Assignment8

May 23, 2019

1 Information

Writer : *Junhyuck Woo*
Std.ID : 20145337
Project : *Polynomial fitting*

2 import library

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

3 Function: Generate matrix

```
In [2]: def generate_matrix(points, sample_number, order):
            matrix = np.zeros((sample_number, order+1), dtype=int)
            for i in range(sample_number):
                for j in range(order+1):
                    matrix[i][j] = pow(points[i], j)
            return(np.asmatrix(matrix))
```

4 Function: Generate Polynomial function

```
In [3]: def generate_function(x, coefficient):
            y = 0
            coefficient = np.asarray(coefficient)
            for i in range(len(coefficient)):
                y += coefficient[i] * pow(x, i)
            return(y)
```

5 Function: Calculate distance

```
In [4]: def distance(x, y):
            d = (x-y)**2
            s = np.sum(d)
```

```
    r = np.sqrt(s)
    return(r)
```

6 Define a polynomial curve

$f(x) = 3x - 4$

7 Define a domain of the function.

$-50 \leqq X \leqq 50$

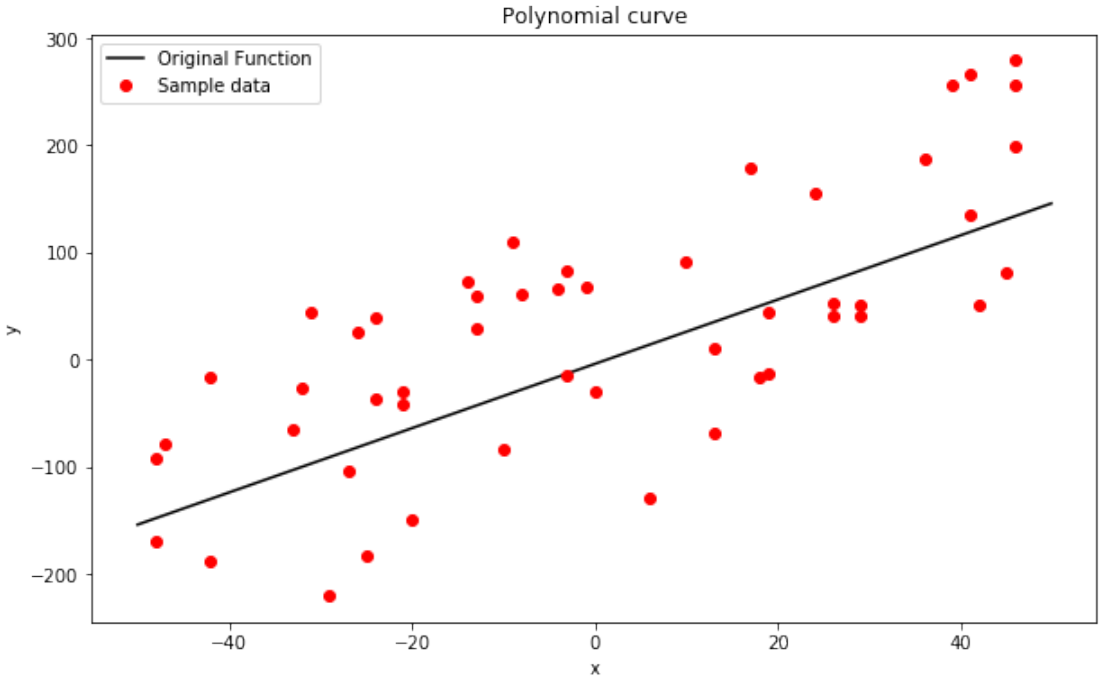
```
In [5]: x = np.arange(-50, 50, 0.1)
        y = 3*x - 4
```

8 Generate noisy

```
In [6]: e = np.int64(np.random.uniform(-150, 150, 50).astype(int))
        point_x = np.int64(np.random.uniform(-50, 50, 50).astype(int))
        point_noised = 3*point_x-4+e
```

