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

实验项目名称：多项式拟合曲线

实验题目：多项式拟合曲线

班级：1503104
学号：1153710506
姓名：薛仲豪

设计成绩	报告成绩	指导老师

一、实验目的

通过实验，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

人工合成一组 sin(2πx)的数据，作为训练集合。N=10，x 在[0,1]上均匀分布，对每一个目标值 t 加一个均值为 0 的高斯噪声。
使用不同的方法，用多项式拟合生成的数据。观察和分析不同方法，不同阶数对结果的影响。
理解过拟合现象，并尝试各种克服过拟合的方法。
加入正则项后进行实验，调试正则项的参数，观察正则项的影响。
不允许调用 matlab 的 curfit 函数，不允许调用 matlab 的优化函数，自己写优化函数。
实验在 matlab 的环境进行，具体配置如下：

OS 名称: Microsoft Windows 10 家庭中文版
OS 版本: 10.0.15063 暂缺 Build 15063
处理器: 安装了 1 个处理器。
[01]: Intel64 Family 6 Model 60 Stepping 3 GenuineIntel ~2601 Mhz
物理内存总量: 16,269 MB
虚拟内存: 最大值: 19,198 MB

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 算法原理

本次实验是使用多项式曲线拟合正态曲线，假设给定 N 个观测数据的训练集合

$$\mathrm{X} \equiv (x_1, \dots, x_n)$$

$$\mathrm{T} \equiv (t_1, \dots, t_n)$$

用向量 W 来表示一个多项式$y(x, W) = w_0 + w_m x + \cdots + w_m x^m$，我们的目标就是使得 W 表示的多项式更好的拟合原本的正态曲线。为了描述拟合的好坏程度，建立误差函数$E(W) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, W) - t_n\}^2$。误差函数的值越小，说明拟合效果越好。这样，我们的目标就转化为了一个优化问题

$$\underset{W}{\operatorname{argmin}}\{E(W)\}$$

本次实验就是使用不同的方法来解决这个优化问题。

最小二乘法

一个显然的方法就是最小二乘法，最小二乘法（又称最小平方法）是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。
因为 E 是 W 的二次函数，求 E 的极值点，实际就是找 E 对 W 的梯度为 0 的点。
首先，求导

$$
\begin{aligned}
\nabla_w \|Xw-y\|^2 &= \nabla_x (Xw-y)^T(Xw-y) \\
&= \nabla_x (w^T X^T Xw - y^T Xw - w^T X^T y + y^T y) \\
&= \nabla_x (w^T X^T Xw) - 2\nabla_x (y^T Xw) + \nabla_x (y^T y) \\
&= 2X^T Xw - 2X^T y + 0 \\
&= 2X^T (Xw-y)
\end{aligned}
$$

令导数为 0，即可解出 W

$$
\begin{aligned}
2X^T(Xw-y) &= 0 \\
X^T Xw &= y \\
w &= (X^T X)^{-1} y
\end{aligned}
$$

另外，后面会使用带有正则项的误差函数，这里提前给出其梯度和最小二乘法的推导，后面不再重复推导。

$$
\begin{aligned}
\nabla_w (\|Xw-y\|^2 + \lambda\|w\|^2) &= \nabla_w ((Xw-y)^T(Xw-y) + \lambda w^T w) \\
&= \nabla_w (w^T X^T Xw - y^T Xw - w^T X^T y + y^T y + \lambda w^T w) \\
&= \nabla_w (w^T X^T Xw) - 2\nabla_w (y^T Xw) + \nabla_w (y^T y) + \nabla_w (\lambda w^T w) \\
&= 2X^T Xw - 2X^T y + 0 + 2\lambda w \\
&= 2X^T (Xw-y) + 2\lambda w
\end{aligned}
$$

最小二乘法的结果为

$$
w = (X^T X + \lambda I)^{-1} y
$$

梯度下降法

梯度下降法也是一个最优化算法，通常也称为最速下降法。最速下降法是求解无约束优化问题最简单和最古老的方法之一，虽然现在已经不具有实用性，但是许多有效算法都是以它为基础进行改进和修正而得到的。最速下降法是用负梯度方向为搜索方向的，最速下降法越接近目标值，步长越小，前进越慢。

算法步骤如下：

第 1 步：给定初始数据，起始点$x^{(0)}$，给定终止误差$\varepsilon > 0$，令$k \coloneqq 0$.

