Assignment 08

과제 정의

[Polynomial fitting]

Solve a least square problem to find an optimal polynomial curve for a given set of two dimensional points.

Demonstrate the effect of the degree of polynomial in fitting a given set of points.

- choose a polynomial curve and generate points along the curve with random noise
- plot the generated noisy points along with its original polynomial without noise
- plot the approximating polynomial curve obtained by solving a least square problem
- plot the approximating polynomial curve with varying polynomial degree

모듈 정의

그래프를 그리기 위해 Python3 matplotlib module 을 사용합니다

수학적 연산을 하기 위해서 sympy 모듈을 사용합니다.

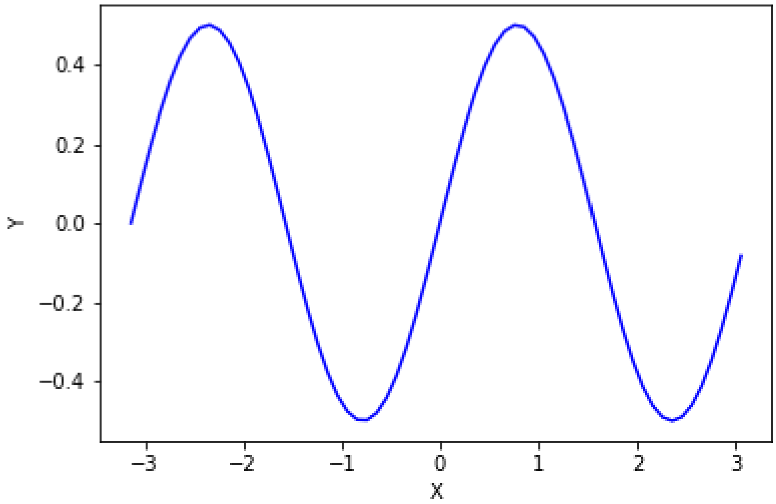
노이즈 연산을 주기 위하여 random 모듈을 사용합니다.

In [33]:
```
import matplotlib.pyplot as plt
import numpy as np
import random
import sympy
```

1. 기본 그래프

$$x \in A, -pi \leq A \leq pi$$
$$cos(x) * sin(x)$$

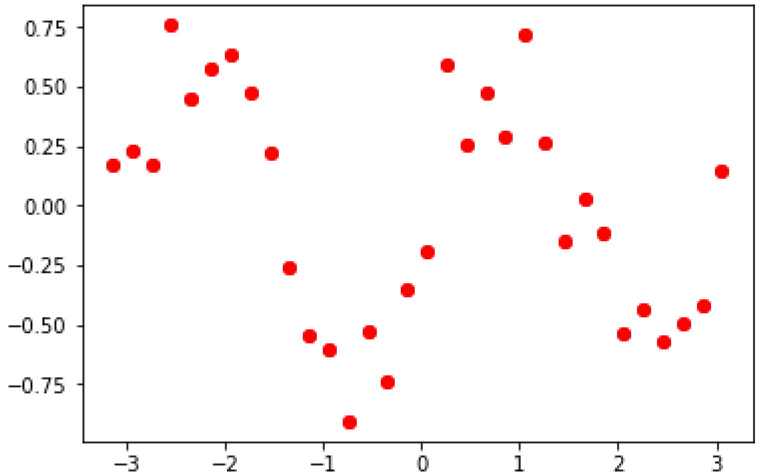
In [68]:
```
pi = 3.141592
plt.xlabel("X")
plt.ylabel("Y")
xn = np.arange(pi * -1 , pi * 1, 0.1)
yn = [sympy.cos(v) * sympy.sin(v) for v in xn]
plt.plot(xn, yn,'b-')
plt.show()
```



2. 그래프에 노이즈를 줘서 least square 할 data 생성

- 그래프를 기반으로 0.2 * np.random.randn(1) 으로 노이즈를 줍니다.

