Integrating R and C++ exemplified by stochastic gradient MCMC sampling

Panagiotis Papastamoulis

Department of Statistics
Athens University of Economics and Business, Greece

Post-graduate Seminar
AUEB Stat Seminars

January 15, 2021

Overview

Rcpp

- R is an extensible system
- R code can be augmented with compiled code
- E.g. "Writing R extensions" manual
  - C
  - C++
  - Fortran
- The Rcpp package simplifies the usage of C++ in R
  - Eddelbuettel and François (2011)
  - Eddelbuettel, 2013
  - http://CRAN.R-project.org/package=Rcpp

FAQ

- Detailed documentation: `Rcpp-FAQ.pdf` vignette available online

FAQ

- Detailed documentation: `Rcpp-FAQ.pdf` vignette available online
- What do I need
    - a development environment with a suitable compiler, header files and required libraries
    - `R` should be built in a way that permits linking and possibly embedding of `R`
    - standard development tools such as `make`, etc

FAQ

- Detailed documentation: `Rcpp-FAQ.pdf` vignette available online
- What do I need
  - a development environment with a suitable compiler, header files and required libraries
  - `R` should be built in a way that permits linking and possibly embedding of `R`
  - standard development tools such as `make`, etc
- What compiler can I use
  - the GNU Compiler Collection
  - along with the corresponding `g++` compiler for the `C++` language

FAQ

- Detailed documentation: `Rcpp-FAQ.pdf` vignette available online
- What do I need
  - a development environment with a suitable compiler, header files and required libraries
  - `R` should be built in a way that permits linking and possibly embedding of `R`
  - standard development tools such as `make`, etc
- What compiler can I use
  - the GNU Compiler Collection
  - along with the corresponding `g++` compiler for the `C++` language
- More specifically
  - Windows: install `Rtools` toolchain
  - OS X: install Apple Developer Tools (`Xcode` (OS X$\leqslant$10.8) or `Xcode Command Line Tools` (OS X$\geqslant$10.9)
  - Linux: ☺

The standard way (without Rcpp)

```
Description:
     Functions to make calls to compiled code
     that has been loaded into R.
Usage:
            .C(.NAME, ...)
      .Fortran(.NAME, ...)
```

`.NAME`: a character string giving the name of a `C` function or `Fortran` subroutine

The standard way (without Rcpp)[1]

- **R CMD SHLIB** `[options] [-o dllname] files`

[1] note to Windows useRs: **`Rtools`** is required here

The standard way (without Rcpp)[1]

- **R CMD SHLIB** `[options] [-o dllname] files`
  1. Compile the given source files
  2. Create a dynamically loadable library (DLL)

[1] note to Windows useRs: **Rtools** is required here

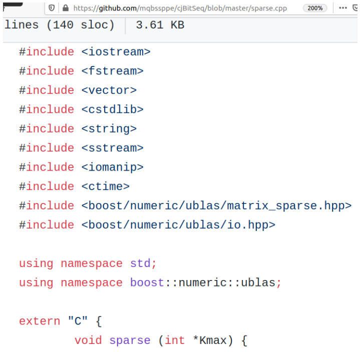
The standard way (without Rcpp)[1]

- **R CMD SHLIB** `[options] [-o dllname] files`
    1. Compile the given source files
    2. Create a dynamically loadable library (DLL)
- Finally, the command **`dyn.load`** links the specified DLL to the executing R image

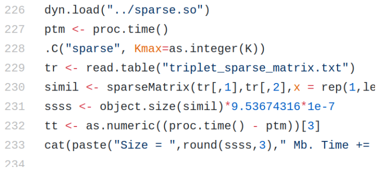
[1] note to Windows useRs: **`Rtools`** is required here

The standard way (without R Rcpp)[1]

- **R CMD SHLIB** `[options] [-o dllname] files`
  1. Compile the given source files
  2. Create a dynamically loadable library (DLL)
- Finally, the command **dyn.load** links the specified DLL to the executing R image


