```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd


parameters = {"axes.labelsize": 20, "axes.titlesize": 30, 'xtick.labelsize': 12, "ytick.labelsize": 12, "legend.fontsize": 12}

plt.rcParams.update(parameters)


'''파일 받아오기'''

M0 = pd.read_csv('C:₩₩Users₩₩kim07₩₩Downloads₩₩lin_regression_data_01.csv', header=None)  # 데이터파일 불러오기

M0 = M0.to_numpy(dtype=float)
# numpy array 형식으로 변환



x_vector = M0[:, 0]                                                                    # M
matrix의 1번째 column성분들을 x_vector에 넣는다.

y_vector = M0[:, 1]                                                                     # M
matrix의 2번째 column성분들을 y_vector에 넣는다.

x_dummy = np.ones([50, 1]).reshape(len(x_vector), 1)


y_vector = np.reshape(y_vector, [50, 1])



'''by polynominal BSF analystic solution 구하는 함수'''

def PBSF_ANALYSTIC_SOLUTION(k):
```

```
    x_pbsf_matrix = x_vector                                                    #
x_pbsf_matrix 1행 설정
    for i in range(2, k + 1):
        x_pbsf_matrix = np.column_stack([x_pbsf_matrix, x_vector**i])           # Low
방향으로 k제곱 한것 쌓기

    x_pbsf_matrix = np.column_stack([x_pbsf_matrix, x_dummy])                   #
dummy data 추가
    x_pbsf_matrix_T = np.transpose(x_pbsf_matrix)                               #
transpose

    w = np.dot(np.dot(np.linalg.inv(np.dot(x_pbsf_matrix_T,
x_pbsf_matrix)),x_pbsf_matrix_T), y_vector)    # analystic solution
    return w, x_pbsf_matrix

k_list = [3, 4, 5, 6, 7, 8, 9, 10]                                              #k_list
MSE_pbsf_hist = np.array([]).reshape(-1,1)

# x_plot 데이터
x_pbsf_plot_start = np.min(x_vector) - 1
x_pbsf_plot_end = np.max(x_vector) + 1
x_pbsf_step = 0.1
x_pbsf_plot_pre = np.arange(x_pbsf_plot_start, x_pbsf_plot_end, x_pbsf_step)
x_pbsf_plot_pre = np.reshape(x_pbsf_plot_pre, [len(x_pbsf_plot_pre), 1])

weight_table = {}                                                               #표만들기
```

위한 weight table, 표만드는 것은 gpt 참고함

```
for k in (k_list):                                                    #k_list의 요소들 반복
    w_pbsf, x_pbsf = PBSF_ANALYSTIC_SOLUTION(k)
    y_pbsf_hat = np.dot(x_pbsf, w_pbsf)



    weight_table[f'k = {k}'] = w_pbsf.flatten()

    #예측용 데이터 생성(x, y)
    x_pbsf_plot = x_pbsf_plot_pre
    for i in range(2, k + 1):
        x_pbsf_plot = np.column_stack([x_pbsf_plot, x_pbsf_plot_pre ** i])

    x_pbsf_dummy_plot = np.ones(len(x_pbsf_plot))
    x_pbsf_dummy_plot = np.reshape(x_pbsf_dummy_plot, [len(x_pbsf_dummy_plot), 1])
    x_pbsf_plot = np.column_stack([x_pbsf_plot, x_pbsf_dummy_plot])

    y_pbsf_hat_plot = np.dot(x_pbsf_plot, w_pbsf)

    #regression 그래프
    fig = plt.figure()
