統計学 2：R 入門

矢内　勇生

2018-04-08 (改訂：2019-04-12, 2020-08-15)

このページでは、Rの基本的な使い方を解説する。 RとRStudioのインストールについては、以下の資料を参照されたい。

- Linux (Ubuntu) 編 (https://yukiyanai.github.io/jp/resources/docs/install-R_ubuntu.pdf) (PDF, 4.6MB)
- macOS 編 (https://yukiyanai.github.io/jp/resources/docs/install-R_macOS.pdf) (PDF, 4.9MB)
- Windows 編 (https://yukiyanai.github.io/jp/resources/docs/install-R_windows.pdf) (PDF, 5.8MB)

Rの基礎

RのコードはRのConsoleに直接入力してもいいし、スクリプトに保存してRから呼び出してもよい。 スクリプトを使うときは、スクリプトファイルを**file_name.R** のように **.R** ファイルとして保存する。そして、RのConsole で `source("file_name.R")` とすれば、スクリプト全体が実行される。

Rコマンドの区切りは改行である。改行すれば、1つのコマンドが終了したと認識される。 ただし、括弧が閉じていなかったり、行末に二項演算子 (+ など) があるときは、コマンドが次の行まで続いていると認識される。1行に複数のコマンドを書きたいときは ; で区切る。

Rは大文字と小文字を区別する。したがって、Var1 と var1 は異なる変数として認識される。 変数名は英数字と_ [アンダースコア]のみで構成するべきである（日本語も使えるが、トラブルの元なので避けるべき）。ただし、頭文字に数字は使えない。

スペースは1つ以上ならいくつあっても1つのスペースがある場合と同じである。また、演算子の前後のスペースはあってもなくてもよい（コードの読み易さを考えてスペースの有無を決めること）。

はコメントの開始として扱われる。行頭に # を書くと、その行すべてがコメント

として扱われる。 行の途中に # を書くと、 # 以降がコメントとして扱われる。（コメントを書く作業は、コマンドを書く作業と同様に大切である。詳しくは第3回の授業で説明する。）

Rに用意されている関数の使い方についてHelp を参照したいときは ?関数名 （または help(関数名)） とする。ウェブブラウザでHelpを参照したいときは、 help.start() とする。

インストール済みのパッケージを利用するときは、 library("パッケージ名") とする。たとえば、**ggplot2** パッケージを使いたいなら、 library(ggplot2) とする。パッケージをインストールする際は、 install.packages("パッケージ名") とする。その際、どのレポジトリからダウンロードするか尋ねられるので、自分に一番近いところ選ぶ。（パッケージをインストールする度にレポジトリを指定するのが面倒なら、.Rprofile であらかじめレポジトリを指定しておく。）

Rを電卓代わりに使う

Rは電卓の代わりとして使うことができる。 たとえば、

```
1 + 1       # 足し算
```

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

```
100 - 20    # 引き算
```

```
## [1] 80
```

```
5 * 8       # 掛け算
```

```
## [1] 40
```

```
2 / 3       # 割り算
```

```
## [1] 0.6666667
```

```
2 ^ 3       # 累乗
```

```
## [1] 8
```

```
sqrt(2)     # 平方根
```

```
## [1] 1.414214
```

```
2 ^ (1 / 2) # sqrt() と同じ
```

```
## [1] 1.414214
```

などの計算ができる。 計算の順番を指定するときは、() で囲めばよい。

```
(5 * (2 + 1)) ^ 3  # (2 + 1) を最初に計算し、それに5を掛けてから
        最後に三乗する
```

```
## [1] 3375
```

変数の利用

Rでは、変数（正確にはオブジェクト）を（ある程度）自由に作ることができる。 変数の名前は自由に決めてよい（ただし、数字から始まるものはだめ。また、“-” [ハイフン] はマイナスと区別できないので使えない）。 たとえば、

```
a <- 1
b <- 2
```

とすると、a, b という2つの変数ができる。ここで <- は変数に値を割り当てること

を意味する。（<- の代わりに = を使うこともできるが混乱の元になるので、変数を定義するときは常に <- を使うことにする。）

<- はショートカットキーを使って入力する。

- macOS: “option” + “-”（optionキーと - [マイナス] キーを同時に押す）
- Windows: “Alt” + “-”

