Linguagem de Programação S Fundamentos e Aplicações em Recursos Florestais

5. Noções de Programação

5.4. Vetorização

5.4.1. Argumentos não Vetoriais

Existem algumas operações vetoriais que não são possíveis devido ao desenho original das funções do R. Em várias funções, somente um dos argumentos pode ser um vetor com comprimento maior que um, enquanto que os demais argumentos são assumidos como vetores unitário (comprimento 1). Nesse caso, não será possível realizar uma operação vetorial variando os argumentos que a função assume como vetores unitários.

Por exemplo, a função **dnorm** tem três argumentos essenciais:

```
> args(dnorm)
function (x, mean = 0, sd = 1, log = FALSE)
NULL
```

- O argumento **x** é o vetor dos valores para os quais se deseja a densidade probabilística segundo a distribuição Gaussiana. Ele pode ser um vetor de qualquer comprimento.
- Os argumentos **mean** e **sd** são os valores de média e desvio padrão (os parâmetros da distribuição Gaussiana) que controlam a função de densidade probabilística. Esse argumentos devem ser unitários.

Logo, a função **dnorm** pode calcular um vetor de densidades de qualquer comprimento, mas apenas para uma média e um desvio padrão.

Consideremos um exemplo numérico da função **dnorm**:

```
1 ######################################################
2 ########### 5. NOÇÕES DE PROGRAMAÇÃO
3 ########### 5.4. Vetorização
4 ######################################################
5 # 5.4.1. Argumentos não Vetoriais
6
7 args(dnorm)
8
```

```
 9 medias = seq(25, 30, length=7)
10 x = seq(-3,3, length=20)
11 y = dnorm(x, mean=medias, sd=2)
12 length(y)
13
14 sum(y - dnorm(x, mean=medias[1], sd=2))
15
```

O vetor **medias** possui comprimento 7, mas apenas o primeiro elemento dele foi utilizado no cálculo da densidade pela função **dnorm**.

5.4.2. A Função "sapply"

Mas se quisermos construir um gráfico das densidades em função da média? A função **sapply** vem em nosso socorro:

```
16 # 5.4.2. A Função "sapply"
17
18 minhas.medias = seq(25, 30, length=7)
19 meus.dados = seq(12,40, length=10)
20 y = sapply(minhas.medias,
21         function(med, dados, devpad)
22                 dnorm(x=dados, mean=med, sd=devpad),
23         dados=meus.dados,
24         devpad=2
25  )
26 dim(y)
27 y
28
```

A estrutura básica da função **sapply** é:

```
sapply(X, FUN, . . .)
```

onde:

- **X** é o vetor de valores;
- **FUN** a função a ser aplicada a cada elemento de **X**;
- "**. . .**" outros argumentos a serem passados à função **FUN**.

Refazendo o exemplo de aplicação da função **sapply**:

```
29 minhas.medias = seq(-1, 1, length=100)
30 meus.dados = seq(-4,4, length=100)
31 y = sapply(minhas.medias,
```

```
32         function(med, dados, devpad)
33                 dnorm(x=dados, mean=med, sd=devpad),
34         dados=meus.dados,
35         devpad=1
36  )
37 matplot(meus.dados, y, type="l", pch="", lty=1, col=colors())
38 matplot(minhas.medias, t(y), type="l", pch="", lty=1, col=colors())
39
40 plot(minhas.medias, apply(y, 2, function(x) sum(-log(x)) ), type="l")
41
```

5.4.3. A Função "Vectorize"

A função **sapply** nos permite vetorizar um dos argumentos de uma função, mas digamos que precisemos vetorizar dois argumentos. Como fazê-lo?

A melhor maneira é vetorizarmos a função com que trabalhamos.

Considere a seguinte situação, desejamos construir um gráfico com a superfície da soma de quadrados do resíduo, para mostrar que a “estimativas de quadrados mínimos” realmente estão no ponto mínimo dessa superfície.

O primeiro passo é construir uma função que calcule a soma de quadrados do resísuo (RSS = residual sum of squares). Usaremos como exemplo uma relação linear simples entre altura (**h**) e o CAP (**cap**) das árvores dos caixetais (caixeta.csv):

```
42 # 5.4.3. A Função "Vectorize"
43
44 RSS = function(b0, b1, h, cap)
45 {
46         h.pred = b0 + b1*cap
47         rss = sum( (h - h.pred)^2 )
48 }
49
```

Podemos agora construir vetores para os coeficientes do modelo linear simples (**b0** e **b1**), entretanto, não basta aplicar a função **RSS** às combinações dos vetores **b0** e **b1**:

```
50 cax = read.csv("caixeta.csv",as.is=TRUE)
51 b0 = seq(-50,200, length=50)
52 b1 = seq(-0.2,0.30, length=50)
53 z = outer(b0, b1, RSS, cax$h, cax$cap)
54
```

Ao executar o cálculo, o R tomará os elementos dos vetores de forma paralela. Mas como o comprimento dos maiores vetores (**h** e **cap**) não é um múltiplo dos menores (**b0** e **b1**) o cálculo não

será possível.

```
55 length(b0); length(b1); length(cax$h); length(cax$cap)
56
```

Para que façamos o R executar o cálculo tomando todos elementos de **b0** para cada elemento de **b1** e somando sobre os vetores de dados (**cax$h** e **cax$cap**) é necessário vetorizar a função **RSS** informando quais argumentos deverão ser vetorizados:

```
57 RSSv = Vectorize( FUN=RSS, vectorize.args = c("b0","b1"))
58
```

Na nova função **RSSv**, os argumentos **b0** e **b1** estão vetorizados, tornado a combinação possível:

```
59 b0 = seq(-50,200, length=200)
60 b1 = seq(-0.2,0.30, length=200)
61 z = outer(b0, b1, RSSv, cax$h, cax$cap)
62 dim(z)
63
```

Agora podemos construir um gráfico com isolinhas da soma de quadrados do resíduo:

```
64 image(b0,b1,z, col=rainbow(12))
65 contour(b0, b1, z, levels=seq(1, 15,length=9)*10^6, col="blue",add=TRUE)
66 abline( h = b1[ col(z)[z==min(z)] ], col="darkgreen")
67 abline( v = b0[ row(z)[z==min(z)] ], col="darkgreen")
68
```

5.4.4. Ordem de Prioridade dos Operadores do R

Outro aspecto formal importante da linguagem R é a ordem de prioridade de seus operadores. Além das regras de precedência usuais para as operações matemáticas, é essencial conhecer a prioridade dos outros operadores:

```
=================================
==============================
 OPERADOR         DESCRIÇÃO
PRIORIDADE
=================================
==============================
  $               seleção de
componentes                ALTA
 [  [[            indexação
|
 ^                potência
```

```
|
 -                menos unitário
|
 :                operador de
seqüência                    |
 %nome%           operadores
especiais                     |
 *  /             multiplicação,
divisão                      |
 +  -             adição e
subtração                        |
< > <= >= == !=   comparação
|
 !                não
|
 & &&  |  ||      e, ou
|
 ~                fórmula
estatística                     \/
 <<- <- -> =      atribuição
BAIXA
=================================
==============================
```

Exemplos de como a ordem de prioridade pode gerar resultados diferentes:

```
69 # 10.4.4. Ordem de Prioridade dos Operadores do R
70
71 -1:10
72 -(1:10)
73
74 1:3^2
75 (1:3)^2
76
77 1:5+2
78 1:(5+2)
79
```