Estatística Computacional I

Lupércio França Bessegato
Dep. de Estatística/UFJF

Roteiro Geral

1. Programando em R
2. Preparação, limpeza e manipulação de dados
3. Gráficos em R
4. Tópicos especiais
5. Referências

Programando em R

Vetorização

Vetorização

- Conversão de operações repetidas com escalares (números simples) em operações com vetores ou matrizes.
  - √ Exemplo: $\sum_{i=1}^{n-1} \frac{e^{-x_{i+1}}}{x_i + 10}$

```
>   # exemplo
> mat <- 666
> set.seed(mat)
> xVet <- rnorm(1000)
> ene <- length(xVet)
> xNum <- xVet[-1]
> xDen <- xVet[-ene]
> sum(exp(xNum)/(xDen +10))
[1] 158.1719
```


- Comentários:
  - √ Vetor é a estrutura de dados elementar no R
    - Consiste em uma coleção de coisas
  - √ Vetorização permite que muitas das construções de loop possam ser tornadas implícitas
    - Em geral, são mais rápidas do que o código R explícito equivalente
  - √ O R é uma linguagem interpretada
    - Todos os detalhes sobre a definição de variáveis (tipo, tamanho, etc) são atendidos pelo intérprete
    - Essas definições (tipo, estrutura, alocação, etc.) são trabalhadas em todo comando

- √ Linguagens que suportam vetorização
  - Toda instrução que usa um dado numérico atua sobre um objeto que é definido como um vetor
- √ O formato vetorial permite utilizar rotinas de Álgebra Linear muito eficientes

- Exemplo:
  - √ Loop com `for`:

```
>   # exemplo
> # loop com for
> set.seed(666);
> m <- 8; n <- 5;
> # matriz com números aleatórios (n x m)
> # m vetores de tamanho n de uma normal(0, 1)
> mat.norm <- replicate(m, rnorm(n))
> dim(mat)
[1]  5 10
> mat.norm <- data.frame(mat.norm)
> for (i in 1:n) {
+   for (j in 1:m) {
+     mat.norm[i, j] <- mat.norm[i, j] + 10*sin(0.75*pi)
+ # cat(i, j, mat.norm[i, j], "\n") # use esse comando para "ver" o loop
+   }
+ }
> mat.norm
       X1       X2       X3       X4       X5       X6       X7       X8
1 7.824379 7.829464 9.221110 6.995241 6.378969 5.588793 7.826064 7.334248
2 9.085422 5.764883 5.300837 7.929368 5.888024 5.944419 6.429579 7.500286
3 6.715933 6.268548 7.935721 7.415968 8.339919 5.307225 8.502200 5.586848
4 9.099236 5.278827 5.350912 6.488615 6.758516 6.008449 6.446517 7.251076
5 4.854193 7.029035 7.205193 7.857238 7.101639 5.728069 7.300063 9.132128
```

√ Loop usando vetorização:

```
> # loop vetorizado
> set.seed(666);
> m <- 8; n <- 5;
> # matriz com números aleatórios (n x m)
> # m vetores de tamanho n de uma normal(0, 1)
> mat.norm <- replicate(m, rnorm(n))
> dim(mat)
[1]  5 10
> mat.norm <- data.frame(mat.norm)
> mat.norm <- mat.norm + 10*sin(0.75*pi)
> mat.norm
        X1       X2       X3       X4       X5       X6       X7       X8
1 7.824379 7.829464 9.221110 6.995241 6.378969 5.588793 7.826064 7.334248
2 9.085422 5.764883 5.300837 7.929368 5.888024 5.944419 6.429579 7.500286
3 6.715933 6.268548 7.935721 7.415968 8.339919 5.307225 8.502200 5.586848
4 9.099236 5.278827 5.350912 6.488615 6.758516 6.008449 6.446517 7.251076
5 4.854193 7.029035 7.205193 7.857238 7.101639 5.728069 7.300063 9.132128
```


√ Tempo de execução:

