Rcpp简介

温灿红

中国科学技术大学管理学院

大纲

- 初识Rcpp
- Rcpp中的C++数据类型
- Rcpp糖
- Rcpp的拓展

初识Rcpp

为什么需要Rcpp

- 在**R**中进行高性能计算，我们之前已经学过
  - 向量化编程
  - apply族函数
  - 内置函数sum，prod，cumsum，cumprod等
  - 并行计算
- 但仍然有一些计算不能用上述方法进行优化，那么一个折中的办法就是把低效的部分变成成C++代码，即把已经用R 代码完成的程序中运行速度瓶颈部分改写成C++代码，提高运行效率。然后在**R**中调用该C++代码，实现一些高级的功能，如画图、简单的汇总等。

Rcpp

- Rcpp 可以很容易地把C++ 代码与R 程序连接在一起，可以从R 中直接调用C++代码而不需要用户关心那些繁琐的编译、链接、接口问题。
- Rcpp 可以在R 数据类型和C++ 数据类型之间容易地转换。
- Rcpp 支持把C++ 代码写在R 源程序文件内，执行时自动编译连接调用；
- 也支持把C++ 代码保存在单独的源文件中，执行R 程序时自动编译连接调用；
- 对较复杂的问题，应制作R 扩展包，利用构建R扩展包的方法实现C++ 代码的编译连接。

Rcpp的安装

- R中直接执行`install.packages("Rcpp")`。
- 除了要安装Rcpp 包之外，还需要安装编译软件。
  - MS Windows：需要安装RTools 包，这是用于C, C++, Fortran 程序编译链接的开发工具包，是自由软件。 从CRAN网站下载RTools工具包， 链接为(https://cran.r-project.org/bin/windows/Rtools/)。 并将其安装到默认位置“C:\RTools” 中，否则RStudio中使用Rcpp可能会出错。
  - Mac：从应用商店安装Xcode软件
  - Linux： 执行如下命令安装编译软件

    ```
    sudo apt-get install r-base-dev
    ```
- 更多的可见《Rcpp:R和C++的无缝整合》。

测试Rcpp是否正确安装

```
require(Rcpp)
```

```
## 载入需要的程辑包：Rcpp
```

```
cppFunction("double sumC(double x, double y){
  double s;
  s = x + y;
  return s;
} ")
sumC(3.5, 6.5)
```

```
## [1] 10
```

- 如果没有安装成功会弹出窗口提醒你需要安装的工具。

我的第一个Rcpp程序

- 前面用到的cppFunction()函数是在R代码里嵌入了C++函数，这种方式适合C++代码比较短而简单的情形。
- 更好的方式是把程序保存在"*.cpp"文件中，然后调用sourceCpp()编译使用。注意这里的路径里不要含有空格或者中文，否则编译C++程序的时候会报错。

```
library(Rcpp)
sourceCpp("sumCpp.cpp")
sumC(1, 2)
```

```
## [1] 3
```

```
1  #include <Rcpp.h>
2  using namespace Rcpp;
3
4  // [[Rcpp::export]]
5  double sumC(
6      double x,
7      double y
8  ) {
9    double s;
10   s = x + y;
```

头文件

Rcpp识别标记

我的第一个Rcpp程序

- **头文件**。代码#include <Rcpp.h>引入了名为Rcpp的头文件，表明代码需要使用到Rcpp中的模板或库等内容。
- **输出**。C++程序中编写函数文件若想输入到R中进行调用，须在函数前加上//[[Rcpp::export]]这种特殊的注释。
- **函数主体** 和普通的C++文件一样。

```
double sumC(double x,
double y){

body;

return ( s );

}
```

- double指定返回值的数据类型
- function(parameter) 函数定义方式，括号内是形参
- return (s)返回值
- 每行代码的末尾紧跟着分号";"

