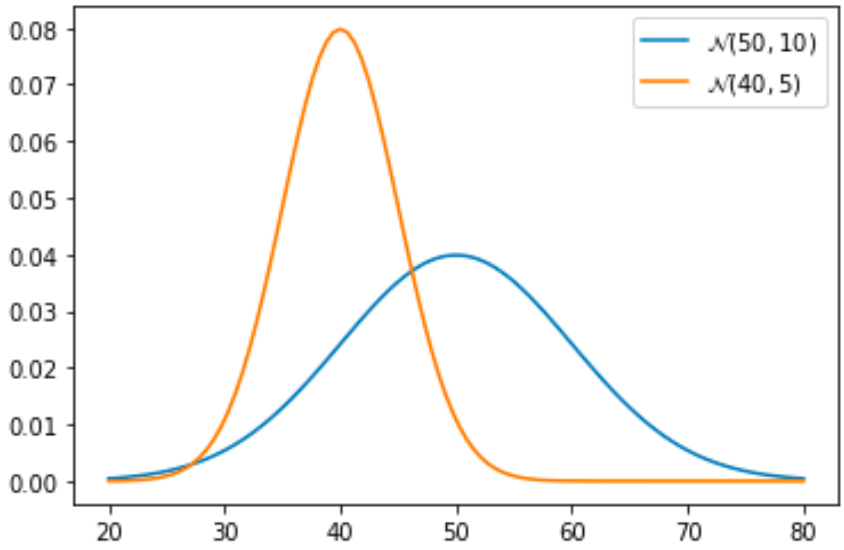
일변량 정규분포

$$X \sim N(\mu,\sigma),\ f(x\,|\,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{(x-\mu)^2)}{2\sigma^2})$$

scipy.stats.norm.pdf(x,평균=loc, scale 표준편차)

```
import numpy as np
import scipy.stats as st
mu=50; std=10
x=np.linspace(mu-3*std,mu+3*std,num=100)
fx1=st.norm.pdf(x,50,10)
fx2=st.norm.pdf(x,40,5)
```

```
import matplotlib.pyplot as plt
plt.plot(x,fx1,label="$\mathcal{N}(50,10)$")
plt.plot(x,fx2,label="$\mathcal{N}(40,5)$")
plt.legend(loc=1)
plt.show()
```



평균 50, 표준편차 10을 따르는 정규분포 100개 난수 생성하여 x 오브젝트에 저장

```
x=st.norm.rvs(50,10,100)
```

행렬 matrix 주요 개념

행렬 matrix

$$A_{n\times p}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{np} \end{bmatrix} \Rightarrow A_{3\times 3}=\begin{bmatrix} 1 & 2 & -1 \\ 0 & 1 & 2 \\ 1 & 0 & 2 \end{bmatrix}, B_{3\times 2}=\begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 1 & 2 \end{bmatrix}$$

- 간편식 $\{A_{ij}\}$ - 첨자 i는 행, j는 열을 나타냄
- 행 차수 n, 열 차수 p인 행렬, 차수 (nXp)인 행렬 - 기호 대문자 A, B, ...

벡터 vector $\underline{x}$

- 열의 차수가 1인 행렬을 열 벡터(column vector) : 벡터의 기본 $\underline{x}=\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$
- 행의 차수가 1인 행렬을 행 벡터(row vector) $\underline{y}^T=(3 \quad 2)$

스칼라 scalr

- 행과 열의 차수 모두가 1인 경우 행렬은 scalar (스칼라)이다.

특수한 행렬

정방 sqaure 행렬

- 행과 열의 차수가 동일한 행렬 - $A_{n\times n}$

대각 diagonal 행렬

- 정방행렬 중 대각원소를 제외한 모든 원소가 0인 경우 - $D_3=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -2 \end{bmatrix}$
- 대각행렬의 대각원소의 합을 trace라 한다. $tr(D)=\sum_{i=1}^{n} d_{ii}=-1, tr(A)=4$

항등 identiry 행렬

- 대각행렬 중 대각원소를 모두 1인 경우 : $I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- 항등행렬의 trace는 차수 n이고 숫자의 1과 같은 역할을 한다.

원행렬, 원벡터

- 모든 원소가 1인 행렬을 원 행렬(J로 표현), 1인 벡터($\underline{1}$으로 표현)를 원벡터라 한다.

- 만약 $\underline{1}_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$이면 $\underline{1}^T\underline{1} = 1, \underline{1}\underline{1}^T = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = J_3$이 된다.

전치 transpose

- 행렬 $A_{n\times p}$ 에서 행과 열을 바꾸는 것을 전치라 하고 A'(혹은 A^T)로 나타내며 차수는 $p \times n$ 이다.

- $A'_{3\times 3} = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 0 \\ -1 & 2 & 2 \end{bmatrix}', \underline{x}' = [1 \quad 2 \quad 1]$

대칭 symetric 행렬

- 원 행렬과 대칭 행렬이 동일한 행렬, 즉 $A = A^T$를 만족하는 행렬 A를 대칭행렬이라 한다.
- 대칭행렬이려면 반드시 정방행렬이여야 하며 항등행렬은 모두 대칭 행렬이다.

사칙연산 Algebra

행렬의 합(차) addition / subtraction

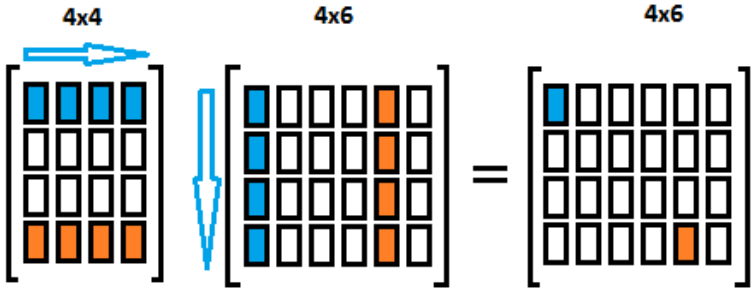
- 대응하는 원소의 합/차로 계산된다.
- 합(차) 계산이 가능하려면 두 행렬의 차수는 동일해야 한다. conformable

$$\mathbf{A}+\mathbf{B} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

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

행렬의 곱 multiplication

- 행렬에서 곱의 conformable(적합) 조건 : 앞 행렬(벡터)의 열의 차수와 뒤 행렬 행의 차수가 동일
- 곱의 결과는 (앞 행렬의 행 차수)x(뒤 행렬의 열의 차수)인 행렬(벡터, 스칼라)이다.
- 행렬A[i번째 행]x행렬B[j번째 열] 대응 곱 =(AB)행렬[(i, j) 원소]



(성질)

- $AB \neq BA$ (일반적으로 성립하지 않는다)
- 행렬 곱의 대칭 : $(AB)' = B'A'$
- $A'A$은 대칭행렬이다.