第 2 步：求梯度向量模的值：$\left\|\nabla f(x^{(k)})\right\|$

若$\left\|\nabla f(x^{(k)})\right\| < \varepsilon$，停止计算，输出$x^{(k)}$，作为极小点的近似值，否则转下一步

第 3 步：构造负梯度方向：

$$p^{(k)} = -\nabla f(x^{(k)})$$

第 4 步：进行一维搜索。无论用哪种方法求得λ_k后，令

$$x^{(k+1)} = x^{(k)} + \lambda_k p^{(k)} = x^{(k)} - \lambda_k \nabla f\left(x^{(k)}\right)$$

置$k \coloneqq k + 1$，转第 2 步。

共轭梯度法

共轭梯度法是介于最速下降法与牛顿法之间的一个方法，它仅需利用一阶导数信息，但克服了最速下降法收敛慢的缺点，又避免了牛顿法需要存储和计算 Hesse 矩阵并求逆的缺点，共轭梯度法不仅是解决大型线性方程组最有用的方法之一，也是解大型非线性最优化最有效的算法之一。 在各种优化算法中，共轭梯度法是非常重要的一种。其优点是所需存储量小，具有步收敛性，稳定性高，而且不需要任何外来参数。

算法步骤如下：

第 1 步：选取初始数据

选取初始点$x^{(0)}$，给出终止误差$\varepsilon > 0$

第 2 步：求初始梯度

计算$\nabla f(x^{(k)})$，若$\left\|\nabla f(x^{(k)})\right\| \le \varepsilon$，停止迭代，输出$x^{(0)}$，否则转第 3 步

第 3 步：构造初始搜索方向

令

$$p^{(0)} = -\nabla f(x^{(0)})$$

令$k \coloneqq 0$，进行第 4 步

第 4 步：进行一维搜索，求λ_k，使

$$f\left(x^{(k)} + \lambda_k p^{(k)}\right) = \min_{\lambda \ge 0}(x^{(k)} + \lambda_k p^{(k)})$$

令$x^{(k+1)} = x^{(k)} + \lambda_k p^{(k)}$。转第 5 步。

第 5 步：求梯度向量

计算$\nabla f(x^{(k)})$，若$\left\|\nabla f(x^{(k)})\right\| \le \varepsilon$，停止迭代，输出$x^*$的近似值，$x^* \approx x^{(k+1)}$，否则进行第 6 步.

第 6 步：检验迭代步数

若$k + 1 = n$，令$x^{(0)} \coloneqq x^{(n)}$，转第 3 步。否则进行第 7 步。

第 7 步：构造搜索方向，用 F-R 公式，取

$$p^{(k+1)} = -\nabla f\left(x^{(k+1)}\right) + \beta_k p^{(k)}$$

$$\beta_k = \frac{\left\|\nabla f\left(x^{(k+1)}\right)\right\|^2}{\|\nabla f(x^{(k)})\|^2}$$

令$k := k + 1$，转第 4 步。

2. 算法的实现

最小二乘法

最小二乘法的实现非常简单，因为推倒的结果就是矩阵的形式表示的，而matlab可以直接进行矩阵计算，因此只需要实现一个函数

```
function [ W ] = Normal( X, y )
```

X 是数据，y 是目标（标签）。W 是输出的系数，用来确定多项式。
根据推到结果，该函数应该输出

$$W = (X^T X)^{-1} X^T y$$

写成 matlab 的形式就是

```
W = pinv(X'*X)*X'*y;
```

其中 pinv 函数是伪逆，较 inv 函数更加高效，精确。

梯度下降法

梯度下降算法是一个迭代的过程。
首先，利用随机函数生成一个 W，作为初始值。
接着，求出在这一点上误差函数的梯度，将这一梯度乘步长（学习速率）后加在 W 上，就得到了新的 W。
主要的迭代过程使用 while 循环实现，循环终止的条件是前两次迭代得到的 W 的插值小于终止误差。循环的主体和前面的步骤一样，首先计算在当前 W 上误差函数的梯度，将这一梯度乘步长后加在 W 上，得到新的 W。（具体的算式在之前原理中）
循环结束后，所得的 W 就是最终的结果。

