Biostatistics 615/815 Lecture 13: R packages, and Matrix Library

Hyun Min Kang

October 18th, 2012

Writing an R package

Why write a package?

- Package is a good way to publish your software into the world
- Bundled package can be exposed to public repository, such as the Comprehensive R Archive Network (CRAN).
- >4,000 packages are publicly available at CRAN

Ingredients for making R package

- A set of R functions to include as library
- C++ code for increased efficiency, if available
- Documentation of each function provided (with examples)

Structure of a simple R package `logFET`

- `logFET/DESCRIPTION` : Basic description of the package
- `logFET/NAMESPACE` : Names of public functions to use as library
- `logFET/R/logFET.R` : R wrapper of log Fisher's exact test
- `logFET/src/RlogFET.cpp` : C++ implementation of fast Fisher's exact test
- `logFET/man/logFET.Rd` : Documentation of logFET function

logFET/DESCRIPTION

```
Package: logFET
Version: 0.0.1
Date: 2012-10-18
Title: Example package for BIOSTAT615/816 at U Michigan
Author: Hyun Min Kang
Maintainer: Hyun Min Kang <hmkang@umich.edu>
Depends: R (>= 2.15.0)
Description: Simple version of fisher's exact test
License: GPL (>= 2)
URL: http://goo.gl/9DoFo
```

logFET/NAMESPACE

```
export(logFET)
useDynLib(logFET)
```

logFET/R/logFET.R

```
logFET <- function(a, b, c, d) {
  .Call("fastLogFET",a,b,c,d) ## calls a C++ function
}
```

logFET/man/logFET.Rd

```
\name{logFET}
\alias{logFET}
\title{Fisher's Exact Test returning log10 p-values}
\description{ Compute log10(p-value) for two-sided Fisher's exact test }
\usage{ logFET (a, b, c, d) }
\arguments{
  \item{a}{The first cell count in the 2x2 contingency table}
  \item{b}{The second cell count in the 2x2 contingency table}
  \item{c}{The third cell count in the 2x2 contingency table}
  \item{d}{The last cell count in the 2x2 contingency table}
}
\details{
  All the input arguments are assumed to be integers. Exceptions are not handled.
}
\value{ log10(p-value) of the two-sided Fisher's exact test }
\author{Hyun Min Kang \email{hmkang@umich.edu}}
\examples{
  logFET(2,7,8,2)  ## compute Fisher's exact p-value for (2,7)/(8,2)
}
```

logFET/src/RlogFET.cpp

```
#include <R.h>
#include <Rinternals.h>
#include <Rdefines.h>
#include <cmath>
extern "C" {
  double logFac(int n) {
    double ret;
    for(ret=0.; n > 0; --n) { ret += log((double)n); }
    return ret;
  }

  double logHypergeometricProb(double* logFacs, int a, int b, int c, int d) {
    return logFacs[a+b] + logFacs[c+d] + logFacs[a+c] + logFacs[b+d] - logFacs[a]
           - logFacs[b] - logFacs[c] - logFacs[d] - logFacs[a+b+c+d];
  }

  void initLogFacs(double* logFacs, int n) {
    logFacs[0] = 0;
    for(int i=1; i < n+1; ++i) {
      logFacs[i] = logFacs[i-1] + log((double)i);
    }
  }
```

logFET/src/RlogFET.cpp (cont'd)

```
double logFishersExactTest(int a, int b, int c, int d) {
  int n = a + b + c + d;

  double* logFacs = new double[n+1]; // dynamically allocate memory
  initLogFacs(logFacs, n);

  double logpCutoff = logHypergeometricProb(logFacs,a,b,c,d);
  double pFraction = 0;
  for(int x=0; x <= n; ++x) { // among all possible x
    if ( a+b-x >= 0 && a+c-x >= 0 && d-a+x >=0 ) { // consider valid x
      double l = logHypergeometricProb(logFacs,x,a+b-x,a+c-x,d-a+x);
      if ( l <= logpCutoff ) pFraction += exp(l - logpCutoff);
    }
  }
  double logpValue = logpCutoff + log(pFraction);

  delete [] logFacs;
  return (logpValue/log(10.));
}
```

logFET/src/RlogFET.cpp (cont'd)

```
  SEXP fastLogFET(SEXP a, SEXP b, SEXP c, SEXP d) {
    SEXP out;

    PROTECT(a = AS_NUMERIC(a));
    PROTECT(b = AS_NUMERIC(b));
    PROTECT(c = AS_NUMERIC(c));
    PROTECT(d = AS_NUMERIC(d));

    PROTECT( out = allocVector(REALSXP,1) );

    REAL(out)[0] = logFishersExactTest((int)(NUMERIC_POINTER(a)[0]),
                     (int)(NUMERIC_POINTER(b)[0]),
                     (int)(NUMERIC_POINTER(c)[0]),
                     (int)(NUMERIC_POINTER(d)[0]));
    UNPROTECT(5);

    return (out);
  }
};
```

