哈尔滨工业大学计算机科学与技术学院

# 实验报告

课程名称： 机器学习

课程类型： 选修

实验题目： 多项式拟合正弦曲线

# 一、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)

# 二、实验要求及实验环境

## 2.1 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。

## 2.2 实验环境

硬件环境：x64CPU
软件环境：pycharm
库版本号：anaconda python==3.6, numpy==1.17.0

# 三、设计思想（本程序中的用到的主要算法及数据结构）

## 3.1 生成数据并添加噪声

待拟合曲线为 $y=\sin(2\pi x)$，在该曲线上取若干点添加随机噪声。随机噪声由均值为 $\mu$，方差为 $\sigma^2$ 的高斯分布产生。

```
def generate_data(N, mu=0, sigma=0.1, visualize=False):
    # generate true values
    x = np.linspace(0, 1, N, endpoint=True)
    y = np.sin(2 * np.pi * x)

    # add gaussian noise
    for i in range(x.size):
        y[i] += gauss(mu, sigma)

    # visualize
    if visualize:
        plt.scatter(x, y, color='pink')
    return x, y
```

图 1 添加噪声部分代码

例如，生成 10 个数据的示意图如下：

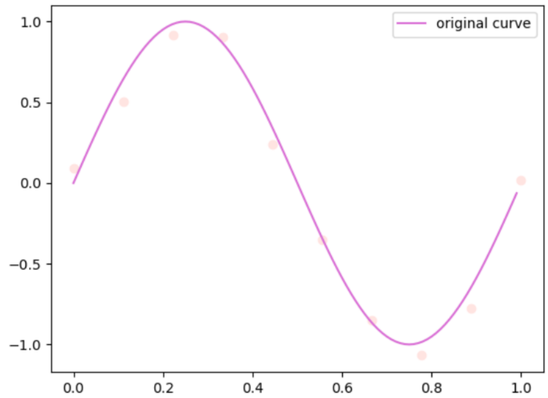


图 2 生成数据 N=10 示意图

## 3.2 多项式拟合曲线原理

本实验中，使用如下形式的多项式函数来拟合数据：

$$y(x,\mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

其中多项式次数为$M$，向量$\mathbf{w} = (w_0, w_1, ..., w_M)$为期望得到的参数。

根据泰勒公式可知，任意一个函数$f(x)$都可以被展开成如下形式：

$$f(x) = \frac{f(x_0)}{0!} + \frac{f'(x_0)}{1!}(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + R_n(x)$$

展开后得到泰勒多项式，其系数构成期望学习的向量$\mathbf{w}$，误差为泰勒余项，常用的包括皮亚诺余项和拉格朗日余项两种形式。因此，用多项式函数拟合任意曲线是可行的。由于该多项式是关于向量$\mathbf{w}$的线性函数，因此被称为线性模型。为使拟合效果尽可能好，需要引入误差函数来对拟合误差进行衡量和评估。

本实验中采用平方误差。对于数据点$x_n$，目标值为$t_n$，预测值为$y(x_n,\mathbf{w})$。损失函数形式如下：

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n,\mathbf{w}) - t_n)^2$$

只需求得使该误差函数取值最小的向量$\mathbf{w}$，即可得到拟合曲线的最优解。

然而，在实际操作中，当多项式阶数过高，模型更加复杂时，参数$\mathbf{w}$中很可能掺杂入训练数据集中的噪声，导致过拟合现象严重。此时，为解决过拟合问题，通常可以引入惩罚项$L(\lambda) = \frac{\lambda}{2}\|\mathbf{w}\|^2$，损失函数变为：

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n,\mathbf{w}) - t_n)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

其中惩罚项的目的是为了抑制向量$\mathbf{w}$中系数的值，可以一定程度上缓解过拟合现象。

## 3.3 解析解求最优解

当$E(\mathbf{w})$取到最小值时，对应的向量$\mathbf{w}$即为最优解。为此，先求$E(\mathbf{w})$对$\mathbf{w}$的偏导$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$。

无惩罚项时：

令$X=\begin{pmatrix} x_1^M & \cdots & x_1 & 1 \\ x_2^M & \cdots & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N^M & \cdots & x_N & 1 \end{pmatrix}, \mathbf{w}=\begin{bmatrix} w_M & \cdots & w_1 & w_0 \end{bmatrix}^T, T=\begin{bmatrix} t_1 & t_2 & \cdots & t_N \end{bmatrix}^T$，则有，

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2}(X\mathbf{w}-T)^T(X\mathbf{w}-T) \\ &= \frac{1}{2}((X\mathbf{w})^T X\mathbf{w} - 2\mathbf{w}^T X^T T + T^T T) \\ &= \frac{1}{2}(\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{w}^T X^T T + T^T T) \end{aligned}$$

求导得到，

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = X^T X\mathbf{w} - X^T T$$

误差函数$E(\mathbf{w})$存在极小值点，且极小值点处导数为 0，只需通过如下方程即可解出$\mathbf{w}$的值：

$$\begin{aligned} X^T X\mathbf{w} - X^T T &= 0 \\ X^T X\mathbf{w} &= X^T T \\ \mathbf{w} &= (X^T X)^{-1} X^T T \end{aligned}$$

同理，在有惩罚项时：

$$E(\mathbf{w}) = \frac{1}{2}(\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{w}^T X^T T + T^T T) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = X^T X\mathbf{w} - X^T T + \lambda\mathbf{w}$$

令导数为 0，则可以解出向量$\mathbf{w}$：

$$\begin{aligned} X^T X\mathbf{w} - X^T T + \lambda\mathbf{w} &= 0 \\ (X^T X + \lambda E)\mathbf{w} &= X^T T \\ \mathbf{w} &= (X^T X + \lambda E)^T X^T T \end{aligned}$$

## 3.4 优化方法求最优解

### 3.4.1 梯度下降算法

梯度下降的原理为，如果实值函数$F(x)$在点$a$处可微且有定义，那么函数在该点处沿着梯度相反的方向$-\nabla F(a)$下降最多。因此，如果$b=a-\gamma\nabla F(a)$，那么$F(b)\geq F(a)$。因而，可以从某个点$x_0$出发，沿着梯度下降的方向不断前进，即使得$x_{n+1}=x_n-\alpha\nabla F(x_n)$，最终就有希望得到$F(x)$的极小值点$x_N$。

