HW13

2016024766 | 김서현

기본 설계

- 우선 주어진 식인 y = 2x - 1에 N(0, 2)를 따르는 랜덤한 noise 12개를 더해서 샘플들을 만들었습니다.
  - N(0, 2)의 Gaussian noise를 더하기 위해서 numpy의 random.randn()함수를 사용했고, 여기에 $\sqrt{2}$를 곱해서 랜덤한 noise 12개를 뽑았습니다.
- 12개의 샘플 중 6개씩 뽑아 RANSAC을 해서 만든 직선과, 12개의 샘플들을 모두 이용해서 least square를 통해 만든 직선을 원래의 식인 y = 2x - 1에 비교하는 방식을 사용했습니다.

RANSAC 구현 방법

1. 12개의 샘플들 중 6개를 뽑아서 Least square를 통해 line fitting을 하였습니다.
   - 랜덤하게 6개를 뽑을 때에는 random.sample() 함수를 이용했습니다.
2. Least square를 통해 구한 직선과 12개 샘플들의 y축방향 거리를 각각 측정합니다.
3. 계산된 12개의 거리 중 1 미만인 것들의 개수를 구합니다.
4. 만일 이전에 구한 개수보다 현재 계산된 개수가 많으면 개수를 업데이트하고 직선의 기울기와 y절편 값을 저장합니다.
5. 만일 이전에 구한 개수와 현재 계산된 개수가 동일하다면 12개 샘플들의 y좌표 벡터와, 계산한 직선상의 12개 y좌표 벡터의 유클리드 거리를 계산합니다.
6. 만일 이전에 구한 유클리드 거리보다 현재 계산된 직선의 유클리드 거리가 더 작다면 유클리드 거리를 업데이트하고 직선의 기울기와 y절편 값을 저장합니다.

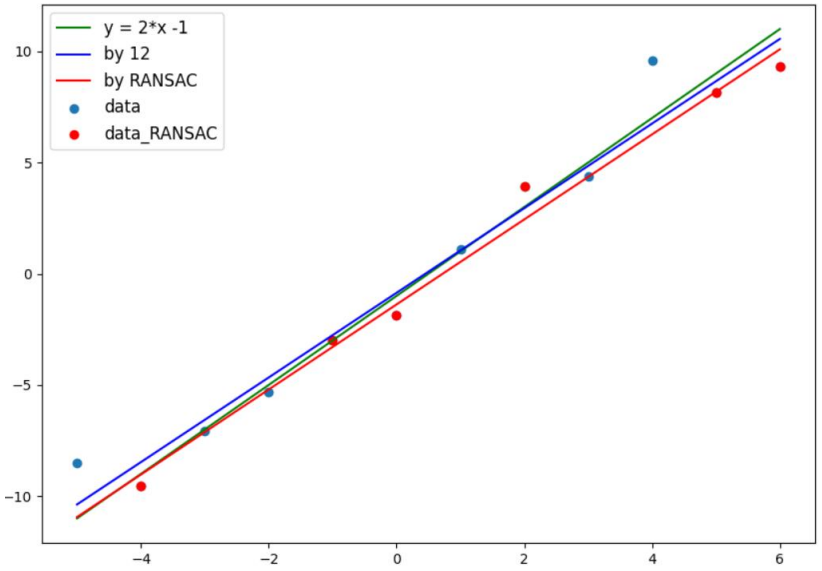
RANSAC 구현 방법

- 이러한 과정을 거리가 1 미만인 샘플 개수가 9개 이상이 될 때까지 반복합니다.
  - 9개 이상이 나오지 않았더라도 반복 횟수가 10000회를 넘어가면 종료합니다.
- 업데이트 규칙을 이렇게 만든 이유는 우선 최대한 많은 개수의 샘플들의 흐름을 따라가야 하고, 만일 같은 수의 샘플들의 흐름을 따라간다면 샘플들과의 전체적인 유사도가 큰 직선을 선택하는 것이 맞다고 생각했기 때문입니다.
  - Outlier에 영향을 최대한 안 받으려면, 최대한 많은 개수의 샘플들의 흐름을 따라가는게 가장 우선순위가 높아야 한다고 판단했습니다.

