Homework 1

李翰韬 16711094 160324

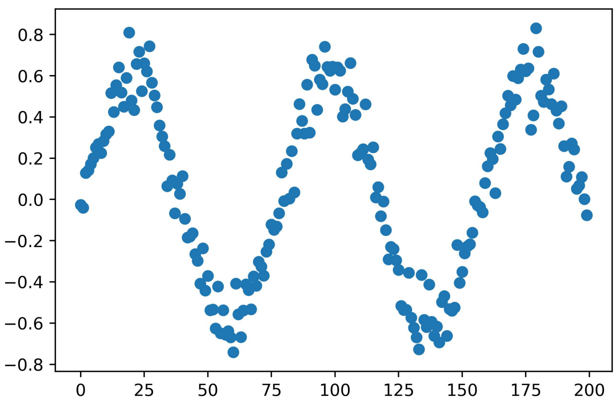
I. Introduction

Given the following one-dimensional time series, try to learn what functional relationship the series and time t are satisfying, fit, and find the corresponding coefficients.

```
[-0.02583269 -0.04051301  0.12799861  0.14173928  0.17152477  0.2011345   0.2514499
 0.26942975  0.22501756  0.28285966  0.31559904  0.32937735  0.51564061  0.42471355
 0.55419245  0.64080295  0.51817686  0.44927417  0.5889517   0.80962372  0.48032986
 0.43297626  0.6569534   0.71691291  0.52587894  0.66087249  0.62079964  0.74368168
 0.56661974  0.50504992  0.44729499  0.35939382  0.30583365  0.25840365  0.06450455
 0.21688696  0.09274139 -0.06697476  0.07592577  0.02670233  0.11388184 -0.09379051
 -0.18439572 -0.17966559 -0.16315278 -0.26450209 -0.29813998 -0.40768861 -0.23685474
 -0.4417905  -0.37100265 -0.53797068 -0.53370135 -0.62568814 -0.42158101 -0.64884222
 -0.53725013 -0.65579792 -0.63923729 -0.6685564 -0.74057211 -0.40855252 -0.55777798
 -0.66742682 -0.5380056  -0.41152255 -0.43906442 -0.53312085 -0.37285735 -0.41915559
 -0.30230357 -0.32560979 -0.36956646 -0.25278871 -0.21815953 -0.12125635 -0.14723625
 -0.1316458 -0.06763261  0.13135476 -0.00786488  0.17251539  0.00390189  0.23420464
 0.03424298  0.31921242  0.46289181  0.38151237  0.31969485  0.55720336  0.32361329
 0.67844812  0.64842566  0.43472184  0.58165945  0.55901493  0.74121444  0.6430677
 0.6243773   0.64464953  0.53233577  0.64055675  0.62415506  0.40256586  0.43861789
 0.52358594  0.66217108  0.4877558  0.41027396  0.21449227  0.2245596   0.24329612
 0.46112313  0.19387336  0.17067689  0.25329545  0.01034272  0.05951573 -0.08156016
 -0.01039697 -0.14914629 -0.29037328 -0.22990902 -0.24089382 -0.29532118 -0.34132541
 -0.51587605 -0.53594197 -0.53559808 -0.35593396 -0.57340654 -0.62218798 -0.66848189
 -0.72710317 -0.36642848 -0.58384893 -0.61940538 -0.41302691 -0.59320198 -0.66224973
 -0.61647815 -0.69304147 -0.49728935 -0.46833321 -0.66194989 -0.5311272  -0.53975736
 -0.52416066 -0.22056537 -0.40471053 -0.3511647  -0.26158341 -0.22755895 -0.21753077
 -0.16181743 -0.00850077 -0.02816639 -0.03632056 -0.06225192  0.07980789  0.16114784
 0.22501789  0.19651355  0.03039382  0.3050647   0.24537894  0.36461378  0.41820554
 0.50340963  0.45723135  0.59927635  0.48420893  0.58802576  0.62976302  0.72999823
 0.62190162  0.63555945  0.33838519  0.40757477  0.82995173  0.71600373  0.50335912
 0.47322202  0.5808286   0.53369145  0.46285591  0.61058995  0.4309896   0.36806185
 0.45229756  0.25896206  0.11139308  0.15846649  0.27199323  0.2427336   0.05156092
 0.06794632  0.10848514  0.00216508 -0.07673958]
```

II. Analysis

Firstly, use a scatter plot to plot the raw data. Try to observe the data characteristics.



