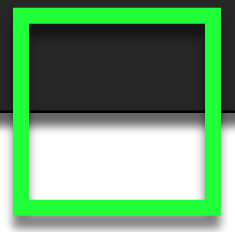
#수치 해석
#과제 4

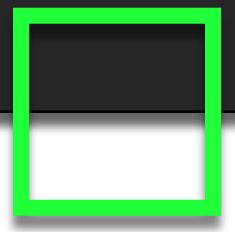
Curve Fitting

컴퓨터소프트웨어학부
2018008395 박정호



#1

문제 해결을 위한 모델링



주어진 상황 분석

이번 과제는 8개의 점을 가지고 이 점을 지나는 이차식 형태의 curve를 추정하는 것이다. 여기서 이차식을 아래와 같은 형태라고 보면,

$$y = ax^2 + bx + c$$

여기서 unknown은 이 이차식의 coefficient인 a, b, c라고 할 수 있다. 즉 (a, b, c)를 구해야 할 벡터 X라고 볼 수 있다. 그럼 주어진 상황을 벡터 X에 대해 나타내면 다음과 같은데,

$$AX = B$$

여기서 A는 위의 이차식에서 (a, b, c)에 dot product되는 $(x^2, x, 1)$을 row vector로 갖는 N*3 행렬이라 할 수 있고, 이차식에서 y에 해당하는 벡터라고 볼 수 있다.

주어진 상황 분석

따라서 주어진 점을 각각 $P_1 \sim P_8$ 이라고 하고, 이 점들의 x, y좌표를 $x_1 \sim x_8$, $y_1 \sim y_8$이라고 하게 되면, 앞에서 언급한 행렬 연산 식은 다음과 같은 형태가 된다.

$$AX = B$$

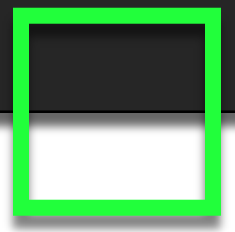
$$\begin{bmatrix} {x_1}^2 & x_1 & 1 \\ {x_2}^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ {x_N}^2 & x_N & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

즉, 랜덤으로 6개의 점을 선택하고, 각 점을 각 행렬에 항에 맞게 계산 후 대입하면 기본적인 모델링은 완성인 것이다.

또한, 여기서 양변의 차이가 최소가 되는 (a, b, c)를 찾는 것이 이 과제의 목표가 될 것이다.

#2

구현



설명에 앞서…

우선 행렬 연산을 해야 하기 때문에 구현의 용이함을 위해서 python을 사용했다. 또한 행렬 연산 관련 method 가 많은 오픈 라이브러리인 numpy를 사용했다.

다만, 사용한 method는 inverse matrix를 구하는 numpy.linalg.inv와, 행렬 곱 연산을 해주는 @ operator 뿐이며, 기타 구현을 필요 이상으로 간단하게 할 수 있는 함수는 사용하지 않았음을 밝힌다.

행렬 A 구축하기

```
def getMatrix():
    global points
    A = np.zeros((6, 3))
    B = np.zeros((6, 1))
    idx = rd.sample(range(8), 2)
    j = 0
    print("Chosen Point :")
    for i in range(8):
        if i == idx[0] or i == idx[1]:
            continue
        print("(%10f, %10f)"%(points[i][0], points[i][1]))
        A[j] = np.array([points[i][0]**2, points[i][0], 1])
        B[j] = points[i][1]
        j += 1

    print("\nMatrix A :")
    print(A)
    print("\nVector B :")
    print(B)
    return A, B
```

global 영역에서 가져오는 points는 주어진 8개의 점을 저장한 List 이다. np와 rd는 각각 numpy와 random 라이브러리의 준말이다.

우선 A와 B를 각각 알맞은 크기의 영행렬로 만들어 둔다. 그 다음 random.sample을 이용해서 8개의 점 중 사용되지 않을 두 점을 고른다.

이제 남은 6개의 점을 찾아서 각각 $(x^2, x, 1)$ 꼴의 벡터로 만들어 준 후, A의 행벡터로 대입했다. 또한 B에는 각 점의 y좌표를 넣어 주었다.

pseudo inverse 구하기

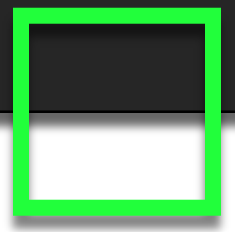
```
def pseudoinverse(A):
    return np.linalg.inv((A.T@A))@A.T
```

기본적으로 numpy에는 numpy.linalg.pinv라는 pseudo inverse를 구하는 method가 존재한다. 하지만, 필자는 numpy.linalg.pinv를 쓰지 않고, numpy.linalg.inv와 행렬곱을 사용해서 pseudo inverse를 직접 구해보았다.

