**哈尔滨工业大学计算机科学与技术学院**

**《机器学习》**

**Lab 1：多项式曲线拟合**

|  |  |  |
| --- | --- | --- |
| **姓名** | **班级** | **学号** |
| 楼雨京 | 1436101 | 1140310415 |

**1 实验目标**

掌握最小二乘法求解（无惩罚项的损失函数）、掌握加惩罚项（2范数）的损失函数优化、梯度下降法、共轭梯度法、理解过拟合、克服过拟合的方法(如加惩罚项、增加样本)

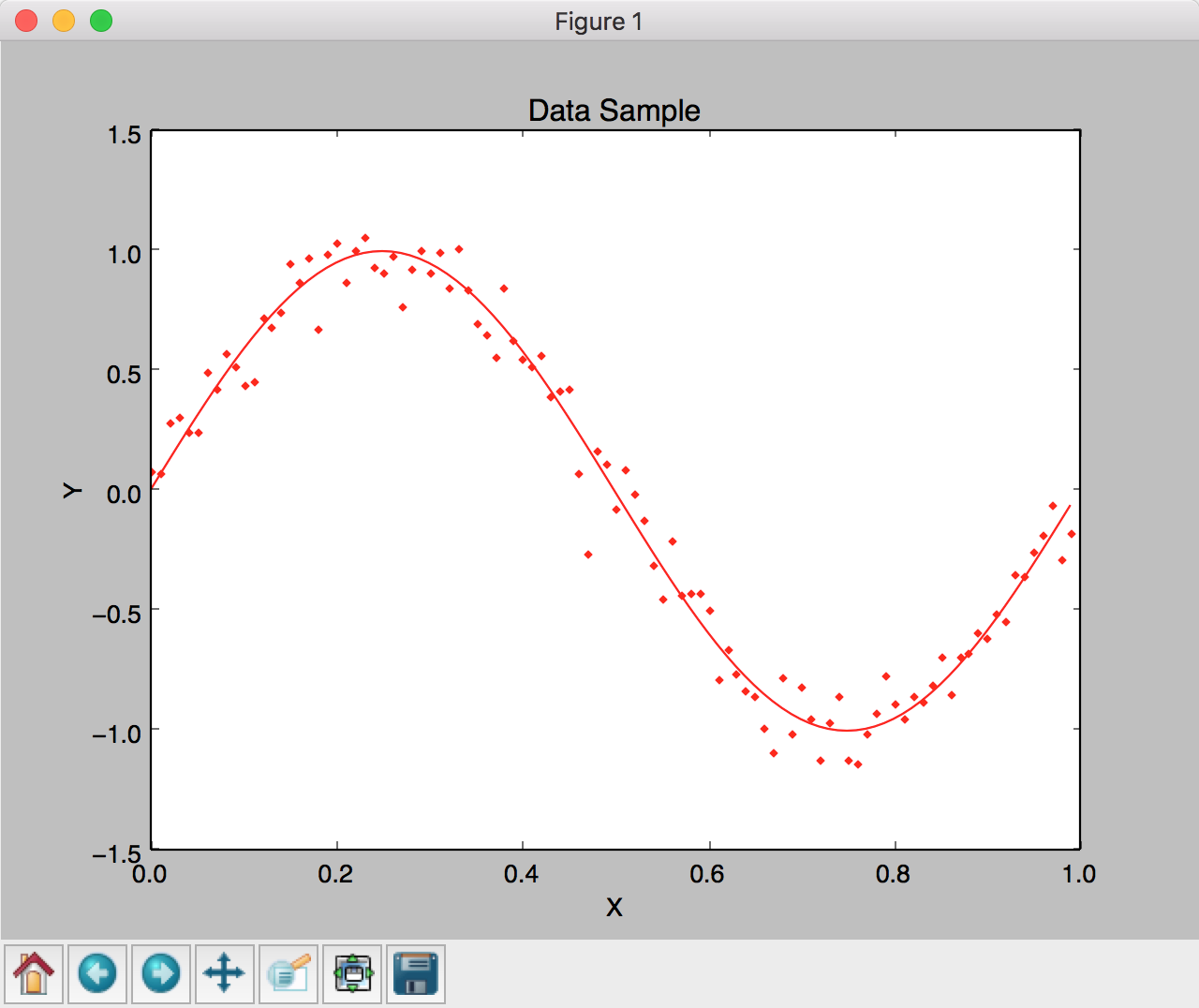
**2 实验环境**

python编译环境

**3 实验过程**

**3.1 生成数据，加入噪声**

本次实验生成的数据为区间 [0,1] 上一个周期的正弦形曲线上沿x轴方向均匀分布的数据点，数据量可以初始设定。然后对每一个点的y分量加入高斯噪声，数据点如下图。



根据以下两函数进行生成数据并加入噪声：

#函数功能：加入高斯噪声

def addGaussian(num, s, sigma):

mu = 0

for i in range(0,len(s)):

offset\_Guassian = random.gauss(mu, sigma)

s[i] = s[i] + offset\_Guassian

return s

#函数功能：用于生成样本数据

def generateData(num, xrange=1, yrange=1, sigma=0.12):

interv = xrange\*1.0/num

x = numpy.arange(0, xrange, interv)

t = yrange\*numpy.sin(2\*numpy.pi\*x)

y = yrange\*numpy.sin(2\*numpy.pi\*x)

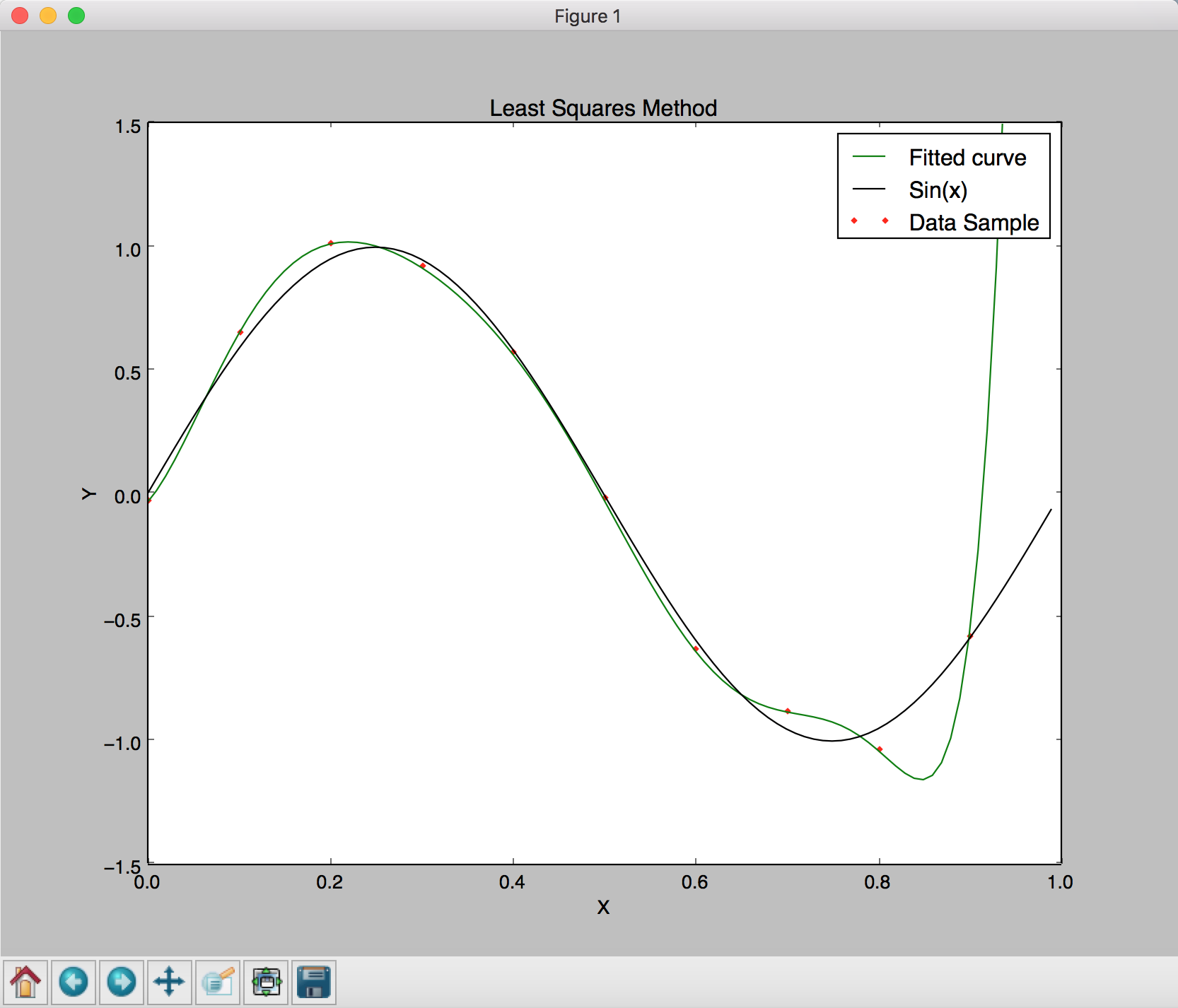
y = addGaussian(y.shape[0], y, sigma)

