Scientific computations within Python: numerical and analytical tools [a]

Pearu Peterson

pearu.peterson@gmail.com

Laboratory of System Biology

Institute of Cybernetics at TUT, Estonia

- What is Scientific Computation about?
- A software problem, solutions, examples
- F2PY — the connection between Python and Fortran
- SciPy — Scientific Tools for Python
- Sympy/SympyCore — Symbolic manipulation libraries for Python

[a] *Abstract:* Python is a very powerful high-level programming language that is especially suitable for proto-type software development - this is what scientific computing is often all about. Within the last 10 years many tools have been developed for Python to carry out various scientific computing tasks. In this presentation I'll give a short overview of the tools like F2Py and SciPy that are often used for numerical computations and comment on the attempts to implement Computer Algebra Systems to perform some analytical computations within Python.

Scientific Computation is about ...

- constructing mathematical models — *software math*
- developing numerical solution techniques — *software hardware math*
- analyzing and solving scientific problems — *scientific*

...using computers.

A software problem in Scientific Computation

What software tools to use for a specific problem?

Condition

The aim of a Scientist is to solve scientific problems.

Optimal solution

Minimizes model construction and computational efforts.

Extreme solutions

1. **Use of high-level interactive environments**
   - Matlab, Python, etc.,
   - Easy to learn and use $\Rightarrow$ rapid model development,
   - Universal components $\Rightarrow$ slow computations.
2. **Use of low-level programming tools**
   - C/C++, Fortran, etc.,
   - More to learn and write $\Rightarrow$ slow model development,
   - Design own components $\Rightarrow$ (possibly!)[a] fast computations.

[a] Using low-level tools does not automatically mean better performance. It certainly gives more options to shoot yourself in the foot...

Proposed optimal solution

3. Use of low-level programming tools from high-level interactive environment

- Call high-performance C/C++/Fortran routines from high-level environments
- Computationally intensive parts in a low-level language $\Rightarrow$ fast computations
- Model construction and experiment setup in a high-level language $\Rightarrow$ rapid model development
- 10 years ago: requires mixed language programming skills $\Rightarrow$ even more to learn and write :(
- BUT(!) nowadays: use automatic code and wrapper generators $\Rightarrow$ an amazingly easy way to get huge performance boosts and rapid model development

Example

Problem:

Solve 2D Laplace problem $\Delta u = 0$ with specified boundary condition for $u(x, y)$

Algorithm:

$u_{i,j} \equiv u(i\delta_x, j\delta_y)$, $i = 0 \ldots n_x - 1$, $j = 0 \ldots n_y - 1$

while iteration index < 100 **do**

  for i = 1 to $n_x - 2$ **do**

    for j = 1 to $n_y - 2$ **do**

$$u_{i,j} = \frac{(u_{i-1,j}+u_{i+1,j})\delta_y^2+(u_{i,j-1}+u_{i,j+1})\delta_x^2}{2\delta_x^2+2\delta_y^2}$$

Timings:[a]

Tools	Time taken (sec)
Python	1500.0
Python+NumPy expression	29.3
Blitz	9.5
Python+Fortran	2.9
Pyrex/Cython	2.5
Matlab	29.0
Octave	60.0
Pure C++	2.16

[a] Source: http://www.scipy.org/PerformancePython/

Fortran and Python connection

Fortran

- A (relatively) low-level compiled programming language
- Dominant language for scientific computing
- High-performance high-quality algorithms available

Python

- Interpreted interactive object-oriented programming language
- Powerful high-level data types, useful modules, easily extendable
- Very clear syntax, ideal language for prototype development

F2PY — Fortran to Python interface generator

- To reuse available Fortran code within Python
- To extend Python with high-performance computational modules
- Also suitable for wrapping C libraries to Python
- Available since 1999, stable and complete for wrapping Fortran 77 codes

F2PY example

```
c file: dot.f
      FUNCTION dot(n, x, y)
c     dot product of two vectors
      INTEGER n, i
      DOUBLE PRECISION dot, x(n), y(n)
      dot = 0d0
      DO i = 1, n
        dot = dot + x(i) * y(i)
      ENDDO
      END
```

```
$ f2py dot.f -m foo -c
```

```
>>> from foo import dot
>>> dot([1,2],[3,4])
11.0
```

