Polynomial Curve Fitting

***Abstract*—This document contains theory involved in curve fitting.**

1 Objective

The objective is to fit best line for the polynomial curve using regularization.

2 Generate Dataset

Create a sinusoidal function of the form

$$y = A\sin 2\pi x + n(t) \tag{2.0.1}$$

n(t) is the random noise that is included in the training set. This set consists of N samples of input data i.e. x expressed as shown below

$$x = \left(x_1, x_2, .., x_N\right)^T \tag{2.0.2}$$

which give the corresponding values of y denoted as

$$y = \left(y_1, y_2, .., y_N\right)^T \tag{2.0.3}$$

The Fig 0 is generated by random values of x_n for

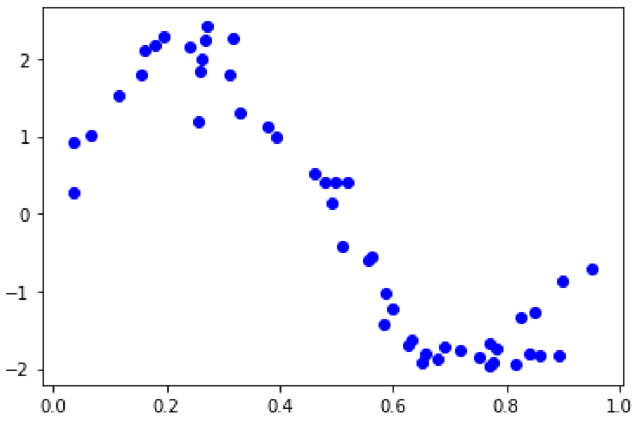

Fig. 0: Sinusoidal Dataset with added noise

n =1,2,..,N. where N=50 in the range [0,1].

The corresponding values of y were generated from the Eq (2.0.1).The first term $A\sin 2\pi x$ is computed directly and then random noise samples having a normal(Gaussian) distribution are added inorder to get the corresponding values of y.

```
#Generate the sine curve
import numpy as np
import matplotlib.pyplot as plt

N = 50
np.random.seed(20)
x = np.sort(np.random.rand(N,1),axis=0)
noise = np.random.normal(0,0.3,size=(N,1))
A = 2.5
y = A*np.sin(2*np.pi*x) + noise

plt.scatter(x,y,c='b',marker='o',label='Data with
    noise')
plt.xlabel('x');plt.ylabel('y')
```

Intialize a regularization parameter λ

```
lamda = 0.00000001522
poly_deg = 3
```

Now, add bias and higher order terms. Here we initialize column 1 with 1 and generate the input matrix as follows

```
F = np.zeros(shape = (N,poly_deg+1))
F[:,0] = 1
for i in range(1,poly_deg+1):
    F[:,i] = np.power(x,i).reshape((N,))
```

The generated matrix would look like

$$\mathbf{F} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{N-1} \\ \vdots & & \vdots & & \vdots \\ 1 & \dots & \dots & \dots & x_N^{N-1} \end{pmatrix} \tag{2.0.4}$$

3 Polynomial Curve Fitting

The goal is to find the best line that fits into the pattern of the training data shown in the graph. We shall fit the data using a polynomial function of the form,

$$y(w,x) = \sum_{j=0}^{M} w_j x^j \tag{3.0.1}$$

$$y = \hat{\mathbf{w}}\mathbf{F} \tag{3.0.2}$$

$$y = \mathbf{w}\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{N-1} \\ \vdots & & \vdots & & \vdots \\ 1 & \dots & \dots & \dots & x_N^{N-1} \end{pmatrix} \tag{3.0.3}$$

M is the order of the polynomial The polynomial coefficient are collectively denoted by the vector **w**.The proposed vector **w** of the model referring to Eq (2.0.4) is given by

$$\hat{\mathbf{w}} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T y \tag{3.0.4}$$