一轮梯度下降的代码实现如下：

```
def gradient_descent_per_epoch(params, grads, learning_rate):
    params -= grads * learning_rate
    return params
```

图 3 梯度下降代码实现

本实验中，期望得到$E(\mathbf{w})$的极小值点，因此每轮迭代更新的方程为：

$$\mathbf{w}^{(n+1)}=\mathbf{w}^{(n)}-\alpha\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

每一轮中，依据每个训练数据对向量$\mathbf{w}$进行一次更新，轮数（epoch）可以预先设定。$\alpha$为学习率，属于超参数，需手动调参。

无惩罚项时，

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}=X^TX\mathbf{w}-X^TT$$

$$\mathbf{w}^{(n+1)}=\mathbf{w}^{(n)}-\alpha(X^TX\mathbf{w}-X^TT)$$

有惩罚项时：

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}=X^TX\mathbf{w}-X^TT+\lambda\mathbf{w}$$

$$\mathbf{w}^{(n+1)}=\mathbf{w}^{(n)}-\alpha(X^TX\mathbf{w}-X^TT+\lambda\mathbf{w})$$

### 3.4.2 共轭梯度法

共轭梯度法是用于求解线性方程组的解的一种方法。

对于线性方程组 $AX=b$，若矩阵 $A$ 是实对称正定矩阵，则可以用如下算法求得该线性方程组的唯一解：

$$
\begin{aligned}
&\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
&\mathbf{p}_0 := \mathbf{r}_0 \\
&k := 0 \\
&\text{repeat} \\
&\qquad \alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} \\
&\qquad \mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k\mathbf{p}_k \\
&\qquad \mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k\mathbf{A}\mathbf{p}_k \\
&\qquad \text{if } r_{k+1} \text{ is sufficiently small, then exit loop} \\
&\qquad \beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k} \\
&\qquad \mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k \\
&\qquad k := k+1 \\
&\text{end repeat}
\end{aligned}
$$

图 4 共轭梯度算法

代码实现如下：

```python
def conjugate_gradient(A, b, w):
    r = b - np.dot(A, w)
    p = r
    while np.dot(r, r) > 1e-16:
        a = np.dot(r, r) / np.dot(np.dot(p, A), p)
        w += a * p
        r_pre = r.copy()
        r -= a * np.dot(A, p)
        b = np.dot(r, r) / np.dot(r_pre, r_pre)
        p = r + b * p
    return w
```

图 5 共轭梯度算法代码实现

本实验中，若无惩罚项，待求解方程为 $X^T X\mathbf{w} = X^T T$，将 $\begin{matrix} A = X^T X \\ b = X^T T \end{matrix}$ 代入算法即可求解。若有惩罚项，带求解方程为 $(X^T X + \lambda E)\mathbf{w} = X^T T$，将 $\begin{matrix} A = X^T X + \lambda E \\ b = X^T T \end{matrix}$ 代入算法即可求解。

# 四、实验结果与分析

## 4.1 不同多项式阶数对结果的影响

取 N=10 个数据点，分别在多项式阶数 M=0,1,2,…,9 下进行实验，正则化项系数$\lambda$满足$\ln\lambda=-18$。下图展示 10 种阶数情况下，三种求解方式有无惩罚项的曲线拟合结果。

### 4.1.1 解析解

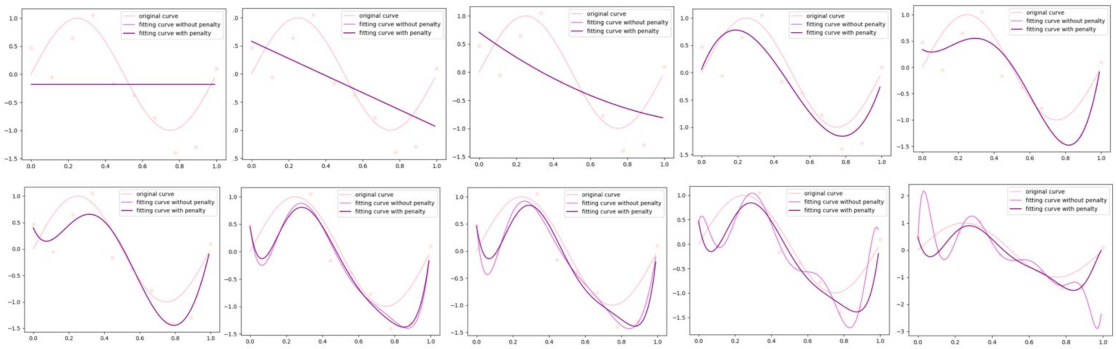


图 6 解析解求解方法在不同多项式阶数下的结果

可以看到，当多项式阶数过小（M=0,1,2）时，拟合效果很差；当多项式阶数过大（M=8,9）时，几乎对于训练数据中每个点都能够正确拟合，然而却不能很好的表示原正弦图象，这便是由于模型过于复杂，过度学习到了噪声信息，出现了过拟合现象。在引入了惩罚项后，这种现象有了明显的缓解，得到的曲线相对来说比较接近原正弦曲线。当 3≤M≤7 时，拟合效果较好，加入惩罚项的影响并不显然。

### 4.1.2 梯度下降

在梯度下降方法中，取迭代次数 epoch=100w，学习率 learning rate=1e-3，分别在多项式阶数 M=0,1,2,…,9 下进行实验，得到结果如下图所示。

在多项式阶数较低（M=0,1,2）时拟合效果仍然很差，当 M=3 时拟合效果较好。然而，当 M≥4 时，与解析解求解方法的情形不同，随着模型逐渐复杂，曲线形态变化不大，似乎并没有出现过拟合现象，得到的曲线也与原正弦曲线较为接近。并且，正则化项的作用并不大。然而，这并不是拟合效果好的表现，而是由于梯度下降过程并未收敛，还未找到全局最优解。如果通过调参，增加迭代次数，调整学习率，加入一些适当的变化（Mini-batch, Adam 等）使得梯度下降能够找到最优解，仍应该是过拟合的状态。

