哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

机器学习实验报告

课程类型 机器学习

题　　目 多项式拟合曲线

学　　号 1160300507

姓　　名 聂晨晞

计算机科学与技术学院

目录

第 1 章 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、共轭梯度法、理解过拟合、克服过拟合的方法（如加惩罚项、添加样本）

第 2 章 实验要求及实验环境

2.1 实验要求

1. 生成数据，加入以高斯分布的噪声
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，随机梯度下降，共轭梯度）
5. 用你得到的数据，解释过拟合
6. 用不同数据量，不同超参数，不同多项式阶数，比较实验效果
7. 语言不限，可以用 matlab, python。求解解析解是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。不允许利用现有的平台例如 pytorch，tensorflow 的自动微分工具

2.2 实验环境

1. 硬件环境：macbook pro mid-2017，3.1Ghz i5 处理器，16GB 内存
2. 软件环境：macOS 10.14+pycharm professional 2018.1+python 版本：3.6.4

第 3 章 设计思想

3.1 数学符号的定义

在讨论各个具体算法之前，本文定义如下的若干符号

1. M 为该多项式的次数
2. N 为样本数
3. $\boldsymbol{x} = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_N \end{bmatrix}^T$ 表示着这 N 个数据对应的 x 坐标
4. $\boldsymbol{T} = \begin{bmatrix} t_0 & t_1 & t_2 & \cdots & t_N \end{bmatrix}^T$ 表示着这 N 个数据对应的答案值
5. $\boldsymbol{w} = \begin{bmatrix} w_0 & w_1 & w_2 & \cdots & w_M \end{bmatrix}^T$ 表示该多项式的所有系数组成的列向量

6. $\boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^M \\ 1 & x_2 & x_2^2 & \cdots & x_2^M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^M \end{bmatrix}$ 表示由 N 个数据 x 生成的矩阵

7. $y(x_n, \boldsymbol{w})$ 表示在系数 $\boldsymbol{w}$ 的情况下，多项式在输入 $\boldsymbol{X}$ 的时候，预测的 y 值组成的向量

8. x_n^i 表示数据矩阵 $\boldsymbol{X}$ 中的第 i 行，第 n 列

3.2 用解析解求解无正则项的 loss 函数

3.2.1 算法原理

没有正则项的 loss 函数可以使用如下的公式进行描述

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

为了求得公式1的最小值，我们对其进行对 $\boldsymbol{w}$ 的求导操作得到如下的等式

$$\begin{aligned} \frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} &= (\boldsymbol{X}\cdot\boldsymbol{w} - \boldsymbol{t})^T \cdot (\boldsymbol{X}\cdot\boldsymbol{w} - \boldsymbol{t}) \\ &= \boldsymbol{w}^{\boldsymbol{T}}\cdot\boldsymbol{X}^{\boldsymbol{T}}\cdot\boldsymbol{X}\cdot\boldsymbol{w} - 2\boldsymbol{w}^{\boldsymbol{T}}\cdot\boldsymbol{x}^{\boldsymbol{T}}\cdot\boldsymbol{t} + \boldsymbol{t}^{\boldsymbol{T}}\cdot\boldsymbol{t} \end{aligned} \tag{2}$$

我们将公式2等于 0，化简，得到如下的结果

$$\begin{aligned} 2\boldsymbol{X_T}\cdot\boldsymbol{X}\cdot\boldsymbol{w} - \boldsymbol{2}\cdot\boldsymbol{X_T}\cdot\boldsymbol{T} &= 0 \\ \boldsymbol{X}^{\boldsymbol{T}}\cdot\boldsymbol{X}\cdot\boldsymbol{w} &= \boldsymbol{X}^{\boldsymbol{T}}\cdot\boldsymbol{T} \\ \boldsymbol{w} &= (\boldsymbol{X}^{\boldsymbol{T}}\cdot\boldsymbol{X})^{-1}\cdot\boldsymbol{X}^{\boldsymbol{T}}\cdot\boldsymbol{T} \end{aligned} \tag{3}$$

3.2.2 算法实现

本次实验中，在公式3的指导下，可以直接通过 numpy 的矩阵表示相关内容进行，解析解的计算。这里需要注意的问题就是在计算矩阵的逆的时候，我们需要首先验证该矩阵是不是可逆矩阵，可以使用矩阵对应的行列式的值不为 0 的方法进行计算，该部分的核心代码如下所示：

```
if det(self.__X.transpose().dot(self.__X)) != 0.0:
    W = inv(self.__X.transpose().dot(self.__X)).dot( \
            self.__X.transpose()).dot(self.__y)
    self.__W = W
```

3.3 用解析解求解带有正则项的 loss 函数的解析解

3.3.1 算法原理

为了解决上一个不带有正则项的 loss 函数可能出现的过拟合问题，在这里引入一个正则项 λ，带有正则项的 loss 函数如下所示

$$\begin{aligned} E(\boldsymbol{w}) &= \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\boldsymbol{w})-t_n\}^2+\frac{1}{2}\left\|\boldsymbol{w}\right\|^2 \\ &= \boldsymbol{w^T\cdot X^T\cdot X\cdot w}-2\boldsymbol{w^T\cdot x^T\cdot t}+\boldsymbol{t^T\cdot t}+\boldsymbol{\lambda\cdot w^T\cdot w} \end{aligned} \tag{4}$$

