ＡＩ基礎セミナー

第４回
数学の基礎（1回目：代数学）

改訂履歴

日付	担当者	内容
2021/05/08	M.Takeda	Git 公開

目次

(1) はじめに

・第４回から第６回にわたり、機械学習で使用する数学の基礎と
その Python での実装について概観します。

・「第４回　数学の基礎（1回目）」では、数列と級数、スカラーとベクトルと行列などの代数学に焦点をあて、
「ムーア・ペンローズの擬似逆行列（Moore-Penrose pseudo-inverse matrix）」を用いて
「Ｄ次元線形回帰モデル」を解析的に解くことを目標とします。
これは回帰モデルの解析解を与えるものです。

・本資料は、機械学習で用いる基本的な数学の参照資料として用意してあります。
はじめから順番に読み進めれば、上記の目標に到達するように構成してあります。

・確認問題は、本文を要約したような問題となっております。
解いてみて、理解度を確認して頂けたらと思います。

(2) 数列と級数

(2.1) 数列と級数と総和

(2.1.1) 数列

・正の整数 x（x ∈ 正の整数の集合）に対して関数 a(x) を x の順に並べたものを
「数列（すうれつ、numerical sequence)」と言います。

a(1), a(2), ・・・, a(n), ・・・

・x＝n における関数 a の値 a(n) を a_n で表現し、次のように記します。

a_1, a_2, ・・・, a_n, ・・・

これを、慣習的に以下の様に表現します。

$\{ a_n \}$　　または　　$\{ a_k \}_{k=1,2,\ldots,n,\ldots}$

数列の各々の数を「項（こう、term)」と言い、特に a_1 を「初項（しょこう、first term)」、
添字 n に対応する項 a_n を「一般項（いっぱんこう、general term)」と言います。
数列の最後の項を数列の「末項（まっこう、last term)」と言います。
自然数（1,2,・・・,∞）を数列として扱う場合のように、「末項」が存在するとは限りません。

・自然数を数列として扱う場合のように末項が定まらないような数列は、
「無限数列（むげんすうれつ、infinite sequence)」と呼ばれ、
末項を持つ数列は「有限数列（ゆうげんすうれつ、finite sequence)」と呼ばれます。

(2.1.2) 級数と総和

・無限数列 $\{ a_n \}$ の項を、加算記号 + でつないだ式

$a_1 + a_2 + \cdots + a_n + \cdots$

を「無限級数（むげんきゅうすう、infinite series)」といい、
ギリシャ文字シグマの大文字 ″Σ″ で、以下の様に表現します。

無限級数	$\sum_{n=1}^{\infty} a_n$　（本資料では　$\Sigma_{n=1}{}^{\infty} a_n$　で記すことにします）

数列のはじめの n 項までの和を、「第ｎ部分和」(部分和（ぶぶんわ、patial sum))と呼び、
Sn または総和記号Σ で、以下の様に表現します。

第ｎ部分和	$S_n = \sum_{k=1}^{n} a_k = a_1 + a_2 + \cdots + a_n$

【出典・参考】
⇒ 「数学公式事典」 黒田孝朗・須田貞之著 文研出版 1978
⇒ https://ja.wikipedia.org/wiki/数列
⇒ https://ja.wikipedia.org/wiki/総和
⇒ https://ja.wikipedia.org/wiki/級数

(2.2) 等差数列

・数列のうち、任意の自然数 n に対して、隣り合う２項 a_n と a_{n+1} の差が一定のものを「等差数列」と言い、隣り合う２項間の差を「公差」といいます。

・初項を a、公差を d とする時、

等差数列

一般項　$a_n = a + (n-1)d$

第ｎ部分和　$S_n = \{2a + (n-1)d\}\ n / 2$

第ｎ部分和の公式の導出は、以下のとおりです：

$$S_n = a_1 + a_2 + \cdots + a_n$$
$$= a_n + a_{n-1} + \cdots + a_1$$

$$\therefore\ 2S_n = (a_1+a_n) + (a_2+a_{n-1}) + \cdots + (a_n+a_1)$$
$$= (a + a+(n-1)d) + (a+d + a+(n-2)d) + \cdots + (a+(n-1)d + a)$$
$$= \{2a + (n-1)d\}\ n$$
$$\therefore\ S_n = \{2a + (n-1)d\}\ n / 2$$

（例：初項1、公差1 の等差数列）

1, 2, 3, ・・・, n, n+1, ・・・

一般項　$a_n = n$

第ｎ部分和　$S_n = \{2+(n-1)\}\ n / 2$

第10部分和　$S_{10} = 55$

（例：初項1、公差2 の等差数列）

1, 3, 5, ・・・, 2n-1, 2n+1, ・・・

一般項　$a_n = 2n - 1$

第ｎ部分和　$S_n = \{2+(n-1)2\}\ n / 2 = n**2$

第10部分和　$S_{10} = 100$

(2.3) 等比数列

・数列のうち、特に任意の自然数 n に対して、隣り合う２項 a_n と a_{n+1} の比が一定のものを「等比数列」と言い、隣り合う２項間の比を「公比」といいます。

・初項を a、公比を r とする時、

等比数列

一般項　$a_n = a\,r^{(n-1)}$

第ｎ部分和　$S_n = \begin{cases} a\,(r^n - 1)/(r - 1) & (r \neq 1 \text{ の場合}) \\ na & (r = 1 \text{ の場合}) \end{cases}$

第ｎ部分和の公式の導出は、以下のとおりです：

$$
\begin{aligned}
S_n &= a_1 + a_2 + \cdots + a_n \\
&= a + ar + ar^2 + \cdots + ar^{(n-1)} \\
rS_n &= ar + ar^2 + \cdots + ar^{(n-1)} + ar^n
\end{aligned}
$$

$$
\begin{aligned}
\therefore \quad rS_n - S_n &= ar^n - a = a(r^n - 1) \\
\therefore \quad (r-1)S_n &= a(r^n - 1) \\
\therefore \quad S_n &= a(r^n - 1)\,/\,(r-1) \qquad (\text{但し} r \neq 1 \text{の場合})
\end{aligned}
$$

（例：初項1、公比1 の等比数列）

1, 1, 1, ・・・, 1, 1, ・・・

一般項　$a_n = 1$

第ｎ部分和　$S_n = n$

第10部分和　$S_{10} = 10$

（例：初項1、公比2 の等比数列）

1, 2, 4, ・・・, 2^{n-1}, 2^n, ・・・

一般項　$a_n = 2^{n-1}$

第ｎ部分和　$S_n = 1(2^n - 1)/1 = 2^n - 1$

第10部分和　$S_{10} = 2^{10} - 1 = 1023$

(2.4) 総乗

・「総乗（そうじょう、product)」とは、数列の全ての積のことです。

・無限数列 $\{ a_n \}$ の項を、乗算記号 × でつないだ式

$$a_1 \times a_2 \times \cdots \times a_n \times \cdots$$

を「無限乗積（むげんじょうせき、infinite product)」または無限積といい、
ギリシャ文字パイの大文字 ″Π″ で、以下の様に表現します。

無限乗積	$\prod_{n=1}^{\infty} a_n$	または	$\prod a_n$	（本資料では	$\prod_{n=1}^{\infty} a_n$	で記すことにします）

・数列のはじめの n 項までの積を、「第ｎ部分積」(部分積（ぶぶんせき、partial product))と呼び、
Pn または総乗記号Π で、以下の様に表現します。

第ｎ部分積	$P_n = \prod_{k=1}^{n} a_k = a_1 \times a_2 \times \cdots \times a_n$

・「積の対数は（底が等しい）対数の和に等しい」という対数関数の性質を用いて、
総乗記号Π は総和記号Σ と、以下の様な関係があります。

$$\log(P_n) = \log\left(\prod_{k=1}^{n} a_k\right) = \log(a_1 \times a_2 \times \cdots \times a_n)$$
$$= \log(a_1) + \log(a_2) + \cdots + \log(a_n)$$
$$= \sum_{k=1}^{n} \log(a_k)$$

（例：初項1、公差1 の等差数列の総乗）

1, 2, 3, ・・・, n, n+1, ・・・	
一般項	$a_n = n$
第ｎ部分積	$P_n = 1 \times 2 \times \cdots \times n$
第10部分積	$P_{10} = 3628800$

【出典・参考】
⇒ https://ja.wikipedia.org/wiki/総乗
⇒ https://ja.wikipedia.org/wiki/対数

(3) ベクトルとスカラー

(3.1) スカラー

・数字一個のことを「スカラー（scalar)」と言います。
・「スカラー」は、小文字で表現します。
　（例）　　a , b

(3.2) ベクトル

・数字を複数個並べたものを「ベクトル（vector)」と言います。
・ベクトルを構成する各数字を「要素（element)」または「成分（component)」と呼びます。
・ベクトルを構成する「要素」の個数を「次元（じげん、dimension)」と呼びます。

・「要素」を縦に n 個並べたものを「n次元縦ベクトル（column vector)」と言い、
　「要素」を横に n 個並べたものを「n次元横ベクトル（row vector)」と言います。

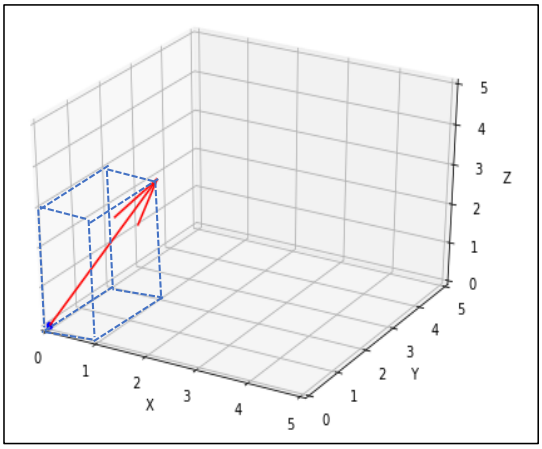
・「スカラー」は、１次元のベクトルと見做すことも出来ます。

・本資料では、「ベクトル」は要素の並びを括弧 () で括り、太文字の小文字で表現します。
　また、「ベクトル」**a** の i番目の要素を a_i で表現します（i=1～n)。