共轭梯度法

共轭梯度法也是一个迭代的过程，虽然其背后的数学原理比较复杂，但实际的操作过程和梯度下降相差不大，主要的区别就是在梯度下降法中，直接使用梯度作为 W 前进的方向；而在共轭梯度法内，梯度不等价于方向，也就是说 W 并不是沿着梯度的方向前进的。
首先，利用随机函数生成一个 W，作为初始值。
接着，求出在这一点上误差函数的梯度，将这一梯度乘步长（学习速率）后加在 W 上，就得到了新的 W。
主要的迭代过程使用 while 循环实现，循环终止的条件是前两次迭代得到的 W 的插值小于终止误差。（至此，和梯度下降完全一样）循环的主体内，首先计算当前 W 上误差函数的梯度，之后根据这个梯度和前一次的梯度计算出β，并根据当前梯度和β得到实际 W 更新的方向，将这个方向乘上步长后加在 W 上，就得到了新的 W。（具体的算式在之前原理中）
循环结束后，所得的 W 就是最终的结果。

四、测试结果

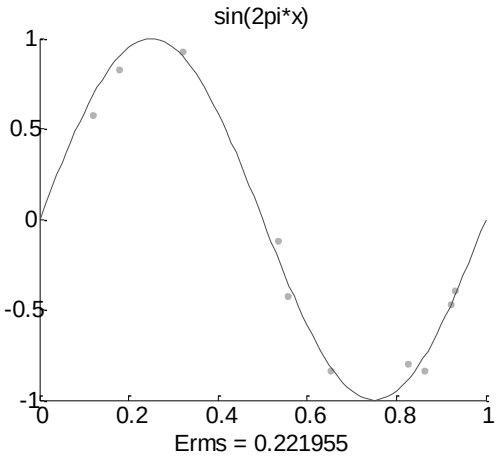
实验数据：

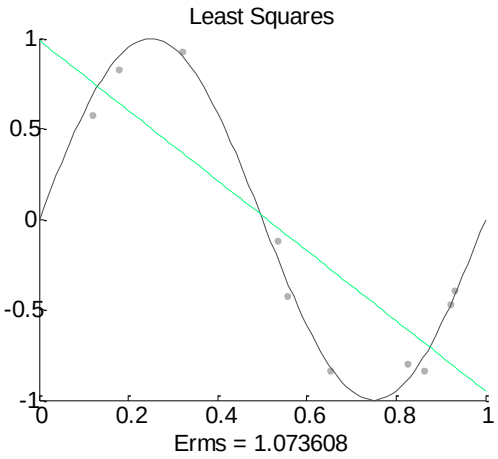
x	y
0.118908	0.579858
0.650181	-0.83239
0.1789	0.828732
0.919854	-0.47045
0.555617	-0.42248
0.929962	-0.39529
0.321446	0.926173

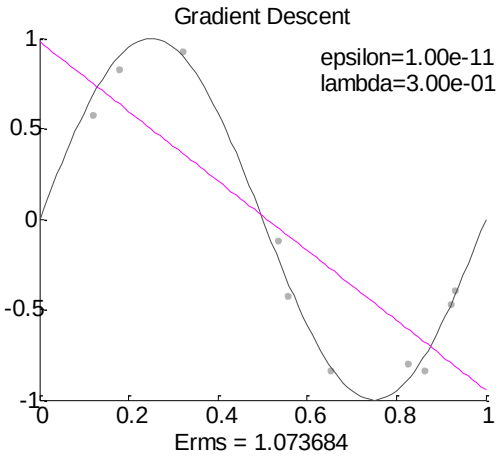
每一次实验结果都用六个图来表示，这六个图分别代表原本的正弦曲线、最小二乘法得到的曲线，梯度下降法得到的曲线、梯度下降法方均差随迭代次数的变化、共轭梯度法得到的曲线、共轭梯度法方均差随迭代次数的变化。这些算法的具体参数在图中体现。

