



哈爾濱工業大學

Harbin Institute of Technology

机器学习实验报告

多项式拟合正弦曲线

课程名称: 机器学习

学院: 计算学部

专业: 计算机科学与技术

学号: 1180300811

姓名: 孙骁

指导老师: 刘扬

2020 年 10 月 6 日

一、 实验目的和要求

1.1 实验目的

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）。

1.2 实验要求

- (1) 生成数据，加入噪声；
- (2) 用高阶多项式函数拟合曲线；
- (3) 用解析解求解两种 loss 的最优解（无正则项和有正则项）；
- (4) 优化方法求解最优解（梯度下降，共轭梯度）；
- (5) 用你得到的实验数据，解释过拟合；
- (6) 用不同数据量，不同超参数，不同的多项式阶数，比较实验效果；
- (7) 语言不限，可以用 matlab, python. 求解解析解时可以利用现成的矩阵求逆. 梯度下降，共轭梯度要求自己求梯度，迭代优化自己写. 不许用现成的平台，例如 pytorch, tensorflow 的自动微分工具.

二、 实验环境

- (1) Anaconda 4.8.4
- (2) Python 3.7.4
- (3) PyCharm 2019.1 (Professional Edition)
- (4) Windows 10 1909

三、 实验原理

3.1 多项式拟合函数

如果 $f(x)$ 在点 x_0 的某个邻域 $U(x_0)$ 内是无穷次连续可微的，记为 $f(x) \in C^\infty(U(x_0))$ ，则 $f(x)$ 可以展开为以下的幂级数：

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + \cdots \quad (1)$$

称式 (1) 为函数 $f(x)$ 在 x_0 诱导出的泰勒级数，特别的，当 $x_0 = 0$ 时，称式 (2) 为 $f(x)$ 诱导出的麦克劳林级数。

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \cdots + \frac{f^{(n)}(0)}{n!}x^n + \cdots \quad (2)$$

由泰勒级数的收敛性定理可以，当拉格朗日型余项 $R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{(n+1)}$ 满足

$$\lim_{n \rightarrow \infty} R_n(x) = 0, \forall x \in U(x_0) \quad (3)$$

时，则我们可以使用高阶的多项式拟合函数，由函数幂级数展开的唯一性可知，在 x 的某邻域内，若函数 $f(x)$ 的各阶导数存在，且满足泰勒级数的收敛性定理时，则 $f(x)$ 可以展开为幂级数，且展开式唯一。

显然函数 $f(x) = \sin(2\pi x)$ 满足泰勒级数的收敛性定理，且满足式(3)，则可以用多项式拟合函数 $f(x) = \sin(2\pi x)$ 。

3.2 拟合函数参数确定

在 m 阶多项式

$$y(x, w) = w_0 + w_1 x + \cdots + w_m x^m \quad (4)$$

中，一共有 $m+1$ 个未知系数，设向量 $w = (w_0, w_1, \dots, w_m)^T$ ， w 即为待求的多项式系数向量。因此我们的目标是求得一个系数向量 w ，使得与原数据的拟合效果最好。在这里我们使用最小二乘法确定最佳的系数向量 w 。

3.2.1 不加惩罚项

定义误差函数

$$E(w) = \frac{1}{2} (Xw - Y)^T (Xw - Y), \quad (5)$$

我们的目标即为求取误差函数的最小值，确定误差函数取最小值时的系数向量 w 。

3.2.2 加入惩罚项

在误差函数上加入惩罚项，当罚项适当时，使得模型复杂度与问题匹配，减小过拟合现象，提高模型的泛化能力。

由此，定义误差函数

$$E(w) = \frac{1}{2} (Xw - Y)^T (Xw - Y) + \frac{\lambda}{2} \|w\|^2. \quad (6)$$

我们的目标即为求取加入惩罚项后误差函数的最小值，确定加入惩罚项之后误差函数取最小值时的系数向量 w 。

3.3 数值解法

3.3.1 不加入惩罚项

对误差函数求导，令导数为0，解得的 w 即为令误差函数取得最小值的目标系数向量。

$$E(w) = \frac{1}{2} (Xw - Y)^T (Xw - Y) \quad (7)$$

$$= \frac{1}{2} (w^T X^T - Y^T) (Xw - Y) \quad (8)$$

$$= \frac{1}{2} (w^T X^T X w - w^T X^T Y - Y^T X w + Y^T Y) \quad (9)$$

$$= \frac{1}{2} (w^T X^T X w - 2w^T X^T Y + Y^T Y) \quad (10)$$

对上式的 w 求偏导，即 $\frac{\partial E}{\partial w} = X^T X w - X^T Y$ ，令偏导数为0，解得

$$w = (X^T X)^{-1} X^T Y \quad (11)$$

3.3.2 加入惩罚项

对加入罚项后的误差函数求导，令导数为 0，解得的 w 即为加入罚项后的误差函数取得最小值的目标系数向量。

$$\tilde{E}(w) = \frac{1}{2} (Xw - Y)^T (Xw - Y) + \frac{\lambda}{2} \|w\|^2 \quad (12)$$

$$= \frac{1}{2} (w^T X^T X w - w^T X^T Y - Y^T X w + Y^T Y) + \frac{\lambda}{2} (w^T w) \quad (13)$$

对上式的 w 求偏导，即 $\frac{\partial \tilde{E}}{\partial w} = X^T X w - X^T Y + \lambda w$ ，令偏导数为 0，解得

$$w = (X^T X + \lambda)^{-1} X^T Y \quad (14)$$

3.4 梯度下降法

梯度下降法是通过迭代求目标函数最小值的一种方法，从数学上的角度来看，梯度的方向是函数增长速度最快的方向，那么梯度的反方向就是函数减少最快的方向。由于上述的两种误差函数为二次型，因此应用梯度下降求得的局部最优解即为全局最优解。

对式 (6) 中的 w 应用梯度下降，在满足迭代误差保持在一定范围后，求得的 w 即为多项式的系数向量。算法伪代码如 1 所示。

Algorithm 1 Gradient Descent

```

Input: train_X,train_Y,learning_rate,deviation,poly_degree,lambda
Output: w_result
    Normalization train_X & train_Y
    Initialize w_result
    Calculate Gradient
    Initialize pre_loss ← 0
    Calculate loss function's result
    while pre_loss - loss > deviation do
        Update w_result with new gradient
        Update pre_loss & loss
        Calculate new gradient
        if Divergence in Iteration Process then
            Make the learning rate half
        end if
    end while
    return w_result

```

3.5 共轭梯度法

虽然梯度下降法的每一步都是朝着局部最优的方向前进的，但是在不同的迭代轮数中会选择非常近似的方向，说明这个方向的误差并没通过一次更新方向和步长更新完，在这个方向上还存在误差，因此参数更新的轨迹是锯齿状。共轭梯度法的思想是，选择一个优化方向后，本次选择的步长能够将这个方向的误差更新完，在以后的优化更新过程中不再需要再在这个方向进行更新。由于每次将一个方向优化到了极小，后面的

优化过程将不再影响之前优化方向上的极小值，所以理论上对 N 维问题求极小只用对 N 个方向都求出极小即可。

对式 (13) 求导，令导数为 0，得到

$$(X^T X + \lambda) w = X^T Y. \quad (15)$$

记 $Q = X^T X + \lambda$ ，则该问题转化为求解 $\min_{w \in R^n} \frac{1}{2} w^T Q w - Y^T X w$ ，采用共轭梯度下降法，理论上最多迭代 n 次即可得到结果。

算法伪代码如 2 所示。

Algorithm 2 Gradient Descent

Input: train_X, train_Y, learning_rate, poly_degree, lam
Output: w_result

```

Normalization train_X & train_Y
Calculate Matrix Q
Initialize w_result
Calculate Gradient
Initialize pre_gradient ← -gradient
for i in range poly_degree do
    Calculate the step of gradient descent
    Update new w_result
    Calculate Residual Vector
    Update the Direction Vector
end for
return w_result

```

四、实验步骤

4.1 生成训练数据与测试数据并测试

利用 Python 自带的 numpy 库均匀生成 [0, 1] 之间的训练数据与测试数据。并针对不同的阶数分别采取六种拟合方法进行拟合（数值解法不加入惩罚项、数值解法加入惩罚项、梯度下降法不加入惩罚项、梯度下降法加入惩罚项、共轭梯度法不加入惩罚项、共轭梯度法加入惩罚项），并调用 numpy 库中的拟合函数 polyfit 进行比较。最后生成数值解法不加入惩罚项时，在训练数据集和测试数据集上使用训练得到的多项式计算损失函数随多项式阶数的变化情况。

```

from numpy_polyfit import *
from generate_picture import *
from analytical_solution import *
from generate_loss_picture import *
from gradient_descent_solution import *
from conjugate_gradient_solution import *

def generate_train_data(mu, sigma):
    .....
    生成训练样本数据，来自函数y=sin(2 * pi * x)，并加入噪声