・向きを持つ線分を「有向線分」と言います。
　始点P_1 から終点P_2 に向かう有向線分を $\overrightarrow{P_1P_2}$ で表現します。

・これ以降は、特に断らない限り、
　3次元のベクトルについて説明します。

・座標原点Ｏ(0,0,0)を始点とし、
　Ｏから終点Ｐ(x,y,z)へ向かう有向線分 $\overrightarrow{OP}$ を
　「位置ベクトル（position vector)」と呼びます。
　位置ベクトルを **r** とすると、
　　r = $\overrightarrow{OP}$ = (x,y,z)
　となります。



※ 上図は、３次元ベクトル **r** = (1,2,3) を、
　位置ベクトルとして表示した例。

（例）

3次元縦ベクトルの例		（成分表示）	3次元横ベクトルの例			（成分表示）
a =	11	a_1 = 11		21	22	23	c_1 = 21
	12	a_2 = 12					c_2 = 22
	13	a_3 = 13					c_3 = 23

(Python での実装例)

```
#*****************************************************
# ﾘｽﾄ04-(03)-1_縦ベクトル・横ベクトル
#*****************************************************
import numpy as np

print("縦ベクトル a --------------------")
a = np.array([[11],[12],[13]])
print( 'type(a)=', type(a) )
print(a)

print("横ベクトル c --------------------")
c = np.array([21,22,23])
print( 'type(c)=', type(c) )
print(c)
```

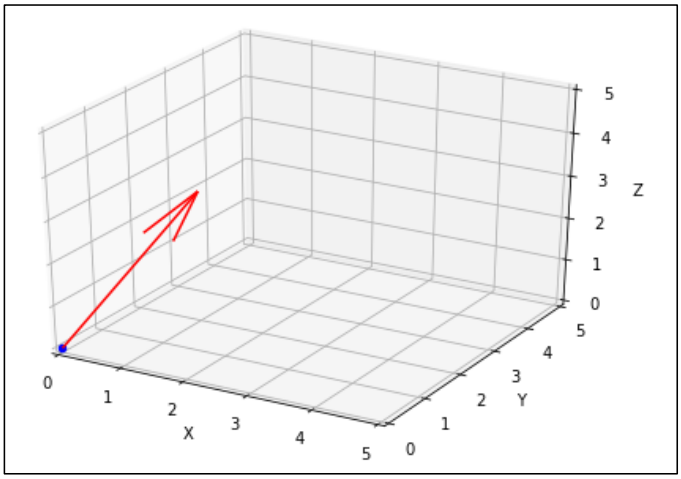
（実行結果）

```
縦ベクトル a ------------------------
type(a)= <class 'numpy.ndarray'>
[[11]
 [12]
 [13]]
横ベクトル c ------------------------
type(c)= <class 'numpy.ndarray'>
[21 22 23]
```

（ベクトルの描画の実装例）

```
#************************************************************
# ﾘｽﾄ04-(03)-2_ 3次元ベクトルを位置ベクトルとして図示
#************************************************************
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D #3次元プロットのためのimport
import numpy as np

fig = plt.figure()
ax = Axes3D(fig)

# 座標の描画範囲
LXs, LYs, LZs = 0,0,0
LXe, LYe, LZe = 5,5,5
ax.set_xlim([LXs, LXe])
ax.set_ylim([LYs, LYe])
ax.set_zlim([LZs, LZe])
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

# 原点(0,0,0) から座標(1,2,3) へ向かうベクトル
vs = np.array([0,0,0])
ve = np.array([1,2,3])

# 位置ベクトルとして描画
ax.scatter3D(vs[0],vs[1],vs[2], "o", color='blue') # 座標原点
ax.quiver(vs[0],vs[1],vs[2], ve[0],ve[1],ve[2], color='red',length=1, normalize=False)
plt.grid()
plt.draw()
plt.show()
```

（出力結果）



【出典・参考】

⇒ http://www.ftext.org/text/section/162

(3.3) ベクトルの和・差・スカラー倍

・「ベクトル」について、和とスカラー倍を次のように定義します：

(1) 和 **a + b**

$$\mathbf{a}+\mathbf{b}=\begin{pmatrix}a_1\\a_2\\\vdots\\a_n\end{pmatrix}+\begin{pmatrix}b_1\\b_2\\\vdots\\b_n\end{pmatrix}=\begin{pmatrix}a_1+b_1\\a_2+b_2\\\vdots\\a_n+b_n\end{pmatrix}$$

(2) スカラー倍 $\lambda\mathbf{a}$

$$\lambda\mathbf{a}=\lambda\begin{pmatrix}a_1\\a_2\\\vdots\\a_n\end{pmatrix}=\begin{pmatrix}\lambda a_1\\\lambda a_2\\\vdots\\\lambda a_n\end{pmatrix}$$

(3) $-\mathbf{a}$ はスカラー倍で $\lambda=-1$ と置いたものに等しい

$$-\mathbf{a}=-1\begin{pmatrix}a_1\\a_2\\\vdots\\a_n\end{pmatrix}=\begin{pmatrix}-a_1\\-a_2\\\vdots\\-a_n\end{pmatrix}$$

(4) 零ベクトル **0** はスカラー倍で $\lambda=0$ と置いたものに等しい

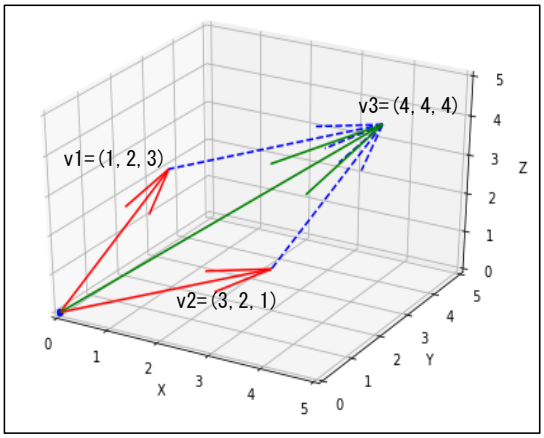
$$\mathbf{0}=\begin{pmatrix}0\\0\\\vdots\\0\end{pmatrix}$$

(5) ベクトルの和・スカラー倍について、以下の法則が成り立ちます：

（**a**、**b**、**c** は同一次元のベクトル、λ、μ はスカラーとします。）

ベクトルの和・スカラー倍についての法則

・和の交換法則	$\mathbf{a}+\mathbf{b}$	=	$\mathbf{b}+\mathbf{a}$
・和の結合法則	$(\mathbf{a}+\mathbf{b})+\mathbf{c}$	=	$\mathbf{a}+(\mathbf{b}+\mathbf{c})$
・０ベクトルとの和	$\mathbf{a}+\mathbf{0}$	=	$\mathbf{a}$
・和が０ベクトルになる逆元	$\mathbf{a}+(-\mathbf{a})$	=	$\mathbf{0}$
・スカラー倍の分配法則	$\lambda(\mathbf{a}+\mathbf{b})$	=	$\lambda\mathbf{a}+\lambda\mathbf{b}$
・スカラー倍の分配法則	$(\lambda+\mu)\mathbf{a}$	=	$\lambda\mathbf{a}+\mu\mathbf{a}$
・スカラー倍の分配法則	$(\lambda\mu)\mathbf{a}$	=	$\lambda(\mu\mathbf{a})$



※ 左図は、３次元ベクトル v1 = (1, 2, 3) と、
v2 = (3, 2, 1) の和 v3=(4, 4, 4) を、
位置ベクトルとして表示した例。

(Python での実装例)

```
#**************************************************
# ﾘｽﾄ04-(03)-3_ベクトルの和・差・スカラー倍
#**************************************************
import numpy as np

print("縦ベクトル a, b --------------")
a = np.array([[11],[12],[13]])
b = np.array([[21],[22],[23]])
lmbda = 2

print( 'type(a)=', type(a) )
print('Vector a', a)
print('Vector b', b)

print("演算 --------------")
print('a + b =')
print(a + b)

print('a + (-a) =')
print( a + (-a) )

print('lmbda =', lmbda)
print('lmbda * a =')
print( lmbda * a )
```

(実行結果)

```
縦ベクトル a, b --------------
type(a)= <class 'numpy.ndarray'>
Vector a [[11]
 [12]
 [13]]
Vector b [[21]
 [22]
 [23]]
演算 --------------
a + b =
[[32]
 [34]
 [36]]
a + (-a) =
[[0]
 [0]
 [0]]
lmbda = 2
lmbda * a =
[[22]
 [24]
 [26]]
```

(ベクトル和の描画の実装例)

```
#**************************************************
# ﾘｽﾄ04-(03)-4_三次元ベクトルの和を図示
#**************************************************
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D #3次元プロットのためのimport
import numpy as np

fig = plt.figure()
ax = Axes3D(fig)

# 座標の描画範囲
LXs, LYs, LZs = 0,0,0
LXe, LYe, LZe = 5,5,5
ax.set_xlim([LXs, LXe])
ax.set_ylim([LYs, LYe])
ax.set_zlim([LZs, LZe])
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

# ベクトル v1, v2 と v3=v1+v2
v0 = np.array([0,0,0]) # 座標原点
v1 = np.array([1,2,3])
v2 = np.array([3,2,1])
v3 = v1 + v2

# 各ベクトルを位置ベクトルとして描画
ax.scatter3D(v0[0],v0[1],v0[2],"o", color='blue') # 座標原点
ax.quiver(v0[0],v0[1],v0[2], v1[0],v1[1],v1[2], color='red',length=1, normalize=False)
ax.quiver(v0[0],v0[1],v0[2], v2[0],v2[1],v2[2], color='red',length=1, normalize=False)
ax.quiver(v0[0],v0[1],v0[2], v3[0],v3[1],v3[2], color='green',length=1, normalize=False)

# ベクトル和の補助線を描画
ax.quiver(v1[0],v1[1],v1[2], v2[0],v2[1],v2[2], color='blue',linestyle='--')
ax.quiver(v2[0],v2[1],v2[2], v1[0],v1[1],v1[2], color='blue',linestyle='--')
plt.grid()
plt.draw()
plt.show()
```

