Introduction

`f90wrap` est une générateur d'interface développé par James Kermode[1], qui permet de générer le module Python interfaçant Fortran 90, et spécialement pour le code du type dérivé.

Globalement, la `f90wrap` s'agit sur 3 deux étapes. Au premier étape, la générateur crée des fichiers de module `.mod` (dans le cas du type dérivé), puis, elle génère des fichiers d'interface sous le format `.py` et `.f90`, et finalement, les combiner dans le dernier étape. Une fois ces fichiers sont générés et combinés, on peut considérer le fichier `.py` vient d'être créé comme un module qui va être appeler dans un autre fichier Python pour tester.

Installation de `f90wrap`

Préparation: Git, Python 3.6+ (optionnel: Python Jupyter Notebook pour tester), `gfortran` 4.6+, `f2py` et `ifort` 12+ (included en intel parallel studio xe).

Installation de la version de développement:

```
pip install git+https://github.com/jameskermode/f90wrap
```

Pour tester et vérifier l'installation, on récupère les fichiers disponibles sur Git par la commande:

```
git clone https://github.com/jameskermode/f90wrap.git
```

et puis pour vérifier que la f90wrap a été bien installé, on lance la commande `make test` dans le répertoire `/examples`.

Utilisation de `f90wrap`

Une fois les fichiers d'interface sont crées, ils sont nommés `f90wrap_test.f90` (ou normé `f90wrap_toplevel.f90` s'il n'y pas de fonction ou subroutine définie à l'externe du module) et `module.py` où:

- `test` est le nom du fichier que l'on veut wrapper et,
- `module` est le nom à nous choisir du fichier qu'on va considérer comme un module de Python.

Cas 1.

Le cas le plus simple (il n'y pas de type dérivé, de modules, de variables et fonctions externes).

Prenons un exemple dans `/ex1`:

- 1e étape: Générer des fichiers d'interface `.f90` et `.py`:

  ```
  f90wrap -m module test.f90
  ```

- 2e étape: Combiner ces fichiers d'interface:

  ```
  f2py-f90wrap -c -m module test.f90 f90wrap_toplevel.f90
  ```

Cas 2.

Avec le type dérivé et le module.

On considère un exemple dans `/ex2`. Ici, on a besoin d'abord de générer le fichier `.mod` avant de générer les interfaces.

- 1e étape: Générer le fichier `.mod` :

```
gfortran -x f95-cpp-input -fPIC -c example.f90
```

- 2e étape: Générer des fichiers d'interface `.f90` et `.py`:

```
f90wrap -m example ./example.f90
```

- 3e étape: Combiner ces fichiers d'interface:

```
f2py-f90wrap -c -m _example f90wrap_example.f90 example.o
```

Cas 3.

Avec le type dérivé, le module et des fonctions définies à l'externe.

Dans ce cas, on considère deux fichiers à wrapper `cyldnad.f90` et `DNAD.f90` dans lesquels un fichier a des fonctions et variables sont définies par l'autre.

On considère l'exemple 3 dans `/ex3`:

- 1e étape: Générer des fichiers `.mod`:

```
gfortran -fPIC -x f95-cpp-input -c DNAD.f90 -o DNAD.o
gfortran -fPIC -x f95-cpp-input -c cyldnad.f90 -o cyldnad.o
```

(notons que l'on lance d'abord le fichier `DNAD.f90` car il y des fonctions et variables dans le fichier `cyldnad.f90` sont définies par le type vue dans le fichier `DNAD.f90`).

- 2e étape: Générer des fichiers d'interface `.f90` et `.py`:

```
f90wrap -m module DNAD.f90 cyldnad.f90
```

- 3e étape: Combiner ces fichiers d'interface:

```
f2py-f90wrap -c -m _module DNAD.o cyldnad.o f90wrap*.f90
```

References

[1] James R Kermode. f90wrap: an automated tool for constructing deep python interfaces to modern fortran codes. *J. Phys. Condens. Matter*, March 2020.