It can be found that the data is roughly distributed as a ***sine*** function with an initial phase of zero. For the ***sine*** function, we can use two methods to fit it:

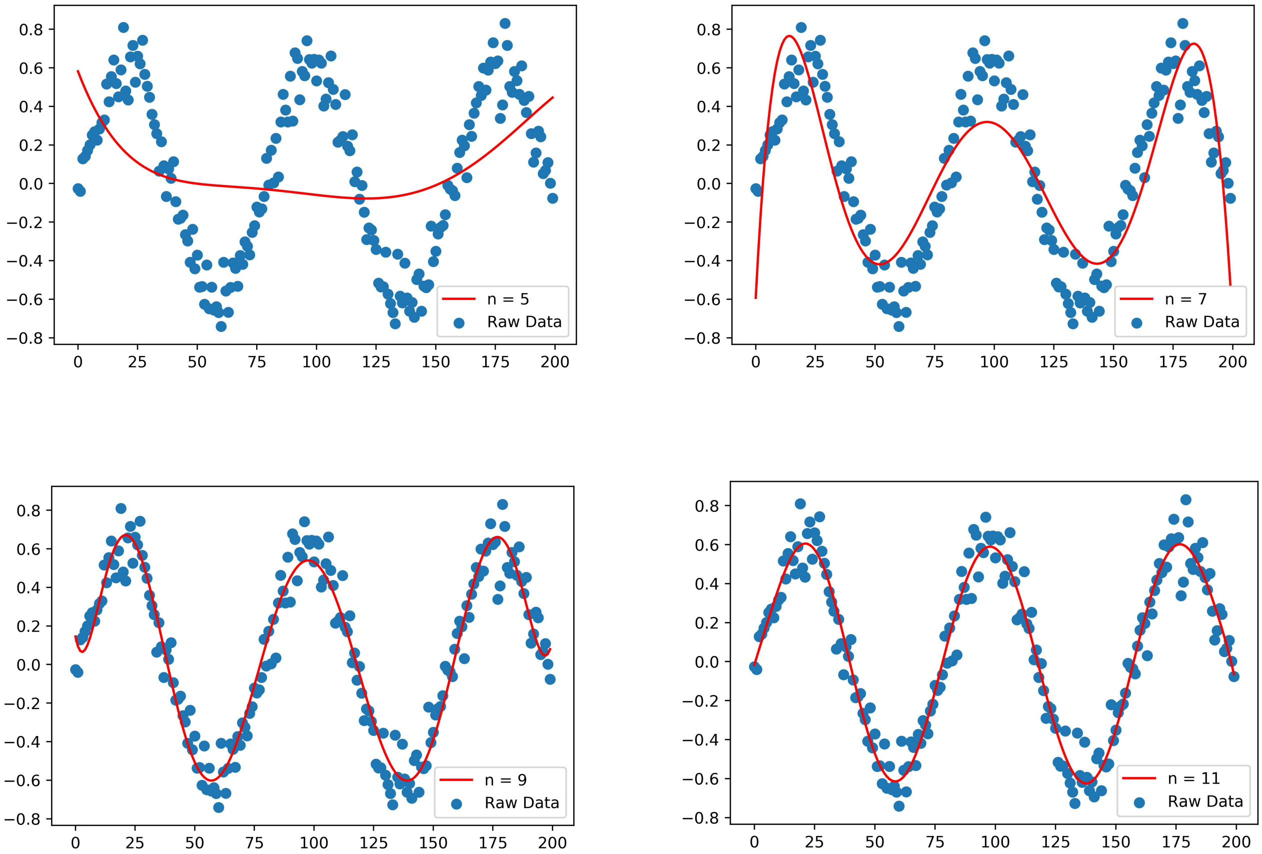
III. Solution

1). Polynomial fitting

First of all, Python and any other programming languages have very convenient high-order polynomial fitting functions. We can use the **Taylor expansion formula** to expand the ***sine*** function into a higher-order polynomial and use the fitting function to fit it.

