f2py : Fortran/C Interface

Neelofer Banglawala nbanglaw@epcc.ed.ac.uk
Kevin Stratford kevin@epcc.ed.ac.uk

Original course authors:
Andy Turner
Arno Proeme

Reusing this material

www.archer.ac.uk

support@archer.ac.uk

EPSRC
Engineering and Physical Sciences Research Council

NERC
SCIENCE OF THE ENVIRONMENT

CRAY
THE SUPERCOMPUTER COMPANY

epcc

THE UNIVERSITY OF EDINBURGH

[f2py] Why interface Fortran or C?

- Provide glue to dynamically organise code
  - Handle complex software coordination provided by Python
- Combine performance of compiled codes with flexibility of Python
  - e.g. incorporate Python analysis and visualisation into existing codebase
  - Provide flexible way to extract results from code using Python
- Reuse code that you already have
  - Gradually introduce new functionality using Python
- *f2py* command-line executable and module come with NumPy
- More info:
  - http://docs.scipy.org/doc/numpy-dev/f2py/
  - http://scipy-cookbook.readthedocs.org
  - http://www.f2py.com/home/

[f2py] Interface with Fortran

- You need to provide *f2py* with:
  - Fortran source code
  - signature file : a file describing the external function and its arguments (*f2py* can help you generate this)
  - Also need access to a Fortran compiler
- *f2py* can :
  - create a signature file containing argument attributes (e.g. *depend*, `optional`) that define the Fortran-Python interface
  - wrap Fortran code in an extension module (*e.g. .so, .pyd* files) that can be called from within Python

[f2py] General recipe

1. **Create a signature file**
   - write your own or
   - *f2py <source_file> -m <module_name> -h <signature_file>.pyf*
   - Typically the signature filename is the same as the source filename
2. **Check the signature file for correctness**
   - Sequence and types of arguments to be passed from Python to Fortran function
   - Argument attributes, such as *depend*
3. **Produce the final extension module**
   - *f2py -c <signature_file> .pyf <source_file>*
4. **Import module into Python and use the external Fortran function!**
   - *from <module> import <function>*
   - The source filename may not be the same as the function name

[f2py] Fortran : `farray_sqrt.f90`

Let's look at an example: *farray_sqrt.f90* takes input array **a_in** of length **n** and returns **a_out**, an array of the square-root of each element of **a_in**

```
! file farray_sqrt.f90
! Fortran Example : calculate the sqrt of each array element
subroutine array_sqrt(n, a_in, a_out)
            implicit none
            integer, intent(in) :: n
            real*8, dimension(n), intent(in) :: a_in
            real*8, dimension(n), intent(out) :: a_out
            integer :: i
            do i = 1, n
                a_out(i) = sqrt(a_in(i))
            end do
end subroutine array_sqrt
```

[f2py] Create a signature file

- f2py can try to create the signature file (*farray_sqrt.pyf*) automatically
  - from a terminal, issue the command:
    f2py farray_sqrt.f90 -m farray -h farray_sqrt.pyf
- The Python module will be called: farray
  - use the *-m* option
- Signature in text file called: *farray_sqrt.pyf*
  - use the *-h* option
  - will not overwrite an existing signature file:
    Signature file "./farray_sqrt.pyf" exists!!! Use --overwrite-signature to overwrite.

In [ ]:
```
# can call from within Python to save exiting notebook...
# use capture to suppress output from stdout
%%capture
!f2py farray_sqrt.f90 -m farray -h farray_sqrt.pyf
```

[f2py] Check signature file

Attributes such as **optional**, **intent** and **depend** specify the visibility, purpose and dependencies of the arguments.

```
! -*- f90 -*- ! Note: the context of this file is case sensitive. python module farray ! in
    interface ! in :farray
        subroutine array_sqrt(n,a_in,a_out) ! in :farray:farray_sqrt.f90
            integer, optional,intent(in),check(len(a_in)>=n),depend(a_in) :: n=len(a_in)
          real*8 dimension(n),intent(in) :: a_in
          real*8 dimension(n),intent(out), depend(n) :: a_out
        end subroutine array_sqrt
    end interface
end python module farray
! This file was auto-generated with f2py (version:2).
! See http://cens.ioc.ee/projects/f2py2e/
```

[f2py] Produce extension module

Once you have verified that the signature file is correct

- Use *f2py* to compile a module file that can be imported into Python:

  f2py -c farray_sqrt.pyf farray_sqrt.f90

This produces a shared library file called : *farray.so*

In [ ]:
```
# can run command from within notebook, use 'capture' to suppress stdout
%%capture
!f2py -c farray_sqrt.pyf farray_sqrt.f90
```

[f2py] Call external function from Python

In [ ]:
```
# import the extension module
import numpy as np
from farray import array_sqrt
```