https://github.com/mqbsspe/cjBitSeq/blob/master/sparse.cpp 200%

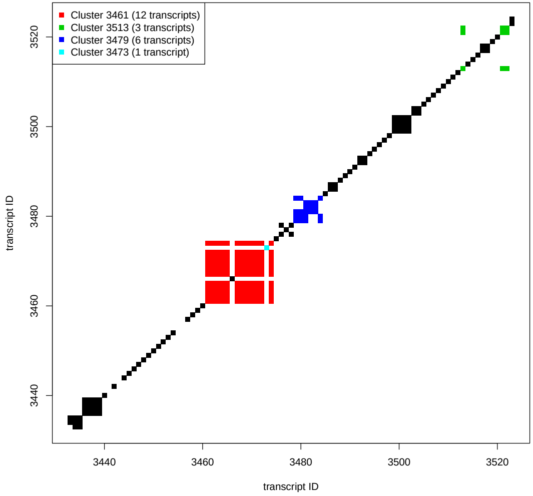
```
lines (140 sloc) | 3.61 KB

#include <iostream>
#include <fstream>
#include <vector>
#include <cstdlib>
#include <string>
#include <sstream>
#include <iomanip>
#include <ctime>
#include <boost/numeric/ublas/matrix_sparse.hpp>
#include <boost/numeric/ublas/io.hpp>

using namespace std;
using namespace boost::numeric::ublas;

extern "C" {
        void sparse (int *Kmax) {
```


```
dyn.load("../sparse.so")
ptm <- proc.time()
.C("sparse", Kmax=as.integer(K))
tr <- read.table("triplet_sparse_matrix.txt")
simil <- sparseMatrix(tr[,1],tr[,2],x = rep(1,le
ssss <- object.size(simil)*9.53674316*1e-7
tt <- as.numeric((proc.time() - ptm))[3]
cat(paste("Size = ",round(ssss,3)," Mb. Time +=
```

C++ code

Call within R

[1] note to Windows useRs: **Rtools** is required here



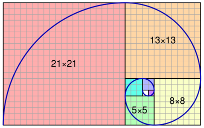
- $K \times K$ sparse matrix, where K is measured in tens of thousands
- Loop through millions/billions of data
- Papastamoulis and Rattray, 2018 (JRSSC)

Rcpp

- Application Programming Interface
- Defines interactions between multiple software intermediaries
- It provides a consistent C++ class hierarchy that maps various types of R objects to dedicates C++ classes
  - vectors
  - matrices
  - functions
  - environments
- Rcpp substantially lowers the barrier for programmers wanting to combine C++ code with R, e.g.
  - only a single header file is needed to use the Rcpp API
- Articles and code examples for the Rcpp package

https://gallery.rcpp.org/

Toy example: Fibonacci sequence[2]



- The Fibonacci sequence F_n is defined as a recursive sum of the two preceding terms in the same sequence

$$F_n = F_{n-1} + F_{n-2}$$

- with initial conditions

$$F_0 = 0 \quad \text{and} \quad F_1 = 1$$

- It looks like

n	0	1	2	3	4	5	6	7	8	9	10	11	$\cdots$
F_n	0	1	1	2	3	5	8	13	21	34	55	89	$\cdots$

[2]Image: By Jahobr - Own work, CC0, Wikipedia

Toy example: Fibonacci sequence

- Let's implement this in R
- Via a simple recursive function

```
fibonacci_R <- function(n){
        if (n < 2) return(n)
        return (fibonacci_R(n - 1) + fibonacci_R(n - 2))
}
```

- It is very inefficient
  - $F_5 = F_4 + F_3$
  - $F_4 = F_3 + F_2$
  - the run time is exponential in n
  - alternative approaches do exist (e.g. *memoization* techniques)
  - normally we would also return an error message in case of non-integer input
  - but we won't be bothered by that

Same with Rcpp

C++ code in file **src/fibonacci.cpp**

```
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
int fibonacci_cpp(const int x){
        if (x < 2)
          return x;
        else
          return (fibonacci_cpp(x - 1)) + fibonacci_cpp(x - 2);
}
```

Compile and call our C++ function inside R session

```
> library("Rcpp")
> sourceCpp("src/fibonacci.cpp")
```

- `sourceCpp()` parses the specified C++ file or source code
- looks for functions marked with the `Rcpp::export` attribute
- A shared library is then built and its exported functions and Rcpp modules are made available in the specified environment
- Now we are ready to call the compiled code inside R

