程序控制结构

温灿红

中国科学技术大学管理学院

序言

- **R**是一种块状结构程序语言，块（block）由大括号划分
- 如果当块只包含一条语句时大括号可以省略
- 程序语句由换行符或者分开分隔
- 以下两种写法是等价的

```
{
      x <- 1
      x
}
```

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

```
x <- 1; x
```

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

大纲

- 分支结构（if-else,ifelse(),switch()）
- 循环结构（for,while,repeat）
- 向量化

分支结构

分支结构的if-else

分支结构包括 if 结构：

```
if(条件) {表达式}
```

和 if-else 结构

```
if(条件) {表达式1} else {表达式2}:
```

- “条件”必须是一个逻辑型标量，不能是 NA
- 例子：判别是否缺失，若是则赋值

```
x <- NA
if(is.na(x)) x <- 0.5
x
```

```
## [1] 0.5
```

if-else 结构之例子

- 例子：绝对值函数

$$|x| = \begin{cases} x & \text{if } x \geq 0, \\ -x & \text{if } x < 0 \end{cases}$$

```
    if (x >= 0) {
      x
    } else {
      -x
    }
```

```
## [1] 0.5
```

- 注意这里的 else 不能是一行的开头，不然**R**会错误的认为这是 if 结构而进行相关的操作。
- 如果表达式中只有一句表达式，那么可以不用大括号而把所有的东西放在一行中。如下面的例子：

```
    if (x >= 0) x else -x
```

```
## [1] 0.5
```

向量的判别： ifelse()

- 上述的绝对值判别不能适用于向量。下面的语句返回的是错误的结果：

```
x <- -2:2
if(x>=0) x else -x
```

- 函数 ifelse() 可以根据一个逻辑向量中的多个条件，分别选择不同结果。

```
ifelse(x >= 0, x, -x)
```

```
## [1] 0.5
```

多个判别表达式（一）

- 在判别算法是否收敛的时候，我们常常要求精度要达到一定的程度或者达到了最大的迭代次数，如

```
eps <- 1
iter <- 10
if (eps < 1e-3 | iter > 100 ) print("converged") else print("Not converged")
```

```
## [1] "Not converged"
```

- 但是如果需要判断的语句本身是一个向量，我们要求至少有一个条件满足即为TRUE

```
eps <- c(1, 0.5, 0)
if (eps < 1e-3 || iter > 100 ) print("converged") else print("Not converged")
```

```
## Warning in eps < 0.001 || iter > 100: 'length(x) = 3 > 1' in coercion to
## 'logical(1)'
```

```
## [1] "Not converged"
```

```
if (any(eps < 1e-3) | iter > 100 ) print("converged") else print("Not converged")
```

```
## [1] "converged"
```

多个判别表达式（二）

- 一般使用&&或（和）||来组合多个逻辑表达式，因为它们只返回一个逻辑型标量
- &&和||有着所谓的“短路效应”：
  - 只要||遇到第一个 TRUE，则返回 TRUE 而不再计算其他表达式
  - 只要 && 遇到第一个 FALSE，则返回 FALSE 而不再计算其他表达式
- &或|是向量化的操作符，作用于向量时会返回多个值。如果非要使用，则利用 any() 或 all() 将其转换为单个逻辑值。

```
(0 > 0) && (all.equal(42%%6, 169%%13))
```

```
## [1] FALSE
```

```
all.equal(42%%6, 169%%13)
```

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

多重 if-else 结构

- 多个分支，可在中间增加 else if，如：

$$\psi(x) = \begin{cases} x^2 & \text{if } |x| \leq 1 \\ 2|x| - 1 & \text{if } |x| > 1 \end{cases}$$

```
x <- 0.5
if (x^2 < 1) {
  x^2
} else if (x >= 1) {
  2*x-1
} else {
 -2*x-1
}
```

```
## [1] 0.25
```

多分枝结构： switch()

- 函数 switch() 可以建立多分枝结构，但不如 if-else 结构容易理解。

```
switch((x <= -1) + (x <= 1) + 1,
       2*x-1,
       x^2,
       -2*x-1
       )
```

```
## [1] 0.25
```

- switch() 函数更适合做因子的多分枝结构。

```
type.of.summary = "mode"

switch(type.of.summary,
       mean=mean(x),
       median=median(x),
       histogram=hist(x),
       "I don't understand")
```

```
## [1] "I don't understand"
```

