クレジット:
UTokyo Online Education 統計データ解析 I 2017 小池祐太

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

統計データ解析 (I) 第 2 回

小池祐太

2017 年 10 月 4 日

パッケージのインストール

- R では, その機能を拡張するために多数のパッケージが用意されている
- 従って, 初期設定で関数が実装されていなくても, その関数を実装しているようなパッケージがすでに開発されていることが多い
- パッケージのインストールには以下の 2 通りの方法がある:
  - RStudio の機能を利用する方法
  - コンソールから行う方法

パッケージのインストール

- パッケージ **ggplot2** のインストール手順 (RStudio の機能を利用)
  1. 右下ペインの “Package” タブをクリック
  2. 左上あたりに “Install” という表示があるのでそれをクリック
  3. 新規ウィンドウが現れるので, “Package” のフォームにインストールしたいパッケージ名 (ここでは ggplot2) を入力し, 下部の “Install” をクリック
  4. パッケージがインストールされる

パッケージのインストール

- インストール済みのパッケージは右下ペインの “Package” タブに表示される
  - パッケージ名の左側のボックスをチェックすると, そのパッケージがロードされて, パッケージに含まれる関数などが利用可能になる
- 左上部の “Update” をクリックすると, インストール済みパッケージを最新版に更新できる

パッケージのインストール

- パッケージ **ggplot2** のインストール手順 (コンソールから)
  1. R のコンソール上で `install.packages("ggplot2")` を実行
  2. パッケージをダウンロードするためのサイト (CRAN のミラーサイト) を選ぶことを要求された場合は, 適当なものを選ぶ (Japan のミラーサイトがよい)
  3. パッケージがインストールされる
- インストールしたパッケージはコンソール上で `library(パッケージ名)` か `require(パッケージ名)` を実行することでロードされる (今の場合 `library(ggplot2)` または `require(ggplot2)` を実行)

データ構造

- R には, 以下のようなデータ構造が用意されている (代表的なもの):
  - ベクトル (vector)
  - 行列 (matrix)
  - リスト (list)
  - データフレーム (data frame)

データ構造: ベクトル

- ベクトルとは, スカラー値の集合
- R オブジェクトは基本的にはベクトルとして扱われる (スカラー値は長さ 1 のベクトルとして扱われる)
- スカラー値として扱われるものには, 実数や複素数以外に, 文字列 (“” で囲まれた文字. “x” など) や論理値 (TRUE, FALSE) などが含まれる
- 実行例 `scalar.r`

データ構造: ベクトル

- 要素 a, b, ... からなるベクトルは以下で生成できる:

  `c(a,b,...)`

  - a, b, ... は数値や文字列が混同していてもよい
- ベクトル x の i 番目の要素は `x[i]` で取り出せる (注: ベクトルの添え字は 1 から始まる)
- `x[c(1,3,4)]` のように添え字もベクトルで指定すると, 指定された添え字に対応する要素からなるベクトルが取り出せる (この場合ベクトル `c(x[1],x[3],x[4])`)
- ベクトル x の長さは `length(x)` で取り出せる

データ構造: ベクトル

- 実数 x から y (x<y) まで 1 ずつ増加していく要素を持つベクトルは x:y で生成できる
  - x>y の場合は x から y まで 1 ずつ減少していく要素を持つベクトルとなる
- より一般に, 実数 x から y (x<y) まで a ずつ増加していく要素を持つベクトルは seq(x,y,by=a) で生成できる
- ベクトル x を n 回繰り返した要素をもつベクトルは rep(x,n) で生成できる
  - 従って, rep(x,n) の長さは length(x) × n となる
- ベクトル x の末尾にベクトル y をつなげたベクトルは c(x,y) で, x の要素を反転したベクトルは rev(x) で生成できる
- 実行例 vector.r

データ構造: ベクトル

演習 1

以下に示すベクトルを R において作成せよ.

