a=r_[1:11]  # 1 minus last number

b=linspace(1,10,10)  # better way
```

- Calling a user-created script

Matlab

```
% run my script with some commands
myscript;
```

Python

```
# in ipython
run myscript.py

# in regular python shell
execfile('myscript.py')
```

Notable Differences in Favor of Matlab

- Clean Syntax for Inputing Range

Matlab

```
a=1:10

b=linspace(1,10,10)
```

Python

```
from numpy import *
a=r_[1:11]  # 1 minus last number

b=linspace(1,10,10)  # better way
```

- Calling a user-created script

Matlab

```
% run my script with some commands
myscript;
```

Python

```
# in ipython
run myscript.py

# in regular python shell
execfile('myscript.py')
```

Notable Differences in Favor of Matlab

- Better integrated plot commands

Matlab

```
x=-10:10
y=x.^2
plot(x,y,'-o')
```

Python

```
from pylab import *
from numpy import *

x=linspace(-10,10,20)
y=x**2

plot(x,y,'-o')
show()
```

Notable Differences in Favor of Matlab

- the "." operators on matrices

Matlab

```
a=[1 2 3; 4 5 6; 7 8 9]
b=[10 20 30; 40 50 60; 70 80 90]

% matrix multiply
c=a*b

% element-by-element multiply
d=a.*b
```

Python

```
from numpy import *
# two choices: matrix or array class

a=mat('[1 2 3; 4 5 6; 7 8 9]')
b=mat('[10 20 30; 40 50 60; 70 80 90]')

# matrix multiply
c=a*b
# element-by-element multiply on matrix
d=multiply(a,b)

a=array(a)
b=array(b)

# matrix multiply on arrays
c=dot(a,b)
# element-by-element multiply on array
d=a*b
```

Notable Differences in Favor of Python

- Namespaces: Scales to Larger Projects

Matlab

```
a=sqrt(2)  % built-in

% uses first fmin in path
fmin('cos',3,4)
```

Python

```
import math
import mymath

a=math.sqrt(2)
b=mymath.sqrt(2)

from scipy.optimize import fmin
from myopt import fmin as fmin2

from math import cos
fmin(cos,3,4) # uses scipy fmin
fmin2(cos,3,4) # uses my fmin
```

- Free as in “beer” and “speech”
- Real object-oriented programming
- Can define functions in a script

Notable Differences in Favor of Python

- Namespaces: Scales to Larger Projects

Matlab

```
a=sqrt(2)  % built-in

% uses first fmin in path
fmin('cos',3,4)
```

Python

```
import math
import mymath

a=math.sqrt(2)
b=mymath.sqrt(2)

from scipy.optimize import fmin
from myopt import fmin as fmin2

from math import cos
fmin(cos,3,4) # uses scipy fmin
fmin2(cos,3,4) # uses my fmin
```

- Free as in “beer” and “speech”
- Real object-oriented programming
- Can define functions in a script

Notable Differences in Favor of Python

- Namespaces: Scales to Larger Projects

Matlab

```
a=sqrt(2)  % built-in

% uses first fmin in path
fmin('cos',3,4)
```

Python

```
import math
import mymath

a=math.sqrt(2)
b=mymath.sqrt(2)

from scipy.optimize import fmin
from myopt import fmin as fmin2

from math import cos
fmin(cos,3,4) # uses scipy fmin
fmin2(cos,3,4) # uses my fmin
```

- Free as in “beer” and “speech”
- Real object-oriented programming
- Can define functions in a script

Notable Differences in Favor of Python

- Namespaces: Scales to Larger Projects

Matlab

```
a=sqrt(2)  % built-in

% uses first fmin in path
fmin('cos',3,4)
```

Python

```
import math
import mymath

a=math.sqrt(2)
b=mymath.sqrt(2)

from scipy.optimize import fmin
from myopt import fmin as fmin2

from math import cos
fmin(cos,3,4) # uses scipy fmin
fmin2(cos,3,4) # uses my fmin
```

- Free as in “beer” and “speech”
- Real object-oriented programming
- Can define functions in a script

Notable Differences in Favor of Python

- Standard Library of Modules for Multiple Purposes

Standard Library

String Services

string:	Common string operations
re:	expression operations
struct:	Interpret strings as packed binary data
difflib:	Helpers for computing deltas
StringIO:	Read and write strings as files
cStringIO:	Faster version of StringIO
textwrap:	Text wrapping and filling
codecs:	Codec registry and base classes
unicodedata:	Unicode Database
stringprep:	Internet String Preparation
fpformat:	Floating point conversions

Data Types

datetime:	Basic date and time types
calendar:	General calendar-related functions
collections:	High-performance container datatypes
heapq:	Heap queue algorithm
bisect:	Array bisection algorithm
array:	Efficient arrays of numeric values

Notable Differences in Favor of Python

- Standard Library of Modules for Multiple Purposes

Standard Library

