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.
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

```python
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

B. Blais  Numerical Computing in Python
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**

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyrex</td>
<td>writing fast compiled extensions (like cmex, but way better)</td>
</tr>
<tr>
<td>wxpython</td>
<td>GUI library</td>
</tr>
<tr>
<td>pywin32</td>
<td>Windows COM Interface</td>
</tr>
<tr>
<td>BeautifulSoup</td>
<td>HTML Parser</td>
</tr>
<tr>
<td>xlrd, pyXLWriter</td>
<td>Reading/Writing Excel Spreadsheets</td>
</tr>
</tbody>
</table>

B. Blais Numerical Computing in Python
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

<table>
<thead>
<tr>
<th>Useful</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pyrex</strong></td>
<td>writing fast compiled extensions (like cmex, but way better)</td>
</tr>
<tr>
<td><strong>wxpython</strong></td>
<td>GUI library</td>
</tr>
<tr>
<td><strong>pywin32</strong></td>
<td>Windows COM Interface</td>
</tr>
<tr>
<td><strong>BeautifulSoup</strong></td>
<td>HTML Parser</td>
</tr>
<tr>
<td><strong>xLrd, pyXLWriter</strong></td>
<td>Reading/Writing Excel Spread-sheets</td>
</tr>
</tbody>
</table>
Introduction
Comparison With Matlab
Advantages
Extensions with Pyrex
Communication

What is Python?
Projects
What You Need

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
Introduction

Comparison With Matlab

Advantages

Extensions with Pyrex

Communication

Initial Comparison

Getting Help

Golden Ratio

Fibonacci

Finance

Optimization and Least Squares

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
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
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

B. Blais Numerical Computing in Python
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
In Matlab, help is contained in the file

```matlab
>> 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...
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...
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 [*]

...
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.
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.deprecate numpy.diagonal   numpy.disp      numpy.double
  numpy.diag      numpy.diff       numpy.divide   numpy.dsplit
  numpy.deprecate
  numpy-diagonal
  numpy-disp
  numpy-double
  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.
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**

```matlab
function goldfract(n)
    p = '1';
    for k = 1:n
        p = ['1+1/(' p ')'];
    end
    p = sprintf('%d/%d',p,q);
    format long
    p = eval(p)
    err = (1+sqrt(5))/2 - p
```

**Python**

```python
def goldfract(N):
    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)
    p=eval(p)
    err = (1+sqrt(5))/2 - p
    print err
```

---

**Initial Comparison**

**Getting Help**

**Golden Ratio**

**Fibonacci**

**Finance**

**Optimization and Least Squares**
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

```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
```

Python: List

```python
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

```python
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=%d
    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

```python
# 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(',')[11])
    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
```

B. Blais Numerical Computing in Python
Financial Data

Plot the Data

```python
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

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

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

Perform the Optimization

```python
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]
```
Introduction

Comparison With Matlab

Advantages

Extensions with Pyrex

Communication

Initial Comparison

Getting Help

Golden Ratio

Fibonacci

Finance

Optimization and Least Squares

Rosenbrock Function of $N$ Variables

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

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

Perform the Optimization

```python
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

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

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

Perform the Optimization

```python
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

```python
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**
  
  ```matlab
  help spam
  ```

  **Python**
  
  ```python
  import spam
  help(spam)
  ```

- **Clean Syntax for Inputing Matrices**

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

  **Python**
  
  ```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**
    ```matlab
    help spam
    ```
  - **Python**
    ```python
    import spam
    help(spam)
    ```

- **Clean Syntax for Inputing Matrices**
  - **Matlab**
    ```matlab
    a=[1 2 3 ; 4 5 6]
    ```
  - **Python**
    ```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

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

Python

```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**

```matlab
da=[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**

```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
```matlab
a=sqrt(2) % built-in
% uses first fmin in path
fmin('cos',3,4)
```

### Python
```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**

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

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

**Python**

```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**

```matlab
a=sqrt(2) % built-in
% uses first fmin in path
fmin('cos',3,4)
```

**Python**

```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
Introduction
Comparison With Matlab
Advantages
Extensions with Pyrex
Communication

Notable Differences in Favor of Python

- Namespaces: Scales to Larger Projects

**Matlab**

```matlab
a=sqrt(2) % built-in
% uses first fmin in path
fmin('cos',3,4)
```

**Python**

