Assignment07

November 15, 2018

1 Assignment07

2 Name : Junha Lee

3 Student ID : 2017220159

4 https://github.com/myosoo/Assisgnment07

4.0.1 Import packages

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from numpy.linalg import inv
        from numpy import linalg as LA
```

4.0.2 A set of data $((x_1, y_1), (x_2, y_2), , (x_n, y_n))$ is generated

```
In [2]: num = 1001
        std = 5

        # x : x-coordinate data
        # y1 : (clean) y-coordinate data
        # y2 : (noisy) y-coordinate data

        def fun(x):
            f = np.abs(x) * np.sin(x)
            return f

        n = np.random.rand(num)
        nn = n - np.mean(n)
        x = np.linspace(-10, 10, num)
        y1 = fun(x)
        y2 = y1 + nn * std
```

4.0.3 Define vandemode matrix :

$$A = \begin{vmatrix} x_0^0 & x_0^1 & \cdots & x_0^p \\ x_1^0 & x_1^1 & \cdots & x_1^p \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^p \end{vmatrix}$$

In [3]:
```
def Vandermonde_Matrix(x, p):
    return np.column_stack(x ** (i) for i in range(p+1))

A_0 = Vandermonde_Matrix(x, p = 0)
A_1 = Vandermonde_Matrix(x, p = 1)
A_2 = Vandermonde_Matrix(x, p = 2)
A_3 = Vandermonde_Matrix(x, p = 3)
A_4 = Vandermonde_Matrix(x, p = 4)
A_5 = Vandermonde_Matrix(x, p = 5)
A_6 = Vandermonde_Matrix(x, p = 6)
A_7 = Vandermonde_Matrix(x, p = 7)
A_8 = Vandermonde_Matrix(x, p = 8)
A_9 = Vandermonde_Matrix(x, p = 9)
```

4.0.4 Define theta :

$\Theta = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y_2}$

In [4]:
```
def Theta(y2, A):
    return (np.linalg.pinv(A)) * np.transpose(np.matrix(y2)) # theta = pseudo inverse A

Theta_0 = Theta(y2, A = A_0)
Theta_1 = Theta(y2, A = A_1)
Theta_2 = Theta(y2, A = A_2)
Theta_3 = Theta(y2, A = A_3)
Theta_4 = Theta(y2, A = A_4)
Theta_5 = Theta(y2, A = A_5)
Theta_6 = Theta(y2, A = A_6)
Theta_7 = Theta(y2, A = A_7)
Theta_8 = Theta(y2, A = A_8)
Theta_9 = Theta(y2, A = A_9)
```

4.0.5 Define approximation model

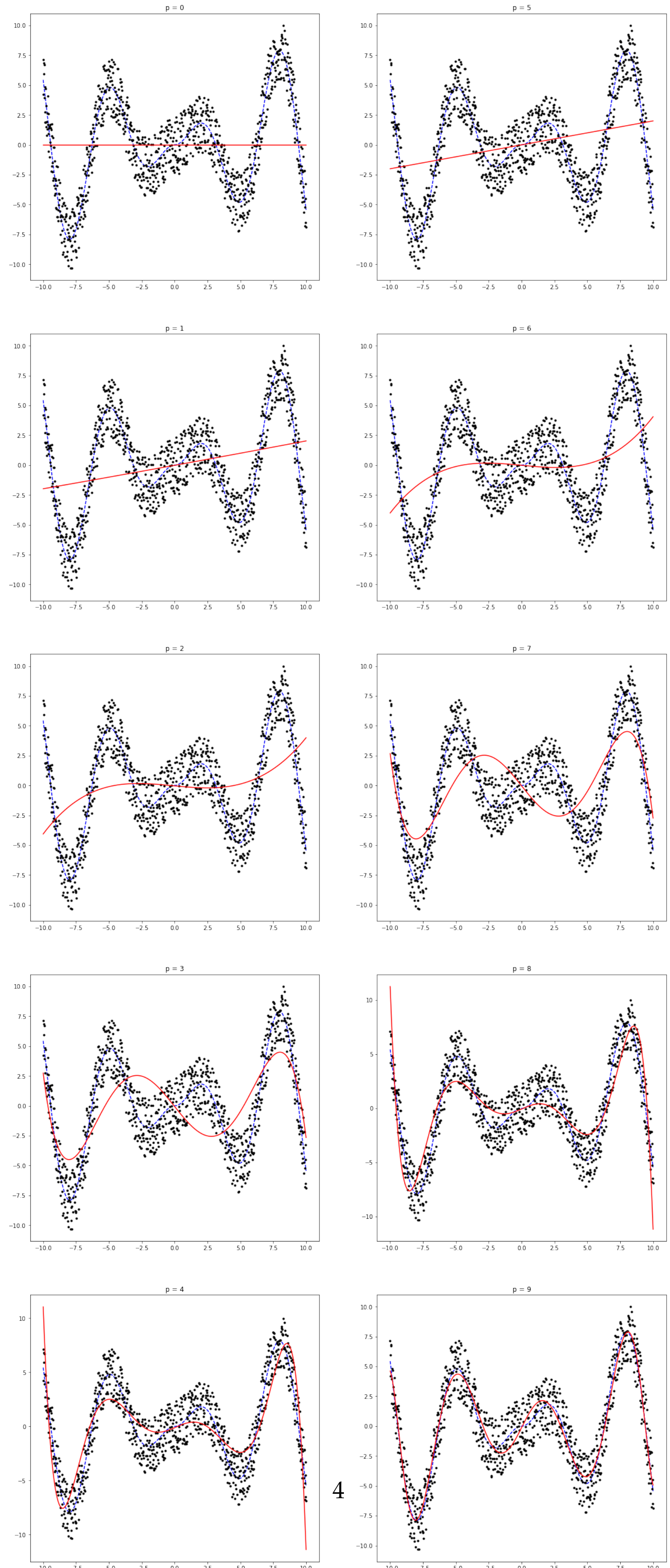
In [5]:
```
def Approximation_model(A, Theta):
    return A * Theta