在Rmd文件中使用C++文件

- 在Rmd文件中， 可以使用特殊的Rcpp代码块， 其中包含C++源代码， 可以直接起到对其调用sourceCpp()的作用。
- 如前面Rcpp代码块为{Rcpp firstChunk}，则在调用需要指定代码块的方式为{rcallFirstChunkInR}。

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

// [[Rcpp::export]]
double sumC(
    double x,
    double y
) {
  double s;
  s = x + y;
  return s;
}
```

```
sumC(1, 2)
```

```
## [1] 3
```

R和C++的输入差异

```
sumC(3L, 6L)
```

```
## [1] 9
```

```
sumC(TRUE, FALSE)
```

```
## [1] 1
```

```
sumC(c(3,4), c(5,6))
```

```
## Error in eval(expr, envir, enclos): Expecting a single value: [extent=2].
```

- C++中的double是标量，如果输入向量的话就会报错。
- 下面我们来讲述Rcpp中的C++数据类型，来跟R进行无缝连接。

Rcpp中的C++数据类型

R与C++的类型转换

- R程序与由Rcpp支持的C++程序之间需要传递数据， 就需要将R的数据类型经过转换后传递给C++函数， 将C++函数的结果经过转换后传递给R。
- 常见的C++中的数据类型有： `int`, `double`, `float`, `bool`, `char`, `void`等。
- 用`wrap()`把C++变量返回到R中。当C++中赋值运算的右侧表达式是一个R对象或R对象的部分内容时， 可以隐含地调用as()将其转换成左侧的C++类型。
- 用`as()`函数把R变量转换为C++类型。当C++中赋值运算的左侧表达式是一个R对象或其部分内容时， 可以隐含地调用wrap()将右侧的C++类型转换成R类型。

Rcpp中的C++数据类型

- Rcpp包为C++定义了NumericVector, CharacterVector, Matrix等新数据类型，可以直接与R的numeric, charactor, matrix对应。对应关系如下：

Rcpp	R	说明
IntegerVector	c(...) or vector(mode = "integer", ...)	整数向量
NumericVector	c(...) or vector(mode = "double", ...)	数值向量
LogicalVector	c(...) or vector(mode = "bool", ...)	逻辑型向量
CharacterVector	c(...) or vector(mode = "character", ...)	字符串向量
IntegerMatrix	matrix()	整数矩阵
NumericMatrix	matrix()	数值矩阵
CharacterMatrix	matrix()	字符串矩阵
DataFrame	data.frame()	数据框
List	list()	列表

IntegerVector类

- 在R中通常不严格区分整数与浮点实数， 但是在与C++交互时，C++对整数与实数严格区分， 所以RCpp中整数向量与数值向量是区分的。
- 在R中，如果定义了一个仅有整数的向量， 其类型是整数(integer)的，否则是数值型(numeric)的，如：

```
x <- 1:5
class(x)
```

```
## [1] "integer"
```

```
y <- c(0, 0.5, 1)
class(y)
```

```
## [1] "numeric"
```

IntegerVector类

- 用C++编写函数，从R中输入整数向量， 计算其元素乘积（与R的prod()函数功能类似）。

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
IntegerVector prod1(IntegerVector x){
    int prod = 1;
    for(int i=0; i < x.size(); i++){
        prod *= x[i];
    }
    return wrap(prod);
}
')
print(prod1(1:5))
```

```
## [1] 120
```

- x.size()返回的是元素的个数。

NumericVector类

- NumericVector类在C++中保存双精度型一维数组， 可以与R的实数型向量(class为numeric)相互转换。这是最常用到的数据类型。

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
double ssp(
    NumericVector vec, double p){
  double sum = 0.0;
  for(int i=0; i < vec.size(); i++){
    sum += pow(vec[i], p);
  }
  return sum;
}
')
ssp((1:4)/10, 2.2)
```