```

```
:param mu: 高斯噪声的均值
:param sigma: 高斯噪声的方差
:return: 生成的训练数据集的X向量和Y向量
"""

train_X = np.linspace(0, 1, data_number)
noise = np.random.normal(mu, sigma, data_number)
train_Y = np.sin(2 * np.pi * train_X) + noise
# print(train_X)
# print(np.sin(2 * np.pi * train_X))
# print(train_Y)
return train_X, train_Y

def generate_test_data():
"""
生成测试样本数据，来自函数y=sin(2 * pi * x)
:return: 生成的测试数据集的X向量和Y向量
"""

test_X = np.linspace(0, 1, 4 * data_number)
test_Y = np.sin(2 * np.pi * test_X)
return test_X, test_Y

def main():
"""
主函数
:return: 无
"""

train_X, train_Y = generate_train_data(mu=0, sigma=0.1)
test_X, test_Y = generate_test_data()
loss_train_list = []
loss_test_list = []
x_list = []

for i in range(11):
    poly_degree = i
    test_X_normalize, test_Y_normalize = normalization(test_X, test_Y,
                                                       poly_degree)
    train_X_normalize, train_Y_normalize = normalization(train_X, train_Y,
                                                       poly_degree)

    title = 'poly degree=' + str(poly_degree) + ', data number=' +
           str(data_number) + '.'

    """
    numpy库的polyfit方法
    """

    numpy_result = np_polyfit(train_X, train_Y, poly_degree)
    generate_picture(train_X, train_Y, numpy_result, 'numpy.polyfit', title)
```

```
"""
数值解法不加惩罚项
"""

analytical_result_without_penalty, result =
    analytical_solution_without_penalty(train_X, train_Y, poly_degree)
loss_train = calculate_loss_function_result(result, train_X_normalize,
                                              train_Y_normalize)
loss_test = calculate_loss_function_result(result, test_X_normalize,
                                             test_Y_normalize)
# print(loss_train[0])
loss_train_list.append(loss_train[0])
loss_test_list.append(loss_test[0])
x_list.append(i)
# print(loss_train)
# print(loss_test)
# print('-----')

generate_picture(train_X, train_Y, analytical_result_without_penalty,
                  'analytical without penalty', title)

"""
数值解法加惩罚项
"""

coefficient_of_penalty = 0.0005
analytical_result_with_penalty =
    analytical_solution_with_penalty(train_X, train_Y,
                                      coefficient_of_penalty,
                                      poly_degree)
generate_picture(train_X, train_Y, analytical_result_with_penalty,
                  'analytical with penalty', title)

learning_rate = 0.1
deviation = 1e-5

"""
梯度下降法，加惩罚项
"""

gradient_descent_result_with_penalty = gradient_descent(train_X, train_Y,
                                                          learning_rate, deviation,
                                                          poly_degree, lam=0.01)
generate_picture(train_X, train_Y, gradient_descent_result_with_penalty,
                  'gradient descent without penalty', title)

"""
梯度下降法，不加惩罚项
"""

gradient_descent_result_without_penalty = gradient_descent(train_X,
                                                             train_Y, learning_rate, deviation,
                                                             poly_degree, lam=0)
```

```

    generate_picture(train_X, train_Y,
                      gradient_descent_result_without_penalty, 'gradient descent with
                      penalty',
                      title)

"""
共轭梯度法，加惩罚项
"""

conjugate_gradient_result_with_penalty =
    conjugate_gradient_solution(train_X, train_Y, poly_degree, lam=0.001
                                )
generate_picture(train_X, train_Y,
                  conjugate_gradient_result_with_penalty, 'conjugate gradient with
                  penalty',
                  title)
"""

共轭梯度法，不加惩罚项
"""

conjugate_gradient_result_without_penalty =
    conjugate_gradient_solution(train_X, train_Y, poly_degree, lam=0
                                )
generate_picture(train_X, train_Y,
                  conjugate_gradient_result_without_penalty,
                  'conjugate gradient without penalty', title)

x_array = np.array(x_list)
loss_test_array = np.array(loss_test_list)
loss_train_array = np.array(loss_train_list)
generate_loss_picture(x_array, loss_train_array, loss_test_array)

if __name__ == '__main__':
    main()

```

4.2 数值解法不加入惩罚项与加入惩罚项

实现 3.3 中的两种不同情况下求解系数向量的数值解法。

```

from normalization_X_Y import *
import numpy as np

def analytical_solution_without_penalty(train_X, train_Y, poly_degree):
    """
    不加惩罚项的数值解法
    :param poly_degree: 多项式次数
    :param train_X: 训练集的X向量
    :param train_Y: 训练集的Y向量

```

```

:rtype: 解向量
"""

X, Y = normalization(train_X, train_Y, poly_degree)
matrix = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(Y)
w_result = np.poly1d(matrix[::-1].reshape(poly_degree + 1))
return w_result, matrix

def analytical_solution_with_penalty(train_X, train_Y, lam, poly_degree):
    """
    加惩罚项的数值解法
    :param poly_degree: 多项式次数
    :param train_X: 训练集的X矩阵
    :param train_Y: 训练集的Y向量
    :param lam: 惩罚项系数
    :return: 解向量
    """

    X, Y = normalization(train_X, train_Y, poly_degree)
    matrix = np.linalg.inv(X.T.dot(X) + lam *
                           np.eye(X.shape[1])).dot(X.T).dot(Y)
    w_result = np.poly1d(matrix[::-1].reshape(poly_degree + 1))
    # print("w result analytical")
    # print(w_result)
    return w_result

```

4.3 梯度下降法

实现 3.4 中的求解系数向量的梯度下降算法。

```

from calculate_loss_function_result import *
from normalization_X_Y import *
from global_number import *

def gradient_descent(train_X, train_Y, learning_rate, deviation, poly_degree,
                     lam):
    """
    梯度下降法求多项式函数的系数矩阵(向量)
    :param lam: 惩罚项系数
    :param poly_degree: 多项式拟合次数
    :param train_X: 训练集的X向量
    :param train_Y: 训练集的Y向量
    :param learning_rate: 梯度下降法的学习率
    :param deviation: 梯度下降法最终允许的误差范围
    :return: 多项式函数的系数矩阵, 迭代次数的list, 每次迭代对应的误差
    """

    X, Y = normalization(train_X, train_Y, poly_degree)
    w = 0.1 * np.ones((poly_degree + 1, 1))

```

```

number = 0
grad = (1.0 / data_number) * (X.T.dot(X).dot(w) - X.T.dot(Y) + lam * w)
loss0 = 0
loss1 = calculate_loss_function_result(w, X, Y)
while abs(loss1 - loss0) > deviation:
    number += 1
    w -= learning_rate * grad
    loss0 = loss1
    loss1 = calculate_loss_function_result(w, X, Y)
    if np.all(loss1 - loss0 > 0):
        learning_rate *= 0.5
    grad = (1.0 / data_number) * (X.T.dot(X).dot(w) - X.T.dot(Y) + lam * w)
w_result = np.poly1d(w[::-1].reshape(poly_degree + 1))
return w_result

```

4.4 共轭梯度法

实现 3.5 中的求解系数向量的共轭梯度算法。

```

from normalization_X_Y import *
import numpy as np

def conjugate_gradient_solution(train_X, train_Y, poly_degree, lam):
    """
    共轭梯度法求损失函数最小值对应的系数向量
    :param train_X: 训练集的X矩阵
    :param train_Y: 训练集的Y向量
    :param poly_degree: 拟合多项式次数
    :param lam: 惩罚项系数
    :return: 系数向量
    """

    X, Y = normalization(train_X, train_Y, poly_degree)
    # Q = (1 / data_number) * (X.T * X + lam * np.mat(np.eye(X.shape[1])))
    Q = np.dot(X.T, X) + lam * np.eye(X.shape[1])
    w = np.zeros((poly_degree + 1, 1))
    grad = np.dot(X.T, X).dot(w) - np.dot(X.T, Y) + lam * w
    r = -grad
    pre = r
    """

    Q: 矩阵X的转置与矩阵X的内积加罚项，用于计算方向向量
    grad: 梯度
    w: 初始化向量
    """

    for i in range(poly_degree + 1):
        """
        A: 下降步长
        """

```

```
pre: 方向向量
r: 残差向量
"""
A = (r.T.dot(r)) / (pre.T.dot(Q).dot(pre))
r_pre = r
w = w + A * pre
r = r - (A * Q).dot(pre)
beta = (r.T.dot(r)) / (r_pre.T.dot(r_pre))
pre = r + beta * pre
w_result = np.poly1d(w[::-1].reshape(poly_degree + 1))
return w_result
```