【出典・参考】

⇒ https://qiita.com/sci_Haru/items/21777764ae50996abd95

⇒ http://kamino.hatenablog.com/entry/python-3d-visualization

(3.4) 転置ベクトル

・「ベクトル」**a** の縦と横を入れ替えることを「転置 (transpose)」と言います。
転置してできたベクトルを「転置ベクトル (transposed vector)」と言い、

$\mathbf{a}^T$ または ${}^t\mathbf{a}$

と表現します。以降の説明では $\mathbf{a}^T$ で記述することとします。

・転置ベクトル $\mathbf{a}^T$ の更に転置ベクトル $(\mathbf{a}^T)^T$ は、元のベクトル **a** に等しいです。

(転置前の行列)

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

(転置行列)

$$\mathbf{a}^T = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix}$$

(Python での実装例)

```
#**************************************************
# ﾘｽﾄ04-(03)-5_転置ベクトル
#**************************************************
import numpy as np

print("縦ベクトル a とその転置ベクトル a.T")
a = np.array([[11],[12],[13]])
print('type(a)=', type(a))
print('a=', a)
print('type(a.T)=', type(a.T))
print('a.T=', a.T)

print("¥n横ベクトル b とその転置ベクトル b.T")
b = np.array([[21,22,23]])
print('type(b)=', type(b))
print('b=', b)
print('type(b.T)=', type(b.T))
print('b.T=', b.T)
```

(実行結果)

```
縦ベクトル a とその転置ベクトル a.T
type(a)= <class 'numpy.ndarray'>
a= [[11]
 [12]
 [13]]
type(a.T)= <class 'numpy.ndarray'>
a.T= [[11 12 13]]

横ベクトル b とその転置ベクトル b.T
type(b)= <class 'numpy.ndarray'>
b= [[21 22 23]]
type(b.T)= <class 'numpy.ndarray'>
b.T= [[21]
 [22]
 [23]]
```

(3.5) ベクトルの大きさ

・ベクトル **a** の大きさは、|**a**| と表現します。
　その値は、要素の二乗和の平方根です。
・これは、ベクトル **a** を位置ベクトルとみなした場合、原点Oから終点までの距離に相当します。

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

（ベクトルの大きさ）

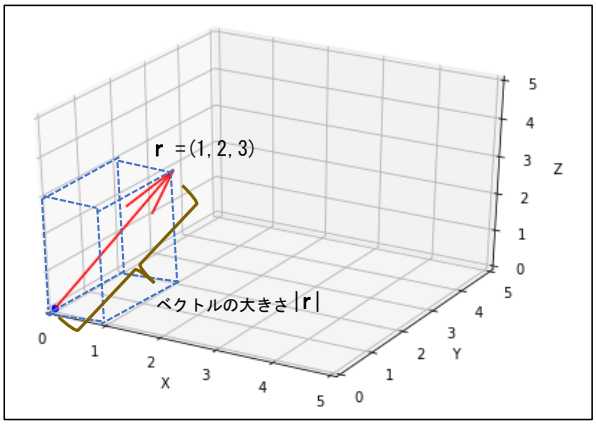
$$|\mathbf{a}| = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} = \sqrt{\sum_{k=1}^{n} a_k^2}$$

（Python での実装例）

```
#*****************************************************
# ﾘｽﾄ04-(03)-6_ベクトルの大きさ
#*****************************************************
import numpy as np

print("横ベクトル a とその大きさ")
a = np.array([1,2,3])
print('type(a)=', type(a))
print('a=', a, ", |a|=", np.linalg.norm(a))
```

（実行結果）

```
横ベクトル a とその大きさ
type(a)= <class 'numpy.ndarray'>
a= [1 2 3] , |a|= 3.7416573867739413
```



※ 左図は、３次元ベクトル **r** = (1,2,3) を、
　位置ベクトルとして表示し、
　原点と終点の距離が
　ベクトルの大きさであることを示します。

(3.6) 内積

・「内積（Inner Product）」は、同一次元の２つのベクトル同士の演算で、
対応する要素同士の積を取り、その総和を結果（スカラー値）とします。
ベクトル **a** とベクトル **b** の内積は、**a・b**（a ドット b）と表現します。

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

（内積）

$$\mathbf{a} \cdot \mathbf{b} = a_1 * b_1 + a_2 * b_2 + \cdots + a_n * b_n = \sum_{k=1}^{n} a_k b_k$$

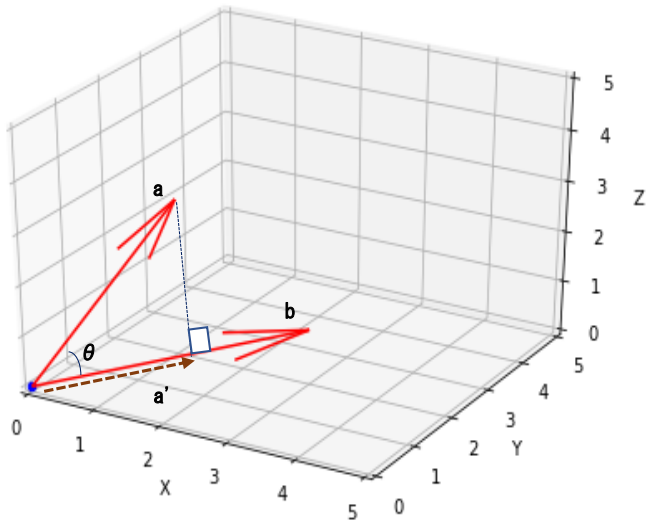
・内積**a・b** は数式としては、下記のように書くことができます。
これは、ベクトル **a** をベクトル **b** に射影したベクトルを **a'** としたとき、
a' と **b** の長さを掛け合わせた値になります。

内積

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos\theta \quad （\theta は\mathbf{a}と\mathbf{b}のなす角度）$$

$$= |\mathbf{a}'||\mathbf{b}| \quad （|\mathbf{a}'| = |\mathbf{a}|\cos\theta）$$

```
type(a)= <class 'numpy.ndarray'> , a= [1 2 3] , |a|= 3.7416573867739413
type(b)= <class 'numpy.ndarray'> , b= [3 2 1] , |b|= 3.7416573867739413
a・b=  10
```



左図は、
a = (1, 2, 3)
b = (3, 2, 1)
の場合の図示。

・上記の関係式を見てわかるように、内積 **a・b** は方向が近い程、大きな値を取ることがわかります。
このことから、内積はベクトル **a** とベクトル **b** の類似度の指標として使用されることもあります。

・後ほど、行列の積で触れますが、**a** と **b** の内積は、
a の転置ベクトル $\mathbf{a}^T$ とベクトル **b** の積になります。

内積と転置ベクトル

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T\mathbf{b}$$

【出典・参考】
⇒ https://qiita.com/kenmatsu4/items/a144047c1b49aa8c7eb0
⇒ https://www.weblio.jp/content/ベクトルを合わせる

(Python での実装例)

```
#**************************************************
# ﾘｽﾄ04-(03)-7_三次元ベクトルの内積
#**************************************************
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D #3次元プロットのためのimport
import numpy as np

fig = plt.figure()
ax = Axes3D(fig)

# 座標の描画範囲
LXs, LYs, LZs = 0,0,0
LXe, LYe, LZe = 5,5,5
ax.set_xlim([LXs, LXe])
ax.set_ylim([LYs, LYe])
ax.set_zlim([LZs, LZe])
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

# ベクトル a, b と内積 a・b
O = np.array([0,0,0]) # 座標原点
a = np.array([1,2,3])
b = np.array([3,2,1])

print('type(a)=', type(a), '', a='', a, '', |a|='', np.linalg.norm(a))
print('type(b)=', type(b), '', b='', b, '', |b|='', np.linalg.norm(b))
print('a・b= ', a.dot(b))

# 各ベクトルを位置ベクトルとして描画
ax.scatter3D(O[0],O[1],O[2],"o", color='blue') # 座標原点
ax.quiver(O[0],O[1],O[2], a[0],a[1],a[2], color='red',length=1, normalize=False)
ax.quiver(O[0],O[1],O[2], b[0],b[1],b[2], color='red',length=1, normalize=False)

plt.grid()
plt.draw()
plt.show()
```

(3.7) 単位ベクトル・基本ベクトル

(1) 単位ベクトル

・大きさが１のベクトルを「単位ベクトル（たんいベクトル, unit vector)」と言います。
ベクトル $\mathbf{a}$ の大きさは、$|\mathbf{a}|$ なので、ベクトル $\mathbf{a}$ と同方向の単位ベクトル $\mathbf{e}_a$ は
次式により求められます：

単位ベクトル

ベクトル $\mathbf{a}$ と同方向の単位ベクトル　$\mathbf{e}_a = \mathbf{a} / |\mathbf{a}|$

(2) 基本ベクトル

・ｎ次元ベクトルで、ｎ個ある各座標軸の正の方向に向う単位ベクトルを

$\mathbf{e}_i$ （ i＝1, 2, ・・・, n)

で表わし，これらをｎ次元空間の「基本ベクトル（fundamental vector)」と呼びます。

・ｎ次元空間の点の位置ベクトル $\mathbf{r}$ は基本ベクトル $\mathbf{e}_i$ （ i＝1, 2, ・・・, n)の一次結合で表されます。
（「一次結合」は、ベクトルの定数倍の和のことです）

ｎ次元空間の単位ベクトルと位置ベクトル

・ｎ次元空間の単位ベクトル

$\mathbf{e}_1$ ＝（ 1, 0, 0, ・・・, 0 ）
$\mathbf{e}_2$ ＝（ 0, 1, 0, ・・・, 0 ）
：　　：
$\mathbf{e}_n$ ＝（ 0, 0, 0, ・・・, 1 ）

