第 2 回 基本演算，データ型の基礎，ベクトルの操作方法

2023 年 2 月 27 日

目 次

1 ここで学ぶ事

- コメント（注釈）の書き方

- 基本演算（四則演算，べき乗，対数計算，絶対値）
- データ型の基礎（logical, numeric, complex, character）
- データ構造とは
- ベクトルの操作方法
- ヘルプの使い方

2 コメント（注釈）の書き方

プログラムのソースコードには後で見たときに分かり易いようにコメント（注釈）を付けておくと良い．R はシャープ記号#が現れると，シャープ記号から行末までを無視する．したがって，プログラム中の様々なコメントは#の後ろに書く．

```
> ##################
> ## コメントの例
> ##################
> cat("ABC")   # これはコメントです
ABC
```

3 基本演算

四則演算と計算の優先順位は通常通り．

```
> 3 * 4
[1] 12
> 3 * 4 + 2
[1] 14
> 3 * (4 + 2)
[1] 18
> 3/4
[1] 0.75
> -5 / -6
[1] 0.8333333
> (-5)/(-6)
[1] 0.8333333
```

べき乗は 2 通りの表記方法がある．べき乗の優先順位は他の四則演算より高い．

```
> 2^3         # ^は他のプログラミング言語でもよく使われる記法
[1] 8
> 2**3        # *を 2 つ繋げてもべき乗
[1] 8
> 2^3*4       # べき乗の優先順位が高い
```

```
[1] 32
> 2^(3*4)
[1] 4096
> 2**3*4      # べき乗の優先順位が高い．**と*が混じった式は見づらい
[1] 32
> 2**(3*4)
[1] 4096
```

平方根.

```
> sqrt(2)
[1] 1.414214
> 2^(1/2)
[1] 1.414214
```

対数計算．単に `log(2)` を実行すると自然対数になる．すなわち，`log(2)`$= \log_e 2 = \ln 2$．対数の底の指定は第 2 引数で行う．`log(100,10)`$= \log_{10} 100$.

```
> log(2)
[1] 0.6931472
> log(100, 10)
[1] 2
> exp(1)
[1] 2.718282
> exp(0.6931472)
[1] 2
```

絶対値（absolute values）.

```
> abs(-0.2)
[1] 0.2
> abs(0.2)
[1] 0.2
```

3.1 チェックリスト

- 四則演算の方法.
- べき乗と優先順位.
- 対数計算.
- 絶対値.

4 データ型の基礎

4.1 データ型とは？

- コンピュータがデータを扱うとき，データが数値なのか，文字列なのか等々，データの種類が分からないとどうやって処理して良いかわからない．
- データはコンピュータ上では 0 と 1 の羅列としてメモリに格納されるので，0 と 1 の塊がどの様なデータを表しているのか識別できる様にしておかなければならない．
- 一般にプログラミング言語はデータに「型」を付けて区別する．

  $\Longrightarrow$_____

- R ではデータ型を_____ と呼ぶ．

4.2 覚えておくべき R のデータ型（mode）

- 論理値（logical）：真，偽を表すデータ型で，TRUE か FALSE のいずれかの値をとる．省略形として T と F も使用可能．
- 実数（numeric）：R で複素数以外の数は numeric 型として扱われる．
- 複素数（complex）：虚数 i を含む数．
- 文字列（character）：ダブルクォーテーションで囲まれた文字列．

mode() 関数は引数のデータ型（mode）を返す．以下，いろいろなデータ型に対する mode() の返り値である．「返り値」とは関数が返す値のことで，「戻り値」とも言う．

```
> mode(TRUE)
[1] "logical"
> mode(FALSE)
[1] "logical"
> mode(pi)      # pi は円周率を表す定数
[1] "numeric"
> mode(2.0)
[1] "numeric"
> mode(2)
[1] "numeric"
> mode(2+2i)
[1] "complex"
> mode("吾輩は猫である")
[1] "character"
```

is. モード名 () で型の種別をテストできる．

```
> is.character("文字列")
[1] TRUE
> is.numeric("文字列")
[1] FALSE
> is.numeric(3.14)
[1] TRUE
> is.numeric(pi)
[1] TRUE
> is.logical(FALSE)
[1] TRUE
> is.logical(1)
[1] FALSE
```

4.3 変数にデータを代入してみよう

変数にデータを代入するには'<-' を使う.

```
> x <- 3              # x という変数を作成し 3 を代入
> x                   # 変数の中身を確認
[1] 3
> is.numeric(x)       # x は実数かどうかテスト
[1] TRUE
> y <- "我輩は猫である"
> y
[1] "我輩は猫である"
> is.character(y)
[1] TRUE
> is.numeric(x)
[1] TRUE
> mode(y)
[1] "character"
```