Building an R package

Copying from instructor's public repository

```
$ cp -R ~hmkang/Public/615/Rpkg/logFET .
```

Building your package

```
$ R CMD build logFET
* checking for file 'logFET/DESCRIPTION' ... OK
* preparing 'logFET':
* checking DESCRIPTION meta-information ... OK
* cleaning src
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building 'logFET_0.0.1.tar.gz'
```

Installing

If you have a root permission

```
$ (sudo) R CMD INSTALL logFET_0.0.1.tar.gz
```

In scs.itd.umich.edu

```
$ R
> install.packages("logFET_0.0.1.tar.gz")
Installing package(s) into '/afs/umich.edu/user/h/m/hmkang/R/x86_64-unknown-linux-gnu-librar
(as 'lib' is unspecified)
inferring 'repos = NULL' from the file name
* installing *source* package 'logFET' ...
** libs
g++ -I/usr/local/R-2.15/lib64/R/include -DNDEBUG  -I/usr/local/include    -fpic  -g -O2
-c RlogFET.cpp -o RlogFET.o
g++ -shared -L/usr/local/lib64 -o logFET.so RlogFET.o
installing to /afs/umich.edu/user/h/m/hmkang/R/x86_64-unknown-linux-gnu-library/2.15/logFET/
** R
** preparing package for lazy loading
### (omitted)
* DONE (logFET)
```

Using logFET package

```
$ R
> library(logFET)
> logFET(2,7,8,2)
[1] -1.638005
> logFET(2000,7000,8000,2000)
[1] -1466.131
> fisher.test(matrix(c(2000,7000,8000,2000),2,2))$p.value
[1] 0
```

Programming with Matrix

Why Matrix matters?

- Many statistical models can be well represented as matrix operations
  - Linear regression
  - Logistic regression
  - Mixed models
- Efficient matrix computation can make difference in the practicality of a statistical method
- Understanding C++ implementation of matrix operation can expedite the efficiency by orders of magnitude

Ways for Matrix programming in C++

- Implementing Matrix libraries on your own
  - Implementation can well fit to specific need
  - Need to pay for implementation overhead
  - Computational efficiency may not be excellent for large matrices

Ways for Matrix programming in C++

- Implementing Matrix libraries on your own
  - Implementation can well fit to specific need
  - Need to pay for implementation overhead
  - Computational efficiency may not be excellent for large matrices
- Using BLAS/LAPACK library
  - Low-level Fortran/C API
  - ATLAS implementation for gcc, MKL library for intel compiler (with multithread support)
  - Used in many statistical packages including R
  - Not user-friendly interface use.
  - `boost` supports C++ interface for BLAS

Ways for Matrix programming in C++

- Implementing Matrix libraries on your own
  - Implementation can well fit to specific need
  - Need to pay for implementation overhead
  - Computational efficiency may not be excellent for large matrices
- Using BLAS/LAPACK library
  - Low-level Fortran/C API
  - ATLAS implementation for gcc, MKL library for intel compiler (with multithread support)
  - Used in many statistical packages including R
  - Not user-friendly interface use.
  - `boost` supports C++ interface for BLAS
- Using a third-party library, `Eigen` package
  - A convenient C++ interface
  - Reasonably fast performance
  - Supports most functions BLAS/LAPACK provides

Using a third party library

Downloading and installing `Eigen` package

- Download at *http://eigen.tuxfamily.org/*
- To install - just uncompress it, no need to build

Using a third party library

Downloading and installing `Eigen` package

- Download at *http://eigen.tuxfamily.org/*
- To install - just uncompress it, no need to build

Using `Eigen` package

- Add `-I ~hmkang/Public/include` option (or include directory containing `Eigen/`) when compile
- No need to install separate library. Including header files is sufficient