(행 벡터)x(열 벡터)

벡터의 곱은 앞 벡터의 열의 원소와 대응하는 뒤 벡터의 행의 원소의 곱을 더한 값을 적으면 된다. 곱이 가능하기 위해서는 앞 행의 차수와 열의 차수는 같아야 하며 행 벡터(1xp행렬)와 열 벡터(px1행렬) 곱은 scalar(1x1행렬)이다.

(열 벡터)x(행 벡터)

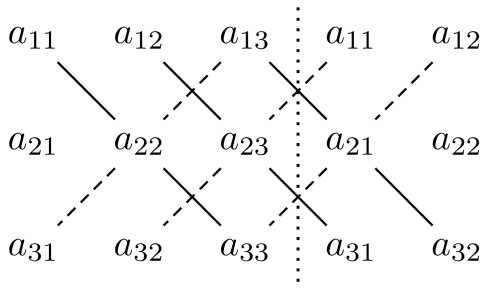
열 벡터 열 원소와 행 벡터의 행 원소의 곱을 (앞의 열 벡터 원소 위치) 행, (뒤의 행 벡터 원수 위치) 열로 하여 행렬을 만든다. 앞의 열 벡터와 차수와 뒤 열 벡터 차수는 같을 필요가 없다. 결과는 (nxp) 행렬이다.

행렬식 Determinant

행렬식은 정방행렬에서만 계산되며 결과는 스칼라 (기호) $\left|A\right|$ 혹은 $\det(A)$

$$\det\begin{bmatrix} a & b \\ c & d \end{bmatrix} = a \bullet d - c \bullet b$$

$$\det\begin{bmatrix} 8 & 3 \\ 4 & 2 \end{bmatrix} = 8 \bullet 2 - 4 \bullet 3 = 16 - 12 = 4 \quad (2 \times 2 \text{ 행렬})$$



$(3 \times 3$ 행렬)

소인자 Minor

지정한 행, 열을 제외한 행렬

$$M_{23} = \det \begin{bmatrix} 1 & 4 & \square \\ \square & \square & \square \\ -1 & 9 & \square \end{bmatrix} = \det \begin{bmatrix} 1 & 4 \\ -1 & 9 \end{bmatrix} = (9 - (-4)) = 13$$

소인자에 의한 행렬식 계산

$$|A| = \sum_{i=1}^{n} a_{ij}(-1)^{i+j} |M_{ij}| = \sum_{j=1}^{n} a_{ij}(-1)^{i+j} |M_{ij}|$$

차수가 3일 경우: $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 7 \\ 8 & 9 & 10 \end{bmatrix}$ 의 행렬식은

$$|A| = 1*(-1)^{1+1} \begin{vmatrix} 5 & 7 \\ 9 & 10 \end{vmatrix} + 2*(-1)^{1+2} \begin{vmatrix} 4 & 7 \\ 8 & 10 \end{vmatrix} + 3*(-1)^{1+3} \begin{vmatrix} 4 & 5 \\ 8 & 9 \end{vmatrix} = 7 \text{ (1번째 행 이용)}$$

$$|A| = 4*(-1)^{2+1} \begin{vmatrix} 2 & 3 \\ 9 & 10 \end{vmatrix} + 5*(-1)^{2+2} \begin{vmatrix} 1 & 3 \\ 8 & 10 \end{vmatrix} + 7*(-1)^{2+3} \begin{vmatrix} 1 & 2 \\ 8 & 9 \end{vmatrix} = 7 \text{ (2번째 행 이용)}$$

(행렬식 성질)

- $|A'| = |A|$, $|AB| = |BA|$, $|AB| = |A||B|$
- 행렬 A의 두 행이 같으면 행렬식은 0이다.
- 한 행(열)의 상수를 곱하여 다른 행에 더해도 행렬식 값은 변하지 않는다.
- 한 행(열)을 다른 행들의 선형 결합으로 표현할 수 있으면 행렬식 0이다.

역행렬

$AB = I$ 만족하는 행렬 B를 행렬 A의 역행렬이라 하고 A^{-1}로 표기한다.

$$A^{-1} = \frac{1}{|A|} adjA = \frac{1}{|A|}$$

(역행렬 성질)

- 역행렬은 unique하다.
- $|A^{-1}| = 1/|A|$, $(A^{-1})^{-1} = A$, $(A')^{-1} = (A^{-1})'$, $(AB)^{-1} = B^{-1}A^{-1}$

계수 rank

- (정의) 행렬에서 선형 독립인 행의 개수와 열의 개수 중 최소값, $rank(A_{m\times n}) \le min(n, m)$
- 정의(full rank) : (nxn)정방 행렬에서 선형 독립인 행(열)의 개수( )가 행렬의 차수 n와 같다면 이 행렬은 full-rank 행렬이라 한다. 즉 $rank(A_{n\times n}) = n$ 이면 full-rank이다.

정의(LIN: linearly independent vector)

$a_1\underline{x}_1 + a_2\underline{x}_2 + \ldots + a_p\underline{x}_p = \underline{0}$ 가 성립하려면 모든 $a_i = 0$ 일 때만 만족한다면 열벡터 ($\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_p$) 는 선형 독립(linearly independent) 벡터라 함

- 0이 아닌 a_i에 대해서 만족한다면 선형 종속(linearly dependent)인 벡터
- 상호 종속인 벡터는 하나의 벡터를 다른 벡터들의 선형 결합으로 표시할 수 있다는 것을 의미한다.

▪ 역행렬이 존재한다.	▪ 역행렬이 존재하지 않는다
▪ full-rank이다. rank(A)=n	▪ full-rank아니다. rank(A)<n
▪ A는 non-singular이다.	▪ A는 singular이다.
▪ |A|≠0	▪ |A|=0
▪ Ax=b 의 해가 존재한다.	▪ Ax=b 의 해가 존재하지 않는다.

$B_{3\times 2} = \begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 1 & 2 \end{bmatrix}$ => $rank(B) = 2$

$M_{3\times 3} = \begin{bmatrix} 1 & -1 & 0 \\ 2 & 0 & 2 \\ 1 & 2 & 3 \end{bmatrix}$ => $rank(M) = 2$ 왜? 첫째 열과 둘째 열의 합은 3째열이다. 그러므로 3번째 열은 종속되어 있음 => 역행렬 M^{-1}은 존재하지 않으며 $M\underline{x} = \underline{0}$을 만족하는 해는 무수히 많다.

일변량 확률변수 x 기대값 행렬 표현

확률변수 x의 n개 관측치 벡터를 $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}$ 이라 하자.

