수치해석 HW #13

Numerical analysis

JaeHoon KANG
강 재 훈

HW #13

CONTENTS

HW #13

01 코드 설명

사용된 library

```
import itertools
import numpy as np
```

Itertools = 12개의 점 중 6개를 선택한 모든 경우의 수를 만들기 위해 사용하였습니다.

Numpy = 가우시안 noise를 발생시키기 위해 사용하였습니다.

이후 pseudo inverse를 구해 line을 생성하기 위해 사용했습니다.

HW #13

01 코드 설명

```python
def caly(x):
    ret = 2*x-1
    return ret

def leastsquare(points):
    global max_error, min_error
    error = 0
    for i in range(6):
        error += (points[i][1] - caly(points[i][0])) ** 2
    max_error = max(max_error, error)
    min_error = min(min_error, error)

    print(points)
    print(error,"\n")
```

코드 설명

caly = 인자로 받은 x값을 이용해 2x-1 값을 계산해 줍니다.

leastsquare = 인자로 받은 6개의 point를 이용해 least square를 실행한 후 사용된 point들과 error를 출력합니다.

HW #13

01 코드 설명

코드 설명

```
point_num = 12
x = [i for i in range(-5,7,1)]
noise = np.random.normal(0,2**0.5,12)
real_y = [ caly(i) for i in x ]

noise_y = []
for i in range(point_num):
    noise_y.append(real_y[i] + noise[i])
```

x = [-5, 6]의 정수값이 저장되어 있습니다.

noise = 평균 = 0, 분산 = 2 (표준편차 = 2**0.5) 인 가우시안 분포에서 12개의 noise를 발생시켜 저장합니다.

real_y = 2x-1에 해당하는 값이 12개 들어있습니다.

noise_y = real_y에 noise를 추가한 값 12개가 들어있습니다.

HW #13

01 코드 설명

```
noise_point = []

for i in range(point_num):
    a,b = x[i], noise_y[i]
    noise_point.append([a,b])

com_noise = list(itertools.combinations(noise_point,6))
```

코드 설명

noise_point = [x, 2x-1+noise] 형태의 좌표 12개가 들어 있습니다.

com_noise = noise_point에서 6개를 골라 만든 모든 경우의 조합이 들어 있습니다.

HW #13

01 코드 설명

```
x = np.array([-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6])
noise_y =  np.array([-9.385465225173377, -12.414363749437356, -6.6095
min_points = [[-3, -6.609567429268855], [-2, -5.639109976112357], [0,
min_x = []
min_y = []
for a,b in min_points :
    min_x.append(a)
    min_y.append(b)

min_x = np.array(min_x)
min_y = np.array(min_y)

A = np.vstack([x, np.ones(len(x))]).T
A2 = np.vstack([min_x, np.ones(len(min_x))]).T

m1, c1 = np.linalg.lstsq(A, noise_y, rcond=None)[0]
m2, c2 = np.linalg.lstsq(A2, min_y, rcond=None)[0]

print("Using 12 sample y = %fx +(%f)" %(m1,c1))
print("Using 6 sample y = %fx +(%f)" %(m2,c2))
```

코드 설명

첫번째 파일의 실행결과에서 나온

최적의 solution을 갖는 6개의 sample 과
12개의 sample을 가지고

각각 line fitting을 한 뒤
2개의 line을 비교합니다.

HW #13

02 실행 결과

```
x :  [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6]
real_y :  [-11, -9, -7, -5, -3, -1, 1, 3, 5, 7, 9, 11]
noise :  [ 1.61453477 -3.41436375  0.39043257 -0.63910998 -2.86172075 -0.55255486
 -1.7271663   2.49692714 -1.26458224 -0.03116884  0.60388093 -0.22883848]
noise_y :  [-9.385465225173377, -12.414363749437356, -6.609567429268855, -5.639109976112357, -5.861720754283326, -1.5525548588979334,
-0.7271662969030217, 5.496927140772376, 3.7354177602376195, 6.968831157394976, 9.603880931397494, 10.77116152245486]
```

x = [-5 ~ 6] 인 정수값

real_y = 2x-1의 값

noise = N(0,2)를 이용해 발생시킨 random 한 값

noise_y = 2x-1 + noise의 값

HW #13

02 실행 결과

```
([-5, -9.385465225173377], [-4, -12.414363749437356], [-3, -6.609567429268855], [-2, -5.639109976112357], [-1, -5.861720754283326], [0, -1.5525548588979334])
23.320264054038088
now min_error :  23.320264054038088
```