```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

In Favor of Matlab
In Favor of Python

B. Blais
Numerical Computing in Python
### Notable Differences in Favor of Python

- **Standard Library of Modules for Multiple Purposes**

<table>
<thead>
<tr>
<th>Standard Library</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String Services</strong></td>
</tr>
<tr>
<td>string:</td>
</tr>
<tr>
<td>re:</td>
</tr>
<tr>
<td>struct:</td>
</tr>
<tr>
<td>difflib:</td>
</tr>
<tr>
<td>StringIO:</td>
</tr>
<tr>
<td>cStringIO:</td>
</tr>
<tr>
<td>textwrap:</td>
</tr>
<tr>
<td>codecs:</td>
</tr>
<tr>
<td>unicodedata:</td>
</tr>
<tr>
<td>stringprep:</td>
</tr>
<tr>
<td>fpformat:</td>
</tr>
<tr>
<td><strong>Data Types</strong></td>
</tr>
<tr>
<td>datetime:</td>
</tr>
<tr>
<td>calendar:</td>
</tr>
<tr>
<td>collections:</td>
</tr>
<tr>
<td>heapq:</td>
</tr>
<tr>
<td>bisect:</td>
</tr>
<tr>
<td>array:</td>
</tr>
</tbody>
</table>

- **Common string operations**
- **Expression operations**
- **Interpret strings as packed binary data**
- **Helpers for computing deltas**
- **Read and write strings as files**
- **Faster version of StringIO**
- **Text wrapping and filling**
- **Codec registry and base classes**
- **Unicode Database**
- **Internet String Preparation**
- **Floating point conversions**
- **Basic date and time types**
- **General calendar-related functions**
- **High-performance container datatypes**
- **Heap queue algorithm**
- **Array bisection algorithm**
- **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)**

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

**Numeric and Mathematical Modules**

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
<td>Mathematical functions</td>
</tr>
<tr>
<td>cmath</td>
<td>Mathematical functions for complex numbers</td>
</tr>
<tr>
<td>decimal</td>
<td>Decimal floating point arithmetic</td>
</tr>
<tr>
<td>random</td>
<td>Generate pseudo-random numbers</td>
</tr>
<tr>
<td>itertools</td>
<td>Functions creating iterators for efficient looping</td>
</tr>
</tbody>
</table>
### Notable Differences in Favor of Python

#### Standard Library (continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>functools</code></td>
<td>Higher order functions and operations on callable objects.</td>
</tr>
<tr>
<td><code>operator</code></td>
<td>Standard operators as functions.</td>
</tr>
<tr>
<td><code>email</code></td>
<td>An email and MIME handling package</td>
</tr>
<tr>
<td><code>mailcap</code></td>
<td>Mailcap file handling.</td>
</tr>
<tr>
<td><code>mailbox</code></td>
<td>Manipulate mailboxes in various formats</td>
</tr>
<tr>
<td><code>mhlib</code></td>
<td>Access to MH mailboxes</td>
</tr>
<tr>
<td><code>mimetypes</code></td>
<td>Tools for parsing MIME messages</td>
</tr>
<tr>
<td><code>mimetypes</code></td>
<td>Map filenames to MIME types</td>
</tr>
<tr>
<td><code>MimeWriter</code></td>
<td>Generic MIME file writer</td>
</tr>
<tr>
<td><code>mimify</code></td>
<td>MIME processing of mail messages</td>
</tr>
<tr>
<td><code>multifile</code></td>
<td>Support for files containing distinct parts</td>
</tr>
<tr>
<td><code>rfc822</code></td>
<td>Parse RFC 2822 mail headers</td>
</tr>
<tr>
<td><code>base64</code></td>
<td>RFC 3548: Base16, Base32, Base64 Data Encodings</td>
</tr>
<tr>
<td><code>binhex</code></td>
<td>Encode and decode binhex4 files</td>
</tr>
<tr>
<td><code>binascii</code></td>
<td>Convert between binary and ASCII</td>
</tr>
<tr>
<td><code>quopri</code></td>
<td>Encode and decode MIME quoted-printable data</td>
</tr>
<tr>
<td><code>uu</code></td>
<td>Encode and decode uuencode files</td>
</tr>
</tbody>
</table>
## Notable Differences in Favor of Python

### Standard Library (continued)

#### Structured Markup Processing Tools

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTMLParser</td>
<td>Simple HTML and XHTML parser</td>
</tr>
<tr>
<td>sgmllib</td>
<td>Simple SGML parser</td>
</tr>
<tr>
<td>htmllib</td>
<td>A parser for HTML documents</td>
</tr>
<tr>
<td>htmlentitydefs</td>
<td>Definitions of HTML general entities</td>
</tr>
<tr>
<td>xml.parsers.expat</td>
<td>Fast XML parsing using Expat</td>
</tr>
<tr>
<td>xml.dom</td>
<td>The Document Object Model API</td>
</tr>
<tr>
<td>xml.dom.minidom</td>
<td>Lightweight DOM implementation</td>
</tr>
<tr>
<td>xml.dom.pulldom</td>
<td>Support for building partial DOM trees</td>
</tr>
<tr>
<td>xml.sax</td>
<td>Support for SAX2 parsers</td>
</tr>
<tr>
<td>xml.sax.handler</td>
<td>Base classes for SAX handlers</td>
</tr>
<tr>
<td>xml.sax.saxutils</td>
<td>SAX Utilities</td>
</tr>
<tr>
<td>xml.sax.xmlreader</td>
<td>Interface for XML parsers</td>
</tr>
<tr>
<td>xml.etree.ElementTree</td>
<td>The ElementTree XML API</td>
</tr>
</tbody>
</table>