```
String Services
        string:                    Common string operations
            re:                   expression operations
        struct:      Interpret strings as packed binary data
       difflib:                Helpers for computing deltas
      StringIO:             Read and write strings as files
     cStringIO:                Faster version of StringIO
      textwrap:                 Text wrapping and filling
        codecs:           Codec registry and base classes
   unicodedata:                          Unicode Database
    stringprep:              Internet String Preparation
      fpformat:                Floating point conversions

Data Types
      datetime:                 Basic date and time types
      calendar:        General calendar-related functions
   collections:     High-performance container datatypes
         heapq:                      Heap queue algorithm
        bisect:                 Array bisection algorithm
         array:         Efficient arrays of numeric values
```

Notable Differences in Favor of Python

Standard Library (continued)

```
        sets:          Unordered collections of unique elements
       sched:                                   Event scheduler
       mutex:                          Mutual exclusion support
       Queue:                        A synchronized queue class
     weakref:                                   Weak references
    UserDict:             Class wrapper for dictionary objects
    UserList:                   Class wrapper for list objects
  UserString:                 Class wrapper for string objects
       types:                          Names for built-in types
         new:             Creation of runtime internal objects
        copy:                 Shallow and deep copy operations
      pprint:                              Data pretty printer
        repr:                   Alternate repr() implementation

Numeric and Mathematical Modules
        math:                            Mathematical functions
       cmath:        Mathematical functions for complex numbers
     decimal:                 Decimal floating point arithmetic
      random:                   Generate pseudo-random numbers
   itertools: Functions creating iterators for efficient looping
```

Notable Differences in Favor of Python

Standard Library (continued)

```
      functools: Higher order functions and operations on callable objects.
       operator:                   Standard operators as functions.

Internet Data Handling
          email:                 An email and MIME handling package
        mailcap:                            Mailcap file handling.
        mailbox:            Manipulate mailboxes in various formats
          mhlib:                            Access to MH mailboxes
      mimetools:                      Tools for parsing MIME messages
      mimetypes:                          Map filenames to MIME types
     MimeWriter:                           Generic MIME file writer
         mimify:                     MIME processing of mail messages
      multifile:        Support for files containing distinct parts
         rfc822:                          Parse RFC 2822 mail headers
         base64:    RFC 3548: Base16, Base32, Base64 Data Encodings
         binhex:                      Encode and decode binhex4 files
       binascii:                     Convert between binary and ASCII
         quopri:       Encode and decode MIME quoted-printable data
             uu:                         and decode uuencode files
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Structured Markup Processing Tools
     HTMLParser:                 Simple HTML and XHTML parser
        sgmllib:                           Simple SGML parser
        htmllib:                  A parser for HTML documents
 htmlentitydefs:          Definitions of HTML general entities
xml.parsers.expat:                Fast XML parsing using Expat
        xml.dom:                 The Document Object Model API
xml.dom.minidom:                Lightweight DOM implementation
xml.dom.pulldom:          Support for building partial DOM trees
        xml.sax:                      Support for SAX2 parsers
xml.sax.handler:                 Base classes for SAX handlers
xml.sax.saxutils:                                SAX Utilities
xml.sax.xmlreader:                   Interface for XML parsers
xml.etree.ElementTree:                 The ElementTree XML API

File Formats
            csv:                 CSV File Reading and Writing
   ConfigParser:                    Configuration file parser
    robotparser:                        Parser for robots.txt
          netrc:                         netrc file processing
         xdrlib:                   Encode and decode XDR data
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Cryptographic Services
        hashlib:                  Secure hashes and message digests
           hmac:           Keyed-Hashing for Message Authentication
            md5:                       MD5 message digest algorithm
            sha:                     SHA-1 message digest algorithm