return x, y, t

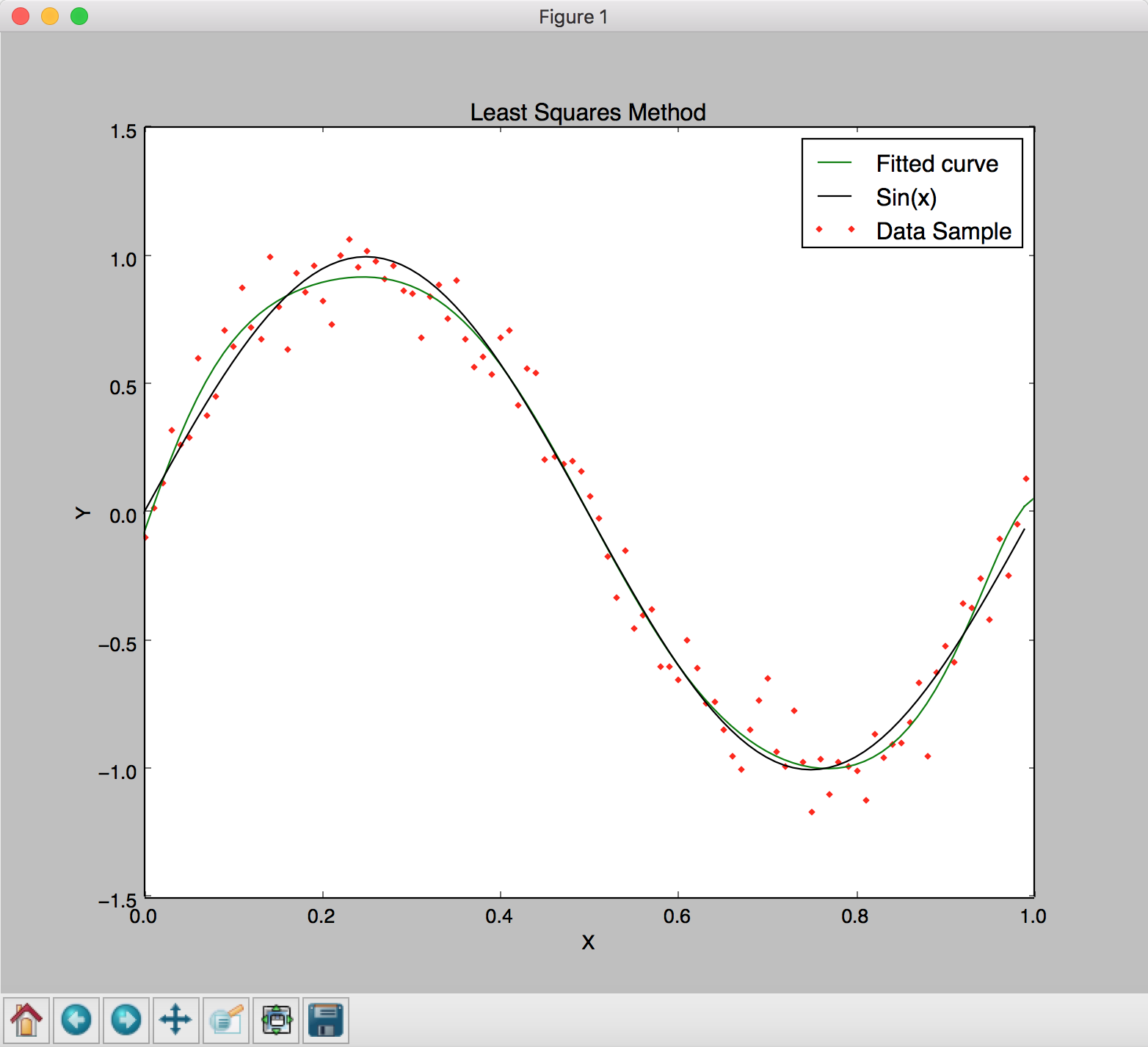
**3.2 最小二乘法多项式拟合曲线**

图中绿线为拟合出来的曲线，黑线为正弦形函数曲线，红点为数据样本点。

实验场景1：数据量为10，多项式阶数为9，未加惩罚项

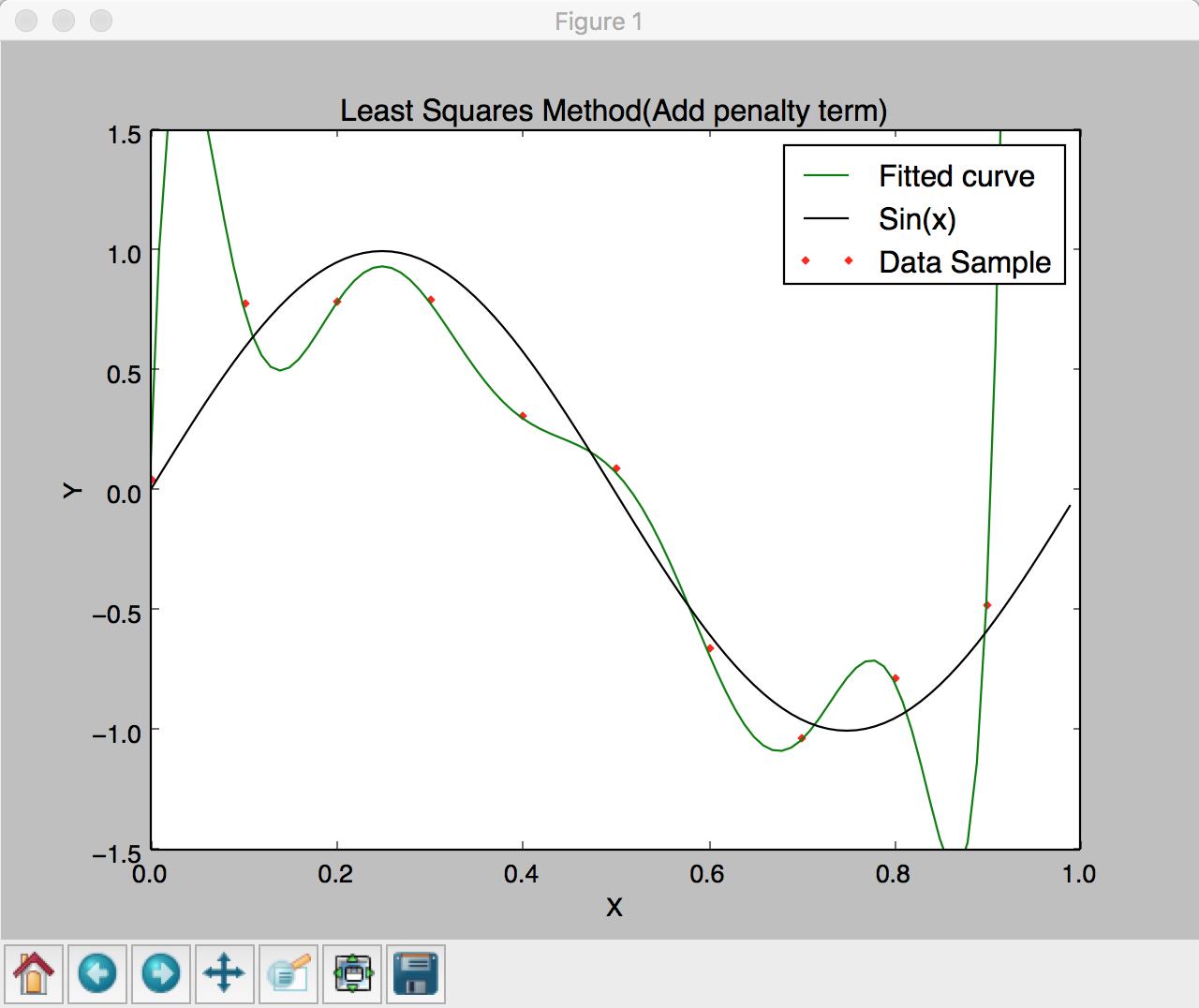


实验场景2：数据量为100，多项式阶数为9，未加惩罚项

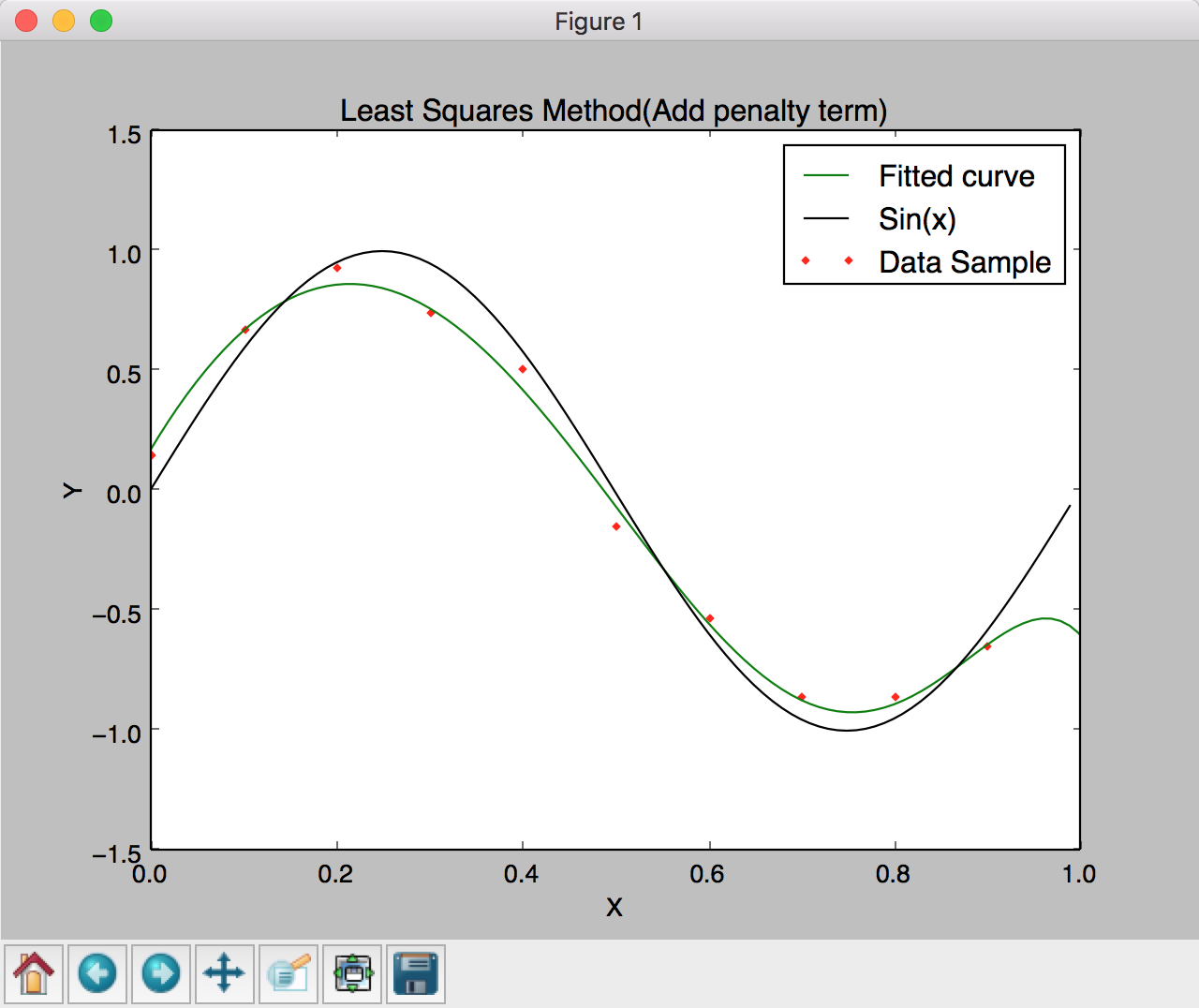


实验场景1和2对比，我们可以发现，当数据量较少的时候，拟合出的曲线误差较大，当数据样本数量恰好为多项式阶数多1时，根据最小二乘法的原理，可以准确地求解出多项式的每一个系数即权值，这时候拟合曲线经过了每一个样本点，是最“准确”的。但是反而实际的拟合效果却不好。这就是所谓的“过拟合”现象。为了避免过拟合的发生，观察实验场景2，数据量增加为100，曲线的拟合效果就比场景1好了很多。所以解决过拟合现象，增加数据样本量其中一种不错的方法。

