assignment07

November 10, 2018

1 Polynomial Fit

Name: ZHU GUANGYU
Student ID: 20165953
Github Repo: assignment07

Samilar with last time we did the *Straight-line fit*. We use *least squares data fitting* to find the approximate value of given data set.

1.1 Least Squares Problem

Suppose we have a tall matrix A, so $Ax = b$ is *over-determined*. For most b, there is no x that satisfies $Ax = b$.

The *least squares problem* is: choosing x to minimize $||Ax - b||^2$.
We call the solution *least squares approxmimata solution* of $Ax = b$, and notate it as $\hat{x}$.

1.2 Least squares data fitting

If we have a scalar y and an n-vector x that are related by model

$$y \approx f(x)$$

x is the *independent variable*,
y is the *outcome*,
$f : R^n \to R$ gives the relation between x and y.

Often x is a feature vector, and y is something we want to predict.
We do not know the true relationship f between x and y. So what we want to do is to find a *approximation* of f.

We define a *linear in the parameters* model:

$$\hat{f}(x) = \theta_1 f_1(x) + \cdots + \theta_p f_p(x)$$

$f_i : R^n \to R$: are *basis fuctions* that we choose.
θ_i: are *model parameters* that we choose.
$\hat{y}^{(i)} = \hat{f}(x^{(i)})$ is prediction of $y^(i)$.

Thus, our goal becomes to choose model parameters θ_i to minimize *residuals* $r^i = y^{(i)} - \hat{y}^{(i)}$. This can be formulated and solved as a **least squares problem**.

If we define
$y^d = (y^{(1)}, \cdots, y^{(N)})$ is vecor of outcomes,
$\hat{y}^d = (\hat{y}^{(1)}, \cdots, \hat{y}^{(N)})$ is vector of predictions,
$r^d = (r^{(1)}, \cdots, r^{(N)})$ is vector of residuals.
have $N \times p$ matrix A with elements $A_{ij} = fj(x^{(i)})$, so $\hat{y}^d = A\theta$.
$||r^d||^2 = ||y^d - \hat{y}^d||^2 = ||y^d - A\theta||^2$

1.3 Polynomial Fit

Polynomial fit means we set out parameter model to a polynomial equation, So we have:

$$\hat{f}(x) = \theta_0 + \theta_1 x + \cdots + \theta_p x^p$$

Our matrix A becomes to:

$$\begin{bmatrix} 1 & x^{(1)} & \cdots & (x^1)^p \\ 1 & x^{(2)} & \cdots & (x^2)^p \\ \vdots & \vdots & & \vdots \\ 1 & x^{(N)} & \cdots & (x^N)^p \end{bmatrix}$$

1.3.1 First, generate data set.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

num     = 1001
std     = 5

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

def fun(x):

    f = np.sin(x) * (1 / (1 + np.exp(-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

plt.plot(x, y1, 'b.', x, y2, 'k.')
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

1.3.2 Second, get θ

Since we have $A\theta = \hat{y}$, the best way to get θ is to time A^{-1} at both side.
By *pseudo inverse*: $(A^T A)^{-1} A$, we can compute the inverse easily.
Before compute inverse of A, let's define A first. The row of A is $[1, x, x^2, \cdots, x^p]$

```
In [2]: def build_matrix(x, p):
            '''Build basic function matrix
            x: input data set
            p: degree of polynomial
            '''

            matrix = []

            for xi in x:
                poly = [xi**dg for dg in range(p+1)]
                matrix.append(poly)

            return np.array(matrix)
```

Now, let's compute the inverse of A.
Here, we use numpy's function `linalg.pinv()` to compute the pseudo inverse.

```
In [3]: def theta(A, y):
            A_inv = np.linalg.pinv(A)
            theta = np.inner(A_inv, y)

            return theta
```

1.3.3 Third, get the $\hat{y}$ function

We have already got θ from above, so we can compute our approximate value now.

```
In [4]: def y_hat(x, y, p):

            A = build_matrix(x, p)
            theta_arr = theta(A, y)

            y_i = np.inner(np.array([x**dg for dg in range(p+1)]).T, theta_arr)

            return y_i
```

1.3.4 Fourth, define residual function

For comparing fit result, we have $\sum_{j=1}^{n} r_j^2$ where $r_j = y_j - \hat{f}(x_j)$

