Day 15: Science Code in Python
Homework Review
Science Code in Python?
Custom Code vs. Off-the-Shelf

• Trade-offs
  – Costs (your time vs. your $$$)
  – Your time (coding vs. learning)
  – Control of software (features, schedule, license, …)
  – Fit of software to problem at hand
  – Reliability

• Rarely a trivial decision
Efficiency of Python

• Python vs. C, C++, Fortran, …

• Example: Prime-number checker (Homework #10)
  – About the same length of program
  – C was about $20 \times$ faster than Python

• Example: Word-frequency counter (Homework #4)
  – C program would be much longer
  – Or, find reliable libraries for things like dictionary
  – Probably still much faster to run, but maybe not $20 \times$

• So… whose efficiency are you measuring?

• Anyway, Python can call compiled C/C++ functions
Does Efficiency Even Matter?

Efficiency

Correctness

Clarity
The Story of Mel

http://rixstep.com/2/2/20071015,01.shtml
Does Efficiency Matter in CHTC?

• No
  – Your time matters… let machines do extra work
  – Code clarity matters… let machines do extra work
  – Increase parallelism… let machines (oh, you know)

• Yes
  – Fair share: The more you use, the less you get
  – Efficient code finishes sooner (e.g., deadlines)

• Maybe
  – Time scale may be a factor (1 vs. 20 seconds? days?)
Science Code in Python
Numeric and Scientific Modules

• Many numeric/scientific computing modules exist
• http://wiki.python.org/moin/NumericAndScientific
• DO NOT REINVENT THE WHEEL!
NumPy: Getting Started

- **NumPy**: Large collection of modules, Python and C, for performing efficient numeric computations

- **Installation required**
  - Includes compiled code, so non-trivial install
  - Ask sysadmin for help!
  - But, *already installed* on CHTC machines

- Visit website for tutorials, examples, etc.
NumPy: Basic Types

• Precise scalar types
  – Not just `int`, but `byte, short, int8, uint64, ...`
  – Not just `float`, but `single, double, float128, ...`

• N-dimensional arrays
  – Viewed as multidimensional arrays or matrices
  – All elements are same type (e.g., `uint64`)
  – Lots of natural operations (e.g., `a + b, conversions, ...`)

• Dates and times
  – Even more expressive than Python built-ins
  – Offsets by year, month, day, hour, ..., attosecond
  – Business days!
NumPy: Universal Functions

• Functions that operate on *elements* of *N*-dim arrays
• More efficient than looping through yourself
• Allow compact expression of vector math

• Examples:
  – add, subtract, multiply, divide, …
  – rint (round to int), sign, negative, …
  – log, log2, log10, sqrt, square, reciprocal, …
  – sin, cos, tan, arcsin, sinh, arcsinh, …
  – bitwise_and, invert, left_shift, …
  – greater, greater_equal, less, less_equal, equal, …
  – maximum, minimum
NumPy: Examples

# ~3.45 secs
a = range(10000000)
b = range(10000000)
c = [a[i] + b[i] for i in xrange(len(a))]

# ~0.25 secs
a = numpy.arange(10000000)
b = numpy.arange(10000000)
c = a + b

a = numpy.array([[-2, 2, 3],
                 [-1, 1, 3],
                 [ 2, 0, -1]])
print numpy.linalg.det(a) # => 6.0
NumPy: Other Features

• HUGE collection of numerical routines

• Highlights:
  – Array creation, manipulation, indexing, input/output
  – Fast Fourier Transforms
  – Linear algebra (matrix math)
  – Random sampling (~35 distributions)
  – Statistics (extremes, central tend., var., histograms)
  – Polynomial math (incl. some basic calculus)
SciPy: Getting Started

- **SciPy**: Large collection of modules, Python and C, for performing scientific computations
- Same as NumPy for installation and efficiency
- Also on CHTC execute machines
SciPy: (Some) Features

• HUGE collection of routines (again)!
• Examples:
  – Functions for mathematical physics
  – Integration, incl. ordinary differential equations
  – Numerical optimization algorithms
  – Variable interpolation
  – Signal processing
  – Linear algebra (again); MATLAB-like syntax, functions
  – Sparse matrices
  – More stats; R-like functionality
  – Clustering algorithms
SciPy: Example