4.4 NULL

- データはコンピュータ上では 0 と 1 の羅列で表現されるが，未使用のデータ領域を全て 0 で初期化した場合，その状態をプログラミング用語で＿＿＿＿＿＿＿＿＿＿＿＿ と呼ぶ.
- R でもデータの領域を初期化するとき等に NULL を使用する.

```
> x <- NULL         # NULL で初期化
> x                 # x の中身は NULL
NULL
> is.numeric(x)
```

```
[1] FALSE
> is.null(x)      # 値が NULL かどうかテスト
[1] TRUE
> mode(x)         # NULL の mode は"NULL"
[1] "NULL"
> x <- FALSE      # x に真偽値を代入
> is.null(x)      # NULL ではない
[1] FALSE
> is.logical(x)
[1] TRUE
> mode(x)
[1] "logical"
```

4.5 チェックリスト

- データ型
- mode
- 返り値，戻り値
- logical 型，numeric 型，complex 型，character 型
- `mode()` 関数
- `is.` モード名 `()` 関数
- 変数の作成・代入方法
- NULL

5 データ構造とは？

コンピュータに処理をさせる場合，1 つのデータを単独で使うことはあまりない．通常，幾つものデータをまとめて扱う．例えば表計算ソフトではデータを「テーブル」という構造で管理する．

このように，ある構造を持ったデータの集合体を＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿ と呼ぶ．

自分でプログラムを書いてオリジナルのデータ構造を定義することも出来るが，R には予め便利なデータ構造が用意されている．

- ベクトル（vector）：同じ型（mode）のデータを保持する 1 次元配列．配列とは複数のデータを格納するための連続したデータ領域で，データを入れる箱が 1 列に並んでいるイメージを想像するとよい．
- 行列（matrix）：同じ型（mode）のデータを保持する 2 次元配列．
- 配列（array）：同じ型（mode）のデータを保持する n 次元配列．

- リスト（list）：任意の型のデータおよびデータ構造を要素に持つリスト構造．リスト構造とは，データの塊が配列のように連続しておらず，各データが次のデータにつながっている（連鎖している）データ構造，例えば数珠玉を各データ要素にたとえると，数珠がリスト構造のイメージになる．
- データフレーム（data.frame）：表計算ソフトのテーブルのような 2 次元のデータ構造で，列には同じ型（mode）のデータが入る．列内のデータが同じ型であれば，違う列の型は異なっていてよい．（正確には「長さが同じベクトル」のリストがデータフレーム．）

最もよく使うのがベクトルとデータフレーム．データフレームについては第 3 回以降で詳しく学ぶ．

5.1 チェックリスト

- データ構造
- 配列
- vector, matrix, array, list の違い．
- データフレーム

6 ベクトルの操作方法

6.1 ベクトルの生成

`c()` 関数の引数にデータを列挙するとベクトルを返す．`c()` は combine の略で，引数を繋げて 1 つのベクトルを生成して返す．

```
> c(1, 2, 3, 4)           # 数値型のベクトルを生成
[1] 1 2 3 4
> c(1, 2, c(3, 4))        # 入れ子にしたベクトルはフラットなベクトルになる
[1] 1 2 3 4
> v <- c("a", "b", "c")   # 文字列のベクトルを生成
> v                       # v の中身を確認
[1] "a" "b" "c"
> c(v, v, c(1, 2, 3))     # 異なる型を渡すと同じ型に変換される．
[1] "a" "b" "c" "a" "b" "c" "1" "2" "3"
```

上の最後の例では，数値がより一般的な文字列型に変換される．ベクトルの要素は全て同じ型でなければならないので，異なる型が与えられた場合，同じ型に変換される点に注意．

6.2 便利なベクトル生成関数

replicate 関数 `rep(x, times)` は `x` を `times` 回複製したベクトルを返す．

```
> rep(1, 5)
[1] 1 1 1 1 1
> rep("文字列", 3)
[1] "文字列" "文字列" "文字列"
> rep(TRUE, 3)
[1] TRUE TRUE TRUE
> rep(c(1, 2), 3)  # 結果はフラットなベクトルに成る
[1] 1 2 1 2 1 2
```

rep() 関数はベクトルを返すので，上の最後のように結果はフラットなベクトルに成る．

sequence 関数 seq(from, to, by) は，from から to まで変化量が by となる数値型のベクトルを返す．by を省略すると変化量の絶対値は 1 になる．

```
> seq(1, 10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(1, 10, 2)  # 2 ずつインクリメント
[1] 1 3 5 7 9
> seq(10, 1)
 [1] 10  9  8  7  6  5  4  3  2  1
> seq(10, 1, -2)
[1] 10  8  6  4  2
```

6.3 要素のインデックス

実行結果の行頭に印字される [1] は，行頭にある要素のインデックス番号を表す．

```
> seq(1, 50)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
[28] 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

```
> 1
[1] 1
```

特にベクトルを生成しなくても，1 つのデータも要素が 1 つのベクトルになる！

ベクトルの要素は [] 括弧内にインデックス番号を指定することでアクセスできる．

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x[1]
[1] 1
> x[2]
[1] 2
```

```
> x[10]
[1] 10
```

複数のインデックスをベクトルにして [] 括弧内に指定すると，複数の要素を抽出できる．

```
> days <- c("日", "月", "火", "水", "木", "金", "土")
> days
[1] "日" "月" "火" "水" "木" "金" "土"
> days[3]
[1] "火"
> days[c(1, 2, 3)]
[1] "日" "月" "火"
> days[1:3]
[1] "日" "月" "火"
> days[seq(2, 7, 2)]  # 2 日目から 1 週分，1 日おきに抽出
[1] "月" "水" "金"
```