对公式4.6对 $\boldsymbol{w}$ 求导，可以得到如下的结果：

$$\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}=\boldsymbol{2\cdot X^T\cdot X\cdot w-2\cdot X^T\cdot T+2\lambda\cdot w} \tag{5}$$

为了求得带正则项的 loss 函数的最小值，我们将公式5右边等于 0，并化简得到如下的结果

$$\begin{aligned} \boldsymbol{2\cdot X^T\cdot X\cdot w-2\cdot X^T\cdot T+2\lambda\cdot w}&=0 \\ \boldsymbol{(X^T\cdot X+\lambda\cdot I)\cdot w}&=\boldsymbol{X^T\cdot T} \\ \boldsymbol{w}&=(X^T\cdot X+\lambda\cdot I)^T\cdot X^T\cdot T \end{aligned} \tag{6}$$

3.3.2 算法实现

该算法的实现并不复杂，同样可以使用 numpy 中与矩阵相关的内容进行计算，主要的局限性也是在于必须要求 $X^T\cdot X+\lambda\cdot I$ 可逆，核心代码如下所示：

```
if det(self.__X.transpose().dot(self.__X)) != 0.0:
    W = inv(self.__X.transpose().dot(self.__X) + lam * np.eye((self.__m+1)) \
        .dot(self.__X.transpose()).dot(self.__y)
    self.__W = W
```

3.4 使用梯度下降法求解带有正则项的 loss 函数

3.4.1 算法原理

梯度下降法是机器学习中的一种常用方法，它通过对待求函数的当前点的梯度的反方向进行规定步长的迭代搜索，可以找到函数的一个局部最小值点。对于多项式回归问题，梯度下降法可以通过如下的伪代码进行描述：

Repeat until converge {

$$
\begin{aligned}
\boldsymbol{w_0} &= \boldsymbol{w_0} - \alpha \cdot \frac{1}{N} \cdot \sum_{i=1}^{N} (y(x_i, \boldsymbol{w}) - t^i) \cdot x_0^i + \lambda \cdot \boldsymbol{w} \\
\boldsymbol{w_1} &= \boldsymbol{w_1} - \alpha \cdot \frac{1}{N} \cdot \sum_{i=1}^{N} (y(x_i, \boldsymbol{w}) - t^i) \cdot x_1^i + \lambda \cdot \boldsymbol{w} \\
\boldsymbol{w_2} &= \boldsymbol{w_2} - \alpha \cdot \frac{1}{N} \cdot \sum_{i=1}^{N} (y(x_i, \boldsymbol{w}) - t^i) \cdot x_2^i + \lambda \cdot \boldsymbol{w} \\
&\vdots \\
\boldsymbol{w_m} &= \boldsymbol{w_m} - \alpha \cdot \frac{1}{N} \cdot \sum_{i=1}^{N} (y(x_i, \boldsymbol{w}) - t^i) \cdot x_m^i + \lambda \cdot \boldsymbol{w}
\end{aligned}
\tag{7}
$$

}

为了加速运算，通过整理，我们将该算法整理成为了如下的矩阵运算形式

$$
\boldsymbol{w} = \boldsymbol{w} - \alpha \cdot \frac{1}{N} \cdot X^T \cdot (X \cdot \boldsymbol{w} - y) + \lambda * \boldsymbol{w} \tag{8}
$$

3.4.2 算法实现

在上一节中，我们论述了梯度下降的矩阵形式，下面通过一个 oython 程序将本算法进行一个实现。实现的基本思路是通过不断的进行以上的矩阵计算，每进行一次计算都是用如下的公式计算 RMS，直到循环次数达到阈值或者两次循环之间的差值不超过另一个阈值，推出循环。

$$
E_{RMS} = \sqrt{2E(\boldsymbol{w} * /N)} \tag{9}
$$

该部分的核心代码如下所示:

```
while True:
    iter += 1
    term_vector = self.__X.dot(W) - self.__y
    error_before = self.root_mean_square()
    W = W - lr / len(self.__x) *  self.__X.transpose().dot(term_vector) + lam * W
    self.__W = W
    error_after = self.root_mean_square()
    if np.abs(error_before - error_after) <= thresh or iter == iteration_thresh:
        break
```

3.5 使用随机梯度下降求解带有正则项的 loss 函数

由于梯度下降使用时，每一次循环都需要利用所有的数据进行矩阵的计算，通常来说，这种方法收敛速度较慢，而随机梯度下降的出现每一次进行更新的时候只使用数据集 X 的某一行，可以大大加快每次循环所需要的时间，同时也可以加快收敛的速度。

