How to Bessel Functions in C

From Difdopwiki

The problem

- There is a basic implementation of the bessel functions in C/C++ in the gsl library (http://www.gnu.org/software/gsl/) . However this implementation does not work for complex arguments, and therefore is not very useful for a general complex problem.
- Slatec (http://www.netlib.org/slatec/) , which is a fortran library has this implemented for complex arguments for double precision, which is the case we are interested in.
- However Slatec is a huge, ancient beast, which uses some weird methods to figure out machine specifications. Or maybe it is just that it is written in fortran.

The Solution

Here I will describe the steps I followed to rip the function zbesj.f (http://www.netlib.org/slatec/src/zbesy.f) from the slatec library and get it working as a standalone fortran library which contains just this file and its dependencies. Here we go.

Step1: The Netlib website has the nice feature that it allows you to download a file and its dependencies. Go to this link (http://www.netlib.org/slatec/src/) and click next to zbesj.f which says "zbesj.f plus dependencies". It will ask you if you want a zip, tgz or plain text file. Choose tgz (don't choose plain text, we will replace some of the files later on). **Step2:** Here is how I extract and organize the fortran files. You decide your own organization:

```
mkdir amos
cp ~/down/netlibfiles.tgz ./amos/
cd amos
tar xvf netlibfiles.tgz
mv slatec/src *.f ./
rm -rf slatec
```

Step3: The machine files i1mach.f and d1mach.f in slatec suck worse than slatec itself (see this article (http://www.nsc.liu.se/~boein/ifip/kyoto/workshop-info/proceedings/einarsson/d1mach.html) for instance for a discussion). Apparently the ones in blas are newer. So download these files from netlib and overwrite the existing ones:

```
wget -N http://www.netlib.org/blas/d1mach.f
wget -N http://www.netlib.org/blas/i1mach.f
```

Step4: In my case I was interested in making a static lib and a wrapper header which I would eventually

include in my c project. So I actually used a automake file as part of the autoconf/automake scripts in the project. You can find a single Makefile I produced for testing here (http://difdop.polytechnique.fr/software/Makefile) (Obviously this is not very portable in comparison to autoconf/automake). This will create a "libamos.a" when you type "make".

Step5: We need to first test that the thing actually works in fortran, before attempting to call it from c. So here is my attempt at a fortran program that prints out the value of J0(0.1+i*0.2):

```
      program bestest
      integer nz,ierr
      double precision zr,zi,cyr,cyi
      external zbesj
      dimension cyr(4),cyi(4)
      zr=0.1d0
      zi=0.2d0
      call zbesj(zr,zi,0.0d0,1,1,cyr,cyi,nz,ierr)
      print *, cyr(1),cyi(1)
      end program
```

save this as bestest.f in that directory, so that you can compile it using

```
make test
```

with the above makefile. When I run it, I get this:

```
./bestest
1.0074890116830091      -1.0037519074414627E-002
```

to compare with the octave result:

```
octave:1> output_precision(17)
octave:2> besselj(0,0.1+i*0.2)
ans = 1.0074890116830091e+00 - 1.0037519074414627e-02i
```

Step6: Ok, so it works. We can add the function declaration to our header file, and link it with -lamos as well as -lgfortran. This obviuosly depend on your project organization. Here is a minimal c file for testing purposes:

```
#include <stdio.h>
#include <complex.h>

extern void zbesj_(double*, double*, double*, int*, int*, double*, double*, int*, int*);

complex besselj(double nu, complex z){
  int kode=1;
  int n=1;
  double zr=creal(z);
  double zi=cimag(z);
  int nz,ierr;
  double cyr[1],cyi[1];
  complex res;
  zbesj_(&zr,&zi,&nu,&kode,&n,cyr,cyi,&nz,&ierr);
  if(ierr!=0){
```

```
    printf("error!\n");
  }
  res=cyr[0]+I*cyi[0];
  return res;
}

int main(void){
  complex J0=besselj(0.0,0.1+0.2*I);
  printf("\nJ0(0.1+0.2i)= %.17f  %+.17f I\n",creal(J0),cimag(J0));
}
```

you can save this as "cbestest.c". This way, you can compile it using the above Makefile by typing "make ctest". Note that the above besselj function is a reasonable first approximation to a wrapper function. Here is the result:

```
./cbestest

J0(0.1+0.2i)= 1.00748901168300908  -0.01003751907441463 I
```

note also that the little differences in the last digits are in fact due to output roundup.

if you want the result of all this as a tar.gz archive, here it is (http://difdop.polytechnique.fr/software/lib_amos.tar.gz) .

Corollary: If you want to use other similar functions (e.g. the modified bessel function), all you need to download is the netlib file (http://www.netlib.org/slatec/src/) corresponding to that function (e.g. zbesi.f), and add it similarly to the make files etc. (see above) most special functions are dependent on the same basic functions.