・ｎ次元空間の点の位置ベクトル　$\mathbf{r} = (r_1, r_2, \cdots, r_n)$

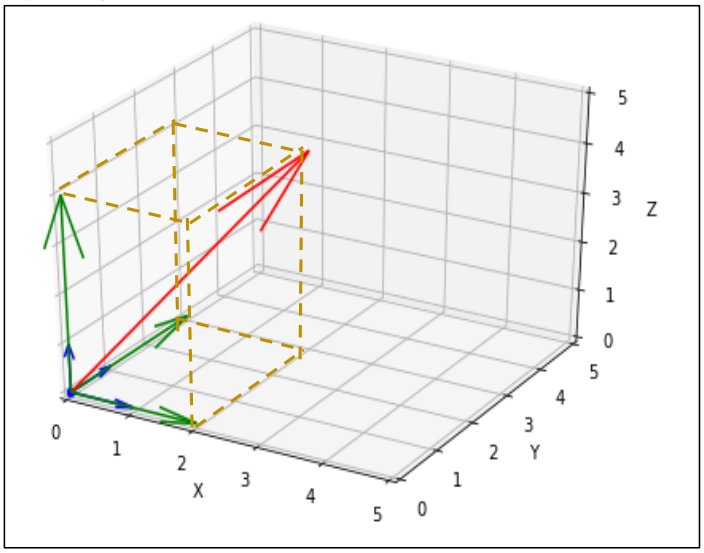
$\mathbf{r} = (r_1, r_2, \cdots, r_n)$
$= r_1\mathbf{e}_1 + r_2\mathbf{e}_2 + \cdots + r_n\mathbf{e}_n$
$= \sum_{k=1}^{n} r_k \mathbf{e}_k$

【出典・参考】
単位ベクトル ⇒ https://ja.wikipedia.org/wiki/単位ベクトル
基本ベクトル ⇒ https://kotobank.jp/word/基本ベクトル-51587
一次結合、線型結合 ⇒ https://ja.wikipedia.org/wiki/線型結合

（三次元ベクトルと基本ベクトルの実装例）

```
#*************************************************
# ﾘｽﾄ04-(03)-8_三次元ベクトルと基本ベクトル
#*************************************************
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D #3次元プロットのためのimport
import numpy as np

fig = plt.figure()
ax = Axes3D(fig)

# 座標の描画範囲
LXs, LYs, LZs = 0,0,0
LXe, LYe, LZe = 5,5,5
ax.set_xlim([LXs, LXe])
ax.set_ylim([LYs, LYe])
ax.set_zlim([LZs, LZe])
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

# 原点(0,0,0) から座標(2,3,4) へ向かうベクトル
# それを基本ベクトルの一次結合として表現する
ex = np.array([1,0,0])
ey = np.array([0,1,0])
ez = np.array([0,0,1])
ve = 2*ex + 3*ey + 4*ez
vs = np.array([0,0,0])

# 位置ベクトルとして描画
ax.scatter3D(vs[0],vs[1],vs[2], "o", color='blue') # 座標原点
ax.quiver(vs[0],vs[1],vs[2], ve[0],ve[1],ve[2], color='red',length=1, normalize=False)
ax.quiver(vs[0],vs[1],vs[2], ex[0],ex[1],ex[2], color='blue',length=1, normalize=False)
ax.quiver(vs[0],vs[1],vs[2], ey[0],ey[1],ey[2], color='blue',length=1, normalize=False)
ax.quiver(vs[0],vs[1],vs[2], ez[0],ez[1],ez[2], color='blue',length=1, normalize=False)

ax.quiver(vs[0],vs[1],vs[2], 2*ex[0],ex[1],ex[2], color='green',length=1, normalize=False)
ax.quiver(vs[0],vs[1],vs[2], ey[0],3*ey[1],ey[2], color='green',length=1, normalize=False)
ax.quiver(vs[0],vs[1],vs[2], ez[0],ez[1],4*ez[2], color='green',length=1, normalize=False)
plt.grid()
plt.draw()
plt.show()
```

（実行結果）



※ 左図で点線は補助線として
手動で追加

(4) 行列

(4.1) 行列とその表現

・ｍ×ｎ個の数字を、縦にm個、横にｎ個、長方形に並べたものを、
「(m、ｎ)型行列（matrix)」または、「m行ｎ列の行列」と言います。
本資料では、「行列」をアルファベットの大文字斜体（*A*～*Z*）で表現します。

・行列を構成するｍ×ｎ個の各数字を「要素（element)」または「成分（component)」といい、
横の並びを「行（row)」、縦の並びを「列（column)」と呼びます。

行列Ａ

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

m行（↓ 行）

← 1行目
← 2行目
：
← m行目

n 列（→ 列）

↑1列目　…　↑2列目　↑n列目

・行列*A*の（i行、j列）にある要素を、行列*A*の「（i,j）成分」と言い、以下の様に表現します。

行列Ａの（i,j）成分

a_{ij}　　または　　$(A)_{ij}$

・要素全体を大きな丸括弧 ″( )″ または、角括弧 ″[ ]″ で囲んで、行列を表現します。

・行列*A*は（i,j）成分 a_{ij} だけを明示して、以下のように略記することもあります。

行列Ａの表現

$A = (a_{ij})$　　または　　$(a_{ij})_{i=1,2\cdots,m,\ j=1,2,\cdots,n}$

または　　$(a_{ij})_{i=1\sim m,\ j=1\sim n}$

または　　$(a_{ij})_{1\leqq i\leqq m,\ 1\leqq j\leqq n}$

・行列のうち、特に「(m,1)型行列」は、「m次元の列ベクトル」と見做すことも出来ます。
行列のうち、特に「(1,n)型行列」は、「n次元の横ベクトル」と見做すことも出来ます。
行列のうち、特に「(1,1)型行列」は、「スカラー」と見做すことも出来ます。

（例）

2行3列の行列の例

$$A = \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{pmatrix}$$

3行2列の行列の例

$$B = \begin{pmatrix} 11 & 12 \\ 21 & 22 \\ 31 & 32 \end{pmatrix}$$

（Python での実装例）

```
#**************************************************
# ﾘｽﾄ04-(04)-1_行列
#**************************************************
import numpy as np

print("2行3列の行列 A --------------------")
A = np.array([[11,12,13], [21,22,23]])
print( 'type(A)=', type(A) )
print(A)

print("3行2列の行列 B --------------------")
B = np.array([[11,12], [21,22], [31,32]])
print( 'type(B)=', type(B) )
print(B)
```

（実行結果）

```
2行3列の行列 A --------------------
type(A)= <class 'numpy.ndarray'>
[[11 12 13]
 [21 22 23]]
3行2列の行列 B --------------------
type(B)= <class 'numpy.ndarray'>
[[11 12]
 [21 22]
 [31 32]]
```

(4.2) 行列の和・差・スカラー倍

・「行列」について、和・差・スカラー倍を次のように定義します：

(1) 和 $A+B$、差 $A-B$

・行列の和・差は、同一の型（m行ｎ列）の行列間で成立し、各成分同士の和・差になります：

$$A+B=(a_{ij})+(b_{ij})=(a_{ij}+b_{ij})$$

$$=\begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{pmatrix}$$

$$A-B=(a_{ij})-(b_{ij})=(a_{ij}-b_{ij})$$

$$=\begin{pmatrix} a_{11}-b_{11} & a_{12}-b_{12} & \cdots & a_{1n}-b_{1n} \\ a_{21}-b_{21} & a_{22}-b_{22} & \cdots & a_{2n}-b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}-b_{m1} & a_{m2}-b_{m2} & \cdots & a_{mn}-b_{mn} \end{pmatrix}$$

(2) スカラー倍 λA

・行列のスカラー倍は、各成分のスカラー倍になります：

$$\lambda A=\lambda(a_{ij})=(\lambda a_{ij})$$

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

(3) 零行列 O

・零行列 O はスカラー倍で $\lambda=0$ と置いたものに等しい：

$$O=(o_{ij})$$

$$=\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

(4) 行列の和・差・スカラー倍について、以下の法則が成り立ちます：

（A、B、C は同一の型（m行ｎ列）の行列、λ、μはスカラーとします。）

行列の和・差・スカラー倍についての法則

・和の交換法則	$A+B$	$=$	$B+A$
・和の結合法則	$(A+B)+C$	$=$	$A+(B+C)$
・零行列との和	$A+O=O+A$	$=$	A
・加えて零行列になる	$A-A$	$=$	O
・スカラー倍の分配法則	$\lambda(A+B)$	$=$	$\lambda A+\lambda B$
・スカラー倍の分配法則	$(\lambda+\mu)A$	$=$	$\lambda A+\mu A$
・スカラー倍の分配法則	$(\lambda\mu)A$	$=$	$\lambda(\mu A)$

（行列の和・差・スカラー倍の実装例）

```
#**************************************************
# ﾘｽﾄ04-(04)-2_行列の和・差・スカラー倍
#**************************************************
import numpy as np

print("2×3型行列 A, B --------------")
A = np.array([[11,12,13], [21,22,23]])
B = np.array([[111,112,113], [121,122,123]])

print( 'type(A)=', type(A) )
print('Matrix A', A)
print('Matrix B', B)

print("行列の和・差 A+B, A-B --------------")
print('A + B =')
print(A + B)

print('A + (-A) =')
print( A + (-A) )

print("行列のスカラー倍 λA  --------------")
lmbda = 3
print('スカラー =', lmbda)
print('スカラー * A =')
print( lmbda * A )
```

（実行結果）

```
2×3型行列 A, B --------------
type(A)= <class 'numpy.ndarray'>
Matrix A [[11 12 13]
 [21 22 23]]
Matrix B [[111 112 113]
 [121 122 123]]
行列の和・差 A+B, A-B --------------
A + B =
[[122 124 126]
 [142 144 146]]
A + (-A) =
[[0 0 0]
 [0 0 0]]
行列のスカラー倍 λA  --------------
スカラー = 3
スカラー * A =
[[33 36 39]
 [63 66 69]]
```

(4.3) 行列の積

・「行列」について、積を次のように定義します：

(1) アダマール積 $A \bigcirc B$

・行列のアダマール積は、同一の型（m行ｎ列）の行列間で成立し、各成分同士の積になります：

$$A \bigcirc B = (a_{ij}) \bigcirc (b_{ij}) = (a_{ij} \times b_{ij})$$

$$= \begin{pmatrix} a_{11}*b_{11} & a_{12}*b_{12} & \cdots & a_{1n}*b_{1n} \\ a_{21}*b_{21} & a_{22}*b_{22} & \cdots & a_{2n}*b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}*b_{m1} & a_{m2}*b_{m2} & \cdots & a_{mn}*b_{mn} \end{pmatrix}$$