In [76]:
```
coordinateX = []
coordinateY = []
for x in np.arange(pi * -1 , pi * 1 , 0.2):
    select_point_y = sympy.cos(x) * sympy.sin(x)
    select_point_y = select_point_y + 0.2 * np.random.randn(1)
    coordinateX.append(x)
    coordinateY.append(select_point_y)
    plt.plot(x, select_point_y,'ro')

xn = np.array(coordinateX,dtype=float)
yn = np.array(coordinateY,dtype=float)
plt.plot(xn, yn,'ro')
plt.show()
```



3. N(차원)을 증가 시킬 때마다 그래프의 변화 확인

A is xn에 대하여 차원이 늘어날 수록 $x^0, x^1, \ldots, x^{N-1}$ 을 따르는 xn를 A행렬에 담는다.

$$A = \begin{bmatrix} xn_1^0 & xn_1^1 & xn_1^2 \ldots \\ xn_2^0 & xn_2^1 & xn_2^2 \ldots \\ xn_3^0 & xn_3^1 & xn_3^2 \ldots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

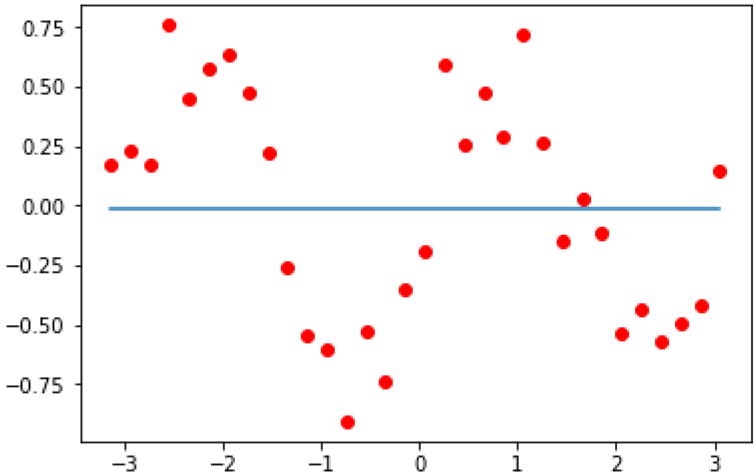
the least square, $yn1$ is given by

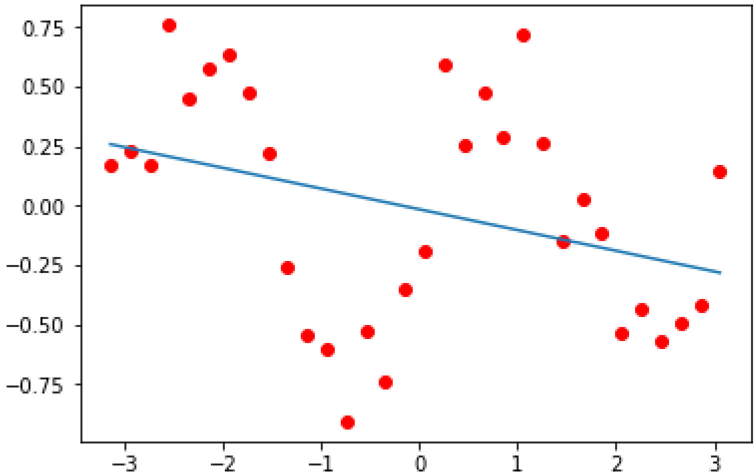
$$yn\mathbf{1} = (A^T A)^{-1} A^T yn.$$

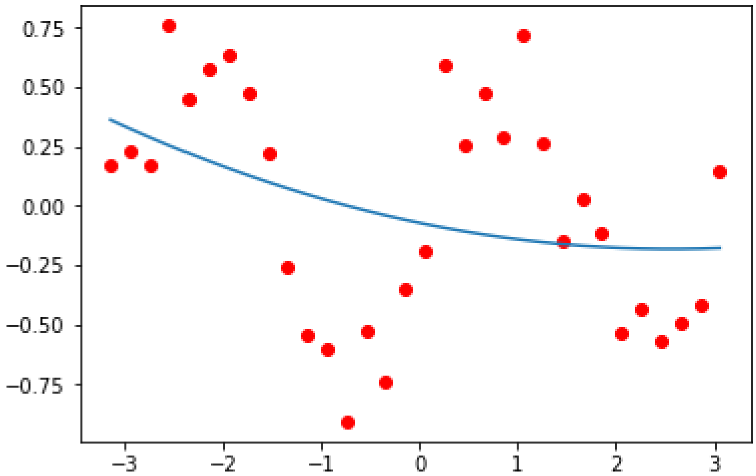
Fit a polynomial $p(x) = p[0] * x^{deg} + \ldots + p[deg]$ of degree deg to points (x, y)