```
## [1] 0.2392496
```

```
sum( ((1:4)/10)^2.2 )
```

```
## [1] 0.2392496
```

LogicalVector类

- LogicalVector 类可以存储C++值true, false, 还可以保存缺失值NA_REAL, R_NaN, R_PosInf, 但是这些不同的缺失值转换到R中都变成NA。如：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
LogicalVector f(){
  LogicalVector x(5);
  x[0] = false; x[1] = true;
  x[2] = NA_REAL;
  x[3] = R_NaN; x[4] = R_PosInf;
  return(x);
}
')
f()
```

```
## [1] FALSE  TRUE    NA    NA    NA
```

```
## [1] FALSE  TRUE    NA    NA    NA
identical(f(), c(FALSE, TRUE, rep(NA,3)))
```

```
## [1] TRUE
```

NumericMatrix类

- NumericMatrix x(n,m)产生一个未初始化的矩阵， 元素为double类型。
- x.nrow()返回行数， x.ncol()返回列数， x.size()返回元素个数。
- 下标从0开始计算。
  - x(i,j)访问x的第i行第j列
  - x.row(i)或x(i,_)访问x的第i行
  - x.column(j)或x(_,j)访问x的第j列。
- NumericMatrix::zeros(n)返回n阶元素为0的方阵。 NumericMatrix::eye(n)返回n阶单位阵。

NumericMatrix类的例子：计算各列模的最大值

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
double testfun(NumericMatrix x){
  int nr = x.nrow();
  int nc = x.ncol();
  double y, y1;
  y = 0.0;
  NumericVector ycol(nr);
  for(int j=0; j<nc; j++){
    ycol = x.column(j);
    y1 = sum(ycol * ycol);
    if(y1 > y) y = y1;
  }
  y = sqrt(y);
  return y;
}
')
x <- matrix(c(1, 2, 4, 9), 2, 2)
print(testfun(x))
```

```
## [1] 9.848858
```

List类

- Rcpp提供的List类型对应于R的list(列表)类型。
- 其元素可以不是同一类型， 在C++中可以用方括号和字符串下标的格式访问其元素。

```
// 产生列表
List L = List::create(e1, e2);

// 产生带元素名字的列表
List L = List::create(Named("name1") e1, Named("name2") e2);

// 按名字提取元素
NumericVector vec = as<NumericVector>(x["vec"]);
```

DataFrame类

- Rcpp的DataFrame类用来与R的data.frame交换信息。

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
DataFrame fdf(){
  IntegerVector vec =
    IntegerVector::create(7,8,9);
  std::vector<std::string> s(3);
  s[0] = "abc"; s[1] = "ABC"; s[2] = "123";
  return(DataFrame::create(
    Named("x") = vec,
    Named("s") = s));
}
')
fdf()
```

```
##   x   s
## 1 7 abc
## 2 8 ABC
## 3 9 123
```

案例学习：接受拒绝法产生beta分布的随机数

假设 $f(x) = b(x(1-x))^a$ 为beta分布的概率密度函数，下面我们通过均匀分布来产生来自于beta分布的随机数。

算法流程：

1. 生成来自于 $U(0,1)$的随机变量 U 和 V；
2. 如果 $V \leq 4^a(U(1-U))^a$，则令 $X = U$，否则直接返回第1步。

案例学习：接受拒绝法产生beta分布的随机数

```
rbetaR <- function(n, a)
{
  flag <- 0
  x <- numeric(n)
  while(flag < n)
    {
      u <- runif(1)
      v <- runif(1)
      if(v <= 4^a*(u*(1-u))^a)
      {
      flag <- flag + 1
      x[flag] <- u
      }
    }
x
}
data <- rbetaR(1000, a=1)
```

案例学习：接受拒绝法产生beta分布的随机数

```
library(Rcpp)
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector rbetaRcpp(const int N, double a){
  int flag = 0;
  NumericVector x(N);
  while(flag < N){
      double u = R::runif(0,1);
      double v = R::runif(0,1);
      double f = pow(4.0,a)*pow(u*(1-u),a);
      if(v <= f){
        flag = flag + 1;
        x[flag] = u;
      }
    }
  return x;
}
')
data <- rbetaRcpp(1000, a=1)
```

