# HW #6 - 선형대 수학 기초 응용

GitHub Code Link

MathVision22
02221081 황지현

- Q1. 다음을 증명하시오
  - $(AB)^{-1} = B^{-1}A^{-1}$
  - $(cA)^{-1} = \frac{1}{c}A^{-1}$
  - $A = \mathrm{diag}(c_1, \cdots, c_n) \rightarrow A^{-1} = \mathrm{diag}(\frac{1}{c_1}, \cdots, \frac{1}{c_n})$ (단, $c_i \neq 0$)

**Q1-1**

A, B 의 역행렬이 존재하면 역행렬을 곱하였을 때 I (단위행렬)가 되는지 확인

(AB)·( $B^{-1}$ $A^{-1}$ )

= AI $A^{-1}$ = A $A^{-1}$

= I

∴ $(AB)^{-1}$ = $B^{-1}$ $A^{-1}$

**Q1-2**

Q1-1 과 동일한 방식을 사용. cA 와 그의 역행렬을 곱하여 E 가 되는지 확인

cA $\frac{1}{c}$ $A^{-1}$ =c $\frac{1}{c}$ A $A^{-1}$

= 1 · I = I

∴ $(cA)^{-1}$ = $\frac{1}{c}$ $A^{-1}$

## Q1-3

Diagonal Matrix : 대각 원소만 0 이 아닌 행렬

$$\text{A} = \text{diag}(\ c_1\ ,\ c_2\ , \dots ,\ c_n\ ) = \begin{pmatrix} c_1 & 0 & \dots & 0 \\ 0 & c_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & c_n \end{pmatrix}$$

$$A^{-1} = \text{diag}(\ \frac{1}{c_1}\ ,\ \frac{1}{c_2}\ , \dots ,\ \frac{1}{c_n}\ ) = \begin{pmatrix} \frac{1}{c_1} & 0 & \dots & 0 \\ 0 & \frac{1}{c_2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \frac{1}{c_n} \end{pmatrix}$$ 라고 가정

두 행렬 A, $A^{-1}$ 을 곱하면

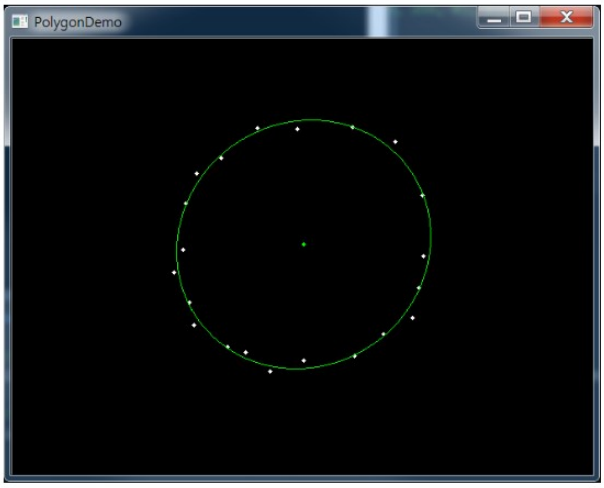
$$\begin{pmatrix} c_1 & 0 & \dots & 0 \\ 0 & c_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & c_n \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{c_1} & 0 & \dots & 0 \\ 0 & \frac{1}{c_2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \frac{1}{c_n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{pmatrix}$$

$$\therefore\ A^{-1} = \text{diag}(\ \frac{1}{c_1}\ ,\ \frac{1}{c_2}\ , \dots ,\ \frac{1}{c_n}\ )$$

### Q1 comment

수학적 정의를 적용하여 수식(행렬식)에 적용하여 풀이할 수 있었다.

- Q2. 사용자로부터 n(n>=3)개의 점을 입력받은 후 입력 받은 점들을 근사하는 원을 구하여 화면에 도시하는 프로그램을 작성하시오.
  - 리포트 + 구현코드(GitHub) 제출




