수치해석
HW #4

Curve fitting

2018007956 김채아

[Cruve fitting code]

```python
import numpy as np
import numpy.linalg as linalg
import random
import matplotlib.pyplot as plt

# sample data list
data = [[-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]]

# 리스트에서 랜덤으로 6개의 값을 추출하고 행렬로 만든다
sampleList = random.sample(data, 6)
matrix = np.array(sampleList)
print('Randomly select 6 points:')
print(sampleList)

# Matrix A 만들기
# data matrix의 1열 추출
a = matrix[:, 0]
# 2 열 추출 (벡터b)
b = matrix[:, 1]

A1 = np.array((a)*(a))
A2 = np.array(a)
A3 = np.ones((6,1))

A = np.column_stack([A1,A2,A3])
print('matrix A:')
print(A)
```

샘플 데이터 8개 중
6개를 랜덤으로 선택하여
행렬A를 만든다

$$A = \begin{bmatrix} x1^2 & x1 & 1 \\ x2^2 & x2 & 1 \\ x3^2 & x3 & 1 \\ x4^2 & x4 & 1 \\ x5^2 & x5 & 1 \\ x6^2 & x6 & 1 \end{bmatrix}$$

$$Ax = b$$

$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y1 \\ y2 \\ y3 \\ y4 \\ y5 \\ y6 \end{bmatrix}$$

```python
# pseudoinverse of A :(A.T * A)^(-1) * A.T
# x = pseudoinverse of A * b
p = linalg.pinv(A)
x = np.dot(np.array(p),b)
print('x:',x)# x = [a b c]

# matlab으로 y=ax^2+bx+c 그래프 그리기
a = np.linspace(-5, 5)
b= x[0]*(a**2) + x[1]*a + x[2]
plt.ylim(0,50)
plt.plot(a,b,'r')

# sample data 좌표 표시
plt.scatter(matrix[:,0],matrix[:,1])

plt.show()
```

파이썬 라이브러리를 이용해서 A의 pseudo inverse를 구하면,
x=(pseudo inverse of A)*b 이므로
벡터 x를 구할 수 있다
그러면 구하려고 했던 이차함수의 계수 a,b,c값을 알 수 있다

[Error computing]

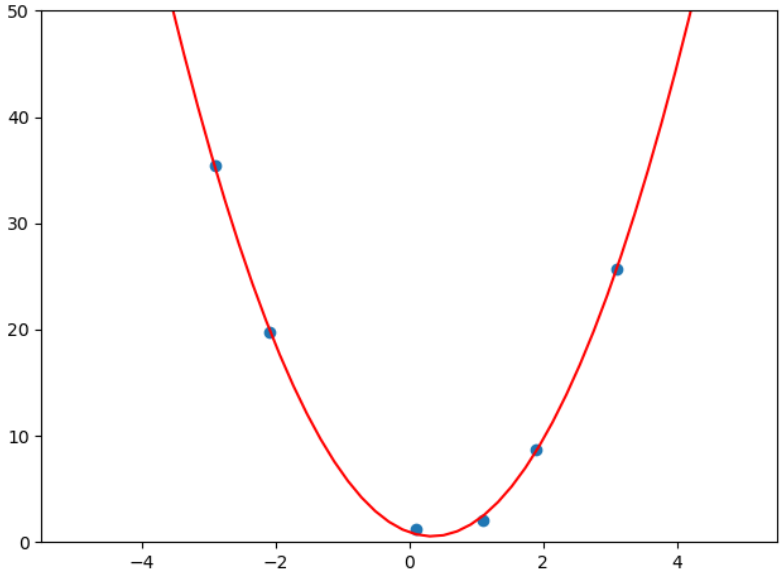
```
# error 구하기
def f(a):
    return x[0]*(a**2) + x[1]*a + x[2]

b = matrix[:, 1]
g = list()
for i in matrix[:, 0]:
    g.append(f(i))
print('b:', b)
print('g:', g)
g = np.array(g)

# error
error = (b-g)**2
print('error: ', end='')
e = 0
for i in range(6):
    e += error[i]
print(e)
```