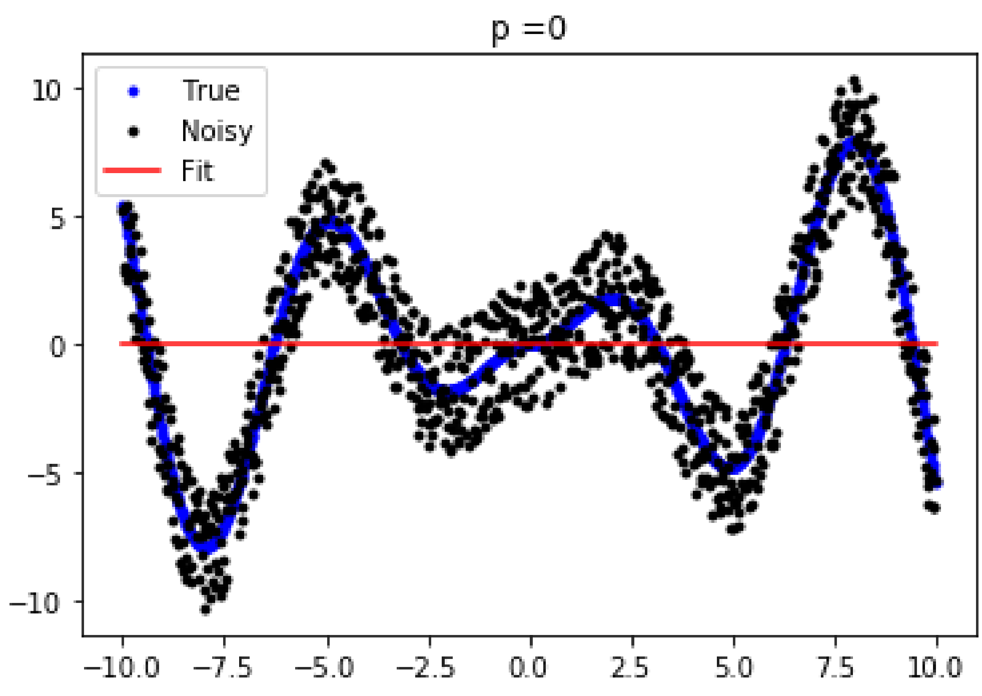
```
In [5]: def residual(y, y_hat):
            y_diff = y - y_hat
            return sum([e**2 for e in y_diff])
```

1.4 Show Result

Let's try some different, and check the error.