– Comando `system.time`:

```
> # medida do tempo de execução do loop em for
> set.seed(666)
> m <- 5000; n <- 10
> mat.norm <- replicate(m, rnorm(n))
> mat <- data.frame(mat.norm)
> system.time(
+   for (i in 1:n) {
+     for (j in 1:m) {
+       mat[i, j] <- mat.norm[i, j] + 10*sin(0.75*pi)
+     }
+   }
+ )
  usuário   sistema decorrido
     6.47      0.00      6.50
```

```
> # medida do tempo de execução vetorizado
> system.time(
+   mat <- mat.norm + 10*sin(0.75*pi)
+ )
  usuário   sistema decorrido
        0         0         0
```


X. INDEX

Comando `Vectorize`

- Cria uma função que vetoriza a ação de seu argumento `FUN`
  - √ Atua em funções que não são compatíveis com vetores
- Sintaxe:

```
Vectorize(FUN, SIMPLIFY = TRUE,
 vectorize.args = arg.names, ...)
```

  - √ `FUN`: função a ser aplicada
  - √ `vectorizze.args`: argumentos que devem ser vetorizados
  - √ `...`: demais argumentos

- Exemplo:

√ Função para calcular a quantidade `nC2`:

```
>   # Exemplo - quantidade nC2
> nC2 <- function(vc, ec){
+        # vc: qte. pessoas; ec: qte. apertos de mãos
+        # Calcula nC2 e retorna a razão
+        ec.max = vc * (vc-1) / 2
+        return (trunc(ec * 100 / ec.max))
+ }
> # 5 pessoas e 4 apertos
> nC2(5, 4)
[1] 40
> # 5 pessoas e 4 apertos
> nC2(5, 4)
[1] 40
> # 6 pessoas e 5 apertos
> nC2(6, 5)
[1] 33
```

√ Quantos apertos de mãos são possíveis nos dois casos?

√ Cálculos com a função nC2

```
> # cálculo vetorizado
> nC2(c(5,6), c(4,5))
[1] 40 33
> # 0 pessoas e 5 apertos
> nC2(0, 5)
[1] -Inf
> # 0 pessoas e nenhum aperto
> nC2(0, 0)
[1] NaN
```

√ Correção – retorna 0 nesses casos

```
>   # Função para corrigir casos em que haja 1 ou nenhuma pessoa
> nC2 <- function(vc, ec){
+ # se não houver pessoas, retorne 0
+       if(vc <= 1) return (0)
+       # calcule nC2 e retorne a razão
+       ec.max = vc * (vc - 1) / 2
+       return (trunc(ec * 100 / ec.max))
+ }
```


√ Cálculo da função nC2 nas duas situações

```
> # 0 pessoas e nenhum aperto
> nC2(0, 0)
[1] 0
> # 0 pessoas e 5 apertos
> nC2(0, 5)
[1] 0
> # cálculo com vetores
nC2(c(5, 0), c(4, 5))
[[1]   40 -Inf
Warning message:
In if (vc <= 1) return(0) :
  a condição tem comprimento > 1 e somente o primeiro elemento será usado
```

√ A função nC2 corrigida não trabalha vetorizada

√ Cálculo vetorizado da função nC2:

```
>   # Cálculo vetorizado
> Vectorize(nC2)(c(5,0), c(4,5))
[1] 40  0
> # vetorização da função
> nC2vet <- Vectorize(nC2)
> # cálculo vetorizado
> nC2vet(c(5,0), c(4,5))
[1] 40  0
> # cálculo escalar
> nC2vet(5, 4)
[1] 40
```

√ `Vectorize` tenta vetorizar todos os argumentos do método fornecido
- Às vezes pode não funcionar

Comandos Auxiliares

Tempo de Execução

- Comando `Sys.time`:

```
> # data e instante do sistema
> Sys.time()
[1] "2018-05-17 00:05:36 BRT"
> # medida de execução de função
> parada.30s <- function() { Sys.sleep(30) }
> inicio.comando <- Sys.time()
> parada.30s()
> fim.comando <- Sys.time()
> fim.comando - inicio.comando
Time difference of 30.06805 secs
```

√ Comando simples e flexível

√ Utiliza as medidas de tempo do processador

- Comando `system.time`:

```
>   # Comando system.time
>
> system.time({ parada.30s() })
  usuário   sistema decorrido
     0.00      0.02     30.24
```