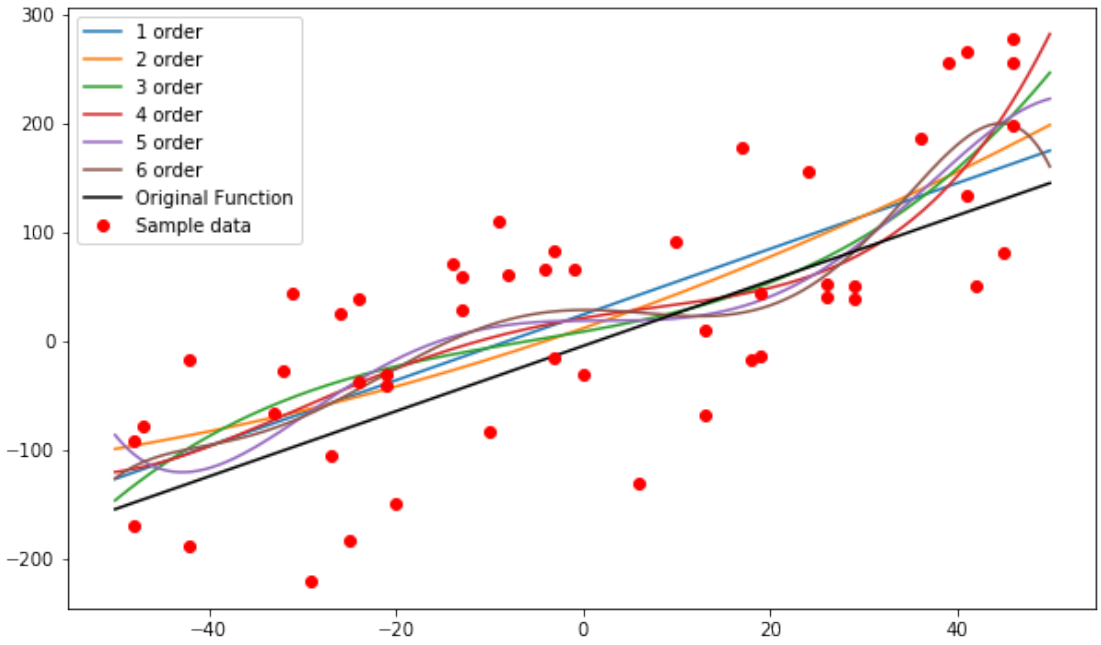
9 Plot the function.

```
In [7]: plt.figure(figsize=(10, 6));
        plt.plot(x, y, color='black', label="Original Function");
        plt.plot(point_x, point_noised, 'ro', label="Sample data");
        plt.title('Polynomial curve'); plt.xlabel('x'); plt.ylabel('y');
        plt.legend()

Out[7]: <matplotlib.legend.Legend at 0x116594ba8>
```



10 Select a point within the domain.

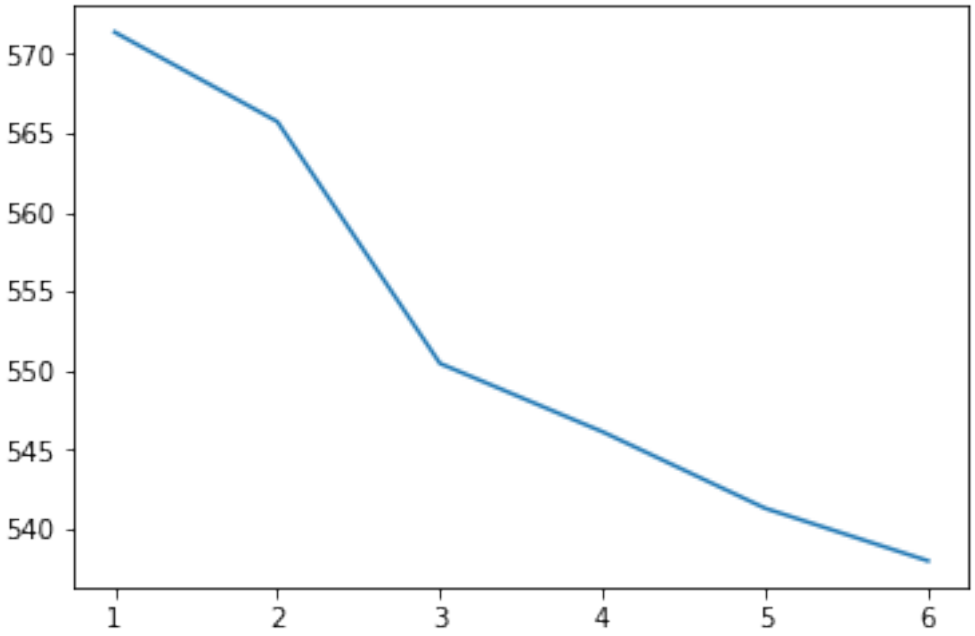
```
In [8]: plt.figure(figsize=(10, 6));
        estimated_y = []
        functions = []
        for i in range(6):
            i+=1
            matrix = generate_matrix(point_x, 50, i)
            pseudo_inverse_matrix = np.linalg.pinv(matrix)

            y2matrix = np.asmatrix(point_noised)
            coefficient = pseudo_inverse_matrix*np.matrix.transpose(y2matrix)

            function = generate_function(x, coefficient)
            estimation = generate_function(point_x, coefficient)

            functions.append(function)
            estimated_y.append(estimation)
            plt.plot(x,function, label=str(i)+" order")

        plt.plot(x, y, color='black', label="Original Function");
        plt.plot(point_x, point_noised, 'ro', label="Sample data");
        plt.legend(loc="upper left")
        plt.show()
```



