

クレジット:

UTokyo Online Education 統計データ解析 I 2017 小池祐太

ライセンス:

利用者は、本講義資料を、教育的な目的に限ってページ単位で利用することができます。特に記載のない限り、本講義資料はページ単位でクリエイティブ・コモンズ 表示-非営利-改変禁止 ライセンスの下に提供されています。

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

本講義資料内には、東京大学が第三者より許諾を得て利用している画像等や、各種ライセンスによって提供されている画像等が含まれています。個々の画像等を本講義資料から切り離して利用することはできません。個々の画像等の利用については、それぞれの権利者の定めるところに従ってください。



統計データ解析 I (平成 29 年度)

東京大学大学院数理科学研究科
統計データ解析教育研究グループ

村田 昇 (早稲田大学, 東京大学)

吉田朋広 (東京大学)

小池祐太 (首都大学東京, 東京大学)

第2章 ベクトルと行列の演算, 関数

データ解析, 多変量解析, パターン認識などで必要となる計算の多くはベクトルと行列を用いた計算である. この章では, これらを R 言語で実現する方法をまとめる.

2.1. ベクトルの計算

まずベクトルのみでのさまざまな計算を纏める. 以下ではベクトルを太字で, 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

$$(2.1) \quad \mathbf{a} = (a_1, a_2, \dots, a_k)$$

のように表す. また, ベクトル \mathbf{a} の第 i 成分を指す場合には $(\mathbf{a})_i$ のように書くこともある.

2.1.1. 和. 同じ長さのベクトルの和および差

$$(2.2) \quad \mathbf{a} \pm \mathbf{b} = (a_1 \pm b_1, a_2 \pm b_2, \dots, a_k \pm b_k)$$

は, 数値の和と差のように扱うことができる. 成分による表現では

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

と書くことができる.

```
> # ベクトルの加減・スカラー倍
> a <- 1:3
> b <- 4:6
> a + b # 足し算
[1] 5 7 9
> a + 1:6 # 長さが異なる場合は足りない方が周期的に拡張される
[1] 2 4 6 5 7 9
> a + 1:5 # 一方の長さがもう一方の長さの整数倍でない場合は警告
[1] 2 4 6 5 7
> a - b # もちろん引き算も可
[1] -3 -3 -3
> 2 * a # スカラー倍
[1] 2 4 6
```

(vector-sum2.r)

2.1.2. 積. ベクトルの積は通常内積

$$(2.3) \quad \mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^k a_i b_i$$

を指すが, データ解析においては要素毎の積 (Hadamard product, Schur product)

$$(2.4) \quad \mathbf{a} \circ \mathbf{b} = (a_1 b_1, a_2 b_2, \dots, a_k b_k)$$

すなわち

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

を計算する場合も多い。2つの意味での積が簡単に計算できるように二項演算子`%*%`(内積) および`*`(要素毎の積) が定義されている。

```
> a <- 1:3
> b <- 4:6
> a %*% b # 内積 (計算結果は 1x1 行列)
      [,1]
[1,]    32
> try(a %*% (1:6))
> # 長さが異なるとエラー (try はエラーが出ても作業を続行するための関数)
> a * b # 要素毎の積 (計算結果はベクトル)
[1]  4 10 18
> a * 1:6 # 長さが異なる場合は足りない方が周期的に拡張される
[1]  1  4  9  4 10 18
> a/b # 除算も成分ごと
[1] 0.25 0.40 0.50
```

(vector-prod2.r)

2.1.3. 初等関数の適用. ベクトルに初等関数 (\sin, \exp, \dots 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 \mathbf{a} に関数 \sin を適用した結果 $\sin(\mathbf{a})$ は

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

となる。

```
> a <- (1:6) * pi/2
> sin(a) # 数値誤差のため正確に 0 とならない成分がある
[1]  1.000000e+00  1.224647e-16 -1.000000e+00 -2.449294e-16  1.000000e+00
[6]  3.673940e-16
> exp(a)
[1]  4.810477  23.140693  111.317778  535.491656  2575.970497
[6] 12391.647808
> log(a)
[1] 0.4515827 1.1447299 1.5501950 1.8378771 2.0610206 2.2433422
```

