

Rcpp 快速入门

黄金山

中国科学技术大学统计与金融系

Why and How

- R 是解释性的语言, 所以运行速度比较慢, 尤其是显示循环.
- 尽量用 R 提供的向量化运算策略, 使用 apply 系列函数.
- 可以通过借助其他编程语言 C, C++, Fortran 来提高程序的运行效率. 由于 R 本身就是用 C, Fortran 写的, 所以 R 提供 C, C++, Fortran 的各种 API.
- R 提供的 C API 对一般人而言很难上手, 而且操作复杂. 幸运的是 Rcpp 为我们提供了很方便简单和高效的 C++ API.
- 你需要一个编译器, 推荐使用 gcc/g++, windows 用户可以通过安装 Rtools 获得, Linux 一般自带.
- 你还需要安装 Rcpp 和 inline 包.

Rcpp class	R typeof
Integer(Vector Matrix)	integer vectors and matrices
Numeric(Vector Matrix)	numeric ...
Logical(Vector Matrix)	logical ...
Character(Vector Matrix)	character ...
Complex(Vector Matrix)	complex ...
List	list (aka generic vectors) ...
Expression(Vector Matrix)	expression ...
Environment	environment
Function	function
XPtr	externalptr
Language	language
S4	S4
...	...

RObject 简介

- Rcpp 中的基本类型是 RObject, 有点像 C 接口中的 SEXP (在标准的 C 接口中所有的 R 对象都是 SEXP), RObject 实际上就是 SEXP 的一个 wrapper.
- 上面的 Rcpp class 全都是 RObject 的子类, Rcpp 为 RObject 和它的子类定义了许多方法, 使它们能够更加方便的使用.

<http://dirk.eddelbuettel.com/code/rcpp/html/> 上看到所有这些类的定义和方法, 下面是个例子:

```
> require(inline)
> robj.src <- '
RObject xnumvec(x);
CharacterVector attrname =
    wrap(xnumvec.attributeNames());
CharacterVector xnames = xnumvec.attr("names");

bool iss4 = xnumvec.isS4();
return List::create(_["x"] = xnumvec,
                   _["xattrname"]=attrname,
                   _["names"] = xnames,
                   _["iss4"] = iss4);
'
> robj.test <- cxxfunction(signature(x = "numeric"),
+                           robj.src, plugin = "Rcpp")
> x = c(a=3,b=4)
```

```
> robject::test(x)
```

```
$x
```

```
a b
```

```
3 4
```

```
$xattrname
```

```
[1] "names"
```

```
$names
```

```
[1] "a" "b"
```

```
$iss4
```

```
[1] FALSE
```

```
> attributes(x)
```

```
$names
```

```
[1] "a" "b"
```

关于 Rcpp 各种数据 (RObject 的子类) 的处理, 下面是一个更为详细的例子:

```
> require(inline)
> mix.src <- '
NumericMatrix nummat(rnumat);
DateVector datevec(dvec);
DatetimeVector dtvec(dt);
DataFrame df(DF);
List lst(2);
lst[0] = nummat(Range(1,2), Range(1,2));
// nummat[_,1] means the second column
lst[1] = nummat(_,1);
lst.push_back(datevec);
lst.push_front(dtvec);
// df[1] or df["b"], append with name "df$b"
lst.push_back(df[1], "df$b");
return lst;
'
```

```
> mix.test <- cxxfunction(signature(rnumat = "numeric",
+                                 dvec = "Date",
+                                 dt = "POSIXct",
+                                 DF = "dataframe"),
+                          mix.src, plugin = "Rcpp")
> rummat <- matrix(rnorm(3*3), 3, 3)
> dvec <- as.Date(1:2, origin = "1970-01-01")
> dtvec <- Sys.time() + 1:2
> DF <- data.frame(a = 1:2 , b = 4:5)
```



```
> mix.test(rummat, dvec, dtvec, DF)

[[1]]
[1] "2012-05-05 17:32:28 CST" "2012-05-05 17:32:29 CST"

[[2]]
      [,1] [,2]
[1,] -0.1569 0.4002
[2,]  1.9407 0.2976

[[3]]
[1] -0.4446 -0.1569  1.9407

[[4]]
[1] "1970-01-02" "1970-01-03"

$`df$b`
[1] 4 5
```