11 Energy Function

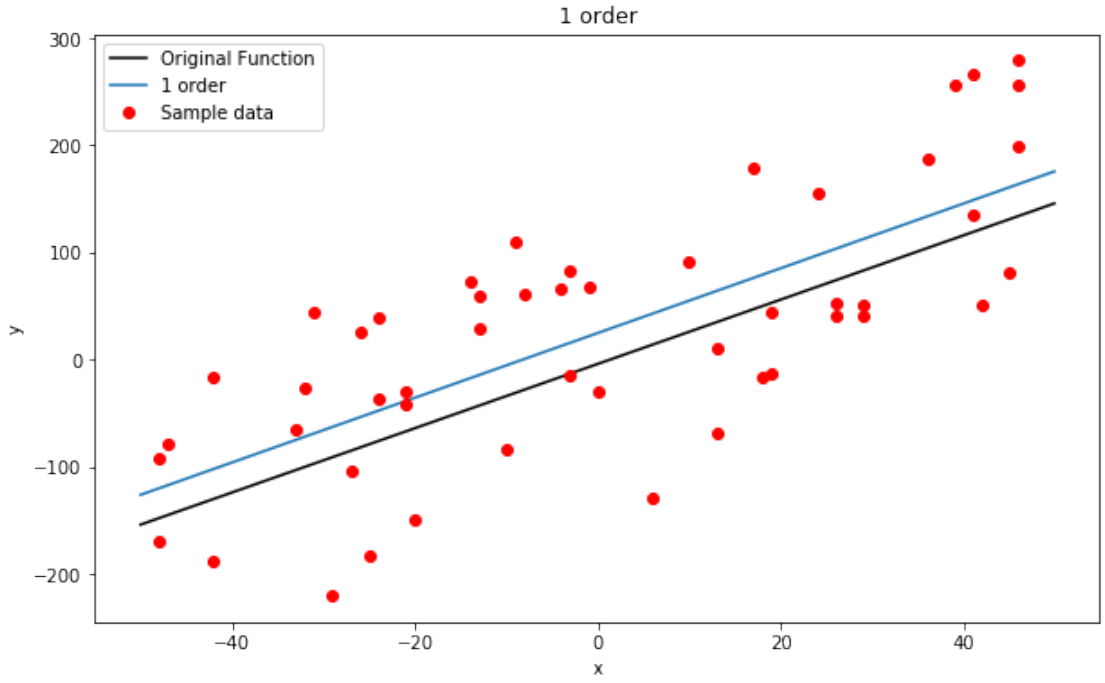
```
In [9]: energy = []
        degree = []
        for i in range(6):
            degree.append(i+1)
            energy.append(distance(estimated_y[i], point_noised))
        plt.plot(degree, energy, label="Energy")
```