- $\sum x_i$ => (행렬 표현) $\sum x_i = (\underline{1}'\underline{x})$ / 평균 $\bar{x} = \sum x_i/n = (\underline{1}'\underline{x})/n$
- $\sum x_i^2$ => $\sum x_i^2 = \underline{x}'\underline{x}$

확률변수 벡터 $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_p \end{bmatrix}$

확률변수 x_i의 평균과 분산을 각각 μ_i, σ_{ii}^2라 하고 (x_i, x_j)의 공분산을 σ_{ij}^2 라 하면

(1) $E(\underline{x}) = \underline{\mu} = \begin{bmatrix} \mu_1 \\ \cdots \\ \mu_p \end{bmatrix}$ (2) $V(\underline{x}) = \Sigma = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \cdots & \sigma_{1p}^2 \\ \sigma_{11}^2 & \sigma_{22}^2 & \cdots & \sigma_{2p}^2 \\ \cdots & \cdots & \cdots & \cdots \\ \sigma_{p1}^2 & \sigma_{p2}^2 & \cdots & \sigma_{pp}^2 \end{bmatrix}$

(3) $E(\underline{a}'\underline{x}) = \underline{a}'\underline{\mu}$: 결과는 스칼라 (4) $V(\underline{a}'\underline{x}) = \underline{a}\Sigma\underline{a}'$

행렬의 미분

상수벡터 $\underline{a}' = \begin{bmatrix} a_1 & a_2 & \cdots & a_p \end{bmatrix}$, 정방행렬 A 에 대하여

(1) $\frac{\partial}{\partial \underline{x}}(\underline{a}'\underline{x}) = \underline{a}$ (2) $\frac{\partial}{\partial \underline{x}}(\underline{x}'\underline{a}) = \underline{a}$

(3) $\frac{\partial}{\partial \underline{x}}(\underline{x}'A\underline{x}) = A\underline{x} + A'\underline{x}$ => (A 대칭행렬) $\frac{\partial}{\partial \underline{x}}(\underline{x}'A\underline{x}) = 2A\underline{x}$

행렬연산 in Python

행렬 입력

$$A_{3\times3}=\begin{pmatrix}1 & 4 & 5\\ -5 & 8 & 9\\ -6 & 7 & 11\end{pmatrix}, B_{2\times3}=\begin{pmatrix}1 & 2 & 3\\ 4 & 7 & 9\end{pmatrix}$$

```
import numpy as np
A=np.array([[1, 4, 5],
  [-5, 8, 9],
  [-6, 7, 11]])
B=np.array([[1, 2, 3],
  [4,7,9]])
```

행렬 인덱스

행렬[행번호, 열번호] -> 모든 위치번호는 데이터프레임과 동일하게 0부터 시작된다.

: 은 일련번호 적용 시 사용된다.

```
A[:,1], B[0,1]
```

```
(array([4, 8, 7]), 4)
```

```
C=A[:,0:2]
C
```

```
array([[ 1,  4],
       [-5,  8],
       [-6,  7]])
```

행 차수

shape : 행과 열 차수, ndim : array 차원개수 size : 행렬 원소 개수

```
A.shape,A.ndim,A.size
```

```
((3, 3), 2, 9)
```

전치 transpose $A_{n\times p}=A^{T}_{p\times n}$

```
print(A.T,'\n ******* \n',B.T)
```

```
[[ 1 -5 -6]
 [ 4  8  7]
 [ 5  9 11]]
 *******
 [[1 4]
 [2 7]
 [3 9]]
```

행렬 쌓기

* 가로 쌓기 : 열의 차수가 같으면 conformable (연산 가능)

* 세로 쌓기 : 행의 차수가 같으면 연산가능

```
print(np.hstack([A,B.T]),'\n *******
\n',np.vstack([A,B]))
```

```
[[ 1  4  5  1  4]
 [-5  8  9  2  7]
 [-6  7 11  3  9]]
 *******
 [[ 1  4  5]
 [-5  8  9]
 [-6  7 11]
 [ 1  2  3]
 [ 4  7  9]]
```

더하기 빼기 : 행과 열의 차수가 동일해야 함

```
A-2*A
```

```
array([[ -1,  -4,  -5],
       [  5,  -8,  -9],
       [  6,  -7, -11]])
```

곱하기 : 앞 행렬의 열의 차수와 뒤 행렬 행 차수가 동일해야 함

```
A@B.T

A.dot(B.T)
```

```
array([[ 24,  77],
       [ 38, 117],
       [ 41, 124]])
```

행렬식 : 정방행렬에만 계산 가능

그러므로 정방행렬이 아닌 B 행렬의 행렬식은 존재하지 않는다.

```
import numpy.linalg as la
la.det(A)
```

```
94.0
```

행렬랭크 : 행 혹은 열의 독립인 차수 (열과 행의 랭크 중 낮은 것)

```
la.matrix_rank(A)
```

```
3
```

행렬식의 0이면 행 혹은 열 중 적어도 하나가 다른 열들과 독립이 아니라는 의미이므로 Full rank가 아니다. 행렬 A의 행렬식이 0이 아니므로 full-rank 3이 된다.

그러나 다음의 경우은 1열과 3열이 동일하므로 랭크는 2가 되고 행렬식은 0이다.

```
E=np.array([[1,1,1],[2,1,2],[4,3,4]])
la.matrix_rank(E),la.det(E)
```

```
(2, 0.0)
```

역행렬 : 정방행렬이고 행렬식이 0(full-rank)이 아닌 경우 존재

```
la.inv(A)
```

```
array([[ 0.26595745, -0.09574468, -0.04255319],
       [ 0.0106383 ,  0.43617021, -0.36170213],
       [ 0.13829787, -0.32978723,  0.29787234]])
```

```
la.inv(A)@A
```

```
array([[ 1.00000000e+00,  8.46545056e-16,  1.23512311e-15],
       [-2.22044605e-16,  1.00000000e+00, -1.11022302e-16],
       [ 0.00000000e+00,  1.11022302e-16,  1.00000000e+00]])
```

연립방정식 해 구하기 $A\underline{x} = \underline{b}$, 해 $\hat{\underline{x}} = A^{-1}\underline{b}$

```
A=np.array([[1,1,1],[2,1,2],[4,3,-4]])
b=np.array([1,4,7])
la.inv(A).dot(b)  #solution
```

```
array([ 3.125, -2.   , -0.125])
```

```
A@la.inv(A)@b
```

```
array([1., 4., 7.])
```

특수한 행렬 만들기

```
np.zeros((2,3)) #영행렬
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

```
np.ones((2,3)) #일행렬
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
np.eye(3) #항등행렬
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
np.full((2,3),3) #동일원소 행렬
```

```
array([[3, 3, 3],
       [3, 3, 3]])
```

```
np.random.random((2,3) #난수생성
```

```
array([[0.859799  , 0.37413985, 0.28712814],
       [0.02358198, 0.31388947, 0.17501232]])
```

평균벡터, 공분산행렬, 상관계수 행렬

행의 차수 n, 열(변수)의 개수 p인 데이터 행렬 $X_{n\times p} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \cdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$

확률변수 열벡터 : $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_p \end{bmatrix}$

Iris 데이터 불러오기 pandas