こうすることで、 <- だけでなく、その前後に半角スペースが**1**つずつ挿入されるので便利である。つまり、順番に

1. a
2. “option” + “-” または “Alt” + “-”
3. 1

と打つと、

```
a <- 1
```

と入力される。ショートカットキーを使わないと、

1. a
2. SPACE
3. <
4. -
5. SPACE
6. 1

と6段階の入力が必要になる。ショートカットキーを使うことで、これを3回に短縮できる。

定義された変数名のみを入力して実行すると、変数の中身が表示される。

```
a
```

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

```
b
```

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

この変数は、計算に使える。

```
a + b
```

```
## [1] 3
```

```
a - b
```

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

```
a * b
```

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

```
a / b
```

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

```
b ^ a
```

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

また、

```
d <- a
a <- 3
```

とすると、$d = 1, a = 3$ となる（各自確かめること）。

また、変数名には日本語も使える。 たとえば、

```
変数1 <- 5
変数2 <- 7
変数1 * 変数2
```

```
## [1] 35
```

とすることも可能である。しかし、変数名に日本語を使うと、（1）英語/日本語の切り替えが面倒であり、（2）文字化け等の予期せぬ問題が生じることがあるので、なるべく日本語の変数名は使わないほうが無難である。

変数の割当と画面への出力を同時に行いたいときは、全体を `()` で囲む。

```
(d <- 3 * 5)
```

```
## [1] 15
```

変数を消去したいときは `rm()` 関数を使う。

```
rm(d)
```

ベクトルと行列

ベクトル (vectors)

Rで特定のベクトル (vector) を作りたいときは、 `c()` を使う。 たとえば、1, 2, 3, 4, 5 という5つの数字からなるベクトル **a** を作るには、

```
a <- c(1, 2, 3, 4, 5)
```

とする。このベクトルを画面に表示すると、

```
a
```

```
## [1] 1 2 3 4 5
```

となる。

ベクトルの中身は文字列でもかまわない。 たとえば、

```
univ_kochi <- c('KUT', 'University of Kochi', 'Kochi Unive
        rsity')
```

とすれば、文字列 (characters) のベクトルができる。 このように、文字列は引用符（ '' でも "" でもよい）で囲む。

ひとつひとつの要素を指定する代わりに、様々な方法でベクトルを作ることが可能である。 たとえば、 seq() 関数を使うと、一連の数字からなるベクトルを作ることができる。

```
seq(1, 20, by = 1)   # 1から20までの整数。1:20 でも同じ
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20
```

```
seq(1, 20, by = 2)   # 1から19までの奇数
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19
```

```
seq(2, 20, by = 2)   # 2から20までの偶数
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
seq(20, 1, by = -5)  # 降順、間隔は5
```

```
## [1] 20 15 10  5
```

```
seq(1, 100, length.out = 10) # 最小値が1、最大値が100で、要素の
        数 (length) が10のベクトル
```

```
##  [1]   1  12  23  34  45  56  67  78  89 100
```

`seq(x, y, by = 1)` の場合はより単純に `x:y` とすればよい。

```
1:20
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20
```

また、 `rep()` 関数も便利である。

```
rep(3, 10)   # 3が10個のベクトル
```

```
##  [1] 3 3 3 3 3 3 3 3 3 3
```

```
rep(c('a', 'b', 'c'), c(3, 1, 2))  # aが3つ, bが1つ, cが2つの
        ベクトル
```

```
## [1] "a" "a" "a" "b" "c" "c"
```

ベクトルのi番目の要素にアクセスするには ベクトル名`[i]` とする。 同時に複数の要素を取り出すこともできる。 たとえば、

```
a <- seq(10, 100, length = 10)
b <- 10:1
a[2]
```

```
## [1] 20
```

```
b[2]
```

```
## [1] 9
```

```
a[3:5]
```

```
## [1] 30 40 50
```

```
a[c(1,3,5)]
```

```
## [1] 10 30 50
```

```
a[c(8, 2, 4)]
```

```
## [1] 80 20 40
```