* 리포트 포함 내용:
- 핵심코드+실행화면 캡처
- 결론

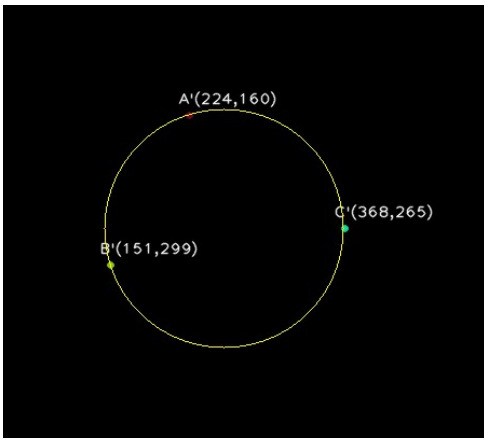
Pseudo Inverse 사용

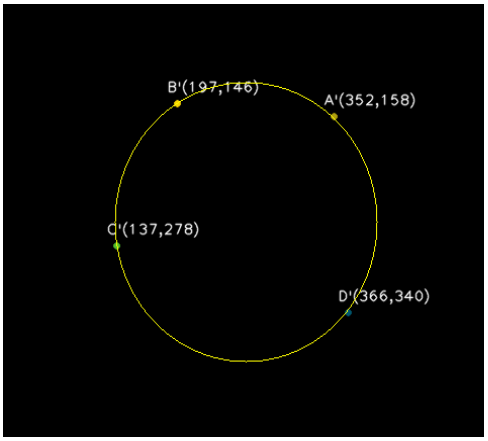
A 의 Pseudo Inverse = $(A^T A)^{-1} \quad A^T$

**핵심 코드**

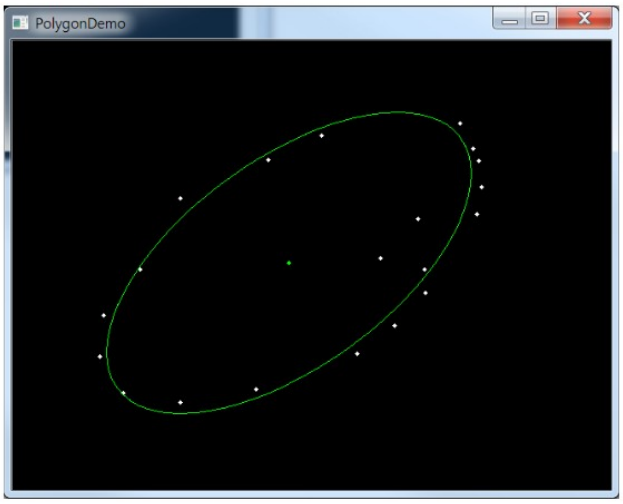
```
if len(left_m) >= 3 :
    l_m = np.array(left_m)
    r_m = np.array(right_m)
    A_plus = np.linalg.inv(l_m.T @ l_m) @ l_m.T
    result = A_plus @ r_m

    a,b,c = result
    center_x = int(-1/2 * a)
    center_y = int(-1/2 * b)
    radius = int(np.sqrt(-c + 1/4*a**2 + 1/4*b**2))
    cv2.circle(bg, (center_x, center_y), radius, (0,255,255), 1)
    cv2.putText(bg, f"the equation of circle  : (x-{center_x})^2 + (y-{center_y})^2 = {radius*2}",
                (5,30),
                cv2.FONT_HERSHEY_PLAIN, 1.0, (255,255,255), 1, cv2.LINE_AA)
    cv2.imshow('polygon', bg)

    draw_on = False
```

**실행 화면**



- Q3. 사용자로부터 n(n>=4)개의 점을 입력받은 후 입력 받은 점들을 근사하는 타원을 구하여 화면에 도시하는 프로그램을 작성하시오.
  - 리포트 + 구현코드(GitHub) 제출




* 리포트 포함 내용: 실행화면 캡처
- singular value 값 확인
- n = 4, n = 5인 경우 테스트 (singular value 값 확인)
- Ax = 0이 되는지 확인
- 결론

**참고 수업자료**

- Ax = 0

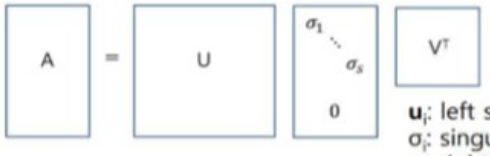
$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

$$\begin{pmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ \vdots & & & & & \\ x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix} = \mathbf{0}$$

$$\mathbf{Ax} = \mathbf{0}$$

- SVD(특이값 분해)

