Fortran to Cython test 1

Source: https://stackoverflow.com/questions/22404060/fortran-cython-workflow

Directory: home/github/NOAA-OWP/t-route/src/python_routing_v02/fast_reach/fortran2cython_test1

1. Fortran source code: gfunc.f90 with multiple subroutines or functions(not here though)

Open

gfunc.f90

~/github/NOAA-OWP/t-route/src/python_routing_v02/fast_reach/fortran2cython_test1

```
module gfunc_module
implicit none
    double precision :: mt
contains
subroutine gfunc(x, n, m, a, b, c)
    double precision, intent(in) :: x
    integer, intent(in) :: n, m
    double precision, dimension(n), intent(in) :: a
    double precision, dimension(m), intent(in) :: b
    double precision, dimension(n, m), intent(out) :: c
    integer :: i, j

    do j=1,m
        do i=1,n
            c(i,j) = exp(-x * (a(i)**2 + b(j)**2))
        end do
    end do
    mt=2.0
    call gfunc2(n,m,c)

end subroutine

subroutine gfunc2(n, m, c)
    integer, intent(in) :: n, m
    double precision, dimension(n,m), intent(inout) :: c
    integer :: i,j

    do j=1,m
        do i=1,n
            c(i,j)= mt*c(i,j)
        enddo
    enddo
end subroutine gfunc2
end module
```

2. Fortran wrapper: pygfunc.f90 (wraps only the very first subroutine to be called from cython)

Open

pygfunc.f90

~/github/NOAA-OWP/t-route/src/python_routing_v02/fast_reach/fortran2cython_test1

gfunc.f90

```
module gfunc1_interface
use iso_c_binding, only: c_double, c_int
use gfunc_module, only: gfunc
implicit none
contains
subroutine c_gfunc(x, n, m, a, b, c) bind(c)
    real(c_double), intent(in) :: x
    integer(c_int), intent(in) ::  n, m
    real(c_double), dimension(n), intent(in) :: a
    real(c_double), dimension(m), intent(in) :: b
    real(c_double), dimension(n, m), intent(out) :: c
    call gfunc(x, n, m, a, b, c)
end subroutine
end module
```

3. Head file

pygfunc.h - /home/dongha.kim/github/NOAA-OWP/t-route/src/python_routing_v02/fast_reach/...

File Edit Search Preferences Shell Macro Windows

```
extern void c_gfunc(double* x, int* n, int* m, double* a, double* b, double* c);
```

4. Cython file with nogil option for multi-thread parallel computing: pygfunc.pyx

Open

pygfunc.pyx

~/github/NOAA-OWP/t-route/src/python_routing_v02/fast_reach/fortran2cython_test1

```
from numpy import linspace, empty
from numpy cimport ndarray as ar
cimport numpy as np

cdef extern from "pygfunc.h" nogil:
    void c_gfunc(double* x, int* n, int* m, double* a, double* b, double* c)

def f2cytest(double x, double a, double b, int n):
    cdef:
        ar[double] ax = linspace(a, b, n)
        ar[double,ndim=2] c = empty((n, n), order='F')
    with nogil:
        c_gfunc(&x, &n, &n, <double*> ax.data, <double*> ax.data, <double*> c.data)
    return c
```

5. Setup file: setup.py that is executed by command-line:

```
(base) [dongha.kim@nwcal-apd-dev1 fortran2cython_test1]$ python3 setup.py build_ext --inplace
```

Open

setup.py

~/github/NOAA-OWP/t-route/src/python_routing_v02/fast_reach/fortran2cython_test1

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext
# This line only needed if building with NumPy in Cython file.
from numpy import get_include
from os import system


# compile the fortran modules without linking
fortran_mod_comp =  'gfortran gfunc.f90 -c -o gfunc.o -O3 -fPIC'
system(fortran_mod_comp)

shared_obj_comp = 'gfortran pygfunc.f90 -c -o pygfunc.o -O3 -fPIC'
#print(f"shared_obj_comp: {shared_obj_comp}")
system(shared_obj_comp)

ext_modules = [Extension(# module name:
                         "pygfunc",
                         # source file:
                         ["pygfunc.pyx"],
                         # other compile args for gcc
                         extra_compile_args=['-fPIC', '-O3'],
                         #extra_compile_args=["-g"],
                         # other files to link to
                         extra_link_args=["gfunc.o", "pygfunc.o"])]

setup(name = 'pygfunc',
      cmdclass = {'build_ext': build_ext},
      # Needed if building with NumPy.
      # This includes the NumPy headers when compiling.
      include_dirs = [get_include()],
      ext_modules = ext_modules)
```

6. Python file to execute the cythonized fortran procedures: pygfunc.ipynb

In [1]:
```
from pygfunc import f2cytest
print(f2cytest(1., a=-1.,b=1., n=4))
```

```
[[0.27067057 0.65838598 0.65838598 0.27067057]
 [0.65838598 1.60147481 1.60147481 0.65838598]
 [0.65838598 1.60147481 1.60147481 0.65838598]
 [0.27067057 0.65838598 0.65838598 0.27067057]]
```

In [2]:
```
import numpy as np
a = np.linspace(-1, 1, 4)**2
A, B = np.meshgrid(a, a, copy=False)
print(np.exp(-(A + B)))
```

```
[[0.13533528 0.32919299 0.32919299 0.13533528]
 [0.32919299 0.8007374  0.8007374  0.32919299]
 [0.32919299 0.8007374  0.8007374  0.32919299]
 [0.13533528 0.32919299 0.32919299 0.13533528]]
```

In [ ]:
```
print(2.0*np.exp(-(A + B)))
```

In [ ]:
```
print(f2cytest(1., a=-1.,b=1., n=4))
```

In [3]:
```
import time

start = time.time()
for i in range(8000):
    f2cytest(1., a=-1.,b=1., n=4)

end = time.time()
print(end - start)
```

```
0.22953462600708008
```

In [7]:
```
import time
from joblib import delayed, Parallel
cpu_pool=5
start = time.time()
with Parallel(n_jobs=cpu_pool, prefer='threads') as parallel:
    jobs=[]
    for i in range(8000):
        jobs.append(delayed(f2cytest)(1., a=-1.,b=1., n=4))

    #results= parallel(jobs)
end = time.time()
print(end - start)
```

```
0.0365908145904541
```