File and Directory Access
        os.path:                      Common pathname manipulations
      fileinput:     Iterate over lines from multiple input streams
           stat:                        Interpreting stat() results
        statvfs:                     Constants used with os.statvfs()
        filecmp:                      File and Directory Comparisons
       tempfile:           Generate temporary files and directories
           glob:              UNIX style pathname pattern expansion
        fnmatch:                    UNIX filename pattern matching
      linecache:                        Random access to text lines
         shutil:                         High-level file operations
       dircache:                          Cached directory listings
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Data Compression and Archiving
        zlib:                   Compression compatible with gzip
        gzip:                             Support for gzip files
         bz2:                  Compression compatible with bzip2
     zipfile:                             Work with ZIP archives
     tarfile:                       Read and write tar archive files
Data Persistence
      pickle:                        Python object serialization
     cPickle:                                    A faster pickle
    copy_reg:                  Register pickle support functions
      shelve:                          Python object persistence
     marshal:               Internal Python object serialization
      anydbm:              Generic access to DBM-style databases
     whichdb:          Guess which DBM module created a database
         dbm:              Simple "cdatabase"d interface
        gdbm:                 GNU's reinterpretation of dbm
      dbhash:    DBM-style interface to the BSD database library
       bsddb:                     Interface to Berkeley DB library
     dumbdbm:                         Portable DBM implementation
     sqlite3:          DB-API 2.0 interface for SQLite databases
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Generic Operating System Services
            os:                         operating system interfaces
          time:                         Time access and conversions
      optparse:             More powerful command line option parser
        getopt:                         Parser for command line options
       logging:                         Logging facility for Python
       getpass:                         Portable password input
        curses:      Terminal handling for character-cell displays
  curses.ascii:                         Utilities for ASCII characters
  curses.panel:                    A panel stack extension for curses.
      platform: Access to underlying platform's identifying data.
         errno:                         Standard errno system symbols
        ctypes:                 A foreign function library for Python.
Optional Operating System Services
        select:                         Waiting for I/O completion
        thread:                         Multiple threads of control
     threading:                         Higher-level threading interface
  dummy_thread:             Drop-in replacement for the thread module
dummy_threading:         Drop-in replacement for the threading module
          mmap:                         Memory-mapped file support
      readline:                         GNU readline interface
   rlcompleter:                  Completion function for GNU readline
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Unix Specific Services
        posix:                 The most common POSIX system calls
          pwd:                              The password database
         spwd:                       The shadow password database
          grp:                                 The group database
        crypt:                   Function to check UNIX passwords
           dl:                      C functions in shared objects
      termios:                            POSIX style tty control
          tty:                         Terminal control functions
          pty:                          Pseudo-terminal utilities
        fcntl:               The fcntl() and ioctl() system calls
        pipes:                       Interface to shell pipelines
    posixfile:             File-like objects with locking support
     resource:                         Resource usage information
          nis:            Interface to Sun’s NIS (Yellow Pages)
       syslog:                       UNIX syslog library routines
     commands:                      Utilities for running commands
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Interprocess Communication and Networking
     subprocess:                              Subprocess management
         socket:                      Low-level networking interface
         signal:                 Set handlers for asynchronous events
         popen2:             Subprocesses with accessible I/O streams
       asyncore:                           Asynchronous socket handler
       asynchat:        Asynchronous socket command/response handler

Internet Protocols and Support
     webbrowser:                     Convenient Web-browser controller
            cgi:                      Common Gateway Interface support
          cgitb:                     Traceback manager for CGI scripts
        wsgiref:         WSGI Utilities and Reference Implementation
         urllib:                       Open arbitrary resources by URL
        urllib2:                   extensible library for opening URLs
        httplib:                                  HTTP protocol client
         ftplib:                                   FTP protocol client
      gopherlib:                                Gopher protocol client
         poplib:                                  POP3 protocol client
        imaplib:                                 IMAP4 protocol client
```

Notable Differences in Favor of Python

Standard Library (continued)

`nntplib:`	`NNTP protocol client`
`smtplib:`	`SMTP protocol client`
`smtpd:`	`SMTP Server`
`telnetlib:`	`Telnet client`
`uuid:`	`UUID objects according to RFC 4122`
`urlparse:`	`Parse URLs into components`
`SocketServer:`	`A framework for network servers`
`BaseHTTPServer:`	`Basic HTTP server`
`SimpleHTTPServer:`	`Simple HTTP request handler`
`CGIHTTPServer:`	`CGI-capable HTTP request handler`
`cookielib:`	`Cookie handling for HTTP clients`
`Cookie:`	`HTTP state management`
`xmlrpclib:`	`XML-RPC client access`
`SimpleXMLRPCServer:`	`Basic XML-RPC server`
`DocXMLRPCServer:`	`Self-documenting XML-RPC server`

Notable Differences in Favor of Python

Standard Library (continued)

