

Least square problem for polynomial regression

import library

```
In [ ]: import numpy as np
import matplotlib.image as img
import matplotlib.pyplot as plt
import matplotlib.colors as colors
```

load data points

- $\{(x_i, y_i)\}_{i=1}^n$

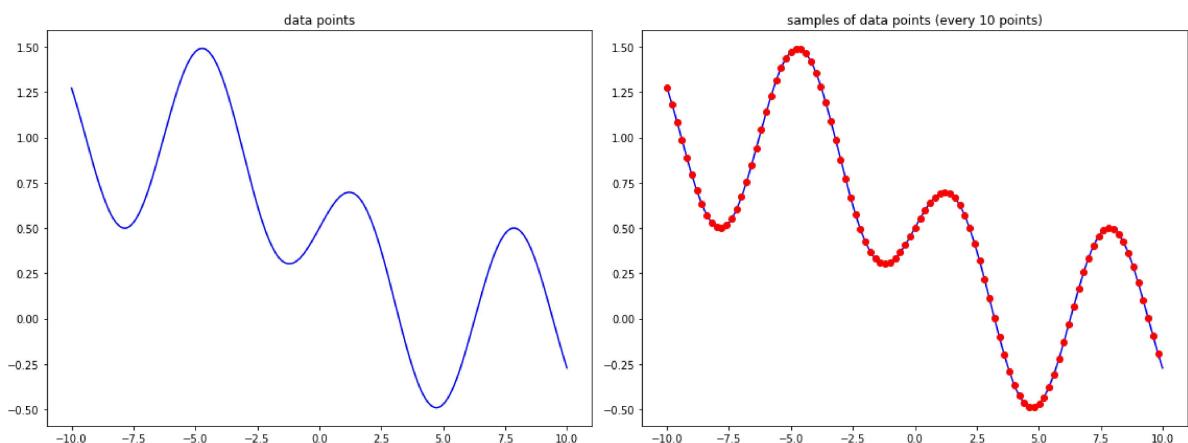
```
In [ ]: filename      = 'assignment_05_data.csv'
data          = np.loadtxt(filename, delimiter = ',')
x             = data[0, :]      # independent variable
y             = data[1, :]      # dependent variable
x_sample      = x[::10]
y_sample      = y[::10]

plt.figure(figsize=(16,6))

plt.subplot(121)
plt.plot(x, y, '-', color = 'blue')
plt.title('data points')

plt.subplot(122)
plt.plot(x, y, '-', color = 'blue')
plt.plot(x_sample, y_sample, 'o', color = 'red')
plt.title('samples of data points (every 10 points)')

plt.tight_layout()
plt.show()
```



solve a linear system of equation $Az = b$

$$A = \begin{bmatrix} x_1^0 & x_1^1 & \cdots & x_1^{p-1} \\ x_2^0 & x_2^1 & \cdots & x_2^{p-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^{p-1} \end{bmatrix}, \quad z = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{p-1} \end{bmatrix}, \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

construct matrix A for the polynomial regression with power $p - 1$

- useful functions : `np.power`

```
In [ ]: def construct_matrix_A(x, p):
    n = len(x)
    A = np.zeros([n, p])

    # ++++++
    # complete the blanks
    #

    for i in range(p):
        A[:, i] = np.power(x, i)

    #
    # ++++++
    #

    return A
```

construct vector b

```
In [ ]: def construct_vector_b(y):
    n = len(y)
    b = np.zeros([n, 1])

    # ++++++
    # complete the blanks
    #

    b[:, 0] = y

    #
    # ++++++
    #

    return b
```

solve the linear system of equation $Az = b$

- without regularization : $\min \frac{1}{2n} \|Az - b\|^2, \quad z = (A^T A)^{-1} A^T b$
- useful functions : `np.matmul`, `np.linalg.inv`, `np.sum`

```
In [ ]: def solve_regression(x, y, p):
    z = np.zeros([p, 1])
```

```

loss      = 0

# ++++++
# complete the blanks
#
n = len(y)
A = construct_matrix_A(x, p)
b = construct_vector_b(y)
z = np.matmul(np.matmul(np.linalg.inv(np.matmul(A.T, A)), A.T), b)
loss = np.sum(np.power(np.matmul(A, z) - b, 2), axis=0) / (2 * n)

#
# ++++++
return z, loss

