C/C++, GSL, RcppGSL

Xueying Tang

Department of Statistics
University of Florida

November 19, 2015

Basic C/C++: Differences between C/C++ and R

- C is compiled; R is interpreted. `ex1.cpp`

```
#include <cstdio>
int main (void) {
    printf("Hello, World!\n");
    return 0;
}
```

To compile and link:

```
$ g++ -c ex1.cpp
$ g++ ex1.o -o ex1
```

or

```
$ g++ ex1.cpp -o ex1
```

To execute:

```
$ ./ex1
Hello, World!
```

Basic C/C++: Differences between C/C++ and R

- C is statically typed; R is dynamically typed.
  This is valid in R:

```
1 > x <- rnorm(10)
2 > x <- "some text"
```

  In C/C++:
  - Variables have to be declared first: assign the name of a variable to a particular type.
  - The type is then fixed for as long as this variable is in scope.
  - A certain number of assignments from one type to another are possible, but the assignment may be losing precision. `ex2.cpp`.

Basic C/C++: Differences between C and C++

C++ is a better C.

- Object-oriented: `class` data type.
- Generic programming and the STL: containers, iterators, algorithms, functors.
  - In C, vectors have to be created statically with size fixed at compile time, or dynamically.
- Template programming:
  Example: create a function that adds two numbers.
  In C:

```
1 int add_int (int x, int y) {
2   return x + y;
3 }
4 double add_double (double x, double y) {
5   return x + y;
6 }
```

Basic C/C++: Differences between C and C++

- Template programming: Example continued
  In C++, overloading is allowed. The compiler knows which one to call by examining the types passed as arguments when the function is called.

```
1 int add (int x, int y) return x + y;
2 double add (double x, double y) return x + y;
```

or we could use templates.

```
1 template <typename T>
2 T add (T x, T y) return x + y;
```

```
1 int m; double x;
2 m = add<int>(10, 12);
3 x = add<double>(1.0, 0.5);
```

The template parameters can not only include types introduced by class or typename, but can also include expressions of a particular type. ex3.cpp

How to use R in C/C++: R Standalone Math Library

The R language is largely implemented in C. It is possible to create a code library that contains the C implementation of the standard functions that are used in C.

Step 1 Building the library on your computer: please refer to
https://cran.r-project.org/doc/manuals/r-devel/R-admin.html#The-standalone-Rmath-library
This library provides mathematical, probability, and random-number functions that used in R.

How to use R in C/C++: R Standalone Math Library

Step 2 Write C/C++ code:

- Include header file:

```
1 #include <Rmath.h>
```

- Include R functions in your code. The function prototypes have exactly the same arguments in the same order as their analogs that are available in an R session. All arguments must be provided!

```
1 // R function: dnorm <- function (x, mean = 0,
      sd = 1, log = FALSE)
2 // In C/C++:
3 double dnorm (double, double, double, int)
```

How to use R in C/C++: R Standalone Math Library

Step 3 Compile, link and execute:

```
$ g++ -c ex4.cpp -DMATHLIB_STANDALONE -I`R RHOME`/
    include
$ g++ -o ex4 ex4.o -L`R RHOME`/lib -lRmath
$ ./ex4
```

How to use GNU Scientific Library

The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It offers a very comprehensive collection of rigorously developed and tested functions for applied scientific computing under a widely-used and well-understood Open Source license. This has lead to widespread deployment of GSL among a number of disciplines.

Step 1 Install GSL. http://www.gnu.org/software/gsl/.
Mac users: `MacPorts`

```
1 $ sudo port install gsl
```

How to use GNU Scientific Library

Step 2 Write C/C++ code. GSL Reference Manual:
http://www.gnu.org/software/gsl/manual/gsl-ref.pdf.
Two of the most important data types in GSL:

```
typedef struct
{
  size_t size;
  size_t stride;
  double * data;
  gsl_block * block;
  int owner;
} gsl_vector;
```

```
typedef struct
{
  size_t size1;
  size_t size2;
  size_t tda;
  double * data;
  gsl_block * block;
  int owner;
} gsl_matrix;
```

To find out what header files to include, which function to use, how to use a specific function, check the reference manual!