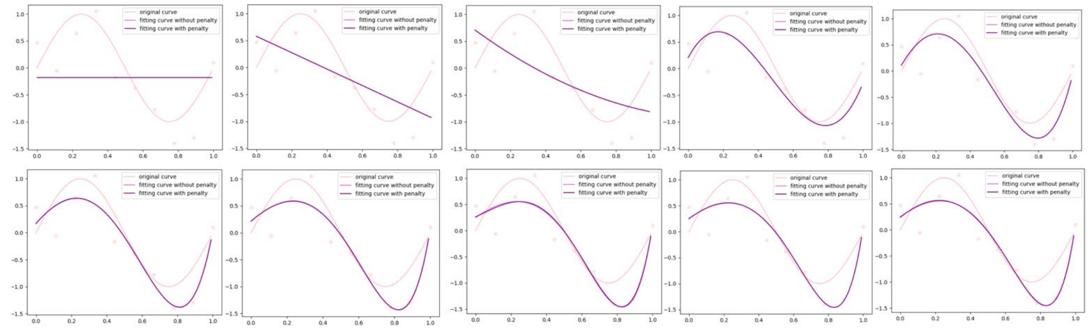


图 7 梯度下降法（epoch=100w, lr=1e-3）在不同多项式阶数下的结果

下图展示了多项式次数 M=9 时，运用梯度下降法训练 100w 轮的损失函数变

化情况，可见损失函数已经接近于 0，随着训练的进行变化甚微。这是因为在接近最优解处梯度变化缓慢，需要更多的迭代次数才能达到最优解状态。

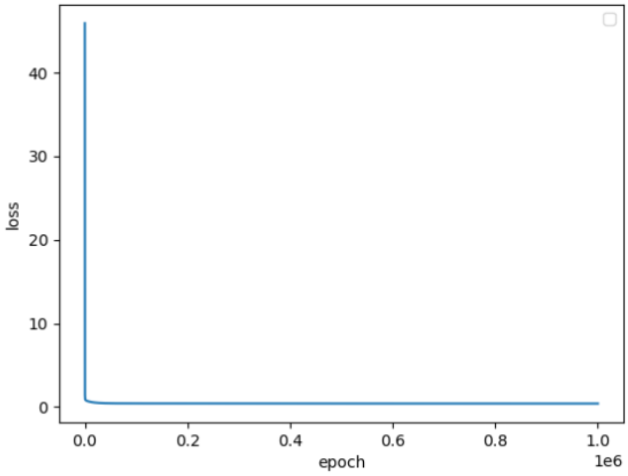


图 8  梯度下降法 M=9, epoch=100w 时损失函数变化情况

**4.1.3  共轭梯度**

共轭梯度方法的结果与解析解求解方法比较相似。随着多项式阶数增加，曲线由欠拟合到拟合又到过拟合，且在过拟合情况下，加入正则化惩罚项对模型效果有较好的改善。

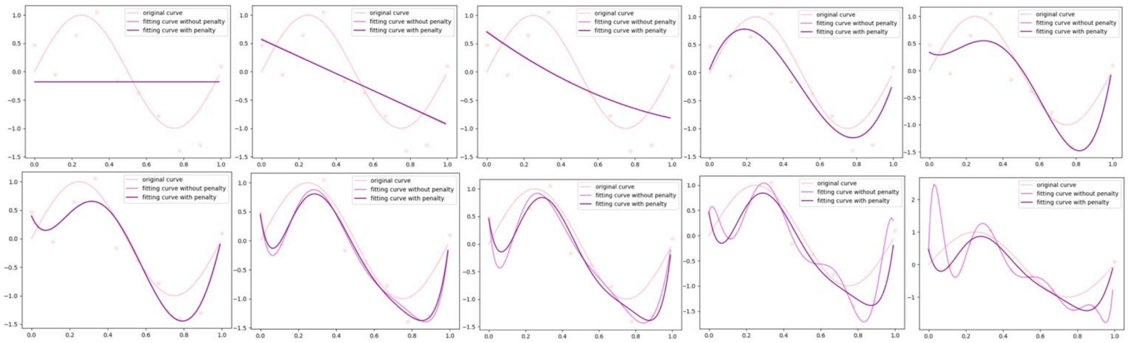


图 9  共轭梯度法在不同多项式阶数下的结果

## 4.2  不同数据量对结果的影响

取多项式次数 M=9，正则化项系数 $\lambda$ 满足 $\ln\lambda = -18$，分别取数据量 N=10, 50，100, 1000 进行实验。

**4.2.1  解析解**

当数据量较小（N=10, 50）时，在用高阶多项式进行拟合时，出现了明显的过拟合现象，在添加惩罚项后有所缓解。当数据量达到 N=100, 1000 时，即使多项式复杂、阶数高，也能很好的拟合原曲线，此时不需要惩罚项也不会出现过拟合现象。

这是由于，随着数据量的增多，数据整体的分布情况更容易被捕捉，使得复杂模型受随机噪声的影响减弱，不易出现过拟合现象。可见增多数据量也是缓解过拟合的方式之一。

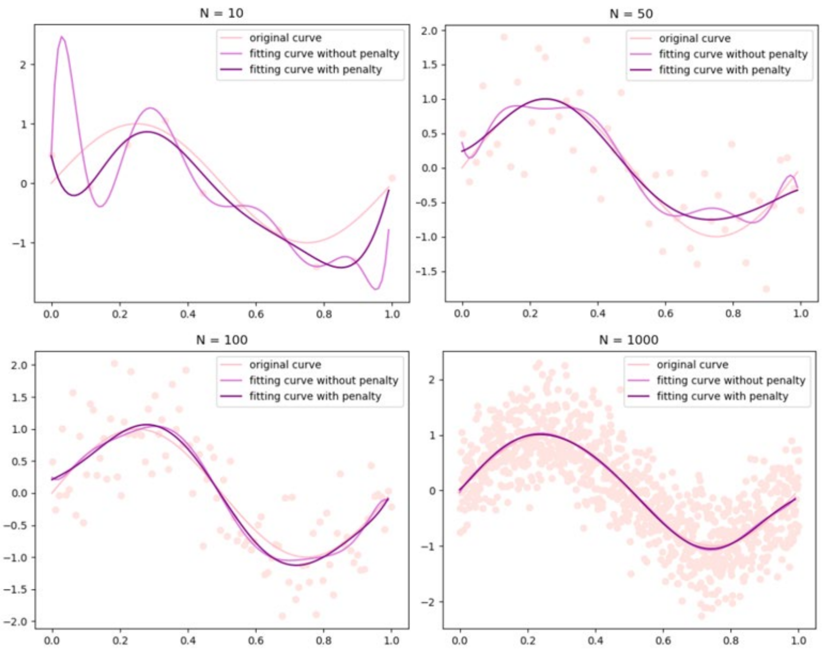


图 10 M=9 时解析解方法在不同数据量下的结果

### 4.2.2 梯度下降

取迭代次数 epoch=100w，学习率 learning rate=1e-3，随着数据量增大，使用梯度下降方法的拟合效果也越来越好，最终得到的拟合曲线与原曲线几乎重合。