```
import pandas as pd
iris=pd.read_csv('https://vincentarelbundock.github.io/Rdatasets/csv/datasets/iris.csv')
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Unnamed: 0     150 non-null    int64  
 1   Sepal.Length   150 non-null    float64
 2   Sepal.Width    150 non-null    float64
 3   Petal.Length   150 non-null    float64
 4   Petal.Width    150 non-null    float64
 5   Species        150 non-null    object 
```

```
df=iris.iloc[:,1:5]
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

Pandas 평균 공분산 상관계수 행렬

```
dfdf.mean()
```

```
Sepal.Length    5.843333
Sepal.Width     3.057333
Petal.Length    3.758000
Petal.Width     1.199333
```

```
df.cov()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	0.685694	-0.042434	1.274315	0.516271
Sepal.Width	-0.042434	0.189979	-0.329656	-0.121639
Petal.Length	1.274315	-0.329656	3.116278	1.295609
Petal.Width	0.516271	-0.121639	1.295609	0.581006

```
df.corr()
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.000000	-0.117570	0.871754	0.817941
Sepal.Width	-0.117570	1.000000	-0.428440	-0.366126
Petal.Length	0.871754	-0.428440	1.000000	0.962865
Petal.Width	0.817941	-0.366126	0.962865	1.000000

평균벡터 $E(\underline{x})_{1\times p}$

one 벡터 : $\underline{1}_n = \begin{bmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{bmatrix}$ 평균벡터 : $E(\underline{x})_{1\times p} = \underline{1}^T X$

```
import numpy as np
df_mat=df.to_numpy()
df_mat.shape, df_mat.size
```

```
((150, 4), 600)
```

```
n=df_mat.shape[0]
one=np.array(np.ones(n))
df_mean=np.dot(one,df_mat)/n
df_mean
```

```
array([5.84333333, 3.05733333, 3.758     , 1.19933333])
```

공분산 행렬 $\Sigma_{p\times p}$

중심화 행렬 : $C_{n\times p} = X_{n\times p} - \underline{1}_{n\times 1} E(X)$

공분산행렬 : $\Sigma_{p\times p} = (C^T C)/(n-1)$

```
C=df_mat-df_mean
df_cov=np.dot(C.T,C)/(n-1)
df_cov
```

```
array([[ 0.68569351, -0.042434  ,  1.27431544,  0.51627069],
       [-0.042434  ,  0.18997942, -0.32965638, -0.12163937],
       [ 1.27431544, -0.32965638,  3.11627785,  1.2956094 ],
       [ 0.51627069, -0.12163937,  1.2956094 ,  0.58100626]])
```

상관계수 행렬 $R_{p\times p}$

상관계수 : $Corr(x_i, x_j) = \dfrac{COV(x_i, x_j)}{\sqrt{V(x_i)}\sqrt{V(x_j)}}$

그러므로 $R = D^{-1}\Sigma D^{-1}$, 대각행렬 D는 공분산행렬의 대각원소의 제곱근 값, 즉 각 확률변수 분산의 제곱근이므로 표준편차이다.

```
import numpy.linalg as la
D=np.diag(np.sqrt(np.diag(df_cov)))
D_inv=la.inv(D)
D_inv@df_cov@D_inv
```

```
array([[ 1.        , -0.11756978,  0.87175378,  0.81794113],
       [-0.11756978,  1.        , -0.4284401 , -0.36612593],
       [ 0.87175378, -0.4284401 ,  1.        ,  0.96286543],
       [ 0.81794113, -0.36612593,  0.96286543,  1.        ]])
```

회귀분석 OLS 행렬 계산 $\underline{y} = X\underline{b} + \underline{e}$, OLS 추정치 $\hat{\underline{b}} = (X^TX)^{-1}X^T\underline{y}$

```
import statsmodels.formula.api as smf
model=smf.ols(formula='iris["Sepal.Width"]~iris["Sepal.Length"]',
data=iris)
model.fit().summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3.4189	0.254	13.484	0.000	2.918	3.920
ris["Sepal.Length"]	-0.0619	0.043	-1.440	0.152	-0.147	0.023

```
y=np.array(iris['Sepal.Width'])
x=np.array(iris['Sepal.Length'])
one=np.ones(iris.shape[0])
X=np.vstack((one,x))
import numpy.linalg as la
la.inv(X@X.T)@X@y
```

```
array([ 3.41894684, -0.0618848 ])
```

다변량 정규분포 $\underline{x}_p \sim MN(\underline{\mu}_p, \Sigma_{p\times p})$ 및 이차형식

차수 p인 확률변수 벡터 $\underline{x}_p \sim N_p(\underline{\mu}_p, \Sigma_{p\times p})$, 평균이 $\underline{\mu}$, 공분산행렬이 Σ인 다변량 정규분포의 확률분포함수는 다음과 같다.

$$f_{\underline{x}}(\underline{x};\underline{\mu},\Sigma) = \frac{1}{(2\pi)^{p/2}\,|\Sigma|^{1/2}} \exp[-1/2(\underline{x}-\underline{\mu})'\Sigma^{-1}(\underline{x}-\underline{\mu})]$$

$\Sigma = \sigma^2 I$

확률변수는 서로 독립이고 등분산을 갖는다.

다변량 표준 정규분포

$\underline{x}_p \sim MN(\underline{\mu}_p, \Sigma_{p\times p})$ => $(\underline{x}-\underline{\mu})^T\Sigma^{-1}(\underline{x}-\underline{\mu}) \sim MVN(\underline{0}, I_{p\times p})$

성질

- 각 확률변수의 조건부 확률분포함수, 주변확률분포함수는 모두 정규분포를 따른다.
- 선형계수벡터 $\underline{a}^T = [a_1\ a_2\ \ldots,\ a_p]_{p\times 1}$에 대하여 $\underline{a}^T\underline{x}_{p\times 1} \sim MVN(\underline{a}^T\underline{\mu}, \underline{a}^T\Sigma\underline{a})$

이차형식 Quardratic form $\underline{x}^T A\,\underline{x}$

다변량 확률변수(열벡터) $\underline{x}_p$, 정방행렬 $A_{p\times p}$에 대하여 $\underline{x}^T A\,\underline{x}$을 이차형식이라 한다.

만약 $A = I$(항등행렬)인 경우

$$\underline{x}^T\underline{x} = x_1^2 + x_2^2 + \ldots + x_p^2$$

Positive semi-Definite matrix 양(준)정부호행렬

행렬 A는 양(준)정부호행렬 <=> 모든 $\underline{x}$에 대하여 $\underline{x}^T A\,\underline{x} > 0, \underline{x}^T A\,\underline{x} \geq 0$(준)

- 양정부호행렬의 고유값은 양수이다. (semi의 경우는 0을 포함한 양수)
- 양정부호행렬은 Choleski-분해(decomposition) $A = LL^*$(L= 하삼각행렬) 가능하다..

이차형식 분포

만약 확률변수 벡터가 평균벡터 $\underline{\mu}$, 공분산 행렬 Σ인 다변량정규분포를 따른다면 $\underline{x}_p \sim MVN(\underline{\mu}_p, \Sigma_{p\times p})$,

이차형식 $\underline{x}^T A \underline{x}$ (A=대칭행렬)에 대하여

- $E(\underline{x}^T A \underline{x}) = tr(A\Sigma) + \underline{\mu}^T A \underline{\mu}$
- $V(\underline{x}^T A \underline{x}) = 2tr(A\Sigma A\Sigma)) + 4\underline{\mu}^T A \Sigma A \underline{\mu}$
- $(\underline{x} - \underline{\mu})^T \Sigma^{-1} (\underline{x} - \underline{\mu}) \sim \chi^2(p)$이다.

이변량정규분포 생성 $\begin{bmatrix} x \\ y \end{bmatrix} \sim BN(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \sigma_x & \sigma_{xy} \\ \sigma_{xy} & \sigma_y \end{bmatrix}), \rho = corr(x,y)$

만약 $Z_1, Z_2 \sim iidN(0,1), X = \sigma_x Z_1 + \mu_x$ and $Y = \sigma_y[\rho Z_1 + \sqrt{1-\rho^2} Z_2 + \mu_y]$은 이변량 정규분포를 따른다. Holst, E. "The Bivariate Normal Distribution." https://www.ami.dk/research/bivariate/.

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim BN(\begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & 1.4 \\ 1.4 & 4 \end{bmatrix})$$