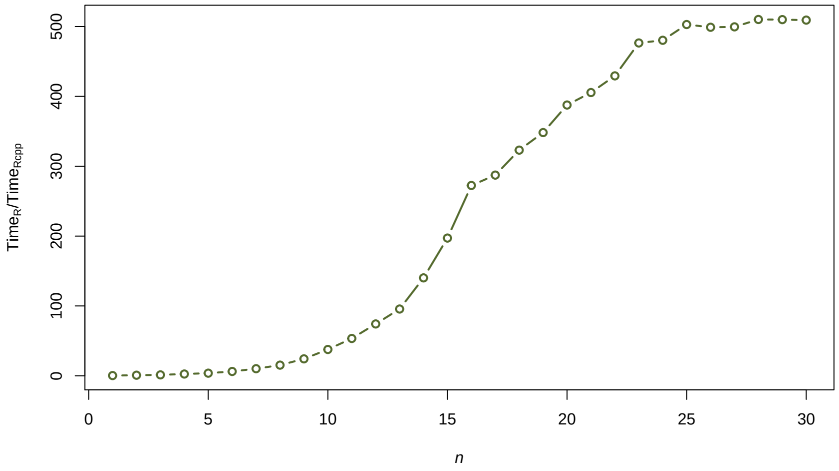
```
> fibonacci_cpp(10)
[1] 55
```

Benchmarking

- Let’s compare the elapsed time needed for computing F_{20}
- The results are averages across 500 replications
- The `microbenchmark` package was used, which provides sub-millisecond accurate timing of expression evaluation

```
> library("microbenchmark")
> library("Rcpp")
> sourceCpp("src/fibonacci.cpp")
> source("src/fibonacci.R")
> m <- microbenchmark(
+      rCode <- fibonacci_R(20),
+      cppCode <- fibonacci_cpp(20),
+      unit = "ms", times = 500)
> print(m, digits = 2)
Unit: milliseconds
     expr   min    lq  mean median    uq   max neval
    rCode 7.551 7.689 8.190   7.83 7.994 13.39   500
  cppCode 0.018 0.018 0.023   0.02 0.023  0.93   500
```

Benchmarking (as a function of n)



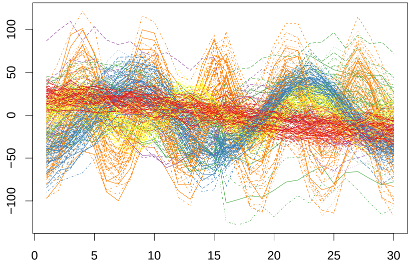
- Relative time required to compute the first n Fibonacci numbers
- The compiled version is over 500 times faster as n increases

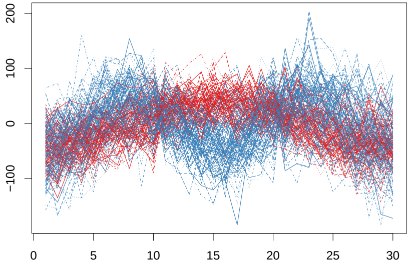
Take-away message

- There is obvious merit in replacing simple `R` code with simple `C++` code.
- Switching implementation languages to `C++` significantly boosts run-time performance
- However, no matter what the chosen implementation language, an exponential algorithm will eventually be inapplicable provided the argument n is large enough

Application 1: Gibbs update

- Clustering multivariate data using Bayesian MFA
  - Papastamoulis, 2018: Overfitting Bayesian mixtures of factor analyzers with an unknown number of components, In *CSDA*
  - Papastamoulis, 2020: Clustering multivariate data using factor analytic Bayesian Mixtures of Factor Analyzers, In *Statistics and Computing*
- CRAN package available online at
  `https://CRAN.R-project.org/package=fabMix`





Finite Mixtures of Factor Analyzers (MFA)

$$(\boldsymbol{x}_i | z_i = k) = \boldsymbol{\mu}_k + \boldsymbol{\Lambda}_k \boldsymbol{y}_i + \varepsilon_i$$

independent for $i = 1, \ldots, n$. Assume

$$\left.\begin{aligned} \boldsymbol{y}_i &\sim \mathcal{N}_q(0, \mathbf{I}_q) \\ \varepsilon_i | z_i = k &\sim \mathcal{N}_p(0, \boldsymbol{\Sigma}_k) \end{aligned}\right\}, \quad \text{independent for } i = 1, \ldots, n$$

where $\boldsymbol{\Sigma}_k = \operatorname{diag}(\sigma^2_{k1}, \ldots, \sigma^2_{kp})$. Then

$$\begin{aligned} \boldsymbol{x}_i | z_i = k, \boldsymbol{y}_i &\sim \mathcal{N}_p(\boldsymbol{\mu}_k + \boldsymbol{\Lambda}_k \boldsymbol{y}_i, \boldsymbol{\Sigma}_k) \\ \mathrm{P}(z_i = k) &= w_k, \quad \text{independent for } i = 1, \ldots, n \\ \boldsymbol{x}_i &\sim \sum_{k=1}^{K} w_k \mathcal{N}_p(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k \boldsymbol{\Lambda}_k^T + \boldsymbol{\Sigma}_k), \end{aligned}$$

where $\sum_{k=1}^{K} w_k = 1$ and $w_k \geqslant 0$, $k = 1, \ldots, K$.

