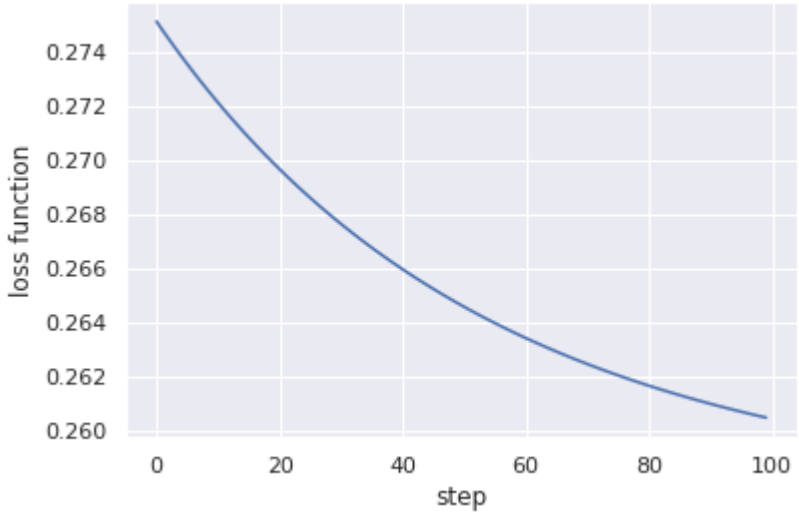
```
from sklearn.linear_model import LinearRegression
import numpy as np
from math import sqrt
import math
import seaborn as sns; sns.set()
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import make_pipeline

def BGD():
  global num_train, num_test
  #initialize
  w_coef_bgd = np.zeros(num_basis_function)
  loss_bgd = np.zeros((iteration_num, ))
  #record the loss and w efficients
  for inter_count in range(iteration_num):
      loss_bgd[inter_count] = sum((np.dot(X_train, w_coef_bgd) - T_train) ** 2) / num_train
      w_coef_bgd = w_coef_bgd - np.dot(np.transpose(X_train), np.dot(X_train, w_coef_bgd) - T_train)*2 / size * 0.01
  #compute the error
  x_bgd = np.linspace(0, 10, size)
  t_learned_bgd = []
  for i in x_bgd:
      t_learned_bgd.append(np.dot(w_coef_bgd, get_features(i)))
  error_bgd = sqrt(sum((np.dot(X_test, w_coef_bgd) - T_test) ** 2) / num_test)

  return loss_bgd, x_bgd, t_learned_bgd, w_coef_bgd, error_bgd

def SGD():
  global num_train, num_test
  #initialize
  w_coef_sgd = np.zeros(num_basis_function)
  loss_sgd = np.zeros((iteration_num, ))
  #record the loss and w efficients
  for inter_count in range(iteration_num):
```

```
    for inter_count in range(iteration_num):
        loss_sgd[inter_count] = sum((np.dot(X_train, w_coef_sgd) - T_train) ** 2) / num_train
        random_id = np.random.randint(0, num_train)
        w_coef_sgd = w_coef_sgd - 2/(inter_count+1) * (np.dot(X_train[random_id], w_coef_sgd) - T_train[random_id]) * X_trai
    #compute the error
    x_sgd = np.linspace(0, 10, size)
    t_learned_sgd = []
    for i in x_sgd:
      t_learned_sgd.append(np.dot(w_coef_sgd, get_features(i)))
    error_sgd = sqrt(sum((np.dot(X_test, w_coef_sgd) - T_test) ** 2) / num_test)

    return loss_sgd, x_sgd, t_learned_sgd, w_coef_sgd, error_sgd

def MAX_LIKE():
    w_coef_maxlike = np.dot(np.linalg.inv(np.dot(np.transpose(X_train), X_train)), np.dot(np.transpose(X_train), T_train))
    #compute the error
    x_maxlike = np.linspace(0, 10, size)
    t_learned_maxlike = []
    for i in x_maxlike:
      t_learned_maxlike.append(np.dot(w_coef_maxlike, get_features(i)))
    error_maxlike = sqrt(sum((np.dot(X_test, w_coef_maxlike) - T_test) ** 2) / len(T_test))

    return x_maxlike, t_learned_maxlike, w_coef_maxlike, error_maxlike

#get the feature by basis functions
def get_features(x):
    return np.array([1, math.sin(x), math.sin(5 * x), math.cos(x), math.cos(5 * x),])

def show_result(x, t, xlabel, ylabel):
  plt.plot(x, t)
  plt.scatter(raw_features[idx_test], T_test, alpha= 0.6)
  plt.xlabel(xlabel)
  plt.ylabel(ylabel)
  plt.show()

def show_loss(loss):
  plt.plot(range(0, iteration_num), loss)
  plt.xlabel("step")
```