    plt.scatter(x_vector, y_vector)
    plt.plot(x_pbsf_plot_pre, y_pbsf_hat_plot, 'r')
    plt.xlabel('weight(g)')
```

```
        plt.ylabel('length(cm)')
        plt.title('Regression graph')
        plt.legend(['data', 'Regression line'])
        plt.grid(True)

        MSE_pbsf = np.mean((y_pbsf_hat - y_vector) ** 2)

        MSE_pbsf_hist = np.append(MSE_pbsf_hist, MSE_pbsf)

for k_val in k_list:
        w = weight_table[f'k = {k_val}']
        print(f'₩nWeights for k = {k_val}:')
        for i, weight in enumerate(w):
                print(f'w{i}: {weight:.4f}')

#MSE 그래프
fig = plt.figure()
plt.plot(k_list, MSE_pbsf_hist)
plt.xlabel('k')
plt.ylabel('MSE')
plt.grid(True)
```

''' by gaussian BSF analystic solution 구하는 함수'''

```
def GBSF_ANALYSTIC_SOLUTION(K):

    x_gbsf_matrix = []

    k = np.arange(K)
#평균과 곱할 k 생성

    mu = np.min(x_vector) + ((np.max(x_vector) - np.min(x_vector)) / (K - 1)) * k     #평균
값 생성

    mu = np.reshape(mu, [len(mu), 1])
#평균 array size 조정

    sigma = (np.max(x_vector) - np.min(x_vector)) / (K - 1)                                  #분
산값


    #기저함수 구하기

    basis = np.zeros([len(x_vector), K])

    for i in range(K):

        basis[:, i] = np.exp(-0.5 * ((x_vector - mu[i]) / sigma) ** 2)



    x_gbsf_matrix = basis

    x_gbsf_matrix = np.column_stack([x_gbsf_matrix, x_dummy])  # dummy data 추가


    x_gbsf_matrix_T = np.transpose(x_gbsf_matrix)  # transpose


    w = np.dot(np.dot(np.linalg.inv(np.dot(x_gbsf_matrix_T,
x_gbsf_matrix)),x_gbsf_matrix_T), y_vector)     # analystic solution

    return w, x_gbsf_matrix
```

```
k_list = [3, 4, 5, 6, 7, 8, 9, 10]
MSE_gbsf_hist = np.array([]).reshape(-1,1)

# x_plot 예측용 데이터 생성
x_gbsf_plot_start = np.min(x_vector) - 1
x_gbsf_plot_end = np.max(x_vector) + 1
x_gbsf_step = 0.1
x_gbsf_plot_pre = np.arange(x_gbsf_plot_start, x_gbsf_plot_end, x_gbsf_step)



weight_table = {}
for k in (k_list):
    K = k
    w_gbsf, x_gbsf = GBSF_ANALYSTIC_SOLUTION(k)
    y_gbsf_hat = np.dot(x_gbsf, w_gbsf)
    x_gbsf_plot = []

    weight_table[f'k = {k}'] = w_gbsf.flatten()



    k = np.arange(K)
    mu = np.min(x_gbsf_plot_pre) + ((np.max(x_gbsf_plot_pre) - np.min(x_gbsf_plot_pre))
/ (K - 1)) * k
    mu = np.reshape(mu, [len(mu), 1])
```

```
sigma = (np.max(x_gbsf_plot_pre) - np.min(x_gbsf_plot_pre)) / (K - 1)


basis = np.zeros([len(x_gbsf_plot_pre), K])
for i in range(K):
    basis[:, i] = np.exp(-0.5 * ((x_gbsf_plot_pre - mu[i]) / sigma) ** 2)


x_gbsf_plot = basis


x_gbsf_dummy_plot = np.ones(len(x_gbsf_plot_pre))
x_gbsf_dummy_plot = np.reshape(x_gbsf_dummy_plot, [-1, 1])


x_gbsf_plot = np.column_stack([x_gbsf_plot, x_gbsf_dummy_plot])




