[simula . research laboratory]

The G3 F2PY for connecting Python to Fortran 90 programs

Pearu Peterson

pearu@simula.no

- F2PY —
  What is it? Example. What more it can do? What it cannot do?
- G3 F2PY —
  The 3rd generation of F2PY. Aims and status.
- New technologies —
  Fortran 66-2003 parser, ExtGen, SymPy.

F2PY — the connection between Fortran and Python

Fortran

- 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 usage 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
>>> print dot.__doc__
dot - Function signature:
  dot = dot(x,y,[n])
Required arguments:
  x : input rank-1 array('d') with bounds (n)
  y : input rank-1 array('d') with bounds (n)
...
>>> dot([1,2],[3,4])
11.0
```

F2PY features

- scans Fortran codes for subroutine/function/data signatures
- call Fortran 77/90, Fortran 90 module, and C functions from Python
- access Fortran 77 COMMON blocks and Fortran 90 module data (also allocatable arrays) from Python
- call Python functions from Fortran and C (callbacks)
- handle Fortran/C data storage issues
- generate 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
- F2PY is part of NumPy — provides N-dimensional array object
- **`http://www.scipy.org/F2py`**

Limitations

- lack of support for Fortran 90 derived types and pointers

G3 F2PY — the Third generation of F2PY

Aims

- Fortran 90 derived types support
- Fortran 90 pointer support
- improve extensibility of F2PY
  - current F2PY code is readable to too few people (me, ...?)

Status

- Fortran 66-2003 reader and parser — completed
- Code analyzer — mostly completed
- Python wrapper generator — completed for scalars
- Python inline interface available, an example will follow

Work in progress — new technology

- Python wrapper generator for arrays, types, pointers
- Fortran 66-2003 parser — suitable for implementing wrappers or translators of Fortran codes to any language
- ExtGen — a python extension module generator
- SymPy (core) — a symbolic expression parser and manipulator

G3 F2PY inline usage example

```
>>> from numpy.f2py.lib.main import compile
>>> code = '''c comment
...       subroutine foo(a)
...       integer a
...       print*, "a=", a
...       end
... '''
>>> m, = compile( # tell f2py that code is Fortran 77
  'c -*- f77 -*-\n'+code,
  'mymodule')
>>> m.foo(3)
 a= 3
>>> m, =compile(code, 'mymodule2',
  extra_args=['--fcompiler=gnu95'])
>>> m.foo(3)
 a=           3
```

For more information, see

http://projects.scipy.org/scipy/numpy/wiki/G3F2PY

ExtGen

- a high-level tool for constructing and building python extension modules
- no extension module writer background required
- **http://www.scipy.org/ExtGen**

Hello example

```
>>> from numpy.f2py.lib.extgen import *
# define extension module component:
>>> m = PyCModule('foo')
# define function component:
>>> f = PyCFunction('hello')
# put a C statement into function body:
>>> f += 'printf("Hello!\\n");'
# add the function to module:
>>> m += f
# compile, build, and return extension module object:
>>> foo = m.build()
>>> foo.hello()
Hello!
>>> print m.generate()
...
```

A Simple Example

(from "Extending and Embedding the Python Interpreter" manual)

```
>>> from numpy.f2py.lib.extgen import *
>>> system = PyCFunction('system',
  PyCArgument('command', 'c_const_char_ptr',
    description='a shell command string'),
  PyCReturn('sts','c_int',
    description='status value returned by shell command'),
  title='Execute a shell command.')
>>> system += 'sts = system(command);'
>>> module = PyCModule('spam', system)
>>> spam = module.build()
>>> sts = spam.system('pwd')
/home/pearu/svn/numpy/numpy/f2py/lib
>>> print spam.system.__doc__
  system(command) -> sts
Execute a shell command.
:Parameters:
  command : a to C const char ptr convertable object
    a shell command string
:Returns:
  sts : a to C int convertable object
    status value returned by shell command
```

SymPy

- F2PY needs a symbolic manipulation tool for computing array dimensions (in Fortran array index starts from 1, in Python and C from 0) → **symbolic** - a symbolic manipulation package in Python
- **SymPy** — a Python library for symbolic mathematics
- **symbolic** is now the core of **SymPy** — 10 to 100x faster than the original core

```
>>> 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.

Links

F2PY
`http://www.scipy.org/F2py`

G3 F2PY
`http://projects.scipy.org/scipy/numpy/wiki/G3F2PY`

ExtGen
`http://www.scipy.org/ExtGen`

SymPy
`http://code.google.com/p/sympy`