クレジット：
UTokyo Online Education 統計データ解析 Ⅱ 2018 小池祐太

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

統計データ解析 II (平成３０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 (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．

以下では特に断らない限り, 実数を成分とするベクトル・行列のみを扱うことにする. ただし, 複素数を成分とするベクトル・行列の R 言語における扱いもほぼ同様である (行列演算における諸定義が実数と複素数との場合で多少異なることがあるので, 興味がある者は線形代数学のテキストを参照すること).

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

$$\boldsymbol{a} = (a_1, a_2, \ldots, a_k) \tag{2.1}$$

のように表す．また，ベクトル $\boldsymbol{a}$ の第 i 成分を指す場合には $(\boldsymbol{a})_i$ 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

$$\boldsymbol{a} \pm \boldsymbol{b} = (a_1 \pm b_1, a_2 \pm b_2, \ldots, a_k \pm b_k) \tag{2.2}$$

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

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

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出ない場合は警告
[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

$$\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{k} a_i b_i \tag{2.3}$$

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

$$\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, a_2 b_2, \ldots, a_k b_k) \tag{2.4}$$

すなわち

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

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, . . . 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(a)` は

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

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．多変量解析で現れる統計量の計算を見通しよく行うためには, 行列に対する種々の演算を利用するのが便利である．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

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

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．

2.2.1. 和. 同じ大きさの行列の和および差

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

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

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

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，行列の各成分の行番号と列番号を入れ替えて得られる行列を**転置行列 (transposed matrix)** と呼ぶが，それは関数`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)

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

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 ($\sin, \exp, \ldots$ など) を適用すると, 成分ごとに計算した結果が返される. 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 1.00000e+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がある．連立 1 次方程式

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2, \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

は, n 次正方行列 A, n 次元列ベクトル $\boldsymbol{x}$ および $\boldsymbol{b}$ をそれぞれ

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \qquad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

で定めると, 行列方程式

$$A\boldsymbol{x} = \boldsymbol{b} \tag{2.9}$$

の形で書くことができるが, この方程式の解の存在や一意性を議論する上で, 行列 A の**行列式 (determinant)** と呼ばれる量が重要な役割を果たす (以下の節を参照すること). 行列 A の行列式は記号 $\det A$ で表される．一般の行列に対する行列式の定義は複雑なためここでは割愛する (興味のある者は線形代数学のテキストを参照すること). A が 2 次正方行列の場合は単純であり,

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

となる．この講義では, 行列式は多変量正規分布の定義において明示的に現れる他, 重回帰分析でも有用である．R においては, 行列式は関数 `det()` を用いて計算することができる．

一方, 正方行列の**トレース (trace)** は対角成分 (対角線上の成分) の総和として定義される．正方行列 A のトレースは記号 $\mathrm{tr}\, A$ で表されることが多い:

$$\mathrm{tr}\, A = \sum_{i=1}^{n} a_{ii}.$$

R においては, トレースを計算するための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 `diag()` とベクトルの和を計算する関数 `sum()` を用いて簡単に計算できる．トレースは, 二次形式で表される統計量の分布の計算において有用である．

```
> (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-det3.r)

2.2.5. 逆行列. 対角成分以外の成分がすべて 0 であるような正方行列を**対角行列**と呼ぶ．特に, 対角成分がすべて 1 であるような対角行列を**単位行列**と呼ぶ．この講義では, n 次単位行列を記号 E_n で表す．R では, n 次単位行列は `diag(n)` で生成することができる．

n 次正方行列 A に対して, n 次正方行列 B が

$$AB = BA = E_n$$

を満たすとき, B を A の**逆行列 (inverse matrix)** と呼び, 記号 A^{-1} で表す (逆行列は存在すれば一意的に定まる). 逆行列をもつような正方行列は**正則 (regular)** もしくは**非特異 (non-singular)** であるという. 逆行列は常に存在するとは限らない. 正方行列 A が正則である (つまり, 逆行列をもつ) ための必要十分条件は, A の行列式 $\det A$ が 0 でないことであるという事実が知られている. 正則行列の逆行列を求めるには関数 `solve()` を用いる (正則でない行列に適用するとエラーとなる).

データ解析においては，正則でも正方でない行列の**一般化逆行列 (擬似逆行列; pseudo-inverse matrix)** がしばしば必要となる．一般化逆行列 $A^\dagger$ とは

$$AA^\dagger A = A \tag{2.10}$$

が成り立つ行列のことで，いくつかの定義がある．一般化逆行列の中で良く用いられるのは **Moore-Penrose の一般化逆行列 (Moore-Penrose pseudoinverse)** と呼ばれるもので，これを A^+ と書くことにすると

$$AA^+A = A \tag{2.11}$$

$$A^+AA^+ = A^+ \tag{2.12}$$

$$(AA^+)^\top = AA^+ \qquad (\top \text{ は転置行列の意}) \tag{2.13}$$

$$(A^+A)^\top = A^+A \tag{2.14}$$

が成立する行列である．これは `MASS` パッケージの中の関数 `ginv()` を用いて求めることができる．