```
Internationalization
       gettext:        Multilingual internationalization services
        locale:                    Internationalization services

Multimedia Services
       audioop:                        Manipulate raw audio data
       imageop:                        Manipulate raw image data
          aifc:                Read and write AIFF and AIFC files
         sunau:                       Read and write Sun AU files
          wave:                          Read and write WAV files
         chunk:                             Read IFF chunked data
      colorsys:                 Conversions between color systems
        rgbimg:                  Read and write "cSGI RGB"d files
        imghdr:                      Determine the type of an image
        sndhdr:                     Determine type of sound file
   ossaudiodev:           Access to OSS-compatible audio devices

Graphical User Interfaces with Tk
       Tkinter:                       Python interface to Tcl/Tk
           Tix:                         Extension widgets for Tk
  ScrolledText:                             Scrolled Text Widget
        turtle:                           Turtle graphics for Tk
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Program Frameworks
           cmd:     Support for line-oriented command interpreters
         shlex:                            Simple lexical analysis

Development Tools
         pydoc:     Documentation generator and online help system
       doctest:                  Test interactive Python examples
      unittest:                            Unit testing framework

The Python Profilers/Debuggers
           pdb:                                    Python Debugger
       hotshot:                 High performance logging profiler
        timeit:      Measure execution time of small code snippets
         trace:         Trace or track Python statement execution

Python Runtime Services
           sys:           System-specific parameters and functions
      warnings:                                    Warning control
        atexit:                                      Exit handlers
     traceback:                Print or retrieve a stack traceback
    __future__:                       Future statement definitions
            gc:                                Collector interface
       inspect:                               Inspect live objects
        fpectl:                   Floating point exception control
```

Notable Differences in Favor of Python

Standard Library (continued)

```
Python Language Services
         parser:                         Access Python parse trees
         symbol:             Constants used with Python parse trees
          token:             Constants used with Python parse trees
        keyword:                        Testing for Python keywords
       tokenize:                        Tokenizer for Python source
       tabnanny:                 Detection of ambiguous indentation
         pyclbr:                       Python class browser support
     py_compile:                        Compile Python source files
     compileall:                      Byte-compile Python libraries
            dis:                    Disassembler for Python byte code
    pickletools:                       Tools for pickle developers.
      distutils:             Building and installing Python modules

MS Windows Specific Services
         msilib:           Read and write Microsoft Installer files
         msvcrt:           Useful routines from the MS VC++ runtime
        _winreg:                          Windows registry access
       winsound:                Sound-playing interface for Windows
```

Notable Differences in Favor of Python

- Other Libraries for Multiple Purposes

Other Libraries - cheeseshop.python.org

BeautifulSoup	HTML/XML parser for quick-turnaround applications
CherryPy 3.0.1	Object-Oriented HTTP framework
Wax 0.3.19	A user-friendly layer on top of wxPython.
xlrd 0.6.1a4	Library to extract data from Microsoft Excel(tm) spreadsheet files
buzhug 0.7	A fast pure-Python database engine
psyco 1.2	Psyco, the Python specializing compiler
TurboGears 1.0.2	front-to-back rapid web development
SQLAlchemy 0.3.7	Database Abstraction Library
Shed Skin 0.0.21	An Optimizing Python-to-C++ Compiler
Golly	An open source, cross-platform Game of Life simulator
pyXLWriter 0.4a2	A library for generating Excel Spreadsheets
VPython	A Python module that offers real-time 3D output,

(Currently 2355 packages as of 5/08/2007)

Notable Differences in Favor of Python

- Other Libraries for Multiple Purposes

Other Libraries - `cheeseshop.python.org`

