Why call C/C++/Fortran functions from R/Python/Matlab?

- Performance: loops are slow in R/Python/Matlab. Best to avoid them by writing vector expressions. Not always easy.
- You already have a function written in C/C++/Fortran and you don’t want to rewrite it (and it would be slow)
- You have a C/C++/Fortran function and you can’t be bothered to write tedious I/O code
- You just enjoy confusing people
- It is possible to interact with C++ at the OOP level at least in R/Python. I never did this
- It is not too hard to call Python from C, which can make for a nice user interface component. I have done this but will not cover it now (ask me if you care)

The eye of the needle: C void functions.

```
void fc(double *a, int *n)
{
 if (!a) return;
 if (!b) return;

 /** do stuff to *a and *n */
 return;
}
```

```
subroutine ff(a, n)
 integer n
 real(kind=8), dimension(1:n)::  a
 !do stuff to a and n
 return
end subroutine ff
```

- C void functions with pointer args and Fortran subroutines work in the same way once they are compiled
- A ‘start of code’ address together with a list of ‘start of data’ addresses
- C++ can look like C to the outside world through its extern “C” { } mechanisms.
- We need to make sure that R/Python/Matlab are passing pointers to the correct sized data
- Get this wrong: segmentation fault.

Convolution example : R to C

```r
dyn.load("convolve.so")

convolvec <- function(f,g,n)
{
    n <- length(f)
    r <-.C("convolve",
      fog=numeric(n),
      f=as.numeric(f),
      g=as.numeric(g),
      n=as.integer(n))
   r$fog/n
}

n <- 1024
x <-seq(-1,1,length=n)
f <- (sin(4*pi*x))^2 + rnorm(n,
sd=0.15)
g <- (sin(pi*x))^2

system.time(fog <- convolver(f,g))
#2.710 seconds
system.time(fog2 <- convolvec(f,g))
#0.006 seconds
```

compile c code with:

```
 > gcc -c -fPIC convolve.c #any other flags
 > gcc -shared -o convolve.so convolve.o
```

R provides dyn.load and .C functions

```c
void convolve(double *fog, double *f, double *g, int
*n)
{
  int i,j,m;
  m = *n;
  for (j = 0; j < m; j++){
      fog[j] = 0.0;
      for (i = 0; i < m; i++){
        fog[j] = fog[j] +   f[i]*g[(i-j+m)%m];
     }
  }
  return;
}
```

Convolution example : R to Fortran

```
dyn.load("convolvef.so")

convolvef <- function(f,g,n)
{
    n <- length(f)
    r <-.Fortran("convolve",
      fog=numeric(n),
      f=as.numeric(f),
      g=as.numeric(g),
      n=as.integer(n))
   r$fog/n
}

n <- 1024
x <-seq(-1,1,length=n)
f <- (sin(4*pi*x))^2 + rnorm(n,
sd=0.15)
g <- (sin(pi*x))^2

system.time(fog <- convolver(f,g))
#2.710 seconds
system.time(fog2 <- convolvef(f,g))
#0.006 seconds
```

compile c code with:

```
> gfortran -c -fPIC convolvef.f90 #any other flags
> gcc -shared -o convolvef.so convolvef.o
```

R provides dyn.load and .Fortran functions. Fortran is just like .C but adds a _ to the function name...

```
subroutine convolve(fog,f,g,n)
  integer n
  real(kind=8), dimension(1:n) :: fog,f,g
  integer i,j
  do j = 1,n
     fog(j) = 0.0
     do i = 1,n
     fog(j) = fog(j) + f(i) * g( 1+mod( i-j+n, n))
     end do
  end do
end subroutine convolve
```

Convolution example : numpy to C/Fortran

```
from ctypes import *;
import numpy as np
import matplotlib.pylplot as plt
convolveso = CDLL("convolve.so") # make sure convolve.so is in LD_LIBRARY_PATH

def convolvec(f,g):
      n = len(f)
      fog = np.zeros(n)
      convolveso.convolve(fog.ctypes.data_as(POINTER(c_double)),
                        f.ctypes.data_as(POINTER(c_double)),
                        g.ctypes.data_as(POINTER(c_double)),
                        pointer(c_int(n)))
      return fog/np.double(n)
n = 1024
x = np.arange(-1,1,2.0/np.double(n))
f = ( np.sin(4.0*np.pi * x))**2 + np.random.normal(0.0,0.15,n)
g = ( np.sin(np.pi * x))**2

fog = convolvec(f,g)

plt.plot(x,f)
plt.plot(x,g)
plt.plot(x,fog)
```

MATLAB

- Matlab is more of a pain
- It has a thing called mex, that many love but I think is just filler. It doesn’t work ‘out-of-the-box’ because Matlab no longer works with the compiler provided with it (lcc)
- It carts around about 50% of an incompatible OS to make linking hard
- It wants you to provide a header for your functions (it does not trust the C interface)
- Can be done though… rough sketch on right

```
 if (~libisloaded('convolve'))
      loadlibrary('convolve.so','convolve.H');
  end

fogp = libpointer(‘doublePtr’,fog);
#etc

calllib(‘convolve’,’convolve’,fogp,fp,gp,np)

fog = fogp.value
```