```

- with regularization : $\min \frac{1}{2n} \|Az - b\|^2 + \frac{\alpha}{2} \|z\|^2$, $z = (A^T A + n\alpha I)^{-1} A^T b$ where I denotes identity matrix
- useful functions : `np.matmul`, `np.linalg.inv`, `np.sum`

```
In [ ]: def solve_regression_with_regularization(x, y, p, alpha):

    z      = np.zeros([p, 1])
    loss   = 0

    # ++++++
    # complete the blanks
    #

    n = len(y)
    A = construct_matrix_A(x, p)
    b = construct_vector_b(y)
    z = np.matmul(np.matmul(np.linalg.inv(np.matmul(A.T, A) + n * alpha * np.identity(p)), A.T), b)
    loss = np.sum(np.power(np.matmul(A, z) - b, 2)) / (2 * n) + np.sum((alpha * np.linalg.norm(z, 2) ** 2))

    #
    # ++++++
    return z, loss
```

approximate by polynomial regression

- $\hat{y} = Az^*$
- useful functions : `np.matmul`

```
In [ ]: def approximate(x, y, p):

    n      = len(y)
    y_hat = np.zeros([n, 1])
    loss   = 0

    # ++++++
    # complete the blanks
    #

    A = construct_matrix_A(x, p)
    (z_star, loss) = solve_regression(x, y, p)
```

```
y_hat = np.matmul(A, z_star)

#
# ++++++return y_hat, loss
```

```
In [ ]: def approximate_with_regularization(x, y, p, alpha):

    n      = len(y)
    y_hat = np.zeros([n, 1])
    loss   = 0

    # ++++++
    # complete the blanks
    #

    A = construct_matrix_A(x, p)
    (z_star, loss) = solve_regression_with_regularization(x, y, p, alpha)
    y_hat = np.matmul(A, z_star)

    #
    # ++++++
    #

    return y_hat, loss
```

functions for presenting the results

```
In [ ]: def function_result_01():

    plt.figure(figsize=(8,6))
    plt.plot(x, y, '-', color='blue')
    plt.title('data points')
    plt.show()
```

```
In [ ]: def function_result_02():

    p = 2
    (y_hat, _) = approximate(x, y, p)

    plt.figure(figsize=(8,6))
    plt.plot(x, y, '-.', color='blue')
    plt.plot(x, y_hat, '--', color='red')
    plt.show()
```

```
In [ ]: def function_result_03():
    p = 4
    (y_hat, _) = approximate(x, y, p)

    plt.figure(figsize=(8,6))
    plt.plot(x, y, '--', color='blue')
```

```
plt.plot(x, y_hat, '--', color='red')
plt.show()
```

In []: def function_result_04():

```
p          = 8
(y_hat, _) = approximate(x, y, p)

plt.figure(figsize=(8,6))
plt.plot(x, y, '--', color='blue')
plt.plot(x, y_hat, '--', color='red')
plt.show()
```

In []: def function_result_05():

```
p          = 16
(y_hat, _) = approximate(x, y, p)

plt.figure(figsize=(8,6))
plt.plot(x, y, '--', color='blue')
plt.plot(x, y_hat, '--', color='red')
plt.show()
```

In []: def function_result_06():

```
p          = 32
(y_hat, _) = approximate(x, y, p)

plt.figure(figsize=(8,6))
plt.plot(x, y, '--', color='blue')
plt.plot(x, y_hat, '--', color='red')
plt.show()
```

In []: def function_result_07():

```
p          = 2
alpha      = 0.1
(y_hat, _) = approximate_with_regularization(x, y, p, alpha)

plt.figure(figsize=(8,6))
plt.plot(x, y, '--', color='blue')
plt.plot(x, y_hat, '--', color='red')
plt.show()
```

In []: def function_result_08():

```
p          = 4
alpha      = 0.1
(y_hat, _) = approximate_with_regularization(x, y, p, alpha)

plt.figure(figsize=(8,6))
plt.plot(x, y, '--', color='blue')
plt.plot(x, y_hat, '--', color='red')
plt.show()
```

