

# Incorporation of Low-Level Computer Code into R

Matt Wand

Statistics Research Forum

20th May, 2008

## The Cost of High-Levelness

How long does this R script take to run on your computer?

```
set.seed(34290)
A <- matrix(runif(100),10,10)
system("date")
for (i in 1:100000) solve(A)
system("date")
```

## High-Level versus Low-Level Languages

High-Level Language	Low-Level Language
Matlab	Fortran 77
R	C and C++
SAS IML	Fortran 90

On my fancy Macintosh laptop this afternoon it took:

21 seconds

## Scaling Up

If problem is increased by factor of 10  $\implies$  3.5 minutes.

If problem is increased by factor of 100  $\implies$  35 minutes.

If problem is increased by factor of 1000  $\implies$  5.8 hours.

If problem is increased by factor of 10000  $\implies$  2.4 days.

## Speeding things up

If I could invert 10000  $10 \times 10$  matrices in a milliseconds then

5.8 hours  $\rightarrow$  1 second

Low-level languages allow us to achieve this sort of speed improvement.

## Modern Computational Statistics

- Bootstrapping.
- Markov Chain Monte Carlo.
- Local polynomial smoothing.
- Simulation studies.

all involve loops like this toy example.

## My R packages on the Comprehensive R Archive Network (CRAN)

name	engine-room
KernSmooth	Fortran 77
SemiPar	whatever lme() uses
LowRankQP	C
feature	Fortran 77

## Elementary Example

Consider the function on  $n \in \mathbb{N}$ :

$$f(n) = \sum_{i=1}^n \sqrt{i}.$$

In R this is

```
sumsqrt <- function(n)
  return(sum(sqrt(1:n)))
```

```
subroutine sumsqrt(n,ans)
integer n,i
double precision ans,xi

ans = 0.0
do 10 i = 1,n
  xi = i
  ans = ans + sqrt(xi)
10 continue

return
end
```

and the Fortran 77 object file `sumsqrt.so` is created from `sumsqrt.f`

## Fortran 77 Answer

```
sumsqrtf77 <- function(n)
{
  dyn.load("sumsqrt.so")
  ans <- 0
  out <- .Fortran("sumsqrt", as.integer(n),
                 ans=as.double(ans))
  return(out$ans)
}
```

where the file `sumsqrt.f` contains the Fortran 77 code:

## Question from My End

Best way to create Fortran 77 object files  
on

Macintosh Laptops?

## Matrix Algebra

Fitting local polynomial smoothers over a grid involves solving hundreds of matrix equations.

In my R package Kernsmooth this is done via the following Fortran 77 code:

```
do 80 k = 1,M
  do 90 i = 1,ipp
    do 100 j = 1,ipp
      indss = i + j - 1
      Smat(i,j) = ss(k,indss)
100    continue
      Tvec(i) = tt(k,i)
90    continue

  call dgefa(Smat,ipp,ipp,ipvt,info)
  call dgesl(Smat,ipp,ipp,ipvt,Tvec,0)

  curvest(k) = Tvec(idrv+1)
80  continue
```

The functions `dgefa` and `dgesl` are LINPACK routines.

## Random Sample Generation

Suppose you want to do Markov Chain Monte Carlo inside Fortran 77 (inside R).

The random random sample generation becomes an issue.

In 2003 I worked up the gumption to ask Brian Ripley (Oxford U.) how to do this.

I was pleased when he responded with the answer on following page....

## Ripley Advice on Random Sample Generation

Matt,

Here is a small example. Files

```
frand.f:
  subroutine frand(n, m, sum)
  dimension x(n)
  double precision x, sum
  call nrand(n, x)
  sum = 0.0
  do 10 i = 1, n
    sum = sum + x(i)**m
  10  end
```

```

frand_helper.c:
#include <R.h>
#include <R_ext/Random.h>

void F77_CALL(nrand)(int *n, double *x)
{
    int i;
    GetRNGstate();
    for (i = 0; i < *n; i++) x[i] = norm_rand();
    PutRNGstate();
}

```

make with

```
R CMD SHLIB -o frand.so frand.f frand_helper.c
```

and call from R  
set.seed(123)

```

dyn.load("frand.so")
.Fortran("frand", as.integer(10), as.integer(3), sum=double(1))$sum
[1] 6.306811

```

is the same as

```
set.seed(123); sum(rnorm(10)^3)
```

I hope that is enough for you to see the pattern.  
Brian

## Making use of Ripley Advice

In 2003 I checked that the Ripley answer worked alright for my MCMC problem.

I'm pleased to report that it did (at least on Linux machine I was using then).

Since then I haven't gotten back to the problem that led to this question (to Ripley).

However, it is part of new ARC grant starting here (with post-doc John Ormerod starting tomorrow!).

## Closing Remarks

- Low-level programming still has an important place in current-day methodological development.
- It can mean the difference between 'feasible' and 'infeasible' for some methods.
- I recommend learning a bit about this – not shying away from it.
- There is some expertise in our group on this topic.