MCMC: Prior parallel tempering + Gibbs sampler

Input: $n_{\text{chains}}, \boldsymbol{x}, K_{\max}, \alpha, \beta, \gamma, g, h$

- For all model parameterizations $m \in \mathcal{M}$ and number of factors $q \in \mathcal{Q}$

```
1 $t = 0$: set initial values
2 $t \leftarrow t + 1$: run in parallel for $j = 1, \ldots, n_{\text{chains}}$:
    2.1 $\boldsymbol{\Omega}^{(t,j)} \sim [\boldsymbol{\Omega} | \boldsymbol{\Lambda}_m^{(t-1,j)}]$
    2.2 $\boldsymbol{\Lambda}_m^{(t,j)} \sim [\boldsymbol{\Lambda}_m | \boldsymbol{\Omega}^{(t,j)}, \boldsymbol{\Sigma}_m^{(t-1,j)}, \boldsymbol{x}, \boldsymbol{y}^{(t-1,j)}, \boldsymbol{z}^{(t-1,j)}]$
    2.3 $\boldsymbol{\mu}^{(t,j)} \sim [\boldsymbol{\mu} | \boldsymbol{\Lambda}_m^{(t,j)}, \boldsymbol{\Sigma}_m^{(t-1,j)}, \boldsymbol{x}, \boldsymbol{y}^{(t-1,j)}, \boldsymbol{z}^{(t-1,j)}]$
    2.4 $\boldsymbol{z}^{(t,j)} \sim [\boldsymbol{z} | \boldsymbol{w}^{(t-1,j)}, \boldsymbol{\mu}^{(t,j)}, \boldsymbol{\Lambda}_m^{(t,j)}, \boldsymbol{\Sigma}_m^{(t-1,j)}, \boldsymbol{x}]$
    2.5 $\boldsymbol{\Sigma}_m^{(t,j)} \sim [\boldsymbol{\Sigma}_m | \boldsymbol{x}, \boldsymbol{z}^{(t,j)}, \boldsymbol{\mu}^{(t,j)}]$
    2.6 $\boldsymbol{w}^{(t,j)} \sim [\boldsymbol{w} | \boldsymbol{z}^{(t,j)}]$
    2.7 $\boldsymbol{y}^{(t,j)} \sim [\boldsymbol{y} | \boldsymbol{x}, \boldsymbol{z}^{(t,j)}, \boldsymbol{\mu}^{(t,j)}, \boldsymbol{\Sigma}_m^{(t,j)}, \boldsymbol{\Lambda}_m^{(t,j)}]$.
3 Propose a swap between two chains $j_1$ and $j_2$
4 Repeat 2 and 3
5 For chain $j = 1$ compute BIC conditionally on the most probable
  number of alive clusters.
```

- Select the best (m, q) model for chain $j = 1$ according to BIC, conditional on $K^{(\text{map})}$.

Full conditionals for $\Lambda, \boldsymbol{\mu}$ (step 2.2 + 2.3)

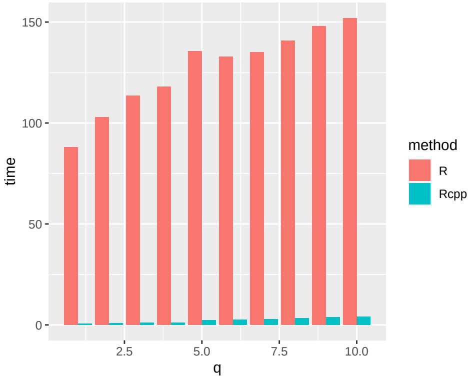
For $k = 1, \ldots, K$ and $r = 1, \ldots, p$

$$\boldsymbol{\mu}_k | \cdots \sim \mathcal{N}_p\left(\boldsymbol{A}_k^{-1}\boldsymbol{B}_k, \boldsymbol{A}_k^{-1}\right)$$
$$\boldsymbol{\Lambda}_{kr\cdot} | \cdots \sim \mathcal{N}_{\nu_r}\left(\left[\boldsymbol{\Omega}^{-1} + \boldsymbol{\Delta}_{kr}\right]^{-1}\boldsymbol{\tau}_{kr}, \left[\boldsymbol{\Omega}^{-1} + \boldsymbol{\Delta}_{kr}\right]^{-1}\right)$$

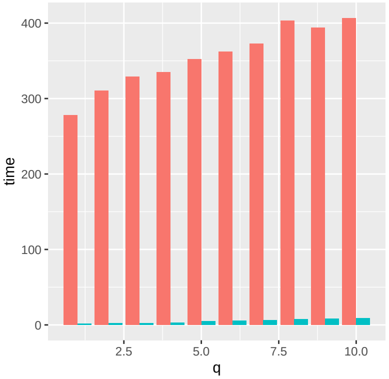
where

$$\boldsymbol{A}_k = n_k\boldsymbol{\Sigma}_k^{-1} + \Psi^{-1}$$
$$\boldsymbol{B}_k = \xi\boldsymbol{\Psi}^{-1} + \boldsymbol{\Sigma}_k^{-1}\sum_{i=1}^{n} I(z_i = k)\left(\boldsymbol{x}_i - \boldsymbol{\Lambda}_k\boldsymbol{y}_i\right)$$
$$\boldsymbol{\Delta}_{kr} = \frac{\sum_{i=1}^{n} I(z_i = k)\boldsymbol{y}_i\boldsymbol{y}_i^T}{\boldsymbol{\sigma}_{kr}^2}$$
$$\boldsymbol{\tau}_{kr} = \frac{\sum_{i=1}^{n} I(z_i = k)(x_{ir} - \mu_{kr})\boldsymbol{y}_i^T}{\sigma_{kr}^2}$$