In [ ]:
```
# view docsting of function (automatically produced)
array_sqrt?
```

In [ ]:
```
# let's use the function
ain = np.array([1.0,4.0,9.0,16.0]);
aout = array_sqrt(ain)
print aout
```

[f2py] fibonacci.f90 I

Use *f2py* to create an extension module for function *fibonacci* and test it in Python.
fibonacci fills input array **a_out** with the first **n** Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13 ... Remember to check the signature file!

```
! file : fibonacci.f90
! Fortran Example :
! calculate first n Fibonacci numbers (not efficient!)
!
subroutine fibonacci(n, a_out)
      implicit none
      integer, intent(in) :: n
      real*8, dimension(n) :: a_out
      integer :: i
      do i = 1, n
        if (i.eq.1) then
          a_out(i) = 0.0
        elseif (i.eq.2) then
          a_out(i) = 1.0
        else
          a_out(i) = a_out(i-1) + a_out(i-2)
        endif
      enddo
end subroutine fibonacci
```

[f2py] fibonacci.f90 II

Let's test *fibonacci* in Python

In [ ]:
```
# create signature file
!f2py fibonacci.f90 -m ffib -h fibonacci.pyf;
```

In [ ]:
```
%%capture
# produce compiled library
!f2py -c fibonacci.pyf fibonacci.f90;
```

In [ ]:
```
# import fibonacci from ffib
from ffib import fibonacci
fibonacci?
```

In [ ]:
```
# type that Fortran expects matter (effect 'd' and 'i')
f = np.zeros(10);
fibonacci(f.size, f) # need to specify n
print f
```

[f2py] Interface with C

- f2py is the simplest way to interface C to Python
- Basic procedure is very similar to Fortran
- Differences:
  - You must write the signature file by hand
  - You must use the **intent(c)** attribute for **all variables**
  - You must define the **function name** with the **intent(c)** attribute
  - **Only 1D arrays can be handled by C**, if you pass a multidimensional array you must compute the correct index.
- Build in exactly the same way as Fortran example (but with different source code!)

[f2py] Interface with C : `carray_sqrt.f90`

```
// file carray_sqrt.f90
// C Example : calculate the sqrt of each array element
//
#include "math.h"
void array_sqrt(int n, double * a_in, double * a_out)
{
       for(int i = 0; i<n; ++i){
         a_out[i] = sqrt(a_in[i]);
       }
}
```

[f2py] Write C signature file

```
! -*- f90 -*-
! Note: the context of this file is case sensitive.

python module carray
interface
  subroutine array_sqrt(n,a_in,a_out)
    intent(c) array_sqrt ! array_sqrt is a C function
    intent(c) ! all arguments are
              ! considered as C based
    integer intent(hide), depend(a_in) :: n=len(a_in) ! n is the length
              ! of input array a_in
    double precision intent(in) :: a_in(n) ! a_in is input array
              ! (or arbitrary sequence)
    double precision intent(out), depend(a_in) :: a_out(n) ! a_out is output array,
              ! see source code
  end subroutine array_sqrt
end interface
end python module carray
```

[f2py] Test `carray.array_sqrt`

In [ ]:
```
# first remove Fortran version of array_sqrt
%reset_selective array_sqrt
```

In [ ]:
```
import numpy as np
import carray as carr
```

In [ ]:
```
# let's use the function
ain = np.array([1.0,4.0,9.0,16.0]);
aout = carr.array_sqrt(ain)
print aout
```

[f2py] Alternatives to f2py

- **Native Python interface**
  - Fully-flexible and portable
  - Complex and verbose
  - Best if you are interfacing a large amount of code and/or have a large software development project
- **Cython** : converts Python-like code into a C library which can call other C libraries
  - Standard C-like Python (or Python-like C)
- **SWIG** (or **S**implified **W**rapper and **I**nterface **G**enerator) : reads header files and generates a library Python can load
  - Very generic and feature-rich
  - Supports multiple languages other than Python (e.g. Perl, Ruby)

[f2py] Alternatives to f2py contd ...

- **ctypes**, **cffi** (C Foreign Function Interface for Python) : both provide "foreign function interfaces", or lightweight APIs, for calling C libraries from within Python
- The goal is to provide a convenient and reliable way to call compiled C code from Python using interface declarations written in C
- **Weave** : includes C/C++ code within Python code and compiles it transparently
- **Boost.python** : helps write C++ libraries that Python can load and use easily
- **PyCUDA** : allows you to include NVIDIA CUDA code within Python. You can also write C code by hand, that can be called by Python.

[f2py] Summary

- Fortran/C can give better performance than Python
- f2py is a simple way to call Fortran/C code from Python
- (much) Simpler for Fortran than for C
- Care needed when using multidimensional arrays in C
- Calling sequence is converted to something more Pythonic:
  - *array_sqrt(n, a_in, a_out)*

    becomes

    a_out = array_sqrt(a_in)