3.5.1 算法设计

随机梯度下降方法和梯度下降的方法主要区别在于，随机梯度下降每一次都只取训练集的一行进行更新，算法如下所示

Repeat until converge {

$$
\begin{aligned}
\boldsymbol{w_0} &= \boldsymbol{w_0} - \alpha \cdot (y(x_i, \boldsymbol{w}) - t^i) \cdot x_0^i + \lambda \cdot \boldsymbol{w} \\
\boldsymbol{w_1} &= \boldsymbol{w_1} - \alpha \cdot (y(x_i, \boldsymbol{w}) - t^i) \cdot x_1^i + \lambda \cdot \boldsymbol{w} \\
\boldsymbol{w_2} &= \boldsymbol{w_2} - \alpha \cdot (y(x_i, \boldsymbol{w}) - t^i) \cdot x_2^i + \lambda \cdot \boldsymbol{w} \\
&\vdots \\
\boldsymbol{w_m} &= \boldsymbol{w_m} - \alpha \cdot (y(x_i, \boldsymbol{w}) - t^i) \cdot x_m^i + \lambda \cdot \boldsymbol{w}
\end{aligned}
\tag{10}
$$

}

同样，我们为了计算方便，可以将以上的算法写成矩阵形式如下：

Repeat until converge {

$$
\boldsymbol{w} = \boldsymbol{w} - \alpha \cdot (X^i \cdot \boldsymbol{w} - T^i) \cdot {X^i}^T + \lambda \cdot \boldsymbol{w} \tag{11}
$$

}

3.5.2 算法实现

经过上一节的讨论，我们得到了随机梯度下降的矩阵形式，我们这一节给出一些相应的完善措施，将该公式完成成一个 python 程序，首先和梯度下降一样，需要设定两个阈值，一个为最大循环的次数，另一个为两次循环之间 RMS 的最小差值，其次，由于随机梯度下降每次循环只需要该数据集合的某一行，当循环次数超过数据集合的行数的时候，我们需要将数据集合进行循环利用，具体的 python 代码实现如下所示：

```
while True:
    diff = self.__y[i] - self.__X[i].reshape(1, -1).dot(self.__W)
    error_before = self.root_mean_square()
    W = W + lr * diff * self.__X[i].reshape(-1, 1) + lam * W
    self.__W = W
    error_after = self.root_mean_square()
    if np.abs(error_before - error_after) < thresh or iter == iteration_thresh:
        break
    i = (i + 1) % len(self.__x)
    iter += 1
```

3.6 共轭梯度法

即使是随机梯度下降法，在本次试验中为了取得较好的结果，通常也需要较多的循环次数，而为了进一步降低循环的次数和加快求解的速度，可以使用共轭梯度法。

3.6.1 算法设计

共轭梯度法是一种用于求解对称正定方程 $A\cdots X=y$ 的方法，由公式6在我们的试验中，A 对应着 $(\boldsymbol{X^T\cdot X+\lambda\cdot I})\cdot\boldsymbol{w}$，我们的 $\boldsymbol{X^T\cdot T}$。该方法的思想主要是找到 n 个两两共轭的共轭方向，每次沿着一个方向优化得到该方向上的极小值，后面再沿其它方向求极小值的时候，不会影响前面已经得到的沿哪些方向上的极小值，所以理论上对 n 个方向都求出极小值就得到了 n 维问题的极小值。该算法的伪代码从维基百科上摘抄如下：

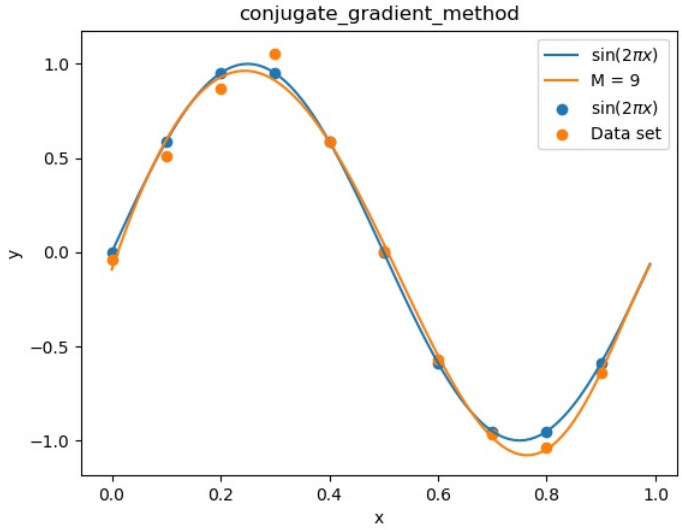


图 3.1: 共轭梯度法伪代码

3.6.2 算法实现

根据上面的伪代码，不难得到共轭梯度法的 python 程序如下所示：

```
r = y - X.dot(self.__W)
p = r
rsold = r.transpose().dot(r)
for i in range(len(self.__x)):
    Xp = X.dot(p)
    alpha = rsold / (p.transpose().dot(Xp))
    self.__W = self.__W  + alpha * p
    r = r - alpha * Xp
    rsnew = r.transpose().dot(r)
    if rsnew < 1e-6:
        break
    else:
        p = r + rsnew/rsold * p
        rsold = rsnew
```