(行列のアダマール積の実装例)

```
#*************************************************
# ﾘｽﾄ04-(04)-3_行列のアダマール積
#*************************************************
import numpy as np

print("1×3型行列 A と B のｱﾀﾞﾏｰﾙ積")
A = np.array([1,2,3])
B = np.array([4,5,6])
print('Matrix A',A)
print('Matrix B',B)
print('A*B =', A*B)

print("¥n2×3型行列 A と B のｱﾀﾞﾏｰﾙ積")
A = np.array([[11,12,13],[21,22,23]])
B = np.array([[1,2,3],[4,5,6]])
print('Matrix A', A)
print('Matrix B', B)
print('A*B =', A*B)
```

(実行結果)

```
1×3型行列 A と B のｱﾀﾞﾏｰﾙ積
Matrix A [1 2 3]
Matrix B [4 5 6]
A*B = [ 4 10 18]

2×3型行列 A と B のｱﾀﾞﾏｰﾙ積
Matrix A [[11 12 13]
 [21 22 23]]
Matrix B [[1 2 3]
 [4 5 6]]
A*B = [[ 11  24  39]
 [ 84 110 138]]
```

【出典・参考】

⇒ https://ja.wikipedia.org/wiki/アダマール積

(2) 積 *AB*

・行列*A, B*の積は、*A*の列サイズと*B*の行サイズが一致する時にのみ成立し、
Aの i行目の各成分と、Bの j行目の各成分同士の積の和が、演算結果行列の(i, j)成分になります。
これは、行列Aの i行目の各成分からなる行ベクトル$\mathbf{a}_i$ と、
行列Bの j行目の各成分からなる列ベクトル$\mathbf{b}_j$ の内積に相当します。

・行列*A*が（m行k列）、*B*が（k行n列）の時、その積は（m行n列）の行列になります。

$$AB = (a_{ij})_{1\leqq i\leqq m,\ 1\leqq j\leqq k}\ (b_{ij})_{1\leqq i\leqq k,\ 1\leqq j\leqq n}$$

$$(AB)_{ij} = \sum_{h=1}^{k}(a_{ih} * b_{hj}) \qquad 1\leqq i\leqq m,\ 1\leqq j\leqq n$$

・以下は、行列の積 *AB* の(2,3)成分の計算例

$$AB = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1n} \\ b_{21} & b_{22} & b_{23} & \cdots & b_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ b_{k1} & b_{k2} & b_{k3} & \cdots & b_{mn} \end{pmatrix}$$

2行目→　　↑3列目

$$(AB)_{23} = a_{21}*b_{13} + a_{22}*b_{23} + \cdots + a_{2k}*b_{k3}$$

(行列の積の実装例)

```
#*************************************************
# ﾘｽﾄ04-(04)-4_行列の積
#*************************************************
import numpy as np

print("1×3型行列 A と 3×1型行列 B の積 --")
print("Pythonでは、行列積が可能なように")
print("行列の向きを自動調整します。 -------")
A = np.array([1,2,3])
B = np.array([4,5,6])
C = np.array([[4],[5],[6]])
print('Matrix A')
print(A)
print('Matrix B')
print(B)
print('Matrix C')
print(C)

print('AB =',A.dot(B))
print('AC =',A.dot(C))

print("¥n2×3型行列 A と 3×2型行列 B の積 --")
A = np.array([[11,12,13],[21,22,23]])
B = np.array([[1,2],[3,4],[5,6]])
print('Matrix A')
print(A)
print('Matrix B')
print(B)

print('type(AB)=',type(A.dot(B)))
print('AB =')
print( A.dot(B))
print(' (AB)11 =', A[0][0]*B[0][0] + A[0][1]*B[1][0] + A[0][2]*B[2][0])
print(' (AB)12 =', A[0][0]*B[0][1] + A[0][1]*B[1][1] + A[0][2]*B[2][1])
print(' (AB)21 =', A[1][0]*B[0][0] + A[1][1]*B[1][0] + A[1][2]*B[2][0])
print(' (AB)22 =', A[1][0]*B[0][1] + A[1][1]*B[1][1] + A[1][2]*B[2][1])
```

(実行結果)

```
1×3型行列 A と 3×1型行列 B の積 --
Pythonでは、行列積が可能なように
行列の向きを自動調整します。 -------
Matrix A
[1 2 3]
Matrix B
[4 5 6]
Matrix C
[[4]
 [5]
 [6]]
AB = 32
AC = [32]

2×3型行列 A と 3×2型行列 B の積 --
Matrix A
[[11 12 13]
 [21 22 23]]
Matrix B
[[1 2]
 [3 4]
 [5 6]]
type(AB)= <class 'numpy.ndarray'>
AB =
[[112 148]
 [202 268]]
(AB)11 = 112
(AB)12 = 148
(AB)21 = 202
(AB)22 = 268
```

(4.4) 正方行列・単位行列・逆行列

・行列のうち、特に行と列の要素数が等しいものを「正方行列（せいほうぎょうれつ, square matrix)」と言い、「ｎ行ｎ列の行列」を「ｎ次の正方行列」と言います。

・ｎ次の正方行列全体の集合をＭとし、その行列を $A, B, C,$ ・・・ とするとき、積について以下の法則が成り立ちます：

行列の積についての法則

・積もまたＭの要素　　$AB \in M$

・積の結合法則　　$(AB)C = A(BC)$

・単位行列 E の存在

全てのA に対して、次式を満たす行列E（∈Ｍ）がある。

この E を「単位行列（たんいぎょうれつ、identity matrix)」という。

$$AE = EA = A$$

・積が単位行列になる逆行列の存在

行列A に対して、行列式(det$A$) ≠ 0 の時、

これを「正則行列（せいそくぎょうれつ、regular matrix)」という。

各々の正則行列A に対して、次式を満たす行列 A^{-1}（∈Ｍ）があり、

これを「逆行列（ぎゃくぎょうれつ, inverse matrix)」という。

$$AA^{-1} = A^{-1}A = E$$

※ 行列の積の交換法則（$AB = BA$）は一般的には成立しません。

※ 行列式(det$A$）については、【出典・参考】を参照のこと。

単位行列 E（∈Ｍ）

$$E = (e_{ij}) = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

$$e_{ij} \begin{cases} =1 & \text{for } i=j \\ =0 & \text{for } i \neq j \end{cases}$$

逆行列 A^{-1}（∈Ｍ）

$$A^{-1} = (\alpha_{ij}) = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1n} \\ \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2n} \\ \vdots & \vdots & & \vdots \\ \alpha_{n1} & \alpha_{n2} & \cdots & \alpha_{nn} \end{pmatrix}$$

行列 $A = (a_{ij})$ について

a_{ij} の余因数を A_{ij} として、

$$\alpha_{ij} = A_{ji}/\det(A)$$

（行と列が入れ替わることに注意！）

※ 余因数については、【出典・参考】を参照のこと。

逆行列についての公式

・A, B を同じ次数の正則行列とするとき、AB, A^{-1} も正則行列であり、それぞれの逆行列は、次式で与えられる：

$$(AB)^{-1} = B^{-1}A^{-1}$$

$$(A^{-1})^{-1} = A$$

（例：２次の正方行列の逆行列）

２次の正方行列Aの逆行列

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad A^{-1} = 1/(ad - bc) \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \qquad \det(A) = (ad - bc)$$

（単位行列・行列式・逆行列の実装例）

```
#************************************************
# ﾘｽﾄ04-(04)-5_単位行列・行列式・逆行列
#************************************************
import numpy as np

print("3×3型単位行列 I")
print(np.identity(3))

print("¥n3×3型行列 A")
A = np.array([[11,2,3],[4,5,6],[7,8,9]])
print('Matrix A')
print(A)

print("¥n行列 A の行列式 det(A)" )
print("行列式の値が０に近い時は、逆行列なしです。" )
detA = np.linalg.det(A)
print('det(A)=', detA)

print("¥n3×3型行列 A の逆行列 invA" )
invA = np.linalg.inv(A)
print('Matrix invA')
print(invA)

print("¥n3×3型行列 A と逆行列 invA との積" )
print("積が単位行列に(近く)なっていますか？" )
print('Matrix A×invA')
print(A.dot(invA))
```

（実行結果）

```
3×3型単位行列 I
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

3×3型行列 A
Matrix A
[[11  2  3]
 [ 4  5  6]
 [ 7  8  9]]

行列 A の行列式 det(A)
行列式の値が０に近い時は、逆行列なしです。
det(A)= -30.000000000000043

3×3型行列 A の逆行列 invA
Matrix invA
[[ 0.1        -0.2         0.1       ]
 [-0.2        -2.6         1.8       ]
 [ 0.1         2.46666667 -1.56666667]]

3×3型行列 A と逆行列 invA との積
積が単位行列に(近く)なっていますか？
Matrix A×invA
[[ 1.00000000e+00 -4.44089210e-16  0.00000000e+00]
 [-2.77555756e-17  1.00000000e+00  0.00000000e+00]
 [ 1.38777878e-17 -1.33226763e-15  1.00000000e+00]]
```

【出典・参考】

余因数行列と逆行列⇒ https://oguemon.com/study/linear-algebra/cofactor-expansion/?type=beta&utm_expid=136223162-0.OG1bPpOfQ7udrbP3Ldo_MQ.1&utm_referrer=https%3A%2F%2Fwww.google.co.jp%2F

行列の積 ⇒ 「数学公式事典」 黒田孝朗・須田貞之著 文研出版 1978

正方行列 ⇒ https://ja.wikipedia.org/wiki/正方行列

行列式 ⇒ https://ja.wikipedia.org/wiki/行列式

(4.5) 転置行列・直交行列

・(m, n)型の行列 A に対して A の (i, j)要素 a_{ij} と (j, i)要素 a_{ji} を入れ替えた
(n, m)型の行列、つまり対角線で成分を折り返した行列を、
「転置行列（てんちぎょうれつ、transpose [of a matrix], transposed matrix)」と言い、