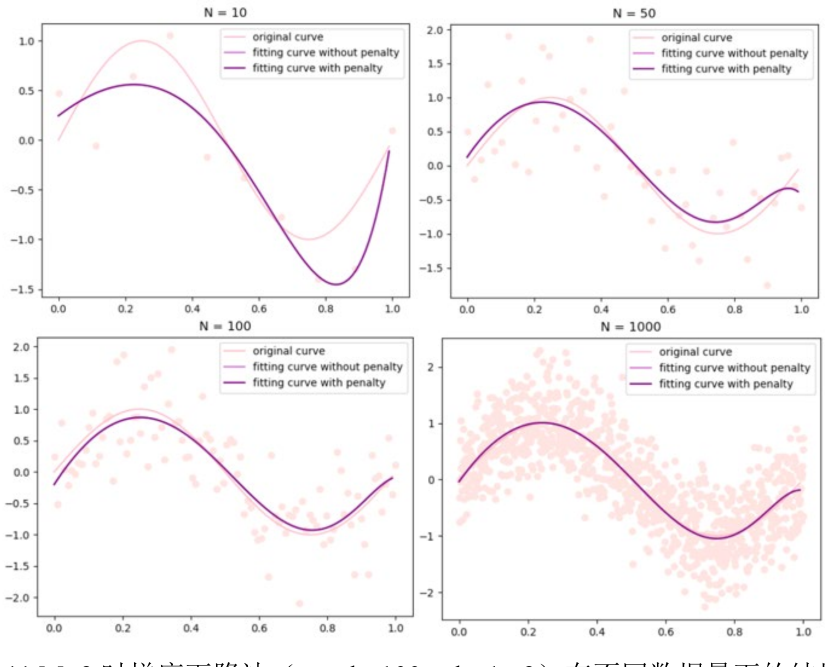


图 11 M=9 时梯度下降法（epoch=100w, lr=1e-3）在不同数据量下的结果

进一步实验发现，当数据量足够大（取 N=1000）时，可以在更少的迭代次数和更小的学习率的情况下，得到较优的拟合曲线。如下图所示，减小迭代次数和学习率也可以比较好地拟合原正弦曲线。因而，在数据量较大时，梯度下降可以在更小的步长下更快收敛。也就是说，更多的数据提供了更强的指导作用，使得梯度下降更容易找到最优解。

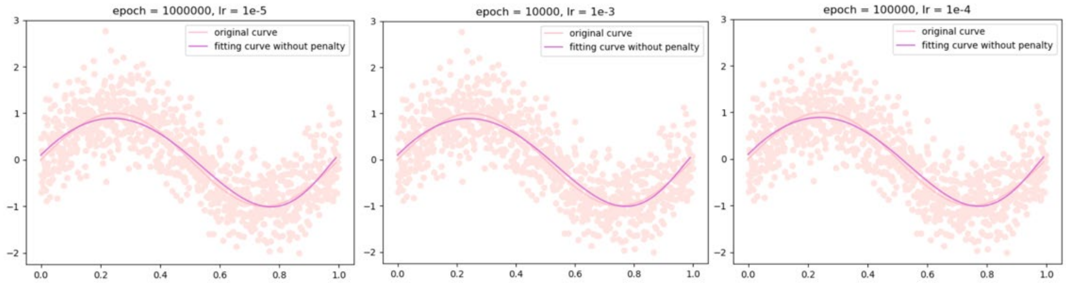


图 12  数据量 N=1000 时减小迭代次数和学习率的结果

#### 4.2.3  共轭梯度

共轭梯度法的结果与解析解法类似，随数据量增大，拟合效果渐佳，过拟合现象减弱。从本次实验的结果看来，在数据量逐渐增大的过程中，解析解法得到的曲线比共轭梯度法更快接近原正弦曲线。

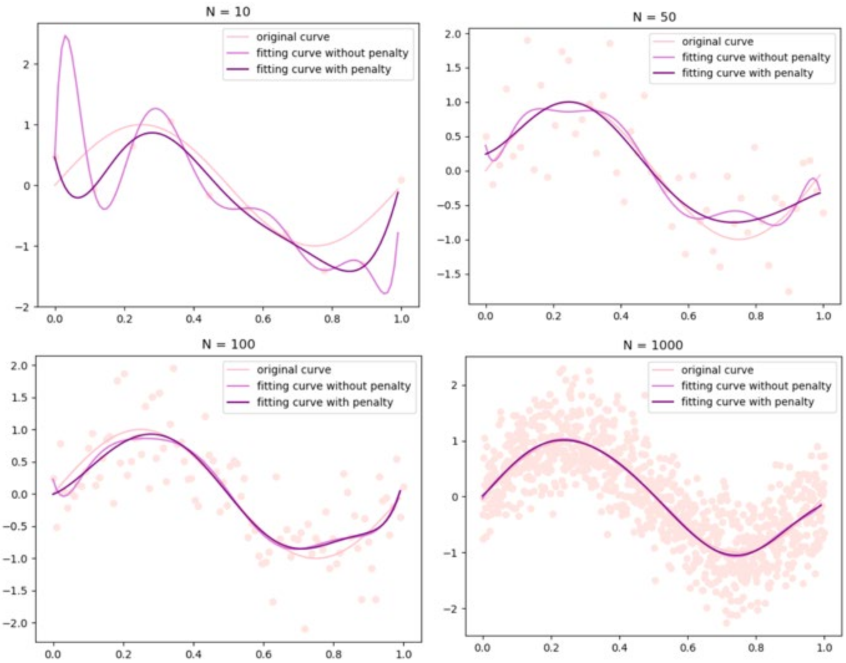


图 13 M=9 时共轭梯度法在不同数据量下的结果

### 4.3  不同超参数对结果的影响

#### 4.3.1  梯度下降的学习率

学习率，即梯度下降中的迭代步长。学习率是重要的超参数之一，若学习率的选择过小，一般会导致梯度下降过慢；若学习率的选择过大，可能导致迭代过程中出现震荡，无法找到最优解。