```
import numpy as np
z1=np.random.normal(0,1,100)
z2=np.random.normal(0,1,100)
x=z1+2
y=2*(0.7*z1+np.sqrt(1-0.7**2)*z2+1)
np.cov(x,y), np.corrcoef(x,y)
```

```
(array([[0.96525685, 1.15429555],
        [1.15429555, 3.68884376]]), array([[1.        , 0.61171685],
        [0.61171685, 1.        ]]))
```

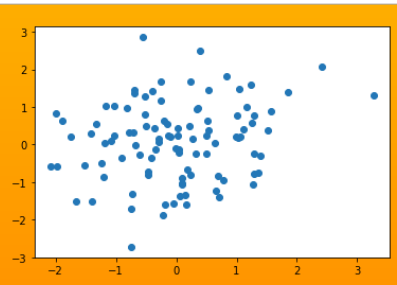
함수 이용 n umpy.random.multivariate_normal

```
import numpy as np
mean = [2, 1]
cov = [[1, 1.4], [1.4,4]]  # diagonal covariance
x, y = np.random.multivariate_normal(mean, cov, 100).T
np.cov(x,y), np.corrcoef(x,y)
```

```
(array([[1.05819297, 1.4231249 ],
        [1.4231249 , 3.85671894]]), array([[1.        , 0.70445229],
        [0.70445229, 1.        ]]))
```

산점도 그리기

```
import matplotlib.pyplot as plt
plt.plot(x, y,'o')
plt.show()
```



Affine transformations of the multivariate normal

만약 $X \sim N(\mu_X, \Sigma_X)$이고 $Y = LX + u$(L=선형변환행렬, u=임의의 벡터)이라면 확률변수(벡터) Y는 정규분포를 따르고 $\mu_Y = u + L\mu_X$ 이고 $\Sigma_Y = L\Sigma_X L^T$이다.

이변량 정규분포를 생성하는 경우에는 $X \sim N(\mu, \sigma^2 I)$을 사용하므로 Y의 공분산은 $\Sigma_Y = LL^T$이다.

선형변환행렬 L은 공분산행렬을 사용한 Cholesky Decomposition(분해)을 이용한다.

(X, Y)의 공분산행렬(양반정치행렬 Positive semidefinite matrix) $\Sigma = LL^*$, L=하한대각행렬, L^*=대각행렬 L의 conjugate transpose이다.

https://peterroelants.github.io/posts/multivariate-normal-primer/

$$Y \sim BN(\begin{bmatrix}2\\1\end{bmatrix}, \begin{bmatrix}1 & 1.4\\1.4 & 4\end{bmatrix})$$

```
d=2 # Number of dimensions
n=100 # sample size
mean = np.matrix([[2], [1]])
covariance = np.matrix([[1,1.4], [1.4,4]])

# Create L
L = np.linalg.cholesky(covariance)
X = np.random.normal(size=(d, n))
Y = L.dot(X) + mean
```

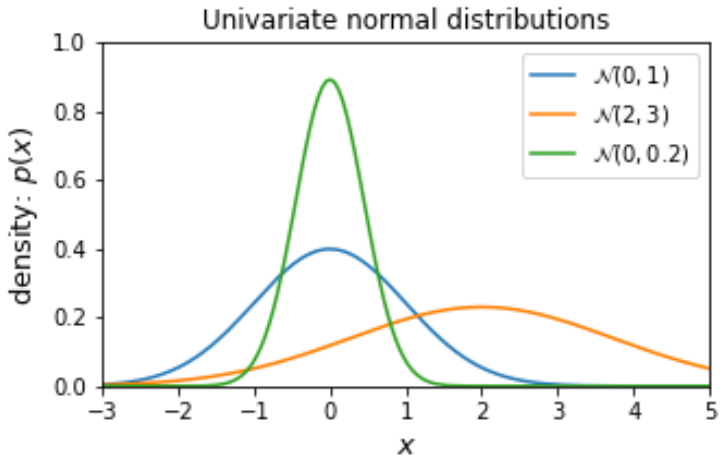
함수 정의 이용 https://peterroelants.github.io/posts/multivariate-normal-primer/

일변량정규분포함수

```
def univariate_normal(x, mean, variance):
    return ((1. / np.sqrt(2 * np.pi * variance)) * np.exp(-(x - mean)**2 / (2 * variance)))
```

일변량 정규분포함수 그리기