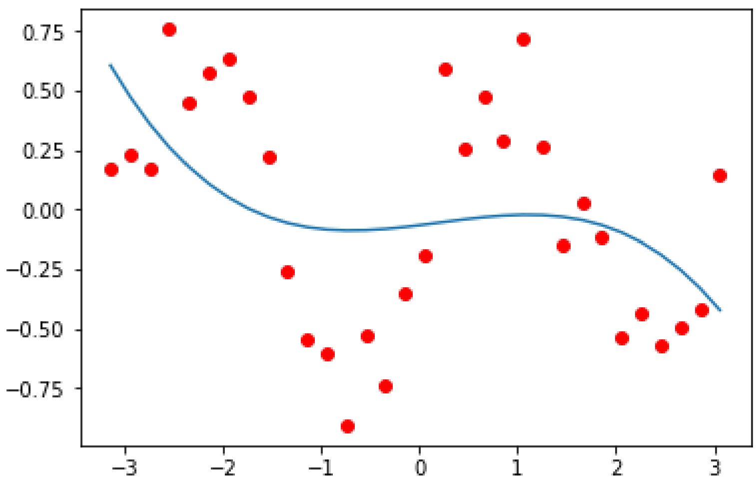
$$y = \begin{bmatrix} x[0]^N * p[0] + \ldots + x[0] * p[N-1] + p[N] = y[0] \\ x[1]^N * p[0] + \ldots + x[1] * p[N-1] + p[N] = y[1] \\ \vdots \\ x[k]^N * p[0] + \ldots + x[k] * p[N-1] + p[N] = y[k] \end{bmatrix}$$

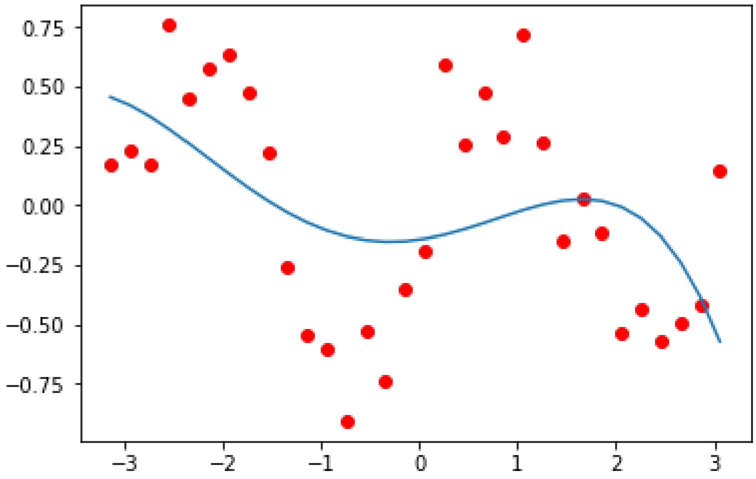
In [78]:
```
for N in range(1,20):
    xn = np.array(coordinateX,dtype=float)
    yn = np.array(coordinateY,dtype=float)
    plt.plot(xn, yn,'ro')
    A = np.column_stack([xn**(N-1-i) for i in range(N)])
    yn1 = np.dot(np.linalg.inv(np.dot(A.T, A)), np.dot(A.T, yn))


    p = np.asarray(yn1)
    x = np.asarray(xn)
    y = np.zeros_like(xn)

    for i in range(len(p)):
        y = y * x + p[i]

    yn1 = y

    plt.plot(xn, yn1)
    plt.show()
```