取 M=3, N=10, epoch=100w，分别在学习率为 0.15, 1e-3, 1e-6 下进行实验，观察结果发现，当学习率为 1e-3 时，模型可以较好收敛，拟合效果较好。当学习率较大达到 0.15 时，随着训练轮数增加，损失函数会突然急剧暴增，甚至产生溢出现象（拟合曲线为 epoch=1000 时得到的，随后会造成溢出），这是由于迭代过程中，在接近最优解的附近发生震荡，导致不仅无法找到最优解，还会距离最优解越来越远。学习率较低仅为 1e-6 时，100w 轮迭代后模型并未收敛到最优解，损失函数下降速率相较学习率为 1e-3 时也慢很多。

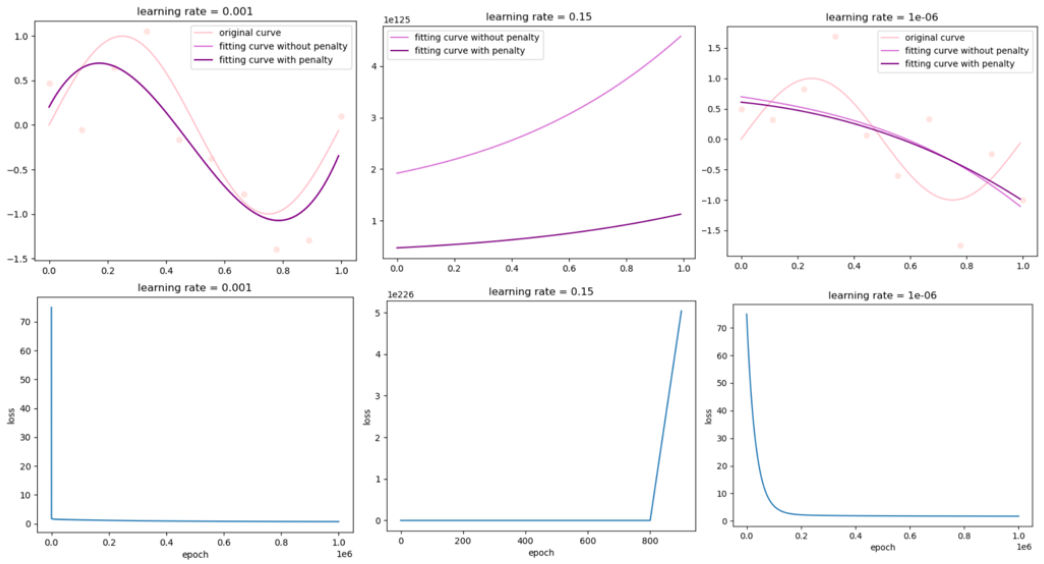


图 14 梯度下降（M=3, epoch=100w）学习率分别为 1e-3, 0.15, 1e-6 时拟合曲线与损失变化

### 4.3.2 梯度下降训练轮数

当学习率取适当值时，随着训练轮数增加，模型逐渐收敛，损失逐渐降低，曲线逐渐靠近最优解。当训练轮数不足时，模型无法达到收敛训练就停止了，但若训练轮数过多，也会导致时间与资源的过度浪费。

在训练数据量 N=10 下，取学习率 learning rate=1e-3，多项式阶数 M=3，在训练轮数为 100, 1w, 100w 下分别进行实验。当训练轮数过少时，模型还未收敛，此后曲线逐渐靠近最优解。

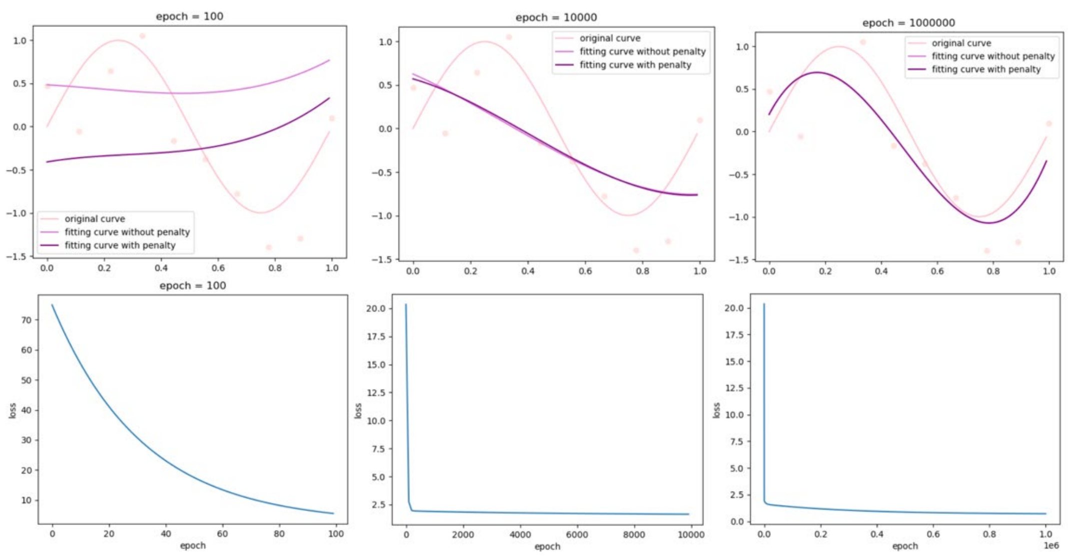


图 15 梯度下降（M=3, lr=1e-3）训练轮数分别为 100, 1w, 100w 时拟合曲线与损失变化

在前文的实验中，训练数据量为 10，多项式阶数为 9 时，在训练 100w 轮时模型仍无法收敛。这里继续取多项式阶数 M=9，学习率 learning rate=1e-3，在 100w 的基础上增加训练轮数，观察训练 1000w 轮和 1 亿轮后，曲线变化如下所示：

