2015005187 최정훈

5과시- 벡터의 선형독립과 기저벡터

2.3 Linear Independence

⇒ linear independent

$C_1V_1+C_2V_2+\cdots+C_nV_n=0$ ①

→ only $C_1=C_2=\cdots=C_n=0$

◦ Rank of A

= # of independent column vectors

= # of " row vectors

= # of pivots in G.E.

= Dim of C(A)

⊙ If G.E of A

generates m non-zero rows → m independent column vectors in A

⊙ Spanning

◦ all linear combinations of vectors $\{v_1,v_2,\cdots,v_n\}$ construct a vector space

$=\{v_1,v_2,\cdots,v_n\}$ span vector space

⊙ Basis (vectors)

⇒ # of minimum linearly independent vectors to span the vector space

⇒ linear combination is unique from basis

◦ Basis is not unique for a vector space

6과시- 내적공간과 정규직교기저

orthogonal basis(vectors) if V_i orthogonal

$v_1,v_2,\cdots,v_n$ $\|V_i\|=1$

$V_i^TV_j=0$

$X=\sum_{i=1}^{n}C_iV_i$ $[v_1v_2\cdots v_n]\begin{bmatrix}c_1\\c_2\\\vdots\\c_n\end{bmatrix}=[X]$

$C_i=V_i^TX$

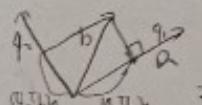
$=\frac{V_i^TX}{V_i^TV_i}$

- If given independent vectors $a_1,a_2,a_3\cdots$

→ find the orthonormal basis vectors

⇒ Gram-Schmidt orthogonalization

⊙ Gram-Schmidt Orthogonalization



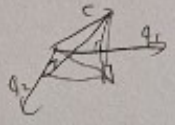
1) $a\to\frac{a}{\|a\|}=q_1$

2) project b onto q_1 $b-(q_1^Tb)q_1+q_1$

$b-(q_1^Tb)q_1$

$\frac{b-(q_1^Tb)q_1}{\|b-(q_1^Tb)q_1\|}=q_2$

$b-(q_1^Tb)q_1+(q_2^Tb)q_2$



$c-\underbrace{\left((q_1^Tc)q_1+(q_2^Tc)q_2\right)}_{\text{normalization}}\perp\begin{matrix}q_1\\q_2\end{matrix}$

q_3

$c=(q_1^Tc)q_1+(q_2^Tc)q_2+(q_3^Tc)q_3$

1) $q_1=\frac{a}{\|a\|}$ 2) $a_j-\sum_{i=1}^{j-1}(q_i^Ta_j)q_i=A_j$ 3) $\frac{A_j}{\|A_j\|}=q_j$ $\therefore a_j=\sum_{i=1}^{j}(q_i^Ta_j)q_i$

8과시 - 일반최소제곱법과 QR분할

◦ Let $q_1,q_2,\cdots,q_n$ be orthonormal

$q_i^Tq_j=\begin{cases}0 & i\neq j\\ 1 & i=j\end{cases}$

$Q=[q_1q_2\cdots q_n]$

$Q^TQ=\begin{bmatrix}-q_1^T-\\-q_2^T-\\\\-q_n^T-\end{bmatrix}[q_1q_2\cdots q_n]=I$

$\Rightarrow Q^T=Q^{-1}$ (Left-inverse)

◦ for $q_1,q_2,\cdots,q_n\in\mathbb{R}^n$ (square sys)

$\Rightarrow X=\sum_{i=1}^{n}C_iq_i$

$X=[q_1q_2\cdots q_n]\begin{bmatrix}c_1\\c_2\\\vdots\\c_n\end{bmatrix}$

$\begin{bmatrix}c_1\\c_2\\\vdots\\c_n\end{bmatrix}=Q^{-1}X=Q^TX=Q^T\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}$

$=\begin{bmatrix}-q_1^T-\\-q_2^T-\\\\-q_n^T-\end{bmatrix}X$ $C_i=\frac{q_i^TX}{q_i^Tq_i=1}$