- 需要注意的是在 C++ 下标是从 0 开始. 一般的向量 (Vector, List or Dataframe) 可以用 [] 来做下标运算, 下标可以是整数 (int) 也可以是字符串 (std:string) 可以说和 R 中的 [] 下标运算完全一致.
- 矩阵运算的下标要用 (), 但是对于矩阵 (NumericMatrix) Rcpp 并不提供 R 中类似的各种运算, 如果有需要, 可以参考 RcppArmadillo package 或者是 RcppEigen package.
- Rcpp 的帮助文档 *Rcpp-quickref.pdf* 提供了关于各种数据类型操作的简要说明, 相当于一个 reference card, 便于速查.

下面提供一个 RcppArmadillo 的例子, 你需要安装这个包.

RcppArmadillo

```
> require(inline)
> arma.src <- '
arma::mat Am = as< arma::mat >(A);
arma::mat Bm = as< arma::mat >(B);
arma::mat Cm = arma::trans(Am - Bm);
NumericVector cv(Cm.begin(), Cm.end());
return cv;
'
> arma.test <- cxxfunction(signature(A = "numeric",
+                                   B = "numeric"),
+                           arma.src,
+                           plugin = "RcppArmadillo")
> A = matrix(1:12, 3, 4)
> B = matrix(2:13, 3, 4)
> arma.test(A, B)

[1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

上面一节主要讲了如何用 Rcpp 提供的类处理 R 中的数据结构. 为了能够利用 C++ 中的数据和现成算法, 需要在 R 的数据类型与 C++ 的数据类型做转换.

as 的作用将 R 中的数据类型转换成 C++ 中的数据类型, 它是一个模板函数:

```
// conversion from R to C++  
template <typename T> T as(SEXP m_sexp);
```

我们还是通过 inline 的例子来说明使用方法:

```
> as.src <- '
std::vector<int> intvec = as< std::vector<int> >(intx);
// wrong way! std::vector<int> impvec(intx);
int N = as<int>(n);
int m = Rf_asInteger(n); //Rf_as*** is C API
List lst(N);
lst[0] = intvec; lst[1] = m;
return lst;
'
> as.test <- cxxfunction(signature(intx = "integer",
+                               n = "integer"),
+                        as.src, plugin="Rcpp")
> intx <- 1:10; n <- 2
> as.test(intx, n)

[[1]]
 [1] 1 2 3 4 5 6 7 8 9 10

[[2]]
 [1] 2
```

wrap 的作用是将 C++ 的数据类型转换成 R 的数据类型,方便将 C++ 的运算结果返回到 R 中,它也是一个模板函数:

```
// conversion from C++ to R  
template <typename T> SEXP wrap(const T& object);
```

和 as 一样很多时候 wrap 都是隐式调用,还是先看例子吧:

```
> wrap.src <- '
std::vector<std::map<std::string,int> > v;
std::map<std::string, int> m1;
std::map<std::string, int> m2;
m1["foo"]=1; m1["bar"]=2;
m2["foo"]=1; m2["bar"]=2; m2["baz"]=3;
v.push_back( m1 ); v.push_back( m2 );
return wrap( v );
'

> wrap.test <- cxxfunction(signature(), wrap.src,
+                           plugin = "Rcpp")
> wrap.test()