√ `elapsed`: tempo de execução da função

√ `user`: tempo da CPU na sessão do R

√ `system`: tempo da CPU gasto pelo sistema operacional no processamento das operações atuais

- Pacote `tictoc`:

√ Exemplo:

```
> library(tictoc)
> tic("modo soneca")
> print("dormindo ...")
[1] "dormindo ..."
> parada.30s()
> print("... acordando")
[1] "... acordando"
> toc()
modo soneca: 30.18 sec elapsed
```


√ Exemplo – *timers* aninhados

```
> # timers aninhados
>
> tic("total")
> tic("geração de dados")
> X <- matrix(rnorm(50000*1000), 50000, 1000)
> b <- sample(1:1000, 1000)
> y <- runif(1) + X %*% b + rnorm(50000)
> toc()
> tic("ajuste de modelo linear")
> model <- lm(y ~ X)
> toc()
> toc()
geração de dados: 4.37 sec elapsed
ajuste de modelo linear: 65.16 sec elapsed
total: 69.55 sec elapsed
> dim(X)
[1] 50000  1000
> length(b)
[1] 1000
```

- Pacote `rbenchmark`:

```
> library(rbenchmark)
>
>          # verificação do tempo de processamento de 3 procedimentos de cálculo
> benchmark("mod" = {
+             X <- matrix(rnorm(1000), 100, 10)
+             y <- X %*% sample(1:10, 10) + rnorm(100)
+             b <- lm(y ~ X + 0)$coef
+           },
+           "pseudoinversa" = {
+             X <- matrix(rnorm(1000), 100, 10)
+             y <- X %*% sample(1:10, 10) + rnorm(100)
+             b <- solve(t(X) %*% X) %*% t(X) %*% y
+           },
+           "sistema linear" = {
+             X <- matrix(rnorm(1000), 100, 10)
+             y <- X %*% sample(1:10, 10) + rnorm(100)
+             b <- solve(t(X) %*% X, t(X) %*% y)
+           },
+           replications = 1000,
+            columns = c("test", "elapsed", "relative", "user.self", "sys.self"))
           teste    réplicas elapsed relative user.self sys.self
1           mod         1000    1.01    7.214      1.02        0
2  pseudoinversa         1000    0.21    1.500      0.20        0
3 sistema linear         1000    0.14    1.000      0.14        0
```

Relativo: razão com o mais rápido

- Comparação de métodos de cálculo de coeficientes de regressão linear:
  - √ Ajuste de modelo linear (comando `lm`)
  - √ Inversa generalizada de Moore-Penrose

$$\mathbf{X}^{+} = (\mathbf{X}^{t}\mathbf{X})^{-1}\mathbf{X}^{t}$$

  - √ Inversa generalizada de Moore-Penrose, sem matrizes inversas explícitas
    - Uso do comando `solve` do R

- Pacote `microbenchmark`:
  - √ Exemplo:

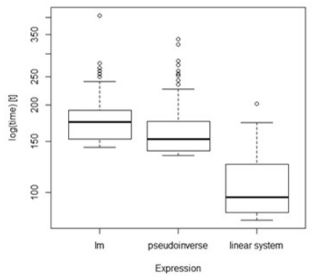
```
> library(microbenchmark)
> set.seed(666)
> n <- 10000
> p <- 100
> # matriz de dados - (valores das 100 explicativos)
> X <- matrix(rnorm(n*p), n, p)
> dim(X)
[1] 10000   100
> # vetor das respostas
> y <- X %*% rnorm(p) + rnorm(100)
> # função para checar os valores dos 101 coeficientes de regressão
> verifica.coefs <- function(values) {
+   tol <- 1e-12
+   max_error <- max(c(abs(values[[1]] - values[[2]]),
+                      abs(values[[2]] - values[[3]]),
+                      abs(values[[1]] - values[[3]])))
+   max_error < tol
+ }
```