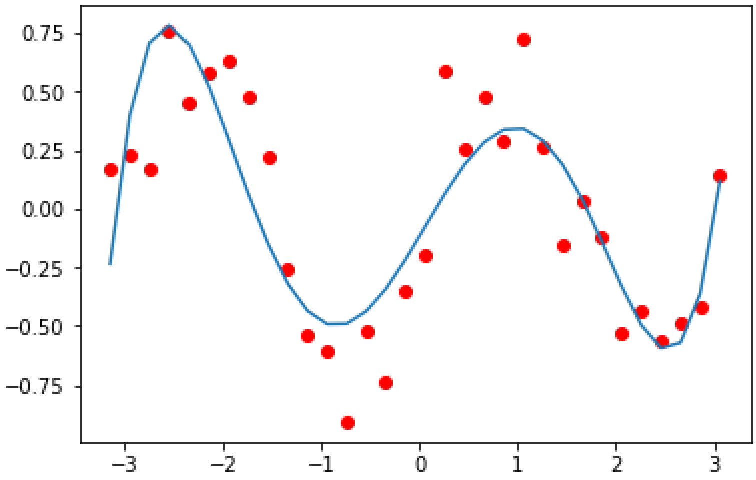

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

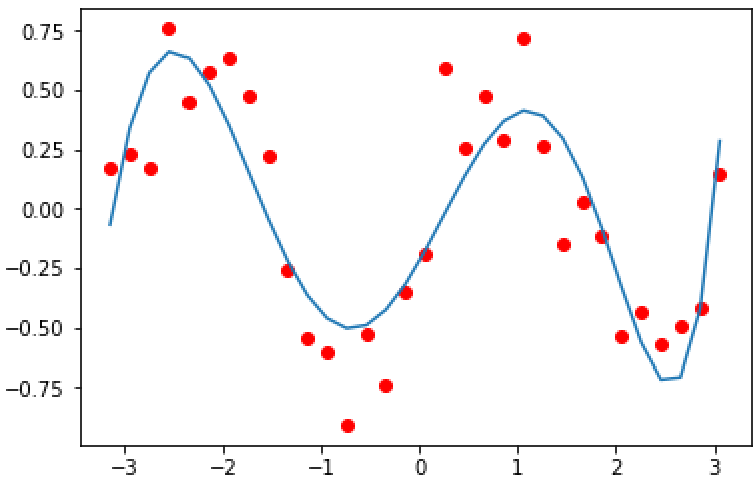

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

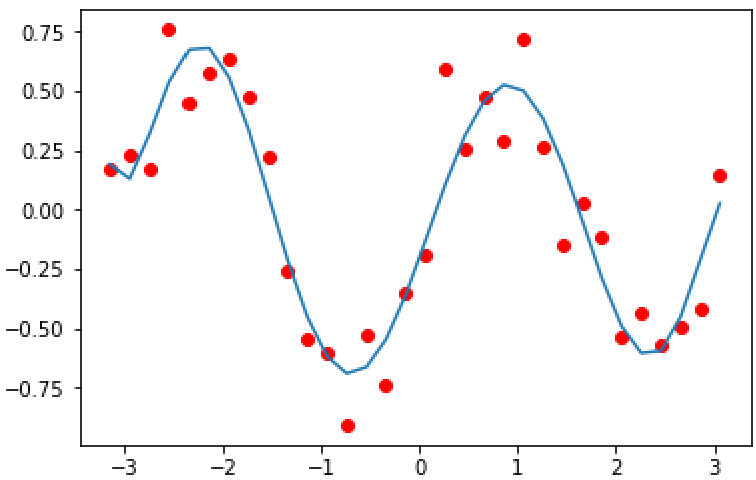

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

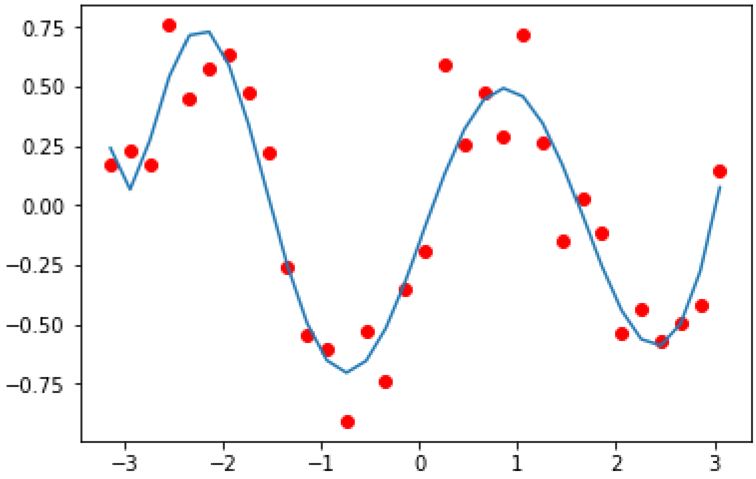