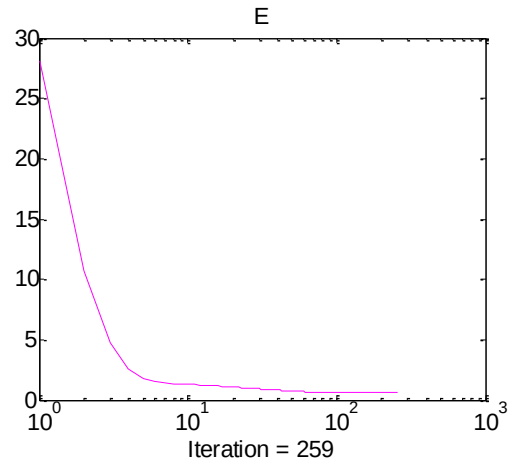
首先，对不同的阶数进行实验：

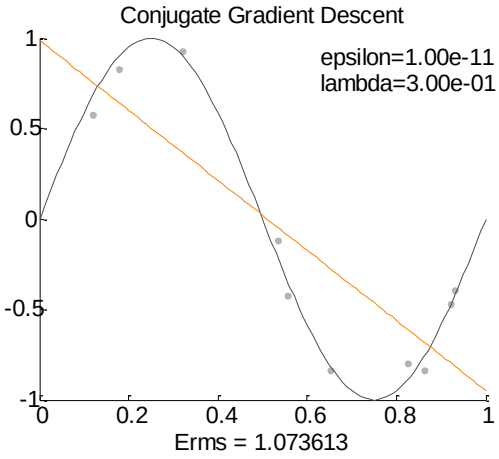
一阶

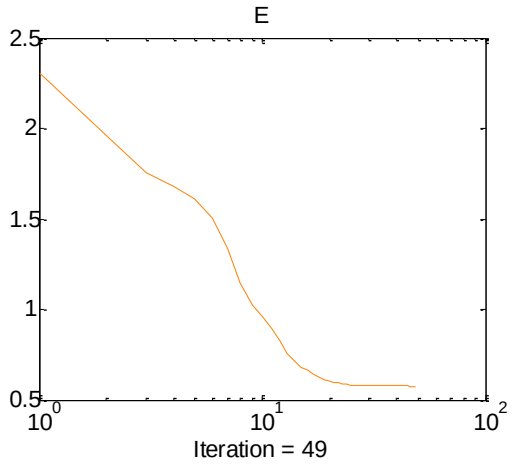

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 1.073608

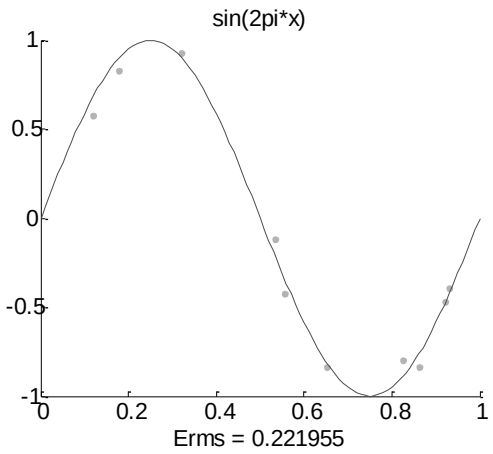

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 1.073684

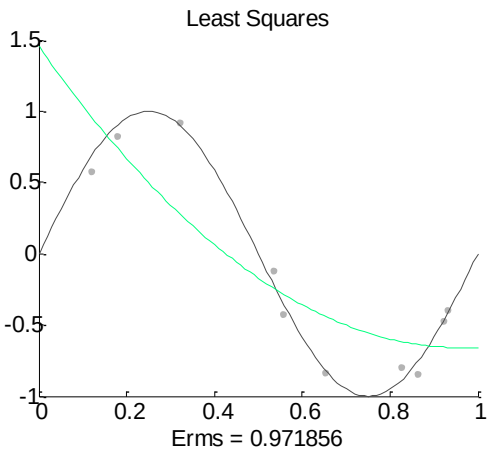

E
Iteration = 259

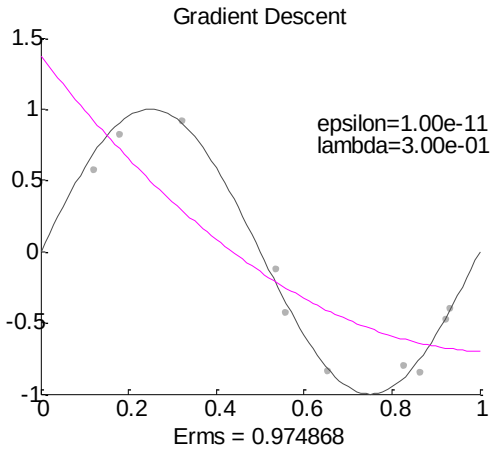