샘플 데이터와 curve fitting 된 그래프의 y값 차이의 제곱들을 더한 값을 에러라고 하면,
샘플 데이터 8개 중 어느 6개가 선택 됐느냐에 따라 에러 값이 달라진다

에러 값을 여러 개 구해보았다)
0.6371932297413745 <-[1.9, 8.7], [3.1, 25.7], [-2.9, 35.4], [-2.1, 19.7], [1.1, 2.1], [0.1, 1.2]
3.6810900073420036 <-[1.9, 8.7], [-2.1, 19.7], [4.0, 41.5], [-2.9, 35.4], [3.1, 25.7], [1.1, 2.1]
3.51723710822567 <-[-2.9, 35.4], [1.1, 2.1], [0.1, 1.2], [3.1, 25.7], [-2.1, 19.7], [4.0, 41.5]
0.9820914134298143 <-[-2.9, 35.4], [0.1, 1.2], [1.9, 8.7], [4.0, 41.5], [-0.9, 5.7], [1.1, 2.1]
0.31609128099518496 <-[3.1, 25.7], [-0.9, 5.7], [0.1, 1.2], [1.9, 8.7], [1.1, 2.1], [-2.9, 35.4]
2.6833184807185493 <-[1.9, 8.7], [3.1, 25.7], [4.0, 41.5], [1.1, 2.1], [0.1, 1.2], [-2.1, 19.7]
3.4189135992982944<-[-2.9, 35.4], [4.0, 41.5], [-0.9, 5.7], [0.1, 1.2], [-2.1, 19.7], [3.1, 25.7]
3.3958742020286325 <-[1.9, 8.7], [-0.9, 5.7], [3.1, 25.7], [1.1, 2.1], [-2.9, 35.4], [4.0, 41.5]
1.993435455244379 <-[3.1, 25.7], [-0.9, 5.7], [-2.1, 19.7], [0.1, 1.2], [1.9, 8.7], [4.0, 41.5]
3.4189135992982798 <-[0.1, 1.2], [-0.9, 5.7], [4.0, 41.5], [-2.9, 35.4], [3.1, 25.7], [-2.1, 19.7]
0.4399579157738923 <-[-0.9, 5.7], [0.1, 1.2], [-2.1, 19.7], [1.9, 8.7], [3.1, 25.7], [-2.9, 35.4]
0.35602374650786817 <-[-2.9, 35.4], [1.1, 2.1], [-2.1, 19.7], [1.9, 8.7], [-0.9, 5.7], [3.1, 25.7]
위의 경우에선 최소 에러는 0.316…이었고
에러가 0.5가 넘지 않는 세가지 경우를 살펴보면, 각각 포함하지 않는 x값은 아래와 같다
(-2.1, 4.0) (1.1, 4.0) (0.1, 4.0)
공통적으로 4.0을 포함하지 않으니 에러가 낮게 나왔다
표본이 적긴 하지만 지금 결과에서는,
[4.0, 41.5] 는 신뢰도가 떨어지는 데이터로,
이 좌표를 선택하지 않아야 더 정확한 이차곡선으로 피팅 된다는 것을 알 수 있다

[코드 실행 결과]

```
Randomly select 6 points:   샘플 데이터에서 랜덤하게 선택된 6개의 좌표
[[1.9, 8.7], [3.1, 25.7], [-2.9, 35.4], [-2.1, 19.7], [1.1, 2.1], [0.1, 1.2]]
matrix A:
[[ 3.61  1.9   1.  ]
 [ 9.61  3.1   1.  ]
 [ 8.41 -2.9   1.  ]
 [ 4.41 -2.1   1.  ]
 [ 1.21  1.1   1.  ]
 [ 0.01  0.1   1.  ]]
x: [ 3.29913407 -2.23038378  0.92367764]   curve fitting 결과값(2차함수의 계수 a,b,c)
b: [ 8.7 25.7 35.4 19.7  2.1  1.2]   b벡터 : 샘플데이터의 y값
g: [8.595822437534952, 25.714166301374004, 35.137508124612665, 20.15666482760706, 2.4622077028885827, 0.7336306059827384]
error: 0.6371932297413745     에러: (b-g)^2 의 합                    g : curve fitting된 그래프의 y값
```