Solve system of linear equations:

\[
\begin{align*}
  x + 3y + 5z &= 10 \\
  2x + 5y + z &= 8 \\
  2x + 3y + 8z &= 3
\end{align*}
\]

```python
>>> A = mat('[[1 3 5; 2 5 1; 2 3 8]]
>>> A
matrix([[1, 3, 5],
        [2, 5, 1],
        [2, 3, 8]])

>>> b = mat('[[10;8;3]]

>>> linalg.solve(A, b)
array([[ -9.28],
       [  5.16],
       [  0.76]])
```
Python vs. R, MATLAB, Octave, …

• Trade-offs!

• Could do everything in Python
  – Consistency
  – No need to move data back and forth

• R / MATLAB / Octave
  – If you already know/use it… why stop?
  – Use Python for wrappers, workflow
Python Jobs for CHTC
Making Python Jobs That Fit CHTC

• Independent batch jobs, 10 minutes – 4 hours

• Python (carefully written) works on many platforms
  – Write submit file to access them (e.g., RHEL 6 trick)
  – Watch out for platform and Python version differences

• Using NumPy/SciPy makes code less portable
  – May need to bring it with you
  – Still may be more portable than compiled C…

• Work on good parallelization

• Long-running jobs? implement self-checkpointing
Self-Checkpointing: Why?

• Suppose your job will run for a long time (> 30 m?)
• May be preempted
• HTCondor will re-run job
• But that means it starts over

• One solution:
  – Periodically write state (checkpoint) to disk
  – Must be sufficient to restart job at that point
  – Job itself must know to look for checkpoint data
  – May need wrapper script to accomplish
Self-Checkpointing: When?

• Balance cost of overhead vs. risk of bad-put
  – Writing anything to disk is slow (relatively speaking)
  – If there is little data, can write more often

• Look for natural checkpoint times
  – Generally, when there is the least data to write
  – Typically, between outermost iterations
  – Could base on iteration count, time, …

• Save only what you need

• Be sure to flush or close checkpoint each time!
Self-Checkpointing: HTCondor Tweak

- Must tell HTCondor to transfer your output back to the submit machine, even when just evicted and waiting for next run
- HTCondor holds files for you, then moves to next machine automatically

\[
\text{when\_to\_transfer\_output} = \text{ON\_EXIT\_OR\_EVICT}
\]
Self-Checkpointing: Writing a Checkpoint

- Simplest example
  - Assume a 1D parameter sweep
  - Assume real code appends to its output each iteration
  - Designed to save checkpoint every 1000\textsuperscript{th} iteration

```python
def save_checkpoint(iter):
    cp_file = open(checkpoint_path, 'w')
    cp_file.write('%d\n' % (iter))
    cp_file.close()

for iter in xrange(start, end + 1):
    do_stuff(iter)
    if ((iter - start + 1) % 1000) == 0:
        save_checkpoint(iter)
```
Self-Checkpointing: Using a Checkpoint

• Continuation of previous example...

```python
if len(sys.argv) != 3: # Handle error
    start, end = map(int, sys.argv[1:])

if os.path.exists(checkpoint_path):
    cp_file = open(checkpoint_path, 'r')
    cp_data = cp_file.readlines().strip()
    cp_file.close()
    cp_start = int(cp_data)
    if cp_start >= start:
        start = cp_start
    else:
        # Potential problem?
```
Final Questions & Thoughts?
Reminder About CHTC Accounts

• CHTC accounts will go away in January
  – Feel free to copy your files off ahead of time

• To get a real account:
  – Email chtc@cs.wisc.edu
  – Include:
    ✦ That you took CS 368 with me this semester
    ✦ Your current username on CHTC
    ✦ Your Principal Investigator’s name
    ✦ A brief (2–3 sentence) description of your project
Homework
Homework

• Use CHTC!
• Do cool new research
• Let us know what you accomplish

Any sufficiently advanced technology is indistinguishable from magic.

— Arthur C. Clarke