```
> (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 # 検算
              [,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 # 検算
              [,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
> det(A) # 0 でない
[1] -78
> (S <- matrix(1:9,3,3)) # 正則でない行列
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> solve(S) # エラーが出る
> det(S) # 0
[1] 0
```

```
> ### 一般化逆行列を求める場合
> # install.packages("MASS") # MASS package をインストール (必要な場合)
> require(MASS) # MASS package を読み込む
> (C <- matrix(1:6,nrow=2,ncol=3))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> (D <- ginv(C)) # 一般化逆行列の計算
           [,1]       [,2]
[1,] -1.3333333  1.0833333
[2,] -0.3333333  0.3333333
[3,]  0.6666667 -0.4166667
> C %*% D %*% C # CC^+C=C であることを確認
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

(matrix-inv3.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) 関数 `ginv()` で計算される行列が Moore-Penrose の一般化逆行列になっていることを確かめよ．

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 (2.9) 式のように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 $\boldsymbol{b}$ 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. 固有値と固有ベクトル

一般に n 次正方行列 A に対して, 複素数 λ と零ベクトルでない n 次元ベクトル $\boldsymbol{x}$ が

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

を満たすとき, λ を A の**固有値 (eigenvalue)**, $\boldsymbol{x}$ を λ に対する**固有ベクトル (eigenvector)** と呼ぶ. 固有値及び固有ベクトルはデータ解析にしばしば用いられ, R では関数 `eigen()` で求められる.

A の転置行列 $A^\top$ が A 自身に一致する場合 ($A^\top = A$), A は**対称**であるという. A が対称ならば, A の固有値はすべて実数であり, かつ A の固有ベクトルを n 個並べて得られる正則行列 V で, $V^{-1}AV$ が対角行列となるようなものがとれることが知られている (この場合 $V^{-1} = V^\top$ となるようにとれる). この操作を A の**対角化 (diagonarization)** と呼ぶ. 対角化に関する詳細は線形代数学のテキストを参照すること.

```
> (A <- matrix(c(1, -1, -1, 1), 2, 2))
     [,1] [,2]
[1,]    1   -1
[2,]   -1    1
> r <- eigen(A) # 結果は固有値と固有ベクトルからなるリスト
> r$values # 固有値
```

```
[1] 2 0
> r$vectors # 固有ベクトルからなる行列
           [,1]       [,2]
[1,] -0.7071068 -0.7071068
[2,]  0.7071068 -0.7071068
> # 第 i 列が r$values[i] に対する固有ベクトルに対応
> solve(r$vectors) %*% A %*% r$vectors # 対角化
     [,1] [,2]
[1,]    2    0
[2,]    0    0
```

(eigen.r)

演習 2.4. A が非負定値 n 次対称行列, すなわち固有値がすべて非負の n 次対称行列のとき, A の対角化を考えることで $A = B^2$ を満たす非負定値 n 次対称行列 B が求められる．この行列 B を計算するプログラムを作成せよ．

2.5. 関数

2.5.1. 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)

2.5.2. 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 2.5. 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.

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

(3) 自身で仕様を決めて, それを満たす関数を作成せよ.

2.6. 参考文献

1. 金明哲著「R によるデータサイエンス (第 2 版)」, 森北出版 (2017 年).
2. 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).
3. 吉田朋広著「数理統計学」, 朝倉書店 (2006 年).