循环结构

for 循环（一）

- for 循环的语法为

```
for (循环变量 in 序列) {表达式}
```

- 如果对向量每个元素遍历并保存结果，应在循环之前先将结果变量产生等长的存储，在循环内为已经分配好存储空间的输出向量的元素赋值。
- 为了产生长度为 n 的数值型向量，用 numeric(n)；为了产生长度为 n 的列表，用 vector("list", n)。

```
n = 10
log.vec = vector(length=n, mode="numeric")
for (i in 1:n) {
  log.vec[i] = log(i)
}
log.vec
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

for 循环（二）

- 如果要对向量元素遍历，建立采用下标访问，以避免出现丢失向量的属性问题

```
    x <- as.POSIXct(c("1981-05-31", "2020-02-22"))
    for(xi in x){
      print(xi)
      }
```

```
## [1] 360086400
## [1] 1582300800
```

```
    for(i in seq_along(x)){
      print(x[i])
      }
```

```
## [1] "1981-05-31 CST"
## [1] "2020-02-22 CST"
```

for 循环（三）

- 对一个向量元素遍历时如果用下标访问， 建议用 `seq_along(x)` 取代 `1:length(x)`，避免出现长度为零时出现错误下标

```
x <- 1:3   # 改成 x <- NULL 试试
for (i in seq_along(x)) {
  print(log(x[i]))
}
```

```
## [1] 0
## [1] 0.6931472
## [1] 1.098612
```

```
x <- 1:3   # 改成 x <- NULL 试试
for (i in 1:length(x)) {
  print(log(x[i]))
}
```

```
## [1] 0
## [1] 0.6931472
## [1] 1.098612
```

嵌套for 循环（一）

```
for (i in 1:4) {
  for (j in 1:i^2) {
    cat(paste(j,""))
  }
  cat("\n")
}
```

```
## 1
## 1 2 3 4
## 1 2 3 4 5 6 7 8 9
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

嵌套for 循环（二）

```
a <- matrix(1:4, nrow = 2)
b <- matrix(1:6, nrow = 2)
c <- matrix(0, nrow=nrow(a), ncol=ncol(b))
if (ncol(a) == nrow(b)) {
  for (i in 1:nrow(c)) {
    for (j in 1:ncol(c)) {
      for (k in 1:ncol(a)) {
        c[i,j] <- c[i,j] + a[i,k]*b[k,j]
      }
    }
  }
  print(c)
} else {
  stop("matrices a and b non-conformable")
}
```

```
##      [,1] [,2] [,3]
## [1,]    7   15   23
## [2,]   10   22   34
```

while 循环

- while 循环的语法为

```
while (循环继续条件) {表达式}
```

- 下面例子展示的是正整数中其log值不超过2的所有log值：

```
i = 1
log.vec = c()
while (log(i) <= 2) {
  log.vec = c(log.vec, log(i))
  i = i+1
}
log.vec
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
```

repeat 循环（一）

- repeat 循环的语法为

```
repeat{
  ...
   if(循环推出条件) break
}
```

- 等价于 while(TRUE)
- 下面例子展示的是正整数中其log值不超过2的所有log值：

```
i = 1
log.vec = c()
# while (TRUE) {
repeat {
  if (log(i)>2) break
  log.vec = c(log.vec, log(i))
  i = i+1
}
log.vec
```

```
## [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
```

repeat 循环（二）

- 小心陷入死循环！！！

```
repeat {
  ans = readline("请问《实用统计软件》课程的主讲老师的名字叫什么？")
  if (ans == "温灿红" || ans == "Wen Canhong") {
    cat("对了！你通过了此次考试！")
    break
  }
  else {
    cat("喔喔，你答错了！后果很严重哦！！！\n")
  }
}
```

向量化

避免循环——向量化（一）

- **R** 是解释型语言，在执行单个运算时， 效率与编译代码相近；在执行迭代循环时， 效率较低， 与编译代码的速度可能相差几十倍。
- **R** 以向量、矩阵为基础运算单元， 在进行向量、矩阵运算时效率很高， 应尽量采用向量化编程。
- **R** 中有很多函数都支持向量化运算，如 abs(), log(), sum(), prod(), cumsum() 等

```
abs(-2:2)
```

```
## [1] 2 1 0 1 2
```

```
log(1:10)
```

```
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
```

避免循环——向量化（二）

$$\psi(x) = \begin{cases} x^2 & \text{if } |x| \leq 1 \\ 2|x| - 1 & \text{if } |x| > 1 \end{cases}$$

- 第一种方法：利用向量化与逻辑下标

```
x <- seq(-1,1, 0.2)
y <- x^2
y[abs(x)>1] <- 2 * abs(x) - 1
y
```

```
##  [1] 1.00 0.64 0.36 0.16 0.04 0.00 0.04 0.16 0.36 0.64 1.00
```

