Homework 01

## 一、 实验目标：

试下多项式曲线拟合

## 二、 实验过程：

拟合函数采用多项式：

$$y(x,w) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$$

拟合方法采用最小二乘法，使误差函数最小：

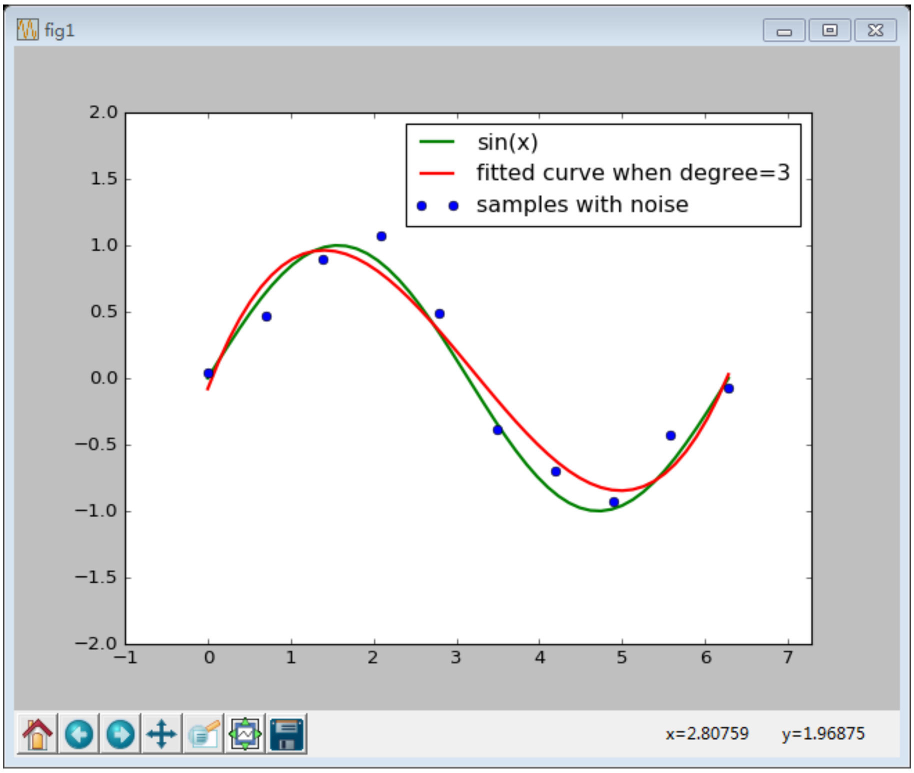
$$E(w) = \sum_{i=1}^{n} \left(y(x_i, w) - t(x_i)\right)^2$$

第一步，生成采样点。x=numpy.linspace()生成固定步长的点，y=numpy.sin(x)，生成其 sin 值。采用 numpy.random.normal(0, 0.1)模拟噪声，生成模拟采样点

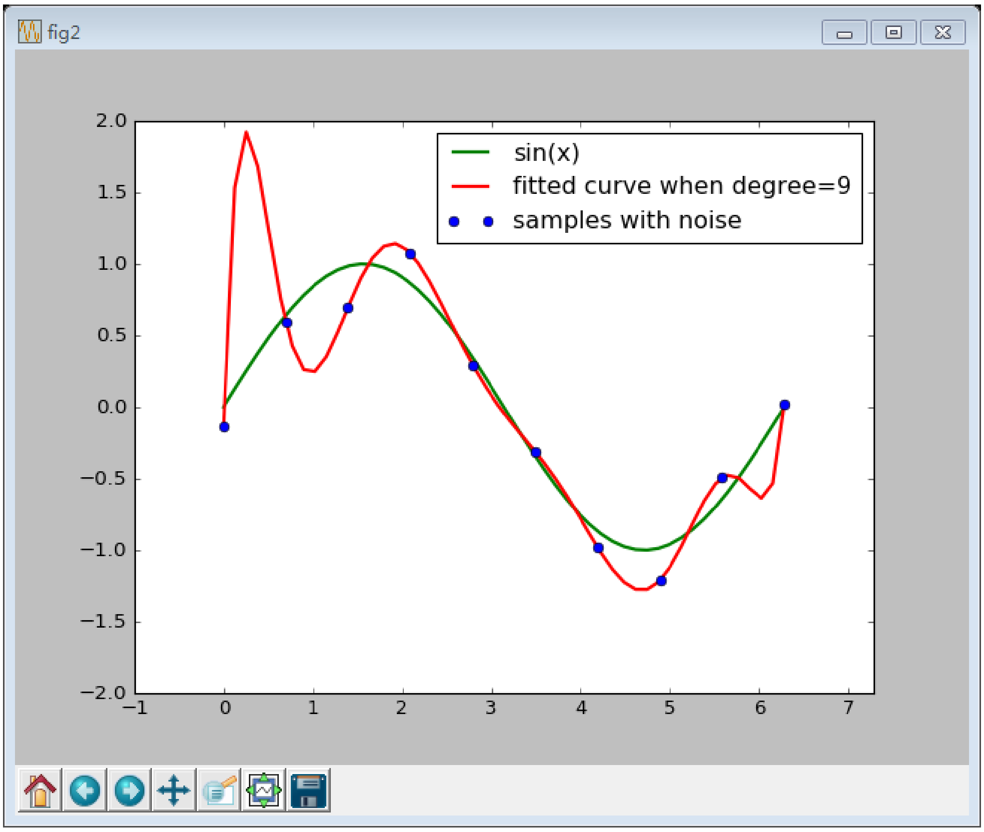
第二步，通过 np.poly1d(p)生成多项式函数，构造 residual(p,x,y)函数，即实际值与预期值的误差。通过 leastsq 函数，使用最小二乘法拟合曲线。并得到最终的多项式参数，绘制图像。