⊙ Q examples

1) Rotation matrix

2) Permutation Matrix

◦ for $q_1,q_2,\cdots,q_n\in\mathbb{R}^m$ (Rectangular Sys)

$Q=[q_1q_2\cdots q_n]$ $m<n$

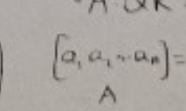
$Qx=b$

$x=(Q^TQ)^{-1}Q^Tb$

I

$[Q^T][Q]=[I]$

Q: Left-inverse for rectangular system

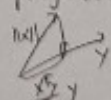


⊙ Q transformation preserves the length and angle.

$\Rightarrow\|x\|^2=x^Tx$ $\|Qx\|^2=x^TQ^TQx=x^Tx$

$\Rightarrow x^Ty$ $(Qx)^T(Qy)=x^TQ^TQy=x^Ty$

◦ projection reduces the length



$\|x^Ty\|\leq\|x\|\|y\|$ $\|Px\|\leq\|x\|$

◦ A=QR factorization

$\underset{A}{[a_1a_2\cdots a_n]}=\begin{bmatrix}(q_1^Ta_1)q_1 & (q_1^Ta_2)q_1 & \cdots & (q_1^Ta_n)q_1\\ & +(q_2^Ta_2)q_2 & \cdots & +(q_2^Ta_n)q_2\\ & & \vdots & \\ & & & +(q_n^Ta_n)q_n\end{bmatrix}$

$=[q_1q_2\cdots q_n]\begin{bmatrix}(q_1^Ta_1) & (q_1^Ta_2) & \cdots & (q_1^Ta_n)\\ 0 & (q_2^Ta_2) & \cdots & (q_2^Ta_n)\\ 0 & 0 & \cdots & \vdots\\ & & & (q_n^Ta_n)\end{bmatrix}$

$Ax=b$

$X=(A^TA)^{-1}A^Tb$

$=(R^TQ^TQR)^{-1}R^TQ^Tb$

$=(R^TR)^{-1}R^TQ^Tb$

행렬식 고유값과 고유벡터의 의미 대각화

$Ax = \lambda x$ → eigenvector

$\Rightarrow (A - \lambda I)X = 0$

$n \times n$

↓ scalar multiplication → eigenvalue

for non-zero $X \rightarrow N(A - \lambda I)$

$\det(A - \lambda I) = 0$ → singular

→ how to find null space(eigenvectors) → Row Reduced Echelon form!

◦ for triangular(or diagonal) matrix

→ eigenvalues = diagonal elements of A

$\det A = \prod_{i=1}^{n} \text{pivot}_i = \prod_{i=1}^{n} d_i = \prod_{i=1}^{n} \lambda_i$

◦ Trace of $A = \sum_{i=1}^{n} a_{ii} = (a_{11} + a_{22} + \cdots + a_{nn})$

5.2 Diagonalization of matrix

1) $A = LU$ 2) $A = QR$

3) $A = S \Lambda S^{-1}$ $S = [e_1 e_2 \cdots e_n]$ $\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \lambda_n \end{bmatrix}$

$Ae_i = \lambda_i e_i$

$[Ae_1 \; Ae_2 \cdots Ae_n] = [\lambda_1 e_1 \; \lambda_2 e_2 \cdots \lambda_n e_n]$

$\rightarrow A[e_1 e_2 \cdots e_n] = [e_1 e_2 \cdots e_n] \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \lambda_n \end{bmatrix}$

$AS = S\Lambda$

$\therefore A = S \Lambda S^{-1}$

Remark 1>

If $\lambda_1, \lambda_2, \cdots, \lambda_n$ are different

then, $e_1, e_2, \cdots, e_n$ are linearly independent!