[[1]]
bar foo
 2  1

[[2]]
bar baz foo
 2  3  1
```

- 上面的例子把一个由 `std::map` 组成的 `std::vector` 用 `wrap` 转换成了一个 R 中的 `List`, 需要注意的是在 `std::map` 中的键值并没有顺序 (并非顺序容器), 我们可以看到返回的 `list` 中的分量是按字母大小排列的.
- 当传入和返回涉及到多个数据类型时, 用 `List` 来传入和返回会比较方便 (会隐式的调用 `as` 和 `wrap`).

有了 `as` 和 `wrap`, 我们在 R 中调用 C++ 函数或算法的一般套路将会是:

- 1 利用 `as` 将 R 的数据类型转换成 C++ 的数据类型.
- 2 利用 C++ 的现成算法, 得到你想要的结果, 这个结果一般是 C++ 的数据类型.
- 3 把这个 C++ 的数据类型用 `wrap` 转换成 R 的数据类型.

- Rcpp 中的 RObject 和 C++ 中容器类型 (vector, map 等) 有很多相似点, 其中之一就是 RObject 可以使用迭代器. 因此 C++ 的泛型算法可以直接利用, 从而提高我们在 C++ 中操作 RObject 的灵活性和便捷性.
- 虽然对于 C++ 的高手而言, 完全可以使用 C++ 的数据类型解决问题; 但 R 的使用者可能会对 RObject 更熟悉. 此外, 使用 RObject 的另一个好处是, 你可以直接调用 R 中的函数来处理它, 尽管这样可能降低你的运行速度.
- Rcpp Sugar 还提供了一大堆支持 RObject 类似于 R 中向量化的操作符, 和函数, 速度也非常快.

迭代器是 C++ 中重要的概念, 它有点像指针, 它提供了对容器中元素访问的一般性办法. 而泛型算法的实现独立于容器类型, 这个与 R 的泛型函数原理上是一致的, 只是实现不同. 如果想仔细学习, 建议阅读 *C++ primer*. 我们还是通过例子来看看如何使用:

```
> algo.src <- '
List input(data); Function f(fun);
List output(input.size());
std::transform(input.begin(), input.end(),
               output.begin(), f); // transform algo
output.names() = input.names();
NumericVector first = input[0], second = input[1];
double s = std::accumulate(first.begin(),
                           first.end(), 0.0); // sum
int p = std::accumulate(second.begin(), second.end(),
                        1, std::multiplies<int>()); //prod
// sum(first * second)
double s2 = std::inner_product(first.begin(),
                               first.end(),
                               second.begin(), 0.0);

NumericVector algo(3);
algo[0] = s; algo[1] = p; algo[2] = s2;
output.push_back(algo);
return output;
'
```

```
> algo.test <- cxxfunction(signature(data = "list",
+                                   fun = "function"),
+                           algo.src, plugin = "Rcpp")
> input <- list(a = seq(0.1, 1, by = 0.1), b = 1:10)
> algo.test(input, summary)
```

\$a

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
0.100	0.325	0.550	0.550	0.775	1.000

\$b

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1.00	3.25	5.50	5.50	7.75	10.00

[[3]]

[1] 5.5 3628800.0 38.5

迭代器访问向量元素的例子, 比下标 [] 访问快 30% 左右:

```
> iter.src <- '
Rcpp::NumericVector xa(a), xb(b);
int n_xa = xa.size(), n_xb = xb.size();
Rcpp::NumericVector xab(n_xa + n_xb - 1);
// other iterator like LogicVector::iterator ...
// or List::iterator can be used as well
typedef Rcpp::NumericVector::iterator vec_iterator;
vec_iterator ia = xa.begin(), ib = xb.begin();
vec_iterator iab = xab.begin();
for (int i = 0; i < n_xa; i++)
    for (int j = 0; j < n_xb; j++)
//you can use [ ] for iterators to get element
        iab[i + j] += ia[i] * ib[j];
return xab;
'
```

```
> iter.test <- cxxfunction(signature(a = "numeric",  
+                                 b = "numeric"),  
+                           iter.src, plugin = "Rcpp")  
> iter.test(1:4, 1:5)  
  