## 三、 实验结果：

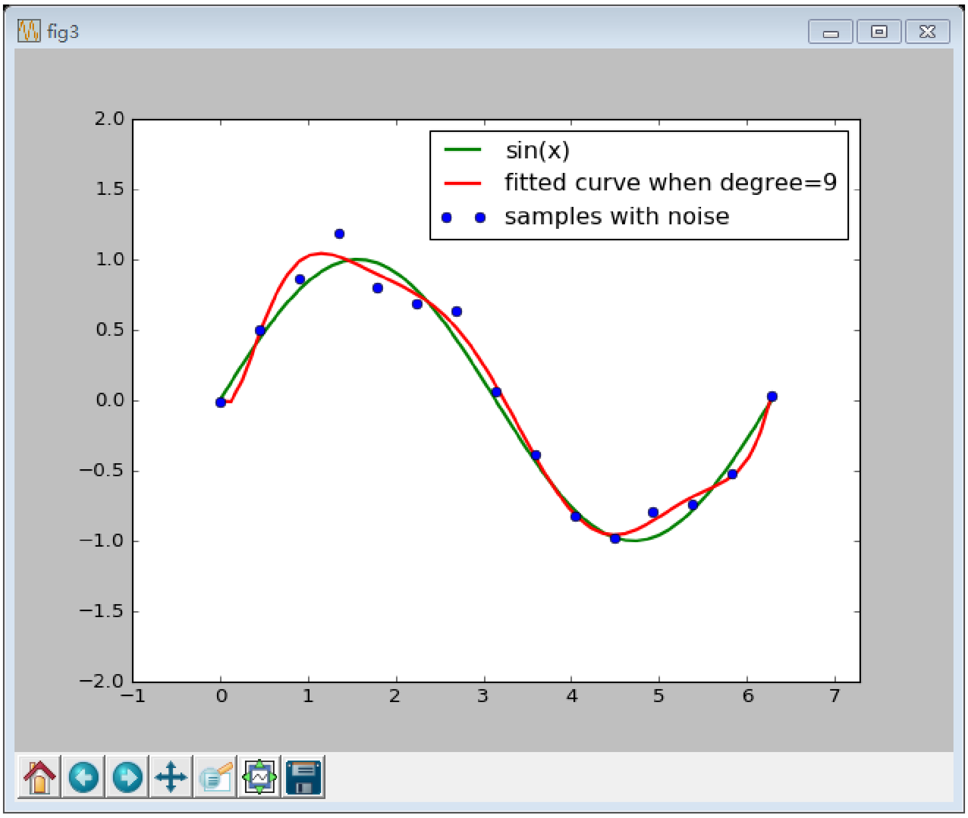
1. 当采样点个数为 10，degree=3 时，拟合曲线



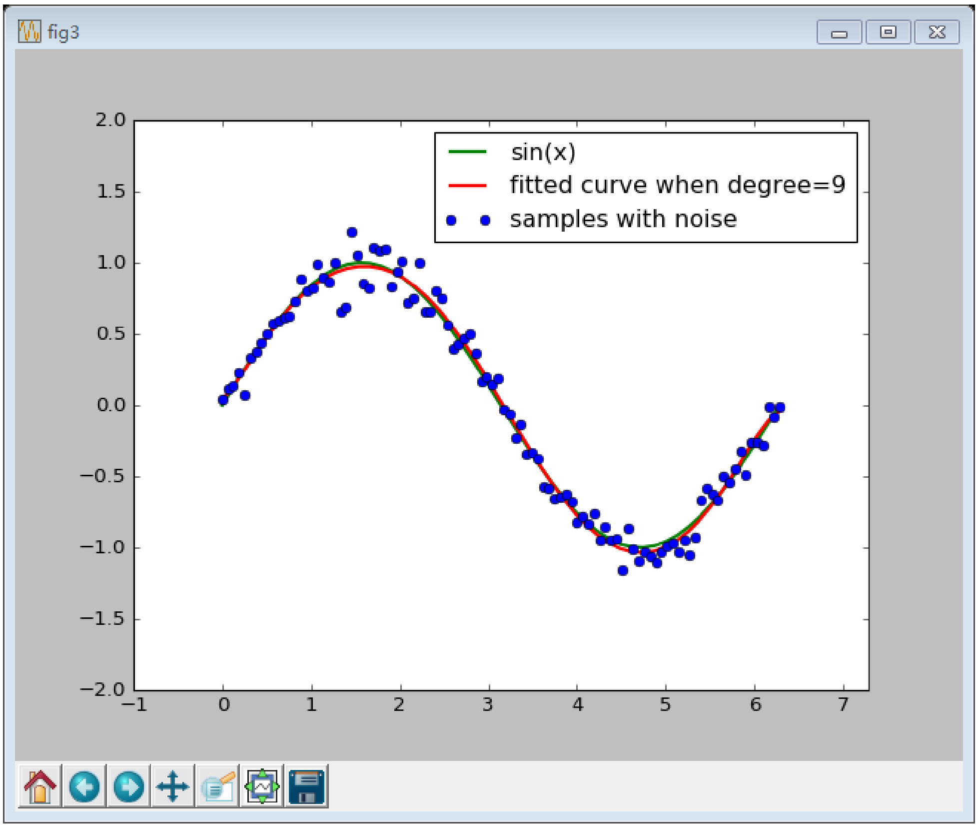

2. 当采样点个数为 10，degree=9 时，拟合曲线



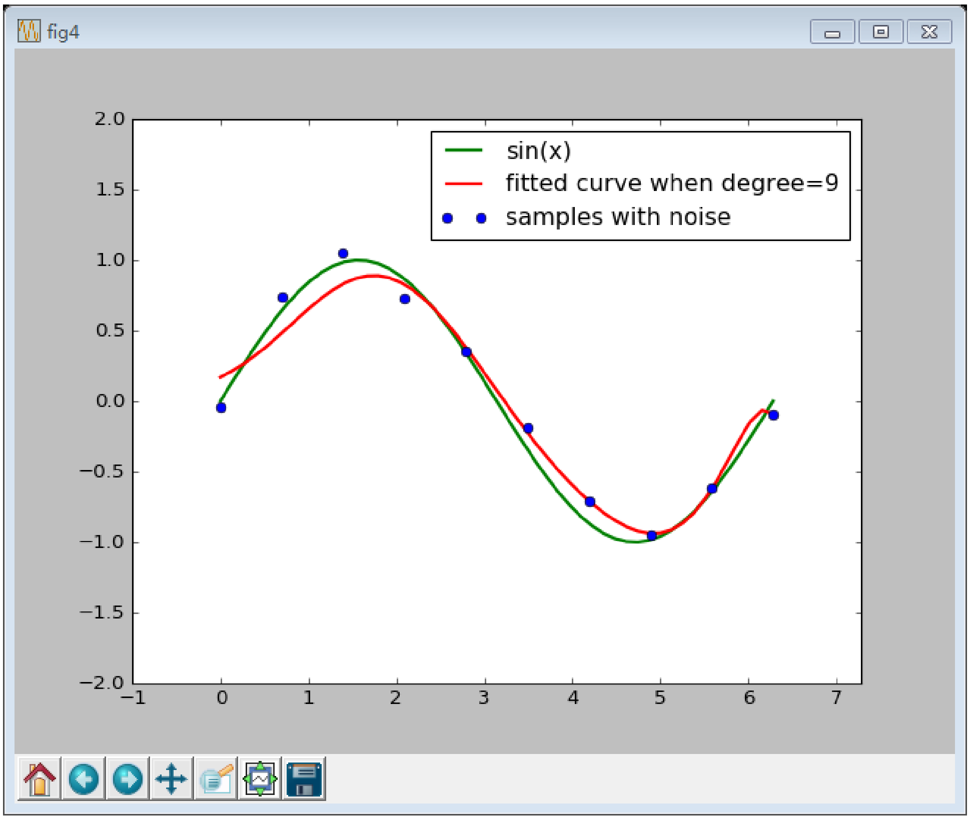

3. 当采样点个数为 15，degree=9 时，拟合曲线




4. 当采样点个数为 100，degree=9 时，拟合曲线




5. 当采样点个数为 10，degree=9 时，加入正则项时，拟合曲线




## 四、 实验代码及注释：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import leastsq
import math

#Generate polynomial function
def fitting_func(p, x):
    f = np.poly1d(p)
    return f(x)