(vector-fun.r)

演習 2.1. ベクトルの計算をしてみよう。

- (1) ベクトルの内積から2つのベクトルがなす角を求めよ。
- (2) 2次元および3次元ベクトルの積としては、これら以外に“外積”がある。どのように計算すればよいか調べよ。

2.2. 行列とその演算

次に行列のみでのさまざまな計算を纏める。以下では行列を大文字で、その要素は下付き添字で表現する。例えば $m \times n$ 行列は

$$(2.5) \quad A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

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。

2.2.1. 和. 同じ大きさの行列の和および差

$$(2.6) \quad (A \pm B)_{ij} = a_{ij} \pm b_{ij}$$

は, ベクトルと同じように記述することができる.

```
> (A <- matrix(1:6,nrow=2,ncol=3)) # ベクトルの行列化
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> (B <- rbind(c(2, 3, 5),c(7, 11, 13))) # 行ベクトルを連結
      [,1] [,2] [,3]
[1,]    2    3    5
[2,]    7   11   13
> (C <- cbind(c(0, 0),c(0, 1),c(1, 0))) # 列ベクトルを連結
      [,1] [,2] [,3]
[1,]    0    0    1
[2,]    0    1    0
> A + B - C
      [,1] [,2] [,3]
[1,]    3    6    9
[2,]    9   14   19
```

(matrix-sum2.r)

2.2.2. 積. 行列の積

$$(2.7) \quad (AB)_{ij} = \sum_{k=1}^m a_{ik}b_{kj}$$

は, 左側の行列の行ベクトルと右側の行列の列ベクトルの内積を各要素とする行列となるので, 左側の行列の行数と右側の行列の列数が一致する場合のみ定義される. この積は二項演算子`%*%`を用いて計算する. なお, 行列の転置 (transpose) は関数 `t()` を用いて計算することができ, ある行列とその転置行列の積が簡単に計算できる. これは分散などの計算に活躍する.

一方, ベクトルと同様に同じ大きさの行列の要素毎の積 (Hadamard product, Schur product)

$$(2.8) \quad (A \circ B)_{ij} = a_{ij}b_{ij}$$

も簡単に計算できるようになっており, これは二項演算子`*`を用いて計算する.

```
> (A <- matrix(1:6,nrow=2,ncol=3)) # ベクトルの行列化
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> (B <- rbind(c(2, 3, 5),c(7, 11, 13))) # 行ベクトルを連結
      [,1] [,2] [,3]
[1,]    2    3    5
[2,]    7   11   13
> (C <- cbind(c(2, 3, 5),c(7, 11, 13))) # 列ベクトルを連結
      [,1] [,2]
[1,]    2    7
[2,]    3   11
[3,]    5   13
> A * B # 要素毎の積
```

```

      [,1] [,2] [,3]
[1,]    2    9   25
[2,]   14   44   78
> A / B # 除算も成分ごと
      [,1]      [,2]      [,3]
[1,] 0.5000000 1.0000000 1.0000000
[2,] 0.2857143 0.3636364 0.4615385
> A %*% C # 行列の積 (結果は 2x2 行列)
      [,1] [,2]
[1,]   36  105
[2,]   46  136
> C %*% B # 行列の積 (結果は 3x3 行列)
      [,1] [,2] [,3]
[1,]   53   83  101
[2,]   83  130  158
[3,]  101  158  194
> A %*% t(A) # 行列 A とその転置行列の積 (結果は 2x2 行列)
      [,1] [,2]
[1,]   35   44
[2,]   44   56

```

(matrix-prod2.r)

