Python for scientific algorithm development

Fernando Perez
Applied Mathematics

CU Boulder

Scipy'06 – Caltech
August 17 2006
I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code.

Good programmers worry about data structures and their relationships.

-- Linus Torvalds
Initial remarks

- Does anyone have my USB drive from the tutorial?
- I gambled on the audience: more scientists new to python than experts on the language.
- I'll try to keep it interesting for the experts.
- Light on the math/physics, used mostly for illustration purposes. Talk to me if you care.
- $$$: DARPA, DOE.
- Collaborators:
  - Gregory Beylkin (CU Boulder)
  - Martin Mohlenkamp (Ohio Univ.)
  - Robert Harrison, George Fann (ORNL)
Modern algorithms: *There is more to life than arrays*

Today's scientific software needs:

- Complex data structures
- To work at a high level of abstraction
- To handle mixed data (genetic sequences, time-tagged data, information from databases...). *It's not just floating-point.*
- Interact with other external systems (web, hardware, software subsystems, ...)

Q: how can we write better scientific software, and do it faster?

A: I don't know...

But I think that Python can help
Dictionaries: flexible, efficient and powerful hash tables

Strings: lots of useful methods.

def word_freqs(text):
    """Return a dictionary of word frequencies for the given text."""

    freqs = {}
    for word in text.split():
        freqs.setdefault(word, 0)
        freqs[word] += 1
    return freqs
class Tree:
    """A binary tree class.""
    def __init__(self, label, left=None, right=None):
        self.label = label
        self.left = left
        self.right = right

    def inorder(t):
        """Return the leaves of t in left-right order.""
        if t:
            for node in inorder(t.left):
                yield node
            yield t.label
            for node in inorder(t.right):
                yield node
Interactive
Explore ideas, data with your fingers

VS
Other good things...

- Choose your code style. Write
  - standalone all-global scripts, or...
  - procedural code, or ...
  - Object Oriented libraries (with a simple object model compared to C++), or...
  - in a functional style.

  Choose the code that best fits your brain or the problem.

- Reuse your existing code (Fortran, C/C++).
- Uniformity (functions are first-class objects).
- Optimize only what really needs speed.
A mathematical problem:

Accurate, adaptive, fast algorithms for:

\[ g = T f \iff g(x) = \int K(x - y) f(y) d^n y \]

where \( n = 2, 3, \ldots, 6 \). (Integral formulations are nice...)

- Electrostatics (Poisson's equation)
- Electrodynamics (Helmholtz)
- Quantum mechanics (Schrodinger – Lippman-Schwinger)
- Lots more...

Yes, this is 'just' a matrix-vector product.
What do you need?

\[ g(x) = \int K(x - y) f(y) \, d^n y \]

Operators:
- Sparse
- Good scaling with \( n \)

Functions:
- Adaptively represented
- Compatible with op. rep.

Some ideas (I won't go into the details):
- Gaussian expansions for the kernel \( K(x-y) \)
- Multiwavelets (think tensor products of Legendre polynomials)
- Adaptive 2-\( n \) trees for function decomposition
What do you want from your code?

• Make functions easily:

\[ f = \text{from_snippet}(\text{nnod}=6, \text{ndim}=2, \text{cutoff}=1e-6, 1.0, \text{'}return \sin(x)\times\cos(y)\text{'}\) \]

• Write code that reads like math (see demo):

\[ h(x) = g(x) - f(x) \]

• How do we do this?
  • scipy's weave.inline
  • dictionaries
  • easy but powerful string handling
  • in-process calling of multiple different libraries (even written in different languages)
Functions: simple adaptive decompositions
Operators: sparsity and high order methods

Multiwavelets (illustrated for $d=1$):

From this to this
Putting it all together (see code)

Redundant tree of input (output skeleton)

Terminal

Non-terminal
The Hydrogen ground state

Plain old Schrodinger

\[ H \psi = E \psi \Leftrightarrow \left[ -\frac{1}{2} \nabla^2 - \frac{1}{r} \right] \psi = E \psi \]

can be written as

\[ \phi = -2 G_\mu V \phi \Leftrightarrow -2 \left( -\nabla^2 + \mu^2 I \right)^{-1} V \phi \]

and at

\[ \mu = \sqrt{-2 E} \Rightarrow \psi = \phi \]

We can try to solve this by iterating to a fixed point:

1) Initialize all variables
2) Compute \( \phi_{\text{new}} = -2 G_\mu V \phi_{\text{old}} \)
3) New Energy, new mu, repeat: \( E = \frac{\langle \psi | H | \psi \rangle}{\langle \psi | \psi \rangle} \Rightarrow \mu = \sqrt{-2 E} \)
# Green's function iteration for the ground-state
# of the Hydrogen atom
# Initialize:
# \( V \) = potential \((1/r)\)
# \( \psi \) = guess wavefunction
# \( G \) = Bound-state Helmholtz operator

```python
for n in range(num_iterations):
    # Apply the Greens' function with current value of \( \mu \)
    psi = -2*G(V*psi)
    # Computation of energy as \( E = \langle \psi | H | \psi \rangle / \langle \psi | \psi \rangle \)
    E = (-1/2.)*psi.weak_laplacian(psi) +
        V.innerproduct(psi*psi) / psi.norm_l2()**2
    # Update the operator to new value of \( \mu \)
    G.mu = sqrt(-2*E)
    # Prepare wavefunction for next iteration
    psi.normalize()
```
Green's function iteration: convergence
A brief summary

• Write code that reads like the science you care about.
  - Python offers a lot of tricks to let you do that.
• Express new ideas and algorithms as simply as possible.
• Put as many tools to explore your data as you can
  - You debug with the same methods you produce plots for a paper.
• Take advantage of excellent libraries.
• f2py, weave.inline (and .blitz), ctypes, pyrex, ...
  ... are very easy to use.
• Have fun coding science!