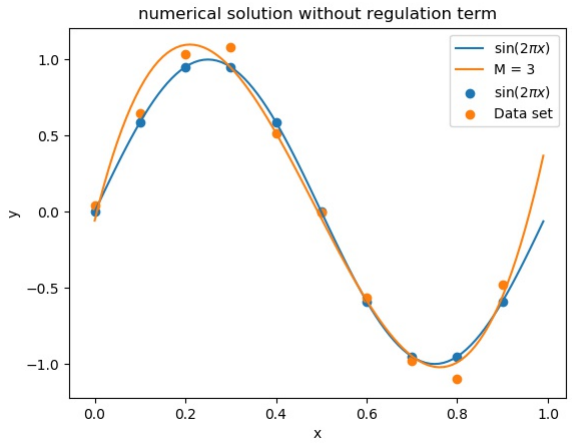


图 4.1: 三阶多项式时未过拟合

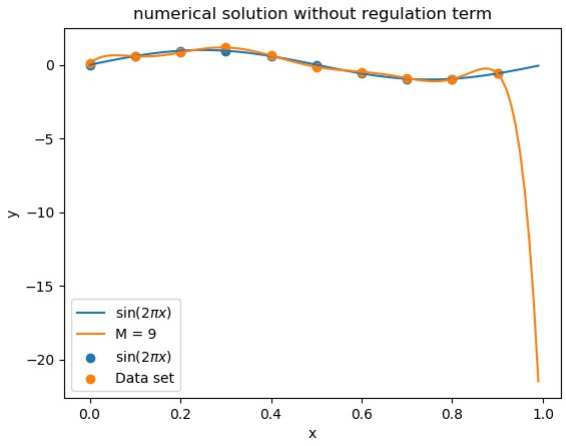


图 4.2: 九阶多项式时过拟合

第 4 章 实验结果与分析

4.1 使用解析解求解不含有正则项的 loss 函数

4.1.1 求解最优解

使用上一个 section 所述的算法，我们在使用 3 阶多项式的时候得到了如图4.1的曲线，它的 RMS 值是：0.09

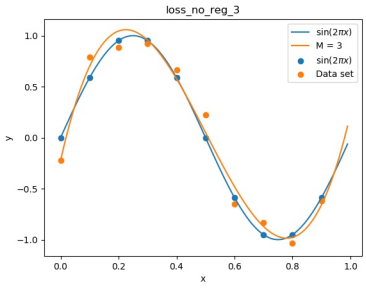
4.1.2 过拟合

在这种情况下，非常容易出现过拟合的情况，只要将多项式的阶数值调到一个较大的时候就可以了，如图4.2所示，出现过拟合的时候该模型过分的拟合了图中的数据点（此时 RMS 为 0.0004）以至于该模型对其他的输入预测能力很弱，就像该图所示意的一样，出现过拟合的时候该多项式的曲线很明显并不是一个和 $\sin(2\pi x)$ 相近的曲线，此时如果再次产生一组数据，使用这一组数据计算 RMS，该模型给出的值为 4.5。

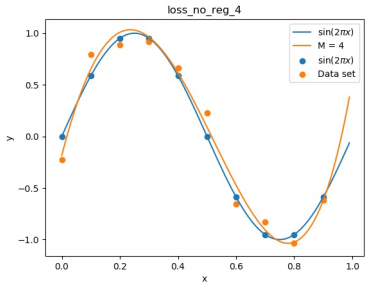
4.1.3 使用不用的数据量，不同的多项式阶数结果比较

为了完成本小节的实验，我选取了 $M = 2, 3, 4, 5, 6, 7, 8, 9$ 共让该模型跑 8 次，得到随着 M 阶数的变化 RMS 分别在训练集和测试集上的表现如图4.3(h)所示。每一次的结果如图4.3所示。可以看出，不改变数据集大小的情况下，随着模型复杂度的上升，出现了过拟合的现象。

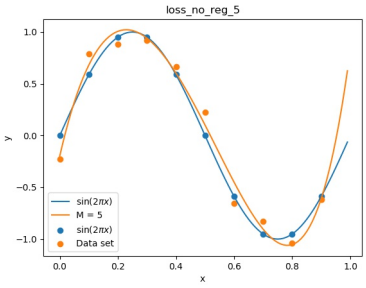
对于不同的数据集大小，我们选取了从 10～100 的数据集大小，每次增加 10，在多项式项数设置为 9 的时候进行了 10 次实验，得到的结果如图4.4所示，可以看出，当数据集大小开始增加的时候，模型预测的曲线逐渐向正确的 $sin(2\pi x)$ 靠拢，说明增加数据集大小是一种克服过拟合的方法。



