Interfaçage de Python avec Fortran

Eric Sonnendrücker

IRMA
Université Louis Pasteur, Strasbourg

projet CALVI
INRIA Lorraine

Journée Python et Calcul Scientifique,
IHP, Paris, 14 décembre 2006

Pourquoi interfacer Python avec Fortran ?

- Python a un certain nombre d'avantages :
  - Langage interprété
  - Facile à programmer
  - Dispose de modules de calcul numérique, de visualisation ...
- Il est néanmoins trop lent pour un des tâches numériques intensives.
- Solution : interfacer avec C ou Fortran
  - Allie le meilleur des deux mondes.
  - Tâches de pré- et post-traitement en Python.
  - Parties numériques intensives en Fortran (ou C)

Quels outils pour l'interfaçage Python-Fortran ?

Trois générateurs d'interface sont disponibles.
Nécessitent tous d'avoir installé le module Numeric

- Pyfort, le plus ancien, uniquement Fortran 77 permet aussi de traiter des fonctions de C structurées comme F77. Développé par Paul Dubois.
  `http://sourceforge.net/projects/pyfortran`
- f2py, Fortran 95 (et C), mais ne permet pas d'interfacer les types dérivés F90. Développé par Pearu Peterson. Bien documenté.
  `http://cens.ioc.ee/projects/f2py2e/`
- Forthon, Fortran 95, y compris interfaçage des types dérivés. Développé par Dave Grote. Pas de documentation autre que des exemples livrés avec le code. `http://hifweb.lbl.gov/Forthon/`

Utilisation de f2py

- Nécessite l'installation de Numeric (Numpy), scipy_distutils (qui gère la compilation et les compilateurs fortran)
- Trois méthodes sont proposées :
  1. Interfacer des subroutines simples sans écrire de code supplémentaire.
  2. Ajouter des directives pour f2py dans le source Fortran pour un interfaçage plus complexe.
  3. Ecrire un fichier d'interface décrivant les subroutines et les données à interfacer. f2py peut générer automatiquement un fichier d'interface simple qu'on peut ensuite éditer.

De premières interfaces

- Le code Fortran

```
 subroutine norme(a,b)
! Calcule la norme
  real(8) :: a,b
  real(8) :: c
  c= sqrt(a*a+b*b)
  print*, "la norme de (",a,",",b,") est ",c
end subroutine norme
```

- Génération de l'interface python avec f2py dans le shell

```
f2py -c norme.f90 -m vect
```

- Dans Python

```
>>> import vect
>>> vect.norme(3,4)
la norme de ( 3. , 4. ) est 5.
```

Ajouter des variables de sortie

- Le code Fortran

```
subroutine norme(a,b,c)
! Calcule la norme
  real(8),intent(in) :: a,b
  real(8),intent(out) :: c
  c= sqrt(a*a+b*b)
end subroutine norme
```

- Génération de l'interface python avec f2py dans le shell

```
f2py -c norme.f90 -m vect
```

- Dans Python

```
>>> import vect
>>> vect.norme(3,4)
5.0
>>> c=vect.norme(3,4)
>>> c
5.0
```

Documentation des fonctions interfacées

Documentation générée automatiquement par f2py

```
>>> print norme.__doc__
norme - Function signature :
c = norme(a,b)
Required arguments :
a : input float
b : input float
Return objects :
c : float
```

Ajouter des directives f2py dans le source Fortran

```
subroutine norme(a,c,n)
   integer :: n
   real(8),dimension(n),intent(in) :: a
!f2py optional, depend(a) :: n=len(a)
   real(8),intent(out) :: c
   real(8) :: sommec
   integer :: i
   sommec = 0
   do i=1,n
      sommec=sommec+a(i)*a(i)
   end do
   c=sqrt(sommec)
end subroutine norme
```

Appel

```
>>> from vect import *
>>> a=[2,3,4]
>>> norme(a)
5.385164807134505
>>> from Numeric import *
>>> a=arange(2,5)
>>> norme(a)
5.385164807134505
>>> print norme.__doc__
norme - Function signature :
c = norme(a,[n])
Required arguments :
a : input rank-1 array('d') with bounds (n)
Optional arguments :
n := len(a) input int
Return objects :
c : float
```

Les directives de compilation f2py

- Syntaxe des déclarations reprises de f90 avec quelques ajouts
- Les déclarations f90 servent donc aussi bien à f90 qu'à f2py
- Certaines directives spécifiques nécessitent d'être commentées en f90 `!f2py ...`
- Quelques exemples :
  - Déclaration de types : `real, integer, character,..`
  - Attributs f90 : `dimension(2,3), intent(in), intent(out), optional,..`
  - Attributs spécifiques f2py : `intent(hide)` : argument caché dans l'appel python, `intent(c)` : interfaçage d'une fonction C,..

