# 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

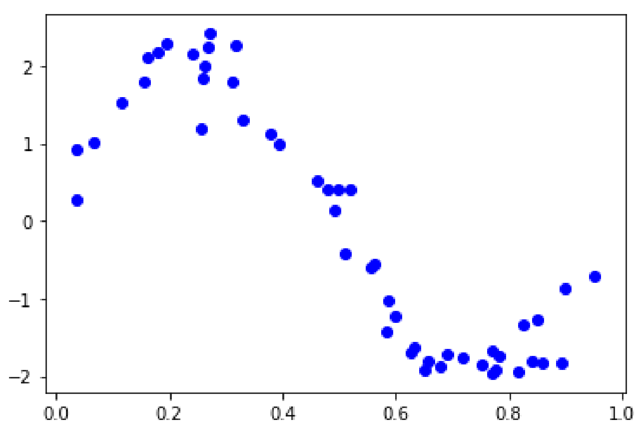


Fig. 0: Sinusoidal Dataset with added noise

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')
```

The following code generates the input matrix F

```
sk_poly_deg=3
poly_feature = PolynomialFeatures(degree=
    sk_poly_deg,include_bias=False)
F = poly_feature.fit_transform(x)
```

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,

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

$$\tag{3.0.2}$$

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

$$\hat{\mathbf{w}} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T y \tag{3.0.3}$$

Now, adding a regularization term to the error function controls the over-fitting and the total error function takes the form

$$E_D(w) + \lambda E_W(w) \tag{3.0.4}$$

One of the simplest forms of the regularizer is given by the sum of the squares of the weight vector element

$$E_W(w) = \frac{1}{2}w^T w \tag{3.0.5}$$

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. The sum-of-squares error function is given by

$$\sum_{n=1}^{N}\left(y_n - w^T f(n)\right)^2 \tag{3.0.6}$$

where F = f(n) Thus, the total error function becomes

$$\frac{1}{2}\sum_{n=1}^{N}\left(y_n - w^T f(n)\right)^2 + \frac{\lambda}{2}w^T w \tag{3.0.7}$$

Eq (3.0.7) can be further expressed as

$$(\mathbf{y} - \mathbf{wF})^T(\mathbf{y} - \mathbf{wF}) + \lambda\left\|\mathbf{w}\right\|^2 \tag{3.0.8}$$

where $\|\mathbf{w}\|^2 = \mathbf{w}^T\mathbf{w}$.Expand Eq (3.0.8). Let

$$E = (\mathbf{y} - \mathbf{wF})^T(\mathbf{y} - \mathbf{wF}) + \lambda\mathbf{w}^T\mathbf{w} \tag{3.0.9}$$

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

$$= \mathbf{y}^T\mathbf{y} - \mathbf{w}^T\mathbf{F}^T\mathbf{y} - \mathbf{w}^T\mathbf{F}^T\mathbf{y} + \mathbf{w}^T\mathbf{F}^T\mathbf{F}\mathbf{w} + \mathbf{w}^T\lambda\mathbf{I}\mathbf{w} \tag{3.0.11}$$

$$= \mathbf{y}^T\mathbf{y} - 2\mathbf{w}^T\mathbf{F}^T\mathbf{y} + \mathbf{w}^T\left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)\mathbf{w} \tag{3.0.12}$$

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

$$\frac{\partial x A^T x}{\partial x} = (A + A^T)x = 2Ax \tag{3.0.13}$$

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 (3.0.12)

$$\frac{\partial E}{\partial \mathbf{w}} = -2\mathbf{F}^T\mathbf{y} + 2\left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)\mathbf{w} = 0 \tag{3.0.14}$$

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

$$\implies \mathbf{w} = \left(\mathbf{F}^T\mathbf{F} + \lambda\mathbf{I}\right)^{-1}\mathbf{F}^T\mathbf{y} \tag{3.0.16}$$

The coefficient $\lambda$ governs the relative importance of the regularization term compared with the sum-of-the-squares term.

A more general regularizer is used for which the regularized error takes the form

$$\frac{1}{2}\sum_{n=1}^{N}\left(y_n - w^T f(n)\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{M}\left|w_j\right|^q \tag{3.0.17}$$

q = 2 corresponds to quadratic regularizer. The case of q = 1 in known as the lasso. It has the property that if $\lambda$ is sufficiently large, some of the coefficients $w_j$ are driven to zero, leading to a sparse model in which the corresponding basis functions play no role.

The error function for q = 1 (Lasso) would look like

$$\frac{1}{2}\sum_{n=1}^{N}\left(y_n - w^T f(n)\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{M}\left|w_j\right| \tag{3.0.18}$$

Regularization allows complex models to be trained on data sets of limited size without severe over-fitting, essentially by limiting the effective model complexity

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')
```

Now, we use the scikit learn and import Lasso

```
model = Lasso(alpha = 0.000005)
model.fit(F,y)
```

Plot the predicted output

```
plt.plot(x,model.predict(F),'r',label='Plot after
    Lasso =')
plt.legend()
plt.show()
```

The plot would look like Fig 0

Python code:

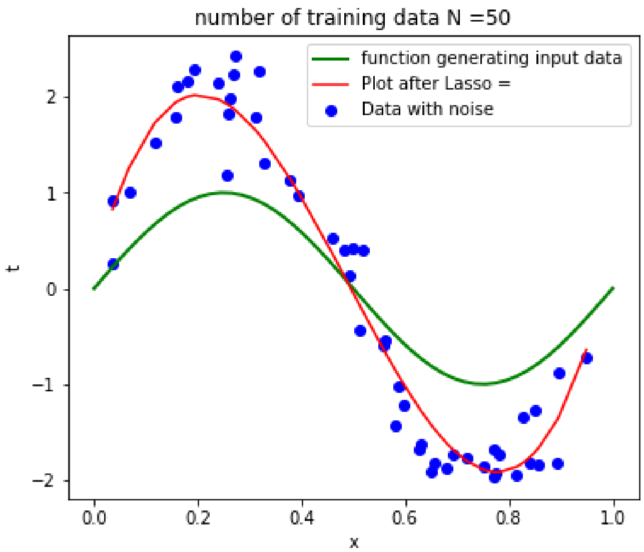


Fig. 0: Curve Fitting after Lasso regression

https://github.com/Hrithikraj2/EE4015_IDP/blob/main/Assignment_4/Assignment_4.ipynb