ベクトル・行列の演算と関数

村田 昇

2020.04.24

ベクトルの計算

ベクトルの表記

- ベクトルは太字で，要素は下付き添字で表す

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

```
a <- c(a1,a2,...,ak) # k次元ベクトルの作成
```

- 別の書き方:

$$(\boldsymbol{a})_i = (\text{ベクトル } \boldsymbol{a} \text{ の第 } i \text{ 成分})$$

ベクトルの加法

- **同じ長さのベクトル** の和および差:
  数値の和と差のように扱うことができる

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

$$(\boldsymbol{a} \pm \boldsymbol{b})_i = a_i \pm b_i$$

```
a + b # 同じ長さのベクトル a,b の和
a - b # ベクトルの差
```

ベクトルの乗法

- **同じ長さの 2 つのベクトル** の乗法:
  (2 **種類ある***)*
  - 成分ごとの積 (Hadamard 積; 要素積)
  - ベクトルの内積
- データ解析ではどちらも良く用いられる

Hadamard 積

- **同じ長さのベクトル** の成分ごとの積

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

$$(\boldsymbol{a} \circ \boldsymbol{b})_i = a_i b_i$$

```
a * b # ベクトルの成分ごとの積
a / b # 成分ごとの商も計算可
```

内積

- **同じ長さのベクトル** の内積

$$\begin{aligned}\boldsymbol{a}\cdot\boldsymbol{b} &= a_1b_1 + a_2b_2 + \cdots + a_kb_k \\ &= \sum_{i=1}^{k} a_ib_i\end{aligned}$$

```
a %*% b # ベクトルの内積
```

行列の計算

行列の表記

- 行列は大文字で，要素は下付き添字で表す

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

```
A <- matrix(c(a11,a21,...,amn),m,n) # m x n 行列の作成
```

- 別の書き方:

$$(A)_{ij} = (\text{行列 } A \text{ の } ij \text{ 成分})$$

行列の加法

- **同じ大きさの行列** の和および差:
  ベクトルと同じように記述することができる

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

```
A + B # 同じサイズの行列の和
A - B # 行列の差
```

行列の乗法

- 2 つの行列の乗法:
  (2 **種類ある***)*
  - **同じ大きさの行列** の成分ごとの積 (Hadamard 積; 要素積)
  - $n \times m$ **型行列と** $m \times l$ **型行列** の積
- データ解析ではどちらも良く用いられる

Hadamard 積

- **同じ大きさの行列** の成分ごとの積

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

```
A * B # 行列の成分ごとの積
A / B # 成分ごとの商も計算可
```

行列の積

- $n \times m$ **型行列** A **と** $m \times l$ **型行列** B の積

$$(AB)_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} \quad (AB \text{ は } n \times l \text{ 行列})$$

```
A %*% B # 行列の積
```

行列式とトレース

- n **次正方行列** A の行列式 $\det(A)$

```
det(A) # 行列式
```

- n **次正方行列** A のトレース (対角成分の総和)

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

関数は用意されていないので以下を利用:

- 関数 `diag()`: 行列の対角成分を取り出す (ベクトル)
- 関数 `sum()`: ベクトルの総和を計算する

```
sum(diag(A)) # 行列のトレース
```

演習

例題

- 適当な 2 次正方行列 A で Hamilton-Cayley の定理

$$A^2 - \mathrm{trace}(A)A + \det(A)E_2 = O_2$$

の成立を確認せよ.
ただし E_2 は 2 次単位行列，O_2 は 2 次正方零行列

解答例

```r
## 行列を作成 (好きに設定してよい)
(A <- matrix(1:4,2,2)-diag(rep(3,2)))
```

```
     [,1] [,2]
[1,]   -2    3
[2,]    2    1
```

```r
## 左辺を計算
A%*%A - sum(diag(A)) * A + det(A) * diag(rep(1,2))
```

```
              [,1]          [,2]
[1,] 1.776357e-15 0.000000e+00
[2,] 0.000000e+00 1.776357e-15
```