1. $(17, 16, \ldots, 4, 3)$ (17 から 3 まで 1 ずつ減少していく要素をもつベクトル)
2. 1 以上 30 以下の奇数を昇順に並べたベクトル
3. すべての要素が π からなる長さ 10 のベクトル

データ構造: 行列

- すべての要素が a である $m \times n$ 行列は

  `matrix(a,m,n)`

  で生成できる
- より一般に, 長さ mn のベクトル
  $x = (x_{11}, \ldots, x_{m1}, x_{21}, \ldots, x_{2n}, \ldots, x_{m1}, \ldots, x_{mn})$ に対して,

  `matrix(x,m,n)`

  で $m \times n$ 行列 $(x_{ij})_{1 \leq i \leq m, 1 \leq n}$ が生成できる
  - `as.vector` を適用することでベクトル化できる:

    ```
    x = as.vector(matrix(x,m,n))
    ```

データ構造: 行列

- 行列 `X` のサイズは `dim(X)` で取り出せる (長さ 2 のベクトル)
- 逆に, ベクトルに dim 属性を指定してやることで行列に変換できる
  - 例えば, 前頁のベクトル `x` に対して

    ```
    dim(x) <- c(m,n)
    ```

    を実行すれば, `x` は行列 `matrix(x,m,n)` に変換される
- 行列 `X` の行数は `nrow(X)` で, 列数は `ncol(X)` で取り出せる
- 行列 `X` の (i, j) 成分は `X[i,j]` で取り出せる
  - ベクトルの場合と同様に添え字をベクトルで指定することで, 部分行列の取り出しも可能
  - `X[i,]`, `X[,j]` でそれぞれ `X` の第 `i` 行, 第 `j` 列が取り出せる

データ構造: 行列

- 長さが等しい複数のベクトル `x`, `y`, ... を列もしくは行ベクトルとする行列を作成することもできる
  - 列ベクトルの場合 ··· `cbind(x, y, ...)`
  - 行ベクトルの場合 ··· `rbind(x, y, ...)`
- `cbind`/`rbind` は行数/列数が等しい複数の行列を横/縦に結合するのにも使える
- 補足 行列の高次元版のデータ構造として, 配列 (array) が用意されている
- 実行例 `matrix.r`

データ構造: 行列

演習 2

以下に示す行列を R において作成せよ

1. $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$
2. $\begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 4 \end{pmatrix}$

データ構造: リスト

- リスト ··· 異なる構造のデータをまとめて 1 つのオブジェクトとして扱えるようにしたもの. リストの各要素はデータ型・クラスともにバラバラであってもよい
- R のオブジェクト x, y, ... を要素とするリストは `list(x,y,...)` で生成される. リスト L の i 番目のオブジェクトには `L[[i]]` でアクセスできる.

データ構造: リスト

- リストは各成分に名前をつけることができる
  - 方法 1: 作成時に `list(name1 = x, name2 = y, ...)` のように書くと, 各成分に `name.1`, `name.2`, ... といった名前がつく
  - 方法 2: すでに作成してある長さ `n` のリスト `L` の各成分に名前をつける (もしくは名前を変更する) には,

    ```
    names(L) <- c(name.1, name.2, ..., name.n)
    ```

    のようにする
- リスト `L` の名前 `name` の成分は, `L$name` もしくは `L[[name]]` で取り出せる
- 実行例 `list.r`

データ構造: データフレーム

- データフレーム $\cdots$ 行列風リスト. リストにおいて, 各成分が長さが等しいベクトルであるようなもの
- イメージ
  - 複数の個体について, いくつかの属性を集計したデータ (例えばある小学校の 1 年生の身長, 体重, 性別, 血液型, $\ldots$ を集計したデータ)
  - リストの各成分はある属性に関する観測データに対応
  - 個体数は集計項目に関わらず変化しないが, 集計項目によっては定量的データ・定性的データの違いが出てくるので, データ型は変わりうる

データ構造: データフレーム

- 長さが等しいベクトル x, y, ... を変数とするデータフレームは, data.frame(x,y,...) で生成される
  - データフレームはリストなのでリストと同様にして各変数にアクセスできる
  - 他方, データフレームは行数がベクトルの長さ (個体数), 列数が変数の個数 (観測項目の数) の行列と同様にアクセスできる
- 実データは上のような表形式のデータであることが多いので, 実データを R に読み込む際に役に立つデータ構造
- 実行例 data.frame.r