In []: def function_result_09():

```
p          = 8
alpha      = 0.1
(y_hat, _) = approximate_with_regularization(x, y, p, alpha)

plt.figure(figsize=(8,6))
```

```
plt.plot(x, y, '--', color='blue')
plt.plot(x, y_hat, '--', color='red')
plt.show()
```

```
In [ ]: def function_result_10():

    p          = 16
    alpha      = 0.1
    (y_hat, _) = approximate_with_regularization(x, y, p, alpha)

    plt.figure(figsize=(8,6))
    plt.plot(x, y, '--', color='blue')
    plt.plot(x, y_hat, '--', color='red')
    plt.show()
```

```
In [ ]: def function_result_11():

    p          = 32
    alpha      = 0.1
    (y_hat, _) = approximate_with_regularization(x, y, p, alpha)

    plt.figure(figsize=(8,6))
    plt.plot(x, y, '--', color='blue')
    plt.plot(x, y_hat, '--', color='red')
    plt.show()
```

```
In [ ]: def function_result_12():

    p          = 4
    (_, loss)  = approximate(x, y, p)

    print('loss = ', loss)
```

```
In [ ]: def function_result_13():

    p          = 16
    (_, loss)  = approximate(x, y, p)

    print('loss = ', loss)
```

```
In [ ]: def function_result_14():

    p          = 4
    alpha      = 0.1
    (_, loss)  = approximate_with_regularization(x, y, p, alpha)

    print('loss = ', loss)
```

```
In [ ]: def function_result_15():

    p          = 16
    alpha      = 0.1
    (_, loss)  = approximate_with_regularization(x, y, p, alpha)

    print('loss = ', loss)
```

results

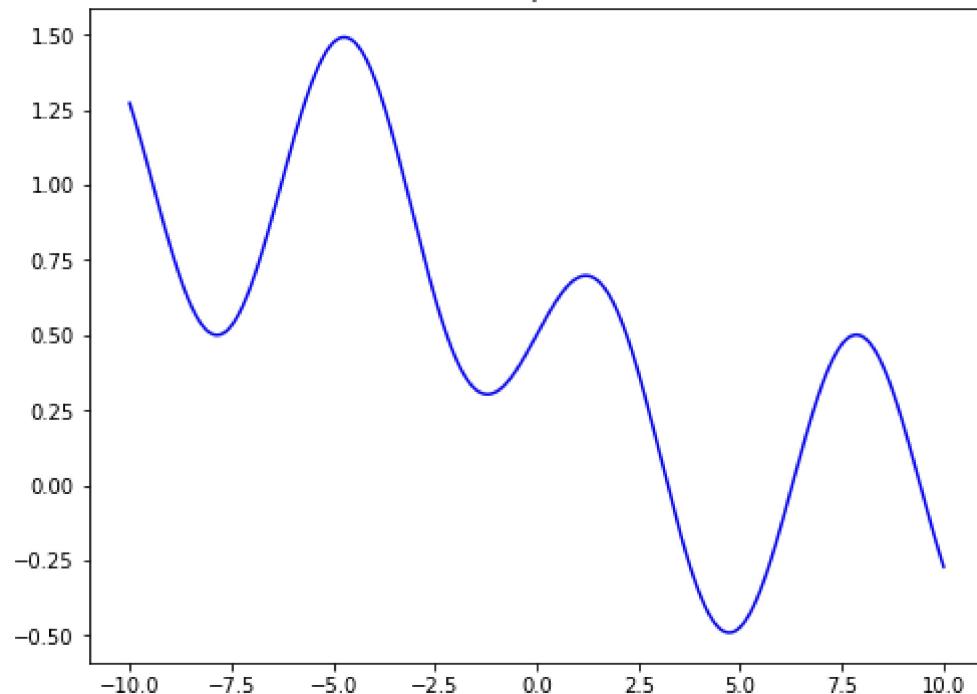
```
In [ ]: number_result = 15

for i in range(number_result):
    title = '## [RESULT {:02d}]'.format(i+1)
    name_function = 'function_result_{:02d}()'.format(i+1)

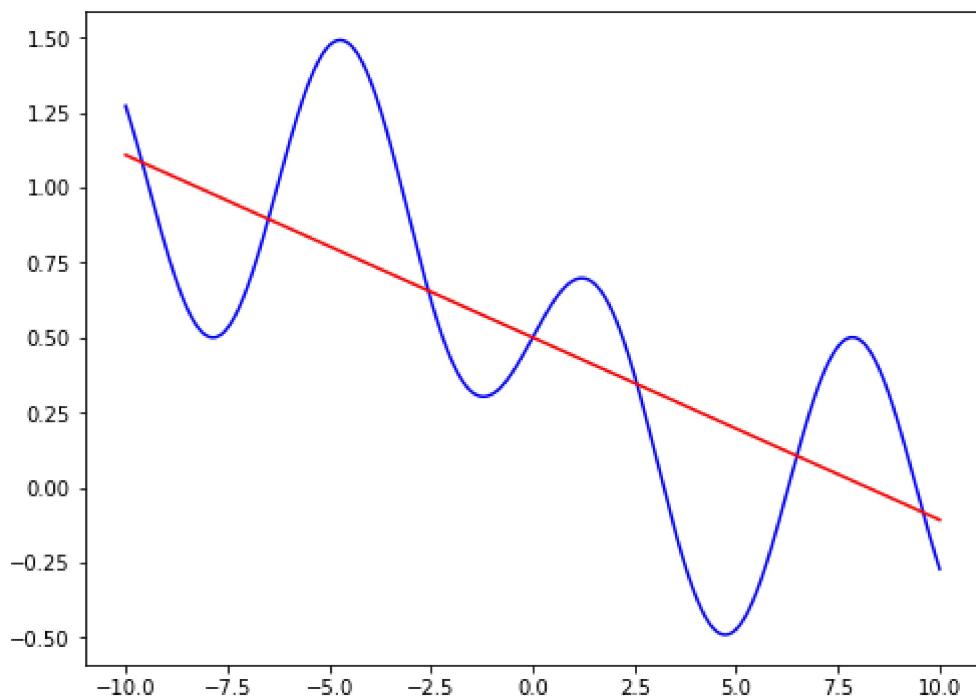
    print('*****')
    print(title)
    print('*****')
    eval(name_function)
```

```
*****
## [RESULT 01]
*****
```

