R 语言基础函数

2026 年 1 月 8 日

常量

常量是直接写在程序中的值，包括数值、字符串等。

- **数值型常量:** 123, 123.45
- **字符型常量:** 使用双引号或单引号包围 ("Li Ming")
- **逻辑型常量:** 只有 `TRUE` 和 `FALSE`
- **缺失值:** 用 `NA` 表示，常见于数据缺失情况。
- `NaN` (Not a Number): 表示未定义或不可表示的数值，如 0 除以 0。
- `Inf`: 表示正无穷或负无穷，通常在除以零时出现。

变量

变量用于保存输入值或计算结果。

- **变量名规则:**
  - 由字母、数字、下划线和句点组成，不能以数字开头。
  - 变量名区分大小写，如 `y` 和 `Y` 是不同的变量。
  - 示例: `x`, `x1`, `X`, `cancer.tab`
- **赋值:** 使用 `<-` 或 `=` 定义变量。

```
1 x5 <- 6.25
2 x6 = sqrt(x5)
```

重要变量类型：逻辑型

逻辑型变量用于存储真（TRUE）和假（FALSE）的值。在数值计算中，逻辑变量值 *TRUE* 自动转换为 *1*，*FALSE* 自动转换为 *0*。

```
TRUE + TRUE          # 求和
## [1] 2
a <- TRUE            # 赋值
is.logical(a)        # 检查类型
## [1] TRUE
class(a)              # 查看变量类型
## [1] "logical"
```

重要变量类型：数值型

数值型变量用于保存数值数据。

```
a <- 23           # 将23赋值给变量a
is.numeric(a)     # 检查是否为数值型
## [1] TRUE

is.logical(a)     # 检查是否为逻辑型
## [1] FALSE

class(a)          # 查看变量类型
## [1] "numeric"
```

重要变量类型：字符型

字符型变量用于保存文本数据。

```
a <- "My"              # 将"My"赋值给a
a                      # 显示a的值
## [1] "My"

is.character(a)        # 检查是否为字符型
## [1] TRUE

class(a)               # 查看变量类型
## [1] "character"
```

向量

向量是将若干个基础类型（数值型、字符型或逻辑型）相同的值存储在一起的结构。各个元素可以通过序号进行访问。
向量的创建方法（1）- **组合功能函数**:

```
# 数值型向量
x1 <- c(1, 2) # 创建第一个数值型向量
x2 <- c(3, 4) # 创建第二个数值型向量
x <- c(x1, x2) # 合并向量
x
## [1] 1 2 3 4
```

向量的创建

方法（2）：冒号生成数值序列

表达式"a:b" 从 a 到 b（步长为 1）。默认生成整数序列。**示例：**

```
> 1:6              # 1 到 6
[1] 1 2 3 4 5 6
> 4:7.6            # 整数部分
[1] 4 5 6 7
> 1.2:5            # 从 1.2 到 5
[1] 1.2 2.2 3.2 4.2
```

向量的创建

方法（3）：Seq() 函数

`seq(from, to, by, length.out)`

- `from`: 向量的起始值。
- `to`: 向量的结束值。
- `by`: 步长，默认值为 1。可为负数以生成递减序列。
- `length.out`: 指定生成的向量长度，自动计算步长。

访问和修改向量元素

正整数下标：使用方括号访问向量元素。**示例**：

```
x <- c(1, 4, 6.25)
x[2]              # 取出第二个元素
[1] 4
x[2] <- 99        # 修改第二个元素
print(x)
[1]  1.00 99.00  6.25
```

R 矩阵

矩阵：二维数组。**向量**可以视为特殊的矩阵（行数为 1 或列数为 1 的矩阵）。**注意**：每个元素必须为相同的数据类型（数值型、字符型或逻辑型）。使用 `matrix()` 函数定义矩阵。示例如下：

```
A <- matrix(11:16, nrow=3, ncol=2)
print(A)
     [,1] [,2]
[1,]   11   14
[2,]   12   15
[3,]   13   16
```

矩阵的定义

- matrix() 函数接受一个向量作为矩阵元素。
- nrow 和 ncol 指定行数和列数。
- 默认按列填充，可使用 byrow=TRUE 按行填充。

```
B <- matrix(c(1, -1, 1, 1),
      nrow=2, ncol=2, byrow=TRUE)
print(B)
      [,1] [,2]
[1,]    1   -1
[2,]    1    1
```