app_y_0 = Approximation_model(A_0, Theta_0)
app_y_1 = Approximation_model(A_1, Theta_1)
app_y_2 = Approximation_model(A_2, Theta_2)
app_y_3 = Approximation_model(A_3, Theta_3)
app_y_4 = Approximation_model(A_4, Theta_4)
```

```
app_y_5 = Approximation_model(A_5, Theta_5)
app_y_6 = Approximation_model(A_6, Theta_6)
app_y_7 = Approximation_model(A_7, Theta_7)
app_y_8 = Approximation_model(A_8, Theta_8)
app_y_9 = Approximation_model(A_9, Theta_9)
```

4.0.6 Plot the polynomial curves that fit the noisy data by the least square error with varying p = 0, 1, 2, 3, ů ů ů 9

In [6]:
```
plt.figure(figsize=(20,50))
plt.subplot(5, 2, 1)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_0, 'r-')
plt.title('p = 0')
plt.subplot(5, 2, 3)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_2, 'r-')
plt.title('p = 1')
plt.subplot(5, 2, 5)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_4, 'r-')
plt.title('p = 2')
plt.subplot(5, 2, 7)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_6, 'r-')
plt.title('p = 3')
plt.subplot(5, 2, 9)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_8, 'r-')
plt.title('p = 4')
plt.subplot(5, 2, 2)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_1, 'r-')
plt.title('p = 5')
plt.subplot(5, 2, 4)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_3, 'r-')
plt.title('p = 6')
plt.subplot(5, 2, 6)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_5, 'r-')
plt.title('p = 7')
plt.subplot(5, 2, 8)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_7, 'r-')
plt.title('p = 8')
plt.subplot(5, 2, 10)
plt.plot(x, y1, 'b--', x, y2, 'k.', x, app_y_9, 'r-')
plt.title('p = 9')
plt.show()
```


p = 0
p = 5
p = 1
p = 6
p = 2
p = 7
p = 3
p = 8
p = 4
p = 9

4.0.7 Define residual : $r_n = y_n - \hat{f}(x_n)$

```
In [7]: def residual(app_y):
            return np.array(np.transpose(np.matrix(y2))) - np.array(app_y) #Resisdual

