REPORT

HANYANG UNIVERSITY

수치해석

[HW#13]

과	목	수치해석
담 당 교 수		이상화
학	과	컴퓨터소프트웨어학부
학	년	3학년
학	번	2018063218
이	름	최성진
제 출	일	2020.11.25

- RANSAC방법을 이용하여 최소제곱해를 구하여라

8

HW#13 (1/2)

- Line fitting using RANSAC and least square
  - 1. Randomly select 6 samples from 12 points
  - 2. Find the fitted line using 6 samples and least square
  - 3. Calculate the error
  - 4. Compare the current error with previous error
    - If current error is smaller than the previous one, save the current solution and remove the previous one
  - 5. Repeat 1~4 until your criterion
- Compare the solution of RANSAC with the solution of whole 10 samples

- 우선, 12개의 point들을 평균이 0, 표준편차가 $\sqrt{2}$인 정규분포로 생성하였습니다.

```
np.random.seed(10) # 특정한 평균과, 분산을 가진 정규분포로 노이즈 생성
mu, sigma = 0, np.sqrt(2)
noise = np.random.normal(mu,sigma,size=12)
```

```
while i <924: # 경우의 수는 총12C6 = 924
np.random.seed(i)

x = np.arange(-5, 7)
pick_x = np.random.choice(x, 6, replace=False) # random으로x를 추출
y = 2 * pick_x - 1
pick_y = 2 * pick_x - 1 + noise[pick_x + 5]

find = np.sum(np.square(pick_y - y))

if find < err: # 최소의err를 가지는 샘플들을 찾기
err = find
p = 0
q = 0
count = i
for p in range(0, 6):
for q in range(0, 2):
if q == 0:
RANSAC[p, q] = pick_x[p]
else:
RANSAC[p,q] = pick_y[p]
i+=1
```

이후, 최소의 error를 가지는 샘플들을 찾아 행렬에 집어넣었습니다.

```
def matrix(M): # x, 1로 구성된A행렬 만들기(6개의 샘플)
M1 = np.zeros((6,2))
i = range(6)
j = range(2)
for i in range(0,6) :
for j in range(0,2) :
if j == 0:
M1[i,j] = M[i,0]
if j == 1:
M1[i, j] = 1

return M1
```

$_{12}C_6$ = 924로 총 924가지의 경우의 수가 있었는데, count를 통해 몇 번째의 경우의 수가 가장 error가 적은지 찾아보았습니다.

```
587 번째에 error가 최소인 해를 구했습니다.
```

주어진 최소의 error를 가진 행렬로 예전에 배운 psuedo-inverse를 통하여 최소제곱해를 구했습니다.

```
############## < Psuedo-inverse 를 이용한 최소제곱해> #############

Amat = matrix(RANSAC) # A
AmatT = np.transpose(Amat) # At
yMat = np.array([RANSAC[:,1]]) # Y
yMatT = np.transpose(yMat) # Yt

answer = np.dot(np.dot(lin.inv(np.dot(AmatT,Amat)),AmatT),yMatT) # (inv(At X A)) X At X b (
Psuedo-inverse )
```

그 결과는 다음과 같습니다.

```
################## <RANSAC을 이용한 최소제곱해> ##################
x의 계수 :  [2.02832071] 상수계수 :  [-0.91324773]
```

그 이후 모든 샘플들을 이용하여 최소제곱해를 구해보았습니다.

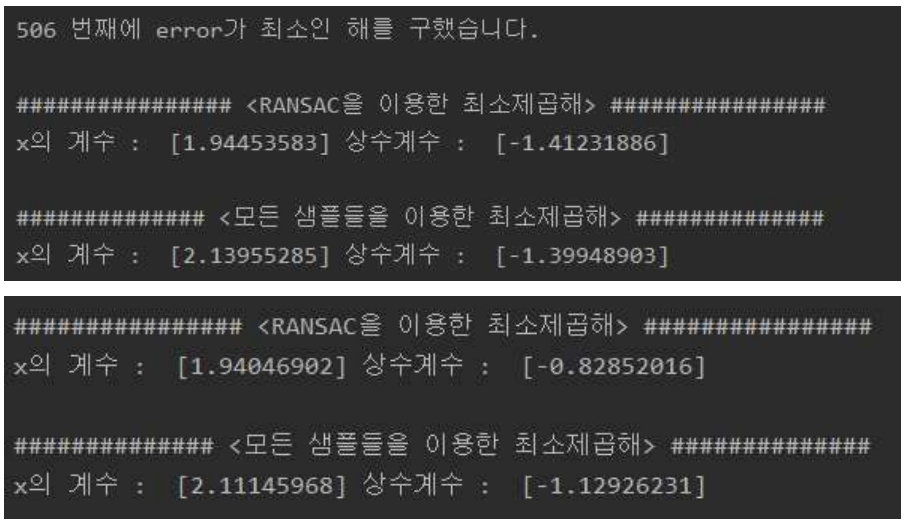
```
############## <모든 샘플들을 이용한 최소제곱해> ##############
ALL = np.zeros((12,2))
for p in range(0, 12):
for q in range(0, 2):
if q == 0:
ALL[p, q] = all_x[p]
else:
ALL[p, q] = all_y[p]
Amat = matrix_all(ALL) # A
AmatT = np.transpose(Amat) # At
yMat = np.array([ALL[:,1]]) # Y
yMatT = np.transpose(yMat) # Yt

answer = np.dot(np.dot(lin.inv(np.dot(AmatT,Amat)),AmatT),yMatT) # (inv(At X A)) X At X b (
Psuedo-inverse )
#
print("############## <모든 샘플들을 이용한 최소제곱해> ##############")
print("x의 계수: ",answer[0],"상수계수: ",answer[1])
```

결과는 다음과 같습니다.

```
############## <모든 샘플들을 이용한 최소제곱해> ##############
x의 계수 :  [2.02547511] 상수계수 :  [-0.74944085]
```

모든 샘플들을 이용한 최소제곱해는 $y \approx 2.03x - 0.75$로 매칭되었지만
RANSAC방법을 이용하여 얻은 최소제곱해는 $y \approx 2.03x - 0.91$로 원하는 직선에 더 근접하게 매칭되었습니다.

RANSAC이 가장 근접한 최소제곱해를 얻는것인지 궁금하여 몇 번의 실험을 더 해보았습니다.



위와 같이 항상 근접하지 않음을 확인할 수 있었습니다. 이번 실험의 결과로 ransac방법도, 모든 sample을 이용한 방법도 어느 것도 항상 옳은 정답을 구하는 것이 아니고 best solution을 구하는 것이 아닌 optimal한 solution을 구하도록 여러 방법을 시도해보는 것이 좋은 접근이라는 것을 깨달았습니다.