**Interfacing C and C++ code with scripting languages Exercise**

*Introduction – skim me for quick start*

This is a simple demo illustrating something you might find very useful in your work. That is, the world is full of useful, bulletproof algorithms written in C or C++. I know of several important reasons why you might want to make use of these algorithms in another language:

1. You know scripting language x (e.g. python on perl) but C or C++ little to nothing about C or C++.
2. Good software is expensive (time=money) to develop so why reinvent the wheel. Wouldn’t it be nice if you could use algorithm y in your scripting language?
3. Scripting languages are often a much faster path to a solution for many problems , BUT they come at a cost. Scripting languages are “interpreted” on the fly not “compiled”. As a result they can perform very badly on compute intensive operations. (Try something like counting from 1 to a billion in C or FORTRAN and comparing that to the a tcl script (particularly bad in this case) to do the same thing. An established solution to this dilemma is to do the numerics in a compiled language and provide a mechanism for the C/C++ code to be accessed by the interpreter. (Note to matlab user. Good matlab code takes advantage of this same idea by a feature of the scripting language experienced people use all the time – the “:” notation. When you do matrix operations with the : operator you “vectorize” your matlab code meaning there are hidden loops done by machine code instead of an interpreted loop. Matlab has its own mechanism to integrate C through the “mex” compilation mechanism.)

The example we will be working with here uses an interface builder called the Simplified Wrapper and Interface Generator (SWIG). The strength of SWIG is an ability to take a set of C procedures or C++ objects, build “wrapper” code to define a generic “interface”, and provide mechanism to expose the compiled code to the following languages: perl, python, tcl/tk, java, ruby, list, chicken, guile, octave (public domain version of matlab), R, Modula-3, php, pike, and probably others. I have seen swig evolve dramatically over the past several years to the point that it is now considered a standard package installed with MacOS and probably most linux distributions. It can even be use on (gulp)Winders if you want to start down that slippery slope.

*Come here to do the actual exercise*

The example were are going to use is contrived but better than a “hello world”style tutorial. Seismic Unix has a really great core library of low level functions in the “cwp” (Center for Wave Propagation) library. The idea here is to suppose I wanted to use some of these algorithms in a scripting language. I happened to choose python because I’m intrigued by the language because it has strong potential for me as a C++ programmer, but it would have been just as easy to do this in say perl. With that let’s proceed:

1. Copy the files from the location I’ll give you.
2. Look at the file cwp.i. This is the “interface” definition file (the “.i” tag). My comments show the basic structure here. For a lot more info see the swig reference manual I will post with this exercise.
3. First build the wrapper routines with swig using this command:

swig -python cwp.i

1. Notice that this step produced two files: cwp\_wrap.c and cwp.py. cwp\_wrap.c is the wrapper C code and cwp.py is the python interface code the python interpreter will need to use this package.
2. Next is the nasty step you could spend a long time on if you were not familiar with this process. That is we have to compile the C code and build a “shared object” (also sometimes called a dynamic library or a dynamic link library (DLL in the wonderful world of Microsoft)). This takes two steps. Copy and paste these lines:

gcc -c -fpic butterworth.c sinc.c mksinc.c stoep.c cwp\_wrap.c \

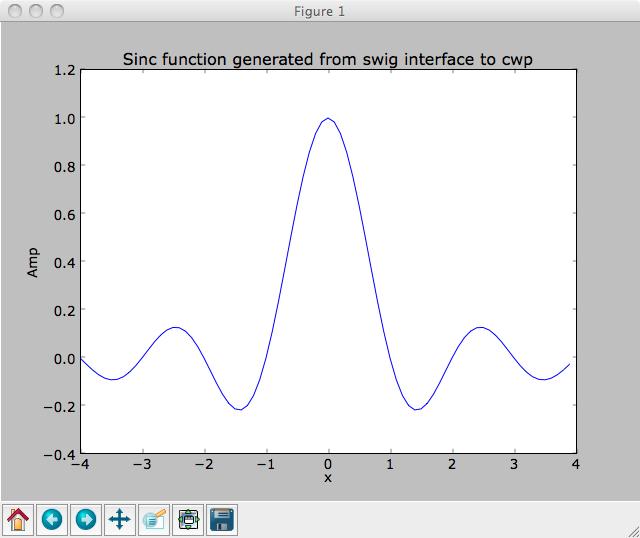
-I/usr/include/python2.6

gcc -shared butterworth.o sinc.o mksinc.o stoep.o cwp\_wrap.o \

-lpython -o \_cwp.so

Note that if you were doing this for the long term you would want to use some build system that wouldn’t require you to remember such a set of obscure arguments. The standard for this in the unix world is the makefile and the “make” utility.

1. Look at the python script in this directory called plotsinc.py. Run it by starting python and pasting the script into the terminal window. You should get a plot like this:



For those familiar with matlab you will note this plot method is as simple as the standard matlab plotting functions.

The primary thing to notice in this script is the call to cwp.fsinc(t[i]) which uses the Seismic Unix C code we compiled above.