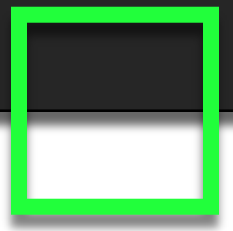
#수치 해석
#과제 13

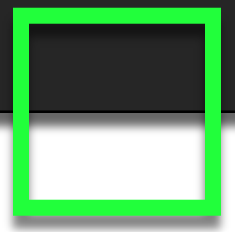
RANSAC

컴퓨터소프트웨어학부
2018008395 박정호



#1

구현



설명에 앞서…

이번 과제는 주어진 직선 위의 12개의 점에 대해서 노이즈를 각 점에 대해 더해주고, 이 점들 중 무작위의 표본을 선정해서 line fitting을 하는 것이다. line fitting 에는 기존 4주차 과제에 사용된 pseudo-inverse 를 사용한 방식을 사용했다.

사용한 라이브러리는 기존 과제들과 동일하게 numpy 를 사용했고, 행렬 연산이나, gaussian noise 추가를 위해서만 사용했다.

모델링 – line fitting

구현 자체는 기존 4주차 과제와 같다. 따라서 구현 자체에 대한 설명은 생략한다.
주어진 점을 각각 $P_1 \sim P_{12}$ 이라고 하고, 이 점들의 x, y좌표를 $x_1 \sim x_8$, $y_{12} \sim y_{12}$이라고 하게 되면, least square 를 구하기 위해 우리는 아래와 같은 연립방정식을 풀어야 한다.

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_N & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

즉, 이 식의 양변을 차이가 최소인 a, b 순서쌍을 구해야 하는 것이 목적인 것이다.
앞에서도 언급했듯이 pseudo-inverse 를 이용한 방식을 사용했고, 이 과제의 핵심은 RANSAC과 전체 샘플에 대한 line fitting 의 차이를 보는 것이므로 자세한 사항은 넘어가기로 한다.

모델링 – 샘플의 생성과 표본 설정

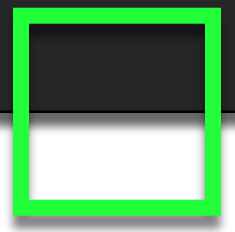
우선 주어진 직선 위의 점 12개를 구하고, 각 점의 y 좌표에 가우시안 분포에 따라 구해진 무작위의 노이즈를 더했다. 사용한 함수는 numpy.random.normal 이고, 평균과 분산 값은 과제에서 명세된 대로 0, 1을 사용했다. 노이즈는 프로그램을 실행할 때마다 매번 바뀌었으며, 따라서 매번 구해지는 직선의 방정식도, 오차의 정도도 매번 다르게 출력되었다.

랜덤하게 6개의 점을 구하는 것은 기존 4주차 과제에서도 수행했기에 이 구현도 기존 과제에서 그대로 가져왔다. 따라서 자세한 설명은 생략한다.

표본으로 사용할 6개의 점을 선정한 이후에는 [x, 1] 을 행으로 갖는 6 * 2 행렬을 만들고, y를 행으로 갖는 6 * 1 행렬을 만들었다. 즉 앞의 연립방정식에서 첫번째 행렬과 3번째 행렬을 만든 것이다.

#2

결과와 비교



몇 차례의 실험 (4회 반복)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-11.32738191  -9.41147045  -9.30549405  -4.4725202   -2.04175103
  -1.85070043  -0.64418813   0.82593611   5.44613305   8.55279379
   9.87802059  11.11453449]
Start RANSAC Process...
Whole Sample Test Result        : ( 2.114956 )x+( -1.327152 )
RANSAC Test Result              : ( 1.990722 )x+( -1.206319 )
Whole Sample Test Error         : 1.297808
RANSAC Test Error               : 1.485176
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-12.74248355  -8.72153     -6.37157721  -5.13041328  -3.30344064
  -0.64453858  -0.20847629   3.57620807   6.22271208   6.26081058
   8.45825011  10.26577897]
Start RANSAC Process...
Whole Sample Test Result        : ( 2.006910 )x+( -1.198347 )
RANSAC Test Result              : ( 1.950491 )x+( -1.111349 )
Whole Sample Test Error         : 0.662330
RANSAC Test Error               : 0.703718
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-11.3238797  -11.18456283  -7.12327055  -3.61752061  -1.50578029
  -2.13818256   0.97363239   3.84209335   5.63906983   7.48831447
   8.3388418   10.77949102]
Start RANSAC Process...
Whole Sample Test Result        : ( 2.050934 )x+( -1.011447 )
RANSAC Test Result              : ( 2.073320 )x+( -1.335384 )
Whole Sample Test Error         : 0.983408
RANSAC Test Error               : 1.087189
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-11.06000399  -8.14643316  -6.76036409  -6.69619713  -2.4831269
  -1.72886366   0.90926809   2.88122561   4.72114235   5.40887568
   9.51703513  10.12380182]
Start RANSAC Process...
Whole Sample Test Result        : ( 1.933551 )x+( -1.242912 )
RANSAC Test Result              : ( 1.933801 )x+( -1.301667 )
Whole Sample Test Error         : 0.548791
RANSAC Test Error               : 0.552229
```

몇 차례의 실험 (10회 반복)

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-10.44613908  -9.17809732  -6.06523518  -4.8430839   -3.60646021
  -0.99221253   0.88439853   2.02317171   3.43520739   6.49754263
   8.46721293  10.50057373]
Start RANSAC Process...
Whole Sample Test Result        : ( 1.878734 )x+( -1.216294 )
RANSAC Test Result              : ( 1.879223 )x+( -1.298010 )
Whole Sample Test Error         : 0.231959
RANSAC Test Error               : 0.238599
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-11.78441988  -7.7258899   -6.55297602  -6.33821783  -1.88982394
  -1.18164941   1.68898403   5.00917334   4.54373712   7.78928528
   7.48147433  12.4792861 ]
Start RANSAC Process...
Whole Sample Test Result        : ( 2.035456 )x+( -0.724481 )
RANSAC Test Result              : ( 2.050261 )x+( -0.946651 )
Whole Sample Test Error         : 1.176663
RANSAC Test Error               : 1.225400
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-11.54291329  -9.02430119  -7.66246657  -3.89058358  -4.68377005
  -1.87949297   0.79585305   4.30716176   3.84127296   7.78812519
  10.85823587  11.57343101]
Start RANSAC Process...
Whole Sample Test Result        : ( 2.131763 )x+( -1.025835 )
RANSAC Test Result              : ( 2.136872 )x+( -0.983835 )
Whole Sample Test Error         : 0.880639
RANSAC Test Error               : 0.882935
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\Users\pch68\HYU_MAT3008\HW13> python .\RANSAC_Line_Fitting.py
Construct Sample with Gaussian Noise
Construct Matrix with Whole Sample...
X : [-5. -4. -3. -2. -1.  0.  1.  2.  3.  4.  5.  6.]
Y : [-12.42645693  -8.65524677  -7.78717324  -5.13615844  -3.76687896
  -0.50641966   1.21313061   3.31048812   5.57840465   7.0805225
  10.35177528  10.8296852 ]
Start RANSAC Process...
Whole Sample Test Result        : ( 2.124053 )x+( -1.054887 )
RANSAC Test Result              : ( 2.117560 )x+( -1.002221 )
Whole Sample Test Error         : 0.313455
RANSAC Test Error               : 0.316399
```