[1] 1 4 10 20 30 34 31 20
```

C++ 中调用 R 中的函数

- 虽然 C++ 是一个很强大的语言,但是它也很复杂,对于一些 R 中已实现的算法和函数,我们希望可以`直接调用`,而不是用 C++ 来重写.
- 前面的例子里其实已经涉及到了如何在 Rcpp 中使用 R 的函数,很简单,就是利用 `Rcpp::Function` 类.
- 对于比较复杂的函数,`Rcpp::Environment`, `Rcpp::Formula`, 以及 `Rcpp::Language` 会让你在 Rcpp 中调用 R 的函数变得很自然.

我们依然通过例子来说明如何使用:

```
> fun.src <- '
Environment stats("package:stats");
Function LM = stats["lm"];
Environment utils("package:utils");
Function read_table = utils["read.table"];
DataFrame somedata = read_table("some data.csv",
                                _["header"] = true,
                                _["sep"] = ",");
Formula formu("y~x"); List lmfit = LM(formu, somedata);
return lmfit["residuals"];
'

> fun.test <- cxxfunction(signature(),
+                          fun.src, plugin = "Rcpp")
> a <- 1:5; b <- a + rnorm(5)
> write.csv(data.frame(x = a, y = b),
+           file = "some data.csv", row.names = F)
> fun.test()

      1      2      3      4      5
-0.17209 0.43137 -0.40550 0.20524 -0.05903
```

Rcpp Sugar

Rcpp Sugar 为 RObject 的子类们重定义了很多运算符和函数, 使得它们在 C++ 中的各种运算更像是在 R 中的运算 (向量化的运算), 速度也会比 C++ 中的循环快:

- $+$ $-$ $*$ $/$ 等运算符对于 NumericVector 等向量类型是向量化的
- 一些比较运算符 $>$ $<$ 等也是向量化的
- 一些常用的数学函数也被定义成向量化的, 如 abs, floor, log, pow, ifelse, pmin...
- 各种概率函数 d/p/q/r 函数, 可以在 Rcpp 中直接调用
- sapply, lapply 函数.

简单的例子:


```
> temp.src <- '
template <typename T>
T square( const T& x){
return x * x ;
}
|

> sugar.src <- '
NumericVector x(4);
// mean = 0 sd = 2.0
RNGScope scope; //set seed
x = rnorm(4, 0, 2.0);
NumericVector y = dnorm(x);
NumericVector z = abs(x - y);
NumericVector w = sapply( z , square<double>);
return List::create(x, y, z, w);
|

> sugar.test <- cxxfunction(signature(), sugar.src,
+                           includes = temp.src,
+                           plugin = "Rcpp")
```

```
> sugar.test()
```

```
[[1]]
```

```
[1] 1.7188 0.2110 -2.5090 0.4325
```

```
[[2]]
```

```
[1] 0.09107 0.39016 0.01714 0.36332
```

```
[[3]]
```

```
[1] 1.62776 0.17920 2.52618 0.06919
```





```
[[4]]
```

```
[1] 2.649595 0.032114 6.381576 0.004787
```

Module, RInside 简介

- Rcpp Module 提供了 R 调用 C++ 一系列函数或者类的简单方法, 你甚至不用做类新转换, Module 会自动分辨类型并转换, 设计的动机来自与 Boost.Python 中的 Python modules. 一般用 Rcpp 写包的时候用 Module 会非常方便.
- 用 Rcpp 写包是一个很好的选择, 参考 *Writing a package that uses Rcpp* .
- RInside 是另外一个包, 它提供了一些 C++ 类使得 C++ code 中嵌入 R 变得很方便. 适用于用到 R 的 C++ 项目.

参考文献

-  *The R Manuals, R core.*
-  *Rcpp Help Documents, Dirk Eddelbuettel and Romain Fran.*
-  *Rcpp Workshop slides, Dirk Eddelbuettel and Romain Fran.*
-  *C++ primer.*

谢 谢!