How to use GNU Scientific Library

Step 3 Compile, link and execute.

```
$ gcc -c HW1_C.c -I/opt/local/include
$ gcc HW1_C.o -L/opt/local/lib -lgsl -lgslcblas -o
    aaa
$ ./aaa
```

Example: `HW1_C.c`

RcppGSL

The RcppGSL package provides an easy-to-use interface between GSL and R, with a particular focus on matrix and vector data structures. RcppGSL relies on Rcpp.

RcppGSL Data Types: Vectors

GSL vector	RcppGSL
gsl_vector	RcppGSL::vector<double> or RcppGSL::Vector
gsl_vector_int	RcppGSL::vector<int> or RcppGSL::IntVector
gsl_vector_float	RcppGSL::vector<float>
gsl_vector_long	RcppGSL::vector<long>
gsl_vector_char	RcppGSL::vector<char>
gsl_vector_complex	RcppGSL::vector<gsl_complex>
gsl_vector_complex_float	RcppGSL::vector<gsl_complex_float>
gsl_vector_complex_long_double	RcppGSL::vector<gsl_complex_long_double>
gsl_vector_long_double	RcppGSL::vector<long double>
gsl_vector_short	RcppGSL::vector<short>
gsl_vector_uchar	RcppGSL::vector<unsigned char>
gsl_vector_uint	RcppGSL::vector<unsigned int>
gsl_vector_ushort	RcppGSL::vector<insigned short>
gsl_vector_ulong	RcppGSL::vector<unsigned long>

Table: Correspondance between GSL vector types and templates defined in RcppGSL.

RcppGSL Data Types: Vectors

```
1  int i;
2  RcppGSL::vector<double> v(3); // allocate a gsl_vector
3
4  for (i = 0; i < 3; i++) { // fill the vector
5      v[i] = 1.23 + i;
6  }
7
8  double sum = 0.0; // access elements
9  for (i = 0; i < 3; i++) {
10     sum += v[i];
11 }
12
13 v.free(); // (optionally) free the memory
14            // also automatic when out of scope
```

RcppGSL Data Types: Matrices

`RcppGSL::matrix` template

- The mapping rule of `RcppGSL::matrix` is the same as the one of `RcppGSL::vector`.
- An example:

```
1 RcppGSL::matrix<int> mat(10,10);// create a matrix
2
3 for (int i=0; i<10: i++) { // fill the diagonal
4     mat(i,i) = i; // no need for setter function
5 }
```

- Methods

`nrow` extracts the number of rows
`ncol` extract the number of columns
`size` extracts the number of elements
`free` releases the memory

RcppGSL Examples

- `colNorm`
- `fastLm`
- `Eigen`
- `Bspline`

How to use C/C++, and GSL on HPC

Step 1 Software installation. HPC scientists have already done that for you!

Step 2 Code writing. Same as what you do on your laptop.

Step 3 Compilation and linking. This step need a little work.

```
$ module load gsl
$ gcc -c -I$HPC_GSL_INC HW1_C.c
$ gcc HW1_C.o -L$HPC_GSL_LIB -lgsl -lgslcblas -lrt
    -lm -o aaa
```

How to find the path we need?