The values of the coefficients are determined by fitting the polynomial to the training data.This can be done by minimizing an **error function** that measures the misfit between the function y and the training set data points given as

$$\sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{M} x^j w_j\right)^2 \tag{3.0.5}$$

We add a penalty term at the end to modify the error function Eq (3.0.5)

$$\sum_{i=1}^{N}\left(y_i - \sum_{j=0}^{M} x^j w_j\right)^2 + \lambda\sum_{j=0}^{M} w_j^2 \tag{3.0.6}$$

where for some $t > 0$, $\sum_{j=1}^{M} w_j^2 < t$, Eq (3.0.6) can be further expressed as

$$(\mathbf{y} - \mathbf{wF})^T(\mathbf{y} - \mathbf{wF}) - \lambda\|\mathbf{w}\|^2 \tag{3.0.7}$$

where $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$.Expand Eq (3.0.7). Let

$$E = (\mathbf{y} - \mathbf{wF})^T(\mathbf{y} - \mathbf{wF}) - \lambda\mathbf{w}^T\mathbf{w} \tag{3.0.8}$$

$$= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{wF} - \mathbf{w}^T\mathbf{F}^T\mathbf{y} + \mathbf{w}^T\mathbf{F}^T\mathbf{Fw} + \lambda\mathbf{w}^T\mathbf{w} \tag{3.0.9}$$

$$= \mathbf{y}^T\mathbf{y} - \mathbf{w}^T\mathbf{F}^T\mathbf{y} - \mathbf{w}^T\mathbf{F}^T\mathbf{y} + \mathbf{w}^T\mathbf{F}^T\mathbf{Fw} + \mathbf{w}^T\lambda\mathbf{Iw} \tag{3.0.10}$$

$$= \mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{F}^T\mathbf{y} + \mathbf{w}^T\left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)\mathbf{w} \tag{3.0.11}$$

Evaluate **w** that minimizes E. Here, we make use of matrix differentiation rule given as

$$\frac{\partial xA^Tx}{\partial x} = (A + A^T)x = 2Ax \tag{3.0.12}$$

We get 2Ax when A is symmetric. This can be applied here as $\left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)$. From Eq (3.0.11)

$$\frac{\partial E}{\partial \mathbf{w}} = -2\mathbf{F}^T\mathbf{y} + 2\left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)\mathbf{w} = 0 \tag{3.0.13}$$

$$\implies \left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)\mathbf{w} = \mathbf{F}^T\mathbf{y} \tag{3.0.14}$$

$$\implies \mathbf{w} = \left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)^{-1}\mathbf{F}^T\mathbf{y} \tag{3.0.15}$$

The coefficient λ governs the relative importance of the regularization term compared with the sum-of-the-squares term.

First, we take a random guess of the sine curve on the input data

```
plt.plot(np.linspace(0,1,50),np.sin(2*np.linspace
    (0,1,50)*np.pi),c='g',linewidth=2,label='
    function generating input data')
```

Form the penalized residual sum of squares as follows

```
W = np.linalg.pinv((F.T.dot(F) + lamda*np.eye(
    poly_deg+1))).dot(F.T).dot(y)
```

Plot the predicted output

```
plt.plot(x,F.dot(W),'r',label='After fitting')
plt.legend()
plt.show()
```

You need to vary λ to make it work.

4 Curve fitting using scikit-learn

scikit learn is a machine learning python library which features various algorithms and is designed to interoperate with the numpy and scipy libraries. Here, we import and use linear regression to find the best fit.

```
from sklearn.preprocessing import
    PolynomialFeatures
sk_poly_deg=3
poly_feature = PolynomialFeatures(degree=
    sk_poly_deg,include_bias=False)
x_poly = poly_feature.fit_transform(x)

from sklearn.linear_model import
    LinearRegression
```