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

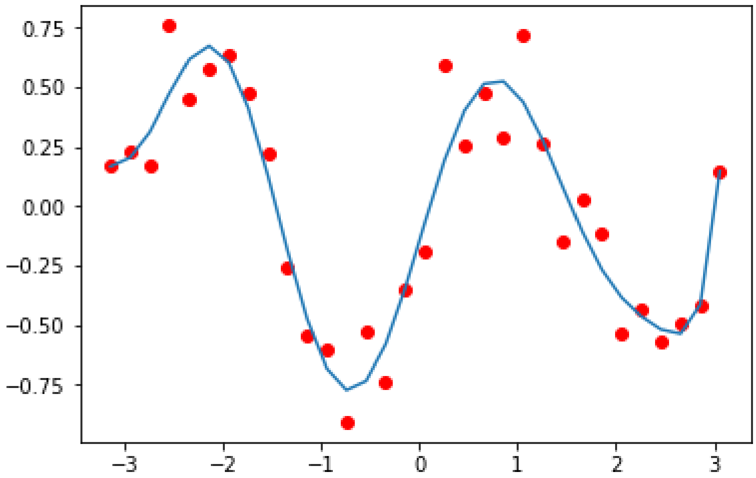

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

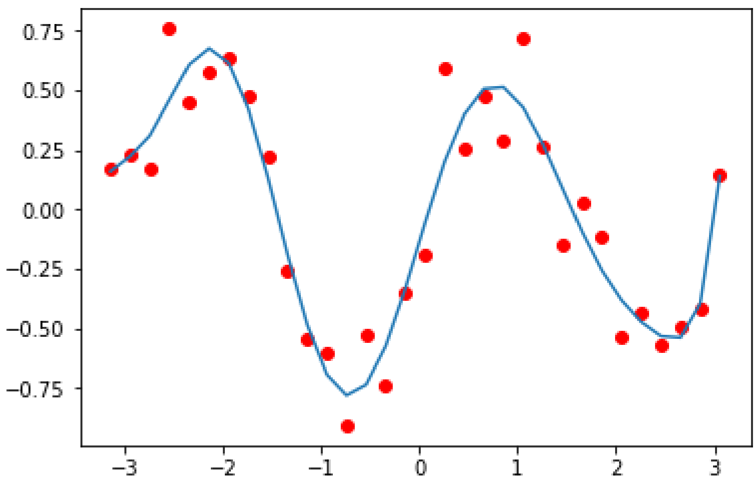

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

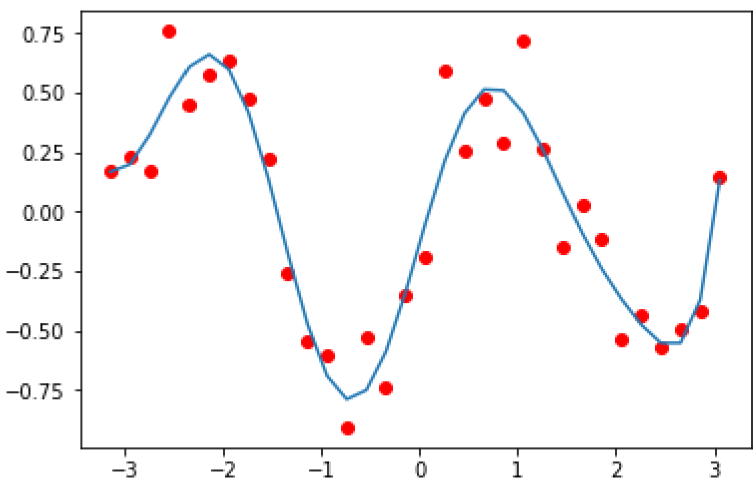

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