테스트 설계

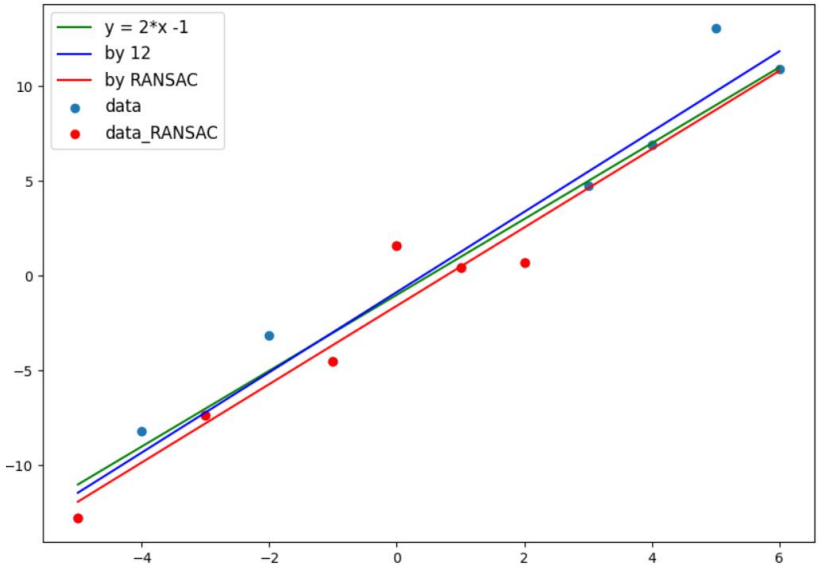
- RANSAC으로 뽑은 직선과, 12개 샘플들을 가지고 least square를 해서 만든 직선을 서로 비교해보고, $y = 2x - 1$ 직선과도 각각 비교해보는 테스트를 진행했습니다.
- 직선의 유사도는 유클리드 거리를 이용해서 계산했습니다.
  - 유클리드 거리가 작을수록 유사도가 큰 것으로 정의했습니다.
- 테스트는 총 10회 진행하였습니다.

테스트 결과 #1, #2 (**A**: y = 2x – 1, **B**: RANSAC, **C**: 12 samples)



RANSAC: y = 1.91x – 1.38
12 samples: y = 1.90x – 0.86

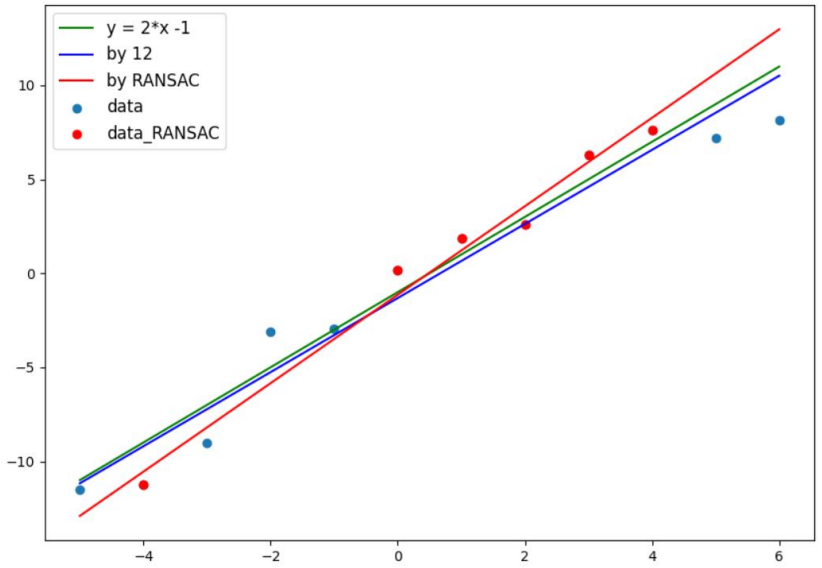
<각 벡터끼리의 유클리드 거리>
- A - B: 1.815
- A - C: 1.206
- B - C: 1.798



