NumPy, Matplotlib and SciPy
HPC Python

Cyrus Proctor
cproctor@tacc.utexas.edu

January 23rd, 2015
NumPy

Python Objects
- High-level number objects: integers, floating point
- Containers: lists, dictionaries

NumPy
- Extension package for multi-dimensional arrays
- Closer to hardware $\rightarrow$ efficiency
- Designed for scientific computation
**NumPy and Python List**

### Python List

In [1]: import numpy as np
In [2]: list = range(100000)

In [3]: %timeit [i**2 for i in list]
100 loops, best of 3: 6.43 ms per loop

In [4]: array = np.arange(100000)

In [5]: %timeit array**2
1000 loops, best of 3: 97.7 us per loop
Why so Slow?

• Dynamic typing requires lots of metadata around variables
• Potentially inefficient memory access
• Interpreted instead of compiled

What can you do?

• Make an object that has a single type and continuous storage
• Implement common functionality into that object to iterate in C
NumPy Features

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions

```python
>>> a = np.array([1.0, 2.0, 3.0])
>>> b = np.array([2.0, 2.0, 2.0])
>>> a * b
array([ 2. , 4. , 6. ])
```

- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities
Array Object

What makes an array so much faster?

- **Data layout**
  - homogenous: every item takes up the same size block of memory
  - single data-type objects
  - powerful array scalar types

- **universal function (ufuncs)**
  - function that operates on ndarrays in an element-by-element fashion
  - vectorized wrapper for a function
  - built-in functions are implemented in compiled C code
- Numpy: contiguous data buffer of values
- Python: contiguous buffer of pointers
ufuncs

- function that operates on ndarrays in an element-by-element fashion
- vectorized wrapper for a function
- built-in functions are implemented in compiled C code

Python function - ufunc

| In [1]: import numpy as np |
| In [2]: import math |
| In [3]: arr = np.arange(100000) |
| In [4]: %timeit [math.sin(i) for i in arr] |
| 10 loops, best of 3: 18.3 ms per loop |
| In [5]: %timeit np.sin(arr) |
| 100 loops, best of 3: 1.77 ms per loop |
| In [6]: %timeit [math.sin(i)**2 for i in arr] |
| 10 loops, best of 3: 27.3 ms per loop |
| In [7]: %timeit np.sin(arr)**2 |
| 100 loops, best of 3: 1.83 ms per loop |

Mathematical functions
How to Create an Array

elements/3_numpy/array.py

```python
import numpy as np

# Create from list
a = np.array([2, 3, 12])

# Create from range
a = np.arange(10)  # 0, 1, 2, 3, 4,..., 9
b = np.arange(0, 10, 2)  # start, end (exclusive), step. 0, 2, 4, 6, 8

# By number of points (start, end, num. points)
a = np.linspace(0, 1, 5)  # 0, 0.25, 0.50, 0.75, 1.0
a = np.linspace(0, 1, 5, endpoint=False)  # 0, 0.2, 0.4, 0.6, 0.8

# Useful arrays
a = np.ones((4, 4))
a = np.zeros((3, 3))
a = np.diag(np.ones(3))
a = np.eye(3)

# With random numbers
np.random.seed(1111)  # sets the random seed
a = np.random.rand(4)  # uniform in [0,1]
b = np.random.randn(4)  # Gaussian

# Uninitialized
a = np.empty((3, 3))

# Resize
a = np.zeros(10)
a = np.resize(a, 20)
```
## Data Types

<table>
<thead>
<tr>
<th>bool</th>
<th>string</th>
<th>int</th>
<th>float</th>
<th>complex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>int8</td>
<td>float16</td>
<td>complex64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int16</td>
<td>float32</td>
<td>complex128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int32</td>
<td>float64</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>int64</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint32</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>uint64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Types

Basic

In [1]: import numpy as np
In [2]: a = np.array([1, 2, 3])
In [3]: a.dtype
Out[3]: dtype('int64')
In [4]: b = np.array([1., 2., 3.])
In [5]: b.dtype
Out[5]: dtype('float64')

