Funções da família apply e funções que dependem da classe

Cristiano de Carvalho

Departamento de Estatística, Universidade Federal de Minas Gerais (UFMG)

Family apply

O R é uma linguagem vetorial e ''loops'' podem e devem ser substituídos por outras formas de cálculo sempre que possível.

Usualmente usamos as funções apply(), sapply(), tapply() e lapply() para implementar cálculos de forma mais direta, mas não necessariamente mais rápida.

apply

- apply() para uso em matrizes, arrays ou data-frames

```r
N = 10000 # Numero de replicas
n = 3000  # tamanho da amostra
x <- matrix(rnorm(N * n), nc = N)
dim(x)
```

```
## [1]  3000 10000
```

```r
# definindo função para calcular todas as médias
f <- function(x){
  Ns = dim(x)[2]
  mx <- numeric(Ns)
  for(i in 1:Ns) mx[i] <- mean(x[,i])
  mx
}
```

```r
system.time(mx1 <- f(x)) # usando o for
```

```
##    user  system elapsed
##    0.72    0.03    0.77
```

```r
system.time(mx2 <- apply(x, 2, mean)) # usando o apply
```

```
##    user  system elapsed
##    1.03    0.05    1.11
```

```r
system.time(mx3 <- colMeans(x)) # outra alternativa
```

```
##    user  system elapsed
##    0.05    0.00    0.06
```

sapply

- sapply() para uso em vetores, simplificando a estrutura de dados do resultado se possível (para vetor ou matriz)

```
fc = function(n, mu, sigma){
  x = rnorm(n, mu, sigma)
}
s = sapply(X=c(5,10,15), FUN =fc, mu=2, sigma=1)
s
```

```
## [[1]]
## [1] 2.253411 1.177220 1.770828 3.602213 2.643071
##
## [[2]]
##  [1] 1.765408 2.503420 2.999617 2.538816 2.492517 3.069
##  [8] 1.869702 2.494621 1.892166
##
## [[3]]
##  [1]  4.90354884  1.89424572  1.07000465  2.92686409  0.
```

mapply

- mapply() para uso em vetores, versão multivariada de sapply()

```
# Gerando amostras
fc = function(n, mu, sigma){
  x = rnorm(n, mu, sigma)
}
s = mapply(fc, n=c(5,10,15), mu=c(0,2,3), sigma=c(1,1,1))
s
```

```
## [[1]]
## [1] -0.11040734 -0.17709568  1.11295726 -0.50816968 -0.
##
## [[2]]
##  [1] 0.3234062 2.5741298 0.7901875 2.5540707 2.0809862
##  [8] 0.5276540 3.5670882 1.1030892
##
## [[3]]
##  [1] 2.8127570 2.5392768 2.4961453 0.9220016 3.1315118
```

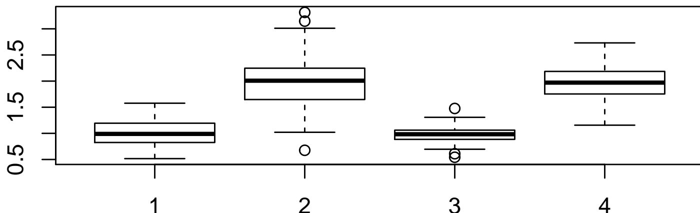
```r
# Gerando uma amostra apenas e calculando o desvio padrão
fc = function(n, mu, sigma){
  x = rnorm(n, mu, sigma);  s = sd(x)
}
s = mapply(fc, n=c(10,10,30,30), mu=c(0,2,0,2),
           sigma=c(1,1,1,1))
s
```

```
## [1] 1.0198189 0.8410599 1.2016240 0.9490403
```