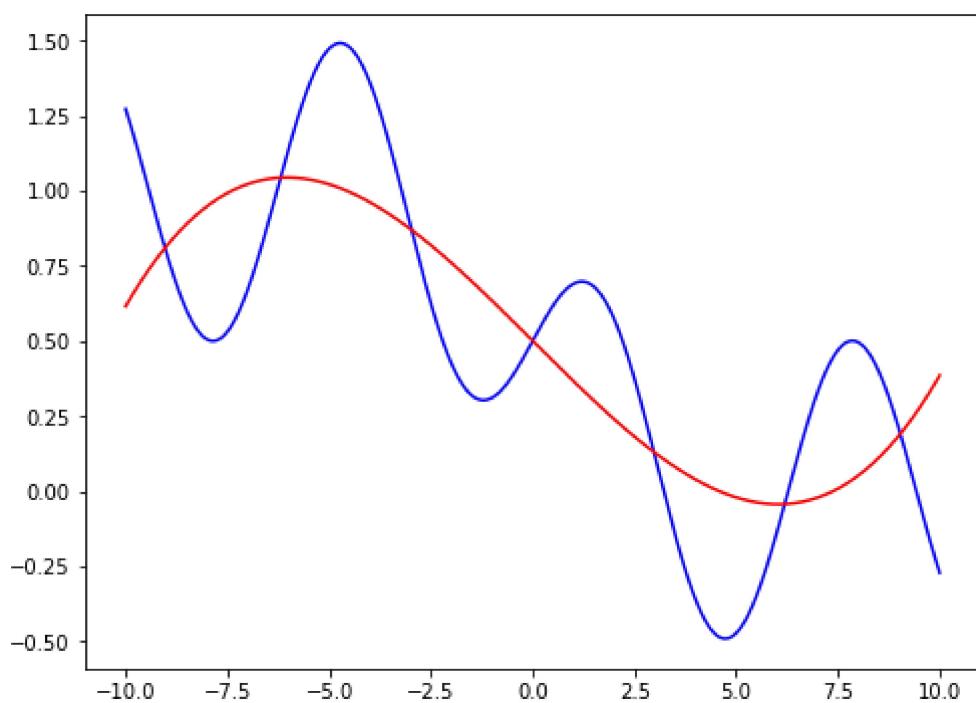
data points



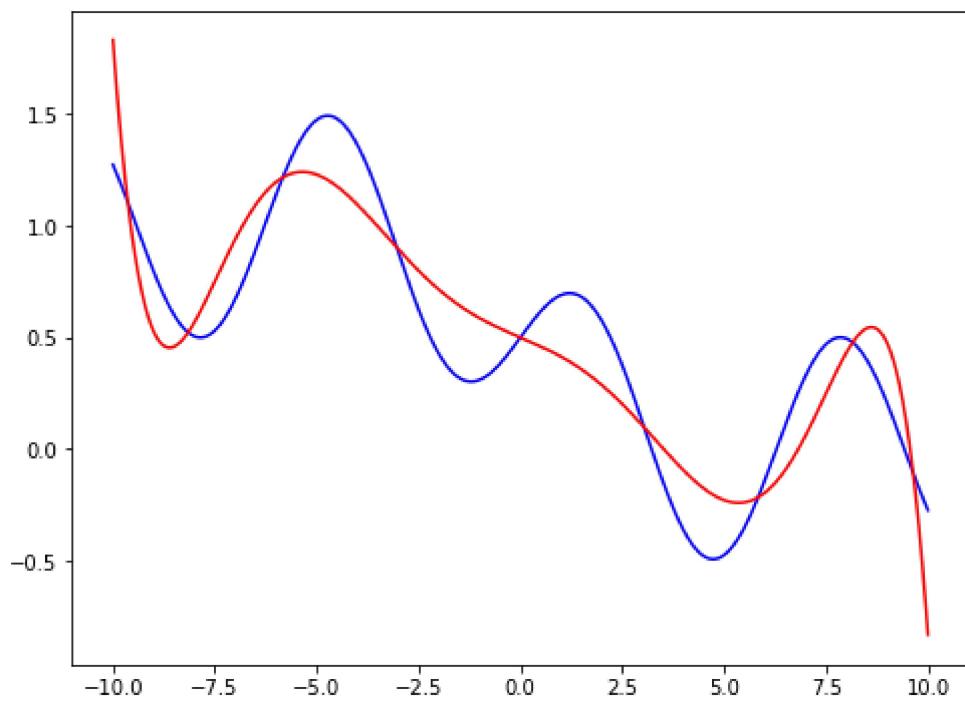
```
*****
## [RESULT 02]
*****
```



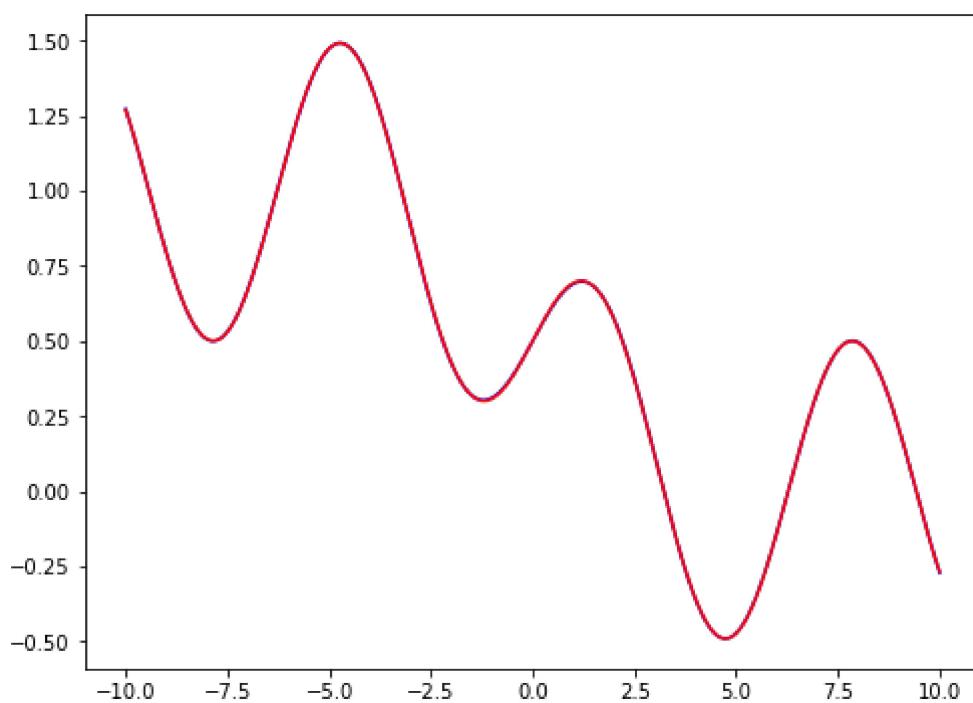
```
*****  
## [RESULT 03]  
*****
```