RANSAC: y = 2.07x – 1.58
12 samples: y = 2.12x – 0.85

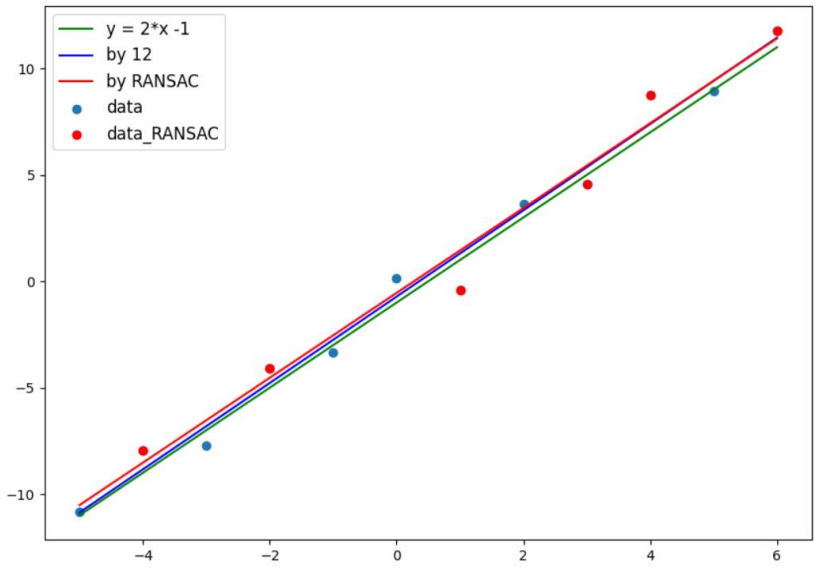
<각 벡터끼리의 유클리드 거리>
- A - B: 2.047
- A - C: 1.562
- B - C: 2.666

테스트 결과 #3, #4 (**A**: y = 2x – 1, **B**: RANSAC, **C**: 12 samples)



RANSAC: y = 2.35x – 1.14
12 samples: y = 1.97x – 1.31

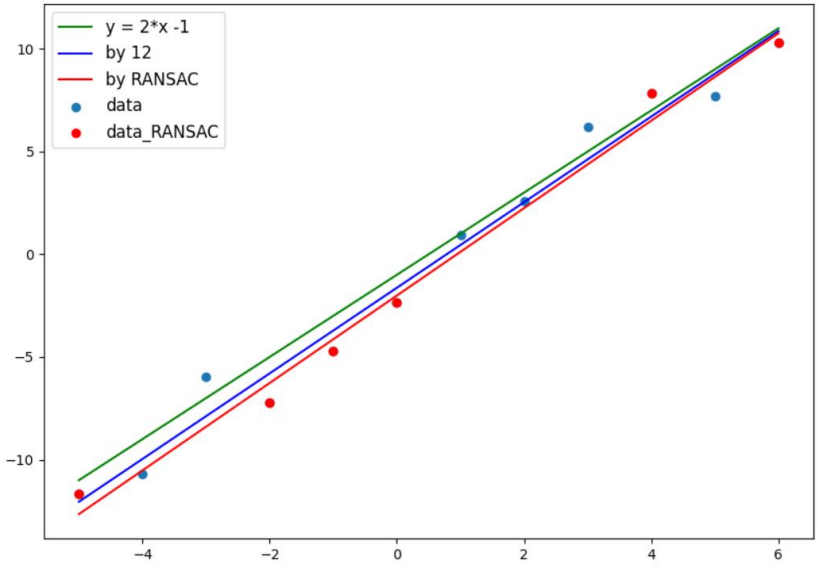
<각 벡터끼리의 유클리드 거리>
- A - B: 4.227
- A - C: 1.191
- B - C: 4.743



RANSAC: y = 1.99x – 0.55
12 samples: y = 2.03x – 0.72

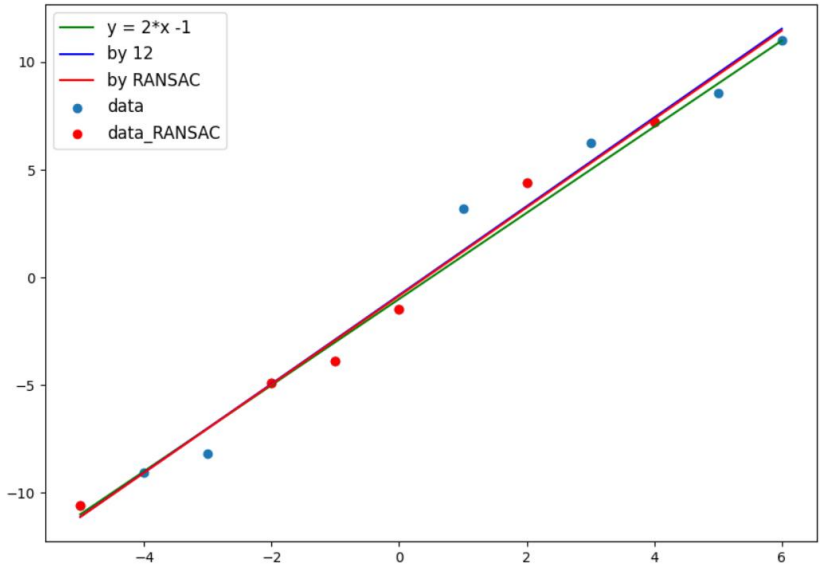
<각 벡터끼리의 유클리드 거리>
- A - B: 1.561
- A - C: 1.070
- B - C: 0.685