        R_0 = residual(app_y_0)
        R_1 = residual(app_y_1)
        R_2 = residual(app_y_2)
        R_3 = residual(app_y_3)
        R_4 = residual(app_y_4)
        R_5 = residual(app_y_5)
        R_6 = residual(app_y_6)
        R_7 = residual(app_y_7)
        R_8 = residual(app_y_8)
        R_9 = residual(app_y_9)
```

4.0.8 Define error : $\sum_{n=1}^{p} r_n^2$

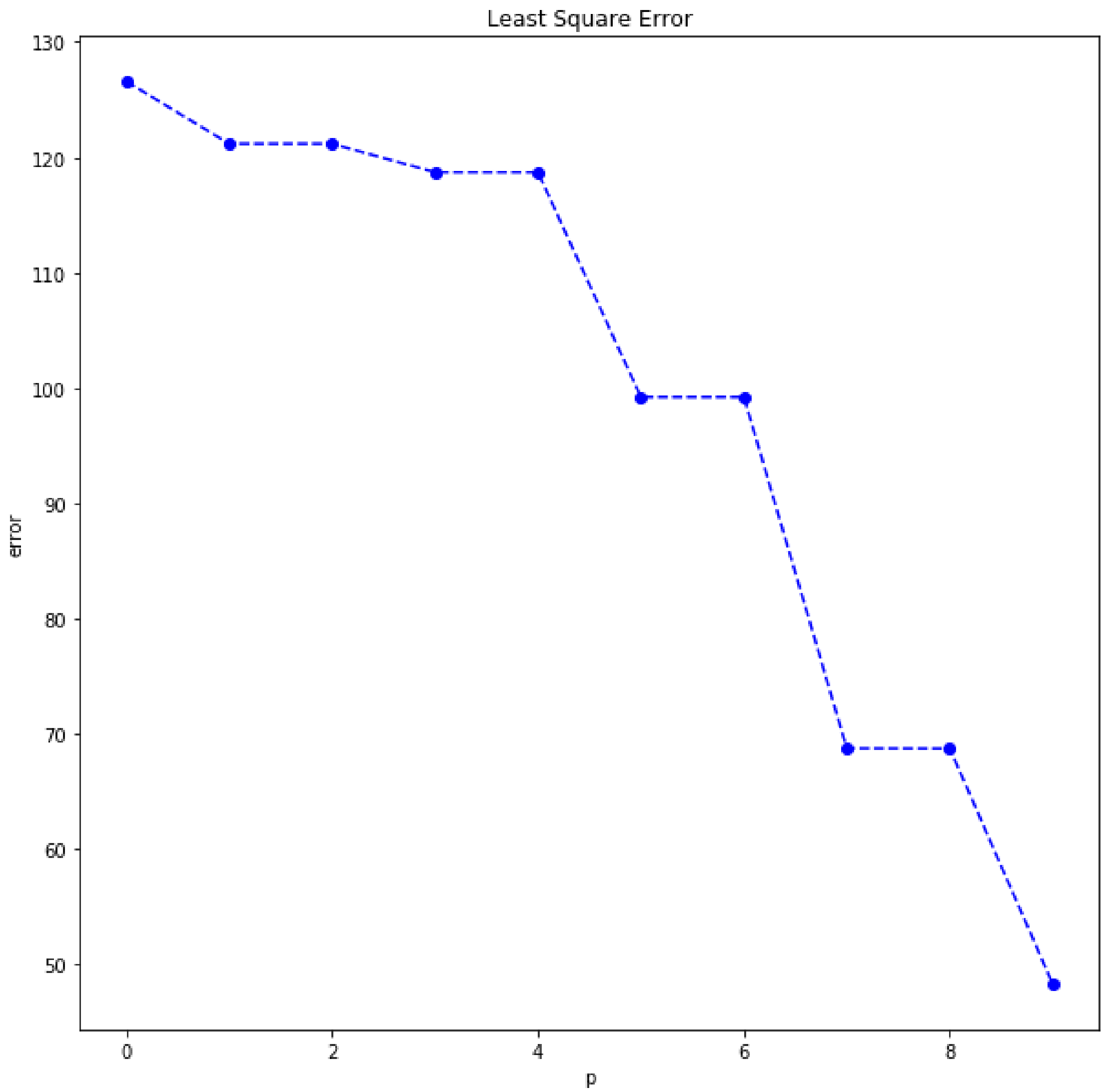
```
In [8]: def Least_Square_Error(R):
            return LA.norm(R)

        LSE_0 = Least_Square_Error(R_0)
        LSE_1 = Least_Square_Error(R_1)
        LSE_2 = Least_Square_Error(R_2)
        LSE_3 = Least_Square_Error(R_3)
        LSE_4 = Least_Square_Error(R_4)
        LSE_5 = Least_Square_Error(R_5)
        LSE_6 = Least_Square_Error(R_6)
        LSE_7 = Least_Square_Error(R_7)
        LSE_8 = Least_Square_Error(R_8)
        LSE_9 = Least_Square_Error(R_9)
```

4.0.9 Plot the error $\sum_{n=1}^{p} r_n^2$ where $r_n = y_n - \hat{f}(x_n)$ is the residual with varying p = 0, 1, 2, 3, ů ů ů 9

```
In [9]: p = np.arange(0, 10, 1)
        LSE = [LSE_0, LSE_1, LSE_2, LSE_3, LSE_4, LSE_5, LSE_6, LSE_7, LSE_8, LSE_9]

        plt.figure(figsize=(10,10))
        plt.plot(p, LSE, color = 'b', marker = 'o', linestyle = '--')
        plt.xlabel('p')
        plt.ylabel('error')
        plt.title('Least Square Error')
        plt.show()
```


Least Square Error
130
120
110
100
90
80
70
60
50
error
0
2
4
6
8
p