Introdução a Linguagem R

Funções de Repetição - lapply

Delermando Branquinho Filho

Looping na linha de comando

Normalmente troca-se while for e outros loops ao programar, mas não particularmente fácil ao trabalhar interativamente na linha de comando. Existem algumas funções que implementam looping para facilitar a vida.

- `lapply`: Loop sobre uma lista e avaliar uma função em cada elemento
- `sapply`: O mesmo que`lapply` mas tenta simplificar o resultado
- `apply`: Aplica uma função sobre as margens de uma matriz
- `tapply`: Aplicar uma função sobre subconjuntos de um vetor
- `mapply`: Versão multivariada do`lapply`

Uma função auxiliar `split` também é útil, especialmente em conjunto com`lapply`.

lapply

`lapply` tem três argumentos:

(1) uma lista`X`;
(2) uma função (ou a Nome de uma função) `FUN`;
(3) outros argumentos através da seu ... argumento. Se `X` não for uma lista, será coagido a uma lista usando `as.list`.

```
lapply
```

```
## function (X, FUN, ...)
## {
##     FUN <- match.fun(FUN)
##     if (!is.vector(X) || is.object(X))
##         X <- as.list(X)
##     .Internal(lapply(X, FUN))
## }
## <bytecode: 0x0000000012786560>
## <environment: namespace:base>
```

O loop real é feito internamente no código C.

`lapply` Sempre retorna uma lista, independentemente da classe da entrada.

```
x <- list(a = 1:5, b = rnorm(10))
lapply(x, mean)
```

```
## $a
## [1] 3
##
## $b
## [1] 0.1427144
```

```r
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
lapply(x, mean)
```

```
## $a
## [1] 2.5
##
## $b
## [1] -0.431183
##
## $c
## [1] 0.632408
##
## $d
## [1] 4.93014
```

```r
> x <- 1:4
> lapply(x, runif)
[[1]]
[1] 0.2675082

[[2]]
[1] 0.2186453 0.5167968

[[3]]
[1] 0.2689506 0.1811683 0.5185761

[[4]]
[1] 0.5627829 0.1291569 0.2563676 0.7179353
```

---

lapply

```r
> x <- 1:4
> lapply(x, runif, min = 0, max = 10)
[[1]]
[1] 3.302142

[[2]]
[1] 6.848960 7.195282

[[3]]
[1] 3.5031416 0.8465707 9.7421014

[[4]]
[1] 1.195114 3.594027 2.930794 2.766946
```

lapply

`lapply` e os amigos fazem o uso pesado de funções anônimas.

```r
x <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))
x$a
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
x$b
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

---

lapply

Uma função anônima para extrair a primeira coluna de cada matriz.

```
lapply(x, function(elt) {elt[,1]})
```

```
## $a
## [1] 1 2
##
## $b
## [1] 1 2 3
```

```
x$a
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
x$b
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

---

sapply

`sapply` irá tentar simplificar o resultado de`lapply` se possível.

- Se o resultado for uma lista onde cada elemento é comprimento 1, então um vetor é retornado
- Se o resultado for uma lista onde cada elemento é um vetor do mesmo comprimento (> 1), uma matriz é retornada.
- Se não conseguir descobrir as coisas, uma lista é retornada

---

sapply

```
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
lapply(x, mean)
```

```
## $a
## [1] 2.5
##
## $b
## [1] 0.1281351
##
## $c
## [1] 0.5683913
##
## $d
## [1] 5.004133
```

```
x$a
```

```
## [1] 1 2 3 4
```

```
x$b
```

```
##  [1] -1.01468994  1.83972583 -0.27338295  0.01620667  1.64616977
##  [6]  0.46230434 -0.26644572 -0.09772048  0.05748852 -1.08830520
```

```
x$c
```

```
##  [1]  1.76179056 -0.89202420 -0.18218465  0.46206811 -0.03428372
##  [6] -0.68796136  2.39350851  1.88795486  0.36862159  1.42758109
## [11]  1.18844888 -1.09670731  0.15177346  1.60869461 -0.72256921
## [16]  1.36765662  1.43193853 -0.04593221  1.74250718 -0.76305633
```

```
x$d
```

```
##   [1] 5.925300 4.193459 7.257066 5.138255 3.101439 5.066818 3.583077
##   [8] 5.864580 3.381334 6.140173 6.987087 5.072616 3.803821 4.640494
##  [15] 3.690335 5.703460 5.901827 5.735807 4.699387 5.785606 4.384201
##  [22] 6.963985 5.232446 3.227681 4.224918 6.305607 6.312685 5.517383
##  [29] 4.601863 5.355043 3.798393 4.301367 4.214754 3.744036 5.597115
##  [36] 4.587906 5.133165 2.251878 5.691778 4.933783 7.440221 5.434903
##  [43] 4.338512 4.232417 4.333906 3.904025 4.943307 5.342624 4.989649
##  [50] 3.898835 4.748161 4.258439 5.486637 3.469577 4.440264 4.672130
##  [57] 5.288631 4.910981 4.113028 6.010932 3.709113 3.427906 5.658264
##  [64] 6.923197 5.200554 4.575250 3.535667 6.354246 4.532959 5.195654
##  [71] 4.558924 4.819090 6.883984 6.091997 5.017270 4.237469 4.260613
##  [78] 4.653977 5.580166 2.728119 4.762720 6.494409 6.221615 6.268367
##  [85] 4.639195 7.386262 3.604501 6.123464 6.177337 3.866083 5.475597
##  [92] 4.780145 4.016821 6.690998 2.945451 5.012005 5.506618 5.039532
##  [99] 6.580266 6.568379
```

---

sapply

```
sapply(x, mean)
```

```
##         a         b         c         d
## 2.5000000 0.1281351 0.5683913 5.0041330
```

```
mean(x)
```

```
## Warning in mean.default(x): argument is not numeric or logical: returning
```

```
## NA
```

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

```
#Warning message:
#In mean.default(x) : argument is not numeric or logical: returning NA
```