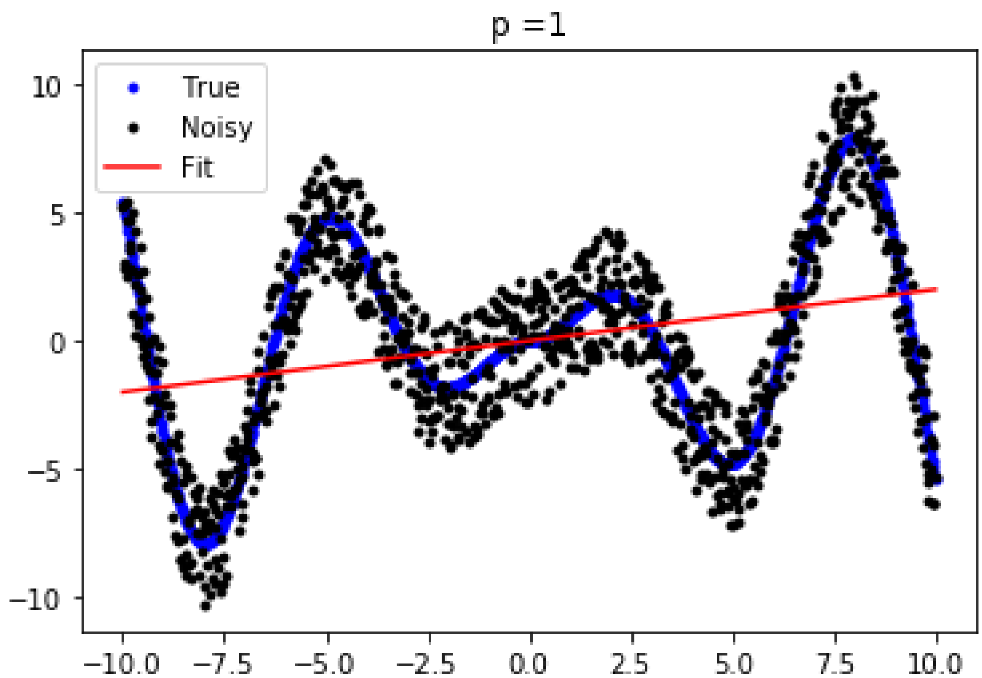
1.4.1 When $p = 0$

In [6]:
```
# list for storing residual of each p
error = []

# When p =0
def plot_graph(x, y_clean, y_noisy, p):

    y = y_hat(x, y_noisy, p)

    error.append(residual(y_clean, y))
    # Plot part
    plt.title("p ={}".format(p))
    plt.plot(x, y_clean, 'b.', x, y_noisy, 'k.', x, y, 'r')
    plt.legend(['True', 'Noisy', 'Fit'], loc='upper left')
    plt.show()

err = plot_graph(x, y1, y2, 0)
```