> A の転置行列
>
> A^T または ${}^tA$

と表現します。以降の説明では A^T で記述することとします。

・行列の転置行列を与える操作のことを「転置（てんち、transpose)」といい、
「A を転置する」などと言います。

> 転置行列
>
> (転置前の行列)
>
> $$A = (a_{ij}) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$
>
> m行
>
> n列
>
> (転置後の行列)
>
> $$A^T = (b_{ij}) = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}$$
>
> n行
>
> m列
>
> $(b_{ij}) = (a_{ji})_{i=1\sim n,\ j=1\sim m}$

・転置行列について以下の法則が成り立ちます：

> 転置行列についての法則
>
> ・転置行列 A^T の更に転置行列 $(A^T)^T$
>
> $$(A^T)^T = A$$
>
> ・行列の和の転置行列
>
> $$(A + B)^T = A^T + B^T$$
>
> ・行列の定数倍の転置行列
>
> $$(\lambda A)^T = \lambda A^T$$ λ は実数
>
> ・積の転置行列
>
> $$(AB)^T = B^T A^T$$

・行列 A の転置行列 A^T が、行列 A の逆行列 A^{-1} に等しい時、
行列 A を「直交行列（ちょっこうぎょうれつ, orthogonal matrix)」と言います。

> 直交行列 A
>
> $$A^T = A^{-1}$$
> $$AA^T = A^TA = E$$

（例）

次の２次の直交行列 A は、原点の周りの角度θの回転移動を表しています：

$$A = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \qquad \mathbf{x}' = A\,\mathbf{x} \qquad \begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

（転置行列の実装例）

```
#************************************************
# ﾘｽﾄ04-(04)-6_転置行列
#************************************************
import numpy as np

print("2×3型行列 A")
A = np.array([[11,12,13],[21,22,23]])
print('Matrix A')
print(A)

print("¥n行列 A の転置行列 AT" )
AT = A.T
print('transposed Matrix of A')
print(A.T)
```

（実行結果）

```
2×3型行列 A
Matrix A
[[11 12 13]
 [21 22 23]]

行列 A の転置行列 AT
transposed Matrix of A
[[11 21]
 [12 22]
 [13 23]]
```

(4.6) 連立一次方程式の解法

・行列の応用の一つとして、連立方程式の解法があります。

・次の２次元の連立一次方程式

$$\begin{cases} b_1 = a_{11} \times x + a_{12} \times y \\ b_2 = a_{21} \times x + a_{22} \times y \end{cases}$$

は、(2,1)型行列 $\boldsymbol{b}$, $\boldsymbol{x}$ と (2,2)型行列$\boldsymbol{A}$ と行列の積として、以下の様に表現できます：

２次元の連立一次方程式

$$\boldsymbol{b} = \boldsymbol{A}\,\boldsymbol{x} \qquad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \qquad \boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} x \\ y \end{pmatrix}$$

$\det(\boldsymbol{A}) \neq 0$ の時、$\boldsymbol{A}$ の逆行列 $\boldsymbol{A}^{-1}$ が存在して、連立一次方程式の解 $\boldsymbol{x}$ は次式で与えられます：

２次元の連立一次方程式の解

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\,\boldsymbol{b}$$

これは３次元以上の連立一次方程式についても同じです。

(4.7) Ｄ次元線形回帰モデルの解法

・行列の応用の一つとして、Ｄ次元線形回帰モデルの解法があります。

(4.7.1) 考え方

・目的変数 y に対し、これを説明するＤ個の説明変数

$$(x_1, x_2, \cdots, x_D)$$

があり、目的変数 y が、Ｄ個の説明変数の線形結合として表現できる場合、
この式を「Ｄ次元線形回帰モデル式」と言います。

Ｄ次元線形回帰モデル式

$$y = w_0 + w_1 \times x_1 + w_2 \times x_2 + \cdots + w_D \times x_D$$
$$= \sum_{d=0}^{D} w_d \times x_d \qquad (\text{但し、} x_0 = 1)$$

Ｄ次元線形回帰モデル式は、結合の係数 $\boldsymbol{w}$ を求めることにより決定出来ます。

$$\boldsymbol{w} = (w_0, w_1, w_2, \cdots, w_D)$$

この係数 $\boldsymbol{w}$ を求める為に、Ｎ個の観測値の組：

$$\begin{cases} \text{No.1}: (x_{1,1}, x_{1,2}, \cdots, x_{1,D} ; t_1) \\ \text{No.2}: (x_{2,1}, x_{2,2}, \cdots, x_{2,D} ; t_2) \\ \quad : \\ \text{No.N}: (x_{N,1}, x_{N,2}, \cdots, x_{N,D} ; t_N) \end{cases}$$

ここで
- $x_{n,i}$ (i=1～D) は、観測値の組 No.n でのi番目の説明変数 x_i の値
- t_n は、観測値の組 No.n での目的変数 y の値（t は教師ﾃﾞｰﾀ teacher の略語）

から、Ｄ次元線形回帰モデル式を適用した時の、
目的変数の観測値とモデル式の計算値との誤差が最小になるように、
結合の係数 $(w_0, w_1, w_2, \cdots, w_D)$ を決定します。

(4.7.2) モデルの行列表現

・「Ｄ次元線形回帰モデル式」は、(D+1,1)型行列 $\boldsymbol{w}$, $\boldsymbol{x}$ を用いて以下の様に表現できます：

Ｄ次元線形回帰モデル式の行列表現

$$y(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} \qquad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix} \qquad \cdots(\text{式4-①})$$

(4.7.3) モデルを観測値から決定する

・Ｎ個の観測値の組で、n番目の組の説明変数群を $\boldsymbol{x}_n$ で表します。

$$\boldsymbol{x}_n = \begin{pmatrix} 1 \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix}$$

n番目の組の説明変数群 $\boldsymbol{x}_n$ に対し「Ｄ次元線形回帰モデル式」を適用して得られる目的変数 y の値は、次式で与えられます：

$$y(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{x}_n$$

この値と、目的変数の観測値の値 t_n との差は次式のようになります。
これが、観測値の n番目の組についての、モデル式の計算値の誤差に相当します。

$$y(\boldsymbol{x}_n) - t_n = \boldsymbol{w}^T \boldsymbol{x}_n - t_n$$

これをＮ個の観測値全体で扱う為に、次の「平均二乗誤差（ﾍｲｷﾝﾆｼﾞｮｳｺﾞｻ, Mean squared error, MSE)」Ｊ(w) を導入します。これは、各観測値の組の目的変数の観測値とモデル式の計算値との誤差の二乗の平均値です。Ｊ が $\boldsymbol{w}$ の関数になっているのは、Ｊ の値が未定の $\boldsymbol{w}$ に依存するからです。

$$\begin{aligned} J(\boldsymbol{w}) &= (1/N) \times \sum_{n=1}^{N} (y(\boldsymbol{x}_n) - t_n)^{**2} \\ &= (1/N) \times \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n)^{**2} \end{aligned}$$

これの最小値を与える $\boldsymbol{w}$ を求めるにあたり、両辺を w_i (i=0～D) で偏微分したものが、0 となるという条件を用います。式を整理した結果は、次式のようになります：

$$\begin{aligned} \partial J(\boldsymbol{w})/\partial w_0 \Rightarrow \quad & 0 = \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n) && (x_{n,0} = 1) \\ \partial J(\boldsymbol{w})/\partial w_1 \Rightarrow \quad & 0 = \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n)\, x_{n,1} \\ \partial J(\boldsymbol{w})/\partial w_2 \Rightarrow \quad & 0 = \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n)\, x_{n,2} \\ \vdots \qquad\qquad & \vdots \qquad\qquad \vdots \\ \partial J(\boldsymbol{w})/\partial w_D \Rightarrow \quad & 0 = \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n)\, x_{n,D} \end{aligned}$$

これは更にベクトルを用いて、次のように表現できます：

$$\begin{aligned} (0, 0, 0, \cdots, 0) &= \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n - t_n)\, \boldsymbol{x}_n^T \\ &= \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{x}_n \boldsymbol{x}_n^T - t_n \boldsymbol{x}_n^T) && \leftarrow \text{行列の分配法則} \\ &= \boldsymbol{w}^T \sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^T - \sum_{n=1}^{N} t_n \boldsymbol{x}_n^T && \cdots(\text{式4-②}) \end{aligned}$$

ここで、Ｎ個の観測値の組：

$$\begin{cases} \text{No.1}: (x_{1,1}, x_{1,2}, \cdots, x_{1,D}; t_1) \\ \text{No.2}: (x_{2,1}, x_{2,2}, \cdots, x_{2,D}; t_2) \\ \quad \vdots \\ \text{No.N}: (x_{N,1}, x_{N,2}, \cdots, x_{N,D}; t_N) \end{cases}$$

を、以下の様な(N,D+1)型行列 $\boldsymbol{X}$, (N,1)型行列 $\boldsymbol{t}$ を用いて表現します：

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ 1 & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{pmatrix} \quad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

($\boldsymbol{X}$ の第一列が 1 なのは、$x_{n,0} = 1$ として扱うことによります)

$\boldsymbol{X}$ により、(式4-②)の右辺第一項のΣ部分は、以下の様に表現できます：

$$\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^T = \sum_{n=1}^{N} \begin{pmatrix} 1 \\ x_{n,1} \\ x_{n,2} \\ \vdots \\ x_{n,D} \end{pmatrix} \begin{pmatrix} 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{pmatrix}$$

$$= \sum_{n=1}^{N} \begin{pmatrix} 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,D} \\ x_{n,1} & x_{n,1}*x_{n,1} & x_{n,1}*x_{n,2} & \cdots & x_{n,1}*x_{n,D} \\ x_{n,2} & x_{n,2}*x_{n,1} & x_{n,2}*x_{n,2} & \cdots & x_{n,2}*x_{n,D} \\ \vdots & \vdots & \vdots & & \\ x_{n,D} & x_{n,D}*x_{n,1} & x_{n,D}*x_{n,2} & \cdots & x_{n,D}*x_{n,D} \end{pmatrix}$$

$$= \boldsymbol{X}^{T}\boldsymbol{X}$$

$$\therefore \quad \sum_{n=1}^{N} \boldsymbol{x}_n\boldsymbol{x}_n^{T} = \boldsymbol{X}^{T}\boldsymbol{X} \qquad \cdots(式4-③)$$