F2PY features

- Scans Fortran codes for subroutine/function/data signatures
- To call Fortran 77/90, Fortran 90 module, and C functions from Python
- To access Fortran 77 COMMON blocks and Fortran 90 module data (also allocatable arrays) from Python
- To call Python functions from Fortran and C (callbacks)
- Handles Fortran/C data storage issues
- Generates documentation strings
- Supports compilers: Absoft, Compact/Digital, HPUX F90, IBM XL, Intel, Lahey/Fujitsu, MIPSpro, NAGWare, Portland, Sun/Forte/WorkShop, Pacific-Sierra, Gnu, GFortran, G95
- Author: Pearu Peterson
- F2PY is part of NumPy — provides multi-dimensional array object
- **`http://www.f2py.org`**

Limitations

- Lack of support for Fortran 90 derived types and pointers — work-in-progress

SciPy — Scientific Tools for Python

- Collections of modules under **scipy** namespace:

  fftpack : discrete Fourier transform algorithms
  integrate : integration routines and differential equations solvers
  interpolation : interpolation tools
  linalg : linear algebra routines
  ndimage : n-dimensional image tools
  optimize : optimization tools
  signal : signal processing tools
  sparce : sparce matrices
  stats : statistical functions
  special : definitions of many usual math functions
  . . . IO and other utilities, C/C++ integration, etc

- The modules contain F2PY or Pyrex generated wrappers to high-performance libraries: ATLAS, BLAS, LAPACK, FFTW, ODEPACK, CEPHES, QUADPACK, ODRPACK, etc
- Authors: Eric Jones, Travis Oliphant, Pearu Peterson, and others
- Large and active community — $\sim$ 44 messages per day
- **`http://www.scipy.org`** — $\sim$ 7000 hits per day

On providing Computer Algebra System for Python[a]

SymPy

- A Python library for symbolic mathematics

```
>>> from sympy import *
>>> x, y = Symbol('x'), Symbol('y')
>>> x+y-x
y
>>> limit(sin(x)/x, x, 0)
1
>>> diff(sin(2*x), x)
2*cos(2*x)
>>> integrate(cos(x)+x,x)
(1/2)*x**2 + sin(x)
>>> cos(x).series(x,5)
1 - 1/2*x**2 + (1/24)*x**4 + O(x**5)
```

  pattern matching, arbitrary precision numbers, functions, symbolic matrices, Pauli and Dirac algebra, some algebraic and differential eqn. solvers, plotting, etc.

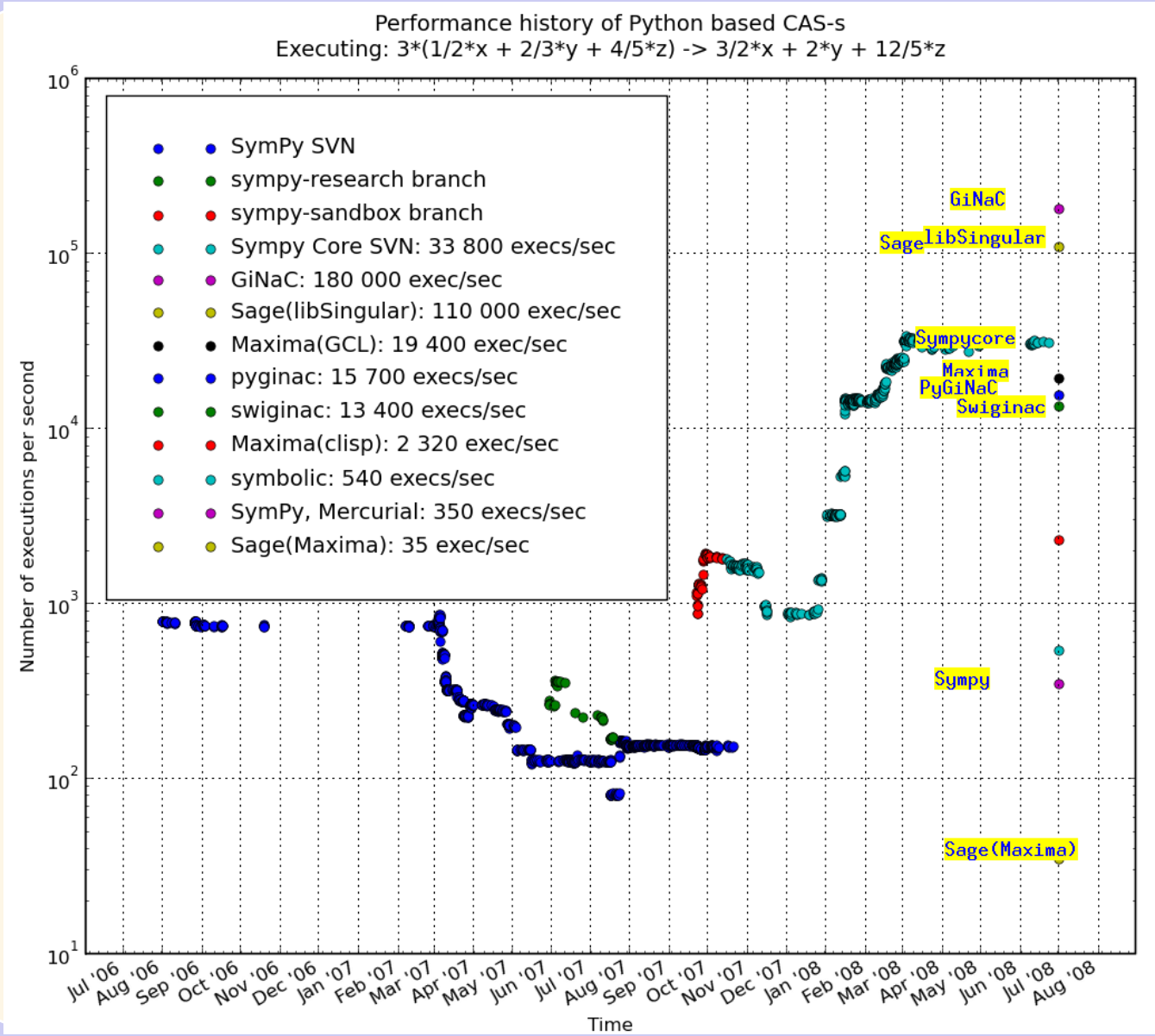
- Authors: community effort
- Slowish, restricted to calculus