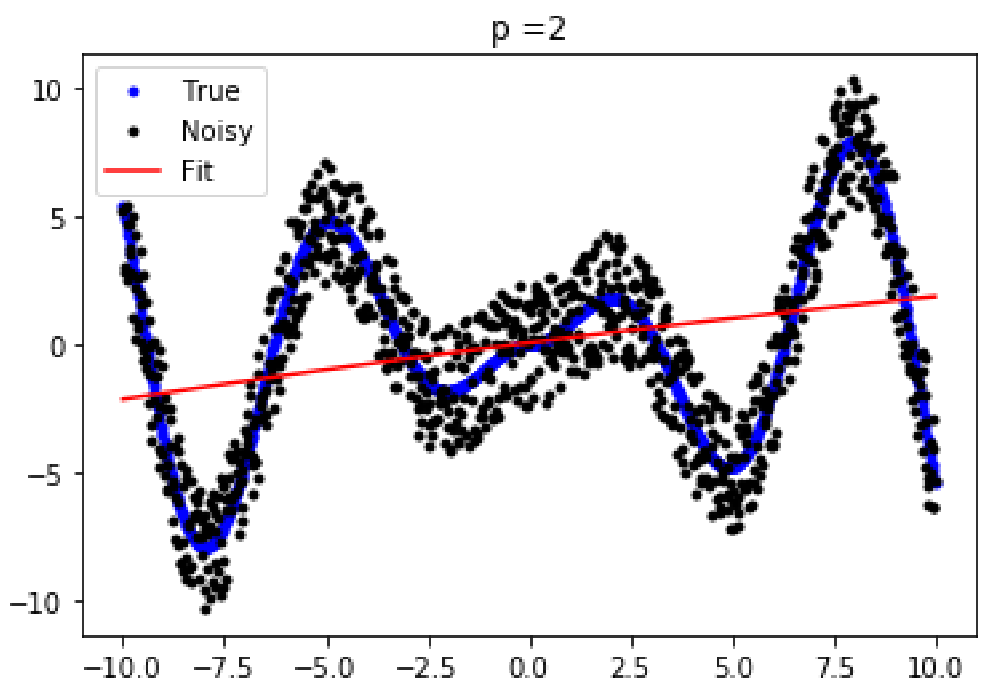
1.4.2 When $p = 1$

In [7]:
```
plot_graph(x, y1, y2, 1)
```



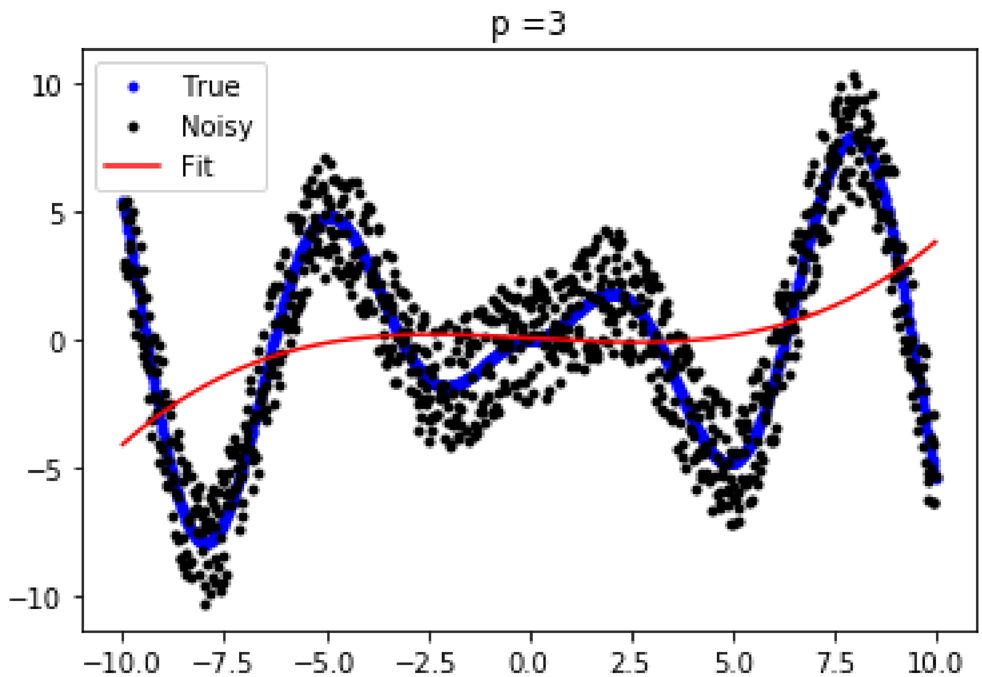
1.4.3 When $p = 2$

```
In [8]: plot_graph(x, y1, y2, 2)
```



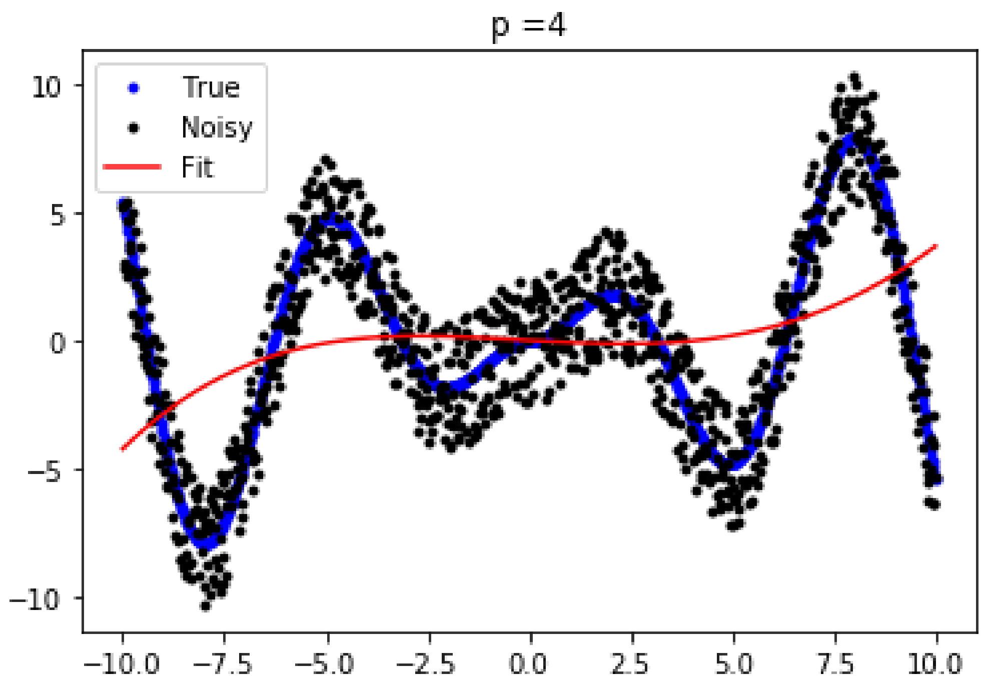
1.4.4 When $p = 3$

```
In [9]: plot_graph(x, y1, y2, 3)
```