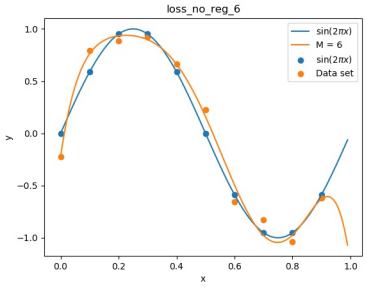
(a) 三阶多项式



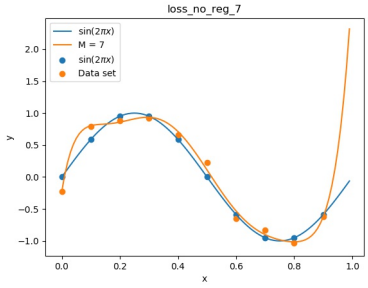
(b) 四阶多项式



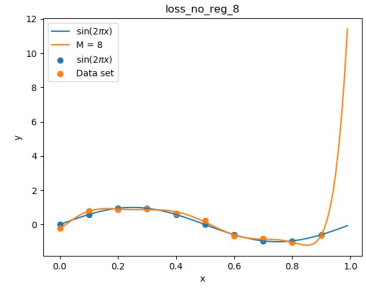
(c) 五阶多项式



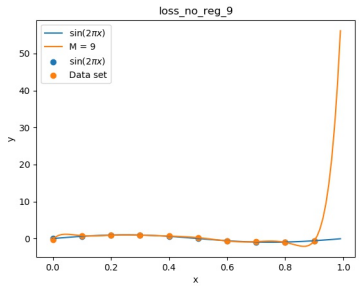
(d) 六阶多项式



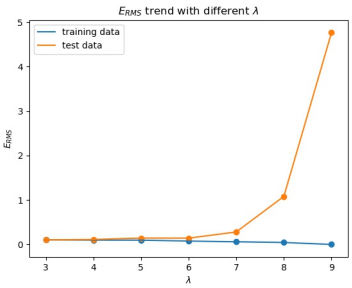
(e) 七阶多项式



(f) 八阶多项式

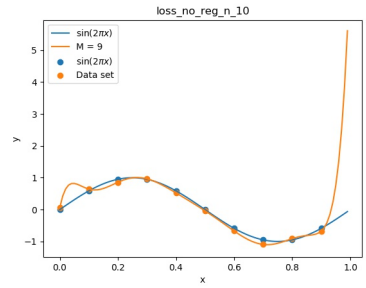


(g) 九阶多项式

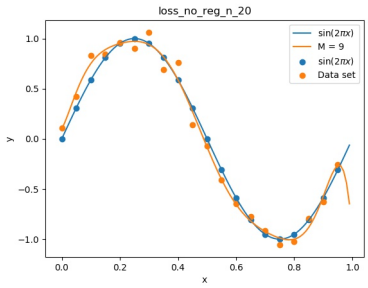


(h) E_{RMS} 随着 m 的增加折线图

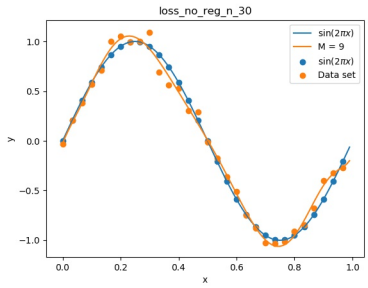
图 4.3: 解析解求不带有正则项 loss 函数的不同多项式系数模型输出结果



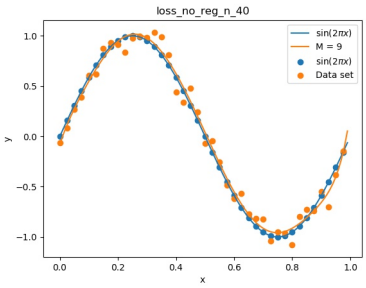
(a) 10 个数据点



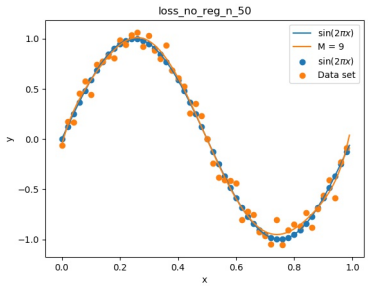
(b) 20 个数据点



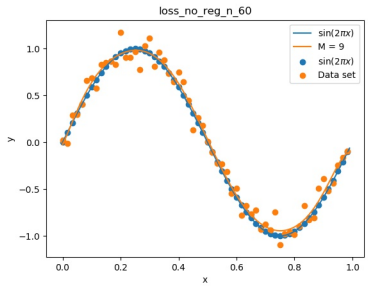
(c) 30 个数据点



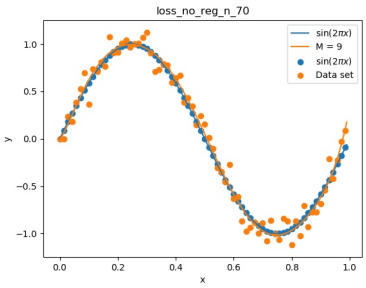
(d) 40 个数据点



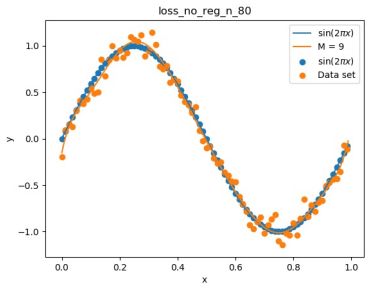