Remark 2>

S is not unique

since $ke \rightarrow$ eigenvector

Remark 3>

The order of eigenvalues is same with that of eigenvectors

Remark 4>

Not all metrices have n linearly independent eigenvectors

$\rightarrow A = S \Lambda S^{-1}$ is not always established.

◦ Power

$A \rightarrow \lambda, e$

$A^* \rightarrow \lambda^*, e$

$Ae = \lambda e$

$A^2 e = \lambda Ae = \lambda^2 e$

$A^k = (S \Lambda S^{-1})^k$

$= S \Lambda^k S^{-1}$

↓

$\begin{bmatrix} \lambda_1^k & & \\ & \lambda_2^k & \\ & & \lambda_n^k \end{bmatrix}$

수치해석 HW#4

Least Square를 이용하여 2차곡선 fitting하기

Python 사용

컴퓨터소프트웨어학부 2015005187 최철훈

목차

1. 8개의 점을 모두 이용하여 2차 곡선 fitting하기

2. 6개의 점을 이용하여 2차 곡선 fitting하기

3. Compare

8개의 점을 모두 이용하기 Pseudo-inverse로 구하기

먼저, 주어진 8개의 점을 모두 이용하여 원래의 2차 곡선을 fitting하였다.

1. 8개의 점

```
D = np.array([[-2.9, 35.4], [-2.1, 19.7], [-0.9, 5.7], [1.1, 2.1],
              [0.1, 1.2], [1.9, 8.7], [3.1, 25.7], [4.0, 41.5]])
```

2. Ax=b에서 b행렬 만들기

```
# Make B
B = np.empty((0, 1), dtype=float)
for i in range(0, 8):
    B = np.append(B, [[D[i, 1]]], axis = 0)
```

3. Ax=b에서 A행렬 만들기

```
## Make 1 column of A
A1 = np.empty((0, 1), dtype=float)
for i in range(0, 8):
    A1 = np.append(A1, [[(D[i, 0])**2]], axis = 0)
## Make 2 column of A
A2 = np.empty((0, 1), dtype=float)
for i in range(0, 8):
    A2 = np.append(A2, [[D[i, 0]]], axis = 0)
## Make 3 column of A
A3 = np.ones((8, 1), dtype=float)
## Make A
A = np.hstack([A1, A2])
A = np.hstack([A, A3])
```

4. Pseudo-inverse구하기

```
# pseudo-inverse
At = np.transpose(A)
AtA = np.dot(At, A)
AtAI = np.linalg.inv(AtA)
PI = np.dot(AtAI, At)
```

5. X구하기

```
# X
X = np.dot(PI, B)
print(X)
```

6. 결과

```
[[ 3.16052477]
 [-2.36059821]
 [ 1.35828072]]
```

8개의 점을 모두 이용하기 Fitting함수로 구하기

앞서 구한 X가 맞는지 확인하기 위해 2차곡선을 fitting하는 함수를 이용하여 바로 구해보았다.

1. Fitting함수인 polyfit을 이용하여 X구하기

```
x = np.array([-2.9, -2.1, -0.9, 1.1, 0.1, 1.9, 3.1, 4.0])
y = np.array([35.4, 19.7, 5.7, 2.1, 1.2, 8.7, 25.7, 41.5])

fit = np.polyfit(x, y, 2)
print(fit)
```

2. 결과

```
[ 3.16052477 -2.36059821  1.35828072]
```

앞서 Pseudo-inverse를 이용하여 구한 X와 같은 값이 나오므로 Pseudo-inverse를 이용하여 구하는 과정에서 제대로 코딩하여 결과를 얻었다는 것을 알 수 있다.

6개의 점을 이용하여 fitting하기 과정

주어진 8개의 점 중 6개의 점을 랜덤으로 선택하여 2차 곡선을 fitting하였다.