Utilisation d'un fichier signature

- On peut générer automatiquement un fichier signature

  `f2py vecteur.f90 -h vecteur.pyf`

- contenu de vecteur.pyf

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

subroutine norme(a,c,n) ! in vecteur.f90
    real(kind=8) dimension(n),intent(in) :: a
    real(kind=8) intent(out) :: c
    integer optional,check(len(a)>=n),depend(a) :: n=len(a)
end subroutine norme
! This file was auto-generated with f2py (version:2.45.241
! See http://cens.ioc.ee/projects/f2py2e/
```

Appel de fonctions Python depuis Fortran

```
subroutine sommef (f,n,s)
  !Calcule somme (f(i), i=1,n)
  external f
  integer, intent(in) :: n
  real, intent(out) ::s
  s=0.0
  do i=1,n
     s=s+f(i)
  end do
end subroutine sommef
```

```
>>> from vect import *
>>>def fonction(i) : return(i*i)
>>>sommef(fonction,3)
14.0
>>>sommef(lambda x :x**3,3)
36.0
```

Interfaçage avec C

- Utilisation obligatoire d'un fichier signature (qui n'est en général pas nécessaire en Fortran où les directives f2py sont directement prises des déclarations du Fortran où rajoutées dans le code source.
- Le nom de la fonction doit être déclaré avec `intent(c)`
- `intent(c)` doit être utilisé pour toutes les déclarations de variables, mais peut être factorisé en le mettant sur une ligne seul avant les déclarations de variables.
- Commande :

```
f2py -c cfuncts.c cfuncts.pyf -m cfuncts
```

Exemple : les fonctions à interfacer

```
void depositrho(double* rho,double* fold, int* indx,double* depx,int*
 indv,double* depv, int nx, int nv){
  int i,j;
  double dx,dv;
  for (i=0; i < nx; i++)
    rho[i] = 0;
  for (i=0; i < nx; i++){
    for (j=0; j < nv; j++){
      dx = depx[i+nx*j];
      dv = depv[i+nx*j];
      rho[indx[i+nx*j]]     += (1-dx)*fold[i+nx*j];
      rho[(indx[i+nx*j]+1)\%nx]  +=    dx  *fold[i+nx*j];
}}}
 void poisson1d\_axi(double * ex, double * rho, double xmin, double dx,
 int nx){
  int i;
  double integral, xi;
   integral=0;
  ex[nx/2] = 0;
  xi=0;
  for (i=1+ nx/2; i <nx; i++){
      xi += dx;
      integral += dx*((xi-dx)*rho[i-1]+xi*rho[i])/2.;
      ex[i] = integral/xi;
      ex[nx-i-1] = -ex[i];
    }}
```

Exemple : le fichier signature

```
python module cfuncts
  interface
    subroutine depositrho(rho,fold,indx,depx,indv,depv,nx,nv)
       intent(c) :: depositrho
       intent(c)
       integer, intent(in) :: nx,nv
       real*8, dimension(nx,nv),depend(nx,nv),intent(in) :: fold,depx,
       real*8, dimension(nx),depend(nx),intent(out)      :: rho
       integer, dimension(nx,nv),depend(nx,nv),intent(in) :: indx,indv
    end subroutine depositrho

    subroutine poisson1d_axi(ex,rho,xmin, dx,nx)
       intent(c) :: poisson1d_axi
       intent(c)
       integer, intent(in) :: nx
       real*8, intent(in) :: xmin, dx
       real*8, dimension(nx),depend(nx),intent(out)      :: ex
       real*8, dimension(nx),depend(nx),intent(in)       :: rho
    end subroutine poisson1d_axi
  end interface
end python module cfuncts
```

Interfaçage d'un module