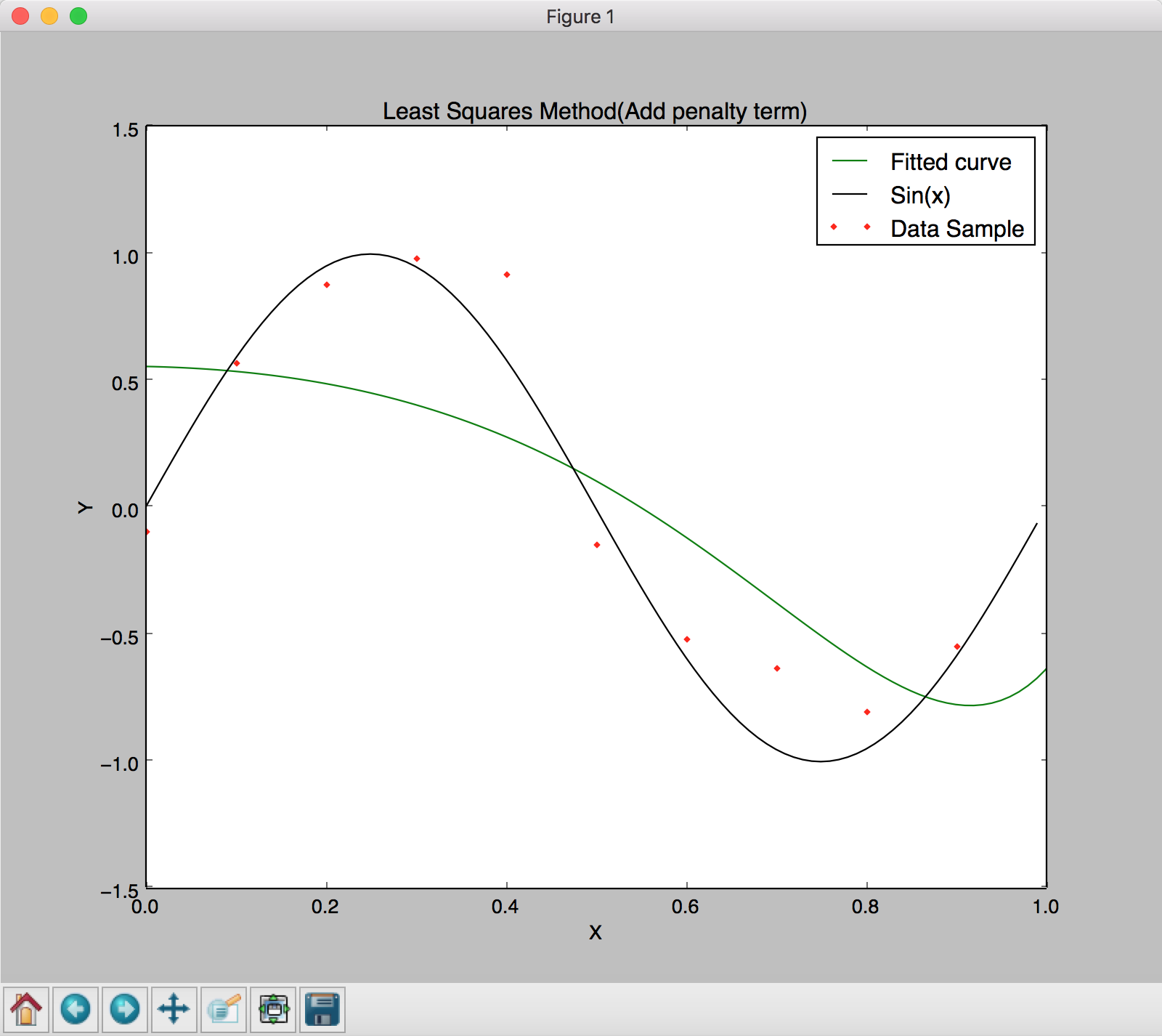
实验场景3：数据量为10，多项式阶数为9，加惩罚项，惩罚系数为0



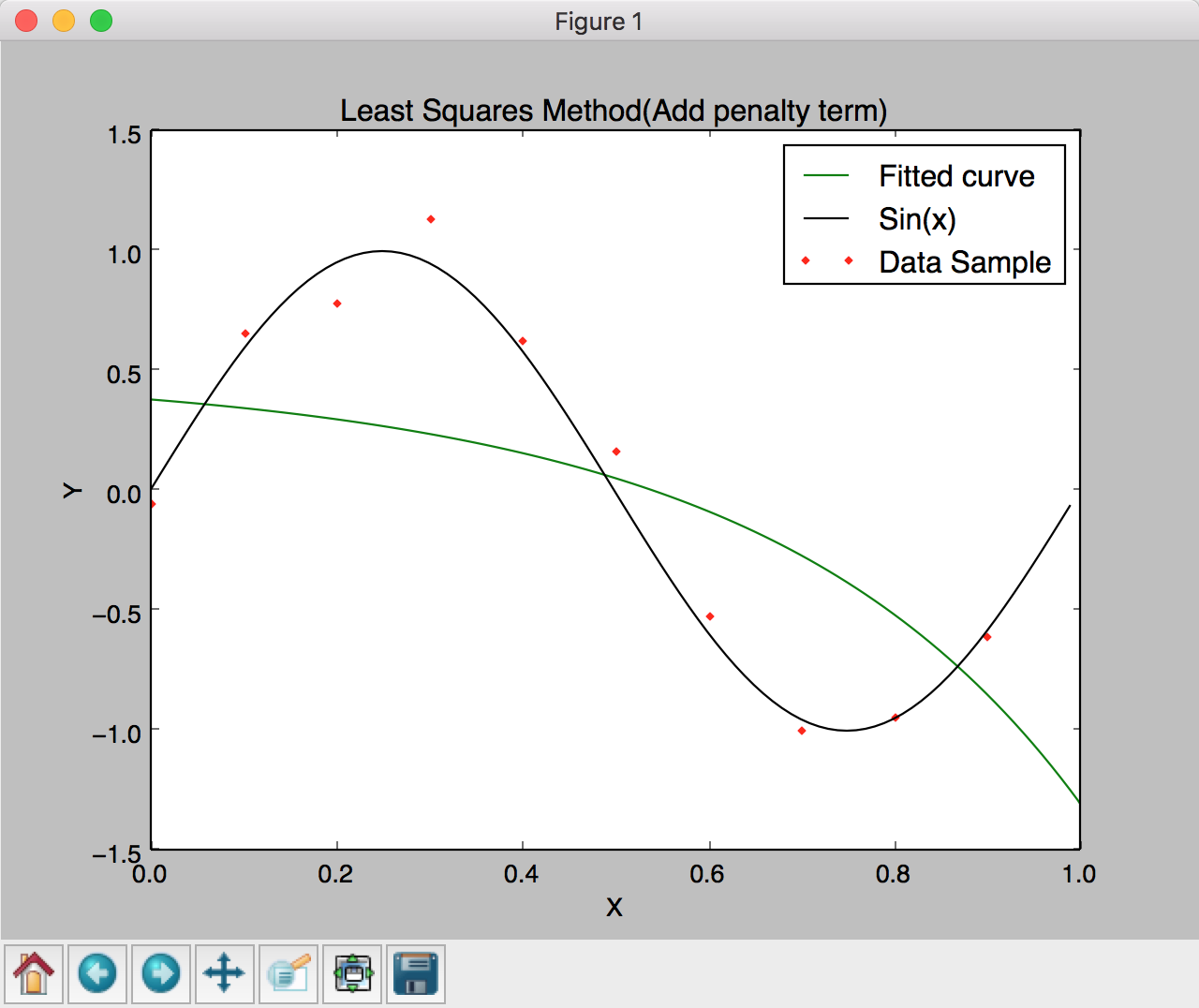
实验场景4：数据量为10，多项式阶数为9，加惩罚项，惩罚系数为0.0001



实验场景5：数据量为10，多项式阶数为9，加惩罚项，惩罚系数为0.1



实验场景6：数据量为10，多项式阶数为9，加惩罚项，惩罚系数为1.0



实验场景3，4，5，6为对损失函数加入惩罚项进行优化后的实验结果，对应的惩罚项参数分别为0，0.0001，0.1，1.0。当惩罚项系数为0时，和原模型相同，就相当于没有加惩罚项。当长发项系数为0.0001时，我们可以看到，曲线拟合得效果不错。但是当惩罚项系数再增加时，拟合曲线会变得越来越简单，到最后甚至成为一条直线。

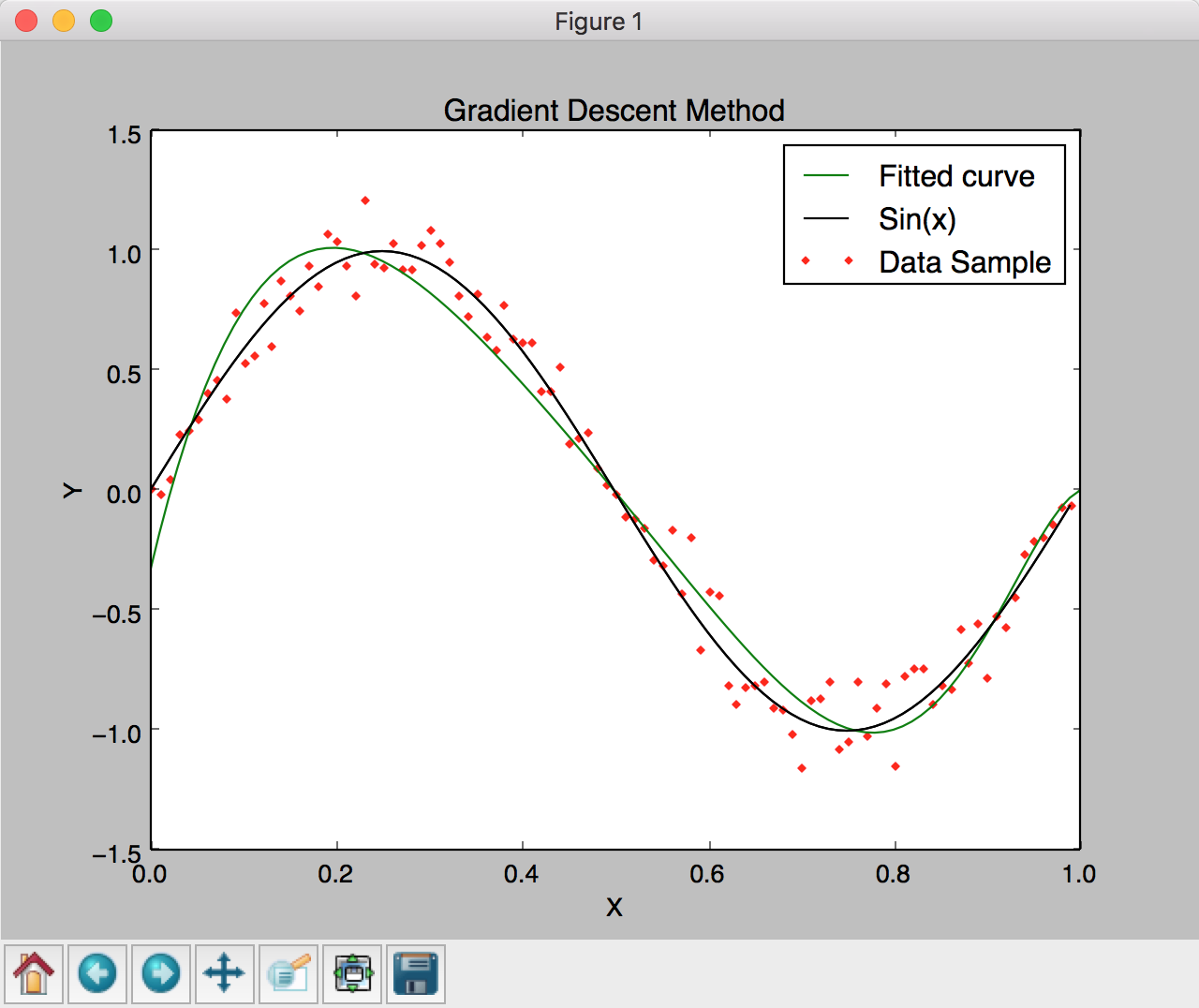
所以可以得出结论，当加入惩罚项，并且设定一个合适的系数，我们可以很好地解决过拟合问题，且和原问题匹配，所以这是另一种解决过拟合的方法。我们再分析可以发现，当惩罚项系数很小时，会逐渐退化成原模型；当惩罚项系数较大时，模型复杂度反而会降低。

