Scientific Programming with Python: part II
Advanced Research Computing
Outline

• MPI programming in Python
• NumPy (SciPy) arrays
• Plotting in Python:
  – bar-charts, pie-charts, mesh-grids
• Plotting and image processing in Python
  – image representation, conversion to array and processing (interpolation)
About me

• Gabriel Mateescu
• I work in the Advanced Research Computing Group
• Email: gabriel@vt.edu
USING MPI IN PYTHON
MPI multiprocessing

• MPI is a widely used API for inter-process communication. It supports
  – general send, receive, broadcast operations
  – HPC-specific operations: all-to-all, reduce, all-reduce, scatter, gather
  – synchronization: barrier and left to right of expressions

• There are many implementations of MPI
  – I prefer Open MPI
MPI4Py

- MPI4Py is one of the Python modules for providing access to MPI functions in Python
- The MPI4Py functions (send, receive reduce) delegate the work to the underlying MPI implementation
- An MPI4Py program is launched like any MPI program

```
mpirun -np 4 python myprogram.py
```
from openmpi.mpi4py import MPI
import random
num_samples = 10e6
comm = MPI.COMM_WORLD
mpisize = comm.Get_size(); rank = comm.Get_rank()
nsamples = int(num_samples/mpisize)
random.seed(rank); inside = 0
for i in range(nsamples):
    x = random.random(); y = random.random()
    if (x*x) + (y*y) < 1: inside += 1
inside = comm.reduce(inside, op=MPI.SUM, root=0)
if rank == 0:
    pi = (4.0 * inside)/num_samples; print ' pi =', pi
NUMPY ARRAYS
NumPy Arrays

• NumPy arrays, also called N-d arrays are strongly typed, homogeneous arrays
  – default type is float

• N-d arrays available in NumPy and SciPy (which used the NumPy arrays)

• NumPy also defines operations on N-d arrays

• N-d arrays can be passed to Python functions that expect a list
Anatomy of Nd-arrays (1)

- all elements must be of the same dtype (datatype)
- the default dtype is float
- arrays constructed from list of mixed dtype will be upcast to the "greatest" common type

Taken from

http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/python/arrays.html
Anatomy of Nd-arrays (2)

The axes of an array describe the order of indexing into the array, e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

The shape of an array is a tuple indicating the number of elements along each axis. An existing array has an attribute a.shape which contains this tuple.

shape=(8,3)

Taken from
http://pages.physics.cornell.edu/~myers/teaching/ComputationalMethods/python/arrays.html
Constructing Arrays (1)

```python
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([i*i for i in range(10) if i%2==1])
c = np.zeros(100)  # array of float zeros
d = np.zeros((2,4), int)  # 2x4 array of int zeros
e = np.ones(10, int)  # array of int ones
f = np.ones((5,5))
i = np.eye(10,10, int)
```
Constructing Arrays (2)

\[ r = \text{np.arange}(0, 10, 2) \quad \# \text{step} = 2 \]

\[ l = \text{np.linspace}(-1.0, 1.0, 20) \quad \# \text{num points} = 20 \]

\# 10x10 array of floats uniformly on \([0., 1.)\)
\[ r = \text{np.random.random}((10,10)) \]

\# 10 random ints uniform on \([0, 5)\)
\[ ri = \text{np.random.randint}(0,5 (10,)) \]
Array Indexing (1)

last_elem = a[-1]  # the last element of the array

i_ind = np.array([0,1,2,3])  # array of indices
j_ind = np.array([1,2,3,4])  # array of indices

# return array([a[0,1], a[1,2], a[2,3], a[1,4]])
a[i_ind, j_ind]

b = np.array([True, False, True, False])

# return array([a[0], a[2]])
a[b]
Array Indexing (2)

a[1:5, 3:7]  # 4x4 sub-block starting at [1,3]
a[2, 3:]    # until end of array
a[:15, :11] # from start of array

a[:, 1]    # column 1
a[:, -8:]  # a slab of width 8
Array Operations (1)

```python
a = np.arange(0, 16).reshape(4,4)
b = np.transpose(a) ; bb = a.T

c = a + b      # add a and b element-wise
d = a * b      # multiply a and b element-wise
e = -a         # negate every element of a

f = a > 0.0    # boolean array indicating
              # which elements are > 0.0
```
PLOTTING IN PYTHON
Overview

• There are several Python packages for plotting
  – The most widely used are Matplotlib and Pylab
• Pylab gathers Matplotlib and Numpy in one namespace
• Plots, bar-charts, pie-charts are supported
• Matplotlib together with statistics packages allows to represent probabilistic distributions
Before you start

• When using Matplotlib on Linux/Mac OS X, your client machine must have an X11 server running, and you must ssh into the machine using X11 forwarding:

  $ ssh -X matplotlib_host

• If connecting from Windows you need to install an X11 server
Loading Data From a CSV File

$ more data.csv
"name","worth","age"
Donald Trump,30,65
Bill Gates,580,55
Tom Cruise,40,41
Mr Manager,10,41
Generate a bar-chart

$ more bar.py
import numpy as np
import matplotlib.pyplot as plt

data = np.recfromcsv('data.csv')
names = [ data[i][0] for i in range(len(data)) ]
worth = [ data[i][1] for i in range(len(data)) ]
ages = [ data[i][2] for i in range(len(data)) ]
N = len(data); ind = np.arange(N); width = 0.2

plt.bar(ind, worth, width, color='r', label='worth')
plt.xlabel('Worth')
plt.xticks(ind+width/2., names )
plt.legend()
plt.title('worth by person')
plt.show()
Thank you.

Questions?