Course: R Language in Computational Probability and Statistics

Lecture 6: Branches, cycles. Vectorized calculations using apply functions

Lecturer: Nataliia Kruhlova

Розгалуження

Мова R розроблена з метою мінімізації використання циклів і розгалужень. Використання логічних масивів при індексації та параметрів зі значеннями за замовчуванням значно обмежує потребу у застосуванні умовних конструкцій типу if...else. В хорошому стилі програмування R відсутність таких конструкцій вважається стандартом, якщо можливо використовувати інші методи. Проте, у деяких випадках використання цих конструкцій може зробити програму ефективнішою і зрозумілішою.

Умовний оператор if else

Умовний оператор задається наступним чином:

if(умова **)**
{ список команд 1
}else
{список команд 2}

Логіка виконання проста: якщо **умова** істинна, то виконується **список команд1**, якщо хибна, то **список команд 2 .**

Допустима й скорочена форма:

ifelse(умова,команда1,команда2),

яка має туж саму логіку виконання. Зауважимо, що частина **else** може бути відсутньою.

Приклад. Нехай в нас є n кульок, з яких потрібно вибрати k штук. Потрібно створити функцію, яка буде визначати число можливих комбінацій. Одним з параметрів нашої функції буде аргумент **with_repetitions,** що відповідає за варіант підрахунку: якщо FALSE, то рахується число комбінацій без повторення, якщо TRUE - то з повтореннями.

```
> combin_count <- function(n, k, with_repretitions = FALSE) {
 if (with_repretitions ==FALSE)
return(choose(n, k))  else
return(choose(n+k-1, n-1)) }
> combin_count(10,7)
[1] 120
> combin_count(10,7,with_repretitions = T)
[1] 11440
```

Умовний оператор switch

Функція **switch()** дозволяє вибирати один із численних варіантів виконання програми залежно від значення певного виразу. Вона має наступний синтаксис:

```
switch(object,
        "value1" = {expr1},
        "value2" = {expr2},
        "value3" = {expr3},
                {other expressions}
   )
```

Якщо **object** набуває значення **value1,** то виконується код **expr1**, якщо - **value2,** то **expr2** відповідно і т.д. У випадку коли не має співпадінь, тоді або виконується код **other expressions**, або результатом буде **NULL** у випадку відсутності **other expressions**.

Приклад. Напишіть функцію, параметром якої є вектор і назва функції, яку ми будемо застосовувати до вектору (середнє, дисперсія, медіана), якщо функції немає в переліку, то виводимо «Не цікаво».

```
> vubir<-function(x,f){
+     switch(f,mean=mean(x),var=var(x),median=median(x),"Не цікаво")
+ }
> vubir(c(-1,0,5,8,-3),"median")
[1] 0
> vubir(c(-1,0,5,8,-3),"mean")
[1] 1.8
> vubir(c(-1,0,5,8,-3),"sd")
[1] "Не цікаво"
```

Цикли while та repeat

Для організації циклічних обчислень в **R** використовуються цикли. Цикл **while()** має наступний синтаксис:

while(умова) {команда}.

Спочатку перевіряють **умову**, і у разі її значення **TRUE,** виконується **команда**, до тих пір поки умова не набуде значення **FALSE.**

Приклад. Напишіть функцію, яка із заданою точністю буде обчислювати суму ряду:

$$\sum_{i=1}^{\infty} \frac{(-1)^n n^3}{2^n}.$$

```
> Summa<-function(eps){s<-0;i<-1; while((i+1)^3/2^(i+1)>eps){s<-s+(-1)^i*i^3/2^
i;i<-i+1};s }
> Summa(10^(-5))
[1] 0.07408302
```

У циклі **while()** первірка відбувається перед початком виконання тіла циклу. Для перевірки умов завершення циклу в інших частинах програми використовують цикл **repeat ()** з командами **break** та **next**.
Формат команди
repeat{команда}.