**3.2 最速下降法多项式拟合曲线**

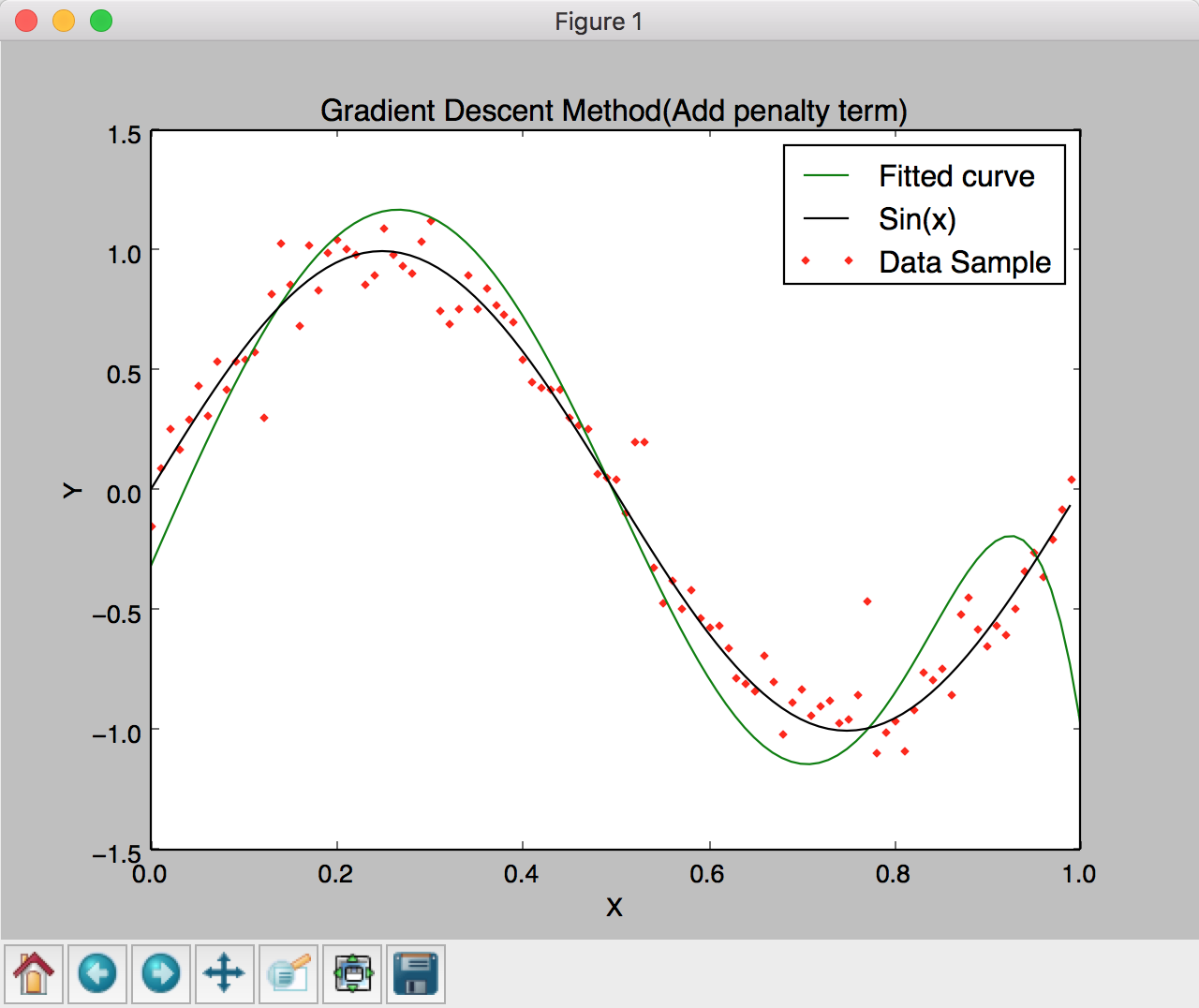
利用最速下降法进行曲线拟合，通过给定步长，不断逼近最优解。在一开始进行曲线拟合时，我采用给定步长的方法进行曲线拟合，这样的好处在于在迭代过程当中不用改变步长，但是随之而来的弊端就是如果要拟合出一个误差尽可能小的多项式，就必须在最初设定一个非常小的步长，这会导致迭代过程变得非常的缓慢。但如果初始步长太大，就可能导致错过截小值点而不能收敛。

为了解决这一问题，最后我采用了一种折半步长的方法，即最初设置一个较大的步长，然后在迭代过程中，当经过一次迭代后，如果梯度的范数比原梯度的范数还大，则表示已经越过了极小值点，这时候就应该回到迭代前的状态，然后将步长缩短为原步长的一半。这么做就很好地解决了迭代慢以及误差大这两者之间的矛盾。由于在极小值点，梯度的范数应该为零，所以可以通过判断梯度范数小于某个小值时迭代结束。