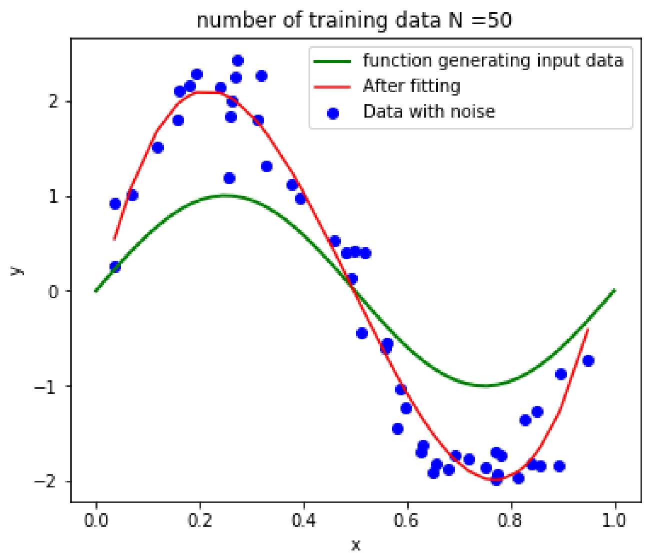

Fig. 0: Curve Fitting for $\ln\lambda = -18$

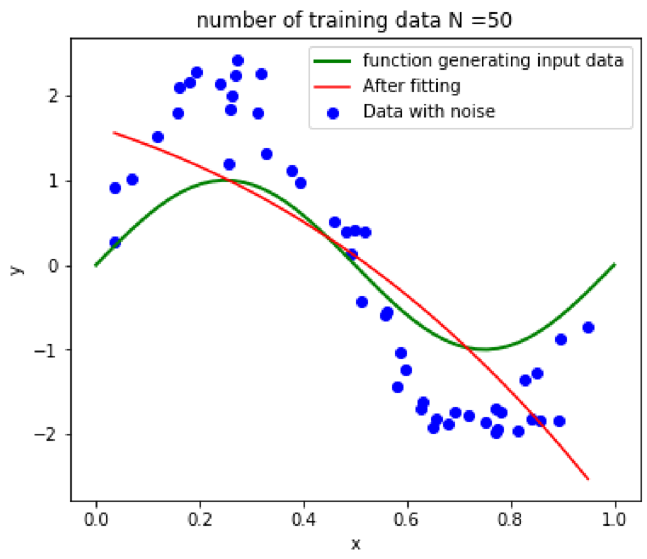

Fig. 0: Curve Fitting for $\ln\lambda = 0$

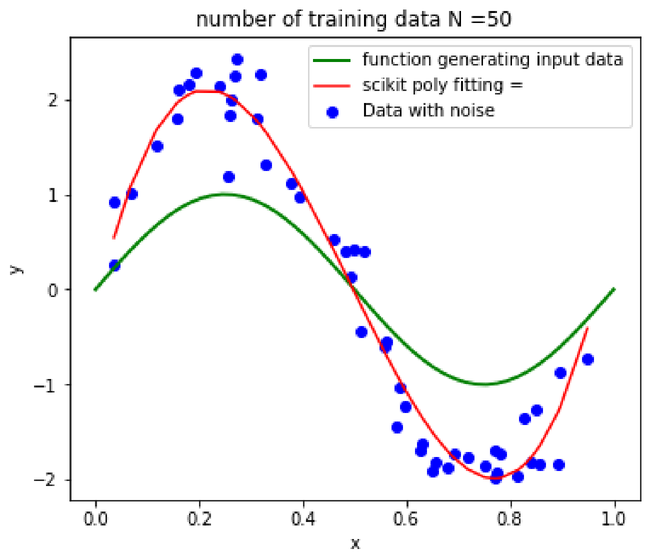

Fig. 0: Using scikit-learn

```
lin_reg=LinearRegression()
lin_reg.fit(x_poly,y)
```

Python code:

```
https://github.com/Hrithikraj2/EE4015_IDP/blob/
    main/Assignment3/
    Assignment_3_Regularization.ipynb
```