A walk-through the different ways to link Python to C

July 31, 2017

We want to code a function that computes

$$f(x) = \sum_{k=0}^{N} c_k \cos(kx)$$

where

$$c_k = \frac{1}{1+k^2}$$

and x is an array.

We time things using the %timeit command in ipython, and use $N = 50$ and $M = 100$.

1 Vanilla Python

1 $\langle * \ 1\rangle\equiv$ 2a ▷

```
from pylab import *
N=50
M=100
c=1.0/(1.0+arange(N+1)**2)
x=linspace(0,10,M)
def fourier(N,c,x):
        z=zeros(x.shape)  # create and initialize z
        for i in range(M):
                zz=0.0;xx=x[i]
                for k in range(N+1):
                        zz += c[k]*cos(k*xx)
                z[i]=zz
        return(z)
```

%timeit -c fourier(N,c,x)
The output:

100 loops, best of 3: 4.2 ms per loop

2 Using Numpy to speed up Python

2a ⟨* 1⟩+≡ ◁1 2b▷

```
from pylab import *
N=50
M=100
c=1.0/(1.0+arange(N+1)**2)
x=linspace(0,10,M)
def fourierv(N,c,x):
        z=zeros(x.shape)  # create and initialize z
        for k in range(N+1):
                z += c[k]*cos(k*x)
        return(z)
```

%timeit -c fourierv(N,c,x)
The output:

1000 loops, best of 3: 280 µs per loop

3 Using Weave

2b ⟨* 1⟩+≡ ◁2a 3▷

```
from pylab import *
from scipy.weave import inline
N=50
M=100
c=1.0/(1.0+arange(N+1)**2)
x=linspace(0,10,M)
def fourc(N,c,x):
        n=len(x)
        z=zeros(x.shape)
        # now define the C code in a string.
        code="""
        double xx,zz;
        for( int j=0 ; j<n ; j++ ){
        xx=x[j];zz=0;
        for( int k=0 ; k<=N ; k++ )
            zz += c[k]*cos(k*xx);
        z[j]=zz;
    }
        """
        inline(code,["z","c","x","N","n"],compiler="gcc")
        return(z)
```

%timeit -c fourc(N,c,x)
The output:

1000 loops, best of 3: 166 µs per loop

4 Cython

We start with the dumb python script and verify its speed through cython.

3 ⟨* 1⟩+≡ ◁2b 4▷

```
str="""
from pylab import *
N=50
M=100
c=1.0/(1.0+arange(N+1)**2)
x=linspace(0,10,M)
def fourier(N,c,x):
        z=zeros(x.shape)  # create and initialize z
        for i in range(M):
                zz=0.0;xx=x[i]
                for k in range(N+1):
                        zz += c[k]*cos(k*xx)
                z[i]=zz
        return(z)
"""
with open("fouriercy0.pyx",mode="w") as f:
        f.write(str)
import pyximport
pyximport.install()
import fouriercy0
```

%timeit -c fouriercy0.fourier(N,c,x)
The output:

100 loops, best of 3: 3.92 ms per loop

Nothing gained so far. Now let us add local variables and see how timing changes.

4 ⟨* 1⟩+≡ ◁3 5▷

```
str="""
import numpy as np
N=50
M=100
c=1.0/(1.0+np.arange(N+1)**2)
x=np.linspace(0,10,M)
cpdef fourier(N,c,x):
        cdef np.ndarray z=np.zeros(x.shape)  # create and initialize z
        cdef int i,k
        cdef double zz,xx
        for i in range(M):
                zz=0.0;xx=x[i]
                for k in range(N+1):
                        zz += c[k]*np.cos(k*xx)
                z[i]=zz
        return(z)
"""
with open("fouriercy1.pyx",mode="w") as f:
        f.write(str)
import pyximport
pyximport.install()
import fouriercy1
```

The program refuses to compile. Cython says:

cdef np.ndarray z=np.zeros(x.shape) # create and initialize z ˆ —
————————————————————

fouriercy1.pyx:8:6: 'np' is not a cimported module

Fair enough. We did not teach cython how to use numpy. So we need an extra cimport line. Add that and recompile.