오차의 계산

오차를 구한 방식은 다음과 같다.

구해진 직선의 방정식에 대해서 샘플의 점 12개의 x 좌표를 대입하고, 결과와 y 좌표를 비교했다. 이 차이의 제곱을 구한 뒤, 이 차이 12개의 평균을 구해서 오차를 구했다.

RANSAC에서는 12개 중 6개만 사용해서 직선의 방정식을 구했지만, 이 직선의 방정식이 전체 샘플에 대해서 얼마나 오차가 나오는지 확인하기 위해 RANSAC의 결과물과 전체 샘플의 결과물 모두에 대해서 같은 방식으로 오차를 계산했다.

결과 분석

우선 크게 두 번 실험을 진행했다. 무작위 표본을 뽑는 횟수를 달리해서 진행했는데, 꽤 유의미한 결과가 나왔다.
우선 실험 결과로 알 수 있는 것은 크게 두 가지였다.

1. RANSAC으로 구한 오차는 전체 샘플에서 구한 오차보다 무조건 크다.
   그 차이가 아주 작아질 수는 있으나, line fitting 의 결과물에 반영되는 데이터의 수가 반감되었기 때문에 결국은 오차가 더 커지는 것 같다.

2. 테스트 횟수를 늘리면 대체로 오차 간의 차이가 줄어든다.
   12개 중 6개의 샘플을 고르는 경우의 수는 꽤 크기 때문에, 자칫하면 outliar 만 고르게 되어서 오차가 상당히 크게 나올 수 있었다. 이를 보완한 것이 RANSAC이긴 하나, 4번의 반복만으로 최적의 표본을 찾는 것은 상당히 어려운 듯 했다. 10번의 반복으로 실험한 결과 거의 전체 샘플의 오차와 비슷할 정도로 최적해가 구해지는 것을 알 수 있었다.

세부적인 고찰

오차가 더 커지는 이유에 대해서 고민해보았다.

아무리 많이 RANSAC을 반복한다고 해도, 결국 결과에 반영되는 데이터는 딱 6개의 점일 뿐이다. 이 점들만으로 fitting 한 직선의 방정식이 다른 6개의 점들과도 가까우리란 보장이 없는 것이다. 그러나 12개의 점을 가지고 line fitting 을 하게 될 경우 오차 12개를 모두 고려해서 구해지는 솔루션이기 때문에 전체 오차가 줄어들 수밖에 없는 것이다.

다시 말해 전체 샘플로 구한 결과는 전체 오차가 최소가 되도록 한 것이기 때문에 아무리 표본을 뽑는다고 해도 이 최소 오차보다 작아질 수는 없다는 것이다.

세부적인 고찰

반복 횟수와 오차의 차이에 대해서 고민해보았다.

앞에서 언급했듯이 표본을 통해서 구해진 오차는 어쨌든 전체 데이터에 대한 오차보다는 클 수밖에 없다. 여기서 중요한 것은 이 오차 간의 차이이다. 오차의 값 자체는 결국 처음 샘플을 생성할 때 더해진 노이즈에 dominate 되기 때문이다. 표본을 얼마나 잘 뽑았는지에 대한 근거는 오차의 절대값이 아니라, 표본 오차와 전체 오차의 차이이다.

12개 중 6개의 점을 고르는 경우의 수는 924이다. 이 많은 경우의 수 중, 어떤 경우는 전체 데이터에서 구해진 직선과 거의 같은 직선을 fitting 할 수 있을 것이다. 그리고 이 경우를 뽑아 내기 위해서는 최대한 많이 표본을 뽑아야 한다.

따라서 표본을 뽑는 횟수가 많으면 많을 수록 전체 데이터와 비슷한 직선을 fitting 하고, 오차 또한 작아지게 되는 것이다.