```
module vecteur
integer :: taille
real, allocatable, dimension(:) :: zero,un
 contains
subroutine norme(a,c,n)
  integer, intent(in) ::n
!f2py intent(hide) :: n=len(a)
  real(8),dimension(n),intent(in) :: a
  real(8),intent(out) :: c
  real(8) :: sommec
  integer :: i
  sommec = 0
  do i=1,n
     sommec=sommec+a(i)*a(i)
  end do
  c=sqrt(sommec)
end subroutine norme

subroutine pscal(a,b,c,n)
 ! produit scalaire de deux vecteurs
  integer, intent(in) ::n
  !f2py intent(hide) :: n=len(a)
  real(8), dimension(n),intent(in) :: a,b
  real(8), intent(out) :: c
  c=0.0
  do i=1,n
     c = c + a(i)*b(i)
  end do
end subroutine pscal
end module vecteur
```

Utilisation de Forthon

- Nécessite l'écriture d'un fichier signature .v avec un syntaxe précise que nous allons découvrir à travers quelques exemples.
- Plus orienté f90 que les deux autres outils
- Permet en particulier l'interfaçage complet des modules et des types dérivés.
- Syntaxe

```
Forthon [options] pkgname [extra Fortran or C files to be
                                compiled or objects to link]
```

- Nécessite au moins 2 fichiers : `pkgname.f90` (fortran), `pkgname.v` (signature)
- Génère `pkgnamepy.so` qui peut être importé dans Python

Exemple

- Le code Fortran

```
subroutine setsqrt(y)
    Use SimpleModule
    real(kind=8):: y
    x = sqrt(y)
    return
end
```

- Le fichier signature

```
example
****** SimpleModule:
i integer /3/ # Sample integer variable
x real # Sample real variable
****** SimpleRoutines:
setsqrt(y:real) subroutine # Sets x to the sqrt(y)
```

Quelques remarques sur le fichier signature

- Un module ne contenant que des données n'a pas besoin d'exister en Fortran. S'il est défini dans la signature, il est créé automatiquement par Forthon
- Syntaxe pour la définition des variables
  `nom` *type* [`/valeur initiale/`][#*commentaire*]

```
i integer /3/ # Sample integer variable
x real # Sample real variable
```

- Syntaxe pour la définition des subroutines
  `nom`*(arguments :type arg) subroutine* [#*commentaire*]

```
setsqrt(y:real) subroutine # Sets x to sqrt(y)
```

Syntaxe du fichier signature (1)

pkgname

```
****** Module1 test:
i integer /3/ # Sample integer variable
a real  # Sample real variable
d(3) real /3*10./ # Sample static array
n integer /0/ # Size of sample array pointer
x(0:n) _real /1/ # Sample array pointer
z(:) _real # Sample array pointer with undefined bounds.
xxx(:,:) _real # Sample multidimensional array
l1 logical /.false./ # Sample logical variable
realvar real /1./
varreal real /2./
```

Syntaxe du fichier signature (2)

```
%%%%% Type2:
 ii integer
 xx real
%%%%% Type1:
 j integer /7/ # Integer element of a derived type
 b real # Real element of a derived type
 e(10) real /8./ # Static array element of a derived type
m integer # Size of array pointer in derived type
 y(0:m) _real /3./ # Array pointer element of a derived type
 static2 Type2 # Pointer to derived type object of the same type
 next _Type1 # Pointer to derived type object of the same type
 xxx(:,:) _real
***** Module2:
 t1 Type1 # Test derived type
 t2 _Type1 # Test derived type pointer
```

Syntaxe du fichier signature (3)

```
***** Subroutines:
testsub1(ii:integer,aa:real) subroutine # call to fortran subroutine
testsub2(ii:integer,aa:real,dd:real) subroutine #setting of variable
                                                 # in fortran.
testsub3(ii:integer,aa:real,nn:integer) subroutine

testsub5() subroutine
testsub6(t:Type1) subroutine
testsub10(ii:integer,aa:real,nn:integer) subroutine
   # This subroutine is declared in a separate fortran file.
testfun(ii:integer,aa:real) real function # function example
```