$$\sin x = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 + o\left(x^5\right)$$

In Python, we can use the fitting function that comes with the NumPy library to perform a polynomial fitting the data with ***n*** times. Regarding the choice of ***n***, several trial attempts were made, and the following results were obtained:



It can be found that the function fit well when ***n=11***. Thus, choose ***n=11*** and the variable coefficients are:

a_i=[4.33631470e-22 -8.41654915e-19 5.82169643e-16 -2.04104271e-13 4.07457672e-11 -4.76696779e-09 3.16050230e-07 -1.05395171e-05 1.28947614e-04 -8.01682959e-04 4.21023294e-02 -1.77141682e-02]

However, it must be clear that the method of polynomial fitting is only applicable to the two hundred raw data points given in the task, and is not applicable to continuous functions in the entire time domain.

In addition, the actual sampling frequency of the raw data is not introduced in this analysis. If the data sampling frequency is very high, the accuracy of the coefficient requirements in polynomial fitting is further improved, which is a considerable challenge for the general computer system.

2). Gradient descent

We hope to use the machine learning algorithm to find the function closest to the sequence, and consider using the gradient descent algorithm to optimize the error function.

Assume that the function is:

$$f(t) = A\sin(\omega t)$$

The gradient descent method that defines the loss function:

$$L = \frac{1}{2N}\sum_{i=1}^{N}(f^{i}(t) - y(t))^2$$

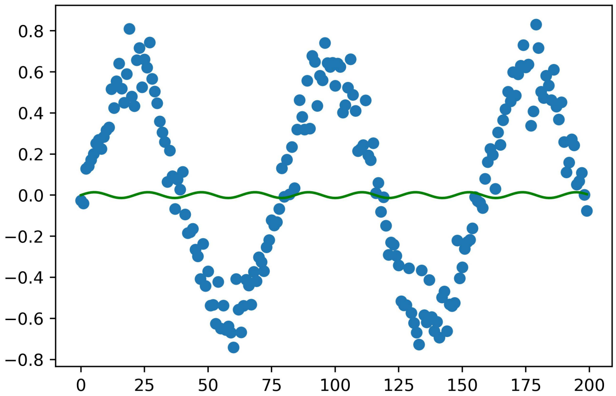
The gradient descent method that defines the various derivatives of the derivation function:

$$A_{i+1} = A_i - r\frac{\partial L}{\partial A} = A_i - \frac{r}{N}\sum \sin(\omega t)\Delta y$$

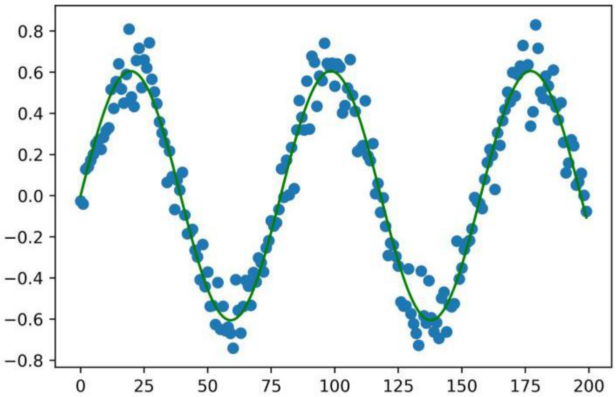
$$\omega_{i+1} = \omega_i - r\frac{\partial L}{\partial \omega} = \omega_i - \frac{r}{N}\sum At\cos(\omega t)\Delta y$$

Where ***r*** is the learning rate.