#### File Formats

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csv</td>
<td>CSV File Reading and Writing</td>
</tr>
<tr>
<td>ConfigParser</td>
<td>Configuration file parser</td>
</tr>
<tr>
<td>robotparser</td>
<td>Parser for robots.txt</td>
</tr>
<tr>
<td>netrc</td>
<td>netrc file processing</td>
</tr>
<tr>
<td>xdrlib</td>
<td>Encode and decode XDR data</td>
</tr>
</tbody>
</table>
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
- db: 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
- dbm: 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
- `mmap`: GNU readline interface
- `rlcompleter`: Completion function for GNU readline
## Standard Library (continued)

### Unix Specific Services

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

<table>
<thead>
<tr>
<th>Standard Library (continued)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nntplib</code></td>
<td>NNTP protocol client</td>
</tr>
<tr>
<td><code>smtplib</code></td>
<td>SMTP protocol client</td>
</tr>
<tr>
<td><code>smtpd</code></td>
<td>SMTP Server</td>
</tr>
<tr>
<td><code>telnetlib</code></td>
<td>Telnet client</td>
</tr>
<tr>
<td><code>uuid</code></td>
<td>UUID objects according to RFC 4122</td>
</tr>
<tr>
<td><code>urllib.parse</code></td>
<td>Parse URLs into components</td>
</tr>
<tr>
<td><code>SocketServer</code></td>
<td>A framework for network servers</td>
</tr>
<tr>
<td><code>BaseHTTPServer</code></td>
<td>Basic HTTP server</td>
</tr>
<tr>
<td><code>SimpleHTTPServer</code></td>
<td>Simple HTTP request handler</td>
</tr>
<tr>
<td><code>CGIHTTPServer</code></td>
<td>CGI-capable HTTP request handler</td>
</tr>
<tr>
<td><code>cookielib</code></td>
<td>Cookie handling for HTTP clients</td>
</tr>
<tr>
<td><code>Cookie</code></td>
<td>HTTP state management</td>
</tr>
<tr>
<td><code>xmlrpclib</code></td>
<td>XML-RPC client access</td>
</tr>
<tr>
<td><code>SimpleXMLRPCServer</code></td>
<td>Basic XML-RPC server</td>
</tr>
<tr>
<td><code>DocXMLRPCServer</code></td>
<td>Self-documenting XML-RPC server</td>
</tr>
</tbody>
</table>
### Standard Library (continued)

#### Internationalization
- **gettext**: Multilingual internationalization services
- **locale**: Internationalization services

#### Multimedia Services
- **audioop**: Manipulate raw audio data
- **imageop**: Manipulate raw image data
- **aiff**: 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**

- **Internationalization**
  - gettext: Multilingual internationalization services
  - locale: Internationalization services

- **Multimedia Services**
  - audioop: Manipulate raw audio data
  - imageop: Manipulate raw image data
  - aiff: 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
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**

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

(Currently 2355 packages as of 5/08/2007)
Notable Differences in Favor of Python

- Other Libraries for Multiple Purposes

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

(Currently 2355 packages as of 5/08/2007)
Extensions with Pyrex

Python

```python
def factorial(n):
    x=1
    for i in range(1,n+1):
        x=x*i
    return x
```

Pyrex

```python
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**

```python
def factorial(n):
    x=1
    for i in range(1,n+1):
        x=x*i
    return x
```

**Pyrex**

```python
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;

B. Blais
Numerical Computing in Python
Factorial

Using Numpy Arrays

C-API

Translation

```c
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 =
/* "/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 =
/* "/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 =
```

B. Blais

Numerical Computing in Python
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]: 78865786736479050355236321393218506229513597768717326329474253324435944996340334292030428401198462390417721213891963883025764279024263710506192662495282993111346285727076331723739698894392244562145166424025403329186413122742829485327752424240757390324032125740557956866022603190417032406235170085879617892222278962370389737472000000000000000000000000000000000000000000000000000000000000L
Function using Array

Pyrex Code

```python
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
```

B. Blais  Numerical Computing in Python
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
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]]
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')
Reading Excel Spreadsheets

```python
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?