(e) 50 个数据点



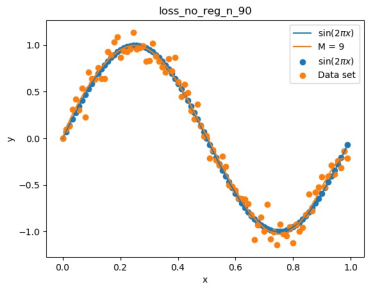
(f) 60 个数据点



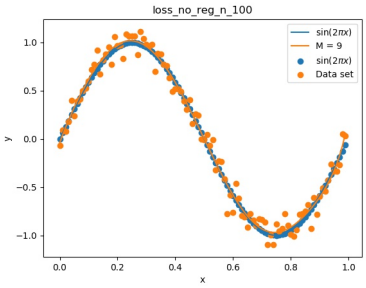
(g) 70 个数据点



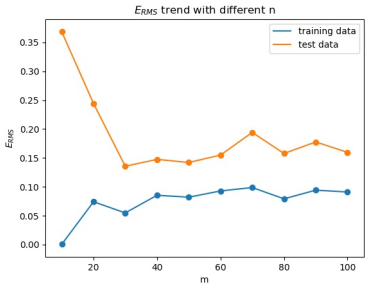
(h) 80 个数据点



(i) 90 个数据点



(j) 100 个数据点



(k) E_{RMS}

图 4.4: 解析解求不带有 loss 函数的改变数据大小模型的输出

4.2 使用解析解求解带有正则项的 loss 函数

4.2.1 求解最优解

在上一小节中我们得到添加数据量可以减少过拟合的发生，但是在实际中获取数据常常是一个费时费力的过程，于是我们需要想一些其他的方法来减少过拟合的发生，添加一个正则项就是解决方法之一，在本实验中经过一番尝试，我认为，在 $\lambda = e^{-18}$ 的时候可以获得较好的结果如图4.6所示。该结果在训练集上的 E_{RMS} 为 0.017，在相互独立的测试集上的结果为 0.159

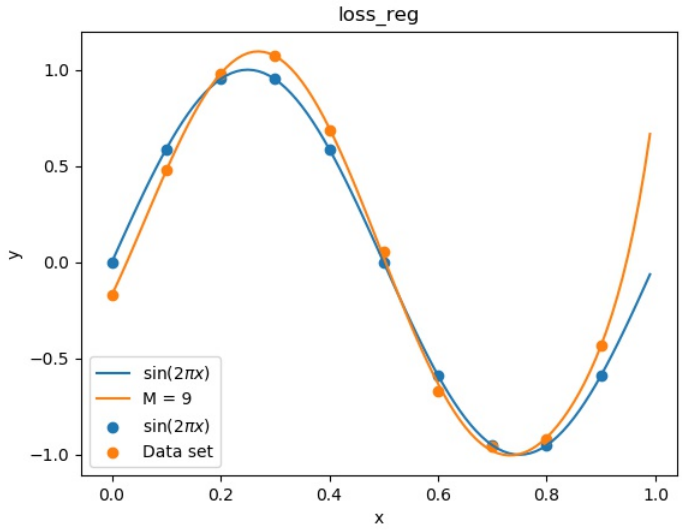


图 4.5: 解析解求带有正则项的 loss 函数 $\lambda = e^{-18}$ 时候得到的结果

4.2.2 改变超参数的模型输出

为了完成这一小节的内容，我将超参数 λ 的值从 e^{-8} 逐渐增大到 e^{0} 得到了如图4.6所示各个结果，从这些结果可以看出，随着 λ 的值逐渐增加，该模型的拟合程度越来越差，相应的 E_{RMS} 无论在训练集或者测试集上都越来越大。

