Polynomial Curve Fitting

***Abstract*—This document contains theory behind curve fitting model**

1 Objective

Our objective is to implement the best fit line polynomial Curve.

2 Load Dataset

Create a sinusoidal data function

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

with random noise included in the target values training set comprising N observations of t, written

$$t = \begin{pmatrix} t_1 & , & . & . & t_N \end{pmatrix}^T \tag{2.0.2}$$

together with corresponding observations of the values of y, denoted

$$y = \begin{pmatrix} y_1 & , & . & . & y_N \end{pmatrix}^T \tag{2.0.3}$$

Fig 0 was generated by choosing values of t_n, for

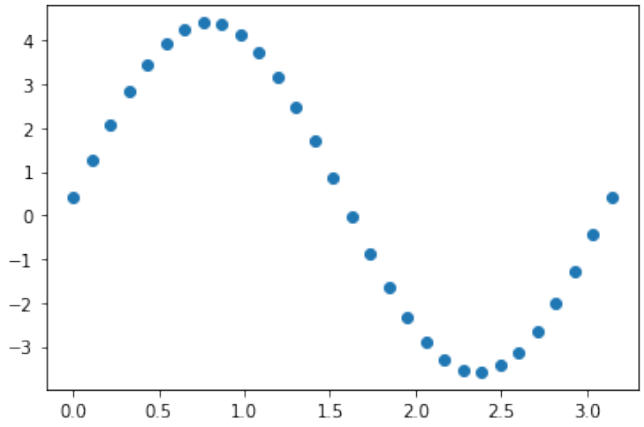

Fig. 0: Sinusoidal Dataset

n = 1, . . . , N, where N= 30 and spaced uniformly. Data set y was obtained by first computing the corresponding values of the function $A \sin 2\pi f t$ and then adding a small level of random noise having a Uniform distribution to each such point in order to obtain the corresponding value.

```
# Data Creation
poly_order = 4
# Number of training samples
N = 30
t = np.linspace(0, np.pi, N)
f=1
A=4
# Generate some numbers from the sine function
y = A*np.sin(2*f*t)
# Add noise
y += np.random.RandomState(1).uniform()
#defining it as a matrix
y_1 = np.asmatrix(y.reshape(N,1))
plt.plot(t, y, 'o');
plt.show()
```

Adding the bias and higher order terms, Here column-1 will always be the value of coefficient, which will always be 1. But to create a matrix we need to consider it as a column.

```
B = np.append(np.ones((N,1)),t.reshape((N,1)),axis
    = 1)
for i in range(0,poly_order-1):
        B = np.append(B,(t.reshape((N,1)))**(i
            +2),axis = 1)
B = np.asmatrix(B)
print(B.shape)
print(B)
```

The matrix we got from the above code,

$$\mathbf{B} = \begin{pmatrix} 1 & t_0 & t_0^2 & \dots & t_0^{N-1} \\ 1 & t_1 & t_1^2 & \dots & t_1^{N-1} \\ 1 & t_2 & t_2^2 & \dots & t_2^{N-1} \\ \vdots & & \vdots & & \vdots \\ 1 & \dots & \dots & \dots & t_N^{N-1} \end{pmatrix} \tag{2.0.4}$$

3 Polynomial Curve Fitting

To find a line that best resembles the underlying pattern of the training data shown in the graph. By using the least squares method to Parameterize our model with the coefficients that best describe the training set before seeing how well the model generalizes to data it hasn't seen before. In particular,

we shall fit the data using a polynomial function of the form, where N is the order of the polynomial.

$$y = \sum_{j=0}^{N} w_j t^j \tag{3.0.1}$$

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

$$y = \hat{\mathbf{w}} \begin{pmatrix} 1 & t_0 & t_0^2 & \dots & t_0^{N-1} \\ 1 & t_1 & t_1^2 & \dots & t_1^{N-1} \\ 1 & t_2 & t_2^2 & \dots & t_2^{N-1} \\ \vdots & & \vdots & & \vdots \\ 1 & \dots & \dots & \dots & t_N^{N-1} \end{pmatrix} \tag{3.0.3}$$

Finding the optimum weights of Model using the **Normal Equation** an analytical approach to Polynomial Regression with a Least Square Cost Function, using (2.0.4)

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

where,
$\hat{\mathbf{w}}$: is hypothesis parameters that define it the best.
B: Input feature value of each instance
y: Output value of each instance.

```
w = (B.T*B).I*B.T*y_1
print(w)
```

Multiplying the above matrix named is as y-pred,

```
y_pred = B * w
y_pred
print(y_pred.shape)
```

Now we quantify how well our model fits the data by using the least squared error (LSE) to calculate the average squared difference between our line and the actual data point in the training set.

```
def LSE_error(y_pred,y_1):
   var = (y_1-y_pred)
   n = len(var)
   MSE= np.asmatrix(y_1-y_pred).T*np.asmatrix
        (y_1-y_pred)
   MSE = (MSE/n)
   return MSE

   error1 = LSE_error(y_1,y_pred)
print(error1)
```

To plot the Predicted and actual plot

```
plt.plot(t,y_1,'o',label = 'Original curve')
plt.plot(t,y_pred,label = 'Fitted curve')
```

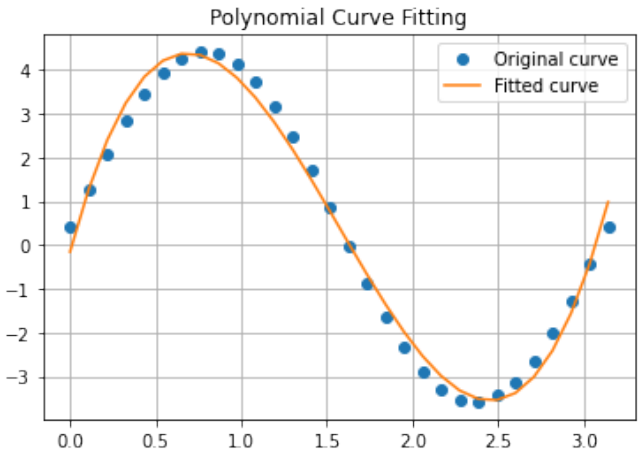

Fig. 0: Fitted Data

```
plt.legend()
plt.grid()
plt.title("Polynomial Curve Fitting")
plt.show()
```

4 Observation

- Model is approximated by a polynomial function
- Noise is added to the training data labels
- With polynomial degree N= 1, we got LSE of 3.51.
- with polynomial degree N= 4 we got LSE of 0.0888215
- with polynomial degree N= 10 we got LSE of 1.29225829e-07

Clearly as the number of parameters crosses the number of training points, the model is performing very poorly