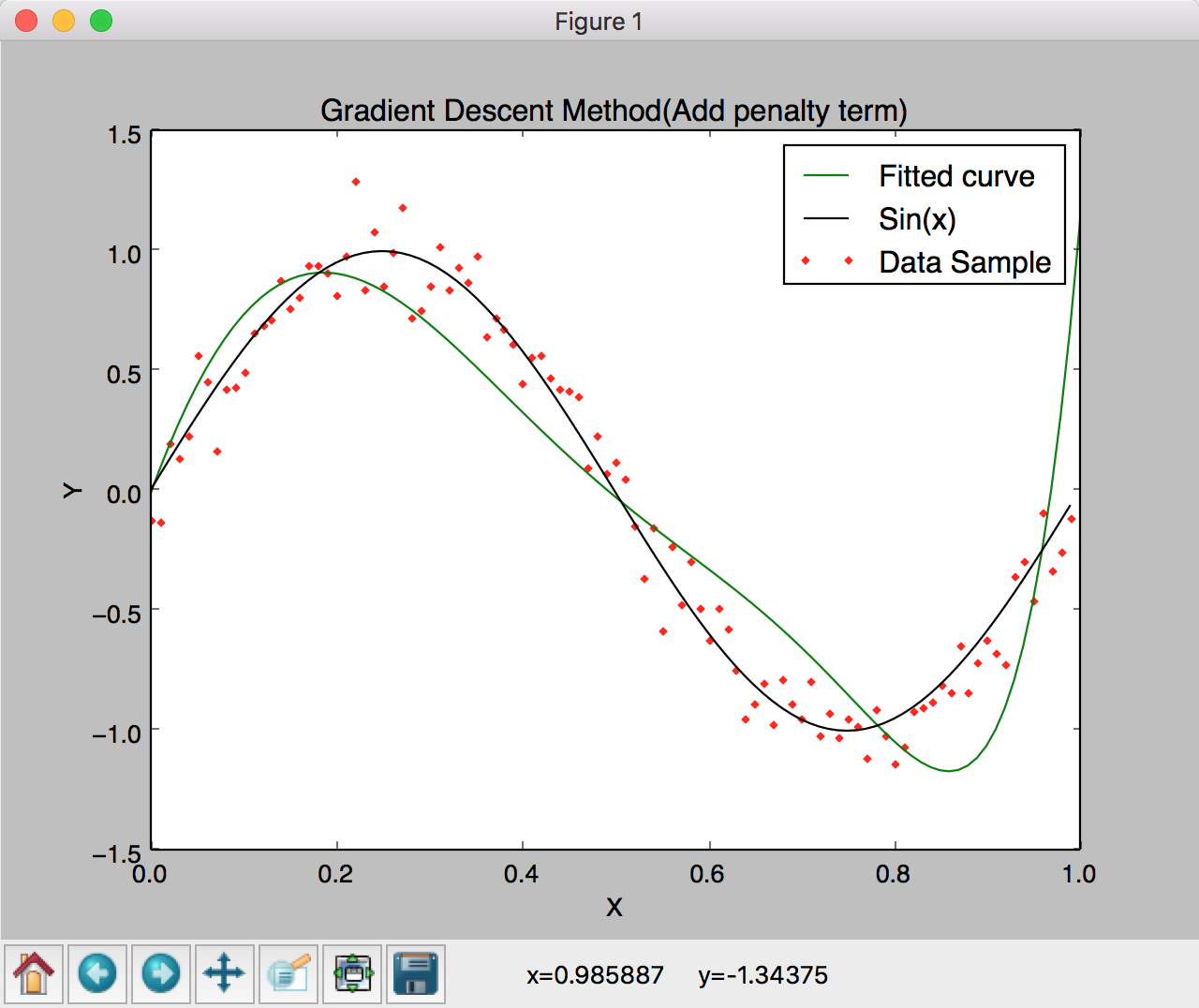
实验场景1：数据量为100，多项式阶数为9，未加惩罚项



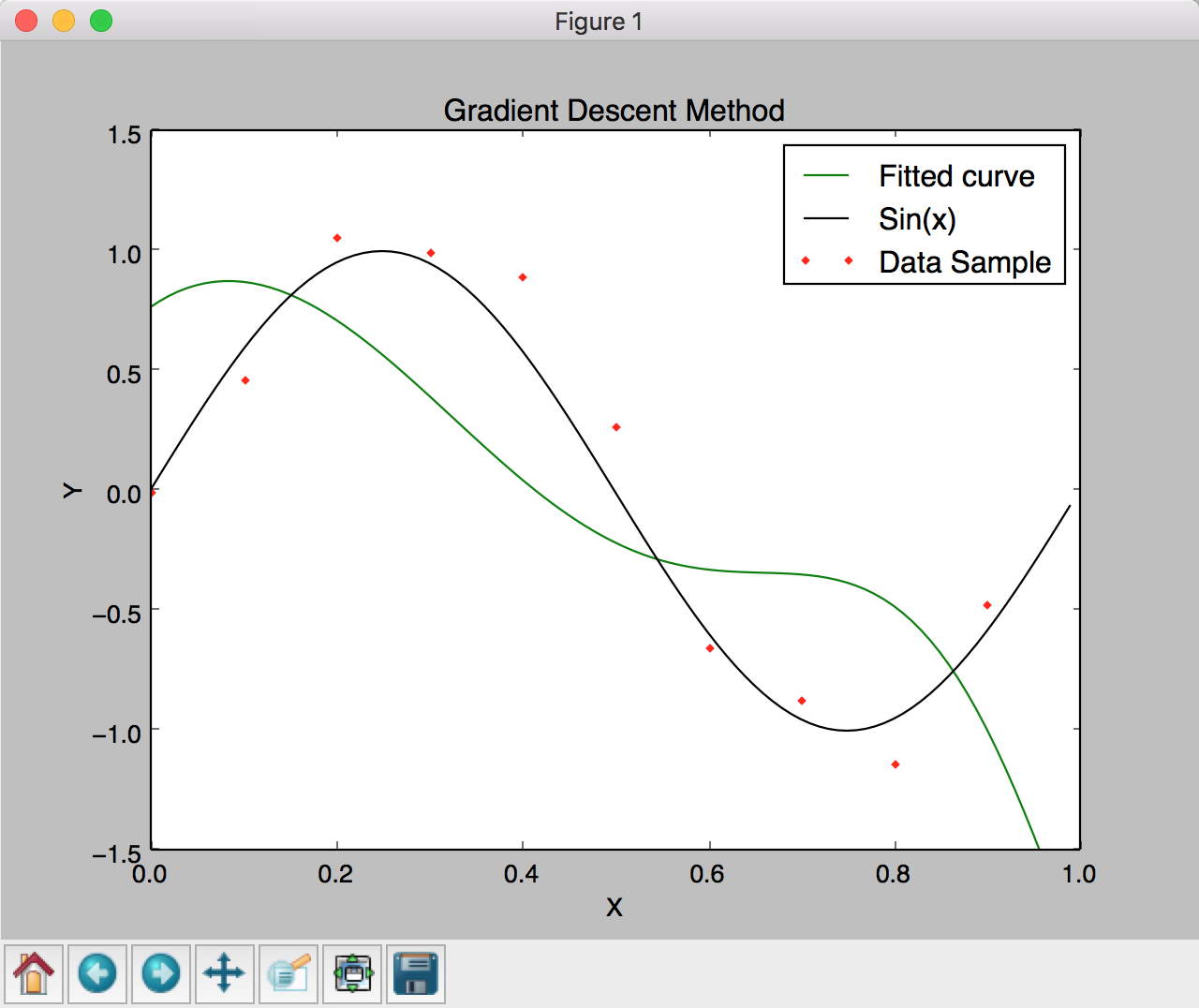
实验场景2：数据量为100，多项式阶数为9，加惩罚项，惩罚项系数0.001



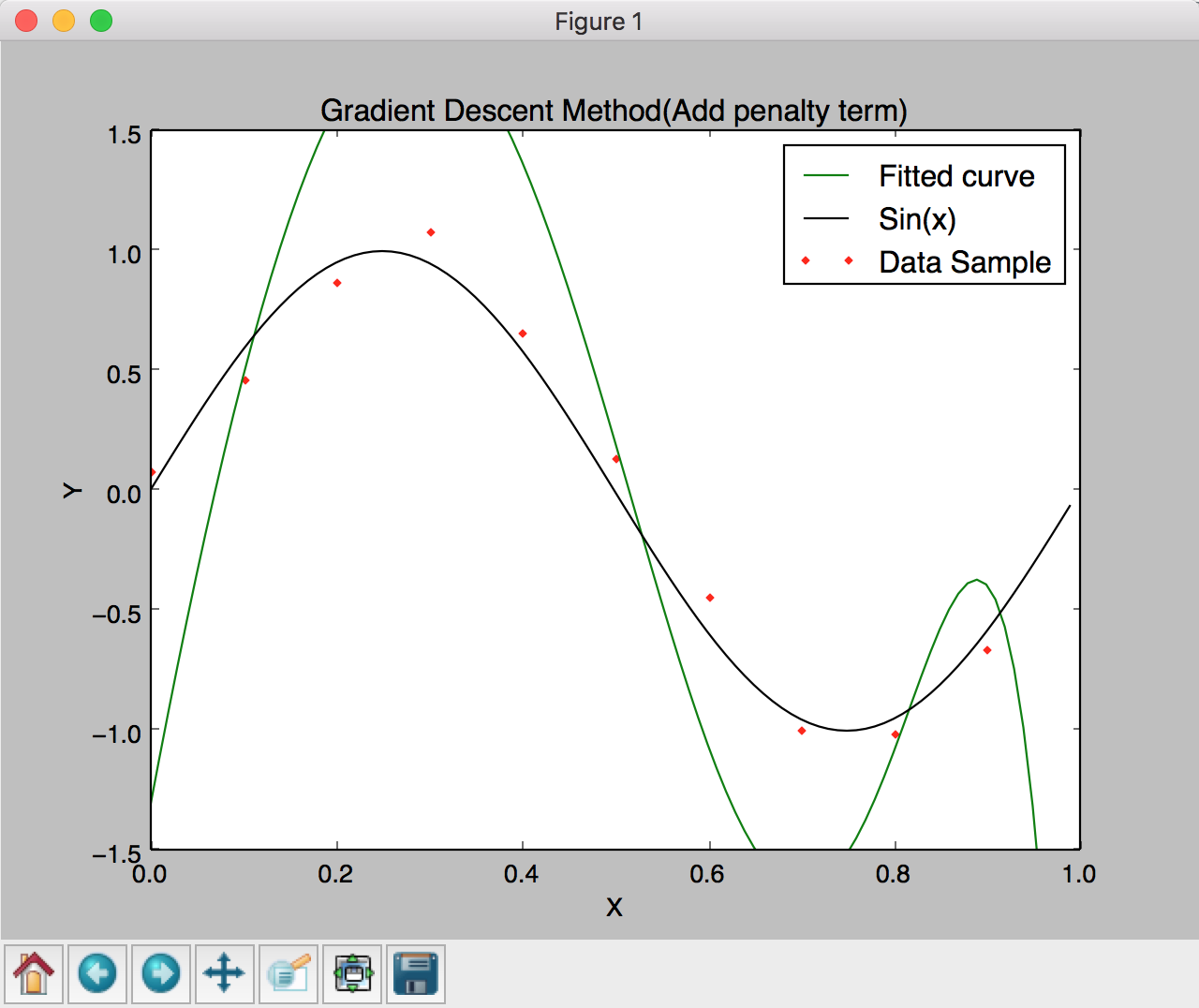
实验场景3：数据量为100，多项式阶数为9，加惩罚项，惩罚项系数1



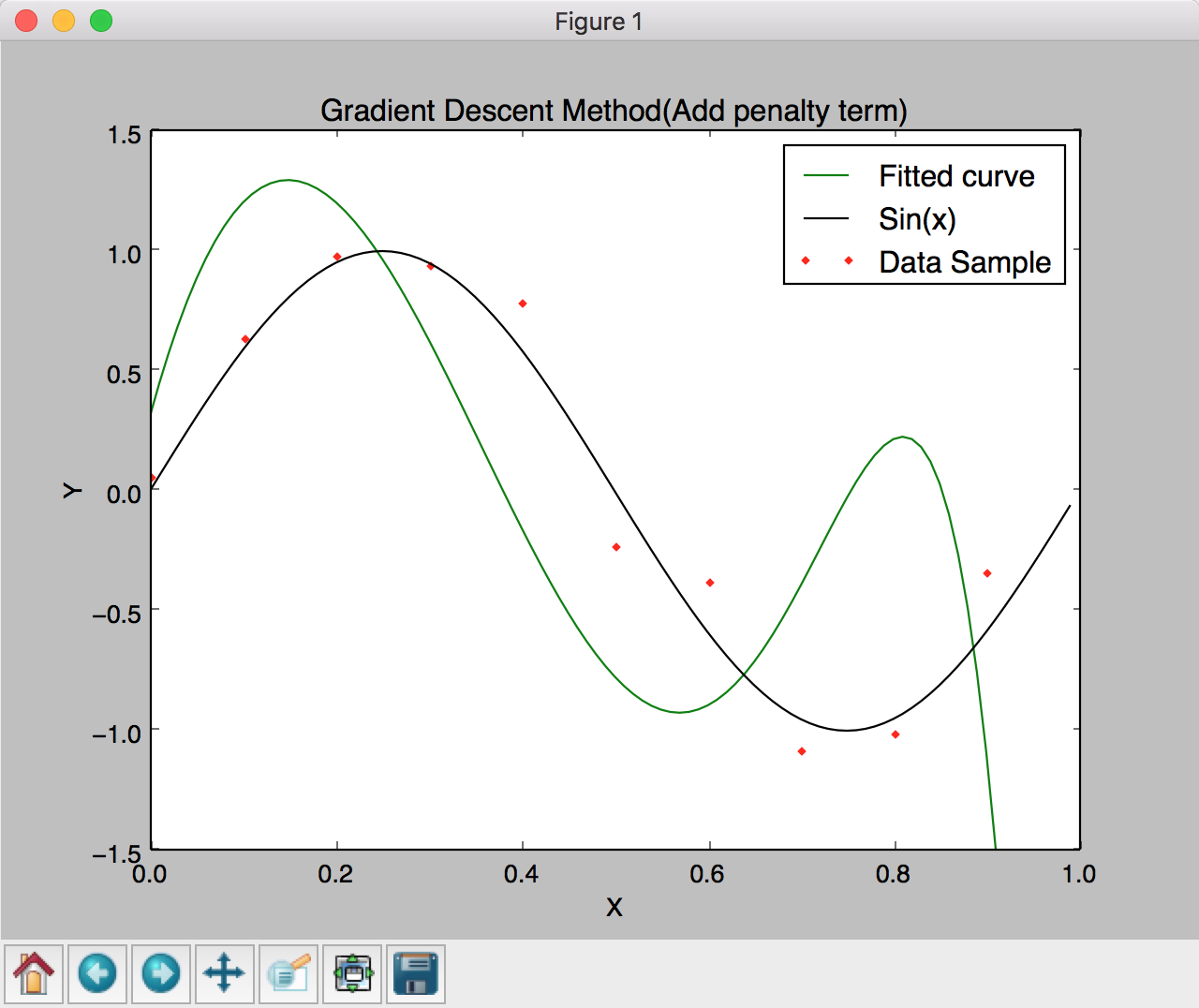