Application 1: Gibbs update for (Λ, μ) (step 2.2 + 2.3)



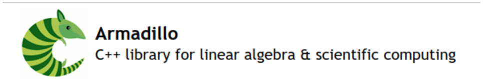
$p = 100$



$p = 200$

- p: number of variables
- q: number of factors
- $n = 1000$ observations, $K = 10$ clusters

The RcppArmadillo package



- Armadillo library for C++
  - is a high quality linear algebra library for the C++ language
  - provides efficient classes for vectors, matrices and cubes; dense and sparse matrices are supported
  - See http://arma.sourceforge.net/speed.html for timing comparisons against other C++ matrix libraries
  - Documentation http://arma.sourceforge.net/docs.html
- RcppArmadillo package for Rcpp
  - it includes the header files from the templated Armadillo library
  - users do not need to install Armadillo itself in order to use RcppArmadillo
  - https://CRAN.R-project.org/package=RcppArmadillo

Back to our application: Gibbs update for $\boldsymbol{\Lambda}, \boldsymbol{\mu}$

- $\boldsymbol{\mu} : K \times p$ matrix
  - in `RcppArmadillo` class `arma::mat`
- $\boldsymbol{\Lambda} : p \times q \times K$ array (three-dimensional matrix)
  - in `RcppArmadillo` class `arma::cube`
- Other useful classes
  - `arma::vec` for vectors
  - `Rcpp::List` for list-like objects
- We are going to define a function that accepts as input
  - fixed hyperparameters
  - sufficient statistics
- and returns a `list` with two arguments
  - `mu`
  - `Lambdas`
- which correspond to the Gibbs update of $\boldsymbol{\Lambda}, \boldsymbol{\mu}$

```r
rFunction <- function(
  SigmaINV,
  suff_stat,
  OmegaINV,
  K,
  priorConst1,
  T_INV,
  v_r){

  p <- dim(suff_stat$sx)[2]
  q <- dim(suff_stat$sy)[2]
  mu <- array(data = 0,
              dim = c(K,p))
  Lambdas <- array(data = 0,
              dim = c(K,p,q))
  ...
  result <- list(
      "Lambdas" = Lambdas,
      "mu" = mu)
  return(result)}
```

```cpp
#include <Rcpp.h>
// [[Rcpp::depends(RcppArmadillo)
using namespace Rcpp;
// [[Rcpp::export]]
List cppFunction(
  arma::mat SigmaINV,
  Rcpp::List suff_stat,
  arma::mat OmegaINV,
  int K,
  arma::vec priorConst1,
  arma::mat T_INV,
  arma::vec v_r){

  int p = SigmaINV.n_cols;
  arma::mat sy = suff_stat["sy"];
  int q = sy.n_cols;
  arma::mat mu(K,p);
  arma::cube Lambda(K,p,q);
  ...
  List result;
  result["Lambdas"] = Lambdas;
  result["mu"] = mu;
  return(result);}
```

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
`http://arma.sourceforge.net/docs.html`

- Initialize zero matrix **`A = arma::zeros(p,p);`**

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

- Initialize zero matrix **`A = arma::zeros(p,p);`**
- Access rows $1, \dots, i$ and columns $1, \dots, j$ of matrix
  `A( arma::span(0, i-1), arma::span(0, j-1) )`

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

- Initialize zero matrix **`A = arma::zeros(p,p);`**
- Access rows $1, \ldots, i$ and columns $1, \ldots, j$ of matrix **`A( arma::span(0, i-1), arma::span(0, j-1) )`**
- Matrix inversion **`inv(A);`**

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
`http://arma.sourceforge.net/docs.html`

- Initialize zero matrix **`A = arma::zeros(p,p);`**
- Access rows $1, \ldots, i$ and columns $1, \ldots, j$ of matrix **`A( arma::span(0, i-1), arma::span(0, j-1) )`**
- Matrix inversion **`inv(A);`**
- Transpose of matrix **`A.t()`**

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

- Initialize zero matrix `A = arma::zeros(p,p);`
- Access rows $1, \ldots, i$ and columns $1, \ldots, j$ of matrix `A( arma::span(0, i-1), arma::span(0, j-1) )`
- Matrix inversion `inv(A);`
- Transpose of matrix `A.t()`
- Convert vector to matrix `repmat(muMean,1,1).t()`

