Using C++ to speed up R code

Théo Michelot

School of Mathematics and Statistics

2 May 2017

Introduction

R is a great tool for data analysis:

- easy to learn;
- designed with statistics in mind;
- thousands of packages.

But the convenience and flexibity come at the cost of speed.

C++ (or e.g. Fortran) is a compiled programming language:

- more complex and rigorous syntax;
- usually very fast.

Introduction

```r
# R
n <- 0

for(i in 1:1e8) # big loop
  n <- n + 1
```

```
Time difference of 28.70 secs
```

```cpp
// C++
int n = 0;

for(int i=1; i<=1E8; i++) // big loop
   n = n + 1;
```

```
Time difference of 0.27 secs
```

Rcpp

Prehistory

The base R function `.Call()` loads C++ functions, but is not very user-friendly.

```
dyn.load("myfun.so")

myfunR <- function(arg1, arg2, arg3) {
  .Call("myfunCpp", arg1, arg2, arg3)
}

# Now I can use myfunR as a normal R function
```

where `myfunCpp` is defined in the file "myfun.cpp".

$\longrightarrow$ Rcpp takes care of all the annoying technical tweaks.

Rcpp

Rcpp is an R package. It can be installed with:

```
install.packages("Rcpp")
```

Then:

- Write code in C++, using a simplified syntax quite close to R;
- Rcpp compiles the C++ code, and generates an R function for each C++ function;
- The C++ functions can be called from R.

Note: more than 800 R packages on CRAN use Rcpp under the hood.

New C++ types

Each variable is defined with a type in C++, e.g. `int`, `double`, `char`...

It is tricky to assign a C++ type to R variables. Rcpp includes a few additional types:

- `NumericVector`, `CharacterVector`, `LogicalVector`...
- `NumericMatrix`, `CharacterMatrix`, `LogicalMatrix`...
- `List`
- `DataFrame`
- ...

Rcpp example

example.cpp:

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

// [[Rcpp::export]]
double matsum_rcpp(NumericMatrix mat)
{
  double res = 0;

  for(int i=0; i<mat.nrow(); i++)
    for(int j=0; j<mat.ncol(); j++)
      res = res + mat[i,j];

  return res;
}
```

Rcpp example

```
library(Rcpp)
sourceCpp("example.cpp")

A <- matrix(1,3,3)
s1 <- matsum_rcpp(A)

B <- matrix(2,3,3)
s2 <- matsum_rcpp(B)
```

```
> s1
[1] 9
> s2
[1] 18
```

RcppArmadillo

Armadillo is a C++ library ($\approx$ package) for linear algebra, and the R package **RcppArmadillo** makes it available from Rcpp.

Features include:

- better vector and matrix types, `arma::vec` and `arma::mat`;
- very fast routines for linear algebra operations, e.g. matrix multiplication, matrix inverse, linear system solving...

RcppArmadillo example

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp;

// [[Rcpp::export]]
arma::mat matprod_rcpp(arma::mat mat1, arma::mat mat2)
{
  if(mat1.n_cols != mat2.n_rows)
    stop("Incompatible matrix dimensions.");

  return mat1*mat2;
}
```

RcppArmadillo example

```
library(RcppArmadillo)
sourceCpp("example.cpp")

A <- matrix(1:9,3,3)
B <- matrix(1:9,3,3)

P <- matprod_rcpp(A,B)
```

```
> P
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150

> A%*%B
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

RcppArmadillo example

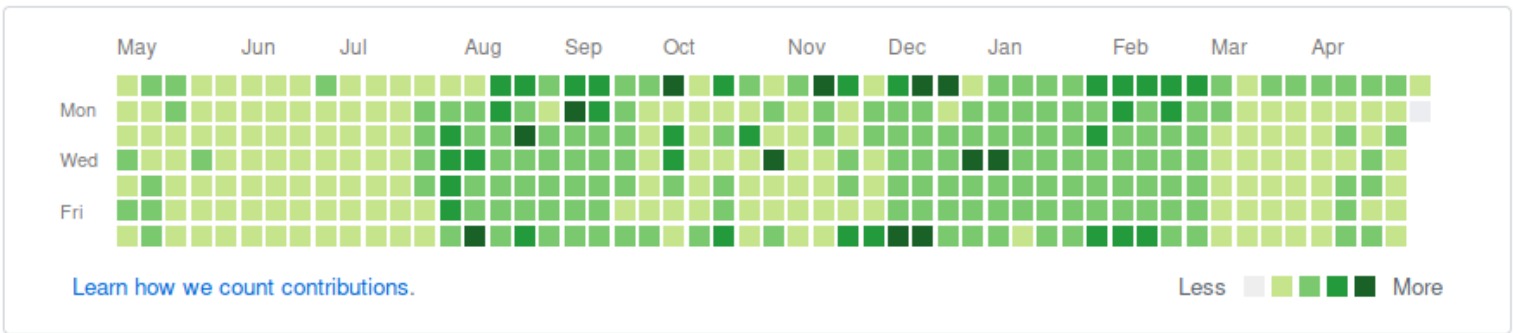
```
A <- matrix(1:8,4,2)
B <- matrix(1:9,3,3)