实验场景4：数据量为10，多项式阶数为9，未加惩罚项



实验场景5：数据量为10，多项式阶数为9，加惩罚项，惩罚项系数0.001



实验场景6：数据量为10，多项式阶数为9，加惩罚项，惩罚项系数为1

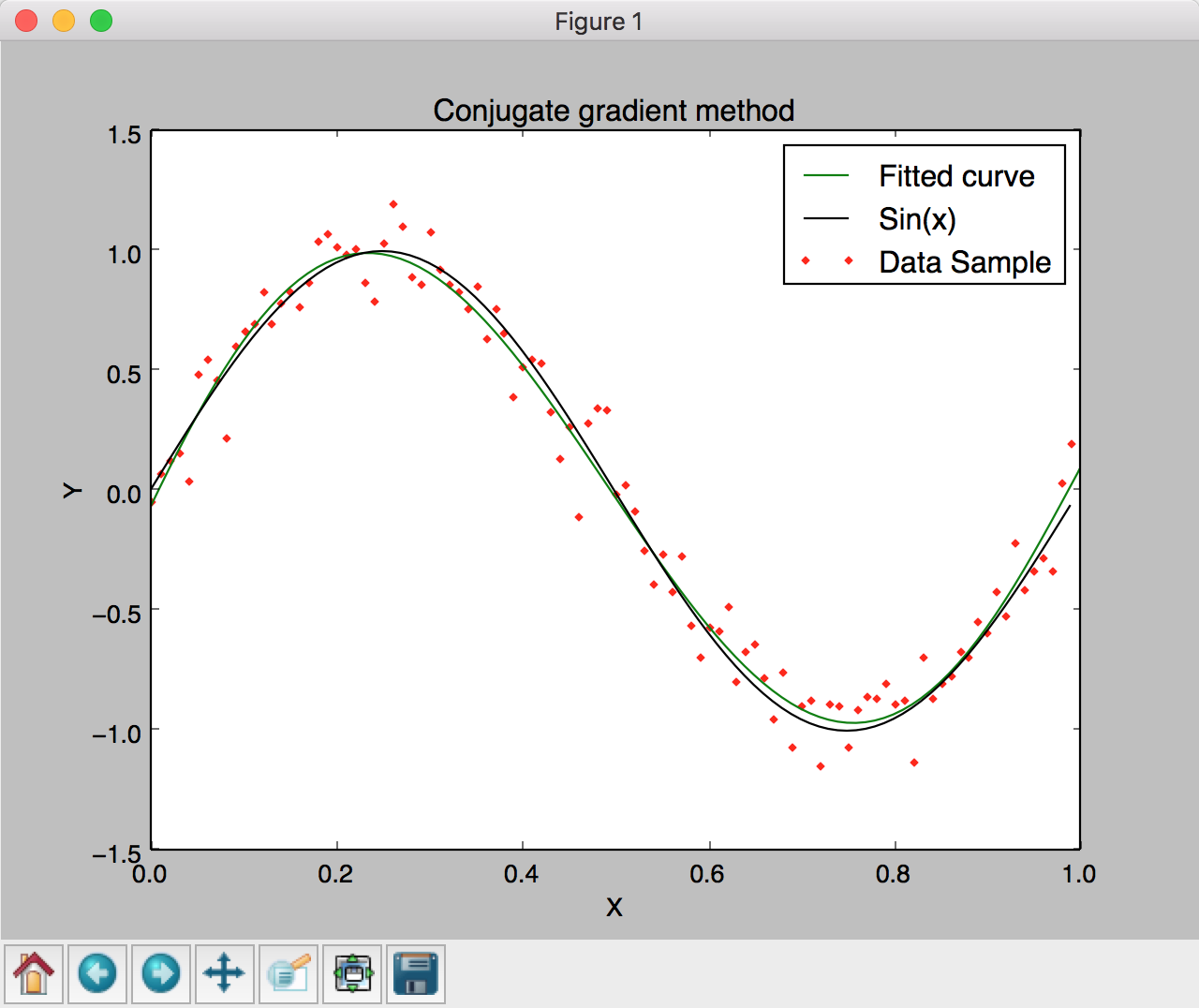


从以上实验数据可以看出，使用梯度下降法进行拟合曲线时，在数据量大的情况下，拟合情况较好，加惩罚项以后，拟合度会有少许提高。但是当数据量很少的时候，拟合情况不太乐观，这时通过加惩罚项可以较好地提高拟合度，但是还是有一定的误差。所以采用梯度下降法时，可以适当地增加数据样本量，这样可以得到较好的结果。