y_gbsf_hat_plot = np.dot(x_gbsf_plot, w_gbsf)


fig = plt.figure()
plt.scatter(x_vector, y_vector)
plt.plot(x_gbsf_plot_pre, y_gbsf_hat_plot, 'r')
plt.xlabel('weight(g)')
plt.ylabel('length(cm)')
plt.title('Regrassion graph')
plt.legend(['data', 'Regrassion line'])
plt.grid(True)
```

```
        MSE_gbsf = np.mean((y_gbsf_hat - y_vector) ** 2)

        MSE_gbsf_hist = np.append(MSE_gbsf_hist, MSE_gbsf)

for k_val in k_list:
        w = weight_table[f'k = {k_val}']
        print(f'₩nWeights for k = {k_val}:')
        for i, weight in enumerate(w):
                print(f'w{i}: {weight:.4f}')

fig = plt.figure()
plt.plot(k_list, MSE_gbsf_hist, '-o')
plt.xlabel('k')
plt.ylabel('MSE')
plt.grid(True)



''' by gaussian BSF GDM으로 weight 구하는 함수'''
def GBSF_GDM(K, epoch, lr):
        '''gaussian basis function 구현 '''
        x_gbsf_matrix = []
        k = np.arange(K)
        mu = np.min(x_vector) + ((np.max(x_vector) - np.min(x_vector)) / (K - 1)) * k
```

```
mu = np.reshape(mu, [len(mu), 1])
sigma = (np.max(x_vector) - np.min(x_vector)) / (K - 1)

basis = np.zeros([len(x_vector), K])
for i in range(K):
    basis[:, i] = np.exp(-0.5 * ((x_vector - mu[i]) / sigma) ** 2)



x_gbsf_matrix = basis
x_gbsf_matrix = np.column_stack([x_gbsf_matrix, x_dummy])  # dummy data 추가

'''경사하강법 이용한 w 구하기'''

for i in range(epoch):
    if i == 0:
        w = np.random.rand(K+1) * 5

    y_hat = np.dot(x_gbsf_matrix, w).reshape([len(y_vector), 1])
    error = y_hat - y_vector

    MSE = np.mean(error ** 2)

    w_dif = sum(2 * error * x_gbsf_matrix)/len(y_hat)
    w = w - lr * w_dif
```

```
    return w, x_gbsf_matrix



#k_list, epoch, lr 설정
k_list = [3, 4, 5, 6, 7, 8, 9, 10]
epoch = 6000
lr = 0.05

MSE_gbsf_hist = np.array([]).reshape(-1,1)

# x_plot
x_gbsf_plot_start = np.min(x_vector) - 1
x_gbsf_plot_end = np.max(x_vector) + 1
x_gbsf_step = 0.1
x_gbsf_plot_pre = np.arange(x_gbsf_plot_start, x_gbsf_plot_end, x_gbsf_step)



weight_table = {}
for k in (k_list):
    K = k
    w_gbsf, x_gbsf = GBSF_GDM(k, epoch, lr)
    y_gbsf_hat = np.dot(x_gbsf, w_gbsf)
    x_gbsf_plot = []
```

```
    weight_table[f'k = {k}'] = w_gbsf.flatten()


    k = np.arange(K)
    mu = np.min(x_gbsf_plot_pre) + ((np.max(x_gbsf_plot_pre) - np.min(x_gbsf_plot_pre))
/ (K - 1)) * k
    mu = np.reshape(mu, [len(mu), 1])
    sigma = (np.max(x_gbsf_plot_pre) - np.min(x_gbsf_plot_pre)) / (K - 1)


    basis = np.zeros([len(x_gbsf_plot_pre), K])
    for i in range(K):
        basis[:, i] = np.exp(-0.5 * ((x_gbsf_plot_pre - mu[i]) / sigma) ** 2)


    x_gbsf_plot = basis


    x_gbsf_dummy_plot = np.ones(len(x_gbsf_plot_pre))