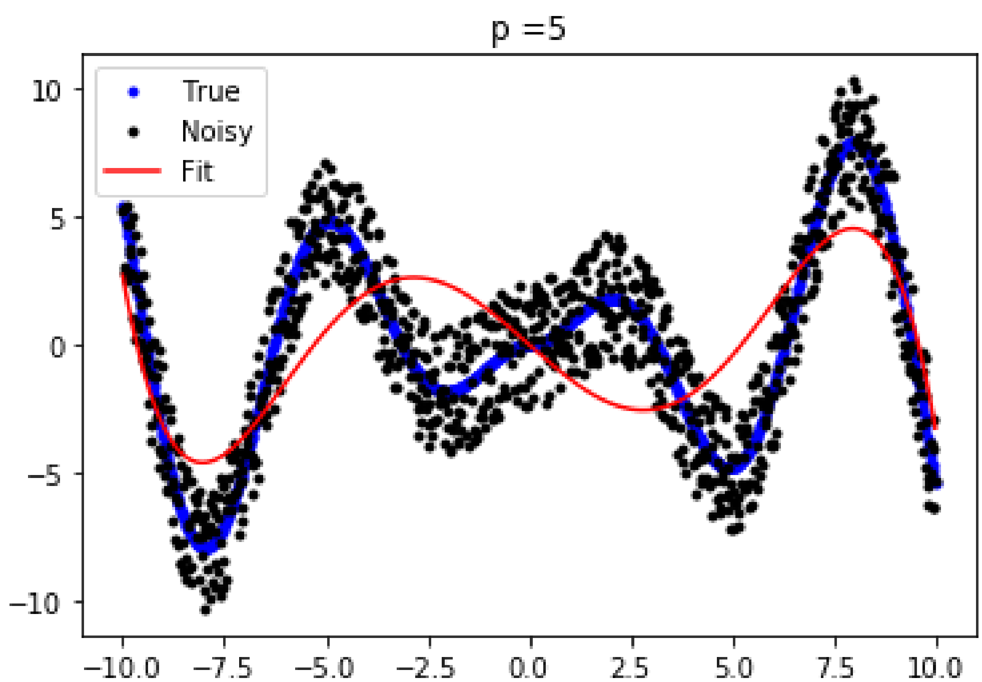
1.4.5 When $p = 4$

```
In [10]: plot_graph(x, y1, y2, 4)
```



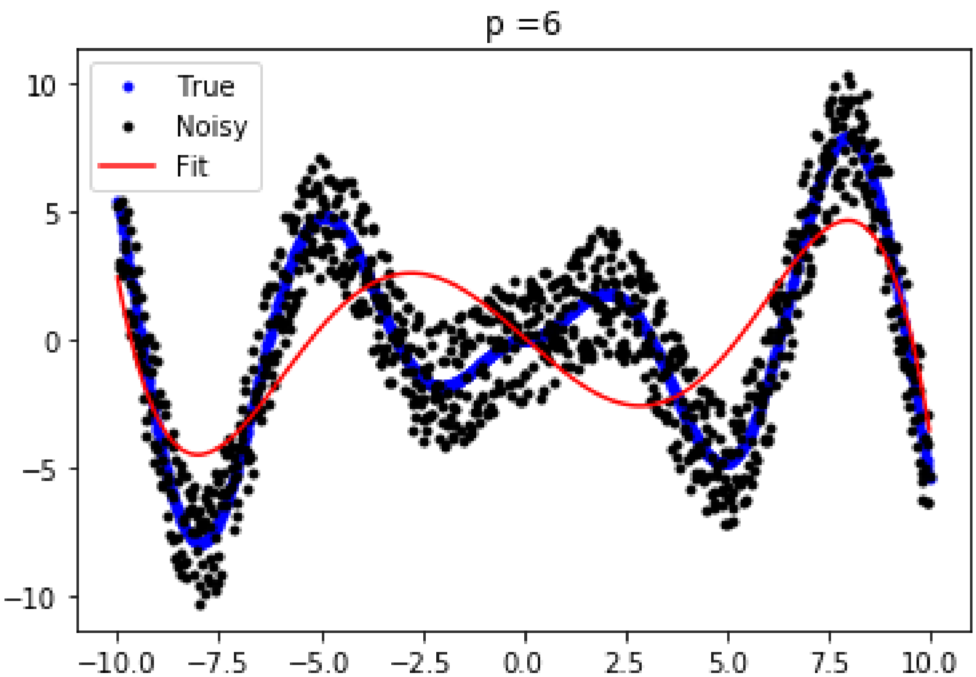
1.4.6 When $p = 5$

```
In [11]: plot_graph(x, y1, y2, 5)
```