least square의 결과 중 예시

첫번째 줄 – 선택된 6개의 point

두번째 줄 – 선택된 6개의 point로 least square 실행 후 나온 error

세번재 줄 – iteration 중 현재 가장 작은 min 값

HW #13

02 실행 결과

```
MAX :  33.270964832698155
```

```
([-5, -9.385465225173377], [-4, -12.414363749437356], [-1, -5.861720754283326], [1, -0.7271662969030217], [2, 5.496927140772376], [3, 3.7354177602376195])
33.270964832698155
now min_error :  14.775695362530337
```

least square의 결과 중 가장 큰 error 값과
해당 error 가 검출될때 사용된 점들입니다.

HW #13

02 실행 결과

```
MIN :  1.2842267508058025
```

```
([-3, -6.609567429268855], [-2, -5.639109976112357], [0, -1.5525548588979334], [4, 6.968831157394976], [5, 9.603880931397494], [6, 10.77116152245486])
1.2842267508058025
now min_error :  1.2842267508058025
```

least square의 결과 중 가장 작은 error 값과
해당 error 가 검출될때 사용된 점들입니다.

HW #13

02 실행 결과

```
Using 12 sample y = 2.086398x +(-1.511010)
Using 6 sample y = 2.032603x +(-1.130565)
```

첫번째 줄 – 12개의 sample을 가지고 line fitting

두번째 줄 – 최적의 error를 가지는 6개의 sample로 line fitting

HW #13

03 결과 분석

```
3693    ([1, -0.7271662969030217], [2, 5.496927140772376], [3, 3.7354177602376195
3694    11.23492752946587
3695    now min_error :  1.2842267508058025
3696
```

```
3069    ([-3, -6.609567429268855], [-2, -5.63910997611235
3070    2.8824234951789043
3071    now min_error :  2.5187228126227357
3072
3073    ([-3, -6.609567429268855], [-2, -5.63910997611235
3074    1.2842267508058025
3075    now min_error :  1.2842267508058025
3076
3077    ([-3, -6.609567429268855], [-2, -5.63910997611235
3078    11.378787455209254
3079    now min_error :  1.2842267508058025
3080
3081    ([-3, -6.609567429268855], [-2, -5.63910997611235
```

현재 가장 작은 error 가 검출 되는 iteration은

총 924 (12C9, 1개의 case가 4줄이니 3696/4 = 924)의 iteration 중

769번째 iteration (3076/4 = 769) 입니다.

따라서 769번까지 iteration을 반복하게 된다면

현재의 noise가 낀 값을 이용해 RANSAC을 통한 최적화 값을 구할 수 있습니다.

HW #13

03 결과 분석

```
397   ([-5, -9.385465225173377], [-4, -12.4143637494
398   24.462649327530418
399   now min_error :  14.775695362530337
400
```

```
1997  ([-4, -12.414363749437356], [-3, -6.609
1998  21.437499027558736
1999  now min_error :  3.48248772836392
2000
```

총 924 (12C9, 1개의 case가 4줄이니 3696/4 = 924)의 iteration 중
100번만 iteration (400/4 = 100)할 경우
solution으로 14.7756.... 을 얻게 되고
500번만 iteration (2000/4 = 500)할 경우
solution으로 3.4824... 을 얻게 됩니다.
또한 가장 큰 error 값과 가장 작은 error 값이
크게 차이나는 경우가 있기 때문에
어떤 샘플이 몇번째 iteration에 뽑히느냐 와
iteration을 몇번 하냐에 따라
RANSAC을 이용해 구하는 solution이 달라지게 됩니다.

HW #13

03 결과 분석

```
Using 12 sample y = 2.086398x +(-1.511010)
Using 6 sample y = 2.032603x +(-1.130565)
```

결국 12개의 sample을 모두 사용하여 fitting 한 line과
최적의 error 값을 가지는 6개의 sample을 사용하여 fitting한 line을 비교해보면

최적의 error 값을 가지는 6개의 sample을 사용하여 fitting 한 line이
우리가 원하는 2x-1 과 더 유사한 line이라는 것을 확인할 수 있습니다.

RANSAC 방법을 잘만 이용한다면
연산에 사용되는 sample 수를 줄일 수 있을 뿐만 아니라
noise를 줄일 수 있어 좀더 최적화 된 solution을 구할 수 있습니다.

감사합니다

THANK YOU

JaeHoon KANG
강 재 훈