練習問題

- 1 から 10 の 2 乗値からなるベクトルを作成せよ.
- 1 から 10 までの和を計算せよ.
- 行列を用いて九九の表を作成せよ.
- 30 度の回転行列を 2 回乗ずると 60 度の回転行列となることを確認せよ.

$$(\text{回転行列}) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

ベクトルと行列の計算

ベクトルと行列の乗法

- 列 (縦) ベクトル・行 (横) ベクトルという **区別はない**
- 行列とベクトルの順序で適切に判断される
- 計算結果は **行列** で表現される

```r
A <- matrix(1:4,2,2); b <- c(5,6) # 行列とベクトルを作成
```

```r
A %*% b # 行列 x ベクトル = 列ベクトル
```

```
     [,1]
[1,]   23
[2,]   34
```

```r
b %*% A # ベクトル x 行列 = 行ベクトル
```

```
     [,1] [,2]
[1,]   17   39
```

連立 1 次方程式の解法

- 連立 1 次方程式
  - A：n 次正則行列
  - $\boldsymbol{b}, \boldsymbol{x}$：$n$ 次元列ベクトル

$$A\boldsymbol{x} = \boldsymbol{b} \qquad \text{(連立 1 次方程式)}$$
$$\boldsymbol{x} = A^{-1}\boldsymbol{b} \qquad \text{(解;}A\text{ が正則な場合)}$$

- 解を求めるには関数 `solve()` を利用する

```
x <- solve(A, b)
```

- ベクトル $\boldsymbol{b}$ の代わりに行列も扱える

逆行列

- 正則な n 次正方行列 A の逆行列 A^{-1}

$$AA^{-1} = A^{-1}A = E_n \quad (E_n \text{ は } n \text{ 次単位行列})$$

- 関数 `solve()` を利用して求めることができる

$$AX = E_n$$
$$X = A^{-1}E_n = A^{-1}$$

```
solve(A,B) # AX=B の解 X を求める
solve(A) # 逆行列 (B が単位行列の場合省略できる)
```

- *(**他にもいくつか方法は用意されている**)*

関数の適用

- ベクトルや行列に関数 ($\sin, \exp, \ldots$ など) を適用すると成分ごとに計算した結果が返される
- ベクトル $\boldsymbol{a}$ ， 行列 A に関数 sin を適用

$$(\sin(\boldsymbol{a}))_i = \sin(a_i)$$

```
sin(a) # 成分ごとに計算される. sin(a)[i]=sin(a[i])
```

$$(\sin(A))_{ij} = \sin(a_{ij})$$

```
sin(A) # 成分ごとに計算される. sin(A)[i,j]=sin(A[i,j])
```

演習

例題

- 適当な 3 次正方行列 A と 3 次元ベクトル $\boldsymbol{b}$ を作成して $\boldsymbol{x}$ に関する連立 1 次方程式

$$A\boldsymbol{x} = \boldsymbol{b}$$

を解け

解答例

```
## 行列とベクトルを作成 (好きに設定してよい)
## rnorm(9) は正規乱数を 9 つ作成する (第 5 回で詳しく説明)
(A <- matrix(rnorm(9),3,3)+diag(rep(1,3)))
(b <- 1:3)
```

```
          [,1]       [,2]       [,3]
[1,] -0.523383 -1.006812 -0.1711131
[2,] -0.635708  1.130970 -0.3170005
[3,] -1.114306  2.313603  2.2462801
[1] 1 2 3
```

```
## 解を計算
(x <- solve(A,b))
A%*%x # 結果の確認 (b になるはず)
```

```
[1] -2.4614721  0.3235156 -0.2187258
     [,1]
[1,]    1
[2,]    2
[3,]    3
```

練習問題

- 1 から 10 の 2 乗値からなるベクトルを作成せよ.
- 2 次元ベクトルを回転行列で変換しても長さが変わらないことを確かめよ.
- 例題の A と $\boldsymbol{b}$ を用いて

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

を計算するとエラーになるが，何故そうなるか理由を考えよ.