```
BeautifulSoup      HTML/XML parser for quick-turnaround applications
CherryPy 3.0.1     Object-Oriented HTTP framework
Wax 0.3.19         A user-friendly layer on top of wxPython.
xlrd 0.6.1a4       Library to extract data from Microsoft Excel(tm) spreadsheet files
buzhug 0.7         A fast pure-Python database engine
psyco 1.2          Psyco, the Python specializing compiler
TurboGears 1.0.2   front-to-back rapid web development
SQLAlchemy 0.3.7   Database Abstraction Library
Shed Skin 0.0.21   An Optimizing Python-to-C++ Compiler
Golly              An open source, cross-platform Game of Life simulator
pyXLWriter 0.4a2   A library for generating Excel Spreadsheets
VPython            A Python module that  offers real-time 3D output,
```

(Currently 2355 packages as of 5/08/2007)

Factorial

- Extensions with Pyrex

Python

```
def factorial(n):

    x=1
    for i in range(1,n+1):
        x=x*i

    return x
```

Pyrex

```
def factorial(int n):
    cdef int i
    cdef double x

    x=1
    for i from 1 <= i <= n:
        x=x*i

    return x
```

Factorial

- Extensions with Pyrex

Python

```
def factorial(n):

    x=1
    for i in range(1,n+1):
        x=x*i

    return x
```

Pyrex

```
def factorial(int n):
    cdef int i
    cdef double x

    x=1
    for i from 1 <= i <= n:
        x=x*i

    return x
```

Compiling

Setup

```
from pyrex_compile import *
compile('factorial_pyrex.pyx')
```

Test

```
In [1]:import factorial_python as python
In [2]:import factorial_pyrex as pyrex

In [3]:%timeit python.factorial(200)
1000 loops, best of 3: 319 microsec per loop

In [4]:%timeit pyrex.factorial(200)
100000 loops, best of 3: 4.17 microsec per loop
```

Compiling

Setup

```
from pyrex_compile import *
compile('factorial_pyrex.pyx')
```

Test

```
In [1]:import factorial_python as python
In [2]:import factorial_pyrex as pyrex

In [3]:%timeit python.factorial(200)
1000 loops, best of 3: 319 microsec per loop

In [4]:%timeit pyrex.factorial(200)
100000 loops, best of 3: 4.17 microsec per loop
```

C-API

Headers

```
/* Generated by Pyrex 0.9.5.1a on Sat Apr 28 11:15:48 2007 */
#include "Python.h"
#include "structmember.h"
#ifndef PY_LONG_LONG
  #define PY_LONG_LONG LONG_LONG
#endif
#ifdef __cplusplus
#define __PYX_EXTERN_C extern "C"
#else
#define __PYX_EXTERN_C extern
#endif
__PYX_EXTERN_C double pow(double, double);

typedef struct {PyObject **p; char *s;} __Pyx_InternTabEntry; /*proto*/
typedef struct {PyObject **p; char *s; long n;} __Pyx_StringTabEntry; /*proto*/

static PyObject *__pyx_m;
static PyObject *__pyx_b;
static int __pyx_lineno;
static char *__pyx_filename;
static char **__pyx_f;
```

C-API

Translation

```
int __pyx_v_n;
int __pyx_v_i;
double __pyx_v_x;
PyObject *__pyx_r;
PyObject *__pyx_1 = 0;
static char *__pyx_argnames[] = {"n",0};
if (!PyArg_ParseTupleAndKeywords(__pyx_args, __pyx_kwds, "i",
                           __pyx_argnames, &__pyx_v_n)) return 0;
/* "/home/bblais/tex/bryant/facdev/spring2007/src/factorial_pyrex.pyx":5 */
__pyx_v_x = 1;

/* "/home/bblais/tex/bryant/facdev/spring2007/src/factorial_pyrex.pyx":6 */
for (__pyx_v_i = 1; __pyx_v_i <= __pyx_v_n; ++__pyx_v_i) {

  /* "/home/bblais/tex/bryant/facdev/spring2007/src/factorial_pyrex.pyx":7 */
  __pyx_v_x = (__pyx_v_x * __pyx_v_i);
}
/* "/home/bblais/tex/bryant/facdev/spring2007/src/factorial_pyrex.pyx":9 */
__pyx_1 = PyFloat_FromDouble(__pyx_v_x); if (!__pyx_1) {__pyx_filename =
                        __pyx_f[0]; __pyx_lineno = 9; goto __pyx_L1;}
__pyx_r = __pyx_1;
__pyx_1 = 0;
```

Watch Out!

Test

```
In [1]:import factorial_python as python

In [2]:import factorial_pyrex as pyrex

In [3]:%timeit python.factorial(200)
1000 loops, best of 3: 319 microsec per loop