테스트 결과 #5, #6 (**A**: y = 2x – 1, **B**: RANSAC, **C**: 12 samples)



RANSAC: y = 2.13x – 2.01
12 samples: y = 2.08x – 1.63

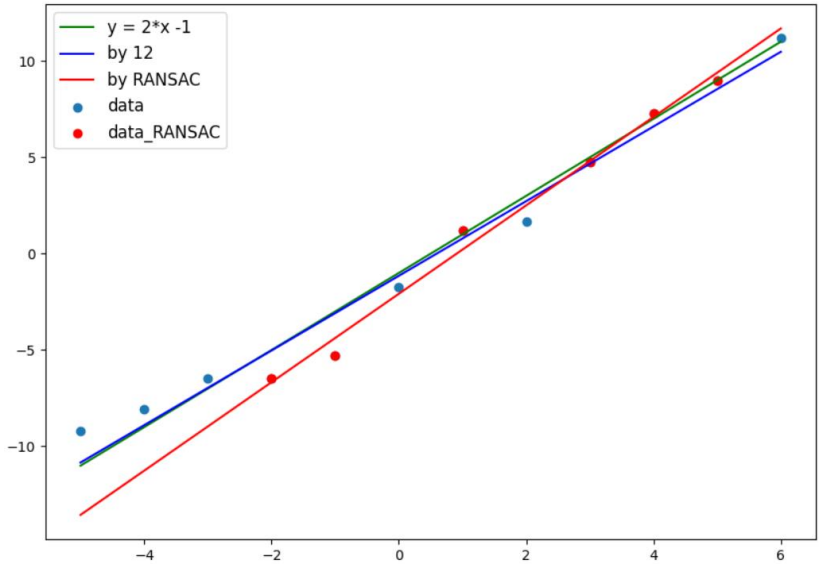
<각 벡터끼리의 유클리드 거리>
- A - B: 3.620
- A - C: 2.276
- B - C: 1.344



RANSAC: y = 2.05x – 0.85
12 samples: y = 2.06x – 0.81

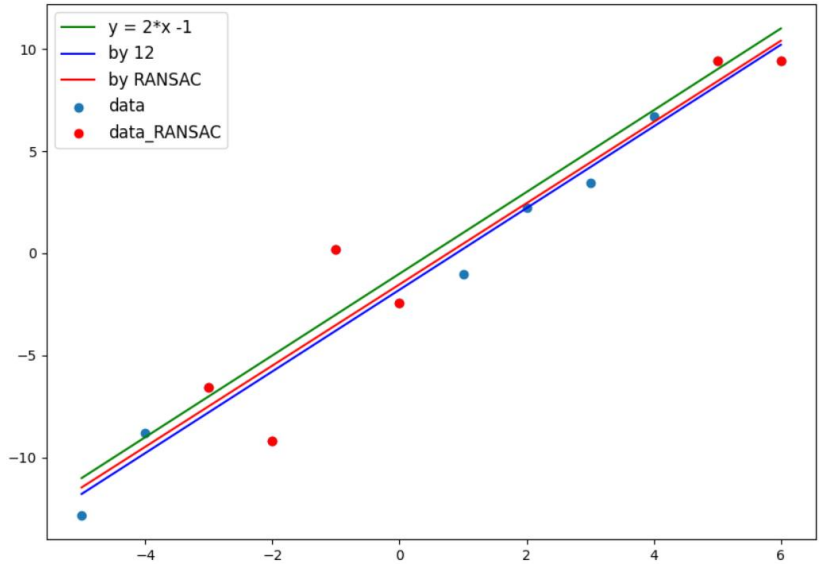
<각 벡터끼리의 유클리드 거리>
- A - B: 0.867
- A - C: 1.041
- B - C: 0.178

테스트 결과 #7, #8 (**A**: y = 2x – 1, **B**: RANSAC, **C**: 12 samples)