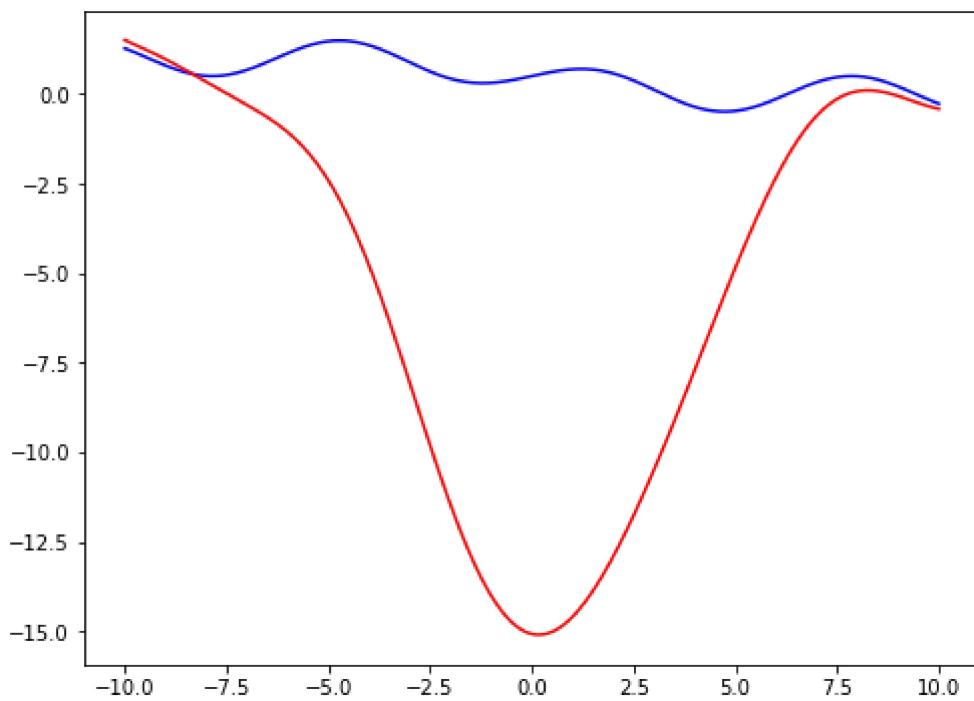
```
*****  
## [RESULT 04]  
*****
```



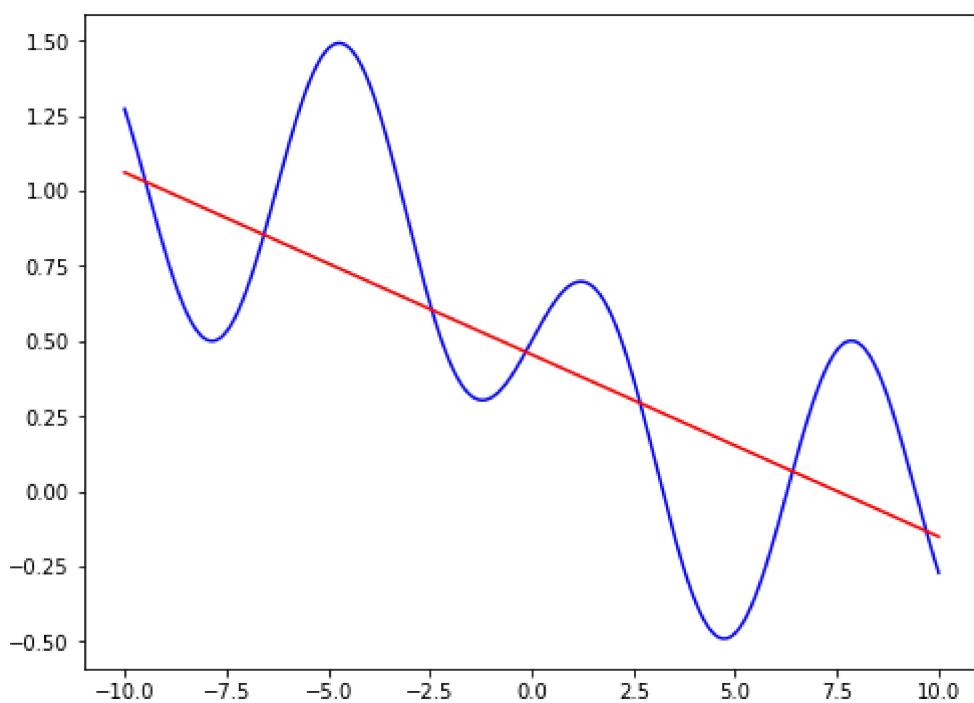
```
*****  
## [RESULT 05]  
*****
```



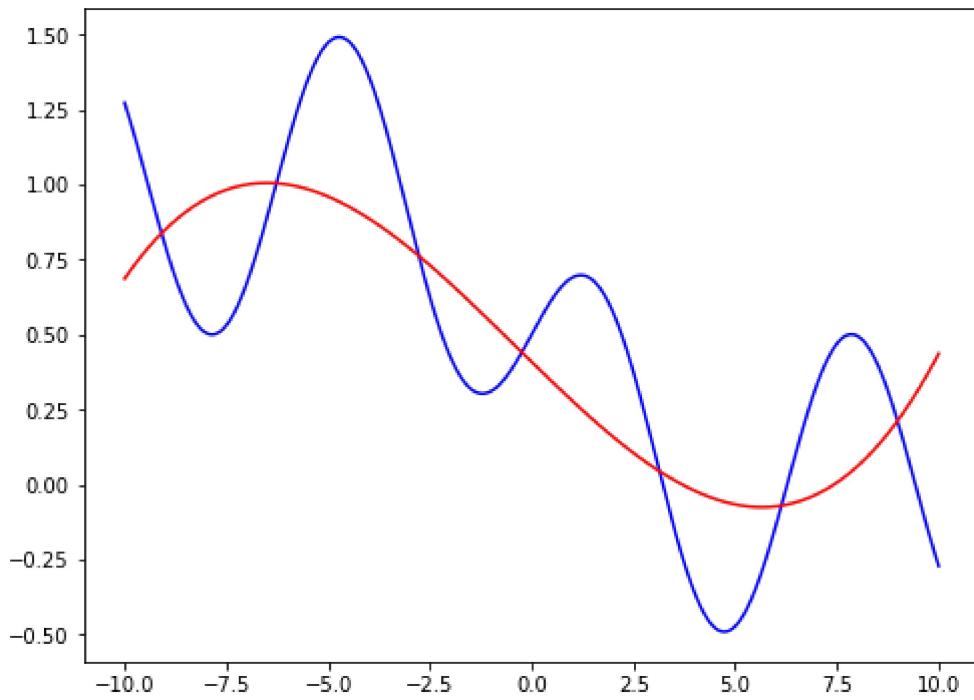
```