五、 实验结果

5.1 数据量为 11 时不同阶数不同方法的拟合效果

5.1.1 数值解法

不加惩罚项的数值解法拟合结果如图 1所示。

加惩罚项的数值解法拟合结果如图 2所示。

5.1.2 梯度下降法

不加惩罚项的梯度下降法拟合结果如图 3所示。

加惩罚项的梯度下降法拟合结果如图 4所示。

5.1.3 共轭梯度法

不加惩罚项的共轭梯度法拟合结果如图 5所示。

加惩罚项的共轭梯度法拟合结果如图 6所示。

5.2 数据量为 100 时不同阶数不同方法的拟合效果

5.2.1 数值解法

不加惩罚项的数值解法拟合结果如图 7所示。

加惩罚项的数值解法拟合结果如图 8所示。

5.2.2 梯度下降法

不加惩罚项的梯度下降法拟合结果如图 9所示。

加惩罚项的梯度下降法拟合结果如图 10所示。

5.2.3 共轭梯度法

不加惩罚项的共轭梯度法拟合结果如图 11所示。

加惩罚项的共轭梯度法拟合结果如图 12所示。

5.3 训练集与测试集的损失函数随阶数的变化

当训练集的数量为 11 时，训练集与测试集的损失函数随阶数的变化如图 13 所示。

当训练集的数量为 100 时，训练集与测试集的损失函数随阶数的变化如图 14 所示。

六、 实验结果分析

根据实验结果，使用多项式对正弦函数 $f(x) = \sin(2\pi x)$ 进行拟合时，多项式的次数越高，其拟合能力越强，在数据量为 11 且不加入惩罚项的情况下，次数较高的多项式出现了过拟合的现象。形成过拟合的主要原因是样本数量过少，而多项式次数过高会导致模型学习能力过强，模型拟合结果过分依赖数据量很少的数据集，而太少的样本存在一定偶然性，这种强拟合能力将并不具有普遍性的特征训练进了模型，从而无法拟合出正弦曲线。在数据集为 100 且不加入惩罚项的情况下，次数相对于 100 来说很低，所以并未出现过拟合的现象。但是在数据量为 11 且加入惩罚项的情况下，次数高的多项式也能减小过拟合现象。

因此，为了减小过拟合现象，可以增加惩罚项或增加数据量。根据实验结果，在目标函数中加入参数的惩罚项后，过拟合现象得到明显改善。这是由于参数增多时，往往具有较大的绝对值，加入正则项可以有效地降低参数的绝对值，从而使模型复杂度与问题匹配。但是当样本足够多时，惩罚项相对于数据就影响不大了。当惩罚项过小时，过拟合现象并没有明显的改善，惩罚项过大时，会出现欠拟合现象。增加数据量可以避免因为数据量较少而产生的随机误差，避免随机误差被当做数据特征被学习进模型中。

图片 13 与图片 14 也表明，在训练集的数据量较少的情况下，模型出现了过拟合的现象，尽管在训练集的损失函数达到了很小，但是测试集的损失函数也变得很大。当训练数据集的数量是 100 时，多项式阶数为 10 时也没有出现过拟合现象，按照理论预测，在多项式阶数与训练集数据规模相一致时会出现过拟合现象。

在实验的过程中发现随着次数的增加，梯度下降法必须要减小学习率，否则很容易无法收敛。因此在梯度下降法运行的过程中需要动态的调节学习率，如果后一次迭代的 loss 比前一次的 loss 大，将学习率减半。若两次迭代的 loss 差值小于一个指定的很小的数，可以认为此时非常接近极值点，停止迭代。

由图像生成的时间可知，共轭梯度法经过有限次的迭代即可达到极小值，相对于梯度下降法，共轭梯度法迭代速度更快。

七、 实验结论

- (1) 数值解法增加惩罚项可以有效地减小过拟合现象；
- (2) 增加训练数据集的规模可以有效地减小过拟合现象；
- (3) 共轭梯度法相对于梯度下降法迭代速度更快，结果更准确；
- (4) 在梯度下降法中，如果学习率过小，迭代次数就会增加，迭代时间变长；学习率过大，可能无法收敛，或者得到错误结果。

参考文献

[1] 李航，《统计学习方法》（第三版）.

[2] 周志华，《机器学习》.