Виконання **команди** повторюється безкінечну кількість разів доти, поки не відбудеться виконання **команди break,** призначеної для виходу з даної конструкції циклу на команду, що слідує за циклом. Команда **next** всередині тіла циклу перериває виконання циклу і передає управління на першу команду тіла.

Той самий приклад виконаємо за допомогою **repeat ().**

Приклад:

```
> Summa_repeat<-function(eps){s<-0
+ i<-1
+ repeat{s<-s+(-1)^i*i^3/2^i;i<-i+1
+ if ((i+1)^3/2^(i+1)<eps) break}
+ s}
> Summa_repeat(10^(-5))
[1] 0.07408302
```

Цикл for

Для роботи з векторами можна використовувати цикл **for,** виконуючи однотипні дії для всіх компонент вектору.

for (i in послідовність)

{

команда

},

де **команда** - вираз, що виконується кожен раз для кожного і-го елементу

об’єкта послідовності.

Приклад. Для даних **AirPassengers** розрахуйте ковзне середнє з інтервалом згладжування 10 (беремо перших 10 членів і розраховуємо їх середнє значення, потім беремо елементи з 2 до 11 і знаходимо їх середнє значення, і т.д.).

```
> new <- c()
> for (i in 1:(length(AirPassengers)-9))
+   new[i] <- mean(AirPassengers[i:(i+9)])
> new
  [1] 129.8 129.0 129.0 127.3 127.0 129.0 129.0 126.7 126.8 130.2 135.3 140.7
 [13] 142.2 142.1 143.5 143.9 145.4 150.7 152.1 152.3 153.1 157.2 163.8 170.8
 [25] 173.0 173.1 174.7 174.0 175.7 177.8 178.1 176.5 178.4 183.0 191.0 197.3
 [37] 199.8 199.9 201.3 201.6 203.1 208.4 210.1 210.0 210.1 215.6 223.7 230.2
 [49] 231.9 230.3 230.8 227.6 222.9 223.5 221.9 218.9 218.1 224.6 232.8 240.7
 [61] 243.5 243.4 247.5 248.2 248.8 252.1 252.6 249.4 251.6 262.1 273.9 284.8
 [73] 289.3 288.8 293.3 295.0 295.8 300.5 300.3 295.7 298.4 308.5 321.6 333.4
 [85] 336.2 334.9 337.8 337.6 336.4 340.2 337.6 331.8 333.5 344.5 360.6 373.9
 [97] 378.0 377.0 380.5 378.9 375.9 376.6 369.2 359.0 355.8 364.5 380.3 390.2
[109] 392.5 389.5 391.4 391.2 390.6 394.9 391.0 383.9 380.6 395.0 415.0 430.3
[121] 437.3 437.5 443.8 444.9 444.4 444.3 443.2 435.6 433.2 449.1 469.0 483.6
[133] 489.2 486.5 490.6
```

Приклад. Напишіть функцію, яка знаходить дільники натурального числа.

```
> divisors<-function(N){
+   d<-numeric(round(N/2))
+   for (i in 1:round(N/2)) if (N%%i==0) d[i]<-i
+ d<-d[d!=0]
+ d<-c(d,N)
+ d
+ }
> divisors(6)
[1] 1 2 3 6
> divisors(24)
[1]  1  2  3  4  6  8 12 24
```

Приклад. Використовуючи функцію з попереднього прикладу, знайдіть всі прості числа серед перших 1000 натуральних чисел.

```
> s<-numeric(999)
> for (i in 2:1000)
+   if (length(divisors(i))==2) s[i]<-i
> s<-s[s!=0]
> s
  [1]   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73
 [22]  79  83  89  97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 17
3 179 181
 [43] 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281
283 293 307
 [64] 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419
421 431 433
 [85] 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557
563 569 571
[106] 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677
683 691 701
[127] 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827
829 839 853
[148] 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977
983 991 997
```

Функції групи apply

- **apply(X,INDEX,FUN=)** — вертає вектор, масив або список, застосовуючи функцію **FUN** до певних елементів матриці або масиву;
  аргумент **MARGIN** вказує напрям застосування функції (чи до рядочків, чи до стовпчиків).
- **lapply(X,FUN)** — вертає список довжини об'єкта x; значення елементів списку будуть результатом застосування **FUN** до x.
- **tapply(X,INDEX,FUN=)** —застосовує функцію **FUN** до кожної групи об'єкта x, розбиття на групи відбувається у відповідності до рівнів факторів; перелік факторів вказується за допомогою аргументу **INDEX**.