[a] It is almost trivial to implement a simple and efficient Python program for manipulating symbolic expressions but highly non-trivial to generalize it to a full-featured and sufficiently efficient CAS

SympyCore

- A research project. The aim is to seek out new high-performance solutions to represent and manipulate symbolic expressions in Python language
- Uses algebraic approach, supports multiple representations[a]
- Supports various mathematical concepts: arithmetics, calculus, polynomials, matrices, sets, logic, functions, operators, etc.
- Sympycore is the fastest Python based CAS core implementation (10-300x faster than Sympy)
- Our goal is to work in the direction of making SympyCore usable for Sympy
- Authors: Pearu Peterson, Fredrik Johansson

[a] There exist no ideal representation of mathematical concepts in a computer program, efficiency depends on a particular application and used algorithms


Performance history of Python based CAS-s
Executing: 3*(1/2*x + 2/3*y + 4/5*z) -> 3/2*x + 2*y + 12/5*z
SymPy SVN
sympy-research branch
sympy-sandbox branch
Sympy Core SVN: 33 800 execs/sec
GiNaC: 180 000 exec/sec
Sage(libSingular): 110 000 exec/sec
Maxima(GCL): 19 400 exec/sec
pyginac: 15 700 execs/sec
swiginac: 13 400 execs/sec
Maxima(clisp): 2 320 exec/sec
symbolic: 540 execs/sec
SymPy, Mercurial: 350 execs/sec
Sage(Maxima): 35 exec/sec
GiNaC
Sage
libSingular
Sympycore
Maxima
PyGiNaC
Swiginac
Sympy
Sage(Maxima)
Number of executions per second
Time
Jul '06
Aug '06
Sep '06
Oct '06
Nov '06
Dec '06
Jan '07
Feb '07
Mar '07
Apr '07
May '07
Jun '07
Jul '07
Aug '07
Sep '07
Oct '07
Nov '07
Dec '07
Jan '08
Feb '08
Mar '08
Apr '08
May '08
Jun '08
Jul '08
Aug '08

Summary of useful tools for Python

NumPy : provides efficient multidimensional array object and F2PY
http://www.scipy.org/NumPy

SciPy : provides many scientific tools (Matlab numerical functionality)
http://www.scipy.org

IPython : an interactive computing environment
http://ipython.scipy.org

matplotlib : a 2D plotting library for publication quality figures (Matlab plotting functionality)
http://matplotlib.sourceforge.net

sympy/sympycore : symbolic mathematics libraries:
http://sympy.google.com
http://sympycore.google.com

etc.

Thanks!