Cornell University
Center for Advanced Computing

Scripting for Data Analysis

Drew Dolgert
Cornell Center for Advanced Computing

Today’s Task

- Not focused on learning R or Python.
- Not showing tour-de-force of cool scripts.
- Focus on combination of scripting and numerical analysis.

What is a Script?

- Go to ~/python
- python simple.py
- Edit it with vi.
- python simple.py

What's the missing step, compared with C?

What Scripting Languages Do We Use?

- Python
- R
- What else?

Why are Scripting Languages Important?

- Dynamic binding ubiquitous, so they pull in lots of libraries.
  - Graphics – Matplotlib, VTK, gnuplot
  - Numerics – Numpy, Scipy
  - Data Transformation – XML, binary packing, NetCDF, HDF
  - Networking – MPI, easy TCP/IP, web services
- Want to do those things in Fortran?

Can’t Fortran use libraries, too?

Why are Scripting Languages Important?

- Languages have nice features that C and Fortran can’t afford.
  - Don’t declare types.
  - Query an object for its type.
  - Inherently object-oriented and/or functional programming styles.
  - Many fewer lines of code for the same task.
- Helps with building GUIs, translating file formats, partitioning large tasks, testing algorithms.

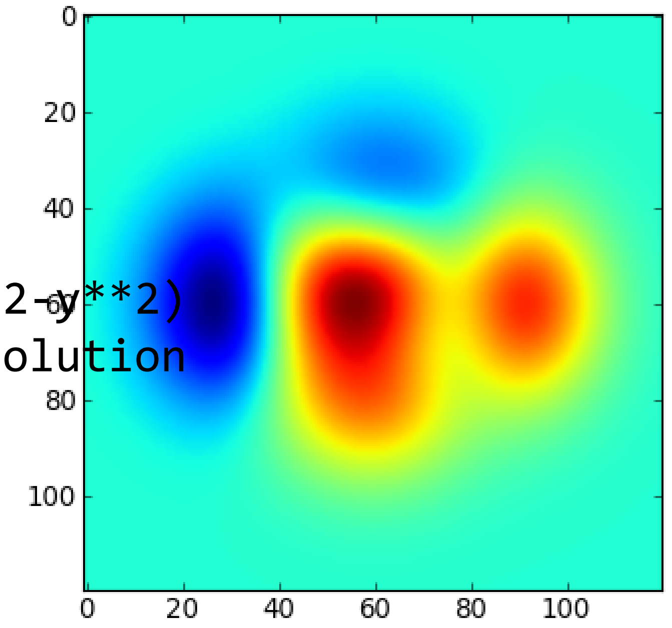
What Python Looks Like

```
import numpy as np
C = np.empty([150,100],dtype=‘i’)
for k in np.arange(0,150):
  for j in np.arange(0,100):
    C[k,j] = compute(k,j)
```

Leading spaces with consistent indent.
Colons indicate start of indentation block.
No declaration of types.

Plotting with Matplotlib

```
from __future__ import division
from pylab import *
def func3(x,y):
  return (1- x/2 + x**5 + y**3)*exp(-x**2-y**2)
# make these smaller to increase the resolution
dx, dy = 0.05, 0.05
x = arange(-3.0, 3.0, dx)
y = arange(-3.0, 3.0, dy)
X,Y = meshgrid(x, y)
Z = func3(X, Y)
ax = subplot(111)
im = imshow(Z, cmap=cm.jet)
im.set_interpolation('bilinear')
show()
```



Can make interactive plots.
Make them right after calculation.

from http://matplotlib.sourceforge.net/examples/pylab_examples/pcolor_demo2.html

XML and Binary Processing

```
dom=parseString(xmlIn)
jobs=dom.getElementsByTagName(‘job’)
jobIds=list()
for job in jobs:
  jobIds.append(
            job.attributes[‘JobID’].nodeValue)
# Write as binary unsigned integers.
for writeJob in jobIds:
  f.write(struct.pack(“B”,writeJob))
```

Math Scripting

- `source pythonpath`
- Invert random matrix with `python/pure_invert.py`
- `time python pure_invert.py 100`
- Increase size
- Is the code reasonable?

Numpy Module for Inversion

- Invert with Numpy
- `time python numpy_invert.py 100`
- What is different in this code?

No Big Computations

- Numpy uses LINPACK
- It’s a rule: do big chunks in libraries

R

- Language adept at statistical computing.
- C-ish in appearance but object-oriented and scoping like functional programming.
- Lots of precise stats functions.
- Built-in plotting.
- Lots of libs.
- Gnu Public License

R in Batch

- batchr directory sends insects to eat corn or die.
- Run once locally
- ~train100/bin/R –no-save –args 0 < main.R > z0.txt
- For a sense, see Kernels.R.
  - distributions$Maxdist is a struct member.
  - Assignment done with <-. Return val is last statement.