ベクトルの演算

Rでは、ベクトルを使った演算が可能である。 たとえば、次のような計算ができる。

```
x <- 1:10
x + 10    # ベクトルxの各要素に10を加える
```

```
##  [1] 11 12 13 14 15 16 17 18 19 20
```

```
x - 5     # ベクトルxの各要素から5を引く
```

```
##  [1] -4 -3 -2 -1  0  1  2  3  4  5
```

```
x * 2     # ベクトルxの各要素に2をかける
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x / 3     # ベクトルxの各要素を3で割る
```

```
##  [1] 0.3333333 0.6666667 1.0000000 1.3333333 1.6666667
2.0000000 2.3333333
##  [8] 2.6666667 3.0000000 3.3333333
```

```
x ^ 2     # ベクトルxの各要素を2乗する
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```
sqrt(x)   # ベクトルxの各要素の平方根（square root）を計算する
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449
490 2.645751 2.828427
##  [9] 3.000000 3.162278
```

また、複数のベクトルを使って、次のような計算ができる。

```
x <- 1:10
y <- -10:-1

# xのi番目の要素とyのi番目の要素を足す（i = 1, 2, ..., 10）
x + y
```

```
##  [1] -9 -7 -5 -3 -1  1  3  5  7  9
```

```
# xのi番目の要素からyのi番目の要素を引く（i = 1, 2, ..., 10）
x - y
```

```
##  [1] 11 11 11 11 11 11 11 11 11 11
```

```
# xのi番目の要素とyのi番目の要素をかける（i = 1, 2, ..., 10）
x * y
```

```
##  [1] -10 -18 -24 -28 -30 -30 -28 -24 -18 -10
```

```
# xのi番目の要素をyのi番目の要素で割る（i = 1, 2, ..., 10）
x / y
```

```
##  [1]  -0.1000000  -0.2222222  -0.3750000  -0.5714286  -
0.8333333  -1.2000000
##  [7]  -1.7500000  -2.6666667  -4.5000000 -10.0000000
```

```
# xのi番目の要素を「yのi番目の要素」乗にする（i = 1, 2, ..., 10）
x ^ y
```

```
##  [1] 1.000000e+00 1.953125e-03 1.524158e-04 6.103516e-0
5 6.400000e-05
##  [6] 1.286008e-04 4.164931e-04 1.953125e-03 1.234568e-0
2 1.000000e-01
```

ベクトル同士の足し算（引き算）をしても、ベクトルの長さは変わらない。

```
length(x); length(y)
```

```
## [1] 10
```

```
## [1] 10
```

```
length(x + y)
```

```
## [1] 10
```

長さの異なるベクトルを使って演算を行うと、短いのほうのベクトルは要素をリサイクルして対応する。

```
x <- 1:10
y <- c(100, 200)
x + y
```

```
##  [1] 101 202 103 204 105 206 107 208 109 210
```

ただし、長いほうのベクトルの長さが短いほうのベクトルの長さの整数倍になっていないときは、警告 (warning) が出る。

```
x <- 1:10
y <- c(100, 200, 300)
x + y
```

```
## Warning in x + y: longer object length is not a multipl
e of shorter object
## length
```

```
##  [1] 101 202 303 104 205 306 107 208 309 110
```

2つのベクトルの内積 (dot product) は `%*%` で、直積 (outer product) は `%o%` または `outer()` で求められる。

```
x <- c(1, 3, 5)
y <- c(10, 20, 30)
x %*% y       # xとyの内積
```

```
##      [,1]
## [1,]  220
```

```
x %o% y       # xとyの直積
```

```
##      [,1] [,2] [,3]
## [1,]   10   20   30
## [2,]   30   60   90
## [3,]   50  100  150
```

```
outer(x, y)  # xとyの直積
```

```
##      [,1] [,2] [,3]
## [1,]   10   20   30
## [2,]   30   60   90
## [3,]   50  100  150
```

行列 (matrices)

Rで行列を作るには、 `matrix()` 関数を使う。 たとえば、

```
(A <- matrix(1:9, nrow = 3, byrow = TRUE))
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```
(B <- matrix(1:9, nrow = 3, byrow = FALSE))
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

のようにする。 ここで、行列Aと行列Bの違いに注目する。 要素全体をひとつの集合としてみると、AとBの行列は全く同じである。 これは、上のコードでは `1:9` という

部分が同じだからである。 しかし、要素の並び方が異なる。 Aを作ったコードは byrow = TRUE となっている。 これは、行 (row) 単位でセルを埋めて行くということである。 それに対し、Bでは byrow = FALSE となっている。 これは行単位でセルを埋めない（したがって、列 [col] 単位で埋める）ということを意味する。 この違いが、AとBの違いを生み出している。 行列を作るときは行数 nrow と列数 ncol を指定するが、要素の合計数が決まっているときは、どちらか一方を指定すれば、もう一方は自動的に決められる。 上の例では、要素の数が9で、行の数に3を指定したので、列の数は自動的に 9 / 3 = 3 になっている。

行列の各行と各列にはそれぞれ名前を付けることができる。