2.2.3. 初等関数の適用. ベクトルの場合と同様に、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

```

> a <- (1:6) * pi/2
> A <- matrix(a, 2, 3)
> sin(A)
      [,1]      [,2]      [,3]
[1,] 1.000000e+00 -1.000000e+00 1.000000e+00
[2,] 1.224647e-16 -2.449294e-16 3.67394e-16
> exp(A)
      [,1]      [,2]      [,3]
[1,] 4.810477 111.3178 2575.97
[2,] 23.140693 535.4917 12391.65
> log(A)
      [,1]      [,2]      [,3]
[1,] 0.4515827 1.550195 2.061021
[2,] 1.1447299 1.837877 2.243342

```

(matrix-fun.r)

2.2.4. 行列式とトレース. 行列に特有な量として行列式とトレース (対角成分の総和) があるが、行列式は関数 $\det()$ を用いて計算することができる。一方、トレースは専用の関数は用意されていないが、対角成分を取り出す関数 $\text{diag}()$ とベクトルの和を計算する関数 $\text{sum}()$ を用いて簡単に計算できる。

```

> (A <- matrix(1:9, nrow=3, ncol=3))
      [,1] [,2] [,3]
[1,]    1    4    7

```

```
[2,]  2  5  8
[3,]  3  6  9
> det(A) # 行列式 (determinant)
[1] 0
> sum(diag(A)) # トレース (trace)
[1] 15
(matrix-det2.r)
```

2.2.5. 逆行列. 正方行列の逆行列 (inverse matrix) を求めるには関数 `solve()` を用いる.

```
> (A <- matrix(c(2, 3, 5, 7, 11, 13, 17, 19, 23),nrow=3,ncol=3))
      [,1] [,2] [,3]
[1,]  2   7  17
[2,]  3  11  19
[3,]  5  13  23
> (B <- solve(A)) # 逆行列の計算
      [,1] [,2] [,3]
[1,] -0.07692308 -0.7692308  0.69230769
[2,] -0.33333333  0.5000000 -0.16666667
[3,]  0.20512821 -0.1153846 -0.01282051
> A %*% B
      [,1] [,2] [,3]
[1,]  1.000000e+00  0 -1.110223e-16
[2,] -4.440892e-16  1 -1.942890e-16
[3,]  0.000000e+00  0  1.000000e+00
> B %*% A
      [,1] [,2] [,3]
[1,]  1.000000e+00  3.552714e-15  1.776357e-15
[2,] -2.220446e-16  1.000000e+00  0.000000e+00
[3,]  0.000000e+00 -1.387779e-16  1.000000e+00
(matrix-inv2.r)
```

演習 2.2. 行列の計算をしてみよう.

- (1) 単位行列と成分が全て 1 の行列を作成せよ.
- (2) 適当な 2 次正方行列 A に対して, Hamilton-Cayley の定理

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

の成立を確認せよ. ここに, E_2 は 2 次単位行列, O は 2 次正方零行列であり, また $\text{tr}(A)$, $\det(A)$ はそれぞれ A のトレース (対角成分の総和), 行列式を表す.

2.3. ベクトルと行列の計算

2.3.1. 行列とベクトルの積. 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.

```
> (A <- matrix(1:16,nrow=4,ncol=4))
      [,1] [,2] [,3] [,4]
[1,]  1   5   9  13
[2,]  2   6  10  14
```

```

[3,] 3 7 11 15
[4,] 4 8 12 16
> (b <- c(2, 3, 5, 7))
[1] 2 3 5 7
> A %*% b # 列ベクトルとして計算
      [,1]
[1,] 153
[2,] 170
[3,] 187
[4,] 204
> b %*% A # 行ベクトルとして計算
      [,1] [,2] [,3] [,4]
[1,] 51 119 187 255

```

(linear-calc2.r)