2.7. 参考: その他の行列の演算

2.7.1. ノルム. 行列のノルムにはいくつか定義があるが, 関数 `norm()` によって作用素ノルム ($p = 1$ および $p = \infty$),

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}| \tag{2.15}$$

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}| \tag{2.16}$$

Frobenius ノルム，

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{2.17}$$

最大ノルム，

$$\|A\|_{\max} = \max\{|a_{ij}|\} \tag{2.18}$$

およびスペクトルノルム

$$\|A\|_2 = \sigma_{\max}(A), \quad (A \text{ の最大特異値}) \tag{2.19}$$

を計算することができる．

```
> (A <- matrix(c(1,0,0,0,2,0,1,-1,-3,0,0,0),nrow=3,ncol=4))
     [,1] [,2] [,3] [,4]
[1,]    1    0    1    0
[2,]    0    2   -1    0
[3,]    0    0   -3    0
> norm(A,type="O") # one norm
[1] 5
> norm(A,type="I") # infinity norm
[1] 3
> norm(A,type="F") # Frobenius norm
[1] 4
> norm(A,type="M") # max norm
[1] 3
> norm(A,type="2") # 2-norm (spectral norm)
[1] 3.408689
```

2.7.2. 特異値分解. 大規模データに対するデータ解析では，正方行列でない任意の行列に対する分解が必要となることがある．一般に $n \times p$ 行列 A に対して，$q = \min\{n, p\}$ とすると，$n \times q$ 行列 U，$p \times q$ 行列 V，成分が非負の q 次対角行列 D が存在して，$U^\top U = V^\top V = E_q$ を満たし，かつ

$$A = UDV^\top$$

と書けることが知られている．この分解を A の**特異値分解 (singular value decomposition)** と呼び，D の対角成分を A の**特異値 (singular value)** と呼ぶ．A の特異値は A から一意的に定まることが知られている．特異値分解は関数 `svd()` で実行できる．

```
> (A <- matrix(1:6, nrow = 2))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> s <- svd(A) # 結果は特異値と行列 U,V からなるリスト
> s$d # 特異値
[1] 9.5255181 0.5143006
> s$u # 行列 U
           [,1]       [,2]
[1,] -0.6196295 -0.7848945
[2,] -0.7848945  0.6196295
> s$v # 行列 V
```

```
          [,1]       [,2]
[1,] -0.2298477  0.8834610
[2,] -0.5247448  0.2407825
[3,] -0.8196419 -0.4018960
> s$u %*% diag(s$d) %*% t(s$v) # 行列 A の再現
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

(svd.r)

2.7.3. コレスキー分解. n 次対称行列 A の固有値がすべて正であるとき, A は**正定値 (positive-definite)** であるという. この場合, n 次上三角行列 R で

$$A = R^\top R$$

を満たすものが一意的に存在する (上三角行列とは, 対角成分より下側の成分がすべて 0 であるような正方行列のこと). 上の分解を A の**コレスキー分解 (Choleski decomposition)** と呼ぶ. コレスキー分解は, 大規模データの相関行列の計算やモデル化において有用である. R では, 関数 `chol()` によってコレスキー分解を計算できる.

```
> (A <- matrix(c(1, 0.5, 0.5, 0.5, 1, 0.5,
+                0.5, 0.5, 1), 3, 3))
     [,1] [,2] [,3]
[1,]  1.0  0.5  0.5
[2,]  0.5  1.0  0.5
[3,]  0.5  0.5  1.0
> (R <- chol(A)) # コレスキー分解
     [,1]      [,2]      [,3]
[1,]    1 0.5000000 0.5000000
[2,]    0 0.8660254 0.2886751
[3,]    0 0.0000000 0.8164966
> t(R) %*% R # 検算
     [,1] [,2] [,3]
[1,]  1.0  0.5  0.5
[2,]  0.5  1.0  0.5
[3,]  0.5  0.5  1.0
> crossprod(R) # 上式と同義
     [,1] [,2] [,3]
[1,]  1.0  0.5  0.5
[2,]  0.5  1.0  0.5
[3,]  0.5  0.5  1.0
> # chol(matrix(c(1,0,0,-1),2,2)) # 正定値でないとエラー
```

(chol.r)