식 자체는 상당히 간단하다. 인자로 받은 A 행렬의 transpose와 A를 곱하고, 이 결과의 역행렬을 다시 A의 transpose와 곱하면 pseudo inverse를 구할 수 있다.

#3

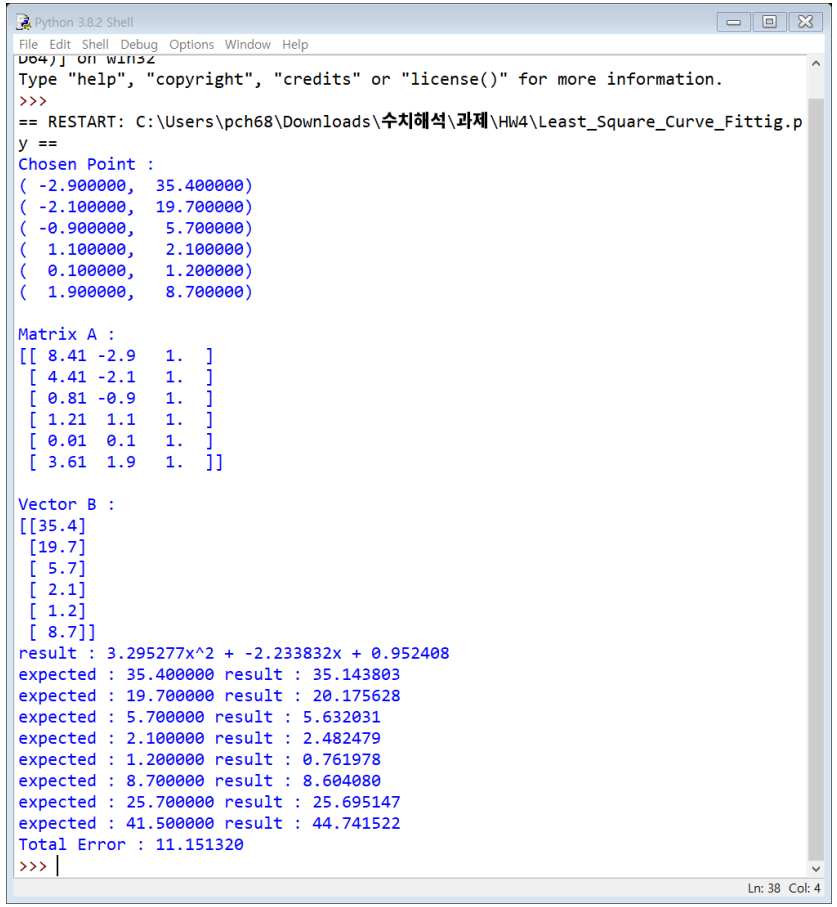
결과와 비교



두 번의 무작위 점 선택 - 결과

결과 1

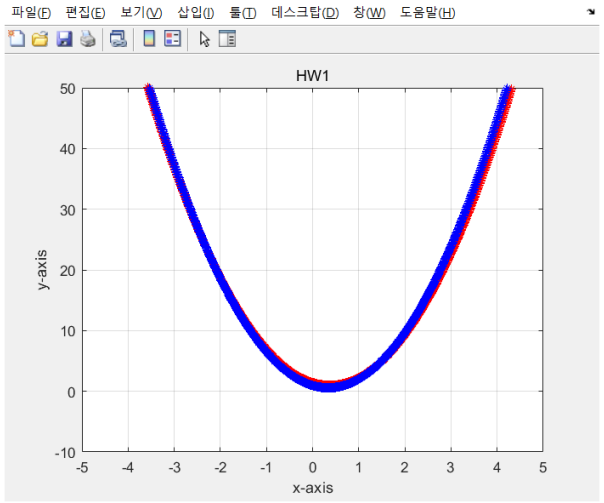
$3.151814x^2 - 2.356234x + 1.425734$



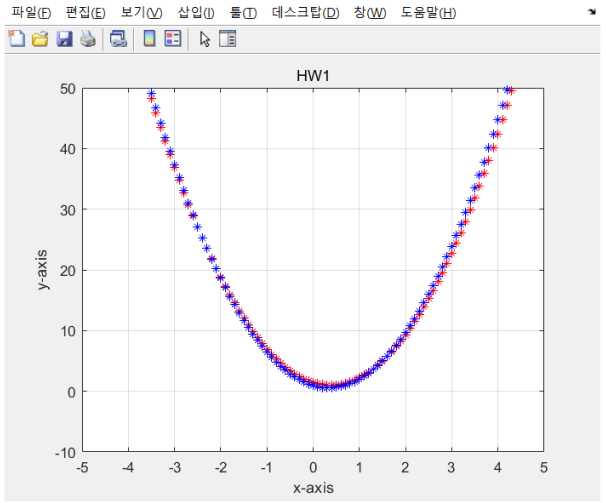
결과 2

$3.295277x^2 - 2.233832x + 0.952408$

그래프를 통한 결과 분석



간격 0.01



간격 0.1