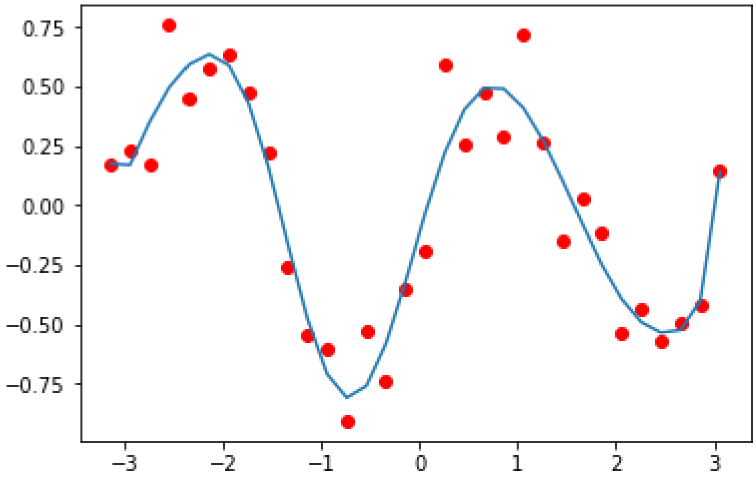

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

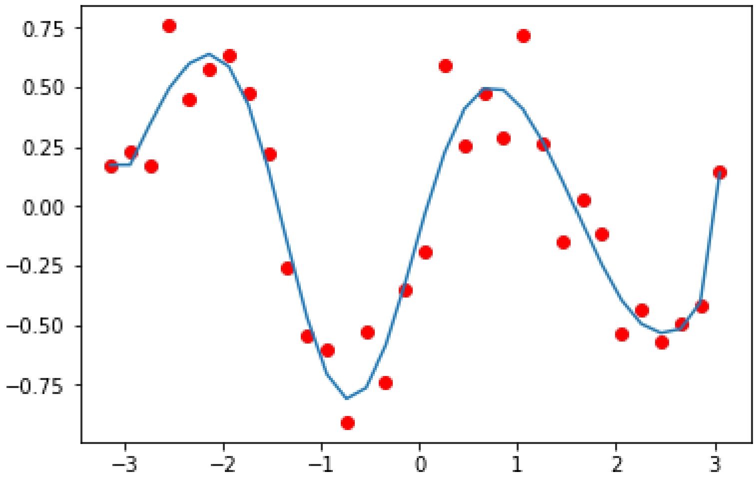

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

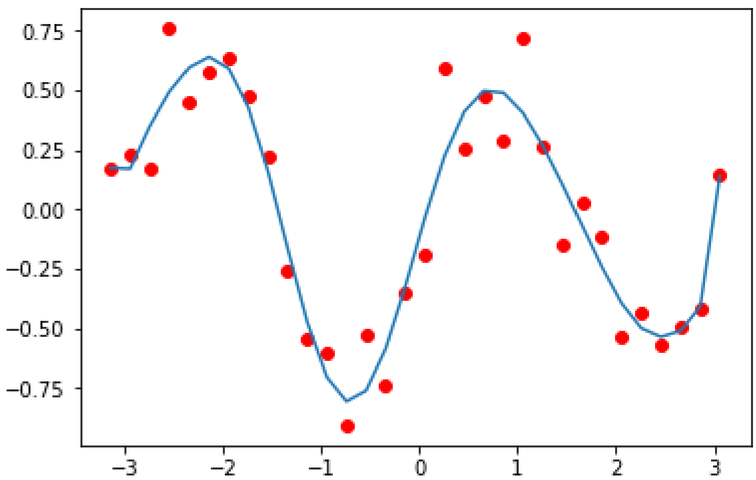

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

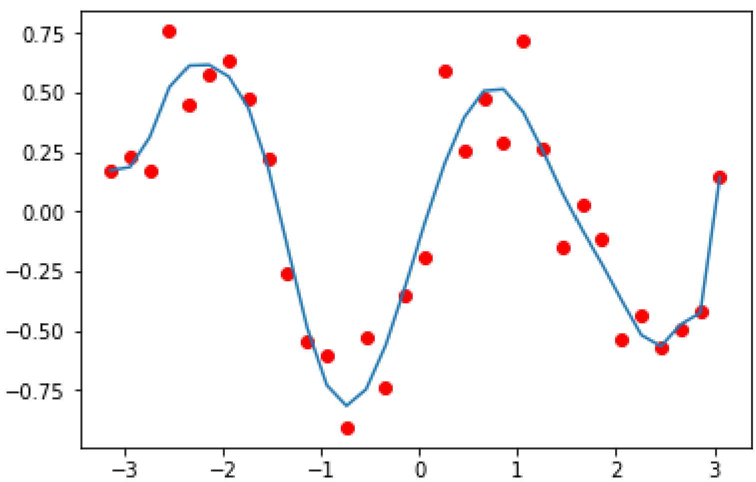