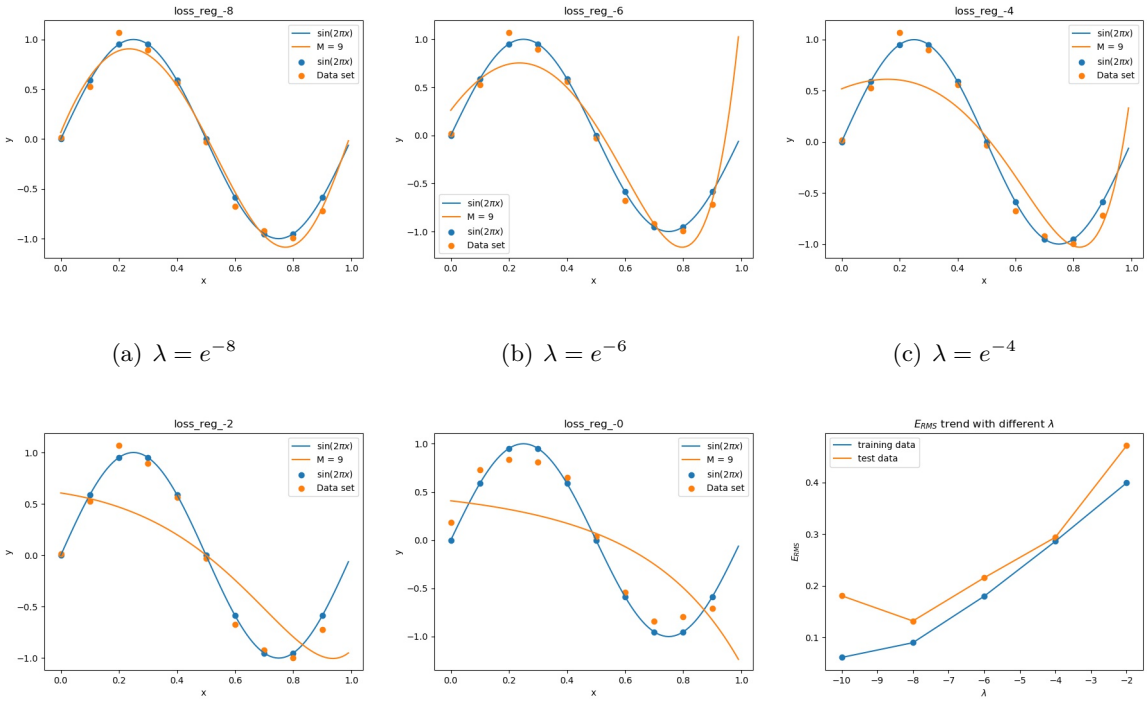
4.3 使用梯度下降法求解带有正则项的 loss 函数

4.3.1 求解最优解

经过一番尝试，我发现，当步长设置在 0.1，λ 设置在 e^{-20} 并且训练 20 万次的时候，可以得到如图4.7所示的较好结果，此时训练集上的 E_{RMS} 是 0.07，在测试集上的 E_{RMS} 是 0.12.

4.3.2 改变超参数的模型输出

为了完成本小节的内容，我将步长 α 从 0.8 逐渐下降到 0. 分别得到了如图4.8的结果, 通过这一组图片可以看出，当步长逐渐变长的时候，在同一组数据集下，它的收敛速度变快了，同时，在该训练集上的 E_{RMS} 也有所下降，但是在测试集上的 E_{RMS} 可能会有所上升。



(a) $\lambda = e^{-8}$ (b) $\lambda = e^{-6}$ (c) $\lambda = e^{-4}$

(d) $\lambda = e^{-2}$ (e) $\lambda = e^{0}$ (f) E_{RMS} 随着 λ 的变化而变化

图 4.6: 解析解带有正则项的 loss 函数改变超参数模型的输出

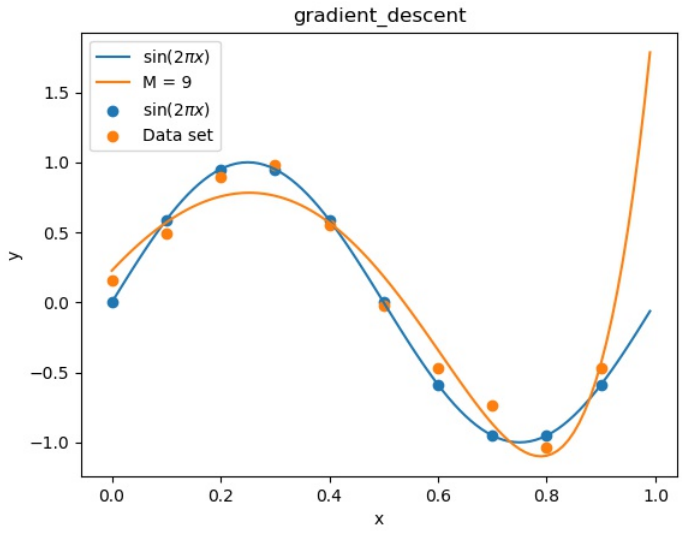
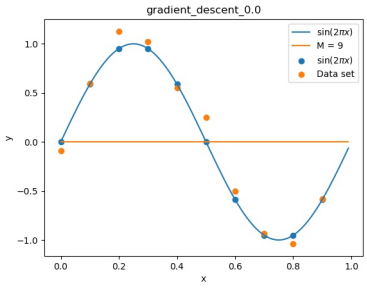
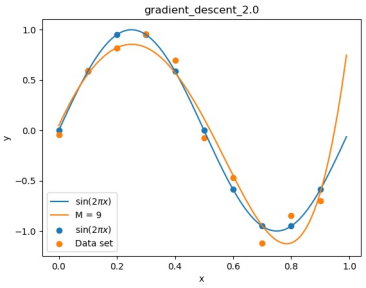


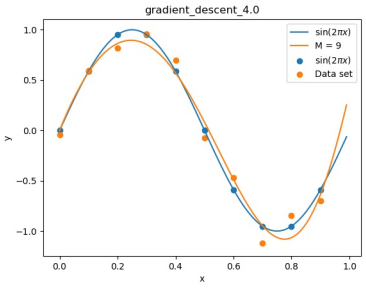
图 4.7: 梯度下降步长 $0.1\lambda = e^{-20}$ 在循环 20 万次后的模型输出