- replicate() repete o uso de alguma função

```r
# Para gerar varias amostras e calcular a média
fc2 = function(n, mu, sigma){
  replicate(100, fc(n, mu, sigma))
  }

s = mapply(fc2, n=c(10,10,30,30), mu=c(0,0,0,0),
               sigma=c(1,2,1,2))
boxplot(s)
```



tapply

- tapply() pode ser usada para calcular o resultado de uma operação sobre dados, para cada um dos níveis de uma segunda variável

```r
# warpbreaks[,-1] contem dois fatores
tapply(warpbreaks$breaks, warpbreaks[,-1], sum)
```

```
##     tension
## wool   L   M   H
##    A 401 216 221
##    B 254 259 169
```

lapply

- lapply() para ser aplicado em listas

```r
x <- list(a = 1:10, beta = exp(-3:3),
          logic = c(TRUE,FALSE,FALSE,TRUE))
lapply(x, mean)
```

```
## $a
## [1] 5.5
##
## $beta
## [1] 4.535125
##
## $logic
## [1] 0.5
```

```r
x <- list(a = 1:10, beta = exp(-3:3),
          logic = c(TRUE,FALSE,FALSE,TRUE))
lapply(x, quantile, probs = 1:3/4)
```

```
## $a
##  25%  50%  75%
## 3.25 5.50 7.75
##
## $beta
##       25%       50%       75%
## 0.2516074 1.0000000 5.0536690
##
## $logic
## 25% 50% 75%
## 0.0 0.5 1.0
```

vapply

- vapply() é similar ao sapply, mas tem um tipo predeterminado de valor retornado. Pode ser mais seguro e rápido

```
fc = function(n, mu, sigma){
  set.seed(1)
  x = rnorm(n, mu, sigma); media = mean(x)
}
sapply(X=c(10,10,30,30), FUN =fc, mu=2, sigma=1)
```

```
## [1] 2.132203 2.132203 2.082458 2.082458
```

```
vapply(X=c(10,10,30,30), FUN =fc, mu=2, sigma=1,
       FUN.VALUE = 0)
```

```
## [1] 2.132203 2.132203 2.082458 2.082458
```

rapply

- rapply() para a versão recursiva do lapply

```r
X <- list(list(am1 = rnorm(100), am2 = rexp(100)),
          am3=rbinom(10,5,0.5))
rapply(X, mean, how = "replace")
```

```
## [[1]]
## [[1]]$am1
## [1] 0.1062985
##
## [[1]]$am2
## [1] 0.9790292
##
##
## $am3
## [1] 2.7
```

```r
rapply(X, mean, how = "list")
```

```
## [[1]]
## [[1]]$am1
## [1] 0.1062985
##
## [[1]]$am2
## [1] 0.9790292
##
##
## $am3
## [1] 2.7
```

```r
rapply(X, mean, how = "unlist")
```

```
##       am1       am2       am3
## 0.1062985 0.9790292 2.7000000
```

Uma comparação de tempo entre for, sapply e lapply

```
nc <- 1e3
nr <- 200

## Vendo várias formas de fazer a mesma coisa
set.seed(1)
res0 <- sapply(1:nc, function(x) rnorm(nr))



set.seed(1)
res1 <- NULL; for (i in 1:nc)
  res1 <- cbind(res1, rnorm(nr))
```

```r
set.seed(1)
res2 <- matrix(NA, nc=nc, nr=nr); for (i in 1:nc)
  res2[,i] <- rnorm(nr)

set.seed(1)
res3 <- do.call(cbind,
                lapply(1:nc, function(x) rnorm(nr)))

set.seed(1)
res4 <- replicate(nc, rnorm(nr))
```

```r
all.equal(res0, res1)
```

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

```r
all.equal(res1, res2)
```

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

```r
all.equal(res2, res3)
```

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

```r
all.equal(res3, res4)
```

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

```r
## removendo todos os objetos declarados anteriormente
rm(list=ls(all=TRUE))

nc <- 5e3
nr <- 200

## com sapply
set.seed(1)
t0 <- system.time(res0 <- sapply(1:nc,
                                  function(x) rnorm(nr)))
rm(res0)

## com for() usando o q a maioria de usuarios faz...
set.seed(1)
t1 <- system.time({res1 <- NULL;
for (i in 1:nc) res1 <- cbind(res1, rnorm(nr))})
rm(res1)
```

```r
## com for() mais eficiente
set.seed(1)
t2 <- system.time({res2 <- matrix(NA, nc=nc, nr=nr);
for (i in 1:nc)   res2[,i] <- rnorm(nr)})
rm(res2)

## com lapply()
set.seed(1)
t3 <- system.time(res3 <- do.call(cbind,
                                   lapply(1:nc,
                                     function(x)rnorm(nr))))
rm(res3)

## com replicate
set.seed(1)
t4 <- system.time(res4 <- replicate(nc, rnorm(nr)))
rm(res4)
```

```
##    user.self sys.self elapsed user.child sys.child
## t0      0.20      0.0    0.22         NA        NA
## t1     34.19      5.3   39.85         NA        NA
## t2      0.22      0.0    0.22         NA        NA
## t3      0.19      0.0    0.19         NA        NA
## t4      0.19      0.0    0.18         NA        NA
```