```
# Plot different Univariate Normals
x = np.linspace(-3, 5, num=150)
fig = plt.figure(figsize=(5, 3))
plt.plot(
    x, univariate_normal(x, mean=0, variance=1),
    label="$\mathcal{N}(0, 1)$")
plt.plot(
    x, univariate_normal(x, mean=2, variance=3),
    label="$\mathcal{N}(2, 3)$")
plt.plot(
    x, univariate_normal(x, mean=0, variance=0.2),
    label="$\mathcal{N}(0, 0.2)$")
plt.xlabel('$x$', fontsize=13)
plt.ylabel('density: $p(x)$', fontsize=13)
plt.title('Univariate normal distributions')
plt.ylim([0, 1])
plt.xlim([-3, 5])
plt.legend(loc=1)
fig.subplots_adjust(bottom=0.15)
plt.show()
```



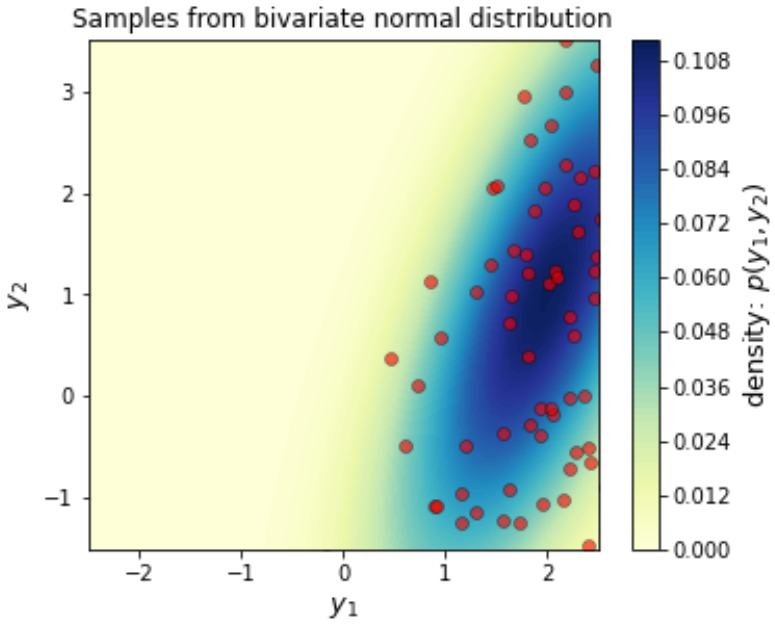
다변량 정규분포함수 정의

```
def multivariate_normal(x, d, mean, covariance):
   x_m = x - mean
    return (1. / (np.sqrt((2 * np.pi)**d *np.linalg.det(covariance))) *
np.exp(-(np.linalg.solve(covariance, x_m).T.dot(x_m)) / 2))
```

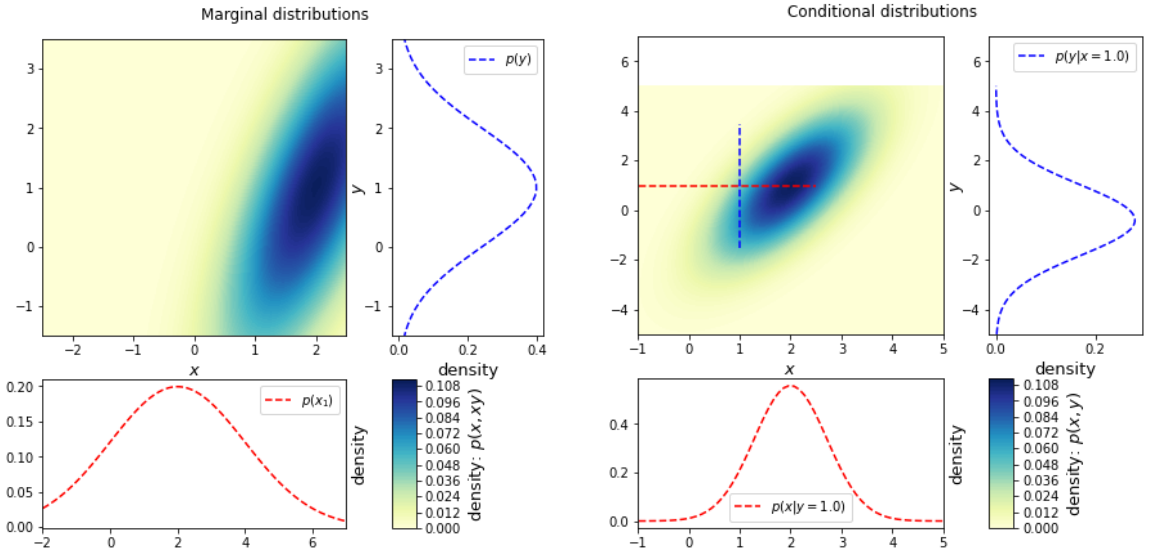
다변량 정규분포함수 그리기

```
def generate_surface(mean, covariance, d):
    """Helper function to generate density surface."""
    nb_of_x = 100 # grid size
    x1s = np.linspace(-5, 5, num=nb_of_x)
    x2s = np.linspace(-5, 5, num=nb_of_x)
    x1, x2 = np.meshgrid(x1s, x2s) # Generate grid
    pdf = np.zeros((nb_of_x, nb_of_x))
    # Fill the cost matrix for each combination of weights
    for i in range(nb_of_x):
        for j in range(nb_of_x):
            pdf[i,j] = multivariate_normal(
                np.matrix([[x1[i,j]], [x2[i,j]]]),
                d, mean, covariance)
    return x1, x2, pdf  # x1, x2, pdf(x1,x2)
```

```
import matplotlib.pyplot as plt
from matplotlib import cm
fig, ax = plt.subplots(figsize=(6, 4.5))
# Plot bivariate distribution
x1, x2, p = generate_surface(mean, covariance, d)
con = ax.contourf(x1, x2, p, 100, cmap=cm.YlGnBu)
# Plot samples
ax.plot(Y[0,:], Y[1,:], 'ro', alpha=.6,
        markeredgecolor='k', markeredgewidth=0.5)
ax.set_xlabel('$y_1$', fontsize=13)
ax.set_ylabel('$y_2$', fontsize=13)
ax.axis([-2.5, 2.5, -1.5, 3.5])
ax.set_aspect('equal')
ax.set_title('Samples from bivariate normal distribution')
cbar = plt.colorbar(con)
cbar.ax.set_ylabel('density: $p(y_1, y_2)$', fontsize=13)
plt.show()
```



주변 정규분포함수_조건부 확률밀도함수 그리기



```
d = 2  # dimensions
mean = np.matrix([[2], [1]])
cov = np.matrix([[1, 1.4], [1.4,4]])

# Get the mean values from the vector
mean_x = mean[0,0]
mean_y = mean[1,0]
# Get the blocks (single values in this case) from
#  the covariance matrix
A = cov[0, 0]
B = cov[1, 1]
C = cov[0, 1]  # = C transpose in this case
```

```
import matplotlib.gridspec as gridspec
from mpl_toolkits.axes_grid1 import make_axes_locatable

fig = plt.figure(figsize=(7, 7))
gs = gridspec.GridSpec(2, 2, width_ratios=[2, 1], height_ratios=[2, 1])
# gs.update(wspace=0., hspace=0.)
plt.suptitle('Marginal distributions', y=0.93)

# Plot surface on top left
ax1 = plt.subplot(gs[0])
x, y, p = generate_surface(mean, cov, d)
# Plot bivariate distribution
con = ax1.contourf(x, y, p, 100, cmap=cm.YlGnBu)
ax1.set_xlabel('$x$', fontsize=13)
ax1.set_ylabel('$y$', fontsize=13)
ax1.yaxis.set_label_position('right')
ax1.axis([-2.5, 2.5, -1.5, 3.5])