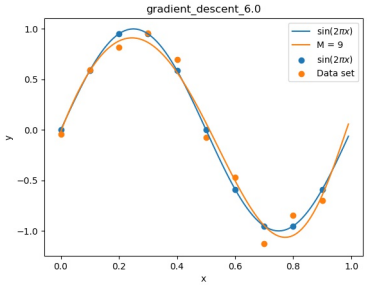
(a) $\alpha = 0$



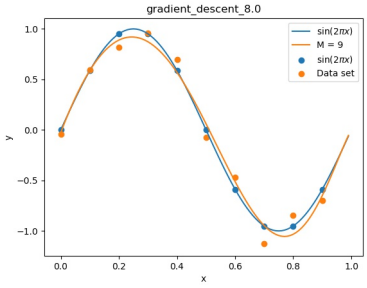
(b) $\alpha = 0.02$



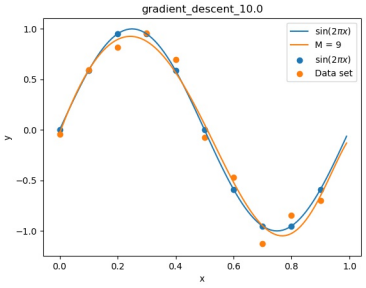
(c) $\alpha = 0.04$



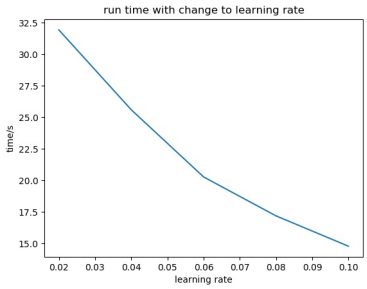
(d) $\alpha = 0.06$



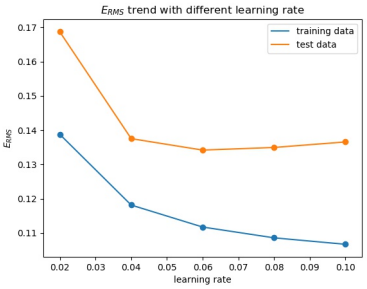
(e) $\alpha = 0.08$



(f) $\alpha = 0.08$

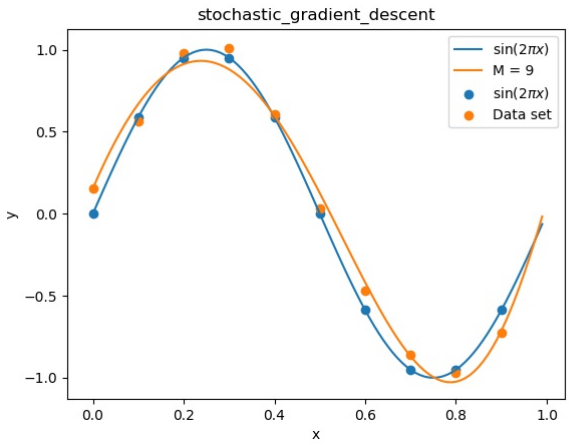


(g) 改变 α 收敛时间变化

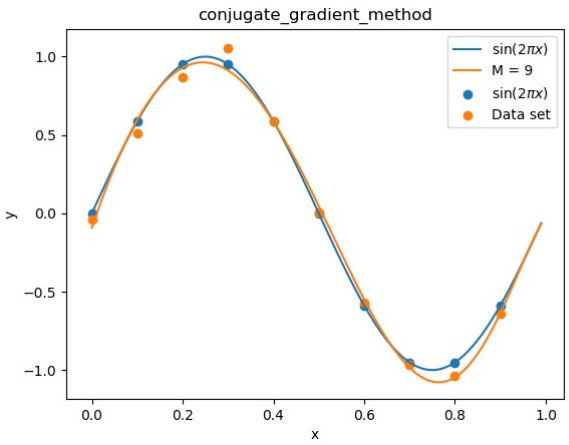


(h) $\alpha = 0.08$

图 4.8: 梯度下降改变 α 模型对应的输出和性能分析



(a) 随机梯度下降法循环 172731 次的模型输出



(b) 共轭梯度法

图 4.9: 随机梯度下降法和共轭梯度法

4.4 随机梯度下降法求解带有正则项的 loss 函数

4.4.1 求解最优解

在本小节中，我们使用随机梯度下降法，对该问题进行求解，得到的结果如图4.9(a)所示，在训练集上的 E_{RMS} 为 0.07，在测试集上的 E_{RMS} 为 0.15，

4.5 共轭梯度法求解带有正则项的 loss 函数

4.5.1 求解最优解

本小节中，我们使用共轭梯度法求解带有正则项的 loss 函数，得到的结果4.9(b)如图所示，在训练集上的 E_{RMS} 为 0.03，在测试集上的 E_{RMS} 为 0.10

第 5 章 结论

在本文中，我们尝试了解析解、梯度下降、随机梯度下降、共轭梯度共四种方法，体会了不同的数据集大小，不同的超参数，不同的多项式阶数对机器学习模型的影响，熟悉了 python 语言中和矩阵运算的相关模块，加深了对于多项式回归这一机器学习基本模型的理解和实现能力。

第 6 章 参考文献

'Pattern recognition and machine learning' Christopher M. Bishop
共轭梯度法英文维基百科 https://en.wikipedia.org/wiki/Conjugate_gradient_method
梯度下降法英文维基百科 https://en.wikipedia.org/wiki/Gradient_descent
随机梯度下降法英文维基百科 https://en.wikipedia.org/wiki/Stochastic_gradient_descent

斯坦福机器学习课程讲义吴恩达 http://www.andrewng.org

A 源代码