결과 1에서 도출한 식은 $3.151814x^2 - 2.356234x + 1.425734$ 이었고,
결과 2에서 도출한 식은 $3.295277x^2 - 2.233832x + 0.952408$ 이었다.

이 식을 가지고 MATLAB을 이용해서 그래프를 그려보았다. 붉은색의 그래프가 결과 1, 푸른색의 그래프가 결과 2인데, 거의 같은 형태를 띄고 있지만, 차이가 있는 부분도 보임을 알 수 있다.

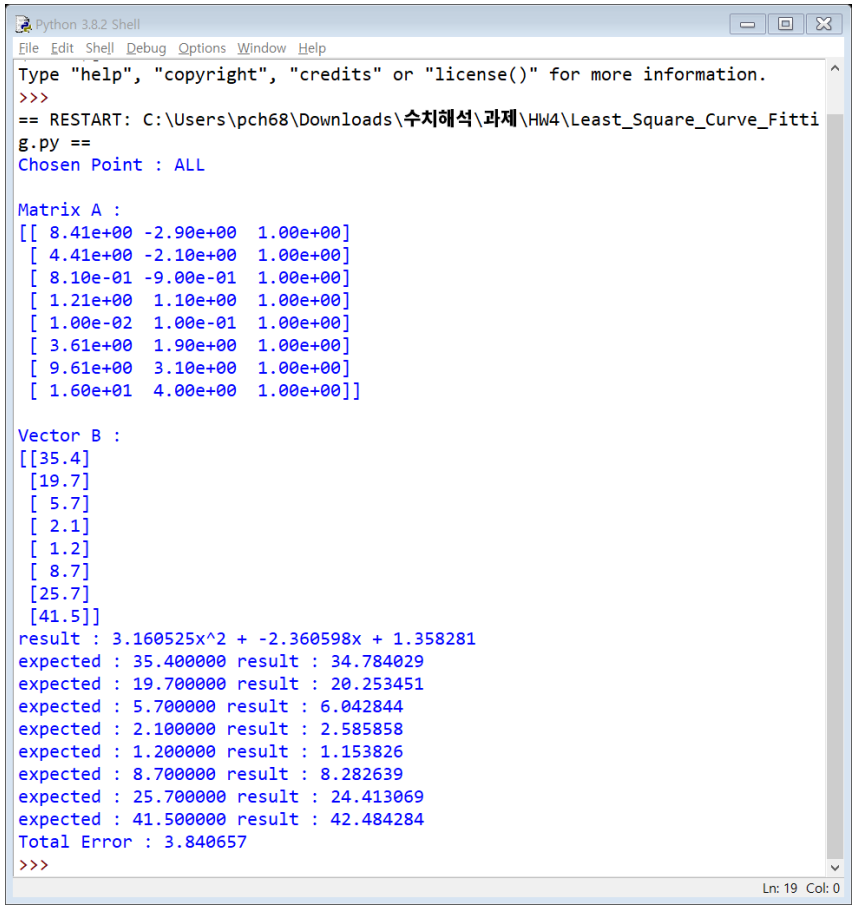
오차를 통한 결과 분석

그래프만으로는 명확한 차이가 보이지 않아서, 직접 주어진 점의 좌표를 대입해서 그 차이를 보았다.
앞의 출력 결과에서, expected와 result로 비교하는 부분과, total error가 그것인데, 대체로 작게는 0.1에서 크게는 3 정도까지도 오차가 나오는 것을 볼 수 있었다.