访问矩阵的元素

使用下标访问矩阵元素：

- `A[1,]`：取出第一行，返回普通向量。
- `A[,1]`：取出第一列，返回普通向量。
- `A[c(1, 3), 1:2]`：取出指定行和列的子矩阵。

```
A <- matrix(11:16, nrow=3, ncol=2)
A[1,]                 # 取出第一行
[1] 11 14
A[,1]                 # 取出第一列
[1] 11 12 13
# 取出第1和第3行的前2列
A[c(1, 3), 1:2]
     [,1] [,2]
[1,]   11   14
[2,]   13   16
```

矩阵元素的修改

使用下标可以修改矩阵的元素：

- 使用 `A[i, j]` 访问第 i 行第 j 列的元素。
- 直接赋值可以修改该元素的值。
- 可以使用下标向量同时修改多个元素。

```
A[2, 1] <- 99 # 修改单个元素
A
     [,1] [,2]
[1,]   11   14
[2,]   99   15
[3,]   13   16
```

数据框

数据框是 R 语言中一种常见的数据结构, 用于存储表格数据。

- 由 n 行和 p 列组成。
- 各列可以包含不同类型的数据
- 同一列中的数据类型必须相同。

使用 `data.frame()` 函数来创建数据框。所有输入变量必须是相同维数的向量。

示例代码:

```
x <- c(160, 175)
y <- c(51, 72)
sex <- c("Female", "Male")
myData <- data.frame(x, y, sex)
class(myData)
[1] "data.frame"
```

在 R 中创建数据框并设置变量名称

可以在创建数据框时指定变量的名称。

```
myData <- data.frame(height = x,
                     weight = y, sex = sex)
myData
  height weight    sex
1    160     51 Female
2    175     72   Male
```

在 R 中提取数据框的数据

可以通过列变量名称提取相应的数据。例如:

```
# 提取身高数据
heights <- myData$height
heights
[1] 160 175
# 提取性别数据
genders <- myData$sex
genders
Levels: Female Male
```

在 R 中查阅数据框列名称

查阅列名称

在 R 语言中，可以通过 names() 函数查阅数据框各列的名称。以下是示例代码：

```
myData <- data.frame(x, y, sex)
#  查阅各列名称
names(myData)   #显示myData各列的名称
[1] "x"  "y"  "sex"
```

在 R 中修改数据框列名称

修改列名称我们可以使用 names() 函数修改数据框的列名称:

```
# 重新命名myData的列名称
names(myData) <- c("h", "w", "Sex")
# 显示当前各列名称
names(myData)
[1] "h"  "w"  "Sex"
```

现在，数据框 myData 的列名称已更改为 h, w 和 Sex。

在 R 中索引数据框内容

在 R 语言中，可以像数组和向量一样通过方括号索引或修改数据框中特定位置的内容。

```
1 myData <- data.frame(height=x,
2                      weight=y,sex=sex)
```

- `myData[1, 1]` 和 `myData$height[1]` 是等价的，它们都代表数据框 `myData` 第 1 行和第 1 列交叉位置的变量值。
- `myData[, 2]` 和 `myData$weight` 也是等价的，它们都表示数据框 `myData` 的第 2 列。

列表的定义

列表可以看成向量结构的另一种推广。它允许其各个分量是任意的 R 语言结构，例如：

- 向量
- 矩阵
- 数据框
- 其他列表

列表结构能够将不同的对象以简单的形式组合在一起，方便编程者调用。

- 列表中的每个元素可以是不同类型的 R 对象，具有更大的灵活性。
- 列表元素可以通过索引或名称进行访问。
- 许多 R 语言函数的计算结果都是以列表的方式表达的。

生成列表

在 R 语言中，可以通过函数 `list()` 来生成列表数据。以下是示例代码：

```
# 生成一个列表
x <- list(u = 2, v = "abcd")  # 列表有两个分量，分别为
x # 显示列表x的内容
$u
[1] 2
$v
[1] "abcd"
```