RANSAC: y = 2.29x – 2.08
12 samples: y = 1.94x – 1.15

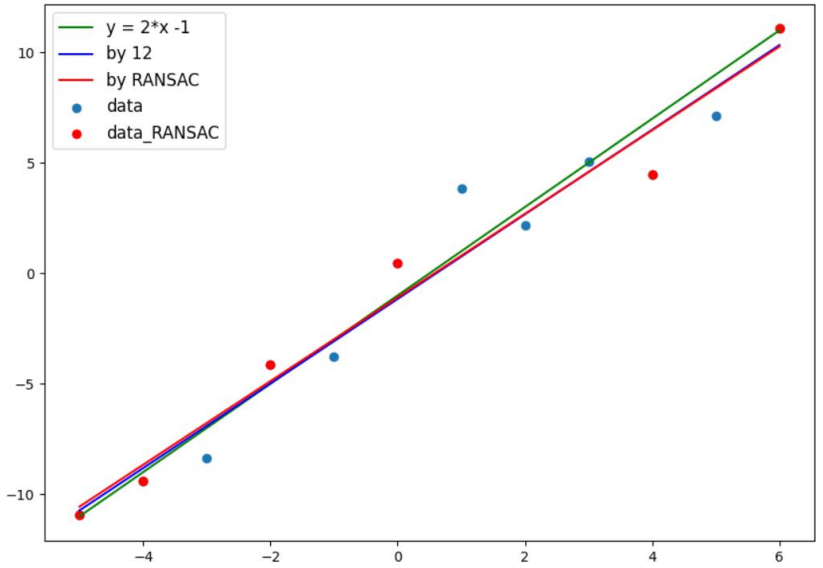
<각 벡터끼리의 유클리드 거리>
- A - B: 4.783
- A - C: 0.974
- B - C: 5.005



RANSAC: y = 1.99x – 1.52
12 samples: y = 2.00x – 1.79

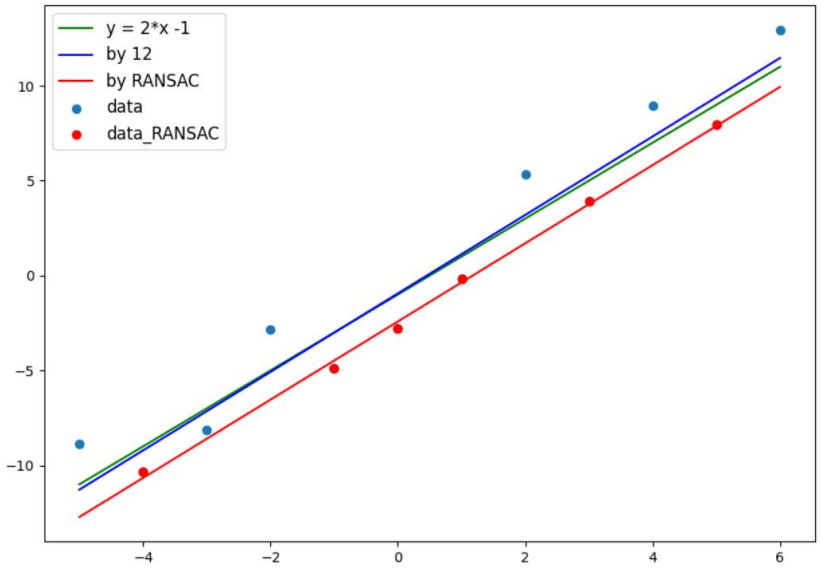
<각 벡터끼리의 유클리드 거리>
- A - B: 1.840
- A - C: 2.723
- B - C: 0.900

테스트 결과 #9, #10 (**A**: y = 2x – 1, **B**: RANSAC, **C**: 12 samples)



RANSAC: y = 1.89x – 1.09
12 samples: y = 1.91x – 1.16

<각 벡터끼리의 유클리드 거리>
- A - B: 1.374
- A - C: 1.231
- B - C: 0.318



RANSAC: y = 2.06x – 2.42
12 samples: y = 2.07x – 0.94

<각 벡터끼리의 유클리드 거리>
- A - B: 4.869
- A - C: 0.869
- B - C: 5.137

결과 분석

- RANSAC으로 구한 직선이 12개를 모두 linear square한 직선보다 $y = 2x - 1$에 더 유사한 경우는 테스트 10번 중 2번이었습니다.
  - RANSAC을 이용해서 직선을 구하는 것보다는 모든 샘플을 다 linear square해서 구하는게 대체로 더 원래 직선에 가깝게 나온다는 것을 알 수 있습니다.
- RANSAC으로 구한 직선이 $y = 2x - 1$ 직선에 더 유사한 경우가 5번, 12개 샘플로 구한 직선에 더 유사한 경우가 5번 나왔습니다.
  - 테스트 횟수를 늘려보면 경향성이 나올 수도 있겠지만 10번의 테스트로는 특별한 경향성이 나타나지 않았습니다.