R から C 言語を使い、R 内部の関数のみで作るよりも高速なプログラムを作成する方法を解説しています。
RjpWiki: R から他言語利用 の応用編です。

最終更新時間：2007 年 05 月 31 日 15 時 06 分 07 秒

なぜ R から C をつかうのか

R の汎用性と C のスピードを同時に実現するためです。数値計算だけなら、MATLAB でやるのが文法も分かりやすく手軽ですが、スピードと汎用性に欠けます。R だけでやると、スピードは MATLAB よりもだいぶ劣ります。Ox というのもあり、スピードが速いそうですが、汎用性に欠けます。特に、無償版はインタープリタ型ではないのと、グラフィックスと結果のデータの扱いに難があります。

別に、Fortran でもよいのですが、少々古く、最近はもっぱら数値計算用のみにつかわれているので、一般性を考えれば、C のほうが無難だと思います。

BLAS と LAPACK について

C でベクトル・行列演算を行う場合、BLAS/LAPACK という、Fortran で書かれたライブラリを使います。BLAS/LAPACK は R に実装されているので、R から C 言語を使う場合、新たに BLAS/LAPACK をインストールしなくてもつかうことができます。

BLAS とは、Fortran で書かれたベクトル・行列演算ライブラリです。LAPACK も Fortran で書かれたライブラリで、最小二乗法や固有値問題ルーチンなどが実装されています。

```
C:\Program Files\R\rw2010\include\R_ext\BLAS.h
```

や

```
C:\Program Files\R\rw2010\include\R_ext\LAPACK.h
```

あたりを参考にしてください。BLAS/LAPACK 関連のものは、すべて Netlib にあります。

（もしかしたら、R 自身が BLAS/LAPACK を内装しているのだから、R の関数を C の内部に組み込んでコンパイルするのとスピードはほとんど変わらない可能性はあります。ただ、実際のところは比較して見ないと分からないのと、BLAS/LAPACK を直接つかったほうが少なくとも遅くはならないというのは確かです。）

最初にすべきこと

Windows の場合は、以下の工程を踏む必要があります。

・RjpWiki: R から他言語利用 : Windows の場合

次に、BLAS と LAPACK を使うため、"Makevars" という名前のファイルを作り、

```
PKG_LIBS = $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)
```

と記述して、R のカレント・ディテクトリにおきます。

例題

例題 1: 内積 (1)

1) "myfunc.c" という名前の C ファイルを作成します。

```
#include <R_ext/blas.h>
void myfunc(double *a, double *b, int *c, double *d)
{
   int one=1;
  *d = F77_CALL(ddot)(c,a,&one,b,&one);
}
```

2) コマンドプロンプトから、

```
Rcmd SHLIB myfunc.c
```

と打ち込むことで、コンパイルできます。

3) R から、

```
dyn.load("myfunc.dll")
myfunc <- function(a,b){
    .C("myfunc",
      arg1=as.double(a),
      arg2=as.double(b),
      arg3=as.integer( (length(a)) ),
      arg4=as.double(0)
      )$arg4
 }と打ち込み、
```

例えば、

```
vec1<-c(1,2,3,4,5,6)
vec2<-c(2,3,4,5,6,8)
myfunc(vec1,vec2)
```

と打ち込めば、各々結果が返されます。

```
myfunc(c(1,2,3,4,5,6), c(2,3,4,5,6,8))
```

でも大丈夫です。

ちなみに、共用ライブラリを切り離すには、

```
dyn.unload("myfunc.dll")
```

と打ち込めばよい。

例題 2: 内積 (2)

"SEXP" を使う場合。

1) "myfunc2.c" という名前の C ファイルを作成します。

```
#include <R.h>
#include <Rinternals.h>
#include <R_ext/blas.h>
SEXP myfunc2(SEXP a, SEXP b)
{
  int na, nb;
  na = length(a); nb = length(b);
  const int one = 1;
  SEXP ans;
  PROTECT(ans = allocVector(REALSXP, one));
  REAL(ans)[0] = F77_CALL(ddot)(&na,REAL(a),&one,REAL(b),&one);
  UNPROTECT(1);
  return(ans);
}
```

2) コマンドプロンプトから、

```
Rcmd SHLIB myfunc2.c
```

と打ち込むことで、コンパイルできます。

3) R から、

```
dyn.load("myfunc2.dll")
myfunc2 <- function(a, b) .Call("myfunc2", a, b)
```

と打ち込み、例えば、

```
myfunc2(c(1,2,3),c(4,5,6))
```

とすれば結果が返されます。

C++ の場合

C++ で関数を書いた場合には、

```
extern "C"{}
```

でくくっておけば R から呼び出せます。

```
//myfunc2.cpp（C++ 独自の機能を使っていないのでよい例ではありません。）
extern "C"{
#include <R.h>
```

```
#include <Rinternals.h>
#include <R_ext/blas.h>
}

extern "C"{
SEXP myfunc2(SEXP a, SEXP b)
{
  int na, nb;
  na = length(a); nb = length(b);
  const int one = 1;
  SEXP ans;
  PROTECT(ans = allocVector(REALSXP, one));
  REAL(ans)[0] = F77_CALL(ddot)(&na,REAL(a),&one,REAL(b),&one);
  UNPROTECT(1);
  return(ans);
}
}
```

optim() について

optim() は R に組み込まれている汎用最適化関数で、多変数関数の最小値（マイナスをつければ最大値）が求められます。

RjpWiki の関数の最大･最小化の項を参照してください。BLAS/LAPACK の場合と同じく、C で書かれたのプログラムの中に直接 optim() のコード（正確には、そこから呼び出される vmmin()、nmmin() や samin()）を組み込むことで、大規模なプログラムを組む場合の高速化が実現できます。

Nelder-Mead 法の nmmin() をつかう場合

まず、"myfunc3.c" というファイル名で以下を保存します。

```
#include <R.h>
#include <Rinternals.h>
#include <R_ext/Applic.h>

double f(int n, double *x, void *dummy){
  double ans = 0;
  ans = x[0]*x[0] + (x[1]+1)*(x[1]+1) + 2;
  return ans;
}

SEXP myfunc3(SEXP dum)
{
  double x[2]={0,0};
  double opar[2]={0,0};
  double Fmin;
  int fncount, ifail;

  nmmin(2, x, opar, &Fmin, f,&ifail, 1.0e-8, 1.0e-8, NULL,
  1, 0.5, 2, 0,&fncount, 1000);

  SEXP ans;
  PROTECT(ans = allocVector(REALSXP, 3));
  REAL(ans)[0] = Fmin;
  REAL(ans)[1] = x[0];
  REAL(ans)[2] = x[1];
  UNPROTECT(1);
  return(ans);
}
```

次に、コマンドプロンプトから

```
Rcmd shlib myfunc3.c
```

でコンパイル。

結果は R のコンソールから

```
dyn.load("myfunc3.dll")
res3 <- .Call("myfunc3")
res3
```

と打ち込めば出ます。