データ構造: データフレーム

演習 3

次の表に対応するデータフレームを作成せよ

	国語	数学
A	90	25
B	80	50
C	70	75
D	60	100

ベクトルの計算

- 以下ではベクトルを太字で (対応する R オブジェクトはタイプライタ体), その要素は下付き添字で表現する
- 例えば k 次元ベクトルは

$$\boldsymbol{a} = (a_1, a_2, \ldots, a_k) \tag{1}$$

のように表す

ベクトルの計算: 和

- 同じ長さのベクトルの和および差

$$\boldsymbol{a} \pm \boldsymbol{b} = (a_1 \pm b_1, a_2 \pm b_2, \ldots, a_k \pm b_k) \tag{2}$$

  は, 数値の和と差のように扱うことができる
- すなわち, a, b が同じ長さのベクトルであれば,

```
a+b
```

  によって (2) を計算できる
- 実行例 `vector-sum2.r`

ベクトルの計算: 積

- 数値の場合と同様に, 同じ長さの 2 つのベクトル a と b の積 a*b が定義されているが, これは成分ごとの積 (Hadamard 積)

$$\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, a_2 b_2, \ldots, a_k b_k) \tag{3}$$

  を計算する
- 一方, ベクトルに対する積は, 内積

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{k} a_i b_i \tag{4}$$

  の方がポピュラーであり, こちらは a%*%b で計算できる
- 実行例 `vector-prod2.r`

ベクトルの計算: 初等関数の適用

- ベクトルに初等関数 (sin, exp, . . . など) を適用すると, 成分ごとに計算した結果が返される
- 例えば, ベクトル **a** に関数 sin を適用した結果 `sin(a)` は

$$(\sin(a_1), \ldots, \sin(a_k))$$

  となる
- 実行例 `vector-fun.r`

行列とその演算

- 以下では行列を大文字で (対応する R オブジェクトはタイプライタ体), その要素は下付き添字で表現する. 例えば $m \times n$ 行列は

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \tag{5}$$

  のように表す
- また, 行列 A の ij 成分を指す場合には $(A)_{ij}$ のように書くこともある

行列とその演算: 和

- 同じ大きさの行列の和および差

$$(A \pm B)_{ij} = a_{ij} \pm b_{ij} \tag{6}$$

  は, ベクトルと同じように記述することができる
- すなわち, `A`, `B` が同じサイズの行列であれば,

```
A+B
```

  によって (6) を計算できる
- 実行例 `matrix-sum2.r`

行列とその演算: 積

- ベクトルの場合と同様に, サイズが同じ 2 つの行列 `A` と `B` に対して, `A*B` は成分ごとの積 (Hadamard 積)

$$(A \circ B)_{ij} = a_{ij} b_{ij} \tag{7}$$

  を計算する
- 一方で, A が (n, m) 行列, B が (m, l) 行列のとき, 通常の意味での行列に対する積 AB

$$(AB)_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \tag{8}$$

  が定義されるが (AB は (n, l) 行列), こちらは `A%*%B` で計算できる
- 実行例 `matrix-prod2.r`

行列とその演算: 初等関数の適用

- ベクトルの場合と同様, 行列に初等関数 (sin, exp, . . . など) を適用すると, 成分ごとに計算した結果が返される
- 例えば, 行列 A に関数 sin を適用した結果 `sin(A)` は, (i, j) 成分が

$$\sin(a_{ij})$$

で与えられる, 行列 A と同じサイズの行列となる
- 実行例 `matrix-fun.r`

行列とその演算: 行列式とトレース

- 正方行列 A の行列式 $\det(A)$ は, 関数 `det()` で計算できる
- n 次正方行列 A のトレース (対角成分の総和)

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{ii}$$

を計算する関数はデフォルトでは用意されていないが, 行列の対角成分を取り出す関数 `diag()` と, ベクトルの総和を計算する関数 `sum()` を組み合わせて

```
sum(diag(A))
```