    x_gbsf_dummy_plot = np.reshape(x_gbsf_dummy_plot, [-1, 1])


    x_gbsf_plot = np.column_stack([x_gbsf_plot, x_gbsf_dummy_plot])




    y_gbsf_hat_plot = np.dot(x_gbsf_plot, w_gbsf)


    fig = plt.figure()
    plt.scatter(x_vector, y_vector)
```

```
        plt.plot(x_gbsf_plot_pre, y_gbsf_hat_plot, 'r')
        plt.xlabel('weight(g)')
        plt.ylabel('length(cm)')
        plt.title('Regrassion graph')
        plt.legend(['data', 'Regrassion line'])
        plt.grid(True)

        MSE_gbsf = np.mean((y_gbsf_hat - y_vector) ** 2)

        MSE_gbsf_hist = np.append(MSE_gbsf_hist, MSE_gbsf)

for k_val in k_list:
        w = weight_table[f'k = {k_val}']
        print(f'₩nWeights for k = {k_val}:')
        for i, weight in enumerate(w):
                print(f'w{i}: {weight:.4f}')

fig = plt.figure()
plt.plot(k_list, MSE_gbsf_hist, '-o')
plt.xlabel('k')
plt.ylabel('MSE')
plt.grid(True)
```

```
M1 = pd.read_csv('C:₩₩Users₩₩kim07₩₩Downloads₩₩lin_regression_data_02.csv')
M1 = M1.to_numpy(dtype = float)

x_work3_matrix = M1[:,0:2]
y0_vector = M1[:, 2]
x0_point = M1[:, 0]
x1_point = M1[:, 1]
y0_point = M1[:, 2]



''' by gaussian BSF analystic solution 구하는 함수'''
def GBSF_3d_ANALYSTIC_SOLUTION(x, K):
    x_gbsf_matrix = []

    k = np.arange(K)
    mu = np.zeros([2, K])
    mu[0] = np.min(x[:, 0]) + ((np.max(x[:, 0]) - np.min(x[:, 0])) / (K - 1)) * k
    mu[1] = np.min(x[:, 1]) + ((np.max(x[:, 1]) - np.min(x[:, 1])) / (K - 1)) * k
    sigma = (np.max(x) - np.min(x)) / (K - 1)

    basis = np.zeros([len(x),len(x), K])

    for i in range(K):
```

```
            basis[:, 0, i] = np.exp(-0.5 * ((x[:, 0] - mu[0, i]) / sigma) ** 2)
            basis[0, :, i] = np.exp(-0.5 * ((x[:, 1] - mu[1, i]) / sigma) ** 2)

    x_gbsf_matrix = basis
    x_work3_dummy = np.ones([len(x_gbsf_matrix), len(x_gbsf_matrix), 1])
    x_gbsf_matrix = np.append(x_gbsf_matrix, x_work3_dummy, axis = 2)
    x_gbsf_matrix = np.reshape(x_gbsf_matrix, [len(x_gbsf_matrix), -1])
    x_gbsf_matrix_T = np.transpose(x_gbsf_matrix)
    w = np.dot(np.dot(np.linalg.pinv(np.dot(x_gbsf_matrix_T,
x_gbsf_matrix)),x_gbsf_matrix_T), y0_vector)

    y = np.dot(x_gbsf_matrix, w)
    return w, x_gbsf_matrix

k_list = [3, 6, 8]
MSE_gbsf_hist = np.array([]).reshape(-1,1)

for k in (k_list):
    w_gbsf, x_gbsf = GBSF_3d_ANALYSTIC_SOLUTION(x_work3_matrix, k)
    y_gbsf_hat = np.dot(x_gbsf, w_gbsf)

    MSE_gbsf = np.mean((y_gbsf_hat - y0_vector) ** 2)

    MSE_gbsf_hist = np.append(MSE_gbsf_hist, MSE_gbsf)
```

```
fig = plt.figure()

plt.plot(k_list, MSE_gbsf_hist, '-o')

plt.xlabel('k')

plt.ylabel('MSE')

plt.grid(True)
```