インデックスに「マイナス」を付けると，そのインデックス番号の要素だけ取り除かれたベクトルを返す．

```
> days[-3]              # 3 つ目の要素だけ取り除いたベクトルを返す．
[1] "日" "月" "水" "木" "金" "土"
> days[-1:-3]           # 1 から 3 つ目の要素を取り除いたベクトルを返す．
[1] "水" "木" "金" "土"
> days[-seq(2, 7, 2)]   # seq(2,7,2) は c(2,4,6) と同じ．それにマイナスを付加
[1] "日" "火" "木" "土"
```

length() 関数はベクトルの要素数を返す．

```
> length(days)
[1] 7
> length(x)
[1] 10
```

6.4 四則演算ふたたび

前に数値に対して四則演算を行ったが，1 つの数値も 1 つの要素から成るベクトルである．R での四則演算は任意の長さのベクトルに対しても行うことができる．ベクトルに対する計算は各要素ごとに行われる．以下では 1 から 50 までの数字が入ったベクトルを作成するのに，コロン「:」を使っている．1:50 は，seq(1,50) の省略表記で，差が 1 のベクトルを作成するときによく使われる．

```
> x <- 1:50    # seq(1, 50) と同じ
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
```

```
[28] 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
> y <- 50:1    # seq(50, 1) と同じ
> y
 [1] 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
[28] 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> x * 2        # 各要素を 2 倍
 [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
[21] 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80
[41] 82 84 86 88 90 92 94 96 98 100
> x / 2        # 各要素を 2 で除す
 [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
[17] 8.5 9.0 9.5 10.0 10.5 11.0 11.5 12.0 12.5 13.0 13.5 14.0 14.5 15.0 15.5 16.0
[33] 16.5 17.0 17.5 18.0 18.5 19.0 19.5 20.0 20.5 21.0 21.5 22.0 22.5 23.0 23.5 24.0
[49] 24.5 25.0
> x + 100      # 各要素に 100 を加える
 [1] 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
[21] 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
[41] 141 142 143 144 145 146 147 148 149 150
> x + y        # x と y は同じ要素数なので各要素同士を加える.
 [1] 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51
[28] 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51
> log(1:5)     # log(c(1, 2, 3, 4, 5)) と同じで各要素に log が適用される
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
> (1:5)^2      # c(1, 2, 3, 4, 5)^2 と同じ
[1]  1  4  9 16 25
```

最後の (1:5)^2 で 1:5 に括弧が付いている点に注意しよう．括弧は計算の優先順位を変える．まず括弧内から先にコンピュータは処理し（プログラミング用語では先に「評価する」という），そのあと 2 乗される．括弧を付けないで 1:5^2 と書いてしまうと，1 から $5^2 = 25$ までのベクトルが生成されるので注意.

```
> 1:5^2        # 1:25 と同じ
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

6.5 チェックリスト

- 「:」の使い方
- c(), rep(), seq() の使い方.
- ベクトル要素のアクセス・削除方法.
- length() 関数

7　ヘルプの使い方

今回，新しい関数を幾つか学んだ．こうした関数の使い方を調べるにはマニュアル（ヘルプ）を使う．例えば，rep() 関数のヘルプを読むには以下を実行する．

```
> help(rep)
```

関数の名前が正確に分からない場合，キーワードで検索することもできる．例えば標準偏差（standard deviation）を求めたいが，適切な関数がわからない時は，以下のように help.search() にキーワードを文字列で指定して関連事項を検索することができる．

```
> help.serach("standard deviation")
```

上を実行すると幾つか標準偏差を求める関数がリストアップされる．その中の stats::sd は，stats パッケージ内の sd() 関数という意味である．ここでさらに help(sd) とすれば関数の詳細説明を読むことができる．

パッケージとは，幾つかのプログラムを一つにまとめて，R に追加インストールしやすくしたもの．これまでプログラムをファイルに記述し，source してファイルから実行する方法を学んだが，プログラムのファイルをパッケージとして一つにまとめ R にインストールすると，今後は source しなくても利用可能になる．help.search() の検索で見つかったものは既にインストール済みなので，特にインストール作業の必要はない．

7.1　チェックリスト

- help()
- help.search()
- パッケージ
- sd() 関数

8　実習

(1) 2, 4, 6, 8, 10 の偶数列が 10 個並ぶ，以下のベクトルを生成せよ．

```
 [1]  2  4  6  8 10  2  4  6  8 10  2  4  6  8 10  2  4  6  8 10  2  4  6  8 10
[26]  2  4  6  8 10  2  4  6  8 10  2  4  6  8 10  2  4  6  8 10  2  4  6  8 10
```

(2) 2, 4, 6, 8, 10 の偶数列が 100 個並ぶベクトルを生成せよ．

(3) sd() 関数をヘルプで検索し使い方を学び，上で生成した 2 つのベクトルの標準偏差をそれぞれ求めよ．