y = 3.29913407x^2-2.23038378x+0.92367764

approximation된 이차함수식을 빨간색 그래프로 그리고,
비교하기 위해 샘플데이터를 점으로 찍어주었다

[코드 실행 결과]

```
Randomly select 6 points:
[[1.9, 8.7], [-2.1, 19.7], [4.0, 41.5], [-2.9, 35.4], [3.1, 25.7], [1.1, 2.1]]
matrix A:
[[ 3.61  1.9   1.  ]
 [ 4.41 -2.1   1.  ]
 [16.    4.    1.  ]
 [ 8.41 -2.9   1.  ]
 [ 9.61  3.1   1.  ]
 [ 1.21  1.1   1.  ]]
x: [ 3.14839702 -2.36528771  1.49913273]
b: [ 8.7 19.7 41.5 35.4 25.7  2.1]
g: [8.3707993117173 66, 20.350667758934 37, 42.412334145618274, 34.83648598793819, 24.42283615433795, 2.706876641453822]
error: 3.6810900073420036
```

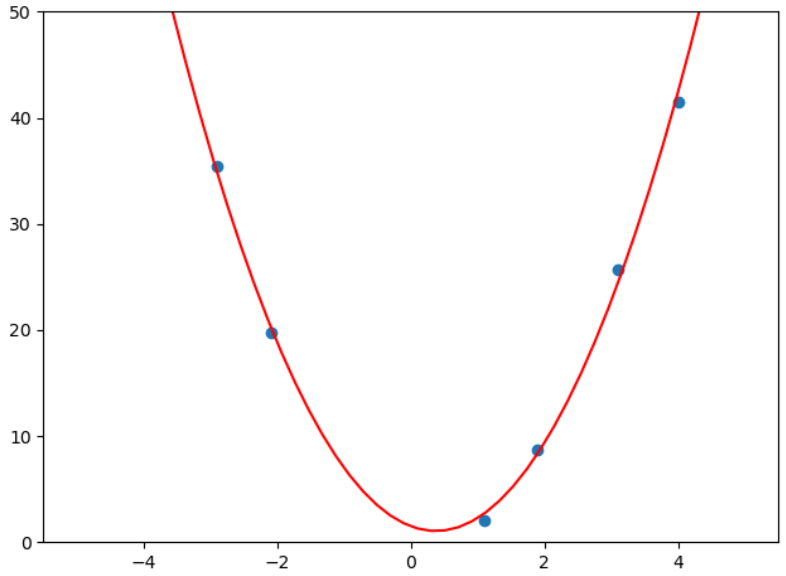
샘플 데이터에서 랜덤하게 선택된 6개의 좌표

curve fitting 결과값(2차함수의 계수 a,b,c)