We can clearly observe from the plot that the amplitude of the function is about 0.7, and the angular velocity is about 0.078. If a random function is used to select the initial values of A and ω randomly, it is found that the fitting effect is variable. Sometimes the correct result can be successfully fitted, and sometimes the fitting result is obviously wrong, like the picture below.



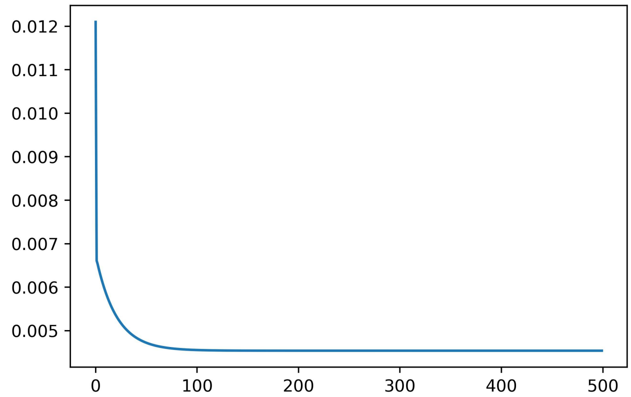
If the estimated values of A and w are directly set, that is, A=0.7, ω=0.078, the correct fitting result can be obtained.



We can obtained that:

$$A \approx 0.6056$$
$$\omega \approx 0.07982$$

The graph of the loss function is as follows:



After a simple analysis, we can know that the previous failure to fit function is due to the loss function entering the local minimum and unable to continue to optimize, resulting in overfitting the function fits within a small range, thereby invalidating the entire gradient descent process.

If you want to avoid this situation, the estimated parameters used in this experiment are straightforward but may not be used in some complicated cases. We can also use algorithms to prevent the loss function into the local optimum, such as the annealing algorithm.

IV. Code

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 25 15:34:27 2020

@author: hantao.li
"""

import matplotlib.pyplot as plt
import numpy as np
import random

Data = '...'
Data = list(map(float,Data.split()))
x = np.linspace(0,len(Data)-1,len(Data))

#plt.figure()
#plt.scatter(x,Data)
#plt.savefig('./TASK1/Raw.jpg',dpi=500)
'''

n_order = 11
p = np.poly1d(np.polyfit(x, Data, n_order))
print(p.coeffs)
plt.plot(x, p(x), color = 'red')
plt.legend(['n = '+str(n_order),'Raw Data'], loc='lower right')
#plt.savefig('./TASK1/n_'+str(n_order)+'.jpg',dpi=500)
'''

omega=random.random()
A=random.random()

omega=0.078
A=0.7

eta=0.05
time=500
Loss_1=np.zeros(time)

def loss_function():
    L=0
    for i in range(len(x)):
```

```
        L=L+(A*np.sin(omega*x[i])-Data[i])**2
    L = L/400
    return L

def div_omega():
    D_o=0
    for i in range(len(x)):

D_o=D_o+(A*np.sin(omega*x[i])-Data[i])*A*np.cos(o
mega*x[i])
    D_o = D_o/200
    return D_o

def div_A():
    D_A=0
    for i in range(len(x)):

D_A=D_A+(A*np.sin(omega*x[i])-Data[i])*np.sin(ome
ga*x[i])
    D_A = D_A/200
    return D_A

def gradient_descent(cur,eta,d):
    return cur-eta*d

for j in range(time):
    Loss_1[j]=loss_function()
    pre=A*np.sin(omega*x)
    omega=gradient_descent(omega, eta, div_omega())
    A=gradient_descent(A, eta, div_A())
pre=A*np.sin(omega*x)

#plt.figure()
#plt.scatter(x,Data)
#plt.plot(x, pre, color = 'green')
#plt.legend(['n = '+str(n_order),'Raw Data'], loc='lower
right')
#plt.savefig('./TASK1/g.jpg',dpi=500)

x_l = np.linspace(0,len(Loss_1)-1,len(Loss_1))
plt.figure()
plt.plot(x_l, Loss_1)
```