Приклад. В змінній **my** збережено фрейм з довільною кількістю числових змінних. Знайдіть максимальне значення в кожному стовпчику (потім в рядочку). Збережіть результат у вигляді вектору.

```
> my<-data.frame(x1=rnorm(10,3,2),x2=rbinom(10,10,1/3),x3=c(-4:5))
> col_max <- apply(my, 2, max)
> col_max
      x1       x2       x3
4.716277 5.000000 5.000000
> row_max<-apply(my,1,max)
> row_max
 [1] 4.000000 3.000000 4.000000 4.000000 3.000000 4.000000 5.000000 4.00000
0
 [9] 4.716277 5.000000
```

Приклад. Напишіть функцію, яка заміняє всі пропущені значення в стовпчику фрейму на відповідне середнє значення стовпчика. Фрейм складається тільки з числових змінних.

```
> t_data <- as.data.frame(list(V1 = c(NA, NA, NA, NA, 13, 12, 9, 10, 8, 9, 11, 11, 1
0, 12, 9), V2 = c(NA, 12, 8, NA, 11, 11, 9, 8, 8, 10, 10, 11, 10, 10, 10), V3 = c(NA,
5, NA, 13, 12, 11, 11, 14, 8, 12, 8, 8, 10, 10, 8), V4 = c(10, 10, 10, 10, 13, 10, 11, 7,
12, 10, 7, 10, 13, 10, 9)))
> na_rm  <- function(x){
+   sm<-function(y){
+     y[which(is.na(y)==T)]<-mean(y,na.rm=T)
+     return(y)
+   }
+   x<-apply(x,MARGIN=2,sm)
+   return(x)
+ }
```

```
> na_rm(t_data)
          V1       V2 V3 V4
 [1,] 10.36364  9.846154 10 10
 [2,] 10.36364 12.000000  5 10
 [3,] 10.36364  8.000000 10 10
 [4,] 10.36364  9.846154 13 10
 [5,] 13.00000 11.000000 12 13
 [6,] 12.00000 11.000000 11 10
 [7,]  9.00000  9.000000 11 11
 [8,] 10.00000  8.000000 14  7
 [9,]  8.00000  8.000000  8 12
[10,]  9.00000 10.000000 12 10
[11,] 11.00000 10.000000  8  7
[12,] 11.00000 11.000000  8 10
[13,] 10.00000 10.000000 10 13
[14,] 12.00000 10.000000 10 10
[15,]  9.00000 10.000000  8  9
```

Якщо нам потрібна команда, яка відбирає тільки кількісні змінні у фреймі, то можна зробити це так:

data_new<- data[sapply(some_data, is.numeric)]

sapply(some_data, is.numeric) повертає вектор логічного типу, який використовується для індексації.

Результат команди:

```
> sapply(mtcars[c(1,3)], sd
)
      mpg      disp
 6.026948 123.938694
```

еквівалентний результату команди:

```
> apply(mtcars[c(1,3)], 2, sd)
      mpg      disp
 6.026948 123.938694
```

Оскільки кожен стовпчик фрейму є елементом списку, то функції **lapply** і **sapply** повертають результат застосування функції до кожного стовпчика. Але **apply** проводить операції саме над матрицями, тому, якщо застосувати **apply** до фрейму з різними типами даних, то R спочатку перетворить всі змінні до одного типу. Іноді це приводить до помилки.

Приклад.

```
> new<-mtcars
> new$gear<-factor(new$gear)
> sapply(new,is.numeric)
  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
 TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
TRUE
> apply(new, 2,is.numeric)
  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FA
LSE FALSE
```

Остання команда показує, що серед даних відсутні числові змінні. Це відбулося тому, що перед застосуванням функції **is.numeric**, дані були перетворені на матрицю, а всі змінні зведені до одного типу - факторного. Результатом і стало FALSE в кожному стовпчику.