Other

In [6]: c = np.array([1, 2, 3], dtype=float)
In [7]: c.dtype
Out[7]: dtype('float64')
In [8]: d = np.array([True, False, True])
In [9]: d.dtype
Out[9]: dtype('bool')
In [10]: e = np.array([1+2j, 3+4j, 5+6*1j])
In [11]: e.dtype
Out[11]: dtype('complex128')
In [12]: f = np.array(['Bonjour', 'Hello', 'Hola'])
In [13]: f.dtype
Out[13]: dtype('S7')  #Strings of max. 7 characters
Linear Algebra dot Function

In [1]: import numpy as np

In [2]: np.dot(np.arange(3), np.arange(3))
Out[2]: 5

In [3]: np.dot(np.arange(9).reshape(3,3), np.arange(3))
Out[3]: array([[ 5, 14, 23]])

In [4]: np.arange(9).reshape(3,3)
Out[4]:
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
NumPy Base Example

Basic MatVec: examples/3_numpy/interactive/matrix_vector.py

```python
#!/usr/bin/env python
import numpy as np
np.set_printoptions(precision=3)

def mat_vec():
    # Read in Column Vector; Store in x
    vector_filename = "my_vector.txt"
x = np.loadtxt(vector_filename, ndmin=2)
    print x
    # Read in Square Matrix; Store in A
    # Use np.loadtxt to read
    # in contents of "my_matrix.txt"
    matrix_filename = ??
    A = ??
    print A
    # Compute "b = A * x" using np.dot(A, x)
    b = ??
    print b
    result_filename = "my_result.txt"
    # Write b to "my_result.txt" with
    # np.savetxt(filename, object_to_save)
    ??
    return A, x, b

if __name__ == "__main__":
    # Run Function mat_vec
    A, x, b = ??
```
SciPy

What’s SciPy?

- Mathematical algorithms and convenience functions built on NumPy
- Organized into subpackages covering different scientific computing domains
- A data-processing and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab
SciPy

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse Eigenvalue Problems with ARPACK
- Statistics (scipy.stats)
- Multi-dimensional image processing (scipy.ndimage)
- File IO (scipy.io)
- Weave (scipy.weave)
SciPy.io

- Matlab files
  - loadmat
  - savemat
- IDL
  - readsav
- Wav sound files
- Arff files
- NetCDF
  - createDimension
  - createVariable

```
examples/3_scipy/netcdf.py

from scipy.io import netcdf

f = netcdf.netcdf_file('simple.nc', 'w')
f.history = 'Created for a test'
f.createDimension('time', 10)
time = f.createVariable('time', 'i', ('time',))
time[:] = np.arange(10)
time.units = 'days since 2008-01-01'
f.close()
```
SciPy.Stats

- Continuous distributions
- Discrete distributions
- Statistical functions
- Masked statistics functions

```python
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# Generate some data for this demonstration.
data = norm.rvs(10.0, 2.5, size=500)

# Fit a normal distribution to the data:
mu, std = norm.fit(data)

# Plot the histogram.
plt.hist(data, bins=25, normed=True, alpha=0.6, color='g')

# Plot the PDF.
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)
title = "Fit results: mu = %.2f, std = %.2f" % (mu, std)
plt.title(title)
plt.show()
```
SciPy. Bessel Functions

\[ x^2 \frac{d^2y}{dx^2} + x \frac{dy}{dx} + (x^2 - \alpha^2)y = 0 \]
SciPy. Bessel Functions

```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from scipy import *
from scipy.special import jn, jn_zeros

def drumhead_height(n, k, distance, angle, t):
    nth_zero = jn_zeros(n, k)
    return cos(t)*cos(n*angle)*jn(n, distance*nth_zero)

theta = r_[0:2*pi:50j]
radius = r_[0:1:50j]
x = array([r*cos(theta) for r in radius])
y = array([r*sin(theta) for r in radius])
z = array([drumhead_height(1, 1, r, theta, 0.5) for r in radius])

fig = plt.figure()
ax = Axes3D(fig)
ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
plt.show()
```
#!/usr/bin/env python
import numpy as np
from scipy import linalg
import matrix_vector as mv
np.set_printoptions(formatter={'float': '{: 0.3f}'.format})