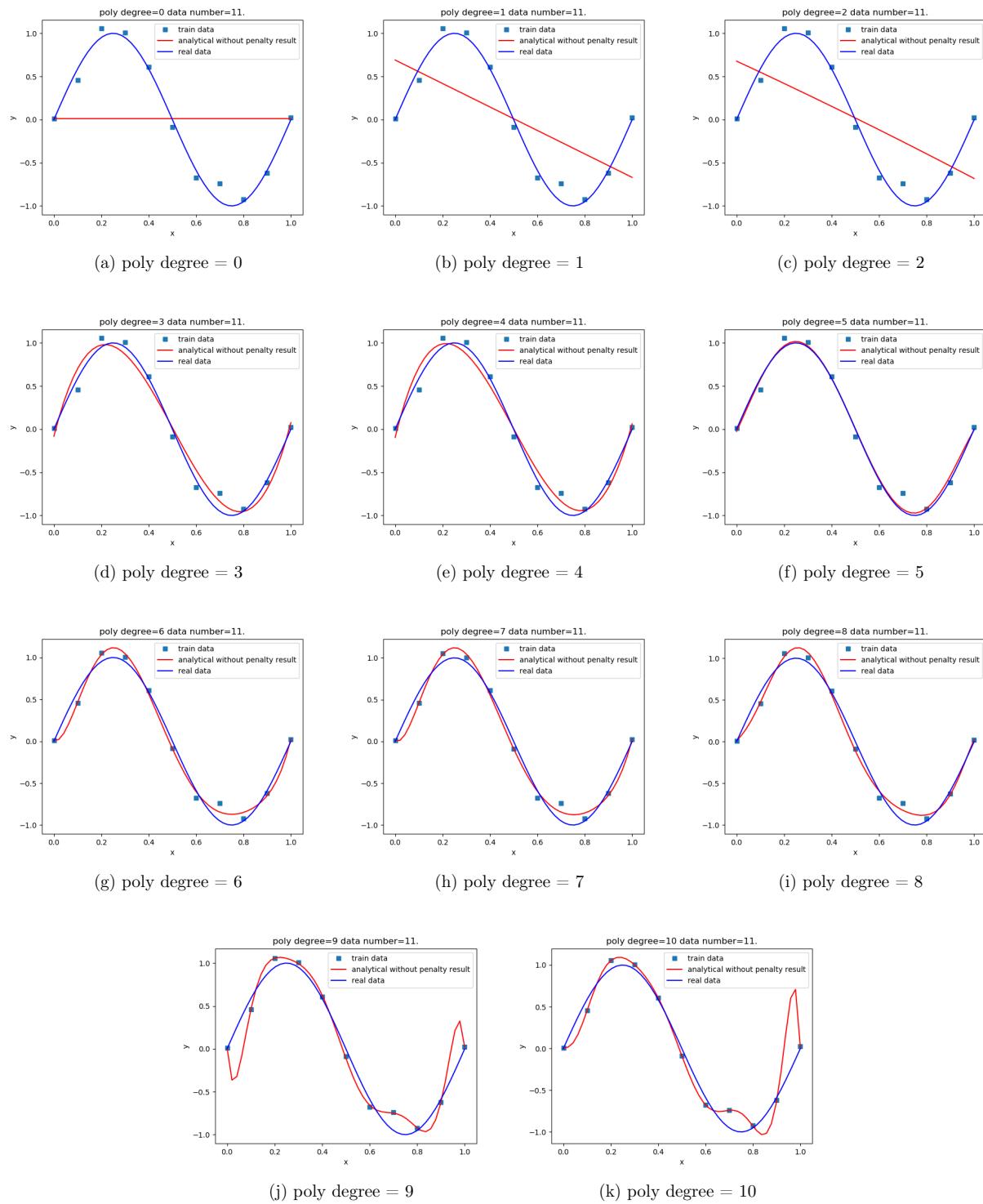


Figure 1: 数据量为 11 时不同阶数不加惩罚项数值解法的拟合效果

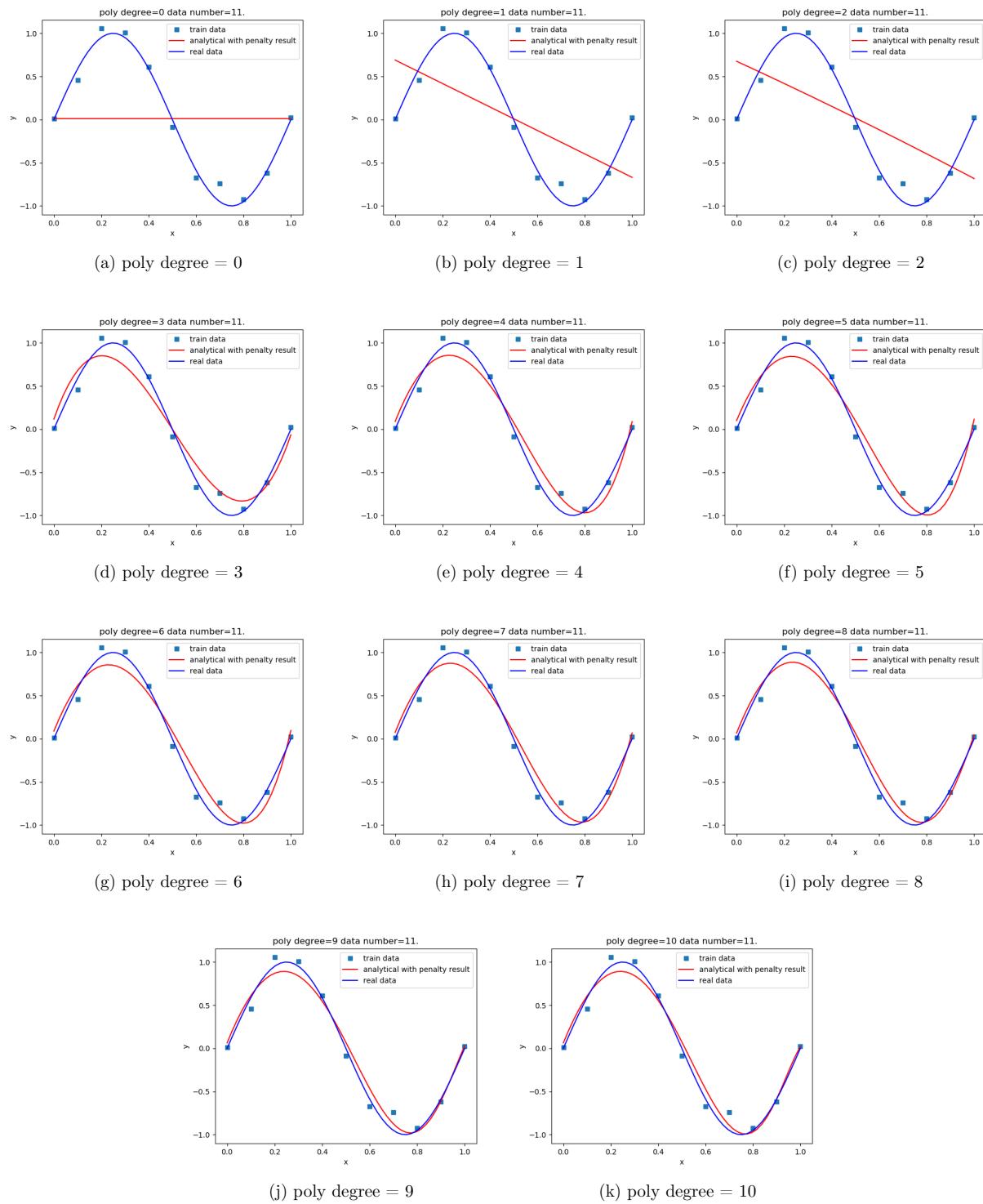


Figure 2: 数据量为 11 时不同阶数加惩罚项数值解法的拟合效果

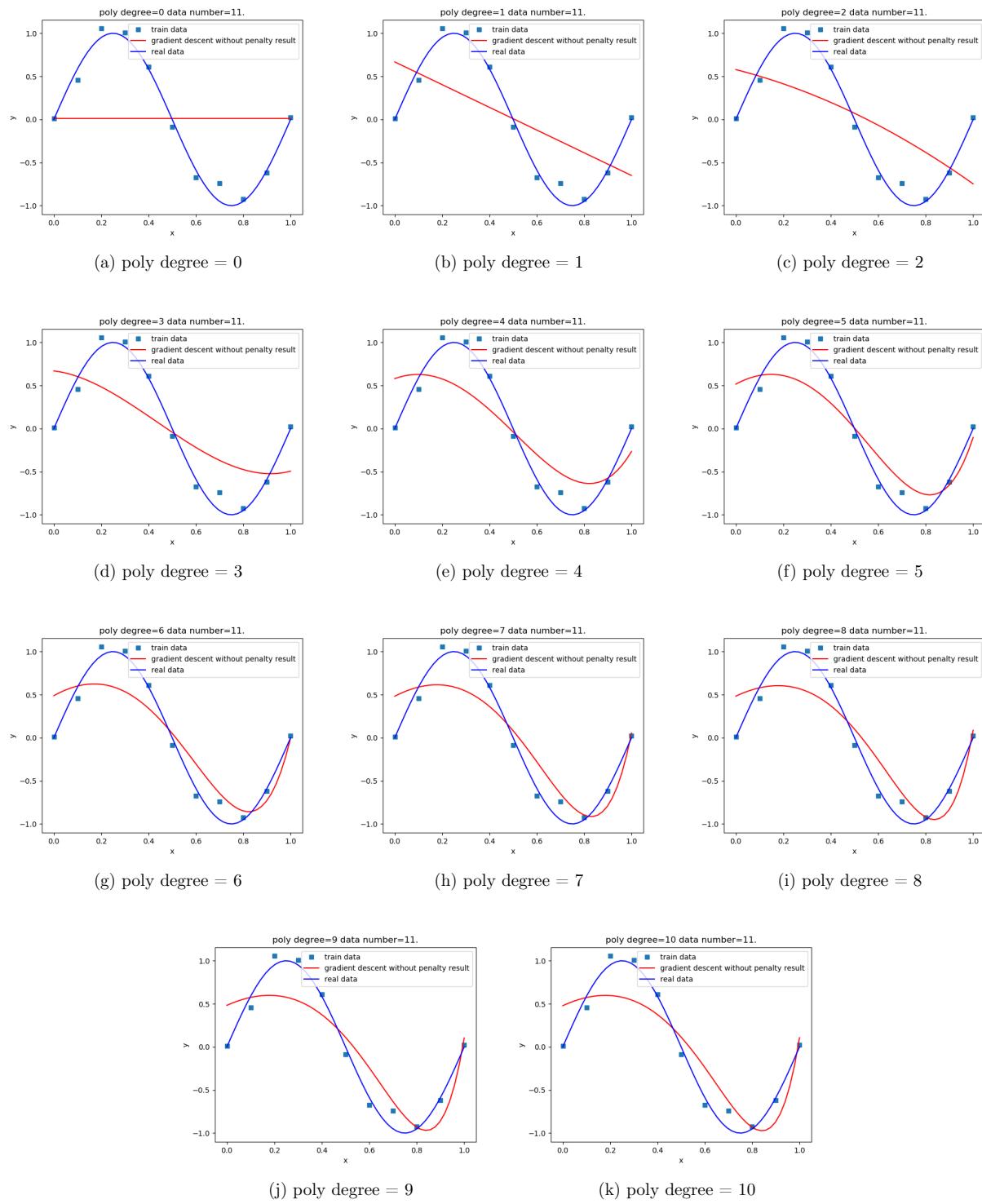


Figure 3: 数据量为 11 时不同阶数不加惩罚项梯度下降法的拟合效果