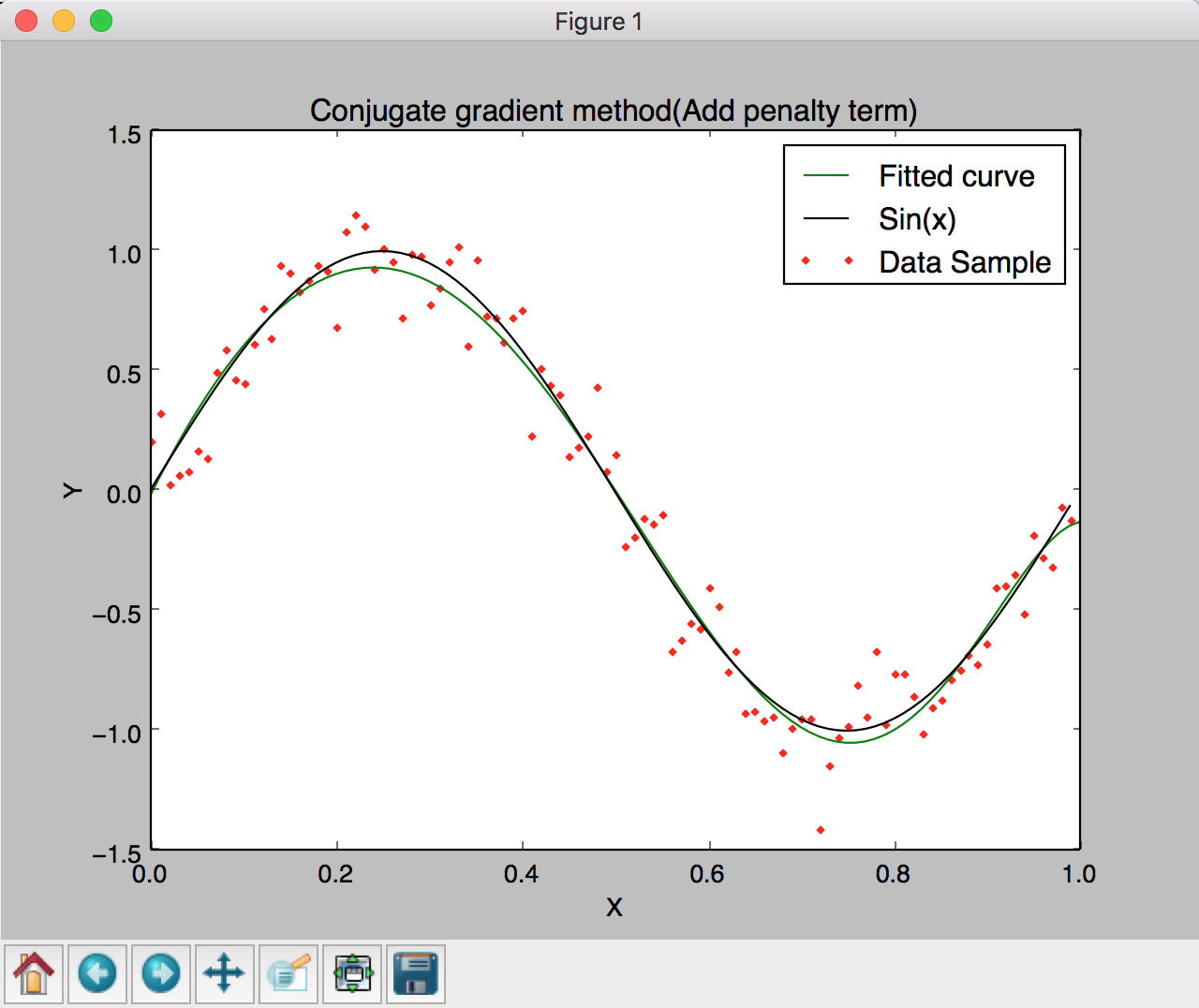
**3.2 共轭梯度法多项式拟合曲线**

共轭梯度法相比较梯度下降法，迭代次数大大减少，一般迭代四五次就能够得到拟合的结果，并且拟合度很高，是一种值得推荐的拟合曲线的方法（效果效率兼得的一种方法）。

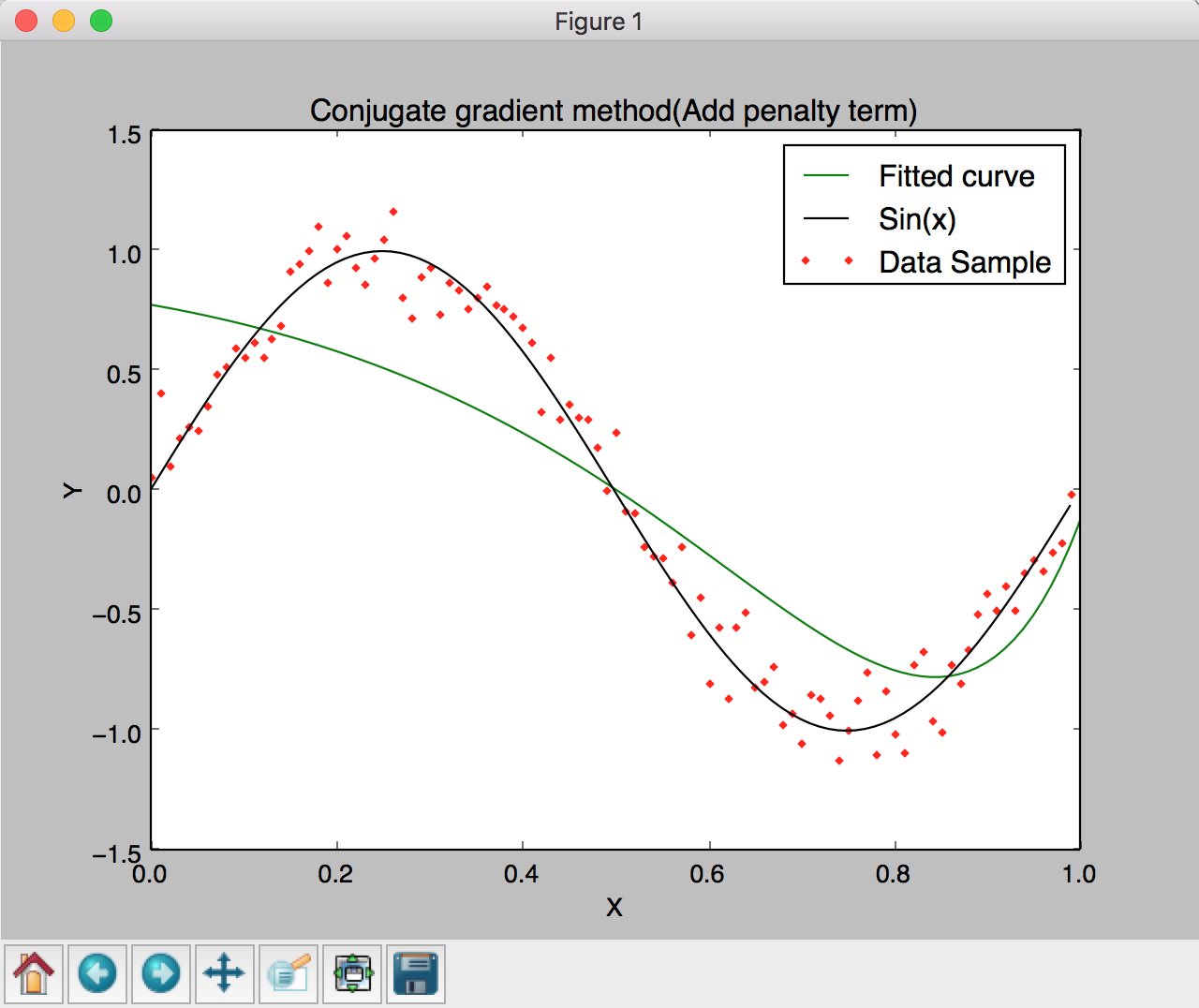
实验场景1：数据量为100，多项式阶数为9，未加惩罚项



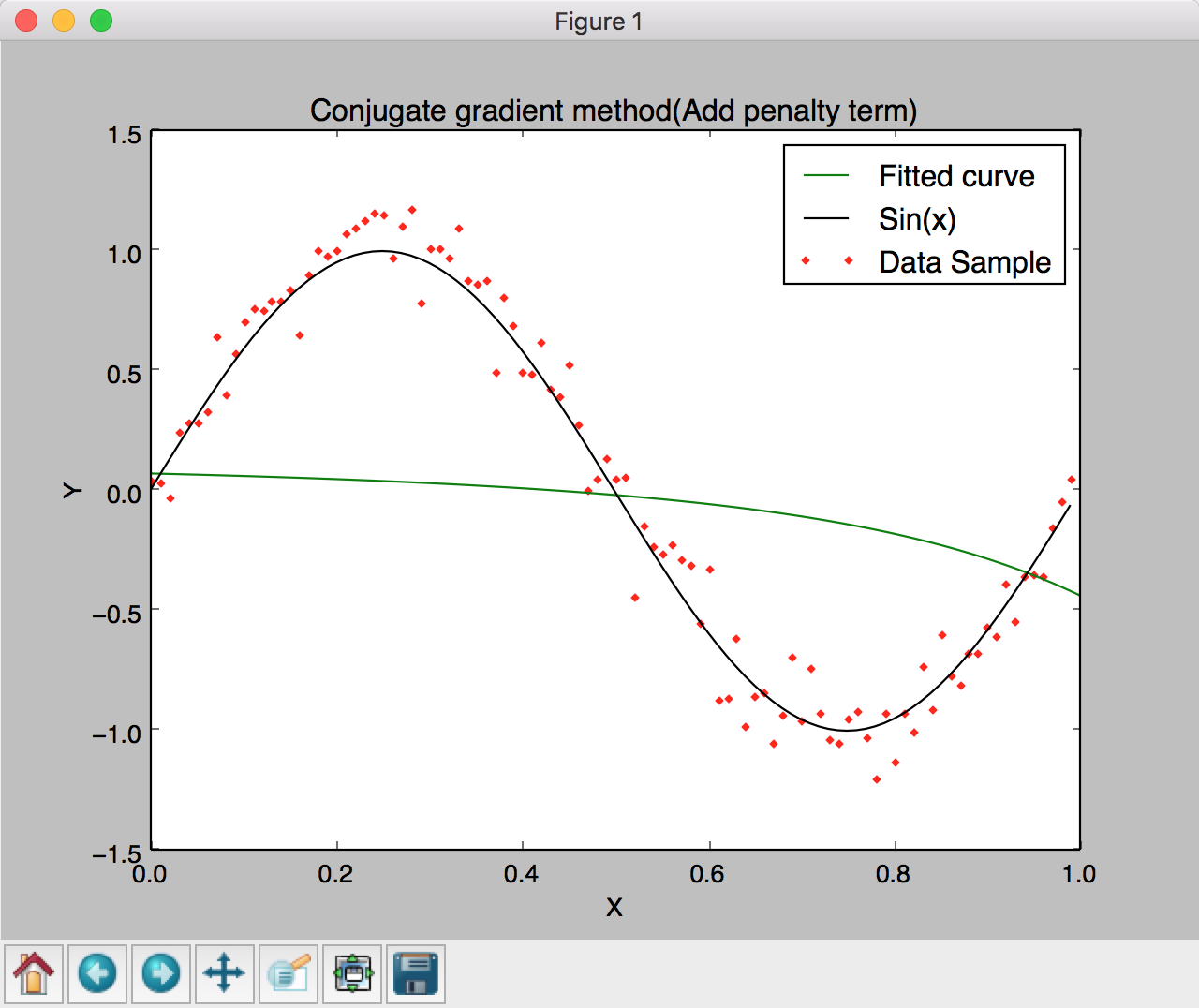
实验场景2：数据量为100，多项式阶数为9，加惩罚项，惩罚项系数0.001



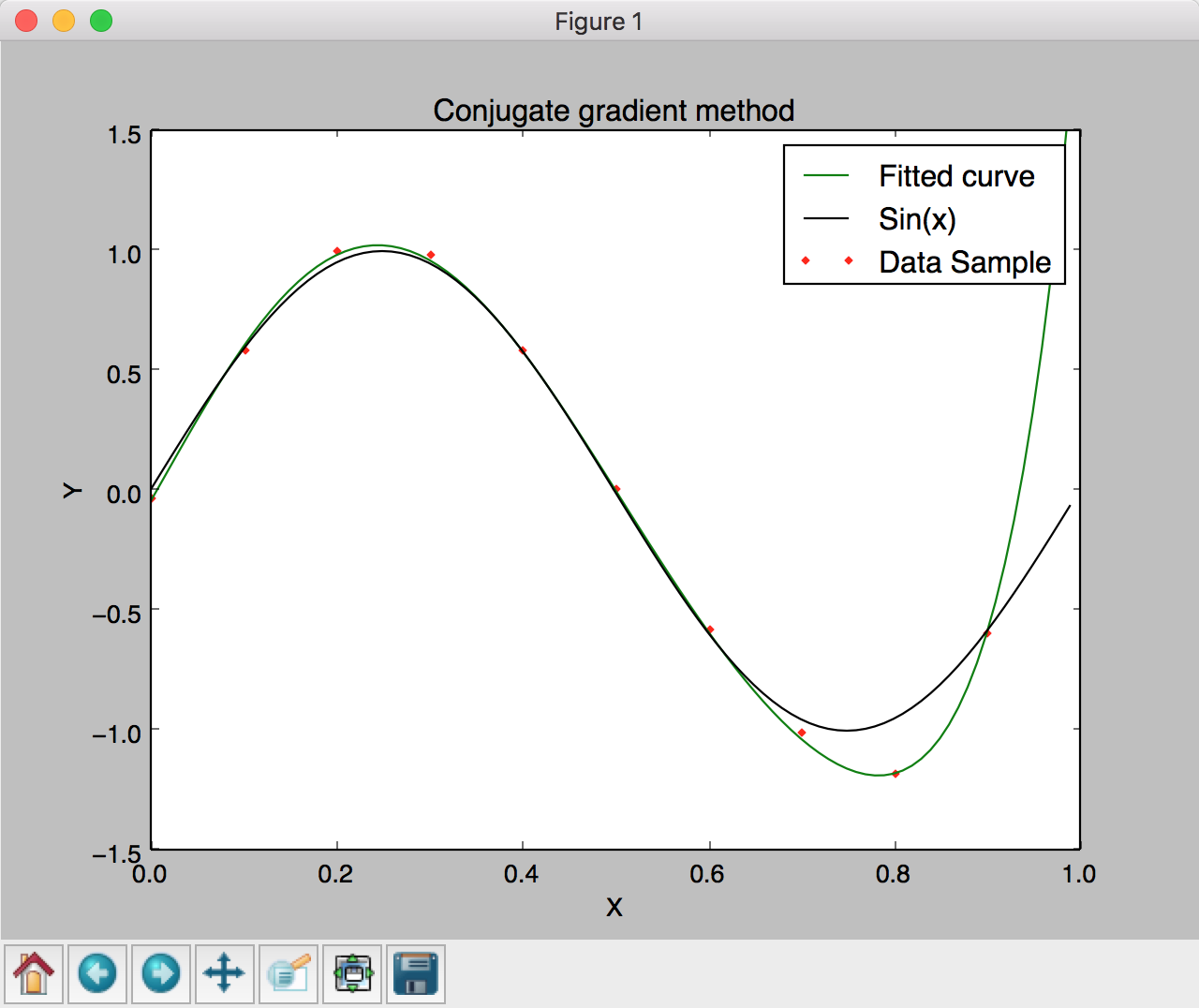
实验场景3：数据量为100，多项式阶数为9，加惩罚项，惩罚项系数1



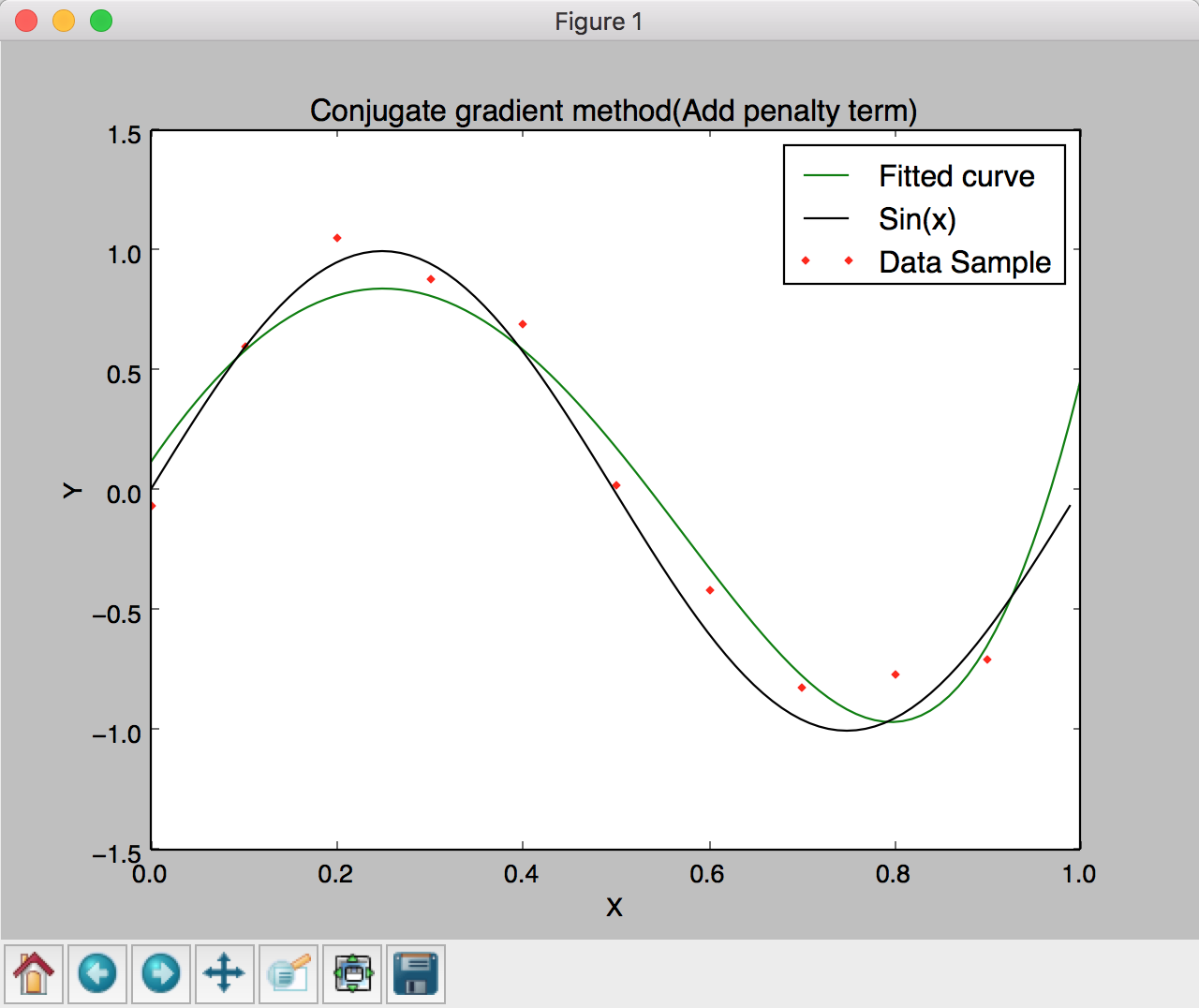
实验场景4：数据量为100，多项式阶数为9，加惩罚项，惩罚项系数100



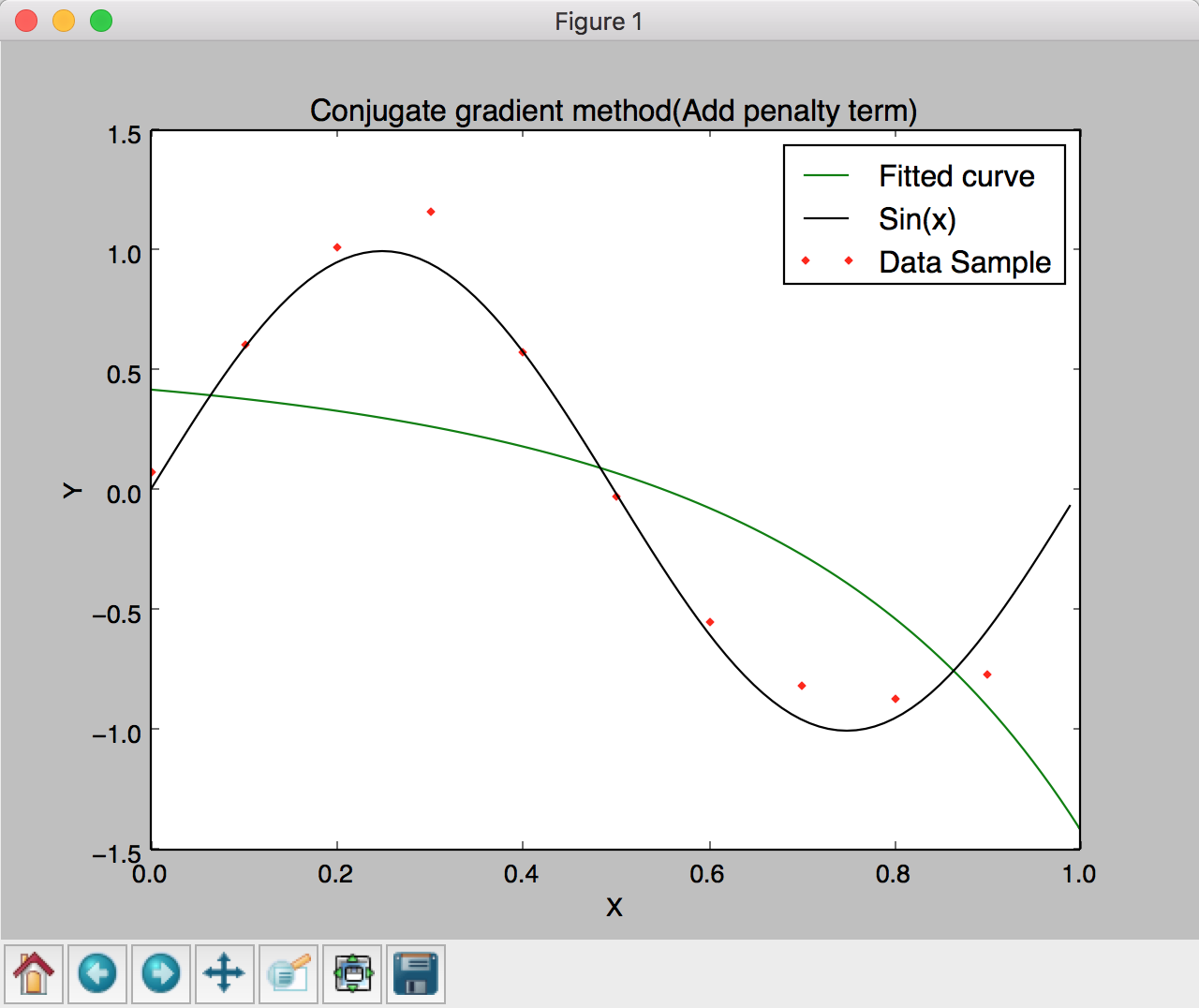
实验场景5：数据量为10，多项式阶数为9，未加惩罚项



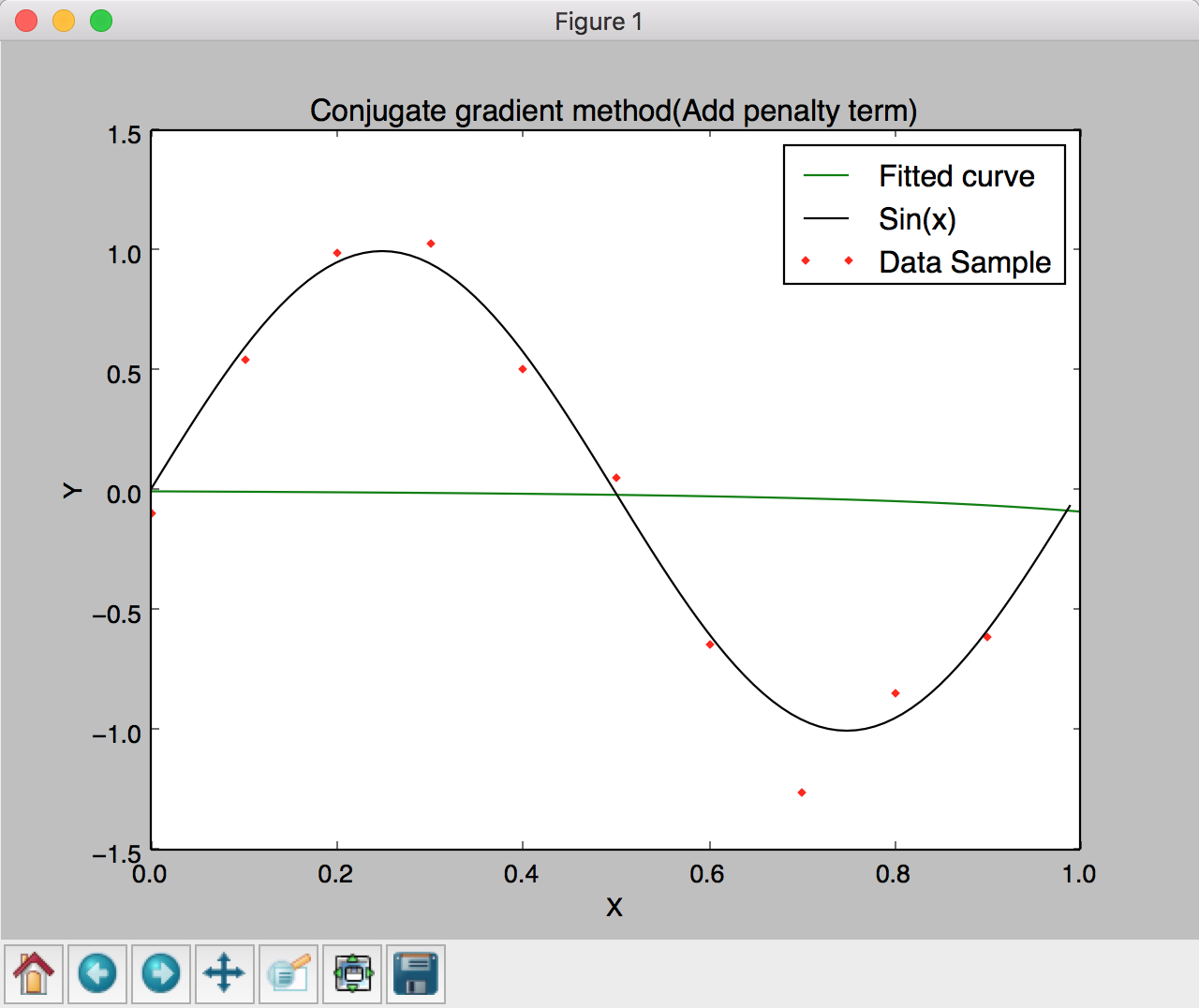
实验场景6：数据量为10，多项式阶数为9，加惩罚项，惩罚项系数0.001



实验场景7：数据量为10，多项式阶数为9，加惩罚项，惩罚项系数1



实验场景8：数据量为10，多项式阶数为9，加惩罚项，惩罚项系数100