$\boldsymbol{X}$、$\boldsymbol{t}$ により、(式4-②)の右辺第二項は、以下の様に表現できます：

$$\sum_{n=1}^{N} t_n\boldsymbol{x}_n^{T} = \sum_{n=1}^{N} t_n \times \begin{pmatrix} 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,D} \end{pmatrix}$$

$$= \sum_{n=1}^{N} \begin{pmatrix} t_n & t_n*x_{n,1} & t_n*x_{n,2} & \cdots & t_n*x_{n,D} \end{pmatrix}$$

$$= \begin{pmatrix} \sum_{n=1}^{N} t_n & \sum_{n=1}^{N} t_n*x_{n,1} & \sum_{n=1}^{N} t_n*x_{n,2} & \sum_{n=1}^{N} t_n*x_{n,D} \end{pmatrix}$$

$$= \begin{pmatrix} t_1 & t_2 & \cdots & t_N \end{pmatrix} \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ 1 & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{pmatrix}$$

$$= \boldsymbol{t}^{T}\boldsymbol{X}$$

$$\therefore \quad \sum_{n=1}^{N} t_n\boldsymbol{x}_n^{T} = \boldsymbol{t}^{T}\boldsymbol{X} \qquad \cdots(式4-④)$$

上記関係式(式4-③)、(式4-④)により、(式4-②)は以下の様に整理されます：

$$(0, 0, 0, \cdots, 0) = \boldsymbol{w}^{T}\boldsymbol{X}^{T}\boldsymbol{X} - \boldsymbol{t}^{T}\boldsymbol{X} \qquad \cdots(式4-②')$$

更にこれに転置行列の関係式を用いることにより、次のように変形できます：

$$\begin{aligned} (0, 0, 0, \cdots, 0)^{T} &= (\boldsymbol{w}^{T}\boldsymbol{X}^{T}\boldsymbol{X} - \boldsymbol{t}^{T}\boldsymbol{X})^{T} \\ &= (\boldsymbol{w}^{T}\boldsymbol{X}^{T}\boldsymbol{X})^{T} - (\boldsymbol{t}^{T}\boldsymbol{X})^{T} \\ &= (\boldsymbol{X}^{T}\boldsymbol{X})^{T}(\boldsymbol{w}^{T})^{T} - \boldsymbol{X}^{T}\boldsymbol{t} \\ &= (\boldsymbol{X}^{T}(\boldsymbol{X}^{T})^{T})\boldsymbol{w} - \boldsymbol{X}^{T}\boldsymbol{t} \\ &= (\boldsymbol{X}^{T}\boldsymbol{X})\boldsymbol{w} - \boldsymbol{X}^{T}\boldsymbol{t} \end{aligned}$$

$$\therefore \quad (\boldsymbol{X}^{T}\boldsymbol{X})\boldsymbol{w} = \boldsymbol{X}^{T}\boldsymbol{t} \qquad \cdots(式4-⑤)$$

(式4-⑤)の両辺に左側から $(\boldsymbol{X}^{T}\boldsymbol{X})$ の逆行列 $(\boldsymbol{X}^{T}\boldsymbol{X})^{-1}$ を掛けることにより、
(式4-①)の $\boldsymbol{w}$ の解析解を得ることが出来ます：

D次元線形回帰モデルの解析解

$$\boldsymbol{w} = (\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{t} \qquad \cdots(式4-⑥)$$

これが、「D次元線形回帰モデル式」の解となります。
この式の右辺の $(\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}\boldsymbol{t}$
には、「ムーア・ペンローズの擬似逆行列 (Moore-Penrose pseudo-inverse matrix)」という名前
がついており、これは線形代数学における逆行列の一般化になっています。

ムーア・ペンローズの擬似逆行列

$$(\boldsymbol{X}^{T}\boldsymbol{X})^{-1}\boldsymbol{X}^{T}$$

※　上記の手法は、本セミナー第９回目「機械学習（３回目：機械学習の概要と教師あり学習（回帰））」
で、線形回帰モデルの解析解を求める手法として、気温の経年変化を解析する実装例を示します。

【出典・参考】

ムーア・ペンローズ逆行列 ⇒ http://zellij.hatenablog.com/entry/20120811/p1

「Pythonで動かして学ぶ！ あたらしい機械学習の教科書」伊藤真著（翔泳社 2018年1月）

(5) 確認問題

以下の空欄に最もあてはまる回答を選択肢から選び、その記号を回答欄に記入してください。
（回答提出用の Excel ファイルは別途配布します）

(1) 数列と級数と総和

・「数列（すうれつ、numerical sequence)」とは、数が列になったものを言います。

より一般的には、 関数 a(x) の定義域が正の整数（つまり x ∈ 正の整数の集合）で、
x = 1, 2, ・・・, n, ・・・ における関数値 a(x) を、x の順に並べたものを「数列」と言います。
a(1), a(2), ・・・, a(n), ・・・

・x = n における関数 a の値 a(n) を a_n で表現し、次のように記します。
a_1, a_2, ・・・, a_n, ・・・
または $\{ a_n \}$ または $\{ a_k \}_{k=1,2,\ldots,n,\ldots}$

数列の各々の数を「項（こう、term)」と言い、特に a_1 を (1.1)＿＿＿＿、
添字 n に対応する項 a_n を (1.2)＿＿＿＿ と言います。
数列の最後の項を数列の (1.3)＿＿＿＿ と言います。(1.3)＿＿＿＿ が存在するとは限りません。
・無限数列 $\{ a_n \}$ の項を、加算記号 + でつないだ式
$a_1 + a_2 + \cdots + a_n + \cdots$

を (1.4)＿＿＿＿ といい、
ギリシャ文字シグマの大文字 ″Σ″ で、以下の様に表現します。

$$\sum_{n=1}^{\infty} a_n$$

数列のはじめの n 項までの和を、(1.5)＿＿＿＿ と呼び、
Sn または総和記号Σ で、以下の様に表現します。

$$S_n = \sum_{k=1}^{n} a_k = a_1 + a_2 + \cdots + a_n$$

・「総乗（そうじょう、product)」とは、数列の全ての積のことです。
無限数列 $\{ a_n \}$ の項を、乗算記号 × でつないだ式
$a_1 \times a_2 \times \cdots \times a_n \times \cdots$

を (1.6)＿＿＿＿ または無限積といい、
ギリシャ文字パイの大文字 ″Π″ で、以下の様に表現します。

$$\prod_{n=1}^{\infty} a_n \quad \text{または} \quad \Pi a_n$$

（選択肢）
(a) 「初項（しょこう、first term)」
(b) 「一般項（いっぱんこう、general term)」
(c) 「末項（まっこう、last term)」
(d) 「無限級数（むげんきゅうすう、infinite series)」
(e) 「無限乗積（むげんじょうせき、infinite product)」
(f) 「第ｎ部分和」(部分和（ぶぶんわ、patial sum))
(g) 「第ｎ部分積」(部分積（ぶぶんせき、partial product))

（回答）

(1.1)	
(1.2)	
(1.3)	
(1.4)	
(1.5)	
(1.6)	

(2)　等差数列、等比数列の部分和、部分積

・以下の（初項3、公差4）の等差数列の第10部分和 S_{10} は (2.1) になる。
　3, 7, 11, ・・・, a_{n-1}, a_n, ・・・　　一般項 $a_n = 3 + (n-1)4$

・以下の（初項3、公比1）の等比数列の第10部分和 S_{10} は (2.2) になる。
　3, 3, 3, ・・・, a_{n-1}, a_n, ・・・　　一般項 $a_n = 3$

・以下の（初項3、公比2）の等比数列の第10部分和 S_{10} は (2.3) になる。
　3, 6, 12, ・・・, a_{n-1}, a_n, ・・・　　一般項 $a_n = 3 \times 2^{(n-1)}$

・以下の（初項1、公差2）の等差数列の第10部分積 P_{10} は (2.4) になる。
　1, 3, 5, ・・・, a_{n-1}, a_n, ・・・　　一般項 $a_n = 1 + (n-1)2$

（選択肢）

(a)　420
(b)　210
(c)　30
(d)　60
(e)　3069
(f)　34459425
(g)　654729075

（回答）

(2.1)	
(2.2)	
(2.3)	
(2.4)	

(3) スカラー・ベクトル・行列

・数字一個のことを「スカラー (scalar)」と言います。

・数字を複数個並べたものを「ベクトル (vector)」と言います。

・m×n 個の数字を、縦にm個、横にn個、長方形に並べたものを、
「(m,n)型行列 (matrix)」または、「m行n列の行列」と言います。

・ベクトルを構成する各数字を (3.1) または「成分 (component)」と呼びます。
・ベクトルを構成する (3.1) の個数を (3.2) と呼びます。

・(3.1) を縦に n 個並べたものを (3.3) と言い、
(3.1) を横に n 個並べたものを (3.4) と言います。

・１次元のベクトルは、(3.5) と見做すことも出来ます。

・行列のうち、特に「(n,1)型行列」は、(3.3) と見做すことも出来ます。
行列のうち、特に「(1,n)型行列」は、(3.4) と見做すことも出来ます。
行列のうち、特に「(1,1)型行列」は、(3.5) と見做すことも出来ます。

(選択肢)

(a) 「n次元縦ベクトル (column vector)」
(b) 「n次元横ベクトル (row vector)」
(c) 「要素 (element)」
(d) 「次元 (じげん、dimension)」
(e) 「スカラー (scalar)」

(回答)

(3.1)	
(3.2)	
(3.3)	
(3.4)	
(3.5)	

(4) ベクトルの大きさと内積

・ベクトル **a** の大きさは、|**a**| と表現します。
その値は、要素の二乗和の平方根です。
・これは、ベクトル **a** を位置ベクトルとみなした場合、
原点Oから終点までの (4.1)＿＿＿＿ に相当します。