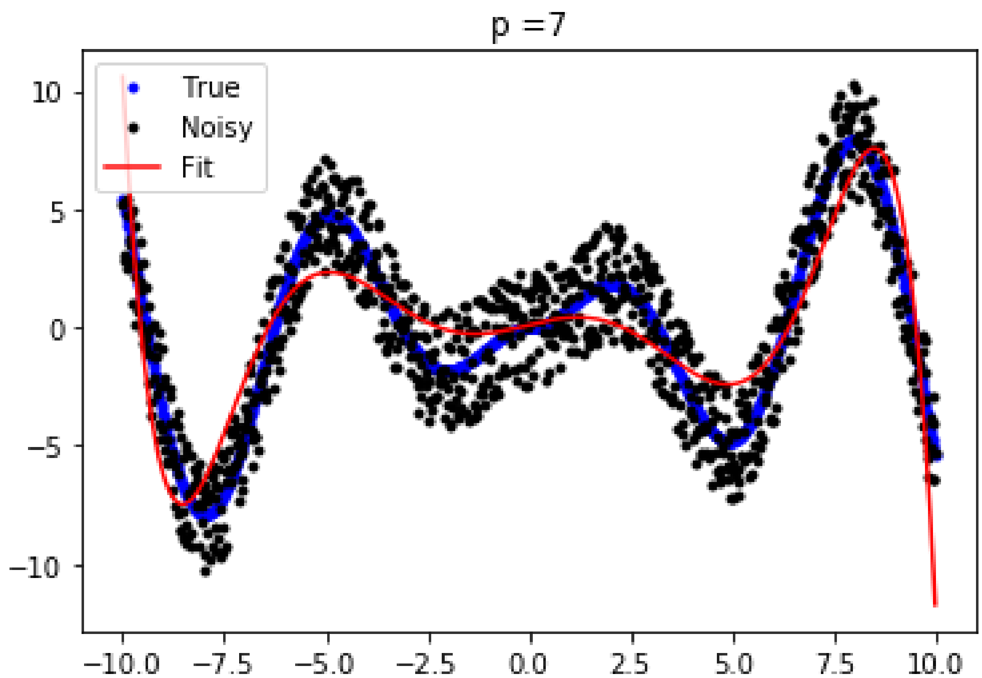
1.4.7 When $p = 6$

```
In [12]: plot_graph(x, y1, y2, 6)
```



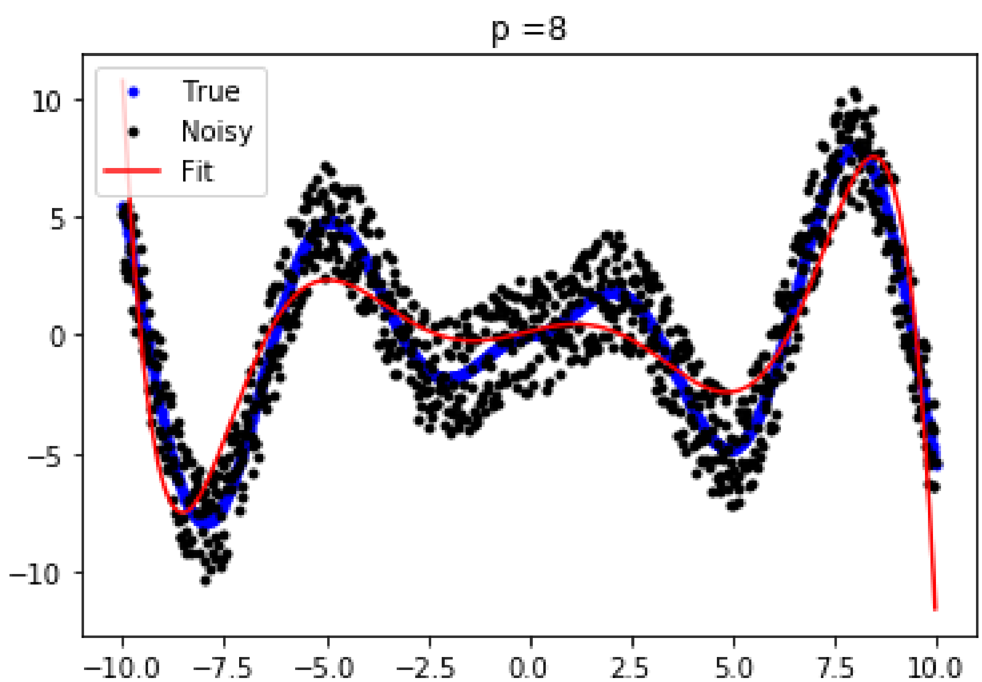
1.4.8 When $p = 7$

```
In [13]: plot_graph(x, y1, y2, 7)
```