对上述8个实验结果分析，可以看出共轭梯度法在未加惩罚项的情况下，拟合的效果都非常的好，无论样本数量是大是小。加上一个系数较小的惩罚项后，可以看到拟合效果一就很好，因为系数小时，和原模型很接近，由于原模型拟合效果好，所以相应小系数的也不错。当增大惩罚项系数时，可以发现，模型逐渐简化。

**4 实验总结**

本次实验，我学习了最小二乘法，梯度下降法，共轭梯度法三种多项式拟合曲线的方法，根据数据样本学习出一条拟合曲线来进行数据预测。同时，还学会了通过加惩罚项，或者改变数据样本规模来改善拟合效果。三种方法各有利弊。最小二乘法拟合曲线，思想简单，便于实现，通过求解线性方程组的方式来进行曲线的拟合，但是当数据量较少时，会产生过拟合的现象，这时我们可以通过增加样本数量或者增加一个带有适当系数的惩罚项进行优化，可以很好地解决过拟合问题。梯度下降法，每次沿着最速下降方向寻找最优解，可以当作一种贪心思想，但是步长的选择是这一方法关键的问题，虽然可以通过折半步长的方法进行优化，但是迭代速度还是比较慢。我个人比较喜欢共轭梯度法，迭代次数少，而且拟合效果很棒，就是背后的数学公式和原理稍微复杂些，但是当我理解了原理后，还是能够比较容易地实现的。

总的来说这次实验收获很大，学会了这几种拟合优化方法，相信在将来的ML学习过程中会经常地用到。

**5 附录**

提交代码中共有7个版本：

version1.py：实现最小二乘法

version2.py：在最小二乘法的基础上加入了惩罚项

version3.py：实现梯度下降法

version3.1.py：通过折半步长的方法实现梯度下降法

version4.py：在梯度下降法的基础上加入了惩罚项

version5.py：实现共轭梯度法

version6.py：在共轭梯度法基础上加入了惩罚项