인공지능수학 개론

(저자 김종락)

(임시적인) 목록

Chapter 3. 행렬의 연산 및 역행렬

연립선형방정식을 쉽게 푸는 방법은 없을까?

중학교 때 변수가 두 개인 일차방정식 두 개를 푸는 방법을 기억할 것이다. 가장 핵심적인 방법은 변수 x, y중의 하나를 소거하여 간단해진 식에서 정답을 구하는 방법이었다. 이것을 소거법이라고 한다. 그런데 놀랍게도 변수가 3개 이상인 경우에도 이 방법이 가장 효율적이라는 것을 알 수 있다. 이것을 가우스-조던 소거법(Gauss-Jordan elimination method)라고 한다.

이 방법을 정방행렬에 적용할 경우 역행렬을 효과적으로 구할 수 있다. 역행렬의 정의, 대각행렬, 삼각행렬 등에 대하여 알아본다. 또한 행렬이 만족하는 대수적 구조에 대하여 알아본다.

■ 가우스-조단 소거법

- **가우스-조단 소거법**(Gauss-Jordan elimination method)
  - 연립선형방정식을 표현한 행렬방정식의 **계수행렬 부분**을 **기약행 사다리꼴 행렬**로 변환하여 해를 구하는 방법

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -2 \\ 4 \end{bmatrix}$$

$$x_1 = 3,\ x_2 = -2,\ x_3 = 4$$

  - 행렬방정식 $A\boldsymbol{x} = \boldsymbol{b}$에 행 연산을 하여 좌변의 행렬 A를 I로 만들면, 행렬방정식의 해를 구할 수 있음.

(예제). 가우스-조르단 소거법으로 다음을 푸시오.

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 1 \\ 3 & -2 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ -1 \end{bmatrix}$$

(정답) $x_1 = 2, x_2 = -1, x_3 = 3$

(Exercise) 가우스 소거법으로 다음을 푸시오.

$$\begin{cases} 2x_1 + 2x_2 + 4x_3 = 18 \\ x_1 + 3x_2 + 2x_3 = 13 \\ 3x_1 + x_2 + 3x_3 = 14 \end{cases}$$

(정답) $x_1 = 1, x_2 = 2, x_3 = 3$

●행렬의 거듭제곱

정방행렬 A에 대하여 A^k는 A를 k번 거듭제곱한 것이다.

(1) $A^0 = I$

(2) $(A^b)^c = A^{bc}$

(3) $A^b A^c = A^{b+c}$

▶ 증명?

▶ A^{-1}의 의미는?

● 행렬 덧셈과 곱셈은 다음의 성질을 만족한다.
크기가 같은 행렬 A, B, C와 스칼라 a, b에 대하여 다음이 만족된다.

(1) $A+\mathbf{0}=\mathbf{0}+A=A$	(합에 대한 항등원인 영행렬)
(2) $IA=AI=A$	(곱에 대한 항등원인 단위행렬)
(3) $A+B=B+A$	(합에 대한 교환법칙)
(4) $(A+B)+C=A+(B+C)$	(합에 대한 결합법칙)
(5) $(AB)C=A(BC)$	(곱에 대한 결합법칙)
(6) $A(B+C)=AB+AC$	(분배법칙)
(7) $(A+B)C=AC+BC$	(분배법칙)
(8) $a(B+C)=aB+aC$	
(9) $(a+b)C=aC+bC$	
(10) $(ab)C=a(bC)$	
(11) $a(BC)=(aB)C=B(aC)$	

▶ 이러한 성질을 만족하는 연산 체계를 무엇이라고 불리는가?

● 역행렬

정방행렬 $A=[a_{ij}]_{n\times n}$에 대해 다음 성질을 만족하는 행렬 B를 A의 **역행렬**inverse matrix 이라 한다.

$$AB = BA = I_n$$

A의 역행렬은 A^{-1}로 나타내고, A^{-1}는 'A의 역행렬' 또는 'A inverse'라고 읽는다. I_n은 $n\times n$ 단위행렬이다. 행렬과 역행렬의 곱은 단위행렬이 되어야 하므로, 다음 성질을 만족한다.