索引列表分量

在 R 语言中，可以通过 [[i]] 或 $ 来索引和修改列表的分量。

```
# 列表有两个分量：u 和 v
x <- list(u = 2, v = "abcd")
# 索引x的第一分量
x$u     # 或者使用 x[[1]]
[1] 2
# 索引x的第二分量
x[[2]]
[1] "abcd"
# 将x的第一分量修改为1:3
x$u <- 1:3   # 修改为向量
# 显示x的第一分量的内容
x[[1]]
[1] 1 2 3
```

names 函数

在 R 语言中，函数 names 不仅能用于数据框，也能用于列表，查阅或修改各个分量的名称。

```
# 创建列表
x <- list(u = 2, v = "abcd")
names(x)
[1] "u"      "v"
names(x) <- c("num", "string")
names(x)
[1] "num"      "string"
```

函数

为了方便使用者，R 语言将具有特定功能的程序代码封装在函数中。下面简单介绍一些常用函数的功能和使用方法。

函数 sum 计算向量的各个分量之和，矩阵的各行（列）元素之和使用函数 rowSums 和 colSums。

```
# 计算向量c(1, 2, 3)的各分量之和
sum(c(1, 2, 3))  # 结果为 6
a <- matrix(1:6, 2, 3, byrow = TRUE)
# 计算 a 的各列之和，结果为3维行向量
b <- colSums(a)
# 计算 a 的各行之和，结果为2维行向量
c <- rowSums(a)
```

绘图功能

R 语言提供了方便的制图功能，可以绘制任何函数的图像。函数 plot 提供了一种绘制平面图形的功能，可以通过问号 ? 来获取该函数的在线帮助。

```
# 创建从0到10的数值序列，步长为0.1
x <- seq(0, 10, 0.1)
# 计算x的正弦值
y <- sin(x)
# 绘制函数曲线
plot(x,y,type="l",main="y=sin(x)",
xlab = "x", ylab = "y")
```

关系运算符

在 R 语言中，关系运算符的计算结果是逻辑数据，表明参与运算的两个变量是否满足特定的关系。不仅两个数之间可以进行关系运算，向量和数之间、矩阵和数之间也可以进行关系运算。

```
# 将向量 1:5 赋值给 X
X <- 1:5
# 判断 X 的各个分量是否小于或等于 4
X <= 4
[1] TRUE TRUE TRUE TRUE FALSE
```

关系运算符的应用

可以利用关系运算符显示向量中满足特定条件的分量，或将这些分量统一改为特定的值。

```
X <- 1:5
# 显示 X 中小于 3 的分量构成的子向量
X[X < 3]
[1] 1 2
# 将 X 中小于 3 的分量都改为 8
X[X < 3] <- 8
X    # 再次显示 X 的结果
[1] 8 8 3 4 5
```

离散均匀分布在 R 语言中的模拟

可以使用函数 sample 来模拟离散均匀分布随机数。其简单调用方式为:

```
sample(x, # 等可能地选择 x 的分量
size,  # size为要选取的变量的个数
replace = FALSE)
# replace为真, 有放回抽样;
#否则, 无放回抽样
```

逻辑运算符

逻辑运算符用于逻辑变量的运算。常用的逻辑运算符及其含义如下：

- !：非运算
- &：与运算
- |：或运算

```
!TRUE     # 非运算结果为 FALSE
!FALSE    # 非运算结果为 TRUE
TRUE & TRUE    # 与运算结果为 TRUE
TRUE & FALSE   # 与运算结果为 FALSE
FALSE & FALSE  # 与运算结果为 FALSE
TRUE | TRUE    # 或运算结果为 TRUE
TRUE | FALSE   # 或运算结果为 TRUE
FALSE | FALSE  # 或运算结果为 FALSE
```

R 语言中的条件语句

在 R 语言中，常用的条件控制语句是 `if` 语句，它有三种主要形式。

- **(1) 单条件 (if)**：基础判断。
- **(2) 复合条件 (if/else)**：二选一判断。
- **(3) 多条件 (if/else if/else)**：多重分支判断。

单条件语句 (if)

调用格式:

```
if (条件表达式) {
    # 仅当条件表达式为 TRUE 时执行
    程序代码
}
```

执行过程：若“条件表达式”成立（结果为 TRUE），则执行“程序代码”；否则，跳过执行。

复合条件语句 (if/else)

调用格式:

```
if （条件表达式）{
    程序代码1   # 条件成立时执行
} else {
    程序代码2   # 条件不成立时执行
}
```

注意事项:

- else 必须紧跟在 if 代码块的右花括号 } 之后，并在同一行。

多条件语句 (if/else if/else)

调用格式:

```
if (条件表达式1) {
    程序代码1
} else if (条件表达式2) {
    程序代码2
} else {
    程序代码3   # 以上条件都不满足时执行
}
```