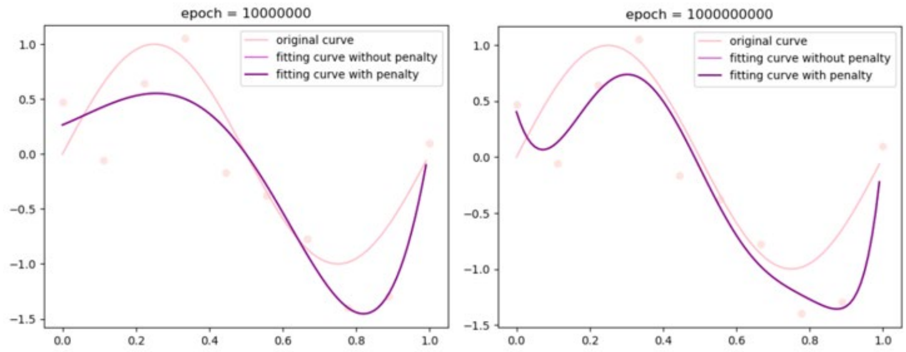


图 16 M=9 梯度下降训练轮数为 100w 和 1 亿的结果

可以看到曲线逐渐已经显露出过拟合的趋势，但仍未达到最优解。如果再继续增加训练轮数，便可能存在达到最优解的情况。故训练轮数的增加有助于为收敛的模型进一步收敛。

#### 4.3.3 惩罚项系数

惩罚项系数$\lambda$调控着模型的复杂度，决定了过拟合的程度。$\lambda$越大，对模型复杂度的抑制作用越强。

取 N=10, M=9，使$\lambda$=e-18, 1，解析解求解方法下模型的拟合效果如下图所示。可见当$\lambda$=e-18 时惩罚项使得过拟合现象得到缓解，曲线能够较好拟合；$\lambda$=1 时惩罚项抑制效果过强，导致模型过于简单，接近于线性，仍不能很好拟合。

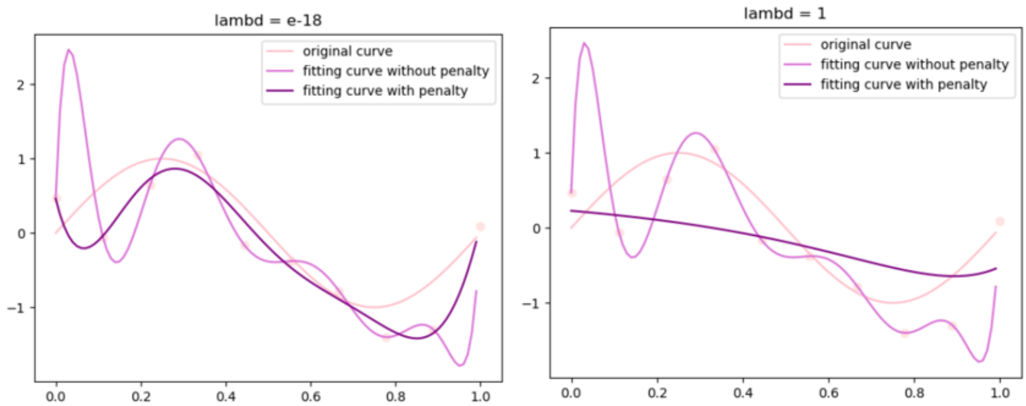


图 17 解析解求解方法$\lambda$=e-18, 1 的结果

### 4.4 三种优化方法的比较

#### 4.4.1 解析解

解析解求解的优点是速度较快，准确度较高；缺点是对样本有条件限制，即根据方程 $\mathbf{w}=(X^TX)^{-1}X^TT$ 求解时，要求 $X^TX$ 必须可逆，否则无法用解析解方法求解。

#### 4.4.2 优化解

4.4.2.1 梯度下降法

梯度下降法对样本没有限制，任意的矩阵 $X$ 都可以利用梯度下降法求解。但缺点也很明显，即运行效率低，对调参要求高。需要找到合适的学习率，经过足够的迭代次数才有可能收敛。同时还有进入局部最优解，无法得到全局最优解的

风险。

4.4.2.2 共轭梯度法

共轭梯度法相较梯度下降法收敛较快，运行时速度有明显提升，且在相同的迭代次数下，共轭梯度法得到的参数对应的损失函数明显小于梯度下降法。但共轭梯度法也对线性方程组的系数有要求，对 $AX=b$，只有当矩阵 $A$ 是实对称正定矩阵时才能使用共轭梯度法。

## 4.5 过拟合现象

### 4.5.1 过拟合现象的分析

当训练数据较少（N=10）、多项式阶数过高（M=9）时，会产生过拟合现象。如下图所示，得到的曲线对于训练数据中的每个点都能精确拟合，但却不是目标正弦曲线。这是由于训练数据过少，模型过于复杂以至于过度学习到了噪声信息，因而出现了过拟合现象。

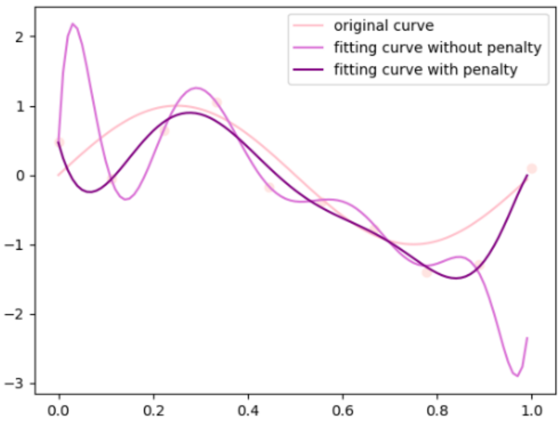


图 18 过拟合现象

此时，学习到的参数权重也剧烈增大。下图比较了在 N=10 时，多项式阶数分别为 M=3 与 M=9 的参数大小，可见 M=9 的过拟合情况下，系数规模过大，这并不是所期待的。因而，需要引入一些方法克服过拟合。

```
M=3:
[[ 18.75866653 -27.27615103   8.28822887   0.06132937]]
M=9:
[[ 6.83340181e+04 -3.18385385e+05  6.23839987e+05 -6.66616209e+05
   4.20823056e+05 -1.58549056e+05  3.41288175e+04 -3.72717874e+03
   1.51578994e+02  4.70857432e-01]]
```