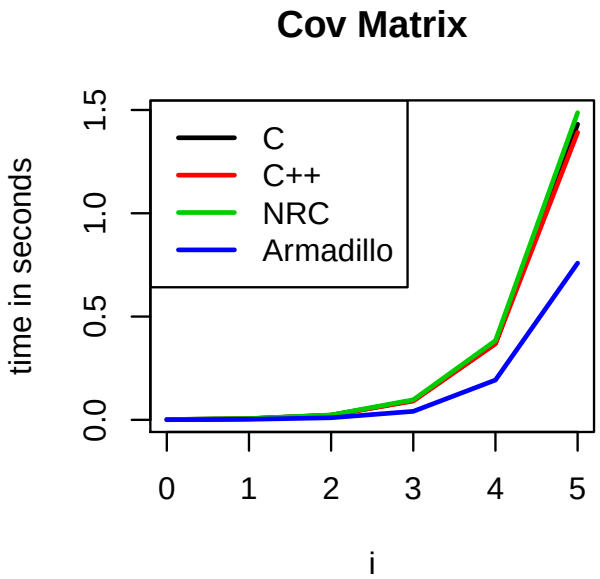
```
$ module spider gsl
```

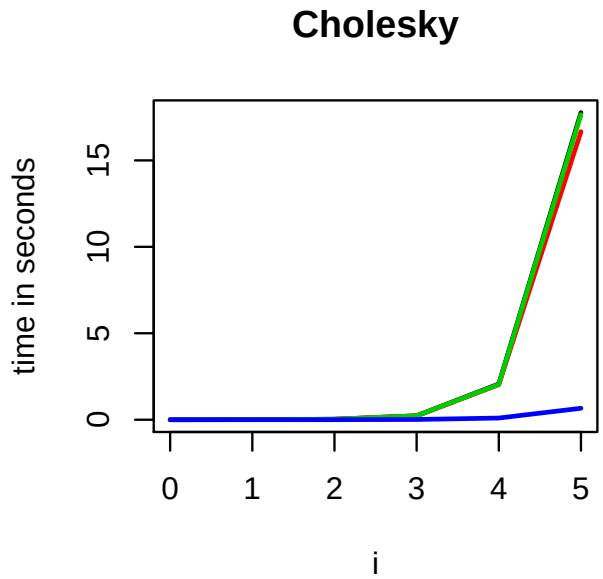
Homework 1 Revisit

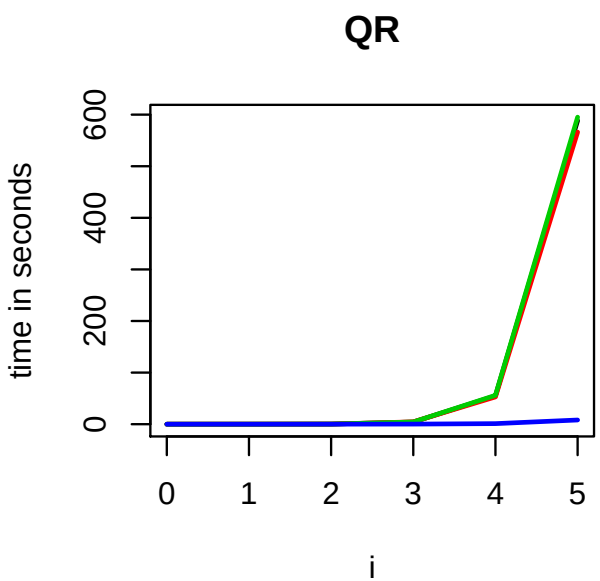
Goal: find out better C/C++ linear algebra libraries.

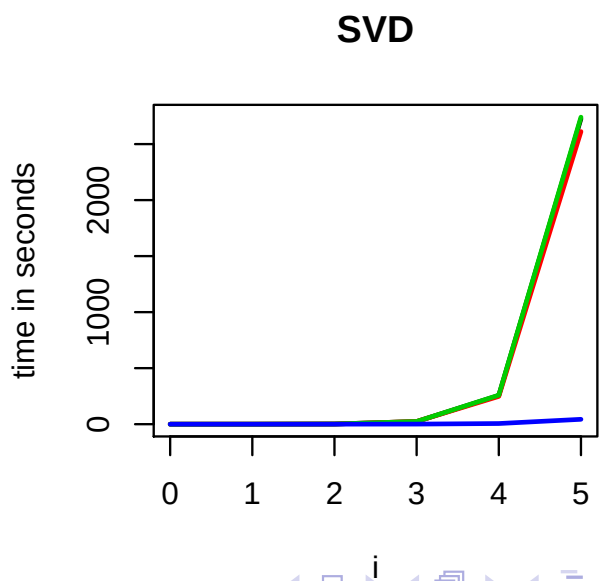
- GSL (C)
- GSL (C++)
- Algorithms provided in Numerical Recipes in C by William et al..
- CLAPACK: Built from Fortran using a Fortran to C conversion utility f2c. (Not working yet!)
- Armadillo (C++): Run on my laptop.
- Eigen (C++): Everything is contained in the header files.

Homework 1 Revistit









Homework 1 Revistit