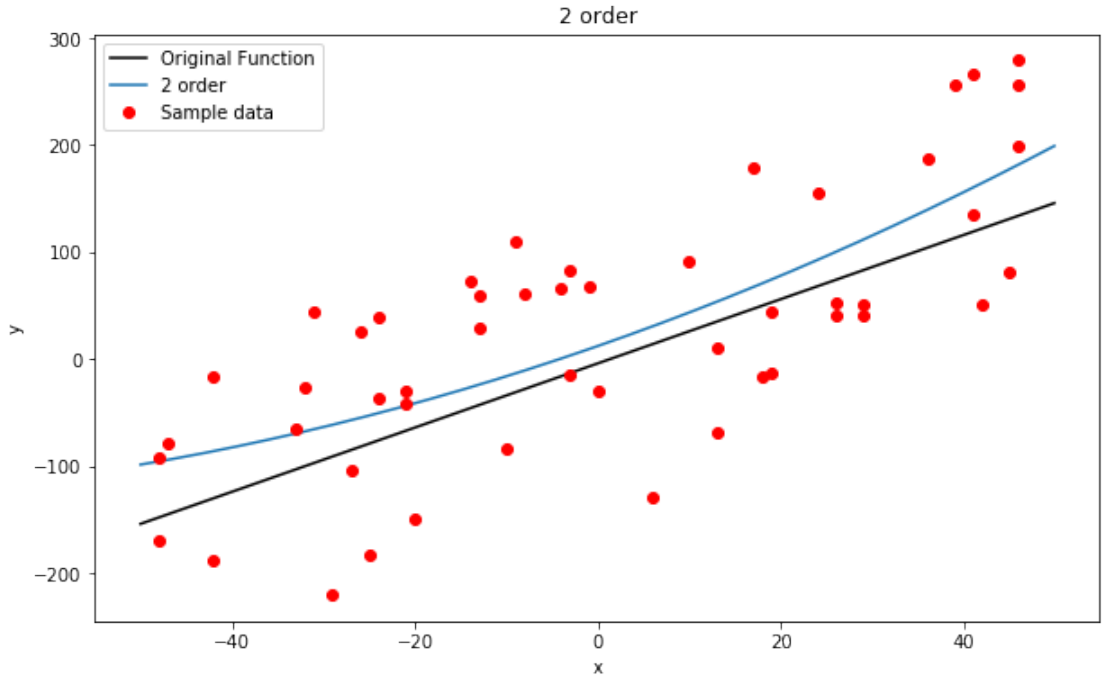
```
Out[9]: [<matplotlib.lines.Line2D at 0x1191bde80>]
```