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 1.073613

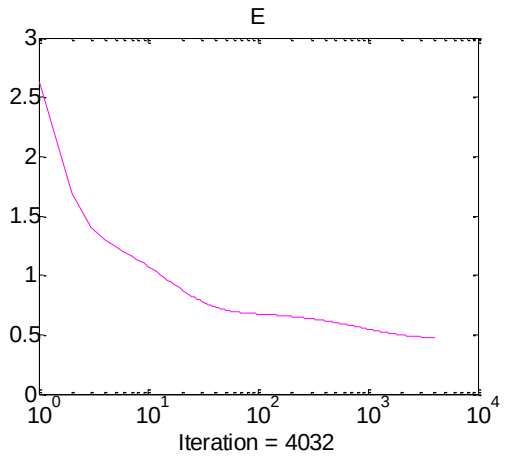

E
Iteration = 49

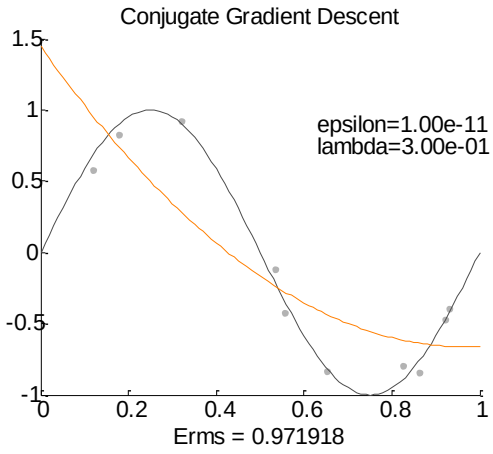
二阶

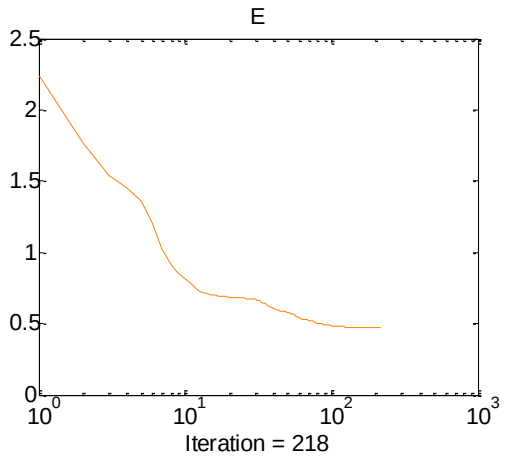

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.971856

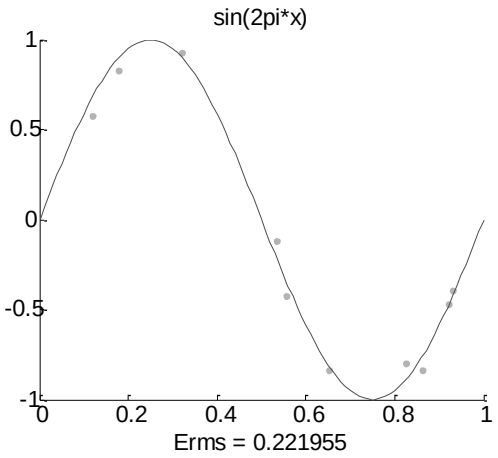

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.974868