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

- Initialize zero matrix `A = arma::zeros(p,p);`
- Access rows $1, \ldots, i$ and columns $1, \ldots, j$ of matrix
  `A( arma::span(0, i-1), arma::span(0, j-1) )`
- Matrix inversion `inv(A);`
- Transpose of matrix `A.t()`
- Convert vector to matrix `repmat(muMean,1,1).t()`
- Matrix calculus `A + B * C`

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

- Initialize zero matrix **`A = arma::zeros(p,p);`**
- Access rows $1, \dots, i$ and columns $1, \dots, j$ of matrix
  `A( arma::span(0, i-1), arma::span(0, j-1) )`
- Matrix inversion **`inv(A);`**
- Transpose of matrix **`A.t()`**
- Convert vector to matrix **`repmat(muMean,1,1).t()`**
- Matrix calculus **`A + B * C`**
- Cube operations `Lambda[ , ,k]` **`Lambda.slice(k)`**

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
`http://arma.sourceforge.net/docs.html`

- Initialize zero matrix **`A = arma::zeros(p,p);`**
- Access rows $1,\ldots,i$ and columns $1,\ldots,j$ of matrix
  `A( arma::span(0, i-1), arma::span(0, j-1) )`
- Matrix inversion **`inv(A);`**
- Transpose of matrix **`A.t()`**
- Convert vector to matrix **`repmat(muMean,1,1).t()`**
- Matrix calculus **`A + B * C`**
- Cube operations `Lambda[ , ,k]` **`Lambda.slice(k)`**
- Compute Cholesky decomposition of matrix **`chol(Sigma)`**

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

- Initialize zero matrix **`A = arma::zeros(p,p);`**
- Access rows $1,\ldots,i$ and columns $1,\ldots,j$ of matrix
  `A( arma::span(0, i-1), arma::span(0, j-1) )`
- Matrix inversion **`inv(A);`**
- Transpose of matrix **`A.t()`**
- Convert vector to matrix **`repmat(muMean,1,1).t()`**
- Matrix calculus **`A + B * C`**
- Cube operations `Lambda[ , ,k]` **`Lambda.slice(k)`**
- Compute Cholesky decomposition of matrix **`chol(Sigma)`**
- $\mathcal{N}(0,1)$ pseudorandom numbers **`M = arma::randn(1, p);`**

Typical Matrix/Stats operations inside `cppfunction`

Detailed documentation available at
http://arma.sourceforge.net/docs.html

- Initialize zero matrix **`A = arma::zeros(p,p);`**
- Access rows $1, \dots, i$ and columns $1, \dots, j$ of matrix
  `A( arma::span(0, i-1), arma::span(0, j-1) )`
- Matrix inversion **`inv(A);`**
- Transpose of matrix **`A.t()`**
- Convert vector to matrix **`repmat(muMean,1,1).t()`**
- Matrix calculus **`A + B * C`**
- Cube operations `Lambda[ , ,k]` **`Lambda.slice(k)`**
- Compute Cholesky decomposition of matrix **`chol(Sigma)`**
- $\mathcal{N}(0,1)$ pseudorandom numbers **`M = arma::randn(1, p);`**
- Single draw from $\mathcal{N}_p(\text{muMean}, \text{Sigma})$
  `repmat(muMean,1,1).t() + M * chol(Sigma)`

Application 2: Mixtures of Multinomial Logit models

- We consider the problem of inferring an unknown number of clusters in replicated multinomial data
- Estimation of finite mixtures of multinomial distributions with or without covariates
- The potentially large number of parameters and the multimodal nature of the likelihood/posterior surface of mixture models impose certain inferential and computational difficulties.
- Under a Bayesian setup, a stochastic gradient Markov chain Monte Carlo (MCMC) algorithm embedded within a prior parallel tempering scheme is devised
- Papastamoulis and Karlis, 2021 (working paper)

Application 2: Mixtures of Multinomial Logit Models

- $(\boldsymbol{y}_i, \boldsymbol{x}_i)$; $i = 1, \ldots, n$
- $\boldsymbol{y}_i = (y_{i1}, \ldots, y_{i;J+1})$ multinomial response ($J + 1$ categories)
- $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{iP})$ covariates
- Joint pmf conditional on K clusters

$$f(\boldsymbol{y}|\boldsymbol{\pi}, \boldsymbol{\beta}, \boldsymbol{x}) = \prod_{i=1}^{n} \sum_{k=1}^{K} \pi_k \frac{S!}{\prod_{j=1}^{J+1} y_{ij}!} \prod_{j=1}^{J+1} y_{ij}^{g_{ikj}} \mathrm{I}_{\mathcal{Y}_J}(\boldsymbol{y}_i).$$

where

$$g_{ikj} = \begin{cases} \dfrac{\exp\{\boldsymbol{\beta}_{kj}^{\top}\boldsymbol{x}_i\}}{1 + \sum\limits_{\ell \leqslant J} \exp\{\boldsymbol{\beta}_{k\ell}^{\top}\boldsymbol{x}_i\}}, & j \leqslant J \\ \dfrac{1}{1 + \sum\limits_{\ell \leqslant J} \exp\{\boldsymbol{\beta}_{k\ell}^{\top}\boldsymbol{x}_i\}}, & j = J + 1 \end{cases} \tag{1}$$

for $i = 1, \ldots, n$; $k = 1, \ldots, K$.