- 第二种方法：利用 ifelse() 函数

```
ifelse(abs(x)<=1, x^2, 2 * abs(x) - 1)
```

```
##  [1] 1.00 0.64 0.36 0.16 0.04 0.00 0.04 0.16 0.36 0.64 1.00
```

向量化的例子（一）

考虑一个班的学生存在生日相同的概率。

- 假设一共有365个生日(只考虑月、日)。 设一个班有 n 个人，则

$$
\begin{aligned}
&P(\text{至少有两人相同生日}) \\
&= 1 - P(n\text{个人生日都不相同}) \\
&= 1 - \frac{365 \times 364 \times \cdots \times (365 - (n-1))}{365^n} \\
&= 1 - \frac{365-0}{365} \times \frac{365-1}{365} \times \cdots \times \frac{365-(n-1)}{365^n}
\end{aligned}
$$

向量化的例子（一）

- 普通循环写法

```
n <- 100                  # 班级学生数
p <- 1
for(i in 0:(n-1)){
  p <- p * (365-i)/365
}
1 - p
```

```
## [1] 0.9999997
```

- 向量化写法

```
1 - prod((365:(365-n+1))/365)
```

```
## [1] 0.9999997
```

向量化的例子（二）

```
a <- matrix(1:4, nrow = 2)
b <- matrix(1:6, nrow = 2)
c <- matrix(0, nrow=nrow(a), ncol=ncol
if (ncol(a) == nrow(b)) {
  for (i in 1:nrow(c)) {
    for (j in 1:ncol(c)) {
      for (k in 1:ncol(a)) {
        c[i,j] <- c[i,j] + a[i,k]*b[k,
      }
    }
  }
  print(c)
} else {
  stop("matrices a and b non-conformab
}
```

```
##      [,1] [,2] [,3]
## [1,]    7   15   23
## [2,]   10   22   34
```

- 等价于

```
a %*% b
```

```
##      [,1] [,2] [,3]
## [1,]    7   15   23
## [2,]   10   22   34
```

replicate() 函数—— for循环在模拟实验的向量化函数

- replicate() 函数用来执行某段程序若干次， 类似于 for() 循环但是没有计数变量。
- 常用于随机模拟。
- replicate() 的缺省设置会把重复结果尽可能整齐排列成一个多维数组输出。
- 语法为

```
replicate(重复次数, 要重复的表达式)
```

replicate() 函数例子

```
    replicate(6, {
      x <- rnorm(100, 0, 1);
      c(mean(x), sd(x)) })
```

```
##             [,1]        [,2]       [,3]       [,4]        [,5]       [,6]
## [1,] 0.08973216 -0.07528947 -0.1168647 -0.1506739 -0.02283174 -0.1581291
## [2,] 0.97789745  0.95788767  0.9404053  0.9782824  1.02023158  0.9880281
```

案例分析：蒙特卡洛积分

- 假设 $X_1, \ldots, X_n$ 是一组独立同分布的样本，其概率密度函数为 f ，则由大数定律可知当 $n \to \infty$ 时有

$$ \hat{\mu}_{MC} = \frac{1}{n} \sum_{i=1}^n h(X_i) \rightarrow \mu = E(h(X_1)) = \int h(x) f(x) dx $$