> P <- matprod_rcpp(A,B)
Error in matprod_rcpp(A, B) : Incompatible matrix dimensions
    .
```

Other functionalities

- Common probability distributions can be used in C++ code written with Rcpp: `R::dnorm`, `R::dgamma`, `R::dexp`...
- An alternative to RcppArmadillo is RcppEigen.
- Dirk Eddelbuettel

Dirk Eddelbuettel
227k • 26 • 411 • 520

2,373 contributions in the last year



Template Model Builder (TMB)

Automatic differentiation

Automatic differentation is a method of numerical evaluation of the gradient of a function.

Idea:

1. Write the function as the combination of many basic functions, like additions, multiplications, exponentials...

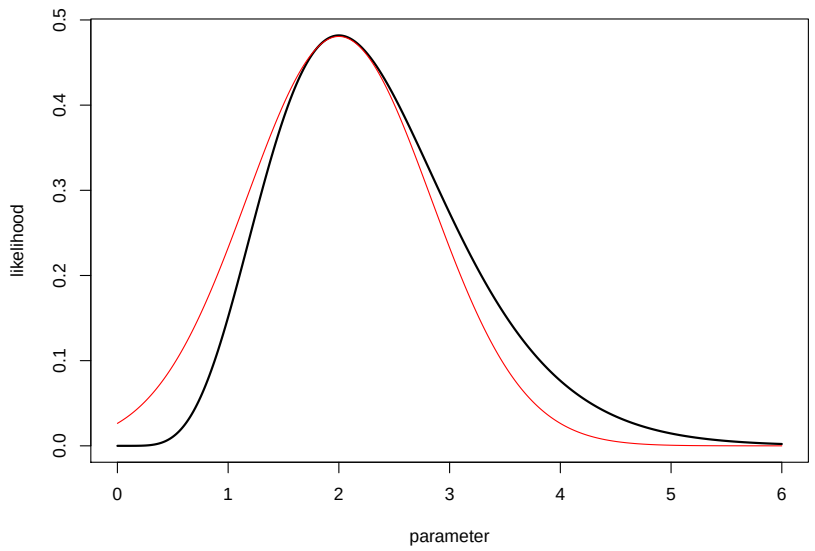
$$f = f_1 \circ f_2 \circ \cdots \circ f_k$$

2. Differentiate each basic function f_i.
3. Use the chain rule to obtain the derivative of f.

$\longrightarrow$ Can come in very handy for numerical optimisation, e.g. maximum likelihood estimation.

Laplace approximation

The **Laplace approximation** of a likelihood function consists in locally approximating it by a Gaussian probability function.



$\longrightarrow$ It's very simple to differentiate the approximate likelihood.

TMB workflow

1. Write the likelihood function in C++ using templates.
   (Or the negative log-likelihood, typically.)
2. Compile the C++ code from R.
3. TMB returns an R object which contains
   - the likelihood function;
   - the gradient (function) of the likelihood.
4. Optimise the likelihood in R.

Optimisers can often take the gradient function as an argument, to make things faster.

TMB example

```
#include <TMB.hpp>

template<class Type>
Type objective_function<Type>::operator() ()
{
  // data (input from R)
  DATA_VECTOR(x);
  // parameters (input from R)
  PARAMETER(mu);
  PARAMETER(sigma);

  Type f = -sum(dnorm(x,mu,sigma,true));
  return f;
}
```

```
library(TMB)
compile("nllk.cpp")
dyn.load(dynlib("nllk"))

# simulate data
x <- rnorm(n=1e4, mean=3, sd=10)

# create likelihood object
nllk <- MakeADFun(data=list(x=x), parameters=list(mu=0,sigma=1))

# estimate maximum likelihood
fit <- nlminb(start=nllk$par, objective=nllk$fn, gradient=nllk$gr,
              lower=c(-Inf,0), upper=c(Inf,Inf))
```

TMB example