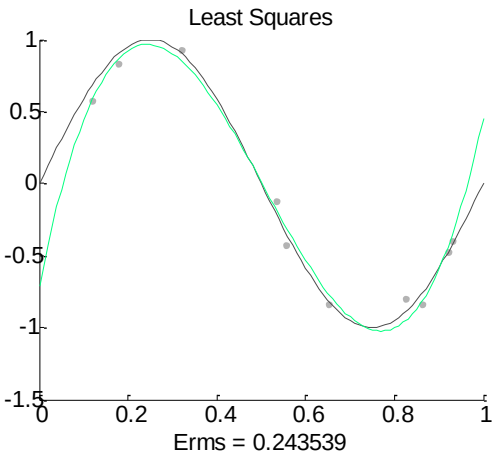

E
Iteration = 4032

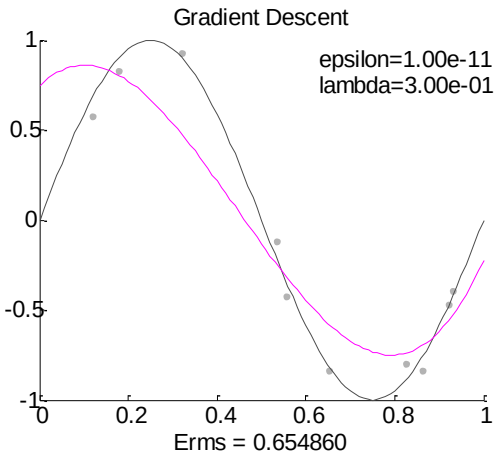

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.971918

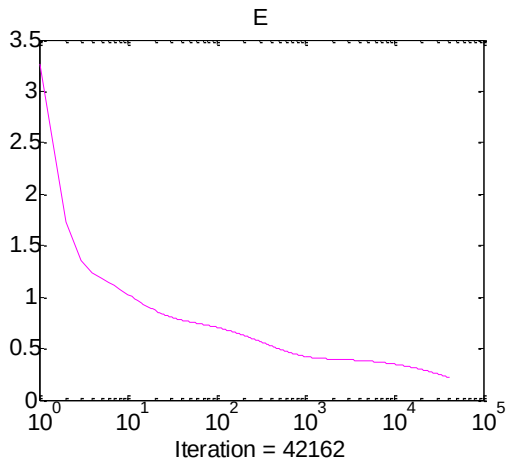

E
Iteration = 218

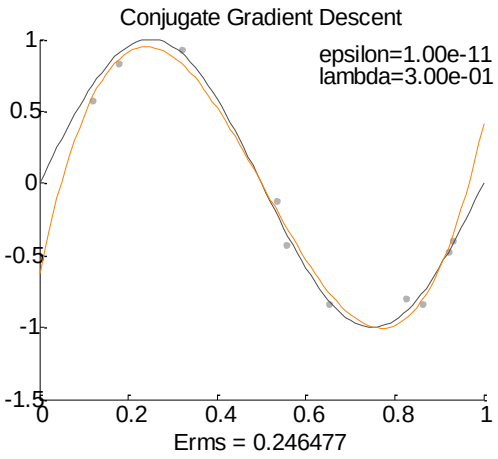
三阶

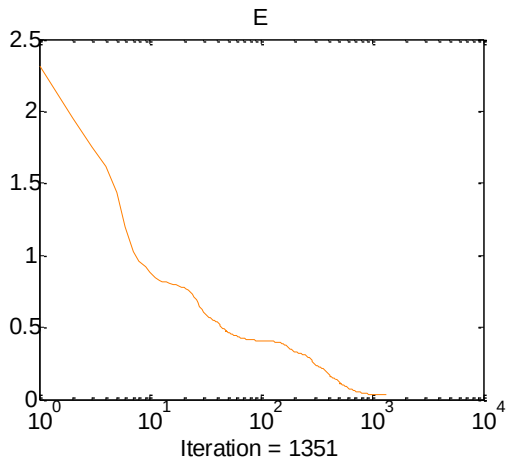

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.243539

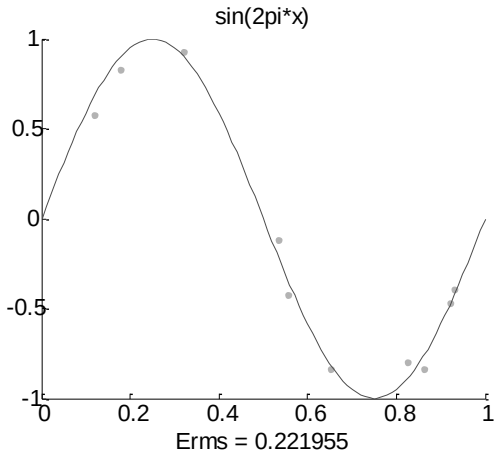