执行过程：程序依次检查条件表达式，一旦发现某个条件成立，则执行相应的程序代码，** 然后跳出整个结构 **。

循环语句概述

循环语句用于重复执行一段程序代码，是实现迭代和批量操作的重要手段。R 语言主要提供两种循环语句：

- **for 循环**：适用于已知重复次数或需要遍历向量元素的场景。
- **while 循环**：适用于重复次数不确定，依赖条件表达式来控制终止的场景。

for 循环语句

调用格式：

```
for(i in 向量v) {
    # 循环体 expr
}
```

执行过程：

1. 循环变量 i 依次取向量 v 中的每一个元素值。
2. 每取一个值，就执行一次循环体内的程序代码。

while 循环语句

调用格式:

```
while(条件表达式) {
    循环体
    # 确保循环体内有能改变条件表达式的代码，防止死循环
}
```

执行过程:

1. 计算条件表达式。
2. 若为 TRUE，则运行循环体，然后返回第 1 步。
3. 若为 FALSE，则结束循环。

函数 (Function) 定义

函数是封装代码逻辑，实现代码重用和模块化的重要工具。

```
name <- function(arg1, arg2, ...) {
    # 函数体：程序代码
    return(result) # 可选，但推荐使用
}
```

- **name**：自定义函数名。
- **arg1, ...**：函数的输入参数。
- **return(result)**：显式指定函数返回值。若省略 return，默认返回函数体中最后一行代码的计算结果。

定积分计算：integrate 函数

R 语言通过内置的 integrate 函数进行一元函数的数值积分。

计算定积分 $\int_0^1 x^2 \, \mathrm{d}x$:

```
# integrate 的第一个参数是一个函数对象
result <- integrate(function(x) x^2,
                    lower = 0, upper = 1)
print(result)
```

运行结果：

$$\$value = 0.3333333 \text{ with absolute error } < 3.7 \times 10^{-15}$$

注意：积分的返回值是一个包含结果和误差信息的列表。

概率分布函数概述 (d/p/q/r)

在 R 中，大多数概率分布函数都遵循 **d/p/q/r** 的命名规则：

- `dxxx(x)`: **D**ensity / **D**istribution function (概率密度或概率质量函数)。
- `pxxx(q)`: **P**robability function (累积分布函数 CDF)，表示 $P(X \leq q)$。
- `qxxx(p)`: **Q**uantile function (分位数函数)，求解 $F(x) = p$ 的 x 值。
- `rxxx(n)`: **R**andom number generation (随机数生成函数)。

(xxx 为分布的缩写，如 'norm', 'binom', 'unif' 等)

二项分布（Binomial）示例

- dbinom(x, size, prob)
- pbinom(q, size, prob)
- rbinom(n, size, prob)

```
> dbinom(5, size = 10, prob = 0.5)
# P(X=5) = 0.246
> pbinom(5, size = 10, prob = 0.5)
# P(X<=5) = 0.623
> rbinom(2, size = 10, prob = 0.5)
# [1] 7 8 (示例值)
```

正态分布（Normal）示例

- dnorm(x, mean, sd)
- pnorm(q, mean, sd)
- rnorm(n, mean, sd)

```
> dnorm(0.5, mean = 0, sd = 1)
# 0.3520...
> pnorm(1.96, mean = 0, sd = 1)
# P(Z<=1.96) = 0.975
> rnorm(3, mean = 0, sd = 1)
# [1] -0.62... 1.03... 0.70...
```

蒙特卡罗方法简介

定义

蒙特卡罗 (Monte Carlo) 方法是一种以概率统计理论为指导的数值计算方法。它使用随机数（或伪随机数）来解决计算问题。

- **核心思想**：当所求解的问题是某随机事件出现的概率，或者是某随机变量的数学期望时，通过“实验”的方法得出频率，以此逼近概率或期望。
- **应用场景**：数值积分, 物理模拟等。

蒙特卡罗估计方法小结

我们可以通过生成随机样本来估计以下各类统计量：

- **期望值** $\mathrm{E}[X]$：直接计算样本均值 $\bar{x}$。
- **概率值** $\mathrm{P}(A)$：计算事件发生的频率（示性函数的均值）。
- **分布函数** $F(x)$：计算 $X_i \leq x$ 的经验比例（Glivenko-Cantelli 定理保证了收敛性）。
- **积分** $\int g(x)\,dx$：转化为期望形式进行估计。