$$A = U\Sigma V^T$$

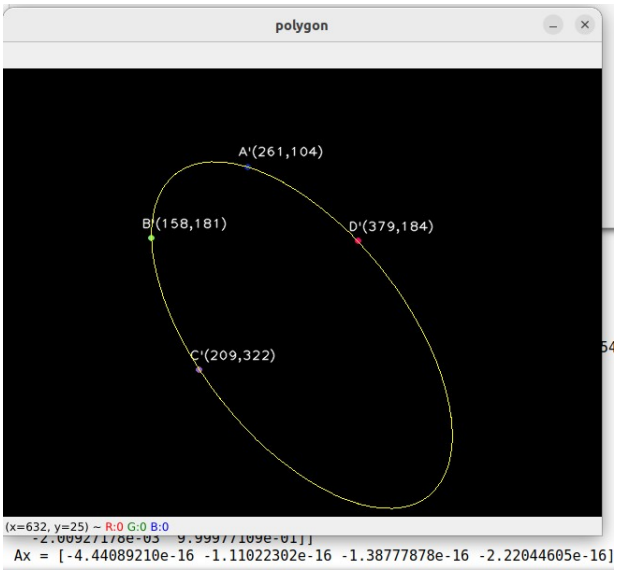


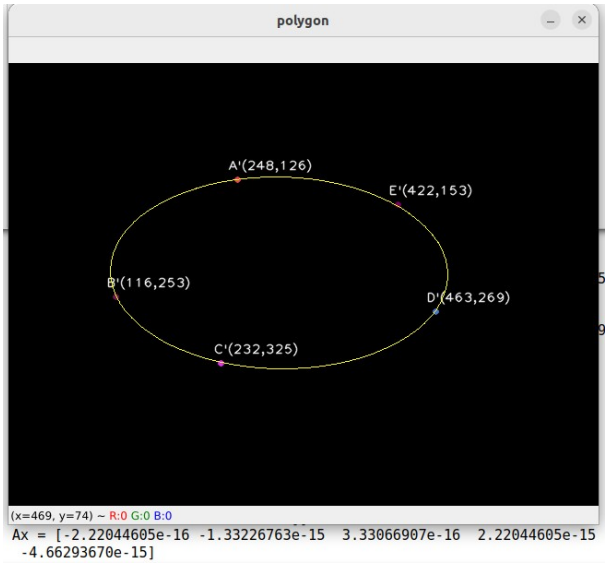

$\mathbf{u}_i$: left singular vector
$\sigma_i$: singular value
$\mathbf{v}_i$: right singular vector

- 선형시스템 $A\mathbf{x} = \mathbf{0}$의 해
  - A를 특이값 분해 ($A = U\Sigma V^T$)
  - V의 열벡터 중에서 최소특이값에 대응되는 열벡터가 해
    - 보통 $\mathbf{x} = \mathbf{v}_n$ (V의 가장 오른쪽 열벡터)이 최소자승(least squares) 해
    - 최소특이값이 0이면 $\mathbf{v}_n$는 Ax =0의 완벽한 해
    - 최소특이값이 0이 아니면 Ax = 0의 근사해 ($\|Ax\|^2$를 최소로 하는)

**실행 화면**

좌측 : n = 4, 우측 : n = 5




Ax = 0 근사값이 나옴을 확인할 수 있다

**핵심 코드**

```
    elif event == cv2.EVENT_RBUTTONDOWN :  # 입력 마침 (우클릭)
        if len(left_m) >= 3 :
            l_m = np.array(left_m)
            U,s,Vt = np.linalg.svd(l_m, full_matrices=True)
            print(f"특이값 : {s}")
            print(f"우측 특이벡터 : {Vt}")
            result = Vt[-1]
            a,b,c,d,e,f = result
            print(f"A*x = {l_m@result}")

            center_x = (2*c*d-b*e) / (b**2-4*a*c)
            center_y = (2*a*e-b*d) / (b**2-4*a*c)

            theta = np.arctan2(b, a-c)/2 # angle

            e1 = a*np.cos(theta)**2 + b*np.cos(theta)*np.sin(theta) + c*np.sin(theta)**2
            e2 = a*np.sin(theta)**2 - b*np.cos(theta)*np.sin(theta) + c*np.cos(theta)**2

            scale_inv = (a*center_x**2 + b*center_x*center_y + c*center_y**2) - f

            l_length = scale_inv/e1
            s_length = scale_inv/e2
            if l_length > 0 and s_length > 0 :
                l_length = np.sqrt(l_length)
                s_length = np.sqrt(s_length)
                center_x, center_y, theta, l_length, s_length = int(center_x), int(center_y), int(theta* 180 / np.pi)
                cv2.ellipse(bg, (center_x, center_y), (l_length, s_length), theta, 0, 360, (0,255,255), 1)

                cv2.imshow('polygon', bg)

            draw_on = False
```

**Q2&3 comment**

근사해를 찾는다는 것이 정해져있는 답이 있지 않다보니 이론을 이해하기 어려웠는데, 코드를 짜는데도 익숙치 않아 수식과 학우분이 올려두신 코드, 오픈소스 라이브러리를 참고하여 스스로 실행해보려 노력하였습니다. 배움을 토대로 많은 연습이 필요하겠다는 것을 다시 한 번 느낄 수 있었습니다.