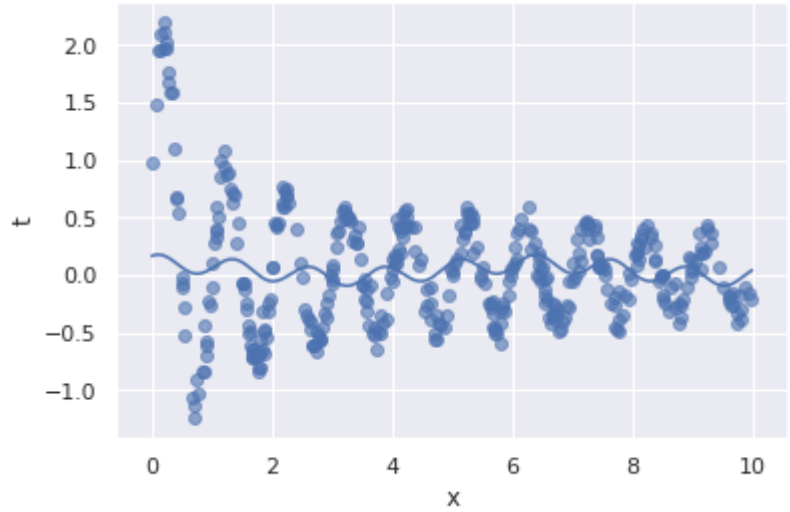
```
    plt.ylabel("loss function")
    plt.show()


if __name__ == '__main__':
    data_url = 'https://raw.githubusercontent.com/DanielXYee/Dataset/master/regression_data.csv'
    regression_data = pd.read_csv(filepath_or_buffer= data_url)
    first_column = regression_data[regression_data.columns[0]]
    second_column = regression_data[regression_data.columns[1]]
    raw_features = first_column.to_numpy()
    labels = second_column.to_numpy()

#Hyper-parameters
    iteration_num = 100
    num_basis_function = 5
    size = 2000

    features = []
    for i in raw_features:
      features.append(get_features(i))
    X_train, X_test, T_train, T_test, idx_train, idx_test = train_test_split(features, labels, np.arange(size), test_size=0.
    num_train = len(T_train)
    num_test = len(T_test)

#BGD
    print("=========BGD Starts==========")
    loss_bgd, x_bgd, t_learned_bgd, w_coef_bgd, error_bgd = BGD()
    show_loss(loss_bgd)
    print("The BGD's w coefficients is:", w_coef_bgd)
    show_result(x_bgd, t_learned_bgd, "x", "t")
    print("The BGD's error is", error_bgd)
    print("=========BGD Ends==========")

    print("============================")
    print("============================")
    print("============================")

#SGD
```

```
    print("=========SGD Starts==========")
    loss_sgd, x_sgd, t_learned_sgd, w_coef_sgd, error_sgd = SGD()
    show_loss(loss_sgd)
    print("The SGD's w coefficients is:", w_coef_sgd)
    show_result(x_sgd, t_learned_sgd, "x", "t")
    print("The SGD's error is", RMSE_sgd)
    print("=========SGD Ends==========")

    print("============================")
    print("============================")
    print("============================")

  #MAX_LIKE
    print("=========MAX_LIKE Starts==========")
    x_maxlike, t_learned_maxlike, w_coef_maxlike, RMSE_maxlike = MAX_LIKE()
    print("The MAX_LIKE's w coefficients is:", w_coef_maxlike)
    show_result(x_maxlike, t_learned_maxlike, "x", "t")
    print("The MAX_LIKE's error is", RMSE_maxlike)
    print("=========MAX_LIKE Ends==========")
```