蒙特卡罗积分：原理

目标：计算定积分 $\theta = \int_a^b g(x)\,dx$。

方法：引入在 (a, b) 上的均匀分布随机变量 $X \sim U(a, b)$，其概率密度函数为 $f(x) = \frac{1}{b-a}$。

我们可以将积分改写为期望的形式：

$$\theta = \int_a^b g(x)\,dx = (b-a)\int_a^b g(x)\,\underbrace{\frac{1}{b-a}}_{f(x)}\,dx = (b-a)\mathrm{E}[g(X)]$$

估计量：

$$\hat{\theta} = \frac{b-a}{n}\sum_{i=1}^{n} g(X_i), \quad X_i \sim U(a, b)$$

直方图 (Histogram)

直方图用于展示**连续变量**的分布情况。使用 `hist()` 函数即可绘制。

```
hist(x, breaks="Sturges", freq=FALSE)
```

参数	含义
`x`	数据向量（必须为数值型）
`breaks`	设置分组边界 (1) 默认为"Sturges"（自动计算最适组距） (2) 指定向量：自定义组边界点
`freq`	纵轴显示设置 `TRUE` (默认)：显示**频数** (Frequency) `FALSE`：显示**频率/密度** (Density)

分类变量的频数统计

对于分类变量（因子），可以使用 table() 函数统计各类别的频数。

```
# 创建示例数据
x <- c("女", "男", "女", "女", "女", "男", "女", "男",
       "女", "女", "男", "男", "女", "男", "女", "女",
       "女", "女", "男", "男", "男", "男", "男")
y <- as.factor(x)

u <- table(y)      # 计算频数
print(u)
# y
# 男  女
# 12  17
```

使用 addmargins() 添加合计行：

```
addmargins(u)
# y    女   男   Sum
```

分类变量的频率统计

使用 prop.table() 计算比例（频率）：

```
prop.table(u)
#  男         女
#  0.4137931  0.5862069
```

结合 addmargins() 显示合计频率：

```
v <- prop.table(u)
addmargins(v)
#  男         女         Sum
#  0.4137931  0.5862069  1.0000000
```

分类数据的条形图 (Bar Plot)

条形图和饼图常用于展示分类数据。在 R 中使用 `barplot()` 绘制条形图。

```
# 绘制频数条形图
barplot(u)
# 绘制频率条形图
barplot(v)
```

饼图 (Pie Chart)

饼图通过扇形面积比例展示各分类变量的频率。使用 `pie()` 函数绘制。

```
print(u)
#  y
#  男  女
#  12  17

pie(u)
```

多变量条形图

展示两个分类变量的关系时，常用的条形图类型：

1. **堆积条形图 (Stacked Bar Plot)**：分为等高（展示比例）和非等高（展示总量）。
2. **并列条形图 (Dodged Bar Plot)**：便于直接比较各组数值。

数据准备：创建列联表

首先将数据存储为矩阵或 table 对象：

```r
# 创建矩阵数据
x <- matrix(c(12, 19, 17, 21), nrow=2, ncol=2)
colnames(x) <- c("男", "女")          # 列名：性别
rownames(x) <- c("文科", "理科")      # 行名：专业

y <- as.table(x)  # 转换为 table 对象
print(y)
```

	男	女
文科	12	17
理科	19	21

表: 变量 y 的内容

绘制等高堆积条形图

等高堆积条形图主要用于比较**结构比例**（即条件概率）。

给定学科时的性别比例

```r
# prop.table(..., margin=2) 表示按列计算比例（列和为 1）
barplot(prop.table(t(y), margin=2))
```

给定性别时的学科比例

```r
barplot(prop.table(y, margin=2))
```

条件概率计算详解

函数 `prop.table(data, margin)` 用于计算比例：

- `margin=1`：按行计算（行和为 1）。
- `margin=2`：按列计算（列和为 1）。

示例结果：

`prop.table(t(y), margin=2)`

	文科	理科
男	0.41	0.48
女	0.59	0.52

`prop.table(y, margin=2)`

	男	女
文科	0.39	0.45
理科	0.61	0.55

注：通过 `help(prop.table)` 可查看更多用法。

普通堆积条形图

普通堆积条形图既能展示各组内部的比例结构，也能展示样本总量的差异。直接使用原始频数数据 y 进行绘制，无需计算比例。

```
barplot(t(y))
barplot(y)
```

