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

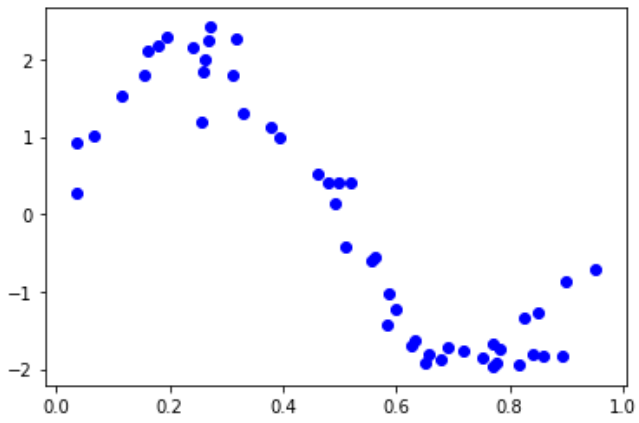

Fig. 0: Sinusoidal Dataset with added noise

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')
```

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,

$$y(w, x) = \begin{pmatrix} w_0 & w_1 & w_2 & . & ...w_N \end{pmatrix} \begin{pmatrix} 1 \\ x \\ x^2 \\ . \\ x^N \end{pmatrix} = \sum_{j=0}^{M} w_j x^j \tag{3.0.1}$$

where M is the order of the polynomial. The polynomial coefficient are collectively denoted by the vector w. Although the polynomial function y(w, x) is a nonlinear function of x, it is a linear function of the coefficients **w**. Functions, such as the polynomial, which are linear in the unknown parameters have important properties and are called **linear models**.

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 $y(x, \mathbf{w})$, for any given value of **w**, and the training set data points. The widely used error function, is given by the sum of the squares of the errors between the

predictions $y(x_n, \mathbf{w})$ for each data point x_n and the corresponding target values t_n

$$E(x) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 \qquad (3.0.2)$$

In Regularization, we add a penalty term λ to the error function (3.0.2) in order to discourage the coefficients from reaching large values. This penalty term takes the form of a sum of squares of all the coefficients, leading to a modified error function of the form

$$E(\tilde{x}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2 \qquad (3.0.3)$$

where $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + ... + w_M^2$ and the coefficient λ governs the relative importance of the regularization term compared with the sum-of-the-squares term.

4 Training the Model

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')
```

Here, we use regularization parameter λ which is used to control the overfitting phenomenon and initialize the polynomial degree.

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

Now we generate the input matrix f which is obtained from the random values of x.

```
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,))
```

Now, we use regularization and fit our model with all the parameters and function to generate the plot.

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

You need to vary λ to make it work.

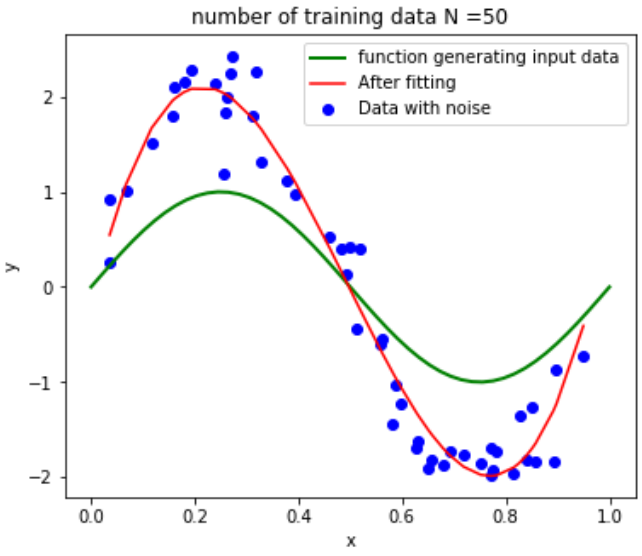

Fig. 0: Curve Fitting for $\ln\lambda = -18$

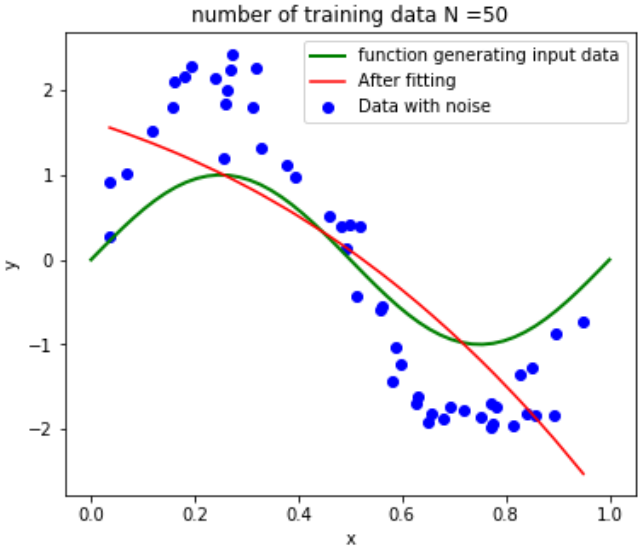

Fig. 0: Curve Fitting for $\ln\lambda = 0$

5 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
lin_reg=LinearRegression()
lin_reg.fit(x_poly,y)
```

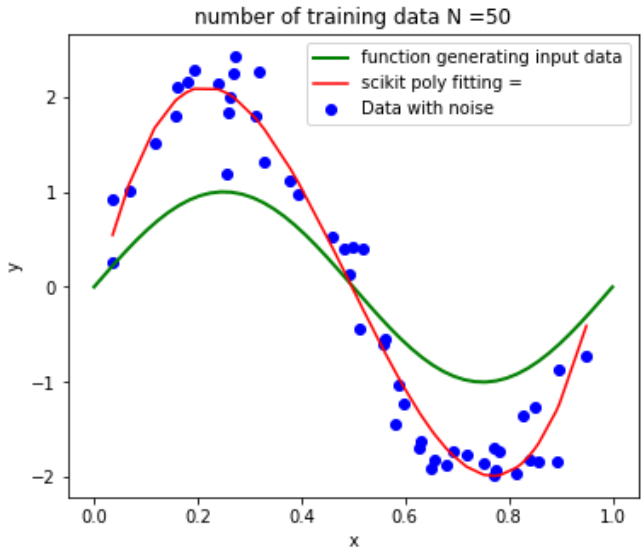

Fig. 0: Using scikit-learn

Python code:

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