*****  
## [RESULT 06]  
*****
```



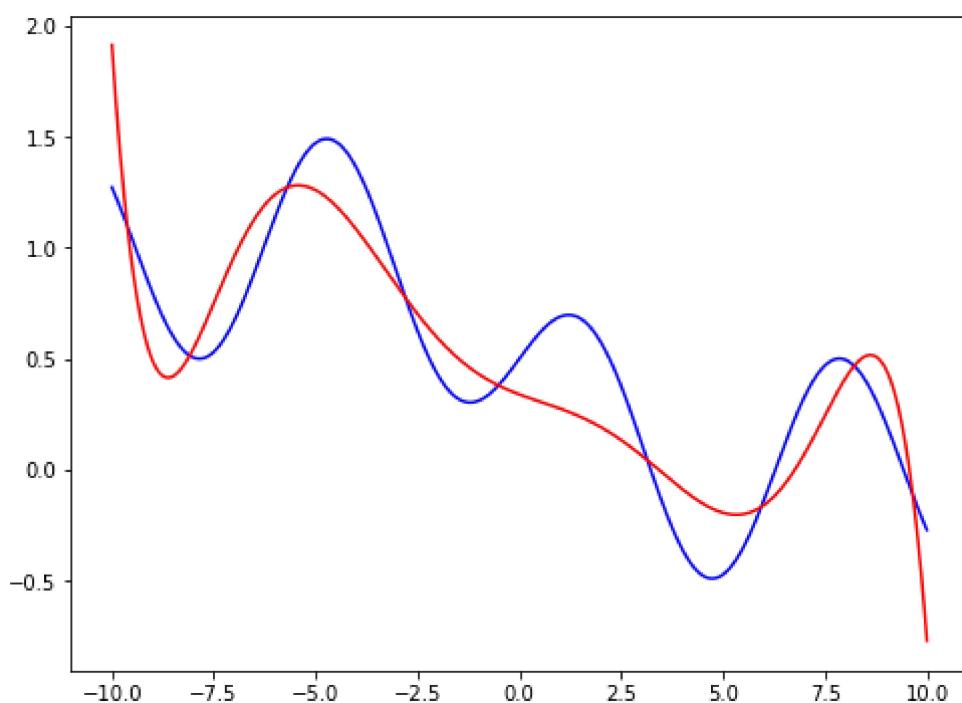
```
*****  
## [RESULT 07]  
*****
```



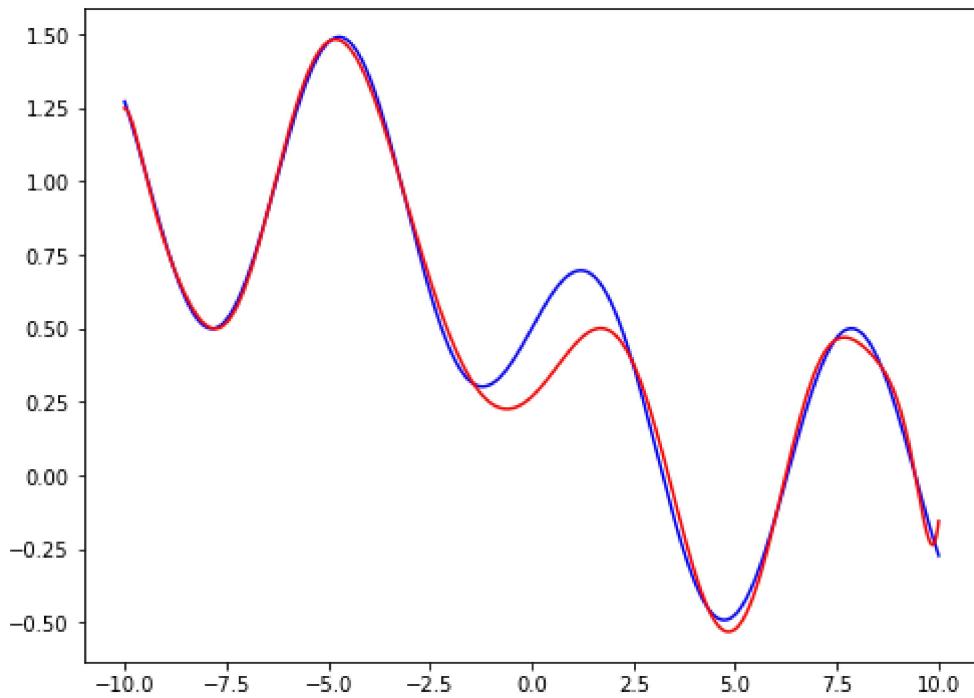
```
*****  
## [RESULT 08]  
*****
```