图 19 M=3 与 M=9 时的参数

### 4.5.2 克服过拟合的方法

4.5.1 在损失函数中增加正则化惩罚项

正如 4.1.1 和 4.1.3 中所讨论的，加入正则化项后，对参数 **w** 的权值大小有了惩罚，可以显著减小系数的值，使得模型结构趋向于简单。下图比较了 M=9 时加入正则化项前后模型参数的权值，可见惩罚项使得模型参数大幅度减小，有利于缓解过拟合现象。

```
without penalty:
[[ 6.83340181e+04 -3.18385385e+05  6.23839987e+05 -6.66616209e+05
   4.20823056e+05 -1.58549056e+05  3.41288175e+04 -3.72717874e+03
   1.51578994e+02  4.70857432e-01]]
with penalty:
[[-3.83585644e+02  8.66557917e+02 -7.83149149e+01 -8.71346005e+02
  -1.97730288e-01  1.13034514e+03 -8.96965157e+02  2.56605446e+02
  -2.34541949e+01  4.59399821e-01]]
```

图 20 M=9 时加入正则项前后的参数

#### 4.5.2 增大数据规模

随着数据规模的增加，随机噪声的影响减弱，训练数据整体的分布情况愈发凸显，使得即使复杂模型也能捕捉到数据的整体特征，而不会出现对于少量数据过拟合的现象，详见 4.2.1 和 4.2.3。但是，这对数据量的要求是很高的，在某些领域，数据资源十分宝贵，单纯增大数据规模很可能是不现实的。

# 五、结论

随着数据量增大，曲线拟合效果越来越好；

随着多项式阶数增加，曲线由欠拟合到拟合到过拟合，但过拟合可以通过正则化方法来缓解，惩罚项的系数控制了对模型复杂度的抑制强度；

在梯度下降中，学习率决定了参数更新的步长，应根据模型结构选择恰当值；随着迭代次数增多，模型逐渐收敛，靠近最优解，但也有可能陷入局部最优解，无法达到全局最优解；

解析解对矩阵的可逆性有要求，优化解相对要求较少，但代价是收敛速率变慢，容易止步于局部最优。

# 六、参考文献


[1] Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.

[2] 周志华. 2016. 机器学习.

[3] https://builtin.com/data-science/gradient-descent

[4] https://en.wikipedia.org/wiki/Conjugate_gradient_method

[5] Strang, G. 2006. Linear algebra and its applications. Belmont, CA: Thomson, Brooks/Cole.


# 七、附录：源代码（带注释）

```python
import random
import numpy as np
from random import gauss
import matplotlib.pyplot as plt
random.seed(0)
np.random.seed(0)
def generate_data(N, mu=0, sigma=0.5):
    # generate true values
    x = np.linspace(0, 1, N, endpoint=True)
    y = np.sin(2 * np.pi * x)
    # add gaussian noise
    for i in range(x.size):
        y[i] += gauss(mu, sigma)
    return x, y
```

```
def analyse_without_penalty(order, x, y):
    vander = np.vander(x, order + 1)  # Vandermonde matrix of x
    w = np.dot(np.matrix(np.dot(vander.T, vander)).I, np.dot(vander.T, y))  # the solution of w
    return w


def analyse_with_penalty(order, x, y, lambd):
    vander = np.vander(x, order + 1)  # Vandermonde matrix of x
    w = np.dot(np.matrix(np.dot(vander.T, vander) + lambd * np.identity(order + 1)).I, np.dot(vander.T, y))
    return w


def gradient_descent_per_epoch(params, grads, learning_rate):
    params -= grads * learning_rate
    return params


def gradient_descent_without_penalty(order, epoch, x, y, learning_rate=0.001, plot_loss=False):
    w = np.random.randn(order + 1)
    vander = np.vander(x, order + 1)

    epochs = []
    losses = []
    for e in range(epoch):
        logits = np.dot(vander, w)  # prediction
        loss = np.sum(np.square(logits - y)) / 2  # the value of loss function
        grads = np.sum((logits - y) * vander.T, axis=-1)  # compute the gradient

        # w -= grads * learning_rate
        w = gradient_descent_per_epoch(w, grads, learning_rate)  # update

        # save epoch and related loss in order to plot later
        if e % 100 == 0:
            epochs.append(e)
            losses.append(loss)
        # epochs.append(e)
        # losses.append(loss)