1. 6개의 점 선택하기

```
# Randomly select 6 points
r = random.sample(range(0, 8), 6)
R = np.empty((0, 2), dtype=float)
for i in r:
    R = np.append(R, [D[i]], axis = 0)
print(R)
```

2. Ax=b에서 b행렬 만들기

```
# Make B
B = np.empty((0, 1), dtype=float)
for i in range(0, 6):
    B = np.append(B, [[R[i, 1]]], axis = 0)
```

3. Ax=b에서 A행렬 만들기

```
## Make 1 column of A
A1 = np.empty((0, 1), dtype=float)
for i in range(0, 6):
    A1 = np.append(A1, [[(R[i, 0])**2]], axis = 0)
## Make 2 column of A
A2 = np.empty((0, 1), dtype=float)
for i in range(0, 6):
    A2 = np.append(A2, [[R[i, 0]]], axis = 0)
## Make 3 column of A
A3 = np.ones((6, 1), dtype=float)
## Make A
A = np.hstack([A1, A2])
A = np.hstack([A, A3])
```

4. Pseudo-inverse구하기

```
# pseudo-inverse
At = np.transpose(A)
AtA = np.dot(At, A)
AtAI = np.linalg.inv(AtA)
PI = np.dot(AtAI, At)
```

5. X구하기

```
# X
X = np.dot(PI, B)
print(X)
```

6개의 점을 이용하여 fitting하기 결과

총 2번 시행하여 2개의 X를 도출하였다.

```
[[-2.1 19.7]
 [ 3.1 25.7]
 [ 1.1  2.1]
 [ 4.  41.5]
 [-0.9  5.7]
 [-2.9 35.4]]
[[ 3.18054193]
 [-2.39915013]
 [ 1.16085405]]
```

첫 번째 6개의 선택된 점과 그에 따른 결과

```
[[ 4.  41.5]
 [ 1.9  8.7]
 [ 1.1  2.1]
 [-2.1 19.7]
 [ 0.1  1.2]
 [-0.9  5.7]]
[[ 3.07895662]
 [-2.24365274]
 [ 1.3153594 ]]
```

두 번째 6개의 선택된 점과 그에 따른 결과

Compare 두 곡선의 비교

앞서 6개의 점으로 fitting한 두 결과에 대한 2차 곡선 그래프를 그려 비교해보았다.

```
[[-2.1 19.7]
 [ 3.1 25.7]
 [ 1.1  2.1]
 [ 4.  41.5]
 [-0.9  5.7]
 [-2.9 35.4]]
[[ 3.18054193]
 [-2.39915013]
 [ 1.16085405]]
```

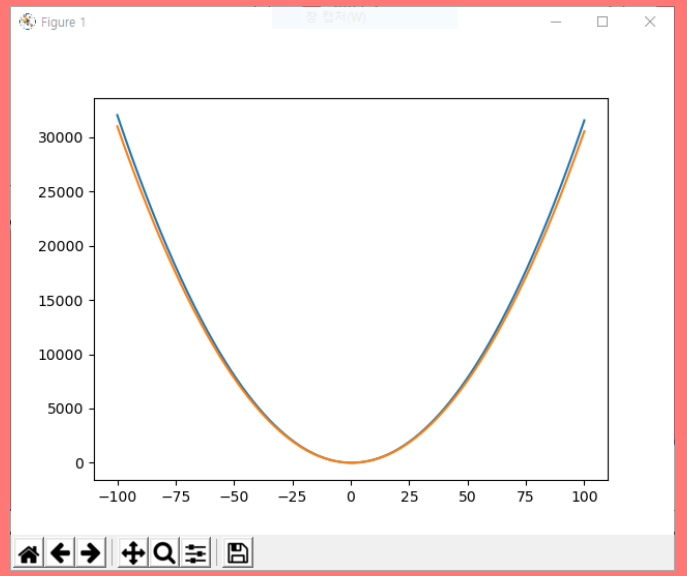
파랑곡선

```
[[ 4.  41.5]
 [ 1.9  8.7]
 [ 1.1  2.1]
 [-2.1 19.7]
 [ 0.1  1.2]
 [-0.9  5.7]]
[[ 3.07895662]
 [-2.24365274]
 [ 1.3153594 ]]
```