- Number of parameters, conditional on K clusters
  $d = K - 1 + KJ(P + 1)$

MALA proposal

- The Metropolis Adjusted Langevin Algorithm (MALA) (Girolami and Calderhead, 2011; Roberts and Tweedie, 1996) is based on the following proposal mechanism

$$\widetilde{\boldsymbol{\beta}} = \boldsymbol{\beta}^{(t)} + \nu\nabla\log f(\boldsymbol{\beta}^{(t)}|\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p}) + \sqrt{2\nu}\boldsymbol{\varepsilon}, \tag{2}$$

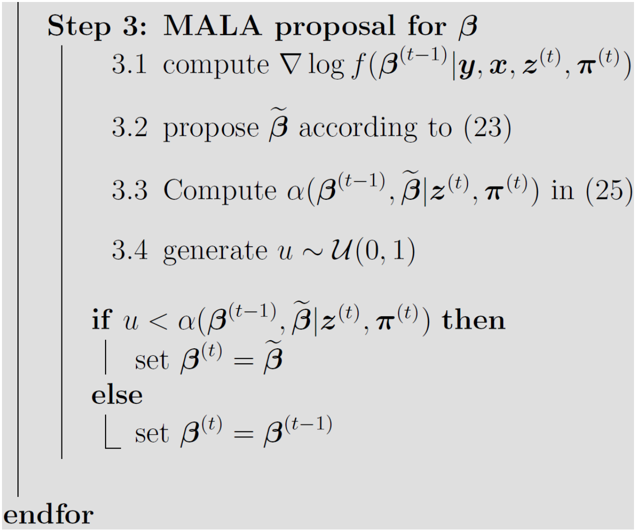
- For $k = 1, \ldots, K$, $j = 1, \ldots, J$, $p = 1, \ldots, P$

$$\frac{\partial \log f(\beta|\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p})}{\partial\beta_{kjp}} = \sum_{i=1}^{n} z_{ik}(y_{ij} - Sg_{ikj})x_{ip} - \frac{\beta_{kjp}}{\tau^2} \tag{3}$$

- Metropolis-Hastings acceptance probability

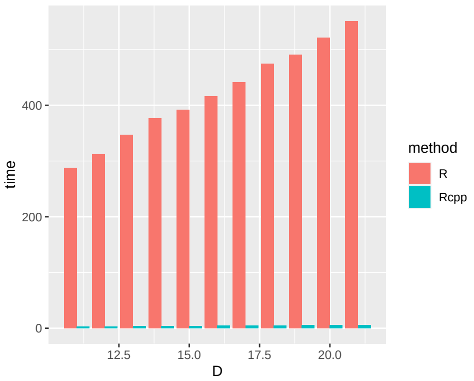
$$\alpha(\boldsymbol{\beta}^{(t)}, \widetilde{\boldsymbol{\beta}}|\boldsymbol{z}^{(t)}, \boldsymbol{\pi}^{(t)}) = \min\left\{1, \frac{f(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}^{(t)}, \widetilde{\boldsymbol{\beta}}, \boldsymbol{p}^{(t)})\pi(\widetilde{\boldsymbol{\beta}})}{f(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}^{(t)}, \boldsymbol{\beta}^{(t)}, \boldsymbol{p}^{(t)})\pi(\boldsymbol{\beta}^{(t)})} \frac{\mathrm{P}\left(\widetilde{\boldsymbol{\beta}} \to \boldsymbol{\beta}^{(t)}\right)}{\mathrm{P}\left(\boldsymbol{\beta}^{(t)} \to \widetilde{\boldsymbol{\beta}}\right)}\right\}, \tag{4}$$