# Plot y
ax2 = plt.subplot(gs[1])
y = np.linspace(-5, 5, num=100)
py = univariate_normal(y, mean_y, A)
# Plot univariate distribution
ax2.plot(py, y, 'b--', label=f'$p(y)$')
ax2.legend(loc=0)
ax2.set_xlabel('density', fontsize=13)
ax2.set_ylim(-1.5, 3.5)

# Plot x
ax3 = plt.subplot(gs[2])
x = np.linspace(-2, 7, num=100)
px = univariate_normal(x, mean_x, B)
# Plot univariate distribution
ax3.plot(x, px, 'r--', label=f'$p(x_1)$')
ax3.legend(loc=0)
ax3.set_ylabel('density', fontsize=13)
ax3.yaxis.set_label_position('right')
ax3.set_xlim(-2, 7)

# Clear axis 4 and plot colarbar in its place
ax4 = plt.subplot(gs[3])
ax4.set_visible(False)
divider = make_axes_locatable(ax4)
cax = divider.append_axes('left', size='20%', pad=0.05)
cbar = fig.colorbar(con, cax=cax)
cbar.ax.set_ylabel('density: $p(x, xy)$', fontsize=13)
plt.show()
```

조건부 확률분포함수

```
# Calculate x|y
y_condition = 1.  # To condition on y
mean_xgiveny = mean_x + (C * (1/B) * (y_condition - mean_y))
cov_xgiveny = A - C * (1/B) * C

# Calculate y|x
x_condition = 1.  # To condition on x
mean_ygivenx = mean_y + (C * (1/A) * (x_condition - mean_x))
cov_ygivenx = B - (C * (1/A) * C)
```

```
# Plot the conditional distributions
fig = plt.figure(figsize=(7, 7))
gs = gridspec.GridSpec(
    2, 2, width_ratios=[2, 1], height_ratios=[2, 1])
# gs.update(wspace=0., hspace=0.)
plt.suptitle('Conditional distributions', y=0.93)

# Plot surface on top left
ax1 = plt.subplot(gs[0])
x, y, p = generate_surface(mean, cov, d)
# Plot bivariate distribution
con = ax1.contourf(x, y, p, 100, cmap=cm.YlGnBu)
# y=1 that is conditioned upon
ax1.plot([-2.5, 2.5], [y_condition, y_condition], 'r--')
# x=-1. that is conditioned upon
ax1.plot([x_condition, x_condition], [-1.5, 3.5], 'b--')
ax1.set_xlabel('$x$', fontsize=13)
ax1.set_ylabel('$y$', fontsize=13)
ax1.yaxis.set_label_position('right')
ax1.axis([-1,5, -5,7])

# Plot y|x
ax2 = plt.subplot(gs[1])
yx = np.linspace(-5, 5, num=100)
pyx = univariate_normal(yx, mean_ygivenx, cov_ygivenx)
# Plot univariate distribution
ax2.plot(pyx, yx, 'b--',
         label=f'$p(y|x={x_condition:.1f})$')
ax2.legend(loc=0)
ax2.set_xlabel('density', fontsize=13)
ax2.set_ylim(-5,7)

# Plot x|y
ax3 = plt.subplot(gs[2])
xy = np.linspace(-5, 5, num=100)
pxy = univariate_normal(xy, mean_xgiveny, cov_xgiveny)
# Plot univariate distribution
ax3.plot(xy, pxy, 'r--',
         label=f'$p(x|y={y_condition:.1f})$')
ax3.legend(loc=0)
ax3.set_ylabel('density', fontsize=13)
ax3.yaxis.set_label_position('right')
ax3.set_xlim(-1,5)

# Clear axis 4 and plot colarbar in its place
ax4 = plt.subplot(gs[3])
ax4.set_visible(False)
divider = make_axes_locatable(ax4)
cax = divider.append_axes('left', size='20%', pad=0.05)
cbar = fig.colorbar(con, cax=cax)
cbar.ax.set_ylabel('density: $p(x, y)$', fontsize=13)
plt.show()
```

벡터 Norm

벡터의 단일 척도로 추상화된 길이이다.

벡터 $\underline{x} = \begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}$ 에 대한 놈 정의는 다음과 같다. $\underline{x} = \begin{pmatrix} 1 \\ 3 \\ 4 \end{pmatrix}$

- $p-Norm: |\underline{x}|_p = (\sum_i |x_i|^p)^{(1/p)}$
- $L1-norm: |\underline{x}| = \sum_i |x_i| = 8$
- $L2-norm: |\underline{x}|_2 = (\sum_i x_i^2)^{1/2} = \sqrt{26}$: 유크리디안 거리
- $L\infty-norm: |\underline{x}|_\infty = max|x_i| = 4$: 최대값n

```
import numpy.linalg as la
L1=la.norm(np.array([[1,3,4]]),1)
L2=la.norm(np.array([[1,3,4]]),2)
print(L1,L2)
```

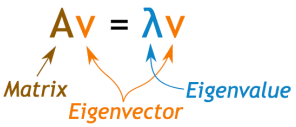
```
4.0 5.099019513592785
```

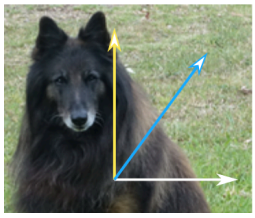
행렬의 L1-NORM은 각 열의 최대 L1-norm 값이고 L2-norm은 모든 원소를 제곱하여 더한 제곱근 값이다. e

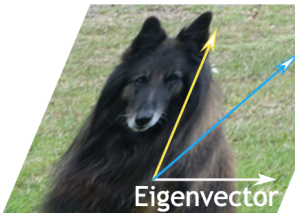
```
B=np.array([[1,0,7],[9,5,8],[3,4,2]])
la.norm(B,1),la.norm(B,2)
```

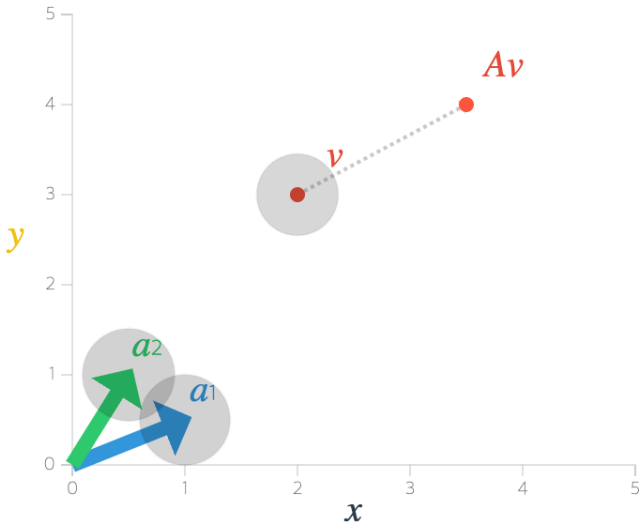
```
(17.0, 14.841898441682643)
```

고유값 eigen value









$$A = \begin{bmatrix} a_1,x & a_2,x \\ a_1,y & a_2,y \end{bmatrix} = \begin{bmatrix} 1.00 & 0.50 \\ 0.50 & 1.00 \end{bmatrix}$$

$$v = \begin{bmatrix} 2.00 \\ 3.00 \end{bmatrix}$$

$$Av = \begin{bmatrix} 3.50 \\ 4.00 \end{bmatrix}$$

행렬 A는 v 벡터를 방향의 변화 없이 변환한다. Eigen 독일어 어원, own 자신의, typycal 전형적인

고유값 λ=1 => 변화없음, $\lambda = 2$=> 길이가 2배, $\lambda = -1$=> 방향 반대

고유값 in Python