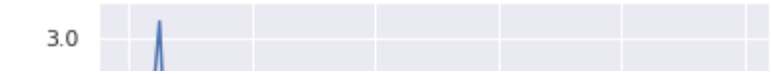
```
=========BGD Starts==========
```

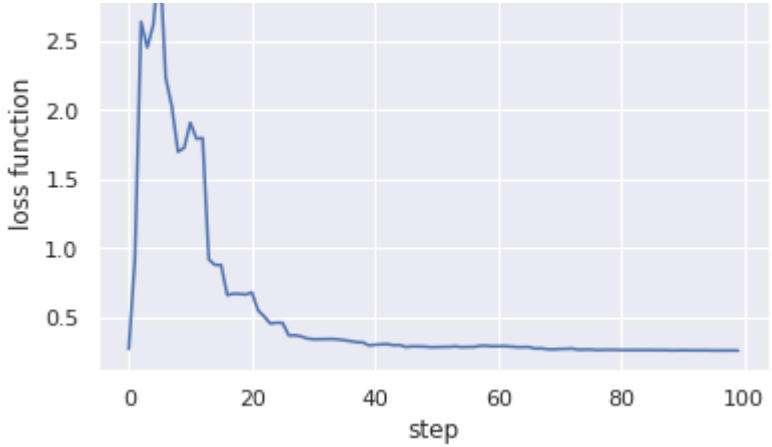



```
The BGD's w coefficients is: [0.04000783 0.0065772  0.04044914 0.05775453 0.06580118]
```

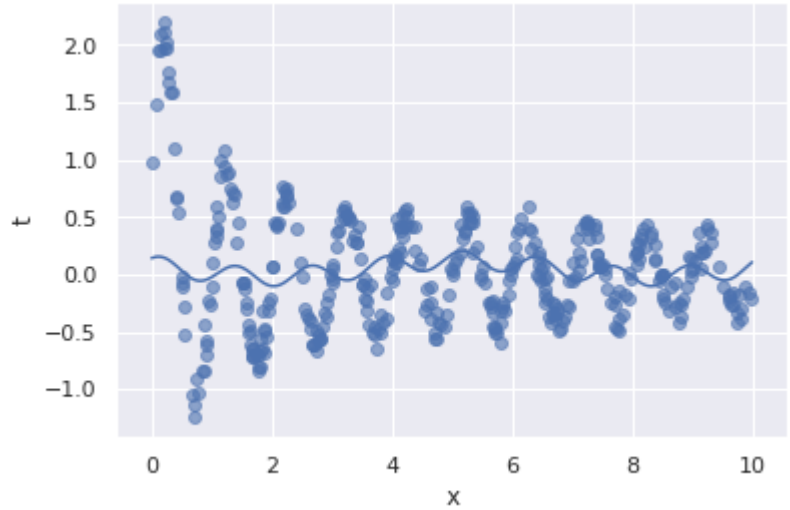



```
The BGD's error is 0.516793908682939
=========BGD Ends==========
==========================
==========================
==========================
=========SGD Starts==========
```

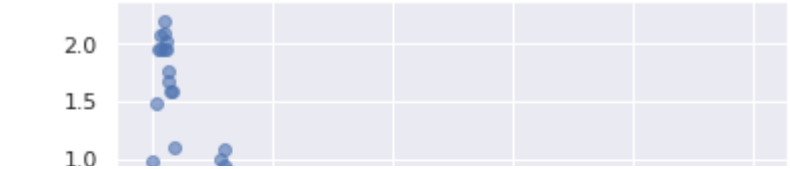


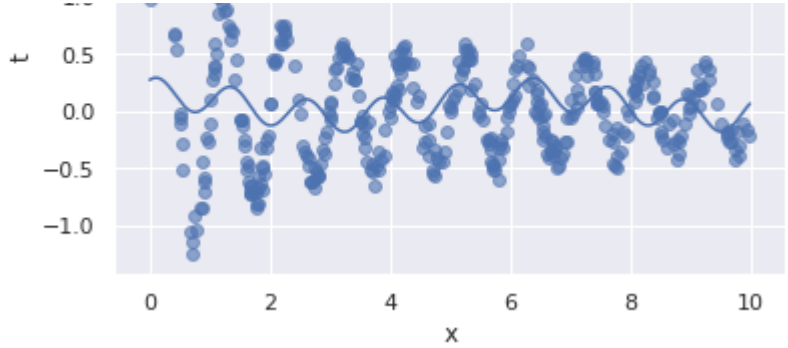



```
The SGD's w coefficients is: [ 0.0506901  -0.06585775  0.05547216  0.02970167  0.06047402]
```




```
The SGD's error is 0.5443396782264754
=========SGD Ends==========
===========================
===========================
===========================
=========MAX_LIKE Starts==========
The MAX_LIKE's w coefficients is: [ 0.05541403 -0.001529    0.06978902  0.10077781  0.11706053]
```






```
The MAX_LIKE's error is 0.5128222642922083
=========MAX_LIKE Ends==========
```

According to the root-mean-square error, the Maximum likelihood method has the lowest error. I think the maximum likelihood has the best performance in this experiment.