```

```
    # plot the tendency of loss
    if plot_loss:
        visualize_loss(losses, epochs)

    return w


def gradient_descent_with_penalty(order, epoch, x, y, lambd, learning_rate=0.001, plot_loss=False):
    w = np.random.randn(order + 1)
    vander = np.vander(x, order + 1)

    epochs = []
    losses = []
    for e in range(epoch):
        logits = np.dot(vander, w)  # prediction
        loss = np.sum(np.square(logits - y)) / 2  # value of loss function
        grads = np.sum((logits - y) * vander.T, axis=-1) + lambd * w  # compute the gradient
        w = gradient_descent_per_epoch(w, grads, learning_rate)  # update

        if e % 100 == 0:
            epochs.append(e)
            losses.append(loss)

    if plot_loss:
        visualize_loss(losses, epochs)

    return w


def conjugate_gradient(A, b, w):
    r = b - np.dot(A, w)
    p = r
    while np.dot(r, r) > 1e-16:
        a = np.dot(r, r) / np.dot(np.dot(p, A), p)
        w += a * p
        r_pre = r.copy()
        r -= a * np.dot(A, p)
        b = np.dot(r, r) / np.dot(r_pre, r_pre)
        p = r + b * p
    return w
```

```python


def conjugate_gradient_without_penalty(order, x, y):
    w = np.random.randn(order + 1)
    vander = np.vander(x, order + 1)
    # AX = b
    A = np.dot(vander.T, vander)
    b = np.dot(vander.T, y)
    return conjugate_gradient(A, b, w)


def conjugate_gradient_with_penalty(order, epoch, x, y, lambd):
    w = np.random.randn(order + 1)
    vander = np.vander(x, order + 1)
    A = np.dot(vander.T, vander) + lambd * np.identity(order + 1)
    b = np.dot(vander.T, y)
    return conjugate_gradient(A, b, w)


# visualize the curve: origin/with penalty/without penalty
def visualize(type, order, x=None, y=None, w=None):
    x_continuous = np.arange(0, 1, 0.01)

    if type == 'naive':
        y_fit = np.dot(np.vander(x_continuous, order + 1), w.T)
        plt.plot(x_continuous, y_fit, color='orchid', label='fitting curve without penalty')

    elif type == 'origin':
        y_sin = np.sin(2 * np.pi * x_continuous)
        plt.scatter(x, y, color='mistyrose')
        plt.plot(x_continuous, y_sin, color='pink', label='original curve')

    elif type == 'penalty':
        y_fit = np.dot(np.vander(x_continuous, order + 1), w.T)
        plt.plot(x_continuous, y_fit, color='purple', label='fitting curve with penalty')

    plt.legend()


# visualize the tendency of loss
```

```python
def visualize_loss(loss, epoch):
    plt.plot(epoch, loss)
    plt.xlabel("epoch")
    plt.ylabel("loss")

    plt.show()


# -------------------- test --------------------
def test_order_analyse():
    x, y = generate_data(10)
    for order in range(11):
        visualize('origin', order, x=x, y=y)
        w1 = analyse_without_penalty(order, x, y)
        visualize('naive', order, w=w1)
        w2 = analyse_with_penalty(order, x, y, lambd=np.exp(-18))
        visualize('penalty', order, w=w2)
        plt.show()


def test_order_gd():
    x, y = generate_data(10)
    for order in range(10):
        visualize('origin', order, x=x, y=y)
        w1 = gradient_descent_without_penalty(order, 1000000, x, y)
        visualize('naive', order, w=w1)
        w2 = gradient_descent_with_penalty(order, 1000000, x, y, lambd=np.exp(-18))
        visualize('penalty', order, w=w2)
        plt.show()


def test_order_cg():
    x, y = generate_data(10)
    for order in range(10):
        visualize('origin', order, x=x, y=y)
        w1 = conjugate_gradient_without_penalty(order, x, y)
        visualize('naive', order, w=w1)
        w2 = conjugate_gradient_with_penalty(order, x, y, lambd=np.exp(-18))
        visualize('penalty', order, w=w2)
        plt.show()
```

```


def test_data_analyse():
    order = 9
    for num in [10, 50, 100, 500, 1000]:
        x, y = generate_data(num)
        plt.title('N = ' + str(num))
        visualize('origin', order, x, y)
        w1 = analyse_without_penalty(order, x, y)
        visualize('naive', order, w=w1)
        w2 = analyse_with_penalty(order, x, y, lambd=np.exp(-18))
        visualize('penalty', order, w=w2)

        plt.legend()
        plt.show()


def test_data_gd():
    order = 9
    for num in [10, 50, 100, 500, 1000]:
        x, y = generate_data(num)
        plt.title('N = ' + str(num))
        visualize('origin', order, x, y)
        w1 = gradient_descent_without_penalty(order, 1000000, x, y)
        visualize('naive', order, w=w1)
        w2 = gradient_descent_with_penalty(order, 1000000, x, y, lambd=np.exp(-18))
        visualize('penalty', order, w=w2)

        plt.legend()
        plt.show()


def test_data_cg():
    order = 9
    for num in [10, 50, 100, 500, 1000]:
        x, y = generate_data(num)
        plt.title('N = ' + str(num))
        visualize('origin', order, x, y)
        w1 = conjugate_gradient_without_penalty(order, 1000000, x, y)
        visualize('naive', order, w=w1)
```

```
        w2 = conjugate_gradient_with_penalty(order, 1000000, x, y, lambd=np.exp(-18))
        visualize('penalty', order, w=w2)

        plt.legend()
        plt.show()


def test_lr_gd(lr):
    order = 3
    x, y = generate_data(10)
    plt.title('learning rate = ' + str(lr))
    visualize('origin', order, x, y)
    w1 = gradient_descent_without_penalty(order, 1000000, x, y, learning_rate=lr)
    visualize('naive', order, w=w1)
    w2 = gradient_descent_with_penalty(order, 1000000, x, y, lambd=np.exp(-18), learning_rate=lr)
    visualize('penalty', order, w=w2)
    plt.show()


def test_epoch_gd(epoch):
    order = 9
    x, y = generate_data(10)
    plt.title('epoch = ' + str(epoch))
    visualize('origin', order, x, y)
    w1 = gradient_descent_without_penalty(order, epoch, x, y, learning_rate=1e-3)
    visualize('naive', order, w=w1)
    w2 = gradient_descent_with_penalty(order, epoch, x, y, lambd=np.exp(-18), learning_rate=1e-3)
    visualize('penalty', order, w=w2)
    plt.show()

    # w1 = gradient_descent_without_penalty(order, epoch, x, y, learning_rate=1e-3, plot_loss=True)


if __name__ == '__main__':
    test_order_analyse()
    test_order_cg()
    test_order_gd()
    test_data_analyse()
```

```python
    test_data_gd()
    test_data_cg()
    # test_epoch_gd(50000000)

    # x, y = generate_data(10)
    # w = gradient_descent_without_penalty(3, 1000000, x, y, plot_loss=True)

    # x, y = generate_data(10)
    # visualize('origin', 9, x, y)
    # plt.title('lambd = 1')
    # w1 = analyse_without_penalty(9, x, y)
    # visualize('naive', 9, w=w1)
    # w = analyse_with_penalty(9, x, y, lambd=1)
    # visualize('penalty', 9, w=w)
    # plt.show()

    # x, y = generate_data(10)
    # w1 = analyse_without_penalty(9, x, y)
    # w2 = analyse_with_penalty(9, x, y, lambd=np.exp(-18))
    # print('without penalty:')
    # print(w1)
    # print('with penalty:')
    # print(w2)

    # w = analyse_with_penalty(order, x, y, np.exp(-18))
    # w = conjugate_gradient_without_penalty(order, 10, x, y)
```