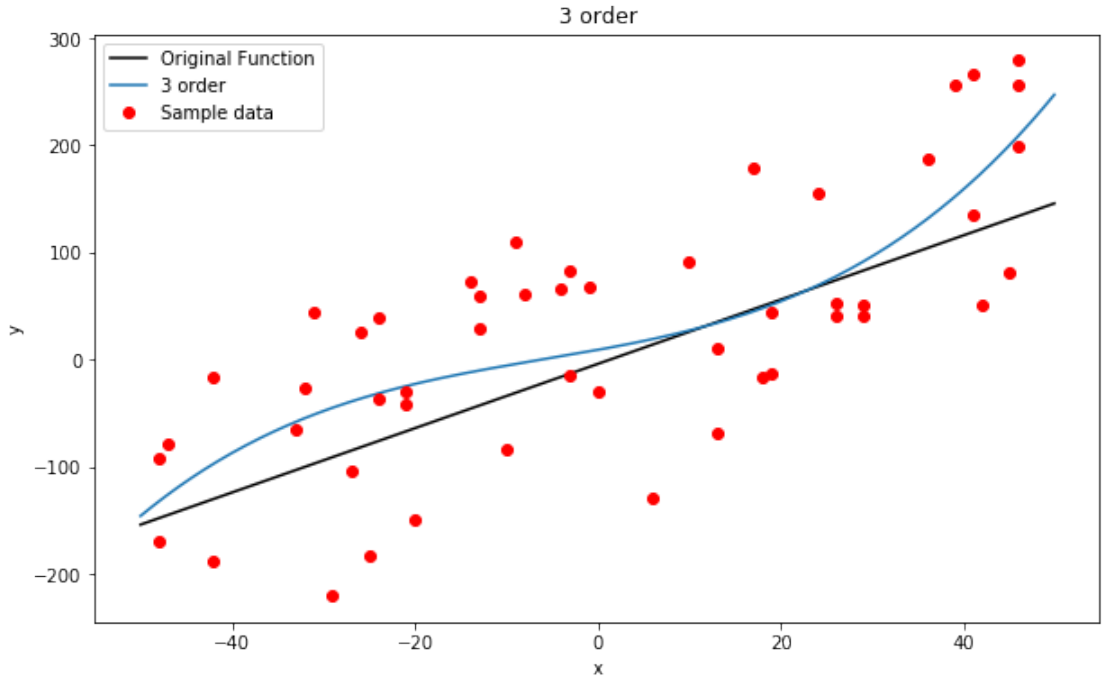


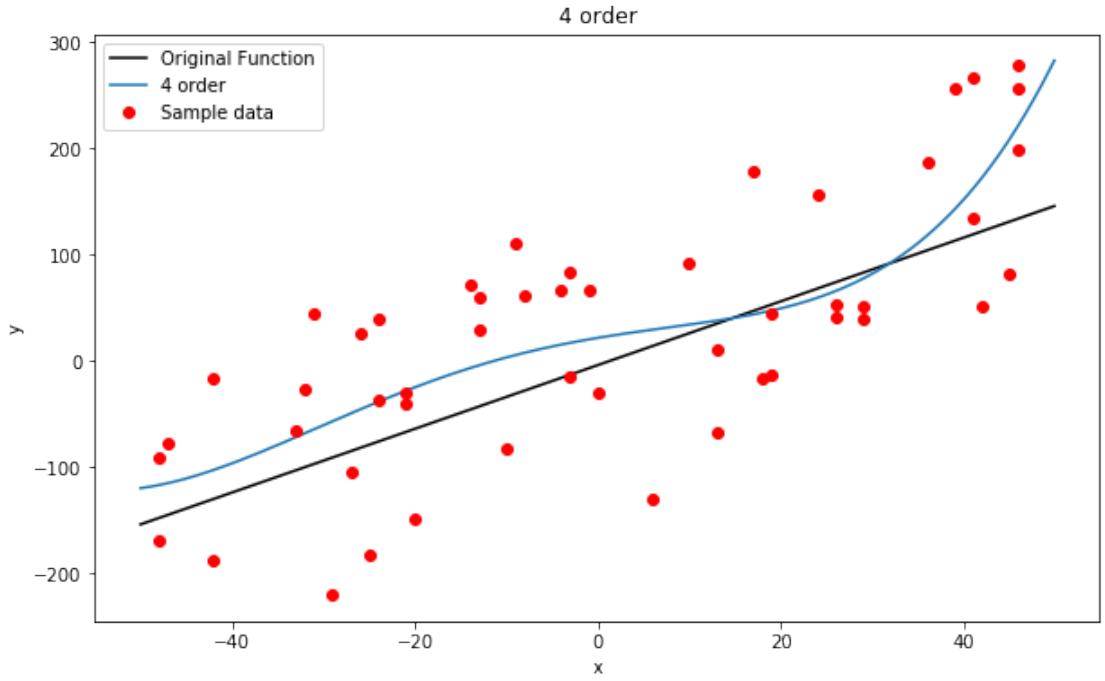
12 Plot graph saperately

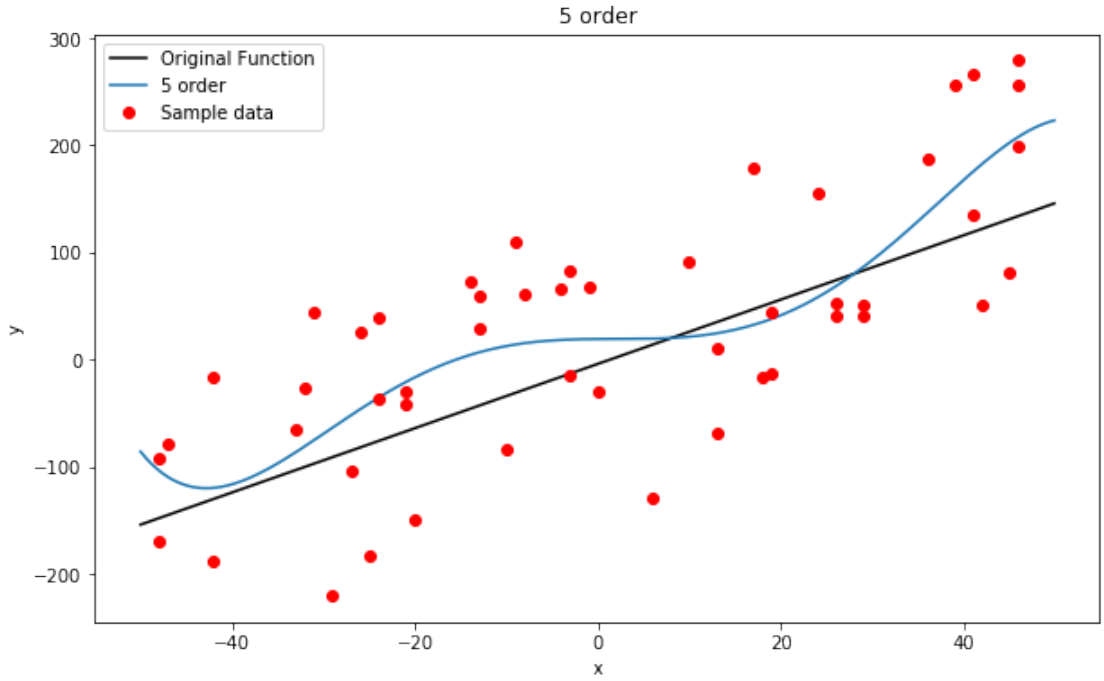
```
In [10]: for i in range(6):
             plt.figure(figsize=(10, 6));
             plt.plot(x, y, label="Original Function", color="black")
             plt.plot(x, functions[i], label=str(i+1)+" order")
             plt.plot(point_x, point_noised, 'ro', label="Sample data");
             plt.legend(loc="upper left")
             plt.title(str(i+1) + ' order'); plt.xlabel('x'); plt.ylabel('y');
             plt.show()
```