案例学习：接受拒绝法产生beta分布的随机数

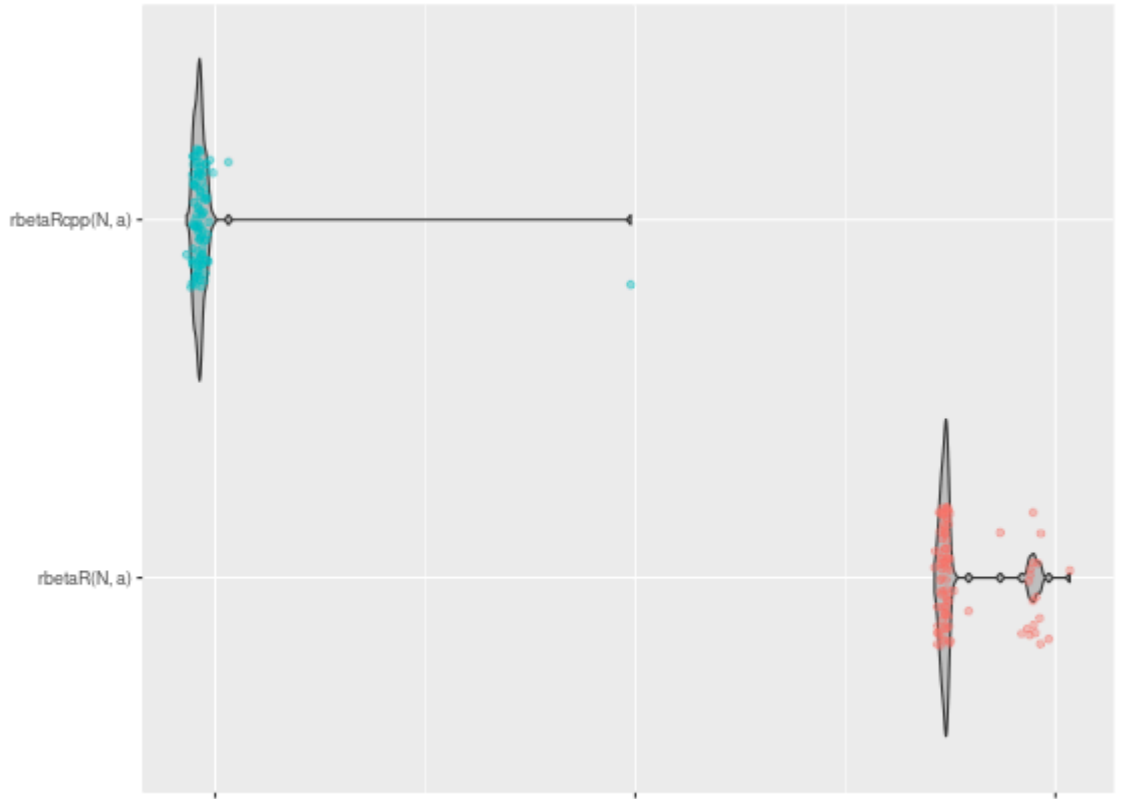
```
library(microbenchmark)
N <- 1000
a <- 1
res <- microbenchmark(rbetaR(N, a), rbetaRcpp(N, a))

print(res)
```

```
## Unit: microseconds
##             expr      min        lq      mean    median        uq       max
##     rbetaR(N, a) 5153.215 5378.4925 6107.8623 5485.1330 5589.6820 10816.556
##  rbetaRcpp(N, a)   85.503   89.9905  100.7257   91.6975   93.1285   975.155
##  neval cld
##    100  a
##    100   b
```