Metropolis Adjusted Langevin Within-Gibbs MCMC

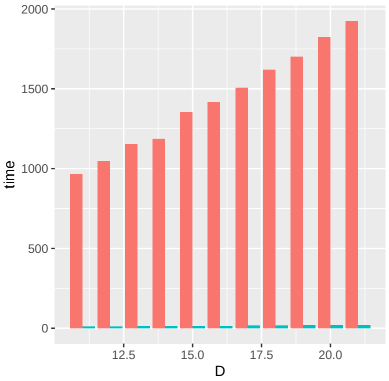


```
    **Step 3: MALA proposal for $\beta$**
        3.1 compute $\nabla \log f(\boldsymbol{\beta}^{(t-1)}|\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{z}^{(t)}, \boldsymbol{\pi}^{(t)})$

        3.2 propose $\widetilde{\boldsymbol{\beta}}$ according to (23)

        3.3 Compute $\alpha(\boldsymbol{\beta}^{(t-1)}, \widetilde{\boldsymbol{\beta}}|\boldsymbol{z}^{(t)}, \boldsymbol{\pi}^{(t)})$ in (25)

        3.4 generate $u \sim \mathcal{U}(0,1)$

      if $u < \alpha(\boldsymbol{\beta}^{(t-1)}, \widetilde{\boldsymbol{\beta}}|\boldsymbol{z}^{(t)}, \boldsymbol{\pi}^{(t)})$ then
          set $\boldsymbol{\beta}^{(t)} = \widetilde{\boldsymbol{\beta}}$
      else
          set $\boldsymbol{\beta}^{(t)} = \boldsymbol{\beta}^{(t-1)}$

endfor
```

Benchmarking



$K = 3$



$K = 10$

- D: number of categories
- K: number of clusters
- $n = 1000$ observations, $p = 5$ covariates

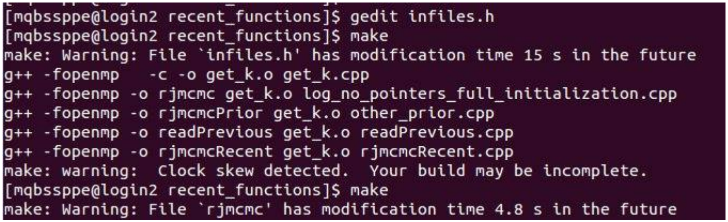
Session/System Info

```
> sessionInfo()
R version 4.0.3 (2020-10-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 20.04.1 LTS
```

Memory	15,5 GiB
CPU	Intel Core i7-4790 CPU @ 3.60GHz $\times$ 8

Further remarks

- Break your R code into smaller segments
- Within each segment put effort to optimize the R code
  - Do you really need a **for** loop?
  - Vectorize as much as you can
- Locate parts of the code which take too much time
- Implement these parts into Rcpp
- Enable parallelization for additional gains
  - parallel package
  - foreach package



```
[mqbssppe@login2 recent_functions]$ gedit infiles.h
[mqbssppe@login2 recent_functions]$ make
make: Warning: File `infiles.h' has modification time 15 s in the future
g++ -fopenmp   -c -o get_k.o get_k.cpp
g++ -fopenmp -o rjmcmc get_k.o log_no_pointers_full_initialization.cpp
g++ -fopenmp -o rjmcmcPrior get_k.o other_prior.cpp
g++ -fopenmp -o readPrevious get_k.o readPrevious.cpp
g++ -fopenmp -o rjmcmcRecent get_k.o rjmcmcRecent.cpp
make: warning:  Clock skew detected.  Your build may be incomplete.
[mqbssppe@login2 recent_functions]$ make
make: Warning: File `rjmcmc' has modification time 4.8 s in the future
```

Welcome to the future :)

Eddelbuettel, D. (2013). *Seamless R and C++ Integration with Rcpp.* ISBN 978-1-4614-6867-7. New York: Springer. DOI: 10.1007/978-1-4614-6868-4.

Eddelbuettel, D. and R. François (2011). "Rcpp: Seamless R and C++ Integration". In: *Journal of Statistical Software* 40.8, pp. 1–18. DOI: 10.18637/jss.v040.i08. URL: http://www.jstatsoft.org/v40/i08/.

Girolami, M. and B. Calderhead (2011). "Riemann manifold Langevin and Hamiltonian Monte Carlo methods". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2, pp. 123–214.

Papastamoulis, P. (2018). "Overfitting Bayesian mixtures of factor analyzers with an unknown number of components.". In: *Computational Statistics and Data Analysis* 124, pp. 220–234. DOI: 10.1016/j.csda.2018.03.007.

— (2020). "Clustering multivariate data using factor analytic Bayesian mixtures with an unknown number of components". In: *Statistics and Computing* 30.3, pp. 485–506.

Papastamoulis, P. and D. Karlis (2021). "Model based clustering of replicated multinomial data". In: *working paper*.

Papastamoulis, P. and M. Rattray (2018). "A Bayesian model selection approach for identifying differentially expressed transcripts from RNA sequencing data". In: *Journal of the Royal Statistical Society. Series C, Applied Statistics* 67.1, p. 3.

Roberts, G. O. and R. L. Tweedie (1996). "Exponential Convergence of Langevin Distributions and Their Discrete Approximations". In: *Bernoulli* 2.4, pp. 341–363. ISSN: 13507265. URL: http://www.jstor.org/stable/3318418.