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}$$

（ベクトルの大きさ）

$$|\mathbf{a}| = \sqrt{a_1{**}2 + a_2{**}2 + \cdots + a_n{**}2}$$

$$= \sqrt{\sum_{k=1}^{n} a_k^2}$$

・ベクトルの「内積（Inner Product)」は、同一次元の２つのベクトル同士の演算で、
対応する要素同士の積を取り、その総和を結果（スカラー値）とします。
ベクトル **a** とベクトル **b** の内積は、**a**・**b**（a ドット b）と表現します。

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

（内積）

$$\mathbf{a}\cdot\mathbf{b} = a_1*b_1 + a_2*b_2 + \cdots + a_n*b_n$$

$$= \sum_{k=1}^{n} a_k b_k$$

・内積**a**・**b** は数式としては、下記のように書くことができます。
これは、ベクトル **a** をベクトル **b** に射影したベクトルを **a'** としたとき、
a' と **b** の長さを掛け合わせた値になります。

$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta \quad (\theta \text{は} \mathbf{a} \text{と} \mathbf{b} \text{のなす角度})$$

$$= |\mathbf{a}'||\mathbf{b}| \quad (|\mathbf{a}'| = |\mathbf{a}|\cos\theta)$$

・上記の関係式を見てわかるように、内積 **a**・**b** は方向が近い程、大きな値を取ることがわかります。
このことから、内積はベクトル **a** とベクトル **b** の (4.2)＿＿＿＿ の指標として
使用されることもあります。

・**a** と **b** の内積は、**a** の転置ベクトル $\mathbf{a}^T$ とベクトル **b** の行列の積になります。

$$\mathbf{a}\cdot\mathbf{b} = \mathbf{a}^T\mathbf{b}$$

（選択肢）

(a) 方向
(b) 距離
(c) 類似度
(d) 大きさ

（回答）

(4.1)	
(4.2)	

(5) 正方行列・単位行列・正則行列・逆行列・転置行列・直行行列

・ｍ×ｎ個の数字を、縦にｍ個、横にｎ個、長方形に並べたものを、
「(m、ｎ)型行列 (matrix)」または、「ｍ行ｎ列の行列」と言います。

・行列のうち、特に行と列の要素数が等しいものを (5.1)＿＿＿＿ と言い、
「ｎ行ｎ列の行列」を「ｎ次の (5.1)＿＿＿＿ 」と言います。

・全てのｎ次の (5.1)＿＿＿＿ A に対して、次式を満たすｎ次の (5.1)＿＿＿＿ E を
(5.2)＿＿＿＿ と言います。

$$AE = EA = A$$

・E は、以下のように対角線上の要素のみ 1 で、他要素は全て 0 になります：

$$E = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

・(5.1)＿＿＿＿ A に対して、行列式($\det A$) $\neq 0$ の時、これを (5.3)＿＿＿＿ といいます。
各々の (5.3)＿＿＿＿ A に対して、次式を満たす行列 A^{-1} があり、
これを、A の (5.4)＿＿＿＿ といいます。

$$AA^{-1} = A^{-1}A = E$$

・(m, ｎ)型の行列 A に対して A の (i, j)要素 a_{ij} と (j, i)要素 a_{ji} を入れ替えた
(ｎ, m)型の行列、つまり対角線で成分を折り返した行列を、
A の (5.5)＿＿＿＿ といい、A^{T} または ${}^{t}A$ と表現します。

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

ｍ行

ｎ列

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

ｎ行

ｍ列

・行列 A の (5.5)＿＿＿＿ A^{T} が、行列 A の (5.4)＿＿＿＿ A^{-1} に等しい時、
行列 A を (5.6)＿＿＿＿ と言います。
(5.6)＿＿＿＿ の例として、回転移動などがあります。

$$A^{T} = A^{-1}$$
$$AA^{T} = A^{T}A = E$$

(選択肢)

(a) 「正方行列 (せいほうぎょうれつ, square matrix)」
(b) 「単位行列 (たんいぎょうれつ、identity matrix)」
(c) 「逆行列 (ぎゃくぎょうれつ, inverse matrix)」
(d) 「正則行列 (せいそくぎょうれつ、regular matrix)」
(e) 「転置行列 (てんちぎょうれつ、transposed matrix)」
(f) 「直交行列 (ちょっこうぎょうれつ, orthogonal matrix)」
(g) 「ユニタリ行列 (unitary matrix)」

(回答)

(5.1)	
(5.2)	
(5.3)	
(5.4)	
(5.5)	
(5.6)	

(6) 転置行列の例

・(2,3)型の次の行列 A の転置行列 A^T は (6.1) になります。

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$$

（選択肢）

(a) $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix}$

(b) $\begin{pmatrix} a_{21} & a_{22} & a_{23} \\ a_{11} & a_{12} & a_{13} \end{pmatrix}$

(c) $\begin{pmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{pmatrix}$

(d) $\begin{pmatrix} a_{21} & a_{11} \\ a_{22} & a_{12} \\ a_{23} & a_{13} \end{pmatrix}$

（回答）

(6.1)	

(7) 逆行列の例と連立一次方程式の解

・次の２次元の連立一次方程式

$$\begin{cases} b_1 = a_{11} \times x + a_{12} \times y \\ b_2 = a_{21} \times x + a_{22} \times y \end{cases}$$

は、(2,1)型行列 b, x と (2,2)型行列A と行列x の積として、以下の様に表現できます：

$$b = A\,x \qquad b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \qquad A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \qquad x = \begin{pmatrix} x \\ y \end{pmatrix}$$

$\det(A) \neq 0$ の時、A の逆行列 A^{-1} が存在して、連立一次方程式の解 x は次式で与えられます：

$$x = A^{-1} b$$

・具体的には解 x は (7.1) になります。

（選択肢）

(a) $\begin{pmatrix} a_{11}/\det A \times b_1 + & a_{12}/\det A \times b_2 \\ a_{21}/\det A \times b_1 + & a_{22}/\det A \times b_2 \end{pmatrix}$

(b) $\begin{pmatrix} a_{22}/\det A \times b_1 - & a_{12}/\det A \times b_2 \\ -a_{21}/\det A \times b_1 + & a_{11}/\det A \times b_2 \end{pmatrix}$

(c) $\begin{pmatrix} a_{22}/\det A \times b_1 + & a_{12}/\det A \times b_2 \\ a_{21}/\det A \times b_1 + & a_{11}/\det A \times b_2 \end{pmatrix}$ ($\det A = a_{11} \times a_{22} - a_{12} \times a_{21}$)

（回答）

(7.1)	

(8) Ｄ次元線形回帰モデルの解法

・行列の応用の一つとして、Ｄ次元線形回帰モデルの解法があります。

・目的変数 y に対し、これを説明するＤ個の説明変数

$$(x_1, x_2, \cdots, x_D)$$

があり、目的変数 y が、Ｄ個の説明変数の線形結合として表現できる場合、
この式を「Ｄ次元線形回帰モデル式」と言います。

$$y = w_0 + w_1 \times x_1 + w_2 \times x_2 + \cdots + w_D \times x_D$$
$$= \sum_{d=0}^{D} w_d \times x_d \qquad (\text{但し、} x_0 = 1)$$

・Ｄ次元線形回帰モデル式は、(D+1,1)型行列 $\boldsymbol{w}$, $\boldsymbol{x}$ を用いて以下の様に表現できます：

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} \qquad \boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} \qquad \boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_D \end{pmatrix}$$

・Ｄ次元線形回帰モデル式は、観測値の組から、最適な結合の係数 $\boldsymbol{w}$ を求めることにより決定出来ます。

・ここで、Ｎ個の観測値の組：

No.1 : $(x_{1,1}, x_{1,2}, \cdots, x_{1,D} ; t_1)$
No.2 : $(x_{2,1}, x_{2,2}, \cdots, x_{2,D} ; t_2)$
:
No.N : $(x_{N,1}, x_{N,2}, \cdots, x_{N,D} ; t_N)$

(t_n は、第n番目の説明変数の組 $\boldsymbol{x}_n=(x_{n,1}, x_{n,2}, \cdots, x_{n,D})$ に対する目的変数 y の観測値の値)

を、以下の様な(N,D+1)型行列 $\boldsymbol{X}$, (N,1)型行列 $\boldsymbol{t}$ を用いて表現します：

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,D} \\ 1 & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,D} \end{pmatrix} \qquad \boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

(第一列が 1 なのは、$x_{n,0} = 1$ として扱うことによります)

これにより、Ｄ次元線形回帰モデルの解析解は次のように求められます：

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t}$$

この式の右辺の

$$(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$$

には、(8.1)＿＿＿＿＿＿ という名前がついており、
これは線形代数学における (8.2)＿＿＿＿＿＿ の一般化になっています。

(選択肢)

(a) 「逆行列（ぎゃくぎょうれつ, inverse matrix)」
(b) 「正則行列（せいそくぎょうれつ、regular matrix)」
(c) 「転置行列（てんちぎょうれつ、transposed matrix)」
(d) 「共役転置行列（きょうえきてんちぎょうれつ, cnjugate transposed matrix)」
(e) 「特殊直交行列（とくしゅちょっこうぎょうれつ, special orthogonal matrix)」
(f) 「ムーア・ペンローズの擬似逆行列 (Moore-Penrose pseudo-inverse matrix)」
(g) 「ウェーブレット行列 (wavelet matrix)」

(回答)

(8.1)	
(8.2)	

「第４回　数学の基礎（1回目）」の理解度試験

提出者　：

提出日　：　　年　　月　　日

回答

No.	回答
(1.1)	
(1.2)	
(1.3)	
(1.4)	
(1.5)	
(1.6)	
(2.1)	
(2.2)	
(2.3)	
(2.4)	
(3.1)	
(3.2)	
(3.3)	
(3.4)	
(3.5)	
(4.1)	
(4.2)	

No.	回答
(5.1)	
(5.2)	
(5.3)	
(5.4)	
(5.5)	
(5.6)	
(6.1)	
(7.1)	
(8.1)	
(8.2)	

採点結果

／ 27

（※黄色の枠　のみ記入をお願いします）

※ ご意見・ご要望など有りましたら、下欄に記してください。