案例学习：接受拒绝法产生beta分布的随机数

```
library(ggplot2)
autoplot(res) +
  geom_jitter(position = position_jitter(0.2, 0), aes(color = expr), alpha = 0.4) +
  aes(fill = I("gray")) + theme(legend.position = "none")
```



Rcpp糖

Rcpp糖

- 在C++中，向量和矩阵的运算通常需要逐个元素进行， 或者调用相应的函数。
- Rcpp通过C++的表达式模板功能和惰性求值技术， 可以在C++中写出像R中对向量和矩阵运算那样的表达式， 这称为Rcpp糖(sugar)。
- R中的很多函数如`sin`等是向量化的， Rcpp糖也提供了这样的功能。 Rcpp糖提供了一些向量化的函数如`ifelse`, `sapply`等。

一个简单的例子（一）

二元函数定义如下：

$$f(x,y) = \begin{cases} x^2 & \text{if } x < y, \\ -y^2 & \text{if } x \geq y \end{cases}$$

- R版本：

```
fooR <- function(x, y){
  ifelse(x<y, x*x, -(y*y))
}
```

一个简单的例子（二）

- 普通C++版本：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
NumericVector fooC(NumericVector x, NumericVector y){
  int n = x.size();
  NumericVector res(n);
  for (int i=0; i < n; i++){
    if(x[i] < y[i]){
      res[i] = x[i] * x[i];
    } else{
      res[i] = - ( y[i]*y[i]);
    }
  }
  return res;
}
')
fooC(c(1, 3), c(4,2))
```

```
## [1]  1 -4
```

一个简单的例子（三）

- Rcpp糖版本：

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
  NumericVector fooRcpp(NumericVector x, NumericVector y){
  return ifelse(x < y, x*x, -(y*y));
}
')
fooRcpp(c(1, 3), c(4,2))
```

```
## [1]  1 -4
```

- 通过向量化运算大大简化了程序

Rcpp糖中的运算符

- Rcpp糖中的运算符支持向量化运算，包括：
  - 二元算术运算符， 如+，－ ，* /及其各种混合。
  - 二元逻辑运算符，如<，>，<=，>=，==，!=。
  - 一元运算符，如-，!。

```
NumericVector res = x * y + y / 2.0;
NumericVector res = x * (y - 2.0);
NumericVector res = x / (y * y);

LogicalVector res = (x + y) < (x * x);

NumericVector res = -x;

LogicalVector res = ! ( y < z );
```

Rcpp糖中的函数

- Rcpp糖中定义了很多函数，非常紧密地匹配同名的R函数。
- 逻辑判断函数： any, all
- 数学函数abs(), exp(), floor(), ceil(), pow()等
- 产生糖表达式的函数：is_na(), seq_along(), seq_len(), pmax(), pmin(), ifelse()
- apply族函数： sapply(), lapply(), mapply()
- 集合运算函数： setdiff(), union_(), intersect(), unique(), sort(), setequal()
- 统计函数： dnorm(), pnorm(), qnorm(), rnorm()等

例子：使用Rcpp糖计算 π（一）

- 在几何概型里，我们可以通过计算单位圆内的和外面的正方形的面积之比，来得到 π 的近似计算。
- R语言版本

```
piR <- function(N){
  x <- runif(N)
  y <- runif(N)
  d <- sqrt(x^2 + y^2)
  return(4.0 * sum(d<1.0) / N)
}
```

例子：使用Rcpp糖计算 π（二）

- Rcpp糖版本

```
sourceCpp(code='
#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]
double piSugar(const int N){
  RNGScope scope; //确保随机数生成器的合适设置
  NumericVector x = runif(N);
  NumericVector y = runif(N);
  NumericVector d = sqrt(x*x + y*y);
  return 4.0 * sum(d<1.0) / N;
}
')
```