のように計算できる

- 実行例 `matrix-det2.r`

行列とその演算

演習 4

適当な 2 次正方行列 A に対して, Hamilton-Cayley の定理

$$A^2 - \mathrm{tr}(A)A + \det(A)E_2 = O$$

の成立を確認せよ. ここに, E_2 は 2 次単位行列, O は 2 次正方零行列である

行列とその演算: 逆行列

- 正方行列 A が正則ならば, その逆行列 A^{-1} は関数 `solve()` で計算できる
- 実行例 `matrix-inv2.r`

ベクトルと行列の計算: 積

- R 言語においては，列ベクトル・行ベクトルという区別はなく，単一のベクトルという概念で扱われている
- このため, 行列とベクトルの積においては, 行列のどちらからベクトルを掛けるかによって自動的に列ベクトルか行ベクトルか適切な方で扱われる
- ただし, ベクトルも行列の一種であるから, 計算結果は行列として表現されることに注意する
- 実行例 `linear-calc2.r`

ベクトルと行列の計算

演習 5

適当な 3 次正方行列 A と 3 次元ベクトル $\mathbf{b}$ を作成し,

```
A%*%b+b%*%A
```

を計算せよ (エラーになる). なぜそうなるか理由を考えよ

関数定義

- 多くの計算機言語と同様, R でもユーザー独自の自作関数を定義できる
- 自作関数の定義には関数 `function` を利用する
- 例 半径 r から球の体積と表面積を求める関数 `myfunc`

```
myfunc <- function(r){
    V <- (4/3) * pi * r^3 # 球の体積
    S <- 4 * pi * r^2 # 球の表面積
    out <- c(V,S)
    names(out) <- c("volume", "area")
    # 返り値に名前をつける
    return(out)
}

myfunc(1) # 実行
```

関数定義

演習 6

三角形の 3 辺の長さ x, y, z から面積 S を計算する関数を作成し, そのコードを area.R という名前のファイルで保存して, `source()` で読み込んで $x = 3$, $y = 4$, $z = 5$ として計算してみよ. なお, “ヘロンの公式” より

$$S = \sqrt{s(s-x)(s-y)(s-z)}, \qquad s = \frac{x+y+z}{2}$$

が成り立つ

制御文

- 一般に最適化や数値計算などを行うためには, 条件分岐や繰り返しを行うための仕組みが必要となる
- R 言語を含む多くの計算機言語では `if` (条件分岐), `for`・`while` (繰り返し) を用いた構文が用意されているが, これを制御文と言う
- 実行例 `control2.r`

制御文

- `if` 文: 書式

  `if(`条件 A`){` プログラム X`}`

  - 条件 A が満たされる場合, プログラム X を実行する
  - 「条件 A が満たされない場合, 代わりにプログラム Y を実行する」としたければ, 以下のように書く:

    `if(`条件 A`){` プログラム X`}else{` プログラム Y`}`

- `for` 文: 書式

  `for(i in M){` プログラム X`}`

  - `M` をベクトルとし, 変数 `i` が `M` の要素となる値すべてを動きながらプログラム X を繰り返し実行する
  - プログラム X は通常変数 `i` によって実行内容が変わる

制御文

- `while` 文: 書式

$$\texttt{while}(\text{条件 A})\{\ \text{プログラム X}\}$$

  - 条件 A が満たされる限り, プログラム X を繰り返し実行する
  - プログラム X は通常実行するたびに実行内容が変わり, いつか条件 A が満たされなくなるように書く

制御文

演習 7

`for` 文を用いて以下の計算をする関数を作成し, 正しく動作するか適当な値を入れて確認せよ

1. 正の整数 n から $n!$ を計算する
2. 行列 X が与えられたとき, 各列の平均を計算する (動作確認は, 例えば行列

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

を利用せよ)

制御文

演習 8

`if` 文を用いて, `for` 文の問題で作成した関数が以下の処理をできるように修正せよ

1. $n = 0$ の場合にも関数が正しく動作するようにする $(0! = 1)$
2. X がベクトルの場合でも正しく動作するようにする

制御文

演習 9

`while` 文を用いて以下の計算をする関数を作成し, 正しく動作するか適当な数値を入れて確認せよ

1. 正の整数 n から $n!$ を計算する
2. 正の実数 x から, $x \leq n$ を満たす最小の整数 n を計算する