并列条形图 (Dodged)

并列条形图将各类别的柱子并排显示，便于直接比较数值大小。关键参数：`beside = TRUE`。

代码示例：

```
1
2 barplot(t(y), beside = TRUE)    # 关键参数：并列显示
3 barplot(y, beside = TRUE)
```

集中趋势的分析

```
1、样本均值  mean()
2、样本中位数 median()
3、样本分位数 quantile(x,a,type=5)
type在1:9取值，代表9种计算分位数的方法
4、众数
#连续型变量样本数据的众数
x=runif(100,5,10)
hist(x)
y <- hist(x,plot=F) #计算x的分组数据统计结果
maxID <- which.max(y$counts) #计算最大频数所在区间序号
mean(y$breaks[maxID:(maxID+1)])#计算第maxID区间的中心
#离散型变量样本数据的众数
x=sample(1:10,100,T)
y <- table(x) #计算x的分组数据统计结果
names(which.max(y)) #计算最大频数
```

集中趋势的分析

```
5.方差： var(x)
x<-c (42,55,64,70,75,78,80,82,82,82,
 85,85,85,85,88,90,90,92,95,99)
y<-c(39,52,61,68,72,76,77,78,79,78,
 83,83,81,81,85,87,86,91,91,98)
var(x)
var(y)
6.标准差： sd (x)
sd(x)
7.标准化 scale(x)
```

QQ 图

使用 `qqnorm` 和 `qqline` 绘制正态 QQ 图。当样本来自正态分布时，散点应近似分布在一条直线附近。

QQ 图的做法如下：设有 n 个观测值 $y_1, y_2, \ldots, y_n$，从小到大排列。对于样本来自 $N(\mu, \sigma^2)$，有：

$$y_i \approx \mu + \sigma x_i$$

其中 x_i 是标准正态分布的分位数。

R 代码示例：

```
qqnorm(x)
qqline(x, lwd=2, col='blue')
```

散点越集中在对角线附近，数据越可能服从正态分布。

盒形图

```
x <-c(42,55,64,70,75,78,80,82,82,82,85,85,85,85,85,88
,90,90,92,95,87)
fivenum(x) #计算五数概括：四分位数，最小值，最大值
boxplot(x,range=1.5,horizontal=F,ylab="分数")
```

其中：range: 默认 1.5，表示上、下两条须线的长度不超过四分位间距（上四分位数-下四分位数）的 1.5 倍 horizontal=T，则为水平的盒形图 x: 也可以是数据框，对其中的每一个变量画盒形图

盒形图

使用 boxplot 函数可以在同一坐标系下绘制来自不同总体的样本数据的盒形图，方便比较它们的分布特征和离群数据情况。只需将不同的样本数据放入一个数据框，然后将该数据框作为 boxplot 的输入变量。

例如，当数据框 xy 的各列为不同总体的样本数据时，使用以下代码绘制盒形图：

```
boxplot(xy)
例：
xy <- data.frame("甲班"=x, "乙班"=y)
boxplot(xy, range=1.5, ylab="分数")
```

apply 函数

```r
x <- matrix(rnorm(1000*5),1000,5)
#生成1000组容量为5的样本，
每个样本都服从标准正态分布，排列成1000行5列的矩阵。
apply(x,MARGIN=1,FUN=mean)
#对x的每一行求均值
```

apply 函数能够极大减少 for 循环的使用:

```r
apply(X, MARGIN, FUN, ...)
```

- x: 运算对象：数组、矩阵、数据框，数据至少是二维的
- MARGIN: 取 1，2，c(1,2) 分别表示对运算对象的行、列、同时对二者运算
- FUN：指定运算函数，如 mean 表示求均值

最大似然估计

R 函数 optim()、nlm()、optimize() 可以用来求函数极值，因此可以用来计算最大似然估计。optimize() 只能求一元函数极值。
正态分布最大似然估计有解析表达式。作为示例，用 R 函数进行数值优化求解。

对数似然函数为：

$$\ln L\left(\mu, \sigma^2\right) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln\sigma^2 - \frac{1}{2\sigma^2}\sum\left(X_i - \mu\right)^2$$