例子：使用Rcpp糖计算 π（三）

```
N <- 1e6

set.seed(1)
resR <- piR(N)


set.seed(1)
resCpp <- piSugar(N)

stopifnot(identical(resR, resCpp))

res <- microbenchmark(piR(N), piSugar(N))

print(res)
```

```
## Unit: milliseconds
##         expr      min       lq     mean   median       uq      max neval cld
##       piR(N) 50.86010 53.59172 67.37965 55.72113 64.77073 227.0972   100  a
##  piSugar(N) 17.47717 18.44450 25.45716 20.72984 26.79492 193.1086   100   b
```

基于Rcpp创建R包

基于Rcpp创建R包

- 选择R包的类型为：R package using Rcpp，基于步骤跟不同的R包基本一样。
- 创建一个新的**C++**文件，它包括一个基本的函数和一些有关开始的说明，保存在`src/`目录下（如没有，可自行创建）。

一个简单的例子

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

// This is a simple example of exporting a C++ function to R. You can
// source this function into an R session using the Rcpp::sourceCpp
// function (or via the Source button on the editor toolbar).

// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
  return x * 2;
}

// You can include R code blocks in C++ files processed with sourceCpp
// (useful for testing and development). The R code will be automatically
// run after the compilation.
//

/*** R
timesTwo(42)
/
```

一个简单的例子

- 包含了两个重要的部分：头文件#include和特别属性// [[Rcpp::export]].
- 对于每一个输出的C++函数，都会自动生成一个R封装函数(位于R/RcppExports.R).。如上面的timesTwo() 看起来是这样子的：

```
timesTwo <- function(x) {
    .Call(`_democpp_timesTwo`, x)
}

timesTwo(1)
timesTwo(10)
```

编译过代码的帮助文档（一）

- 同样可以使用roxygen2 来产生帮助文档，只是这里需要将 #’替换成 //’。

```
//' Multiply a number by two
//'
//' @param x A single interger.
//' @export
// [[Rcpp::export]]
NumericVector timesTwo(NumericVector x) {
  return x * 2;
}
```

- R将会自动产生 R/RcppExports.R文档，如下:

```
#' Multiply a number by two
#'
#' @param x A single interger.
#' @export
timesTwo <- function(x) {
    .Call(`_democpp_timesTwo`, x)
}
```

编译过代码的帮助文档（二）

- .Rd 文档如下：

```
% Generated by roxygen2: do not edit by hand
% Please edit documentation in R/RcppExports.R
\name{timesTwo}
\alias{timesTwo}
\title{Multiply a number by two}
\usage{
timesTwo(x)
}
\arguments{
\item{x}{A single interger.}
}
\description{
Multiply a number by two
}
```

Rcpp的拓展包

Rcpp的拓展

- Rcpp 拓展有四个核心的包：
    1. RcppArmadillo：使得线性代数的引入语法更加接近matlab
    2. RcppEigen： 优化过的线性代数运算
    3. RInnside： 实现在C++中调用R程序
    4. RcppParallel： 基于Rcpp实现并行计算
- 简单的方式：函数`cppFunction()`可以用来在R中调用C++函数。
- 当C++程序代码较多时，直接用`cppFunction()`来调用不太合适，将其单独保存成 .cpp文件并使用`sourceCpp()`加载调用更为灵活方便。

RcppArmadillo和RcppEigen

- **RcppArmadillo**和**RcppEigen**是**Rcpp**中两大科学计算包。
- **RcppArmadillo**扩展包提供了**Armadillo**库的接口。
  - **Armadillo**是一个着力于线性代数及其相关运算的现代C++库，致力于在和脚本语言一样具有表达力的同时，又通过使用包括模板宏编程在内的现代C++设计来提供高效的代码。
- **RcppEigen**扩展包提供了**Eigen**库的接口。
  - **Eigen**是一个使用了现代模板元编程技术的C++库，与**Armadillo**类似，但提供了粒度更细的应用编程接口。
  - 尤其擅长关于矩阵和向量的各种计算和分解。