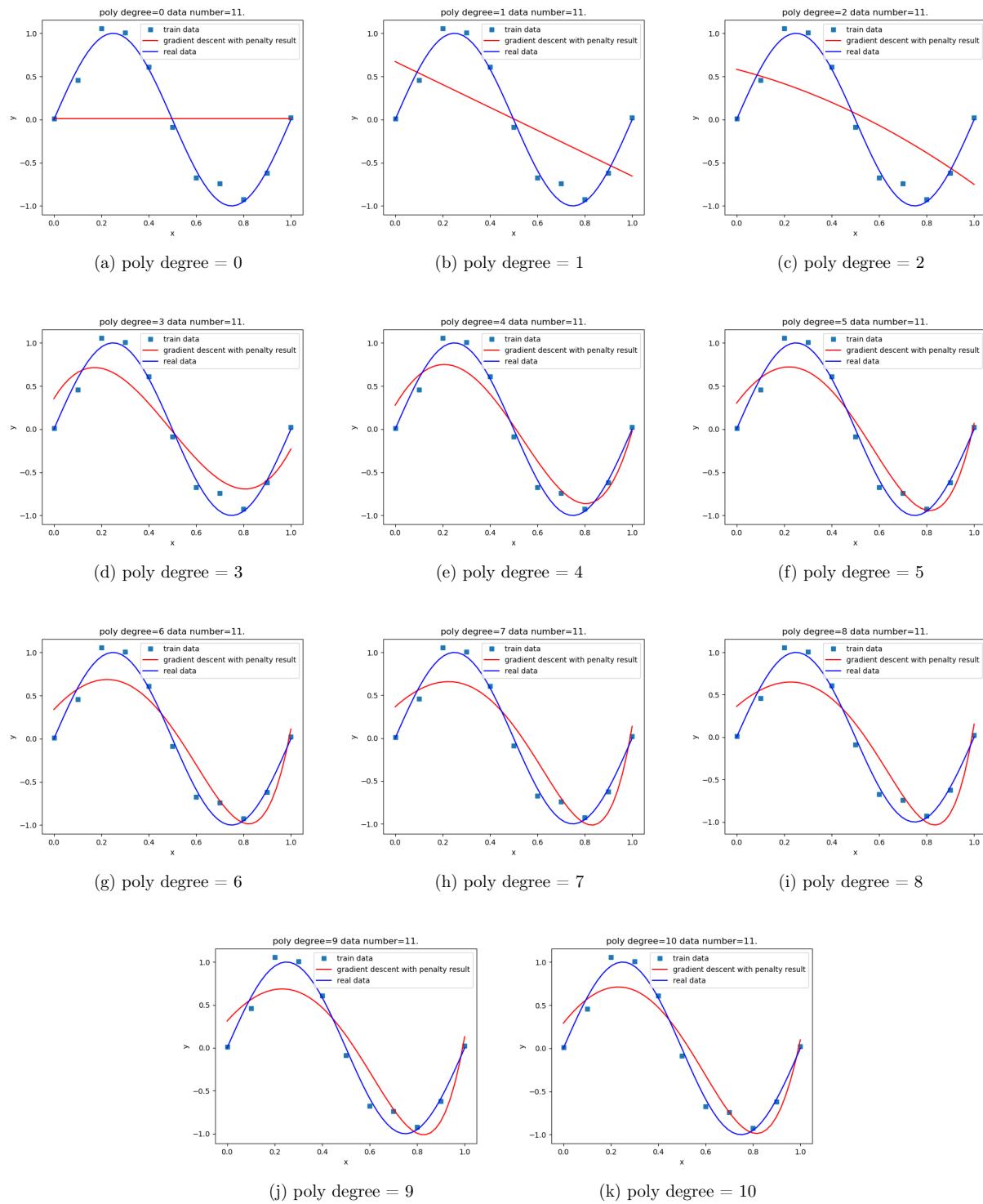


Figure 4: 数据量为 11 时不同阶数加惩罚项梯度下降法的拟合效果

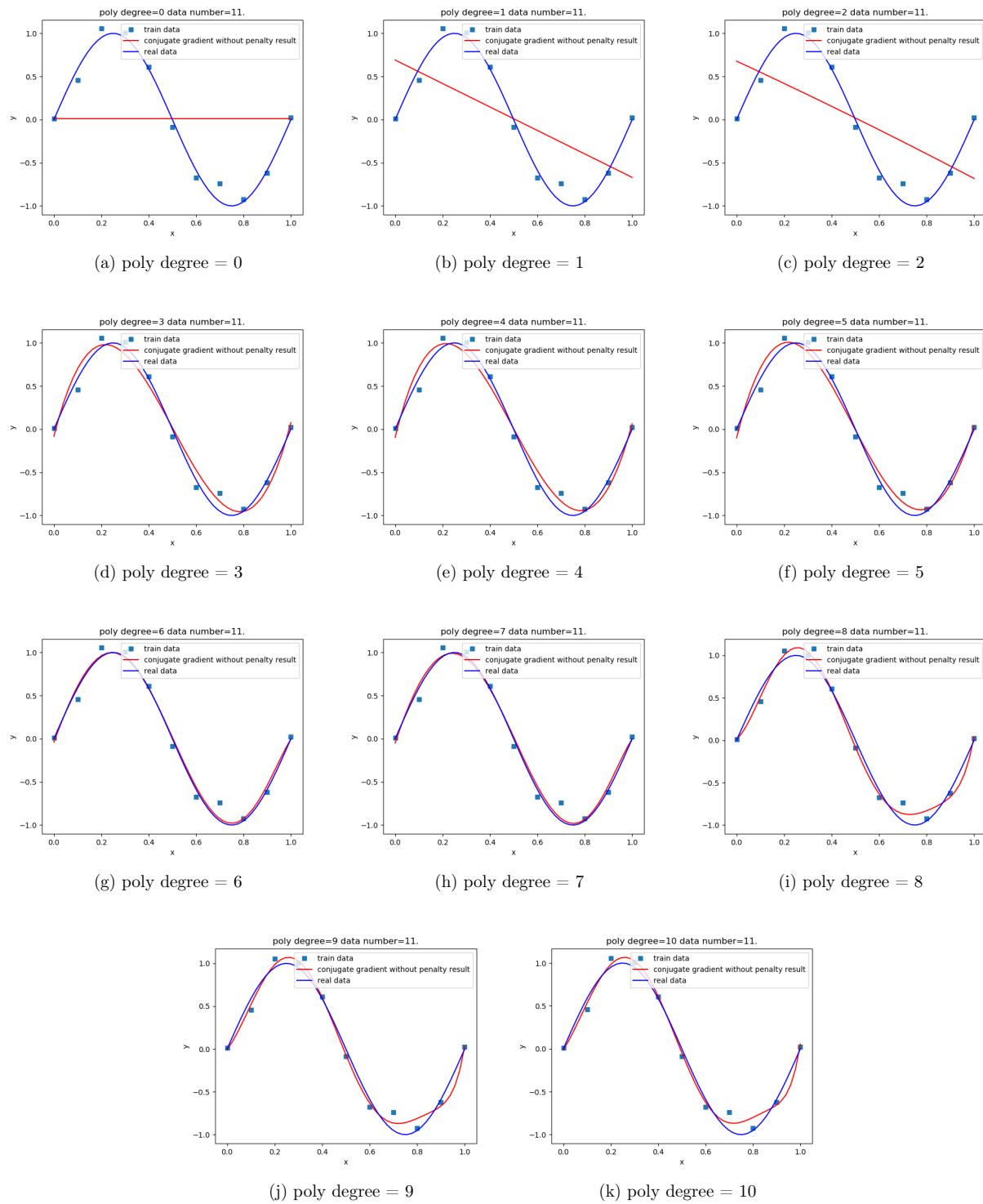


Figure 5: 数据量为 11 时不同阶数不加惩罚项共轭梯度法的拟合效果

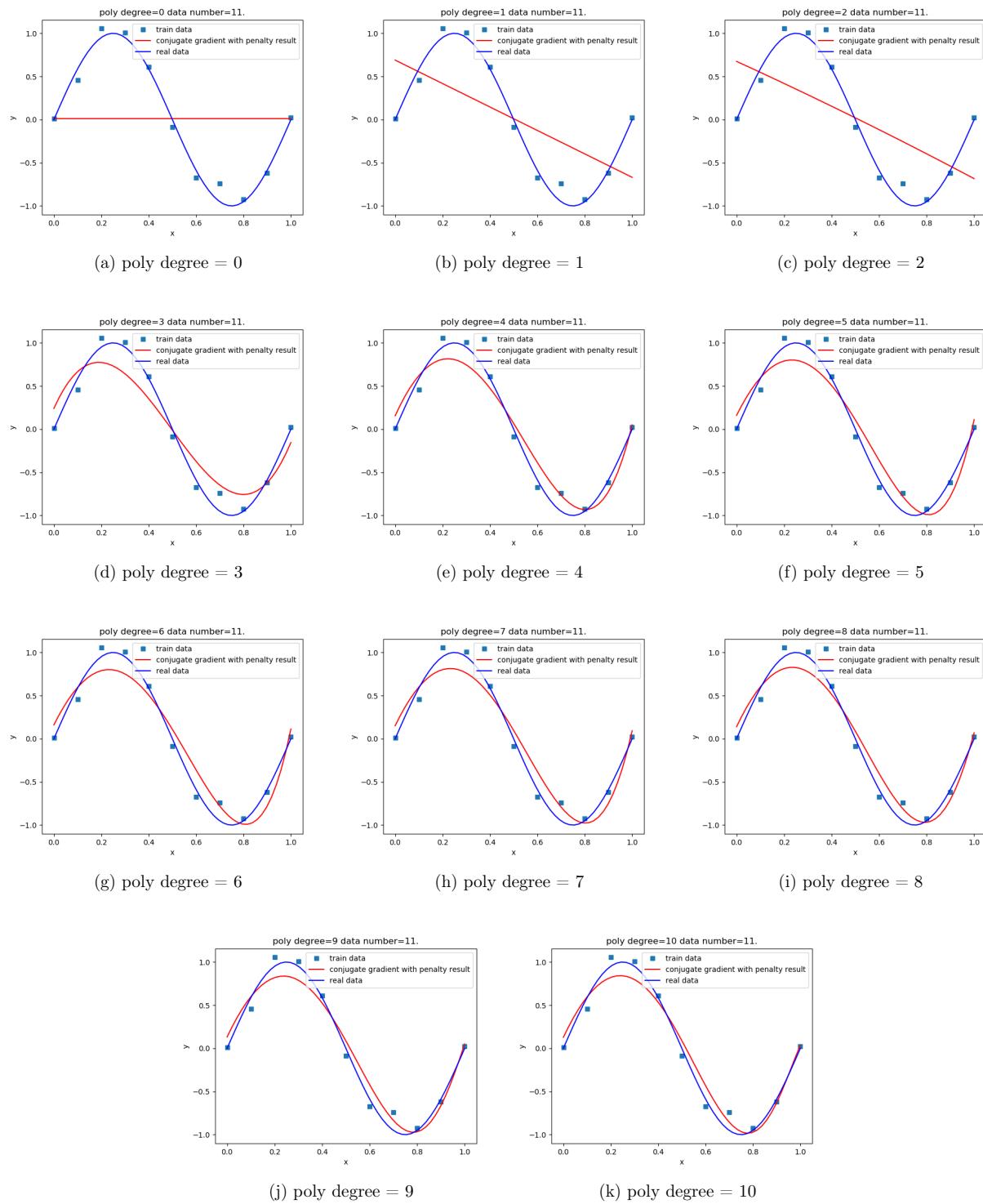