또한 이 오차들의 제곱을 합한 total error도 결과 1에서는 약 3.85, 결과 2에서는 약 11.15 정도로 매우 큰 차이를 보임을 알 수 있었다. 실제로 반복해서 실험해본 결과 오차는 3.8에서 11.15 사이라는 상당히 큰 폭으로 나타나는 것을 알 수 있었다.

무작위로 점을 고르는 것이 오차에 큰 영향을 미친다는 것을 알 수 있었다.

모든 점을 가지고 구한 결과

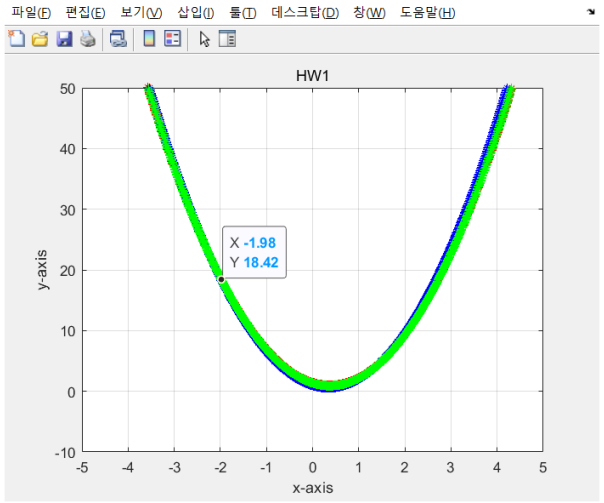


무작위로 선정한 점이 얼마나 큰 영향을 주는지 알아보기 위해 이번에는 무작위 선정 없이 모든 점을 가지고 결과를 구해보았다.

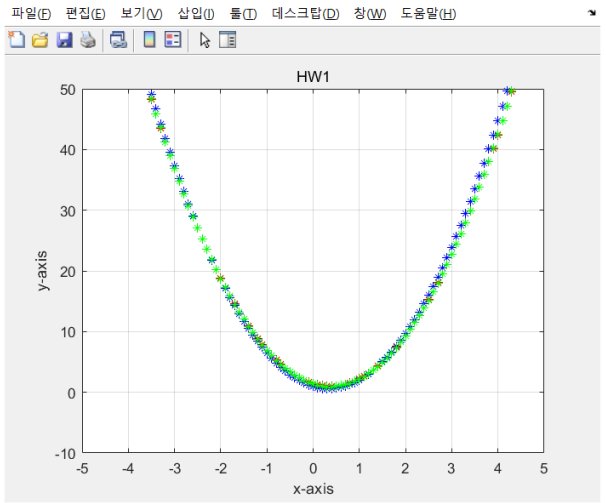
우선 오차가 상당히 작다는 점을 확인할 수 있었다. 또한 각 점에 대한 개별적인 오차도 약 0.05~1.3 정도로 앞의 결과에 비해 상당히 작은 수준을 보여 주었다.

오차의 크기로 보아 앞에서 구한 결과 1은 상당히 잘 구해진 표본임도 확인할 수 있었다.

그래프를 통한 결과 분석



간격 0.01



간격 0.1

앞의 결과 1과 결과 2의 그래프 위에 모든 점을 가지고 구한 이차식의 그래프도 그려보았다.

역시 결과 2에 해당하는 푸른색의 그래프는 초록색 그래프와 약간의 차이를 보였지만, 결과 1의 그래프는 거의 보이지 않는 것을 확인할 수 있었다. 결과 1이 모범적인 표본이었음을 알 수 있는 부분이다.

결과에 대한 고찰

표본을 뽑아서 least square method를 사용하는 것은 결과도출은 상당히 빠를 수 있겠지만, 표본에 따라 그리 좋지 않은 결과를 만날 수도 있음을 확인했다. 데이터셋의 크기가 그리 크지 않거나, 연산장치의 성능이 좋다면 굳이 표본을 고르지 않고 모든 데이터를 가지고 estimate를 진행하는 것이 확실하게 더 좋은 결과를 도출할 수 있을 것이다.

하지만 데이터셋이 너무 크다면 모든 데이터를 사용하는 것은 그리 좋은 선택이 아닐 것이다. 이럴 때는 표본을 여러 번 뽑아서 estimate를 각 표본에 대해 여러 번 구해보는 것이 효율적일 것이다. 잘 골라진 표본은 전체 데이터셋과 비슷한 결과를 도출할 수 있음을 앞의 결과 1을 통해서 확인했기 때문이다.