주황곡선

```
    if j == 1:
        z1 = X[0, 0]
        y1 = X[1, 0]
        x1 = X[2, 0]
    elif j == 2:
        z2 = X[0, 0]
        y2 = X[1, 0]
        x2 = X[2, 0]

k = np.arange(-100, 100, 0.01)
fit1 = z1*k**2 + y1*k + x1
plt.plot(k, fit1)
fit2 = z2*k**2 + y2*k + x2
plt.plot(k, fit2)
plt.show()
```



두 곡선의 비교 결과 큰 차이가 없음을 알 수 있다. 그나마 큰 차이가 있다면 극솟값에서 멀어질수록 조금씩 벌어진다는 것이다. 이외에도 여러 번 시도해 보았으나 매번 비슷하게 큰 차이가 없었지만 간혹 극솟값에서 멀어질수록 벌어지는 경우가 나왔다.

Compare 원래 곡선과의 비교

8개의 fitting한 그래프와 6개의 점으로fitting한 그래프를 비교해보았다.

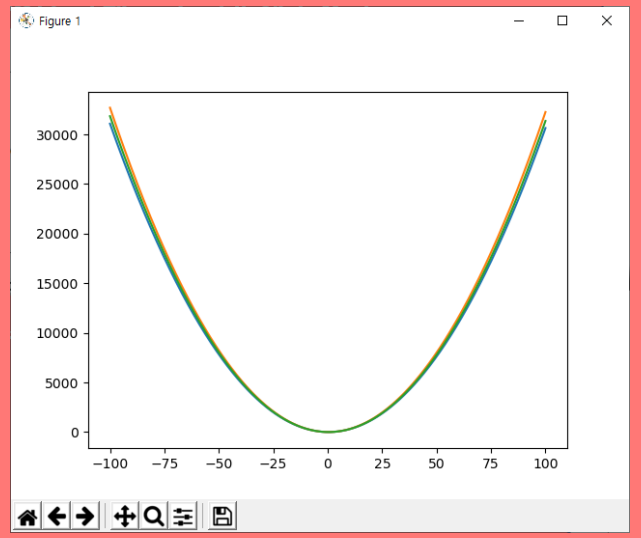
```
[[ 4.   41.5]
 [ 1.1   2.1]
 [ 0.1   1.2]
 [ 3.1  25.7]
 [ 1.9   8.7]
 [-2.1  19.7]]
[[ 3.086668  ]
 [-2.14233456]
 [ 1.44708834]]
```

파랑곡선

```
[[ 0.1   1.2]
 [ 1.1   2.1]
 [ 1.9   8.7]
 [-0.9   5.7]
 [ 3.1  25.7]
 [-2.1  19.7]]
[[ 3.24789136]
 [-2.1379364 ]
 [ 1.01464205]]
```

주황곡선

```
k = np.arange(-100, 100, 0.01)
fit1 = z1*k**2 + y1*k + x1
plt.plot(k, fit1)
fit2 = z2*k**2 + y2*k + x2
plt.plot(k, fit2)
fit3 = 3.16052477*k**2 - 2.36059821*k + 1.35828072
plt.plot(k, fit3)
plt.show()
```



초록색이 8개의 점으로 fitting한 원래의 2차 곡선이다. 두 곡선과 원래 곡선의 비교 결과 앞선 6개의 점으로 fitting한 두 곡선과의 차이와 비슷하게 큰 차이가 없음을 알 수 있다. 그나마 큰 차이가 있다면 극솟값에서 멀어질수록 조금씩 벌어진다는 것이다. 이외에도 여러 번 시도해 보았으나 매번 비슷하게 큰 차이가 없었다. 이를 통해 점의 개수가 많을수록 더 정확한 곡선을 fitting할 수 있다는 사실을 알 수 있다.

마무리

감사합니다.

https://github.com/cheol-hoon/Numerical_Analysis