Figure 6: 数据量为 11 时不同阶数加惩罚项共轭梯度法的拟合效果

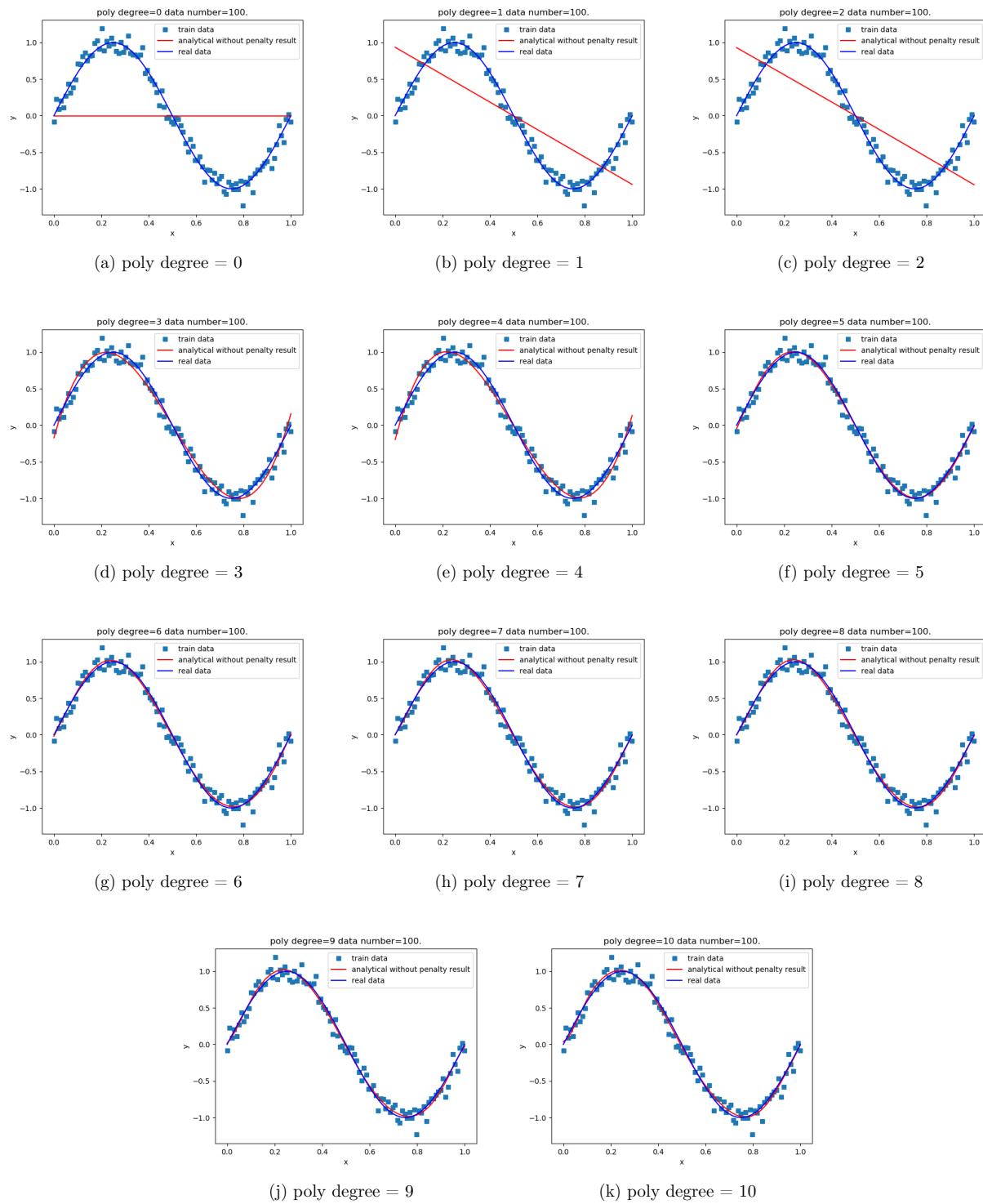


Figure 7: 数据量为 11 时不同阶数不加惩罚项数值解法的拟合效果

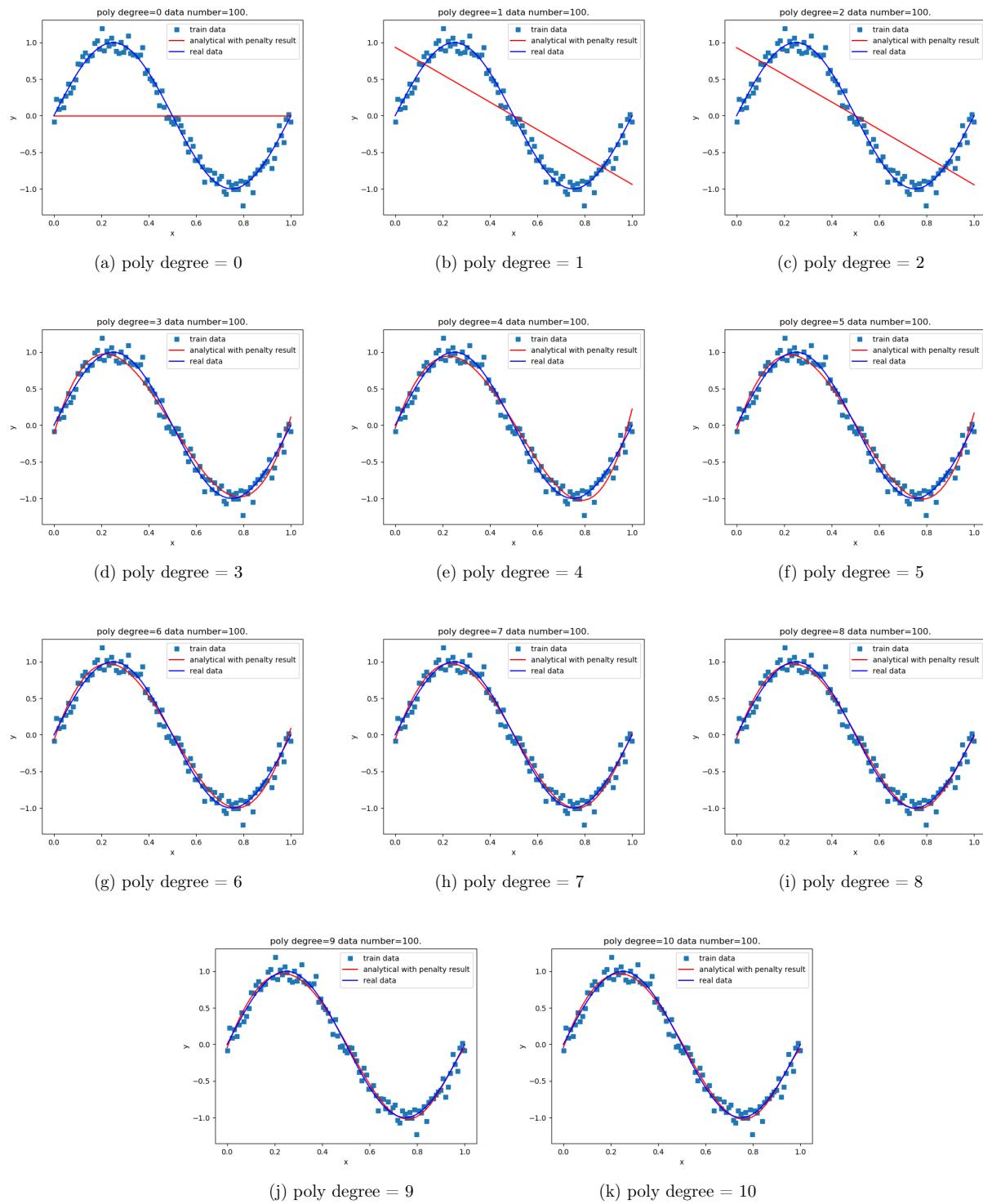


Figure 8: 数据量为 100 时不同阶数加惩罚项数值解法的拟合效果

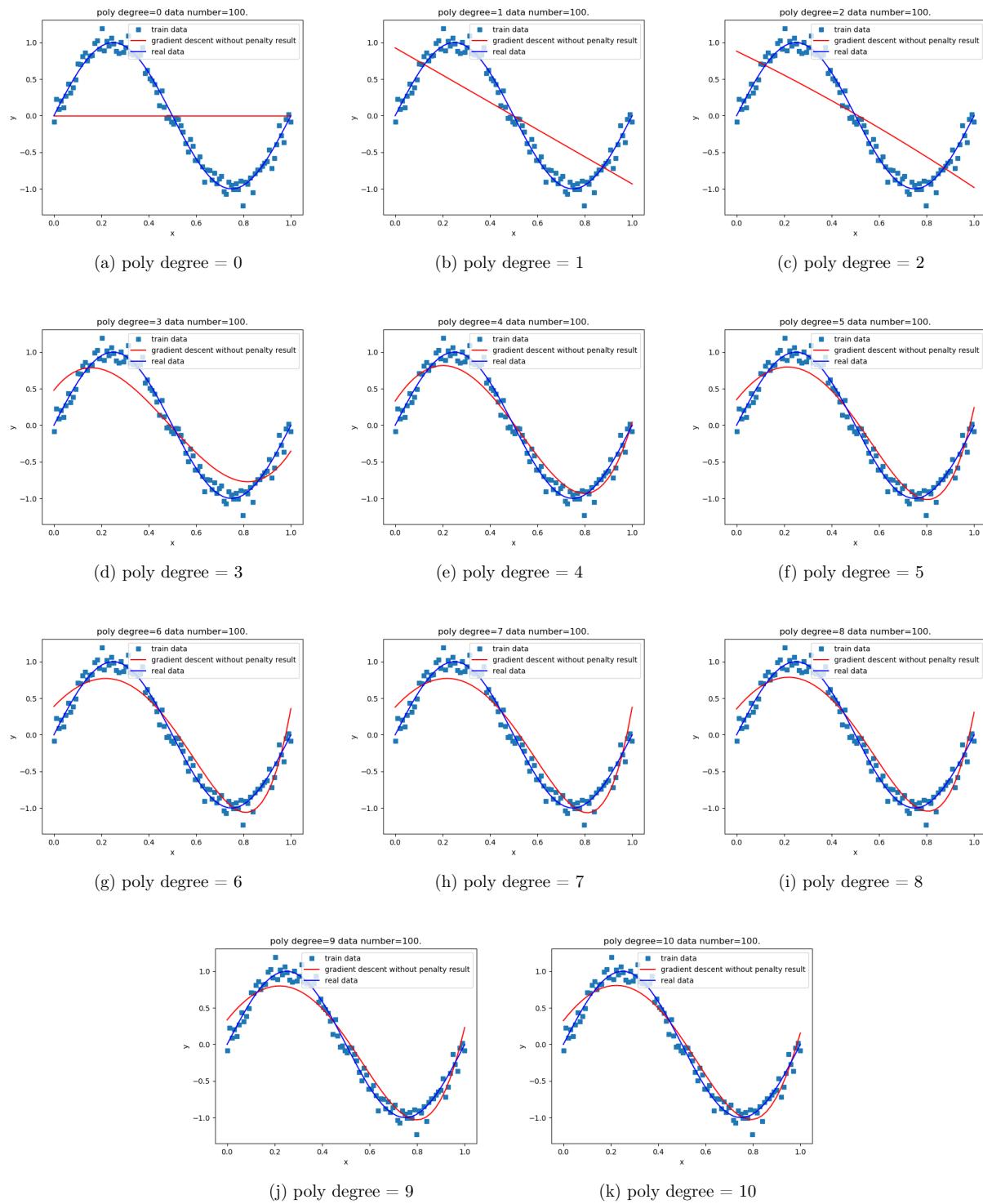