b벡터 : 샘플데이터의 y값

에러: (b-g)^2 의 합

g : curve fitting된 그래프의 y값

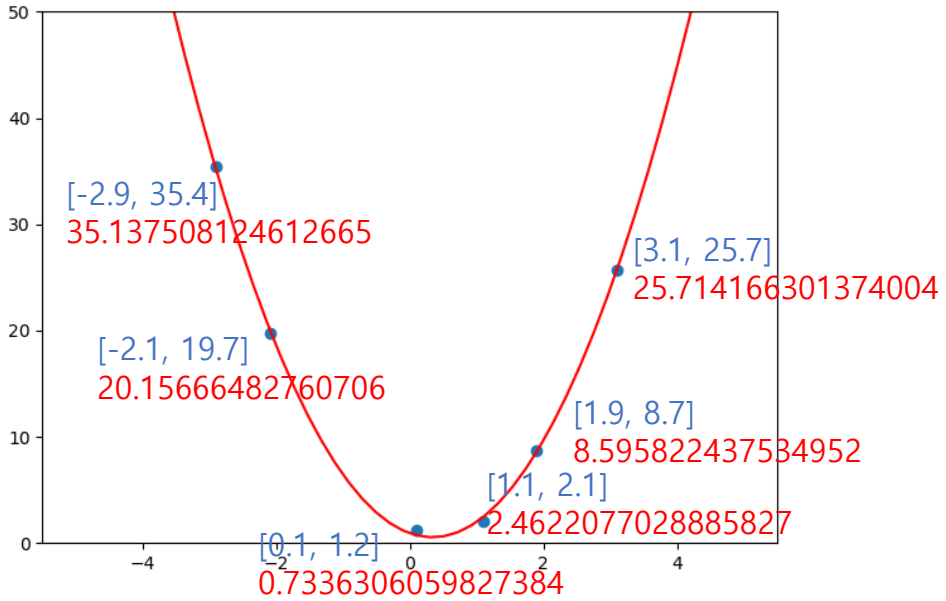


y = 3.14839702x^2-2.36528771x+1.49913273

[Compare two curves]

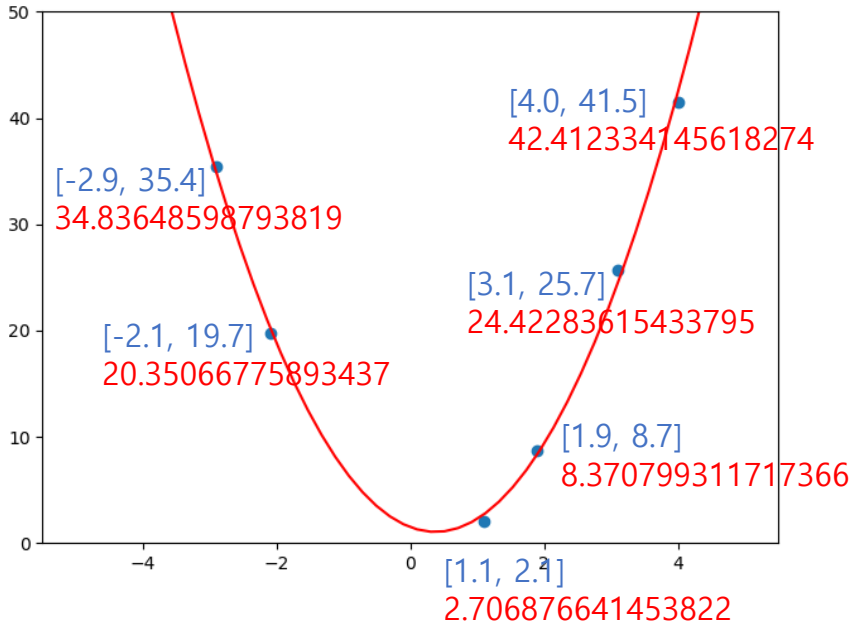
- curve 1

[1.9, 8.7], [3.1, 25.7], [-2.9, 35.4], [-2.1, 19.7], [1.1, 2.1], [0.1, 1.2]
⇒ y = 3.29913407x^2-2.23038378x+0.92367764
error: 0.6371932297413745



- curve 2

[1.9, 8.7], [-2.1, 19.7], [4.0, 41.5], [-2.9, 35.4], [3.1, 25.7], [1.1, 2.1]
⇒ y = 3.14839702x^2-2.36528771x+1.49913273
error: 3.6810900073420036



curve 1의 에러가 약 3정도나 작은 것을 보아 왼쪽 그래프가 더 신뢰도 높은 그래프이다
샘플 데이터가 어떻게 구성 되느냐에 따라 결과가 달라질 수 있다는 것을 확인할 수 있다