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

▶ 역행렬은 실수 a의 역원 a^{-1}과 같은 역할을 한다.

● 역행렬의 uniqueness

n차 정방행렬 A의 역행렬이 존재하면, 하나 뿐이다.

▶ (증명)

● 2차 정방행렬의 역행렬

행렬 $A=\begin{bmatrix} a & b \\ c & d \end{bmatrix}$에 대하여 $ad-bc\neq 0$이면,

역행렬은 $A^{-1}=\dfrac{1}{ad-bc}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$이다.

▶ (문제) 가우스 소거법을 이용하여 증명하시오.

● 역행렬의 성질

n차 정방행렬 A, B가 가역이고 α가 0아닌 스칼라 일 때 다음을 만족한다.

(1) A^{-1}는 가역이고, $(A^{-1})^{-1} = A$이다.

(2) AB는 가역이고, $(AB)^{-1} = B^{-1}A^{-1}$이다.

(3) αA는 가역이고, $(\alpha A)^{-1} = \dfrac{1}{\alpha}A^{-1}$이다.

(4) A^{k}은 가역이고, $(A^{-1})^{k} = (A^{k})^{-1}$이다(여기서 k는 0 이상의 정수이다).

▶ (증명)

● 전치행렬(transpose of A)

$A=(a_{ij})$라고 할 때 $A^T=(a_{ji})$를 A의 transpose라고 한다.

● 전치행렬의 성질

(1) $(A^\top)^\top = A$

(2) $(A+B)^\top = A^\top + B^\top$

(3) $(AB)^\top = B^\top A^\top$

(4) $(\alpha A)^\top = \alpha A^\top$

(5) A가 가역이면, $(A^\top)^{-1} = (A^{-1})^\top$ 이다.

▶ (증명)

● 대칭행렬(symmetric matrix)과 반대칭행렬(skey-symmetric matrix)

$A = A^T$:

$A = -A^T$:

▶ 대칭행렬의 특징은?

▶ 반대칭행렬의 특징은?

● 정방행렬의 표현 방법

(1) 정방행렬 A에 대해 $A + A^\top$는 대칭행렬이고 $A - A^\top$는 반대칭행렬이다.

(2) $A = \frac{1}{2}(A + A^\top) + \frac{1}{2}(A - A^\top)$이다. 즉 정방행렬 A는 대칭행렬인 $\frac{1}{2}(A + A^\top)$와 반대칭행렬인 $\frac{1}{2}(A - A^\top)$의 합으로 나타낼 수 있다.

▶ (증명)

● 대각행렬(diagonal matrix)

정방행렬 $A=(a_{ij})$에 대하여 $a_{ij}=0 \text{ if } i \neq j$일 때 A를 대각행렬이라고 한다.
이 경우 간단히 $A=(d_{ii})$로 표현한다.

▶ 같은 크기의 두 대각행렬의 곱은 대각행렬이다.

▶ 임의의 n차 정방행렬 A와 n차 대각행렬 D에 대하여 AD와 DA의 관계는?

● 대각합(trace(A))

정방행렬 $A=(a_{ij})$에 대하여 대각성분에 있는 원소들의 합을 trace of A라고 한다.

▶ (예제)

● 대각합의 성질

(1) n차 정방행렬 A, B에 대해, $tr(A+B) = tr(A) + tr(B)$이다.

(2) n차 정방행렬 A와 스칼라 c에 대해, $tr(cA) = c \cdot tr(A)$이다.

(3) $n \times m$ 행렬 A와 $m \times n$ 행렬 B에 대해, $tr(AB) = tr(BA)$이다.

(4) n차 정방행렬 A, B, C에 대해, $tr(ABC) = tr(CAB) = tr(BCA)$이다.