Figure 9: 数据量为 100 时不同阶数不加惩罚项梯度下降法的拟合效果

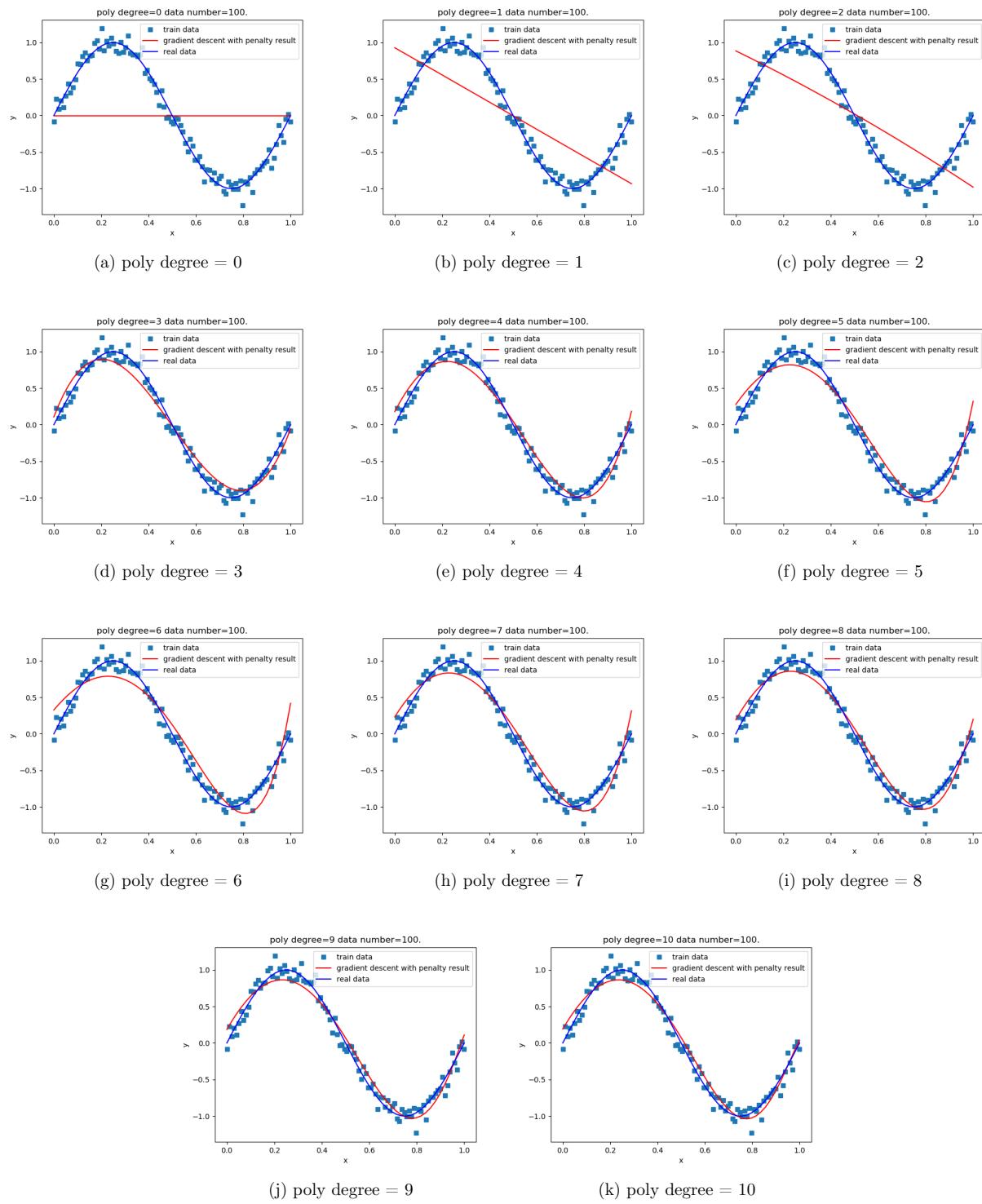


Figure 10: 数据量为 100 时不同阶数加惩罚项梯度下降法的拟合效果

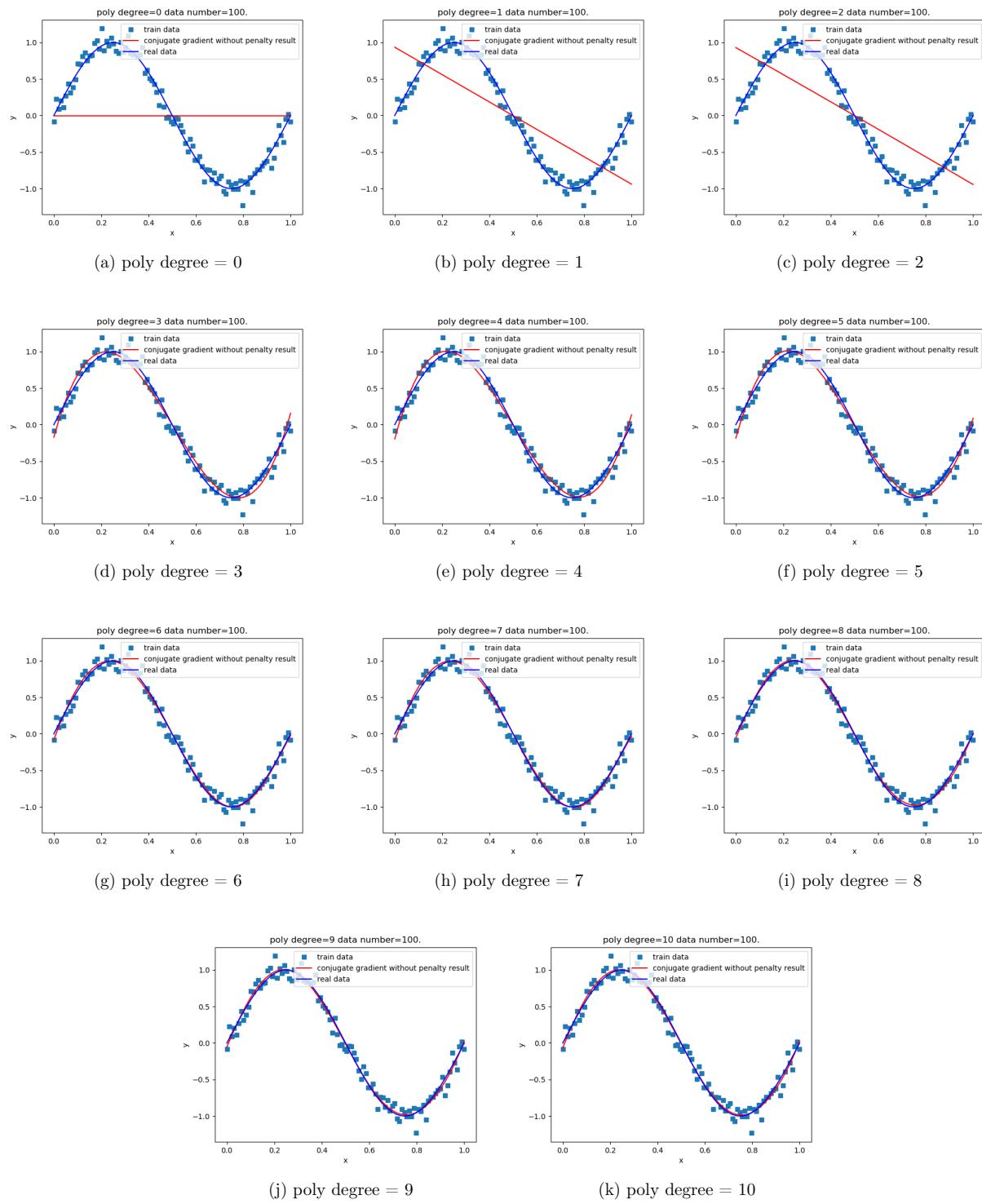


Figure 11: 数据量为 100 时不同阶数不加惩罚项共轭梯度法的拟合效果

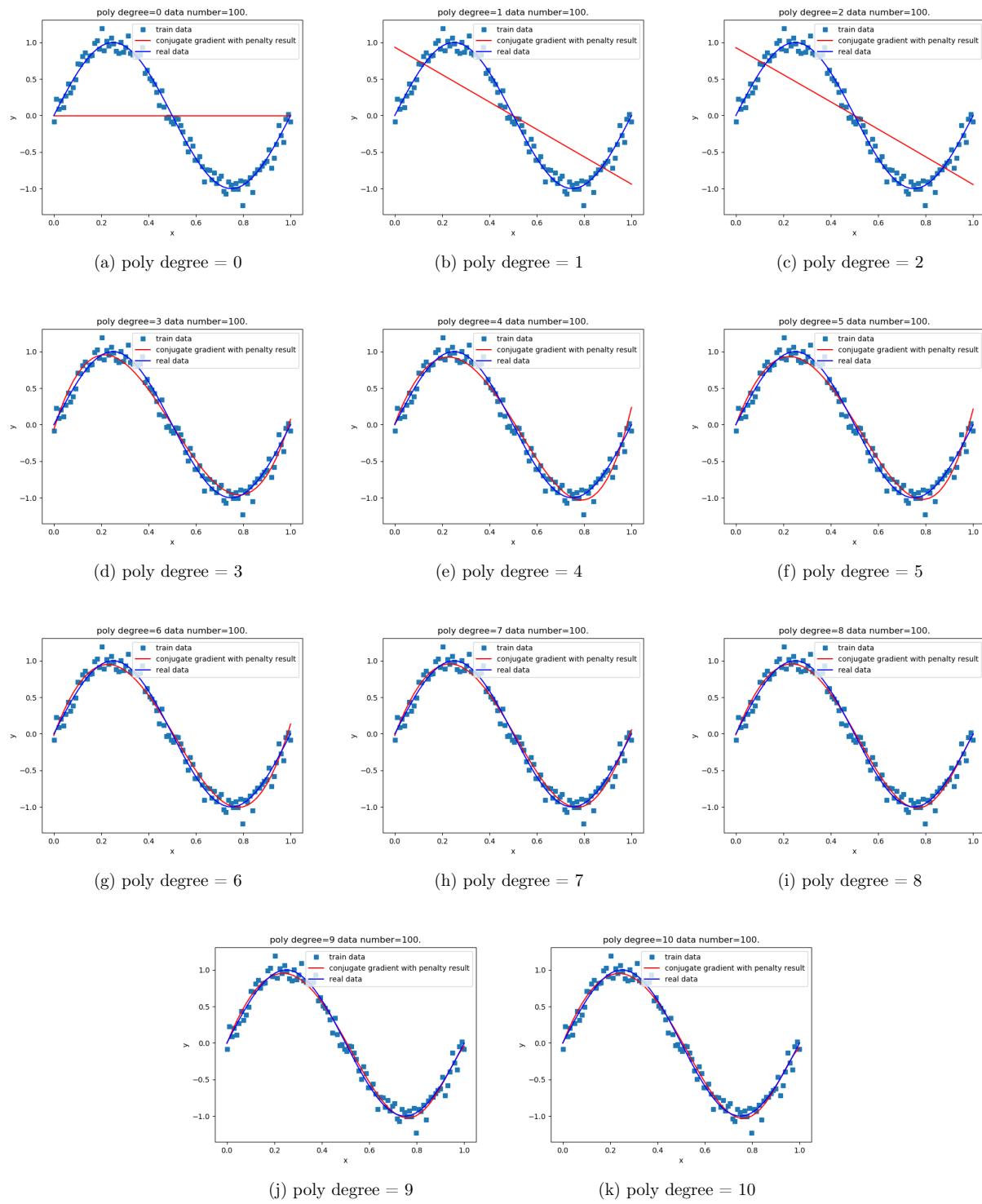