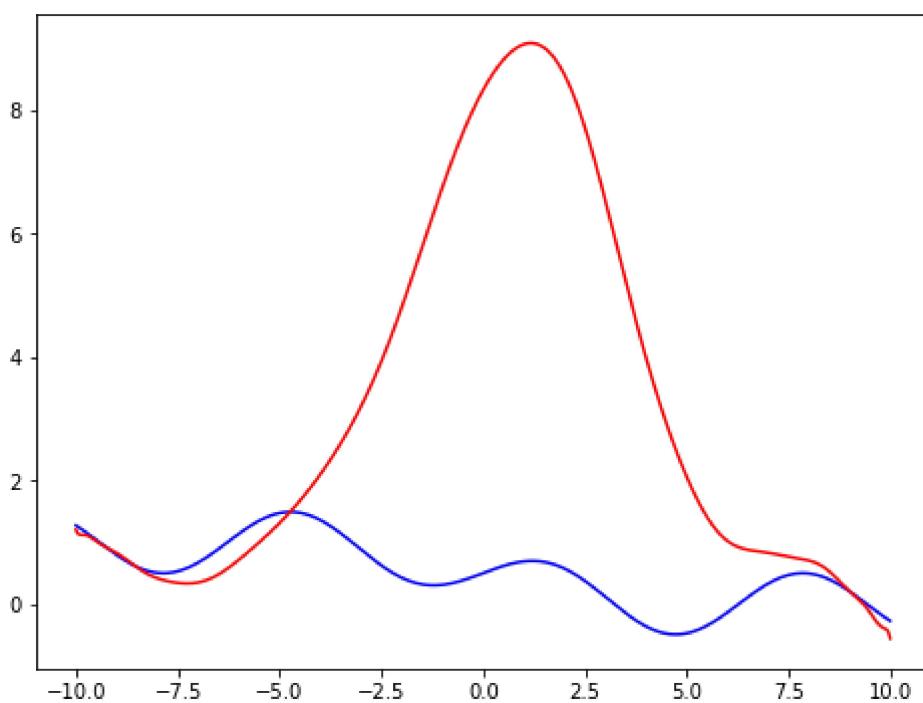
```
*****  
## [RESULT 09]  
*****
```



```
*****  
## [RESULT 10]  
*****
```



```
*****  
## [RESULT 11]  
*****
```



```
*****  
## [RESULT 12]  
*****
```

```
loss = [0.05269781]  
*****
```

```
## [RESULT 13]  
*****
```

```
loss = [1.2082059e-06]  
*****
```

```
## [RESULT 14]  
*****
```

```
loss = [0.06379867]  
*****
```

```
## [RESULT 15]  
*****
```

```
loss = [0.00844493]
```