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

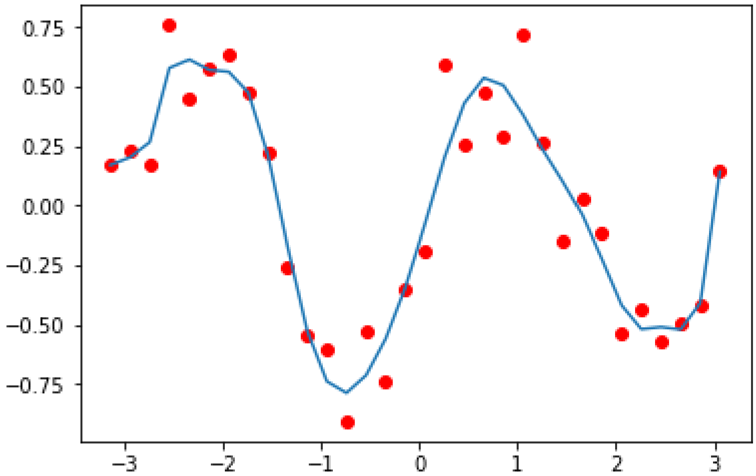

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

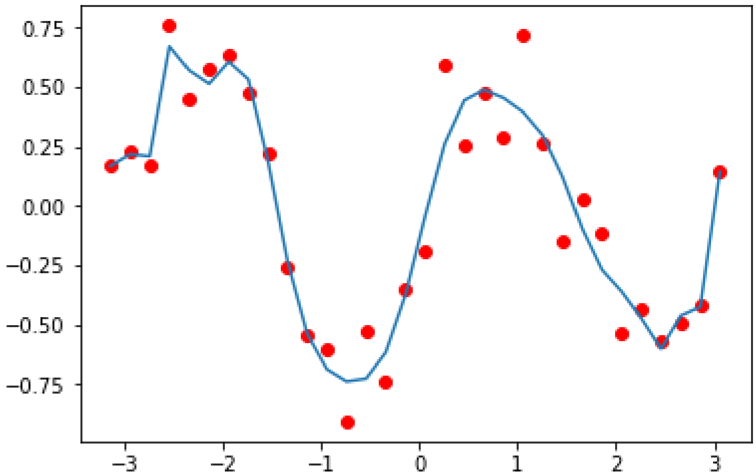

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3

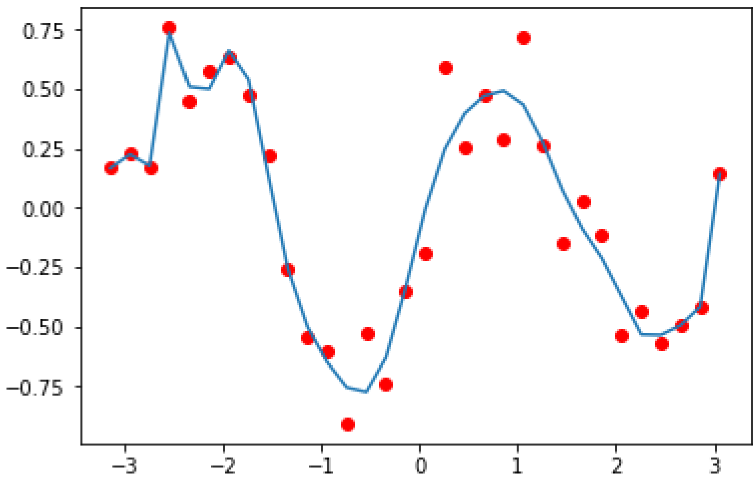

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−3
−2
−1
0
1
2
3