Figure 12: 数据量为 100 时不同阶数加惩罚项共轭梯度法的拟合效果

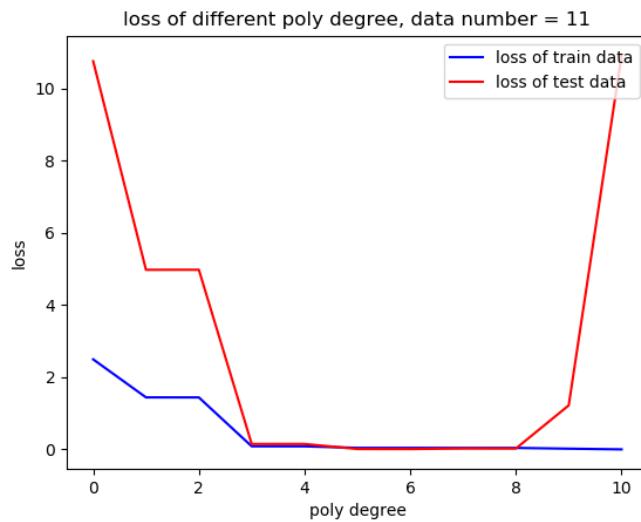


Figure 13: 训练集数量为 11 时，训练集与测试集的损失函数随阶数的变化

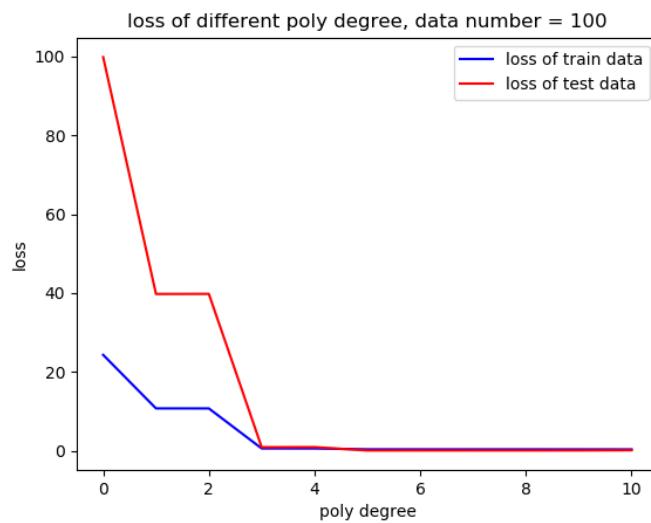


Figure 14: 训练集数量为 100 时，训练集与测试集的损失函数随阶数的变化