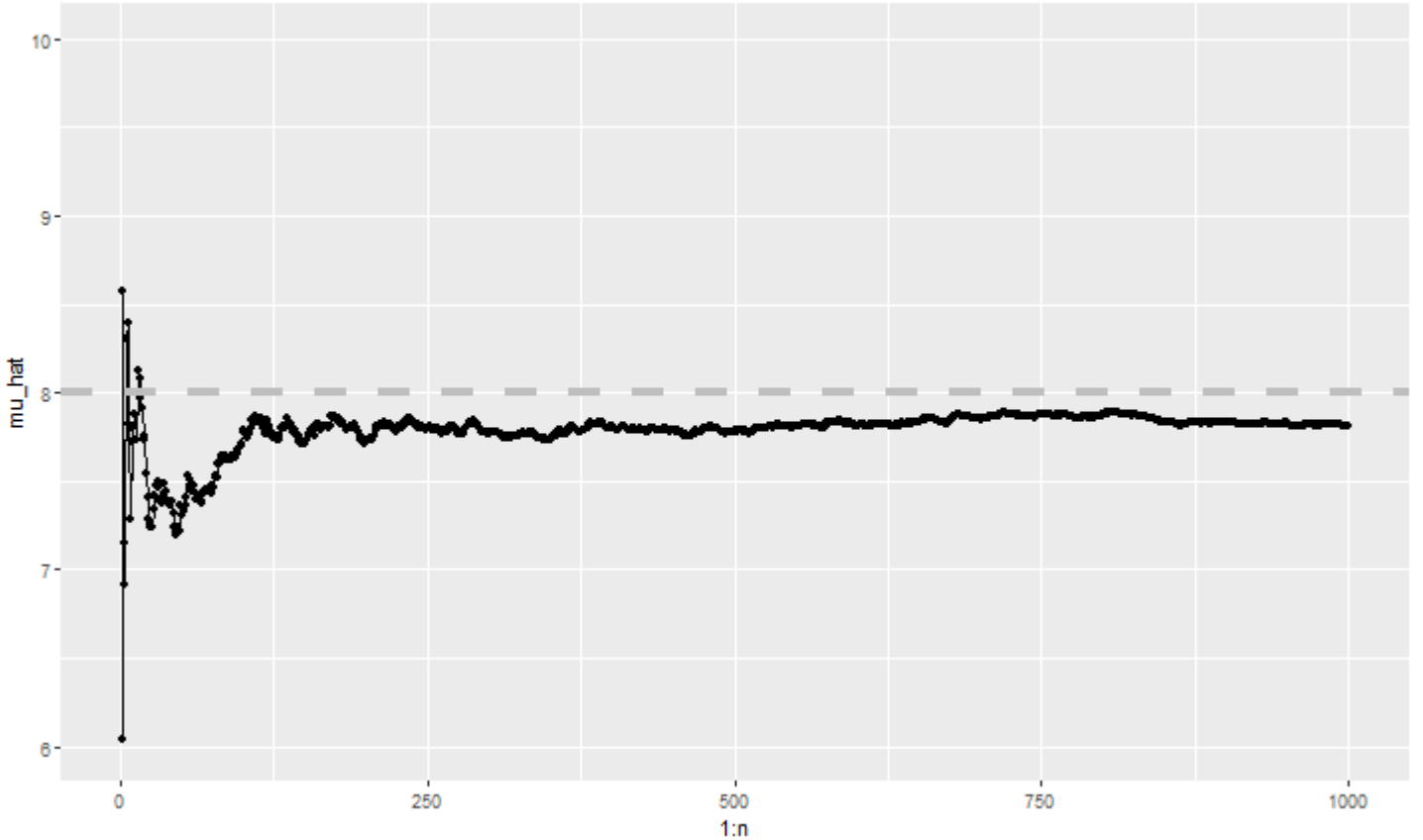
- 假设样本来自于参数为8的gamma分布，$h(x) = x$, 请分析当n增加时 $\hat{\mu}_{MC}$ 是如何变化的?

案例分析：蒙特卡洛积分

```
set.seed(123)
n <- 1000
x <- rgamma(n, 8) # h(x) = x
mu_hat <- (cumsum(x) / (1:n)) # Cumulative average

library(ggplot2)
ggplot(mapping = aes(1:n, mu_hat)) +
  geom_point() +
  geom_line() +
  geom_hline(aes(yintercept = 8), color = "gray", linewidth=1.5, linetype = "dashe
  coord_cartesian(ylim = c(6, 10))
```

案例分析：蒙特卡洛积分



案例分析：蒙特卡洛积分

- 置信区间（基于中心极限定理）

$$ \hat{\mu}_{MC} = \frac{1}{n} \sum_{i=1}^n h(X_i) \sim N(\mu, \sigma^2_{MC}/n) $$

其中

$$\sigma^2_{MC} = \int (h(x) - \mu)^2 f(x) dx.$$

上式可以通过以下式子进行估计

$$ \hat\sigma^2_{MC} = \frac{1}{n-1} \sum_{i=1}^n (h(X_i) - \hat{\mu}_{MC})^2 $$

那么 95%的置信区间为

$$ \hat{\mu}_{MC} \pm \frac{\hat\sigma_{MC}}{n}. $$

案例分析：蒙特卡洛积分

```
set.seed(123)
n <- 1000
x <- rgamma(n, 8) # h(x) = x
mu_hat <- (cumsum(x) / (1:n)) # Cumulative average
sigma_hat <- sd(x)

library(ggplot2)
ggplot(mapping = aes(1:n, mu_hat)) +
  geom_ribbon(
    mapping = aes(
      ymin = mu_hat - 1.96 * sigma_hat / sqrt(1:n),
      ymax = mu_hat + 1.96 * sigma_hat / sqrt(1:n)
      ), fill = "gray") +
  geom_point() +
  geom_line() +
  geom_hline(aes(yintercept = 8), color = "red", linewidth=1.5, linetype = "dashed
  coord_cartesian(ylim = c(6, 10))
```

案例分析：蒙特卡洛积分