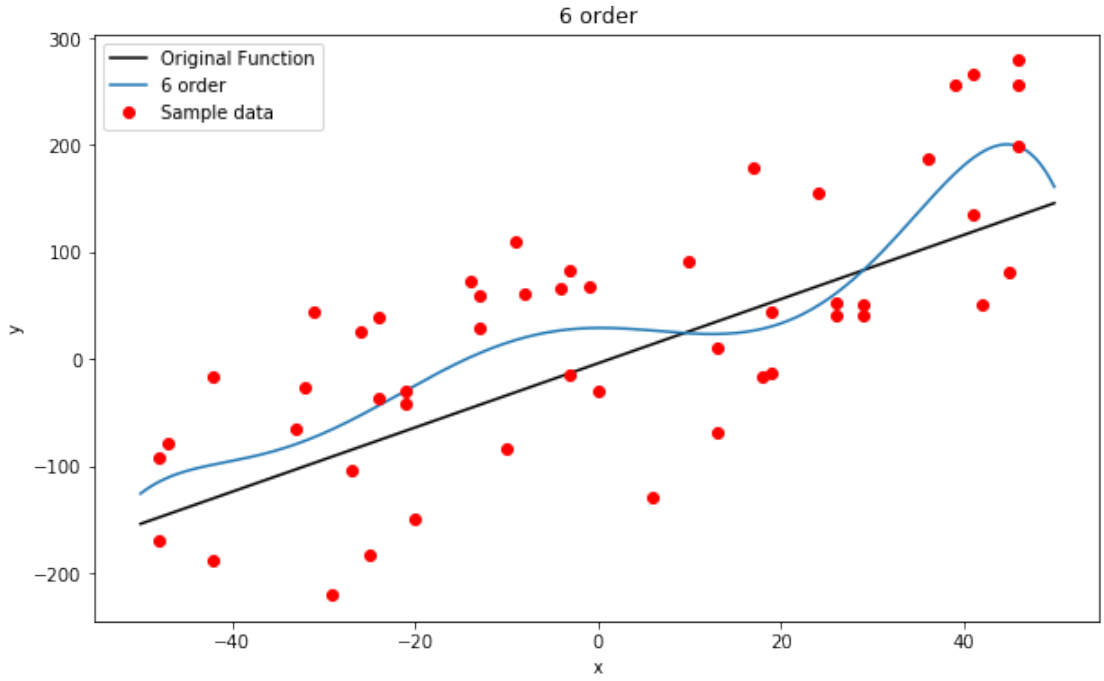

1 order
Original Function
1 order
Sample data
300
200
100
0
−100
−200
y
−40
−20
0
20
40
x


2 order
Original Function
2 order
Sample data
300
200
100
0
−100
−200
y
−40
−20
0
20
40
x


3 order
Original Function
3 order
Sample data
300
200
100
0
−100
−200
y
−40
−20
0
20
40
x


4 order
Original Function
4 order
Sample data
300
200
100
0
−100
−200
y
−40
−20
0
20
40
x


5 order
Original Function
5 order
Sample data
300
200
100
0
−100
−200
y
−40
−20
0
20
40
x


6 order
Original Function
6 order
Sample data
300
200
100
0
−100
−200
y
−40
−20
0
20
40
x