```
> fit
$par
       mu     sigma
2.965458 9.947335

$objective
[1] 37162.43
```

Speed comparison

Normal distribution

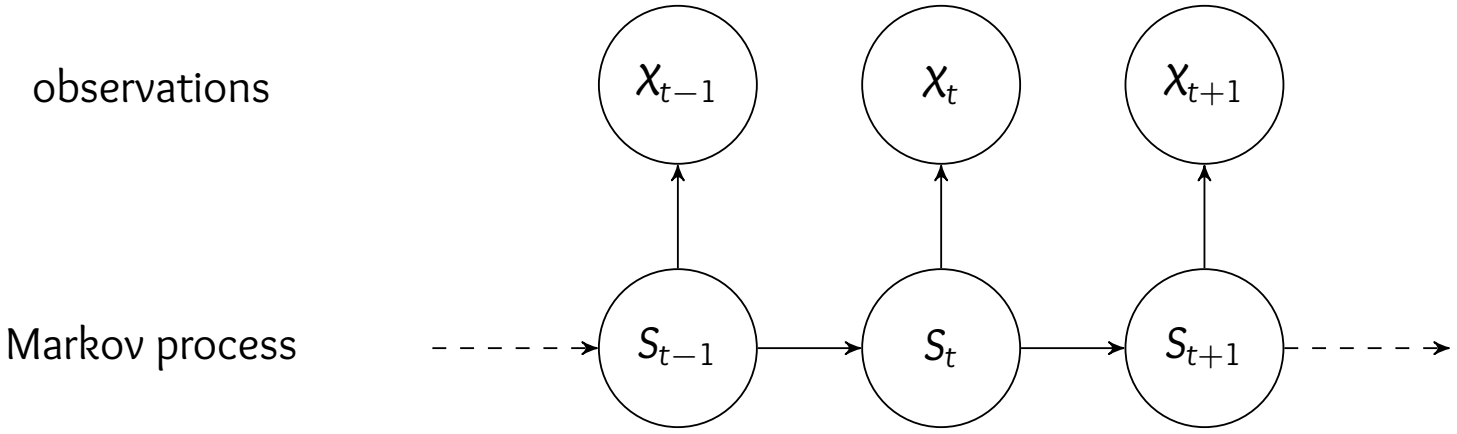
10^6 samples simulated from $N(3, 10^2)$
$\longrightarrow$ Estimate μ and σ by MLE.

	Pure R	Rcpp	TMB
Δt	2.61 secs	1.86 sec	1.84 sec
$\Delta t / \min(\Delta t)$	1.4	1.0	1.0
$\log(L_{\max})$	-3721981	-3721981	-3721981

Normal distribution

```
> dnorm
function (x, mean = 0, sd = 1, log = FALSE)
.Call(C_dnorm, x, mean, sd, log)
<bytecode: 0x27d9268>
<environment: namespace:stats>

> sum
function (..., na.rm = FALSE)  .Primitive("sum")
```

Poisson hidden Markov model



where

$$X_t \sim Poisson(\lambda_{S_t}), \qquad S_t \in \{1, 2\}$$

Parameters to estimate:

- λ_1, λ_2,
- $\gamma_{21} = \Pr(S_{t+1} = 2 | S_t = 1)$, $\gamma_{12} = \Pr(S_{t+1} = 1 | S_t = 2)$.

$\longrightarrow$ The likelihood is a matrix product with $2N$ terms.

Poisson hidden Markov model

10^5 samples simulated from the 2-state Poisson HMM
$\longrightarrow$ Estimate λ_1, λ_2, γ_{12} and γ_{21} by MLE.

	Pure R	Rcpp	TMB
Δt	1.16 min	6.04 secs	1.21 sec
$\Delta t / \min(\Delta t)$	57.5	5.0	1.0
$\log(L_{\max})$	-290672.9	-290672.9	-291233.3

Thanks for your attention

Eddelbuettel, D., and Francois, R. (2011). Rcpp: Seamless R and C++ Integration. *Journal of Statistical Software*, 40(8), 1-18.

Kristensen, K., Nielsen, A., Berg, C.W., Skaug, H., and Bell, B.M. (2016). TMB: Automatic Differentiation and Laplace Approximation. *Journal of Statistical Software*, 70(5), 1-21.