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.654860

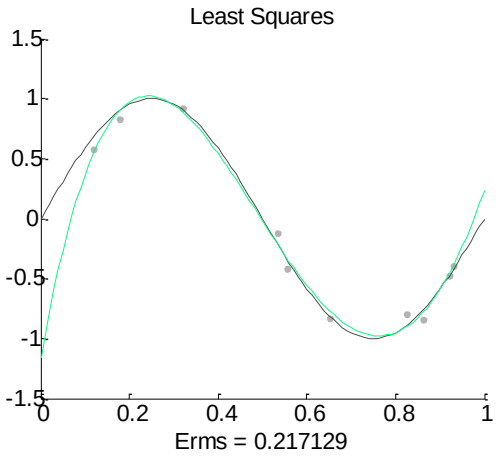

E
Iteration = 42162

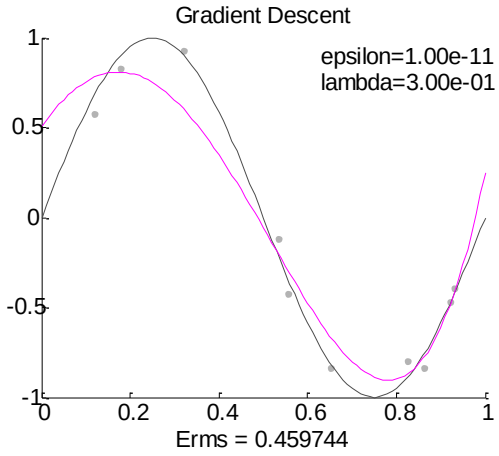

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.246477

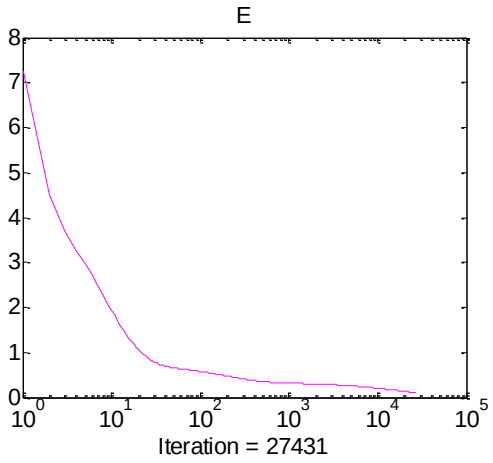

E
Iteration = 1351

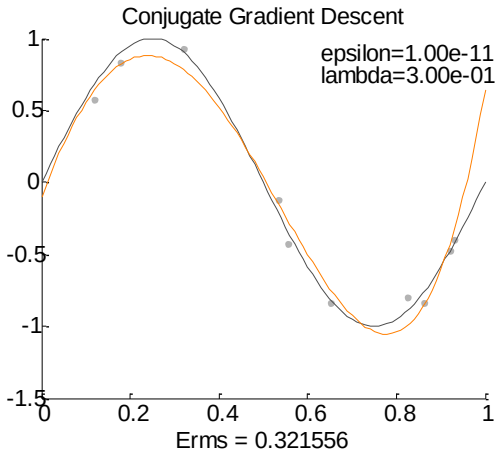
四阶

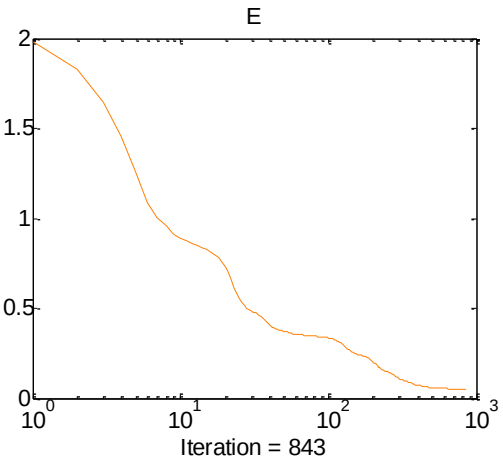

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.217129