In [4]:%timeit pyrex.factorial(200)
100000 loops, best of 3: 4.17 microsec per loop
```

Watch Out!

Test

```
In [1]:import factorial_python as python

In [2]:import factorial_pyrex as pyrex

In [3]:%timeit python.factorial(200)
1000 loops, best of 3: 319 microsec per loop

In [4]:%timeit pyrex.factorial(200)
100000 loops, best of 3: 4.17 microsec per loop

In [5]:pyrex.factorial(200)
Out[5]:inf

In [6]:python.factorial(200)
Out[6]:788657867364790503552363213932185062295135977687173263294742533244
35944996340334292030428401198462390417721213891963883025764279024263710506
19266249528299311134628572707633172373969889439224456214516642402540332918
64131227428294853277524242407573903240321257405579568660226031904170324062
35170085879617892222278962370389737472000000000000000000000000000000000000
000000000000L
```

Function using Array

Pyrex Code

```
cimport c_python
cimport c_numpy
c_numpy.import_array() # Numpy must be initialized

def spam(c_numpy.ndarray A):
    cdef double *p
    cdef double result
    cdef int i,N,nd

    nd=A.nd # number of dimensions

    N=1
    for i from 0<=i<nd: # calculate number of elements
       N=N*A.dimensions[i]

    p=<double *>A.data

    result=0.0
    for i from 0<=i<N:
        result=result+p[i]**2

    return result
```

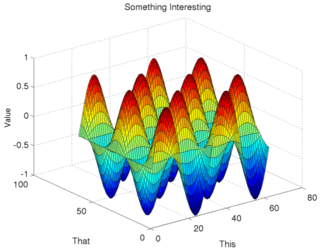
Testing

```
In [1]:from numpy.random import rand
In [2]:from pyrex_numpy import spam
In [3]:a=rand(2,3,4)
In [4]:a
Out[4]:
array([[[ 0.41275586,  0.40248059,  0.52365634,  0.13457172],
        [ 0.10361721,  0.07592018,  0.50031702,  0.65126816],
        [ 0.09734859,  0.82231387,  0.74795067,  0.48530395]],

       [[ 0.17096585,  0.42510408,  0.4848095 ,  0.12744915],
        [ 0.49256875,  0.1358942 ,  0.12986233,  0.86068033],
        [ 0.10222339,  0.46645587,  0.82551456,  0.54402251]]])

In [5]:a.shape
Out[5]:(2, 3, 4)
In [6]:spam(a)
Out[6]:5.481696128900011
In [7]:(a**2).sum()
Out[7]:5.4816961289

In [8]:%timeit spam(a)
1000000 loops, best of 3: 465 ns per loop

In [9]:%timeit (a**2).sum()
10000 loops, best of 3: 31.2 microsec per loop
```

mlabwrap.py

Communicating with Matlab

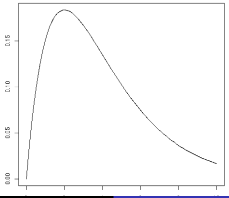
```
from mlabwrap import mlab
from numpy import *

xx = arange(-2*pi, 2*pi, 0.2)
X,Y=mlab.meshgrid(xx,xx,nout=2)
mlab.surf(sin(X)*cos(Y))

mlab.xlabel('This')
mlab.ylabel('That')
mlab.title('Something Interesting')
mlab.zlabel('Value')

a=mlab.svd(array([[1,2], [1,3]]))

print a

# Run:
In [44]:run test_mlab.py
[[ 3.86432845]
 [ 0.25877718]]
```



rpy.py

Communicating with R

```
# Simple script for drawing the chi-squared density
#
from rpy import *

degrees = 4
grid = r.seq(0, 10, length=100)
values = [r.dchisq(x, degrees) for x in grid]
r.par(ann=0)
r.plot(grid, values, type='lines')
```



xlrd.py

Reading Excel Spreadsheets

```
import xlrd
book = xlrd.open_workbook("master.xls")
print "The number of worksheets is", book.nsheets
print "Worksheet name(s):", book.sheet_names()
sh = book.sheet_by_index(0)
print sh.name, sh.nrows, sh.ncols
print "Cell D30 is", sh.cell_value(rowx=29, colx=3)
for rx in range(sh.nrows):
    print sh.row(rx)
```

Conclusions

Questions?