関数の定義

自作関数

- 他の言語と同様に R でも関数を定義できる
- 関数の定義には関数 `function()` を利用する
- 半径 r から球の体積と表面積を求める関数

```
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) # 返り値のベクトルを作る
    names(out) <- c("volume", "area") # 返り値の要素に名前を付ける
    return(out) # 値を返す
```

```
}
myfunc(1) # 実行
```

```
 volume      area
4.18879 12.56637
```

制御構造

制御文

- 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` (繰り返し・回数指定)
  - `while` (繰り返し・条件指定)
- これを **制御文** と言う

`if` 文

- 条件 A が **真** のときプログラム X を実行する

```
if(条件 A) プログラム X
```

- 上記の `if` 文に条件 A が **偽** のときプログラム Y を実行することを追加する

```
if(条件 A) プログラム X else プログラム Y
```

`if` 文の例

- 20200724 が 19 で割り切れるか?

```
if(20200724 %% 19 == 0) {# %% は余りを計算
    print("割り切れます")
    print(20200724 %/% 19) # 商を表示
} else { # {}で囲まれたブロックが 1 つのプログラム
    print("割り切れません")
    print(20200724 %% 19) # 余りを表示
}
```

```
[1] "割り切れます"
[1] 1063196
```

`for` 文

- ベクトル V の要素を **順に** 変数 `i` に代入してプログラム X を繰り返し実行する

```
for(i in V) プログラム X
```

- プログラム X は変数 `i` によって実行内容が変わる

`for` 文の例

- アルファベットの 20,15,11,25,15 番目を表示

```
print(LETTERS) # LETTERS ベクトルの内容を表示
for(i in c(20,15,11,25,15)) {
    print(LETTERS[i]) # 順番に表示
```

```
}
```

```
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
[1] "T"
[1] "O"
[1] "K"
[1] "Y"
[1] "O"
```

while 文

- 条件 A が **真** である限りプログラム X を繰り返す

```
while(条件 A) プログラム X
```

- プログラム X は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
- *(repeat* **文というものもある***)*

while 文の例

- 20200809 を素因数分解する

```
n <- 20200809 # 分解の対象
p <- 2 # 最初に調べる数
while(n != 1){ # 商が 1 になるまで計算する
    for(i in p:n){ # p から n まで順に調べる
        if(n%%i == 0) { # 余りが 0 か確認
            print(i) # 割り切った数を表示
            n <- n/i # 商を計算して分解の対象を更新
            p <- i # 最初に調べる数を更新
            break # for 文を途中で終了
        }
    }
}
```

```
[1] 3
[1] 31
[1] 281
[1] 773
```

演習

例題

- 三角形の 3 辺の長さ x, y, z を与えると面積 S を計算する関数を作成せよ．
- 参考: **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．

解答例

```r
area <- function(x,y,z){
    s <- (x+y+z)/2
    S <- (s*(s-x)*(s-y)*(s-z))^(1/2)
    ## S <- sqrt(s*(s-x)*(s-y)*(s-z)) # 平方根を求める関数を用いても良い
  return(S)
}
area(3,4,5) # 直角三角形で検算
area(12,13,5)
```

```
[1] 6
[1] 30
```

演習

- 非負の整数 n の階乗 $n!$ を求める関数を作成せよ．
- 整数 n の Fibonacci 数を求める関数を作成せよ．

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2}
\end{aligned}
$$

- 行列 X が与えられたとき，各列の平均を計算する関数を作成せよ．
- 前問で X がベクトルの場合にはその平均を計算するように修正せよ．
  *(**関数** `is.vector()` **が利用できる**)*