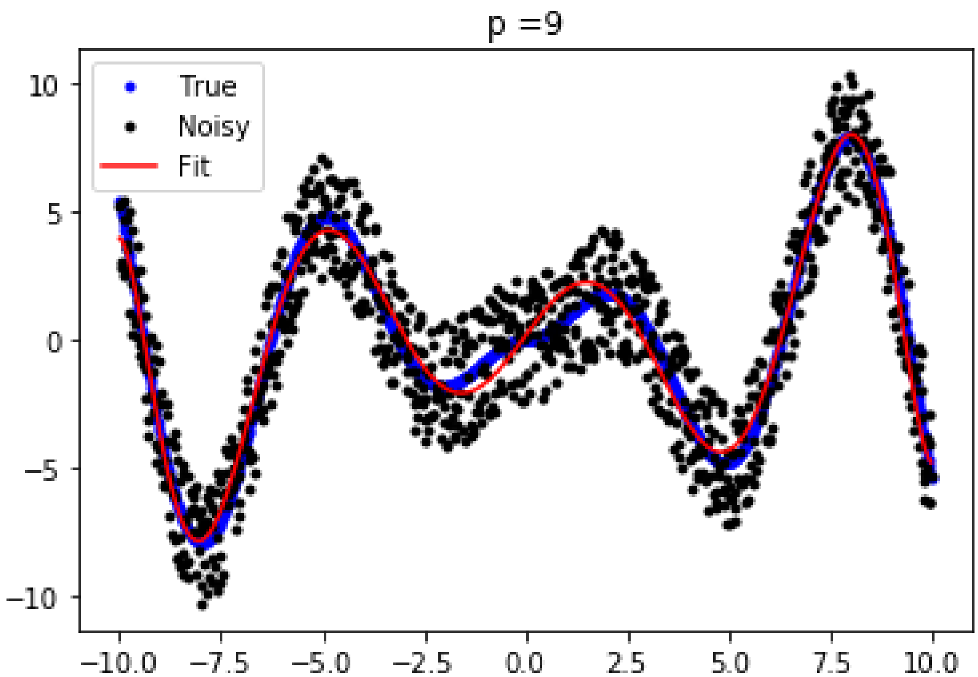
1.4.9 When $p = 8$

```
In [14]: plot_graph(x, y1, y2, 8)
```



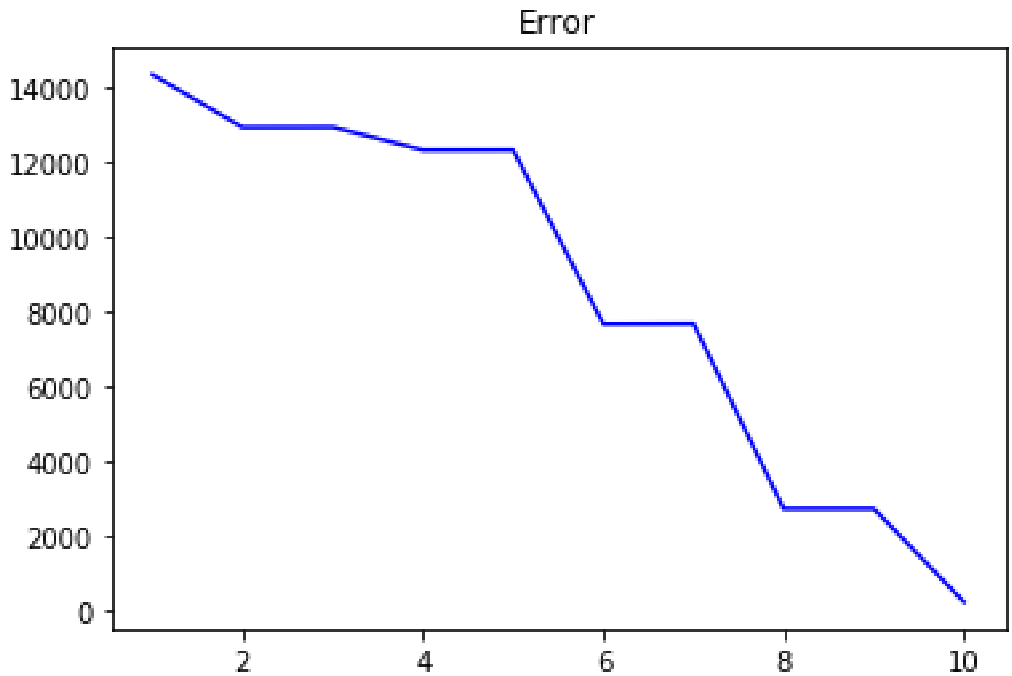
1.4.10 When $p = 9$

```
In [15]: plot_graph(x, y1, y2, 9)
```