5 ⟨* 1⟩+≡ ◁4 6▷

```
str="""
import numpy as np
cimport numpy as np
N=50
M=100
c=1.0/(1.0+np.arange(N+1)**2)
x=np.linspace(0,10,M)
cpdef fourier(N,c,x):
        cdef np.ndarray z=np.zeros(x.shape)  # create and initialize z
        cdef int i,k
        cdef double zz,xx
        for i in range(M):
                zz=0.0;xx=x[i]
                for k in range(N+1):
                        zz += c[k]*np.cos(k*xx)
                z[i]=zz
        return(z)
"""
with open("fouriercy1a.pyx",mode="w") as f:
        f.write(str)
import pyximport
pyximport.install()
import fouriercy1a
```

%timeit -c fouriercy1a.fourier(N,c,x)
The output:

100 loops, best of 3: 3.44 ms per loop

We expected more than this!

We know that cos is a problem. It is a python function right in the middle of the doubly nested for loop. Let us solve that problem.

6 $\langle *\ 1\rangle{+}{\equiv}$ ◁5 7▷

```
str="""
import numpy as np
cimport numpy as np
cdef extern from "<math.h>":
        cdef double cos(double x)
N=50
M=100
c=1.0/(1.0+np.arange(N+1)**2)
x=np.linspace(0,10,M)
cpdef fourier(N,c,x):
        cdef np.ndarray z=np.zeros(x.shape)  # create and initialize z
        cdef int i,k
        cdef double zz,xx
        for i in range(M):
                zz=0.0;xx=x[i]
                for k in range(N+1):
                        zz += c[k]*cos(k*xx)
                z[i]=zz
        return(z)
"""
with open("fouriercy2.pyx",mode="w") as f:
        f.write(str)
import pyximport
pyximport.install()
import fouriercy2
```

%timeit -c fouriercy2.fourier(N,c,x)
The output:

100 loops, best of 3: 860 μs per loop

Better, but we have a ways to go. We are not even as fast as numpy!

We see that x[i], c[i] and z[i] are invoked every iteration. These are python objects. We need to make them c array elements.

7 $\langle * \ 1\rangle{+}{\equiv}$ ◁6 8▷

```
str="""
import numpy as np
cimport numpy as np
cdef extern from "<math.h>":
        cdef double cos(double x)
DTYPE = np.double
ctypedef np.double_t DTYPE_t
N=50
M=100
c=1.0/(1.0+np.arange(N+1)**2)
x=np.linspace(0,10,M)
cpdef fourier(int N,np.ndarray[DTYPE_t,ndim=1] c,np.ndarray[DTYPE_t,ndim=1] x):
        cdef np.ndarray[DTYPE_t,ndim=1] z=np.zeros(M,dtype=DTYPE)
        cdef int i,k
        cdef double zz,xx
        for i in range(M):
                zz=0.0;xx=x[i]
                for k in range(N+1):
                        zz += c[k]*cos(k*xx)
                z[i]=zz
        return(z)
"""
with open("fouriercy3.pyx",mode="w") as f:
        f.write(str)
import pyximport
pyximport.install()
import fouriercy3
```

%timeit -c fouriercy3.fourier(N,c,x)
The output:

10000 loops, best of 3: 169 μs per loop

viola! We have done it!

It takes some effort, but cython can match weave for speed.

5 More Numpy

We can speed up Numpy some more by getting rid of the final for loop as well. We create a matrix

$$A_{ik} = \cos(kx_i)$$

and define the output as a matrix multiplication

$$z_i = A_{ik}c_k$$

8 $\langle * \ 1 \rangle + \equiv$ $\triangleleft 7$

```
from pylab import *
N=50
M=100
c=1.0/(1.0+arange(N+1)**2)
x=linspace(0,10,M)
def fouriervm(N,c,x):
        A=cos(outer(x,arange(N+1)))
        return (dot(A,c))
```

%timeit -c fouriervm(N,c,x)
The output:

1000 loops, best of 3: 175 µs per loop

This is extraordinary. By creating a huge matrix of cosines, we get essentially the same speed as C. Note that we use much more memory. But still, it is extremely surprising.

Notes:

1. The "outer" command takes the two arguments and treats them as vectors, as x_i and k_j. It then constructs a rank 1 matrix out of them as mentioned above.

2. The "dot" command computes the matrix product of A and c.