Integral Monte Carlo

Suponha que queremos resolver

$$I = \int_2^6 x^2 dx = (x^3/3)|_3^6 = 69,333.$$

Seja $Y \sim Unif(2,6)$, logo

$$E[Y^2] = \int_2^6 y^2/4dy$$

Logo, $I = 4E[Y^2]$.

Pela Lei dos grandes números, se $n \to \infty$ então

$$\bar{Y} \stackrel{q.c.}{\to} E[Y]$$

e, também,

$$\overline{g(Y)} \stackrel{q.c.}{\to} E[g(Y)].$$

Logo, podemos aproximar $E[Y^2]$ gerando uma amostra grande Y e calculando a média amostral de $y_1^2, \ldots, y_n^2$.

```r
## Um pouco sobre integral Monte Carlo:

fc = function(x){
  y = x^2
  return(y)
}

## De forma exata: x^3/2
(6^3)/3-(2^3)/3
```

```
## [1] 69.33333
```

```r
## Resolvendo por um metodo computacional:
## quadratura adaptativa
integrate(f = fc, lower = 2, upper = 6)
```

```
## 69.33333 with absolute error < 7.7e-13
```

```r
## O que eh simulação Monte Carlo?
## O que eh integral Monte Carlo?
M = 100000
x = runif(M, 2, 6)
4*mean(x^2)
```

```
## [1] 69.28479
```

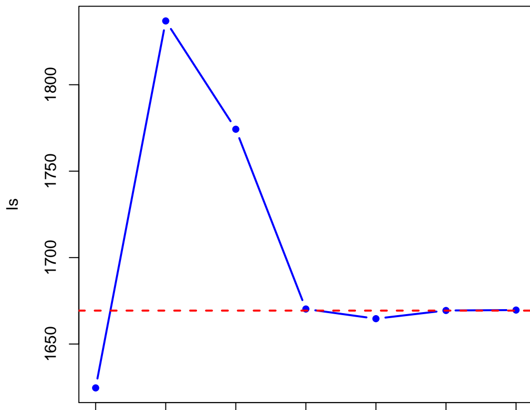
```r
## A precisão desse metodo de integracao eh boa?
## Considere agora a funcao x^2+5*x^3
fc = function(x){
  y = x^2+5*x^3
  return(y)
}
## Valor confiavel
valor = integrate(f = fc, lower = 2, upper = 6)$value

fc2 = function(M){
  x = runif(M, 2, 6)
  return(4*mean(x^2+5*x^3))
}

Ms = 10^(1:7)
Is = sapply(X = Ms , FUN = fc2)
```

```r
## A precisão desse metodo de integracao eh boa?
##Considere agora a funcao x^2+5*x^3

plot(log(Ms,base = 10), Is, type="b",pch=16,col=4,lwd=2)
abline(h=valor,col=2,lty=2,lwd=2)
```



Funções genéricas

O envio do método começa com uma função genérica que decide para qual método específico executar. Todas as funções genéricas têm a mesma forma: uma chamada para UseMethod que especifica o nome genérico e o objeto a ser despachado.

Isso significa que funções genéricas são geralmente muito simples, como mean:

```
mean <- function (x, ...) {
  UseMethod("mean", x)
}
```

Métodos são funções comuns que usam uma convenção de nomenclatura especial:

```
mean.numeric <- function(x, ...) sum(x) / length(x)
mean.data.frame <- function(x, ...) sapply(x, mean, ...)
mean.matrix <- function(x, ...) apply(x, 2, mean)
```

Mais detalhes de como usar no script!