2.3.2. 連立方程式. データ解析の様々な場面で連立一次方程式が現れる。これは行列とベクトルで表現されるので、行列とベクトルを与えることによって連立一次方程式を解く関数 `solve()` が用意されている。

```

> (A <- matrix(c(2, 3, 5, 7, 11, 13, 17, 19, 23),3,3))
      [,1] [,2] [,3]
[1,] 2 7 17
[2,] 3 11 19
[3,] 5 13 23
> (b <- c(2, 3, 5))
[1] 2 3 5
> (x <- solve(A,b)) # A x = b を解く
[1] 1.000000e+00 -2.220446e-16 5.124106e-17
> A %*% x # b と一致するか確認
      [,1]
[1,] 2
[2,] 3
[3,] 5
> ## 行列方程式
> E2 <- diag(1,2,2)
> A <- matrix(c(2,1,1,1),2,2)
> (X <- solve(A,E2)) # A X = E2 を解く
      [,1] [,2]
[1,] 1 -1
[2,] -1 2
> A%*%X # E2 と一致するか確認
      [,1] [,2]
[1,] 1 0
[2,] 0 1

```

(linear-eq2.r)

演習 2.3. 行列とベクトルの計算をしてみよう。

- (1) 適当な 2 次元のベクトルを 120 度回転しなさい。
- (2) $n \times n$ 行列 A と n 次のベクトル b を作成し、 $A \%*\% b + b \%*\% A$ を計算せよ (エラーになる)。何故、そうなるか理由を考えなさい。
- (3) 連立方程式の問題を作成し、それを解きなさい。

2.4. 関数

2.4.1. 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. 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. R 言語においてもこれらの構文は用意されており, 制御文を使うことによって次節で述べるような複雑な計算を行う関数を定義することができる.

```
> ### 条件分岐 (if)
> x <- 5
> if(x > 0) { # 正か否か判定
+   ## 条件が真の場合に実行
+   print("positive")
+ } else {
+   ## 条件が偽の場合に実行
+   print("negative")
+ }
[1] "positive"
> ## else 以下はなくても動く
> if(x > 0) {
+   print("positive")
+ }
[1] "positive"
> ## 評価が簡便な場合
> ifelse(x < 0, "true", "not true")
[1] "not true"
> ### 繰り返し (for, while, repeat)
> y <- 0
> for(i in 1:10) { # 1-10 の合計を計算
+   y <- y + i
+ }
> print(y)
[1] 55
> z <- 1
> n <- 0
> while(z < 100) {# 100 以上になるまで 2 倍し続ける
+   z <- 2 * z
+   n <- n + 1
+ }
> print(z) # 100 を超えた際の z の値 128
[1] 128
> print(n) # 条件を満たすまでの回数 7
[1] 7
```

(control2.r)

2.4.2. 関数の定義. 一般に関数とは入力を規則に従って変換し出力する仕組みを指す. R では, 入力を引数 (argument), 出力を返値 (value) と呼び, 関数 `function()` を用いて自由に関数を定義することができる.

```
> ### 階乗を計算する関数
> fact <- function(n){ # 素直に計算
+   ifelse(n>0, prod(1:n), 1)
+ }
> fact2 <- function(n){ # 再帰的に定義
+   if(n>0) {
+     return(n*fact2(n-1))
+   }
+ }
```

```
+   } else {  
+     return(1)  
+   }  
+ }  
> fact(10)  
[1] 3628800  
> fact2(10)  
[1] 3628800
```

(function.r)

同じ機能を持つ関数でも定義の仕方はいろいろ工夫できる。

演習 2.4. 以下の機能を持つ関数を作ってみよう。

- (1) 2次方程式の3つの係数を入力すると解を出力する関数を作成せよ。
- (2) 第1項, 第2項, および項数を入力すると Fibonacci 数列を指定された項数まで出力する関数を作成せよ。なお, Fibonacci 数列とは下記の漸化式を満たす数列である。

$$a_n = a_{n-1} + a_{n-2}, n = 3, 4, \dots$$

- (3) 自身で仕様を決めて, それを満たす関数を作成せよ。

2.5. 参考文献

1. U. リゲス著, 石田基広訳「R の基礎とプログラミング技法」, 丸善出版 (2012 年).