Example usages of Eigen library

```
#include <iostream>
#include <Eigen/Dense> // For non-sparse matrix
using namespace Eigen; // avoid using Eigen::
int main()
{
  Matrix2d a;          // 2x2 matrix type is defined for convenience
  a << 1, 2,
       3, 4;
  MatrixXd b(2,2);     // but you can define the type from arbitrary-size matrix
  b << 2, 3,
       1, 4;
  std::cout << "a + b =\n" << a + b << std::endl;  // matrix addition
  std::cout << "a - b =\n" << a - b << std::endl;  // matrix subtraction
  std::cout << "Doing a += b;" << std::endl;
  a += b;
  std::cout << "Now a =\n" << a << std::endl;
  Vector3d v(1,2,3);                               // vector operations
  Vector3d w(1,0,0);
  std::cout << "-v + w - v =\n" << -v + w - v << std::endl;
}
```

More examples

```
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;
int main()
{
  Matrix2d mat;              // 2*2 matrix
  mat << 1, 2,
         3, 4;
  Vector2d u(-1,1), v(2,0);  // 2D vector
  std::cout << "Here is mat*mat:\n" << mat*mat << std::endl;
  std::cout << "Here is mat*u:\n" << mat*u << std::endl;
  std::cout << "Here is u^T*mat:\n" << u.transpose()*mat << std::endl;
  std::cout << "Here is u^T*v:\n" << u.transpose()*v << std::endl;
  std::cout << "Here is u*v^T:\n" << u*v.transpose() << std::endl;
  std::cout << "Let's multiply mat by itself" << std::endl;
  mat = mat*mat;
  std::cout << "Now mat is mat:\n" << mat << std::endl;
  return 0;
}
```

More examples

```
#include <Eigen/Dense>
#include <iostream>
using namespace Eigen;
int main()
{
  MatrixXd m(2,2), n(2,2);
  MatrixXd result(2,2);
  m << 1,2,
       3,4;
  n << 5,6,7,8;
  result = m * n;
  std::cout << "-- Matrix m*n: --" << std::endl << result << std::endl << std::endl;
  result = m.array() * n.array();
  std::cout << "-- Array m*n: --" << std::endl << result << std::endl << std::endl;
  result = m.cwiseProduct(n);
  std::cout << "-- With cwiseProduct: --" << std::endl << result << std::endl << std::endl;
  result = (m.array() + 4).matrix() * m;
  std::cout << "-- (m+4)*m: --" << std::endl << result << std::endl << std::endl;
  return 0;
}
```

Time complexity of matrix computation

Square matrix multiplication / inversion

- Naive algorithm : $O(n^3)$
- Strassen algorithm : $O(n^{2.807})$
- Coppersmith-Winograd algorithm : $O(n^{2.376})$ (with very large constant factor)

Determinant

- Laplace expansion : $O(n!)$
- LU decomposition : $O(n^3)$
- Bareiss algorithm : $O(n^3)$
- Fast matrix multiplication algorithm : $O(n^{2.376})$

Computational considerations in matrix operations

Avoiding expensive computation

- Computation of $\mathbf{u}'AB\mathbf{v}$

Computational considerations in matrix operations

Avoiding expensive computation

- Computation of $\mathbf{u}'AB\mathbf{v}$
- If the order is $(((\mathbf{u}'(AB))\mathbf{v})$
  - $O(n^3) + O(n^2) + O(n)$ operations
  - $O(n^2)$ overall

Computational considerations in matrix operations

Avoiding expensive computation

- Computation of $\mathbf{u}'AB\mathbf{v}$
- If the order is $(((\mathbf{u}'(AB))\mathbf{v})$
  - $O(n^3) + O(n^2) + O(n)$ operations
  - $O(n^2)$ overall
- If the order is $(((\mathbf{u}'A)B)\mathbf{v})$
  - Two $O(n^2)$ operations and one $O(n)$ operation
  - $O(n^2)$ overall

Quadratic multiplication

Same time complexity, but one is slightly more efficient

- Computing $\mathbf{x}' A \mathbf{y}$.
- $O(n^2) + O(n)$ if ordered as $(\mathbf{x}' A)\mathbf{y}$.
- Can be simplified as $\sum_i \sum_j x_i A_{ij} y_j$

A symmetric case

- Computing $\mathbf{x}' A \mathbf{x}$ where $A = LL'$
- $\mathbf{u} = L'\mathbf{x}$ can be computed more efficiently than $A\mathbf{x}$.
- $\mathbf{x}' A \mathbf{x} = \mathbf{u}'\mathbf{u}$

Today

R/C++ Interface

- Combining C++ code base with R extension
- C++ implementation more efficiently handles loops and complex algorithms than R
- R is efficient in matrix operation and convenient in data visualization and statistical tools
- R/C++ interface increases your flexibility and efficiency at the same time.

Matrix Library

- `Eigen` library for convenient use and robust performance
- Time complexity of matrix operations