```
row.names(A) <- c('row1', 'row2', 'row3')  # 各行に名前をつける
colnames(A) <- c('col1', 'col2', 'col3')   # 各列に名前をつける
A
```

```
##      col1 col2 col3
## row1    1    2    3
## row2    4    5    6
## row3    7    8    9
```

row.names() には '.' があり、 colnames() にはそれがないことに注意。

行列のi行j列を取り出すには、 行列名[i, j] とする。 例えば

```
A[1, 3]           # 第1行、第3列の要素を取り出す
```

```
## [1] 3
```

```
A[2, c(1, 3)]     # 第2行で、第1列と第3列の要素を取り出す
```

```
## col1 col3
##    4    6
```

```
A[3, ]            # 第3行の要素をすべて取り出す
```

```
## col1 col2 col3
##    7    8    9
```

```
A[, 2]            # 第2列の要素をすべて取り出す
```

```
## row1 row2 row3
##    2    5    8
```

行列の演算

Rでは行列を使った計算ができる。

基本的な演算の結果は次のとおりである。

```
A <- matrix(1:9, ncol = 3)     # 行列Aを定義する
B <- matrix(-4:4, ncol = 3)    # 行列Bを定義する
A + 3   # 行列の各要素に3を加える
```

```
##      [,1] [,2] [,3]
## [1,]    4    7   10
## [2,]    5    8   11
## [3,]    6    9   12
```

```
2 * A   # 行列の各要素を2倍する
```

```
##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
## [3,]    6   12   18
```

```
A + B   # Aのi行j列要素とBのi行j列要素を足す (i, j = 1, 2, 3)
```

```
##      [,1] [,2] [,3]
## [1,]   -3    3    9
## [2,]   -1    5   11
## [3,]    1    7   13
```

```
A * B   # 行列の要素同士の積
```

```
##      [,1] [,2] [,3]
## [1,]   -4   -4   14
## [2,]   -6    0   24
## [3,]   -6    6   36
```

```
A %*% B  # 行列の積
```

```
##      [,1] [,2] [,3]
## [1,]  -30    6   42
## [2,]  -39    6   51
## [3,]  -48    6   60
```

```
B %*% A  # 行列の積
```

```
##      [,1] [,2] [,3]
## [1,]    0   -9  -18
## [2,]    6    6    6
## [3,]   12   21   30
```

```
## AB と BA は異なる
A %*% B == B %*% A  # 要素ごとに等しいかどうか比較する
```

```
##       [,1]  [,2]  [,3]
## [1,] FALSE FALSE FALSE
## [2,] FALSE  TRUE FALSE
## [3,] FALSE FALSE FALSE
```

```
a <- 1:3  # ベクトルを定義する
A %*% a   # （3行3列）x（3行1列）なので結果は3行1列
```

```
##      [,1]
## [1,]   30
## [2,]   36
## [3,]   42
```

行列の転置 (transpose) には `t()` を使う。

```
t(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

逆行列は `solve()` で求める。

```
C <- matrix(c(2, 3, 5,
              7, 11, 13,
              17, 19, 23),
            nrow = 3)
solve(C)
```

```
##             [,1]       [,2]        [,3]
## [1,] -0.07692308 -0.7692308  0.69230769
## [2,] -0.33333333  0.5000000 -0.16666667
## [3,]  0.20512821 -0.1153846 -0.01282051
```

特異行列 (a singular matrix) に `solve()` を使うとエラーになる。

```
S <- matrix(1:9, nrow = 3)
solve(S)
```

```
## Error in solve.default(S): Lapack routine dgesv: system
is exactly singular: U[3,3] = 0
```

エラーメッセージを読めば何がまずいのかわかるので、エラーが出たらエラーメッセージの中身をよく読むこと。 この場合は、行列が特異行列 (singular) であることを教えてくれている。

日本語版を利用している場合は、エラーメッセージも日本語で表示される（はず）。

Enjoy!

授業の内容に戻る (../)