def matrix_inverse(A):
    return linalg.inv(A)

def matrix_exponential(A):
    return linalg.expm(A)

def matrix_calc():
    # Call our mat_vec function
    A, x, b = mv.mat_vec()
    print "A:\n", A
    # Compute the inverse of A
    inv_A = matrix_inverse(A)
    print "Inverse of A:\n", inv_A
    # Compute the matrix exponential of A
    # Using function "matrix_exponential"
    C = ??
    print "C = exp(A):\n", C
    # Compute "A * inv_A" and store in I
    ??
    print "I =", I
    return b, C

# Call matrix_calc if "main function"
??
b, C = matrix_calc()
Automatic Offload (AO) – Stampede Only!

- Feature of Intel Math Kernel Library (MKL)\(^1\)
  - growing list of computationally intensive functions
  - xGEMM and variants; also LU, QR, Cholesky
  - kicks in at appropriate size thresholds (e.g. SGEMM: \((M,N,K) = (2048, 2048, 256)\))
  - Functions with AO

- Essentially no programmer action required
  - more than offload: work division across host and MIC
  - Tips for using MKL on Phi

---

\(^1\) For more information refer to https://www.tacc.utexas.edu/resources/software/ao
Automatic Offload – Stampede Only!

Set at least three environment variables before launching your code:

- `export MKL_MIC_ENABLE=1`
- `export OMP_NUM_THREADS=16`
- `export MIC_OMP_NUM_THREADS=240`

- Other environment variables provide additional fine-grained control over host-MIC work division
- MKL documentation
- Intel MKL Automatic Offload enabled functions
Automatic Offload – Stampede Only!

`examples/3_offload/my_dgemm.py`

### Important Variables

- `OMP_NUM_THREADS` (1..16)
- `MKL_MIC_ENABLE` (0, 1)
- `MIC_OMP_NUM_THREADS` (1..240)
- `OFFLOAD_REPORT` (0..2)
Matplotlib

Matplotlib is a python 2D/3D plotting library which:

- produces publication quality figures
- provides and interactive environment
- generates plots, histograms, power spectra, bar charts, error charts, scatterplots, etc
- allows a high level of customization
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5,6], 'o')
plt.plot([6,5,4,3,2,1])
plt.ylabel('y label')
plt.xlabel('x label')
plt.show()
import matplotlib.pyplot as plt

# add a title
plt.title('This is a title')

# get the individual plots
plt1, = plt.plot([1, 2, 3, 4, 5, 6], 'o')
plt2, = plt.plot([6, 5, 4, 3, 2, 1])

# add the legend
plt.legend([plt1, plt2], ['blue circles', 'green line'])

plt.ylabel('y label')
plt.xlabel('x label')
plt.show()
import numpy as np
import matplotlib.pyplot as plt

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()

# replace 'plt.show()' by
# plt.savefig('plot.png')
```python
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
# import the matrix_calc module as mc
??

def plot():
b, C = mc.matrix_calc()

plt.subplots(1,1)
plt.title("Matrix Exponent")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.scatter(b,b,c='red', marker='o', s=500, label="Diag")
plt.legend()
plt.show()

# Plot b versus b**2; use np.power(array, power)
??

# Plot b versus the diagonal of C (np.diag(C))
# using plt.plot and a linewidth of 5 with a
# label of "diag(C)" and a color of blue.
# Add a scatter plot of b versus np.sqrt(np.diag(C))
??

if __name__ == "__main__":
    # Call the plot function
??
```

License

© The University of Texas at Austin, 2015

This work is licensed under the Creative Commons Attribution Non-Commercial 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/3.0/

When attributing this work, please use the following text: "HPC Python", Texas Advanced Computing Center, 2015. Available under a Creative Commons Attribution Non-Commercial 3.0 Unported License.