З **sapply** і **lapply** такого не трапиться, оскільки функція застосовується до елементів стовпчиків як до змінних у списку.

Давайте розглянемо приклад для функції **mapply**.

Приклад. Нехай в нас є матриця розмірності 15 на 5 і ми хочемо назвати рядки і стовпчики таким чином:

Ряд_1, Ряд_2, ..., Ряд_15 - для рядків,

Стовпчик_1, Стовпчик_2, ..., Стовпчик_5 – для стовпчиків.

```
> m <- matrix(runif(15 * 5), nrow = 15)
> m_names <- mapply(paste, list("Ряд", "Стовпчик"), list(1:15, 1:5), sep = "_")
> row.names(m)<-m_names[[1]]
> colnames(m)<-m_names[[2]]
> m
       Стовпчик_1  Стовпчик_2  Стовпчик_3 Стовпчик_4 Стовпчик_5
Ряд_1  0.24115674 0.166701560 0.008462766  0.3580784 0.33218426
Ряд_2  0.29600726 0.348365102 0.681689504  0.2467823 0.83558090
Ряд_3  0.21932644 0.049343733 0.334435345  0.4050594 0.50355385
Ряд_4  0.77122401 0.398599699 0.638647173  0.5630563 0.53680637
Ряд_5  0.43472872 0.536972761 0.771794592  0.2191896 0.72284587
Ряд_6  0.80951978 0.326304783 0.387200515  0.6741809 0.27033343
Ряд_7  0.37746043 0.255583087 0.270373945  0.9579923 0.42444449
Ряд_8  0.42620528 0.896407681 0.969762856  0.2029768 0.38109052
Ряд_9  0.10185023 0.968536147 0.986812179  0.7541293 0.31923872
Ряд_10 0.16266059 0.148997330 0.202979652  0.7318692 0.28281939
Ряд_11 0.03739831 0.079221071 0.919916154  0.4157445 0.02714962
Ряд_12 0.97175361 0.006399704 0.657427575  0.8945952 0.68816512
Ряд_13 0.83090351 0.825109422 0.061049327  0.3284828 0.31659143
Ряд_14 0.05780640 0.858532029 0.223004733  0.0316720 0.10124213
Ряд_15 0.16496434 0.427707137 0.981504376  0.9770758 0.90092755
```

Розглянемо ще один нюанс застосування функції **sapply**. Напишемо функцію, яка буде обчислювати медіани для всіх числових змінних фрейму.

```
> get_m<- function(x){
+   num_x <- sapply(x, is.numeric)
+   sapply(x[, num_x], median)
+ }
```

Розглянемо як вона працює на різних даних.

Приклад.

```
> get_m(mtcars)
   mpg    cyl   disp     hp   drat     wt   qsec     vs     am   gear
19.200  6.000 196.300 123.000  3.695  3.325 17.710  0.000  0.000  4.000
  carb
 2.000
```

```
> t_data<-data.frame(x1=c(1:10),x2=factor(sample(c(0,1),10,replace=T)))
> get_m(t_data)
 [1]  1  2  3  4  5  6  7  8  9 10
```

Другий фрейм містить тільки одну кількісну змінну. Функція просто вивела всі значення числової змінної, а не знайшла медіану. Це відбулося тому, що ми неправильно звернулися до змінних фрейму.

frame[i] – одержуємо фрейм,

frame[[i]] –результатом буде вектор,

frame[, i] – одержимо вектор.

У випадку однієї кількісної змінної звернення x[, i] дасть нам вектор, а **sapply** застосує функцію до кожного елемента вектору, а не застосує до всієї змінної.

Тоді нашу функцію потрібно переписати у вигляді:

```
> get_median <- function(x){
+   num_x <- sapply(x, is.numeric
)
+   sapply(x[num_x], median)}
>  get_median(test_data)
 x1
5.5
```