- √ Verificação do tempo de processamento:
  - Comando `microbenchmark`.

```
> # verificação do tempo de processamento de 3 procedimentos de cálculo
> mbm <- microbenchmark("mod = { b <- lm(y ~ X + 0)$coef },
+                "pseudoinversa" = {
+                  b <- solve(t(X) %*% X) %*% t(X) %*% y
+                },
+                "sistema linear" = {
+                  b <- solve(t(X) %*% X, t(X) %*% y)
+                },
+    # argumentos do comando
+                check = verifica.coefs, times = 100L)
> mbm
Unit: milliseconds
          expr      min        lq     mean    median        uq      max neval
           mod 143.6041 153.29100 182.1130 175.45494 191.9938 406.3161   100
 pseudoinversa 134.3636 139.13713 167.5937 153.13120 175.5606 336.8410   100
sistema linear  80.6214  85.46624 106.9313  96.49853 125.1804 202.2652   100
> str(mbm)
Classes 'microbenchmark' and 'data.frame':      300 obs. of  2 variables:
 $ expr: Factor w/ 3 levels "lm","pseudoinverse",..: 3 3 2 3 2 2 1 1 1 3 ...
 $ time: num  8.06e+07 8.47e+07 1.39e+08 9.11e+07 1.41e+08 ...
```

√ Visualização gráfica dos resultados:

```
> # grafico dos resultados de microbenchmark
> boxplot(mbm)
```

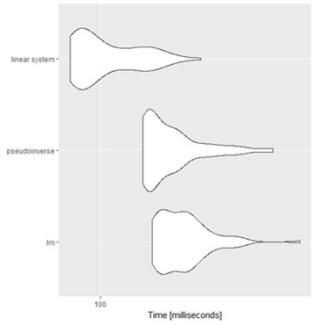


√ Distribuição dos tempos de processamento de cada procedimento de cálculo dos coeficientes.

√ Visualização gráfica dos resultados:

– Integração com o pacote `ggplot2`

```
> # grafico dos resultados - pacote ggplot2
> library(ggplot2)
> autoplot(mbm)
```



√ Densidade dos tempos de processamento de cada procedimento de cálculo dos coeficientes.

• Comentário:

√ A função `microbenchmark` verifica automaticamente os resultados das expressões, comparando-as com uma função especificada pelo usuário.

• Comando `proc.time`:

```
> # Comando proc.time
>
> n <- 100
> inicio <- proc.time()[3]
> barra <- winProgressBar(title = "Progress Bar", min = 0, max = n)
> for(i in 1:n) {
+ # Interrompe execução das funções do R durante 0,1 seg.
+ Sys.sleep(.1)
+ setWinProgressBar(barra, i, title = paste(round((i/n)*100), "% Concluído"))
+ }
> Sys.sleep(0.5)
> invisible(close(barra))
> decorrido <- proc.time()[3] - inicio
> cat("Tempo decorrido:",floor(decorrido/60), "min", decorrido%%60,"seg. \n")
Tempo decorrido: 0 min 11.47 seg.
```

√ `winProgressBar`:

– Barra de progresso durante cálculos demorados

√ Comparação – abordagens com e sem loop:

```
>   # exemplo
> g <- rnorm(100000)
> h <- rep(NA, 100000)
>
> # Incia o relógio!
> ptm <- proc.time()
>
> # Soma 1 - loop no vetor
> for (i in 1:100000){
+ h[i] <- g[i] + 1
+ }
> proc.time() - ptm
  usuário   sistema decorrido
     0.16      0.01      0.17
> # abordagem sem loop
> ptm <- proc.time()
> h <- g + 1
> proc.time() - ptm
  usuário   sistema decorrido
        0         0         0
```

192

Estatística Computacional I - 2020

Referências

Bibliografia Recomendada

- ALBERT, J.; RIZZO, M. *R by Example*. Springer, 2012.
- CHRISTIAN, N. *Basic Programming*, Lecture Notes
- DALGAARD, P. *Introductory statistics with R*. Springer, 2008.
- KLEIBER, C.; ZEILEIS, A. *Applied econometrics with R*. Springer, 2008.
- GARDENER, M. *Beginning R: The statistical programming language*. John Wiley & Sons, 2012.

286

Estatística Computacional I - 2020