```
import numpy.linalg as la
eigval,eigvec=la.eig(np.array([[1,0
.5],[0.5,1]]))
print(eigval,'\n',eigvec)
```

```
[1.5 0.5]
```

고유값 구하기

차수가 k인 정방행렬 $A_{k\times k}$(선형계수행렬), 벡터 $\underline{x}$로 이루어진 고유 방정식(characteristic equation) $A\underline{x} = \lambda\underline{x}$를 만족하는 λ들을 고유치(eigen value, characteristic value, latent value)라 하고 각 고유치에 대해 고유방정식을 만족하는 벡터를 고유 벡터(eigen vector)라 한다.

$(A - \lambda I)\underline{x} = 0$을 만족하는 해는 Cramer Rule에 의해 $det(A - \lambda I) = 0$을 만족하는 λ를 구하면 된다. 이렇게 구한 것을 고유값이라 한다.

- 차수가 n인 정방행렬 $A_{n\times n}$ 의 $|A - \lambda I|$ 행렬식을 계산한다. 이 방정식은 차수가 n인 λ의 다항방정식이다. $f(\lambda) = a_1\lambda^n + a_2\lambda^{n-1} + \ldots$

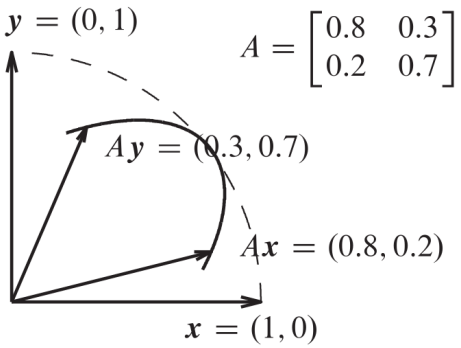
- $f(\lambda) = a_1\lambda^n + a_2\lambda^{n-1} + \ldots = 0$ 특성방정식(characteristic equation) 를 만족하는 $\lambda_1, \lambda_2, \ldots\lambda_n$ (n개 존재) 고유치(eigen value, characteristic value, latent value)라 한다.
- 이렇게 구한 λ_i는 $A - \lambda I$ 행렬을 singular로 만든다

고유벡터 구하기

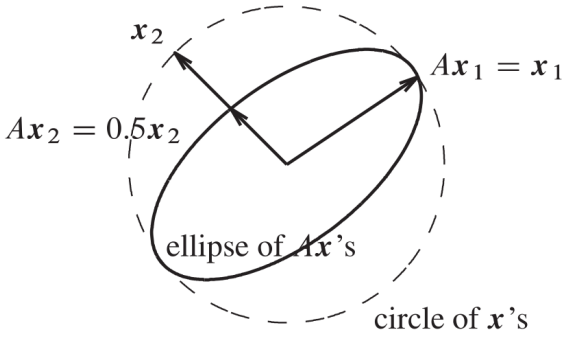
- 각 λ_i에 대하여 $(A - \lambda I)\underline{x} = \underline{0}$을 만족하는 $\underline{x}$을 구하면 이를 고유벡터라 한다.
- [단점] $A - \lambda I$ 행렬을 singular이므로 고유벡터 $\underline{x}$는 무수히 많이 존재한다.
- [장점] 이 성질 때문에 요인 분석에서 요인 회전이 가능하다.
- 활용 시 $\underline{x}'\underline{x} = 1$을 만적하는 정규(Norm) 고유벡터만를 사용한다. $\underline{e}_i$ - 정규 고유벡터 기호

성질

- 고유치의 합은 행렬 A의 대각원소의 합니다. $\sum \lambda_i = tr(A)$
- 고유벡터는 서로 독립이다. => 좌표를 독립(상호 독립, 직교) 공간에 표현한다.



These are not eigenvectors



$A\boldsymbol{x}$ lines up with $\boldsymbol{x}$ at eigenvectors

활용

여기서는 고유치와 고유 벡터 계산 방법에 대해서는 다루지 않겠지만 언급하고 싶은 것은 고유치와는 달리 고유벡터는 무수히 많이 존재한다는 것이다. 이 성질 때문에 요인 분석에서 요인 회전이 가능하다.

고유치가 사용되는 예제를 보면 $\underline{n}_t$ 를 시점 t에서의 각 연령 분포 벡터라 하고 행렬A를 각 연령에서 시점 t에서 (t+1)까지 살아 남을 확률에 대한 행렬이라면 $\underline{n}_{t+1} = A\underline{n}_t$ 이다. 만약 시간에 따라 인구의 분포가 stable 하다면 $\underline{n}_{t+1} = \lambda\underline{n}_t$ 가 될 것이고 $A\underline{n}_t = \lambda\underline{n}_t$ 이 될 것이다.

개념 [데이터 차원 축소에 사용]

또한 고유 벡터를 구하는 식에서 $A\underline{e_i} = \lambda_i \underline{e_i}$을 살펴보면 행렬 A의 크기 고유치에 나타나 있다. 즉 고유치는 행렬 A의 크기(변동)에 대한 정보이다. 이 개념이 다변량 분석에 이용된다.

- 데이터 변수의 변동의 크기, 데이터의 공분산으로부터 구한 고유값은 원 변수들의 총변동의 크기를 반영함
- 변수의 변동(분산)은 관심 개체를 구별하는 능력 - 변수의 변동이 크다는 것은 개채를 구별할 수 있는 능력이 크다는 것임
- 예를 들어 H대학 통계학과 학생들의 고등학교 수학 성적과 영어성적을 조사한 자료이다.
- 수학평균=70점, 분산=20점 - 영어평균=80, 분산=5점
- 어느 과목을 이용하여 학생들의 능력을 구별하는 것이 좋을까? 당연히 변동(분산)이 큰 수학성적으로 학생들을 구별해야 용이하게 구별할 수 있음

인구 이동 마코프 체인. https://setosa.io/ev/markov-chains/