Mixed Programming R with C/C++/Fortran

Shiyuan

Mixed programming to Speed Up Computation

- R/C interface
  - .C
  - .Call
- R/Fortran interface
  - .Fortran
- R/C++ interface
  - Use Rcpp package

Use the .C interface

The C source file, **test.c**

- The return type must be “void”
- All function arguments are pointers

```
void timesTwo(int *n1, double *x){
   //multiply every elements in x by 2
   // the length of x is n
   int i, n;
   n = n1[0];
   for(i=0;i<n;i++){
       x[i] = x[i]*2;
   }
}
```

Compile the C file in the command line

> R CMD SHLIB test.c

```
dhcp-10-202-128-141:testC shiyuanhe$ ls
test.R  test.c
dhcp-10-202-128-141:testC shiyuanhe$ R CMD SHLIB test.c
/opt/local/bin/gcc -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG
-I/usr/local/include -I/usr/local/include/freetype2 -I/opt/X11/include    -fPIC
  -Wall -mtune=core2 -g -O2  -c test.c -o test.o
/opt/local/bin/gcc -dynamiclib -Wl,-headerpad_max_install_names -undefined dynam
ic_lookup -single_module -multiply_defined suppress -L/Library/Frameworks/R.fram
ework/Resources/lib -L/usr/local/lib -o test.so test.o -F/Library/Frameworks/R.f
ramework/.. -framework R -Wl,-framework -Wl,CoreFoundation
dhcp-10-202-128-141:testC shiyuanhe$ ls
test.R  test.c  test.o  test.so
```

In R, load the dynamic library by **dyn.load()** and call the C function by **.C()**

```
> n = 3
> x = c(2.5, 9.6, -29)
>
> dyn.load("test.so")
> res = .C("timesTwo",n = as.integer(n),
+     x = as.double(x))
> res
$n
[1] 3

$x
[1]   5.0  19.2 -58.0
```

Use the .Fortran interface

Use the example of G. Kovacs, S. Zucker and T. Mazeh (2002), a box fitting algorithm in search for periodic transits

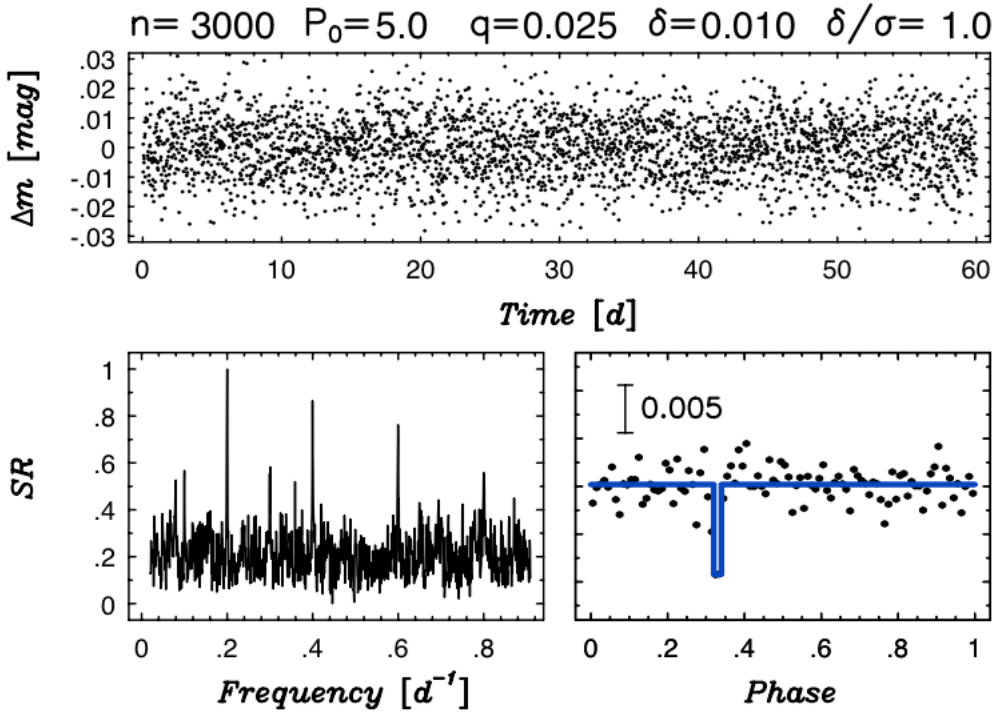

Fig. 1. An example of the power of the BLS method. The time series, the normalized BLS frequency spectrum and the folded time series are shown in the upper and lower two panels, respectively. The signal parameters are displayed in the header (see text for details).

The Fortran code on Github

```
      subroutine eebls(n,t,x,u,v,nf,fmin,df,nb,qmi,qma,
     &p,bper,bpow,depth,qtran,in1,in2)
c
c     Input parameters:
c     ~~~~~~~~~~~~~~~~~
c
c     n    = number of data points
c     t    = array {t(i)}, containing the time values of the time series
c     x    = array {x(i)}, containing the data values of the time series
c     u    = temporal/work/dummy array, must be dimensioned in the
c            calling program in the same way as  {t(i)}
c     v    = the same as  {u(i)}
c     nf   = number of frequency points in which the spectrum is computed
c     fmin = minimum frequency (MUST be > 0)
c     df   = frequency step
c     nb   = number of bins in the folded time series at any test period
c     qmi  = minimum fractional transit length to be tested
c     qma  = maximum fractional transit length to be tested
c
c     Output parameters:
c     ~~~~~~~~~~~~~~~~~~
c
c     p    = array {p(i)}, containing the values of the BLS spectrum
c            at the i-th frequency value -- the frequency values are
c            computed as  f = fmin + (i-1)*df
c     bper = period at the highest peak in the frequency spectrum
c     bpow = value of {p(i)} at the highest peak
c     depth= depth of the transit at   *bper*
c     qtran= fractional transit length  [ T_transit/bper ]
c     in1  = bin index at the start of the transit [ 0 < in1 < nb+1 ]
c     in2  = bin index at the end   of the transit [ 0 < in2 < nb+1 ]
```

Download the Fortran code as file code.f

Compile in the command line
> R CMD SHLIB code.f

In R, read data and define all related variables, then load the dynamical library and call Fortran function by .Fortran

```
27 dyn.load("code.so")
28 res = .Fortran("eebls",n = as.integer(n),
29     t = as.double(t), x = as.double(x),
30     u = as.double(u), v = as.double(v),
31     nf = as.integer(nf), fmin = as.double(fmin),df = as.double(df),
32     nb = as.integer(nb), qmi = as.double(qmi), qma = as.double(qma),
33     p = as.double(p), bper = as.double(bper), bpow = as.double(bper),
34     depth = as.double(depth), qtran = as.double(qtran),
35     in1 = as.integer(in1), in2 = as.integer(in2))
```

Use Rcpp package

Sample C++ code, test.cpp

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
  int n = x.size();
  x = x * 2;
   return x;
}
```

- NumericVector behaves like the C++ standard container
  - x.size()
  - x.begin(), x.end()
  - x[i], x(i)
- Directly compatible data type with R: NumericVector, NumericMatrix, List, Function….

Directly compile the code and use it in R

```
> Rcpp::sourceCpp("test2.cpp")
> x = 1:10
> timesTwo(x)
 [1]  2  4  6  8 10 12 14 16 18 20
```