▶ (증명)

● 삼각행렬(triangular matrix)
- upper triangular matrix U

- lower triangular matrix L

▶ (예제)

● (정리)

(1) 상삼각행렬과 상삼각행렬의 곱은 상삼각행렬이다.
(2) 하삼각행렬과 하삼각행렬의 곱은 하삼각행렬이다.

▶ (증명)

● 열벡터와 행벡터의 곱

$m \times 1$ 벡터 $\boldsymbol{u}$와 $n \times 1$ 벡터 $\boldsymbol{v}$가 있을 때, 열벡터인 $\boldsymbol{u}$와 행벡터인 $\boldsymbol{v}^\top$ 의 곱 $\boldsymbol{uv}^\top$는 $m \times n$ 행렬이 된다.

$$\boldsymbol{uv}^\top = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \cdots & u_mv_n \end{bmatrix}$$

▶ (예제)

$$\boldsymbol{u} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}, \quad \boldsymbol{v} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} \quad \Longrightarrow \quad \boldsymbol{uv}^\top = \begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} 2 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 6 & 12 & 18 \end{bmatrix}$$

이것을 $u \otimes v$ (u tensor v)라고도 쓴다.

● 행렬 A와 열벡터 x에 대하여 Ax를 쉽게 계산하는 방법은?

●

■ 실습

▶ 행렬을 출력하는 함수

```
import numpy as np

# 행렬 A를 출력하는 함수
def pprint(msg, A):
    print("---", msg, "---")
    (n,m) = A.shape
    for i in range(0, n):
        line = ""
        for j in range(0, m):
            line += "{0:.2f}".format(A[i,j]) + "₩t"
        print(line)
    print("")
```

```
A = np.array([[1., 2.], [3., 4.]])
pprint("A", A)
```

```
--- A ---
1.00    2.00
3.00    4.00
```

```
Ainv1 = np.linalg.matrix_power(A, -1) # matrix_power( )를 사용한 역행렬 A^-1 계산
pprint("linalg.matrix_power(A, -1) => Ainv1", Ainv1)

Ainv2 = np.linalg.inv(A) # inv( )를 사용한 역행렬 A-1 계산
pprint("np.linalg.inv(A) => Ainv2", Ainv2)

pprint("A*Ainv1", np.matmul(A, Ainv1)) # 행렬 A와 역행렬 A-1의 곱
pprint("A*Ainv2", np.matmul(A, Ainv2)) # 행렬 A와 역행렬 A-1의 곱
```

```
--- linalg.matrix_power(A, -1) => Ainv1 ---
-2.00   1.00
1.50    -0.50

--- np.linalg.inv(A) => Ainv2 ---
-2.00   1.00
1.50    -0.50

--- A*Ainv1 ---
1.00    0.00
0.00    1.00

--- A*Ainv2 ---
1.00    0.00
0.00    1.00
```

```python
B = np.random.rand(3,3) # 난수를 이용한 3x3 행렬 B 생성
pprint("B =", B)
Binv = np.linalg.inv(B) # 역행렬 B-1 계산
pprint("Binv =", Binv)
pprint("B*Binv =", np.matmul(B, Binv)) # 행렬 B와 역행렬 B-1의 곱
```

```
--- B = ---
0.44    0.40    0.42
0.28    0.13    0.48
0.52    0.41    0.84

--- Binv = ---
6.88    12.70   -10.76
-1.23   -11.83  7.43
-3.69   -2.11   4.26

--- B*Binv = ---
1.00    0.00    -0.00
0.00    1.00    -0.00
0.00    0.00    1.00
```

```python
# CX = D의 해 계산
C = np.array([[5, 3, 2, 1], [6, 2, 4, 5], [7, 4, 1, 3], [4, 3, 5, 2]])
D = np.array([[4], [2], [5], [1]])
x = np.matmul(np.linalg.inv(C), D)
pprint("x", x) # 해 x 출력
pprint("C*x", np.matmul(C, x)) # C*x의 결과가 D와 같은지 확인
```

```
--- x ---
1.31
-0.38
-0.31
-0.77

--- C*x ---
4.00
2.00
5.00
1.00
```