1.4.11 Error

```
In [16]: x_axis = [x for x in range(1, 11)]

         plt.title("Error")
         plt.plot(x_axis, error, 'b-')
         plt.show()
```

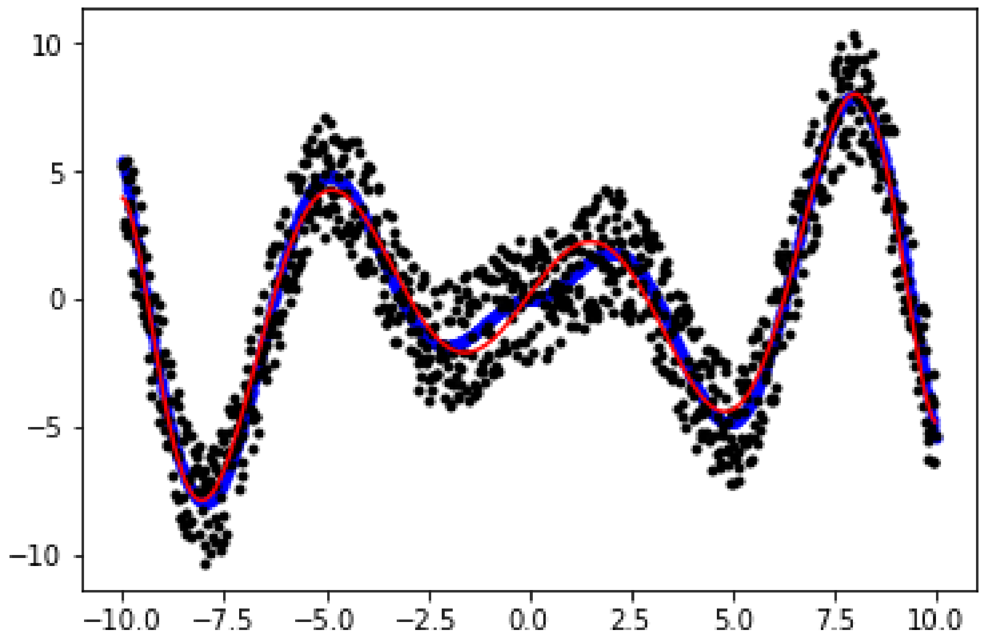


1.5 *Use numpy function

numpy also has functions for polynomial fit:

- polyfit: compute θ
- polyval: use given input and θ to compute the approximate value

```
In [17]: z = np.polyfit(x, y2, 9)

         y_val = np.polyval(z, x)
         plt.plot(x, y1, 'b.', x, y2, 'k.', x, y_val, 'r')
         plt.show()
```



We can see from the graphs, the result of `numpy.polyfit` and our function are same.

1.6 Find optimum p

Because we have *residule*, we can use it to find the best p.

In [20]:
```
import math

err_prv = math.inf
p = 0
err = residual(y1, y_hat(x, y2, p))
p_op = 0
ite = 3

while(ite):
    if (err >= err_prv):
        if (ite == 3):
            p_op = p - 1
        ite -= 1
    else:
        ite = 3

    err_prv = err
    p += 1
    err = residual(y1, y_hat(x, y2, p))

print("The optimal value of p is {}.".format(p_op))
```

```
The optimal value of p is 15.
```

```
In [21]: plot_graph(x, y1, y2, 15)
```