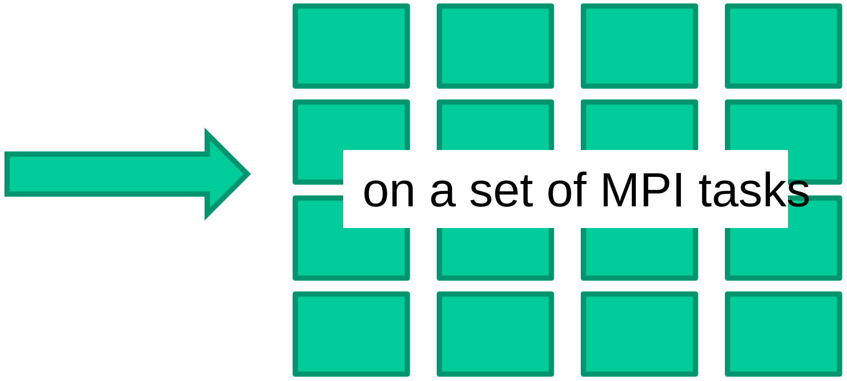
Run Insect Code in Parallel

- Look at ranger.sh for how batch works.
- Modify ranger.sh to run many copies of main.R.

farm-out-work

Execute lines from text file

```
./mycalc –init 0 > mycalc0.out
./mycalc –init 1 > mycalc1.out
./mycalc –init 2 > mycalc2.out
```



- Code in ~train100/farm-mpic
- Executable at ~train100/bin/farm
- ibrun runs things under MPI
- `ibrun ~train100/bin/farm -v -t tasks.txt`

Running Subprocesses

- Script is in charge
- Runs other codes in sequence or parallel
- Converts in-between

```
from subprocess import Popen, PIPE
calcProcess = Popen(“./calcStuff”, stdout=PIPE, stderr=PIPE)
(out,err) = calcProcess.communicate()
if out.find(‘adjusted’)>=0:
    print “Calculation adjusted.”
else:
  print “Calculation fixed.”
```

Parallel Python

- MPI with mpi4py or mpipython.
- IPython
- Twisted
- Multiprocess

- But if it’s C and Fortran that do all the real work, MPI within Python is too fine-grained…

Wrap Code with Script

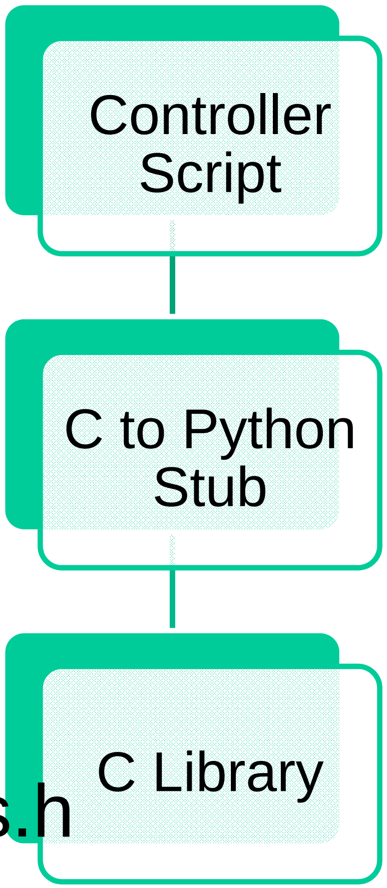
- Take a simple code
  - double* init(int atomCnt)
  - void move_atoms(atoms,atomCnt)
  - double atom_temp(atoms,atomCnt)
  - void retemp_atoms(atoms,atomCnt)
- Run with a main.c
- Run from Python

Compile the C version

- In swigatoms directory, in makefile, look at “make exe”
  - gcc main.c main.c atoms.c –o atoms
- Check out loops in main.c
- Run with ./atoms

- What if you want to change the loops?

Run C Under Python



5. Script imports stub
4. setup.py installs module
3. icc compiles stub
2. SWIG generates _atoms_module.c
1. SWIG reads atoms.h

Compile the Python/C Hybrid

- Make a shared library from the C code
  - “make swig” in the makefile. Check it out.
  - Creates _atoms_module.c and _atoms.so.
- Python -> atoms_module.c -> atoms.c
- Ask Python to install module in its library.
  - python setup.py install –prefix=$HOME

- What does that get me?

Run Atoms Interactively

- python
- import atoms
- dir(atoms)
- a=atoms.init(32)
- dir(a); print a
- Try the main.c loops interactively.

- Why is this any better?

Simulated Experiments are Complex

- Control of complex boundary conditions, external forces
- Implement in main.c.
- Write a control script, implemented with lex and yacc.
- Or run under Python / Ruby.

Advantages / Drawbacks

- Wrapping is an extra step, sometimes hairy.
- Faster or slower?

Distributions

- Enthought
- www.python.org
- ActiveState

- 2.3 – works, some packages past it
- 2.5 – current, supports most numerical packages now
- 2.6 – has language features that ease transition to 3.0
- 3.0 – not yet for scientific packages