#Generate residual between polynomial function and real value
def residuals(p, x, y):
    return y - fitting_func(p, x)

def residuals_norm(p, x, y):
    ret = y - fitting_func(p, x)
    ret = np.append(ret, np.sqrt(regularization)*p)
    return ret

m0 = 3
m1 = 9
m2 = 15
```

```
regularization = 1

#Generate samples
x_10 = np.linspace(0, 2*math.pi, 10)
x_15 = np.linspace(0, 2*math.pi, 15)
x_100 = np.linspace(0, 2*math.pi, 100)
x_display = np.linspace(0, 2*math.pi)
y_10 = np.sin(x_10)
y_15 = np.sin(x_15)
y_100 = np.sin(x_100)
y_display = np.sin(x_display)
y_noise_10 = [np.random.normal(0, 0.1) + i for i in y_10]
y_noise_15 = [np.random.normal(0, 0.1) + i for i in y_15]
y_noise_100 = [np.random.normal(0, 0.1) + i for i in y_100]

#Generate random polynomial parameters
p0 = np.random.randn(m0+1)
p1 = np.random.randn(m1+1)

#fitting the curve with leastsq()
plsq0 = leastsq(residuals, p0, args=(x_10, y_noise_10))
plsq1 = leastsq(residuals, p1, args=(x_10, y_noise_10))
plsq2 = leastsq(residuals, p1, args=(x_15, y_noise_15))
plsq3 = leastsq(residuals, p1, args=(x_100, y_noise_100))
plsq4 = leastsq(residuals_norm, p1, args=(x_10, y_noise_10))

fig1 = plt.figure("fig1")
plt.ylim(-2, 2)
plt.xlim(-1, 2*math.pi+1)
plt.plot(x_display, y_display, color="green", linewidth="2", label="sin(x)")
plt.plot(x_display, fitting_func(plsq0[0], x_display), color="red", linewidth="2", label="fitted
curve when degree=3")
plt.plot(x_10, y_noise_10 ,"o", color="blue", label="samples with noise")
plt.legend()
plt.show()


fig2 = plt.figure("fig2")
plt.ylim(-2, 2)
plt.xlim(-1, 2*math.pi+1)
plt.plot(x_display, y_display, color="green", linewidth="2", label="sin(x)")
plt.plot(x_display, fitting_func(plsq1[0], x_display), color="red", linewidth="2", label="fitted
curve when degree=9")
```

```
plt.plot(x_10, y_noise_10 ,"o", color="blue", label="samples with noise")
plt.legend()
plt.show()

fig3 = plt.figure("fig3")
plt.ylim(-2, 2)
plt.xlim(-1, 2*math.pi+1)
plt.plot(x_display, y_display, color="green", linewidth="2", label="sin(x)")
plt.plot(x_display, fitting_func(plsq2[0], x_display), color="red", linewidth="2", label="fitted
curve when degree=9")
plt.plot(x_15, y_noise_15 ,"o", color="blue", label="samples with noise")
plt.legend()
plt.show()

fig3 = plt.figure("fig3")
plt.ylim(-2, 2)
plt.xlim(-1, 2*math.pi+1)
plt.plot(x_display, y_display, color="green", linewidth="2", label="sin(x)")
plt.plot(x_display, fitting_func(plsq3[0], x_display), color="red", linewidth="2", label="fitted
curve when degree=9")
plt.plot(x_100, y_noise_100 ,"o", color="blue", label="samples with noise")
plt.legend()
plt.show()

fig4 = plt.figure("fig4")
plt.ylim(-2, 2)
plt.xlim(-1, 2*math.pi+1)
plt.plot(x_display, y_display, color="green", linewidth="2", label="sin(x)")
plt.plot(x_display, fitting_func(plsq4[0], x_display), color="red", linewidth="2", label="fitted
curve when degree=9")
plt.plot(x_10, y_noise_10 ,"o", color="blue", label="samples with noise")
plt.legend()
plt.show()
```