定义 R 的优化目标函数为上述对数似然函数去掉常数项以后乘以，求其最小值点。

最大似然估计

目标函数为：

```
objf.norm1 <- function(theta, x){
  mu <- theta[1]
  s2 <- exp(theta[2])
  n <- length(x)
  res <- n*log(s2) + 1/s2*sum((x - mu)^2)
  res
}
```

其中 θ_1 为均值参数 μ θ_2 为方差参数 σ^2 的对数值。x 是样本数值组成的 R 向量。

optim 函数

可以用 optim 函数来求极小值点。下面是一个模拟演示:

```
n<-30
mu0 <- 20
sigma0 <- 2
set.seed(1)
x <- rnorm(n, mu0, sigma0)
theta0 <- c(0,0)
ores <- optim(theta0, objf.norm1, x=x)
print(ores)
theta <- ores$par
mu <- theta[1]
sigma <- exp(0.5*theta[2])
```

optimize()

函数 optimize() 进行一元函数数值优化计算。例如，在一元正态分布最大似然估计中，在对数似然函数中代入 $\mu = \bar{x}$ ，目标函数：

$$h\left(\sigma^2\right) = \ln\left(\sigma^2\right) + \frac{1}{\sigma^2}\sum\left(X_i - \bar{X}\right)^2$$

下面的例子模拟计算 σ^2 的最大似然估计：

```
n<-30
mu0 <- 20;  sigma0 <- 2;  set.seed(1)
x <- rnorm(n, mu0, sigma0); mu <- mean(x)
ss <- sum((x - mu)^2)/length(x)
objf <- function(delta, ss) log(delta) + 1/delta*ss
ores <- optimize(objf, lower=0.0001,
                 upper=1000, ss=ss)
delta <- ores$minimum;sigma <- sqrt(delta)
print(ores)
```

optimize()

```
## $minimum
## [1] 3.30214
##
## $objective
## [1] 2.194568
##
## 真实sigma= 2 公式估计sigma= 1.817177
## 数值优化估计sigma= 1.817179
```

该函数的计算结果会存在列表中，是有两个分量的列表变量:

```
#第一个分量$maximum: 最优的参数取值
#第二个分量$objective: 最优（最大或最小）的函数值
```

未知总体方差情况下的均值检验

- 对于双边假设检验问题 (2)，可以使用 R 语言中的 ‘t.test‘ 函数：

```
h <- t.test(x, alternative = "two.sided", mu = mu0)
pValue <- h$p.value
```

- 对于单边假设检验问题 (3)，使用以下 R 代码：

```
h <- t.test(x, alternative = "greater", mu = mu0)
pValue <- h$p.value
```

- 对于单边假设检验问题 (4)，使用以下 R 代码：

```
h <- t.test(x, alternative = "less", mu = mu0)
pValue <- h$p.value
```

总体方差未知双正态总体均值的 t 检验

- 对于双边假设检验问题:

```
h <- t.test(x, y, alternative = "two.sided")
pValue <- h$p.value
```

- 对于单边假设检验问题 $H_0 : \mu_1 \geqslant \mu_2$:

```
h <- t.test(x, y, alternative = "greater")
pValue <- h$p.value
```

- 对于单边假设检验问题 $H_0 : \mu_1 \leqslant \mu_2$:

```
h <- t.test(x, y, alternative = "less")
pValue <- h$p.value
```

线性回归模型求解函数 lm

- 在 R 语言中，使用函数 ‘lm‘ 计算线性回归模型中参数 β 的最小二乘估计。
- 函数 ‘lm‘ 的结果是一个特殊结构的列表，称为 ‘lm‘ 型数据。
- 该列表包含参数 β 的最小二乘估计的相关结果。

```
1 S <- lm(y ~ x)
```

- ‘lm‘ 型变量有多个分量，其中最常用的包括：
  - ‘coefficients‘：模型参数的估计结果
  - ‘residuals‘：残差估计结果

提取模型参数和估计结果

- 在得到 ‘lm‘ 型变量后，可以使用 R 函数 ‘coef‘ 和 ‘predict‘ 提取模型参数和响应变量的估计结果。
- 示例代码：

```
# 生成线性回归模型
myReg <- lm(y ~ x)

# 提取参数估计结果
coef(myReg) # 提取 lm 列表 myReg 中的参数估计结果

# 计算响应变量的估计结果
predict(myReg) # 计算响应变量的估计结果
```

- 其中 ‘myReg‘ 存储的是函数 ‘lm‘ 的计算结果。

Thanks!