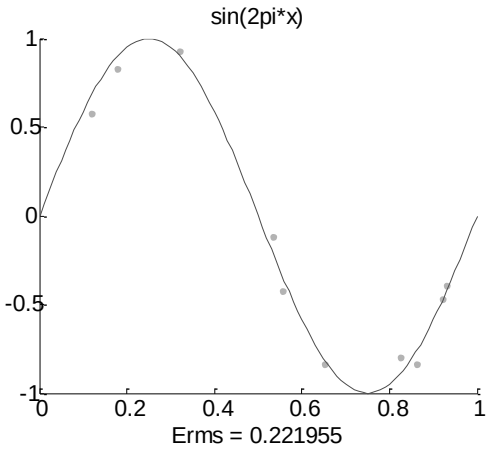

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.459744

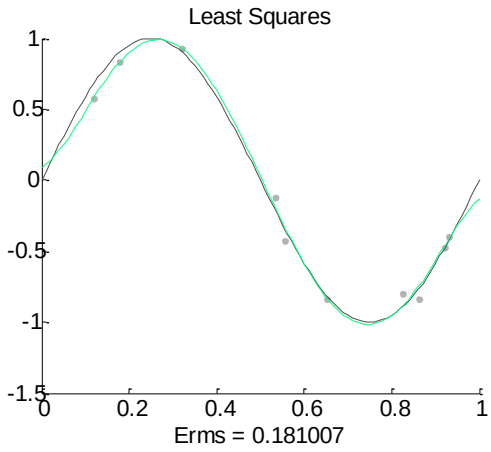

E
Iteration = 27431

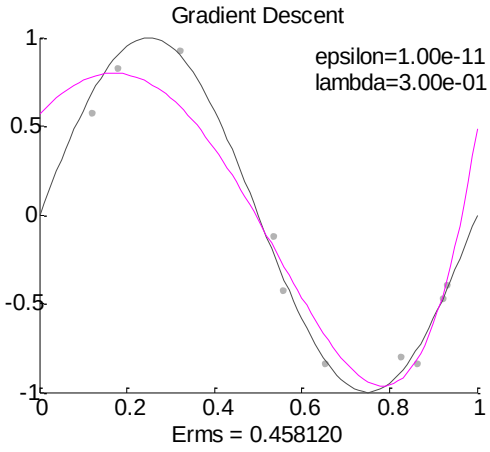

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.321556

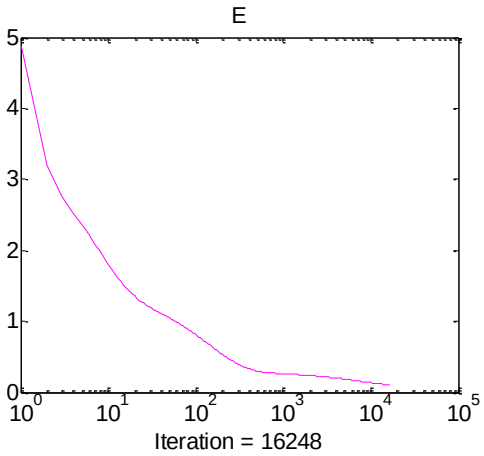

E
Iteration = 843

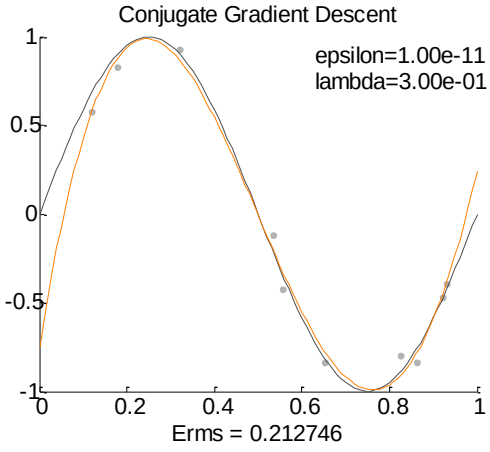
五阶

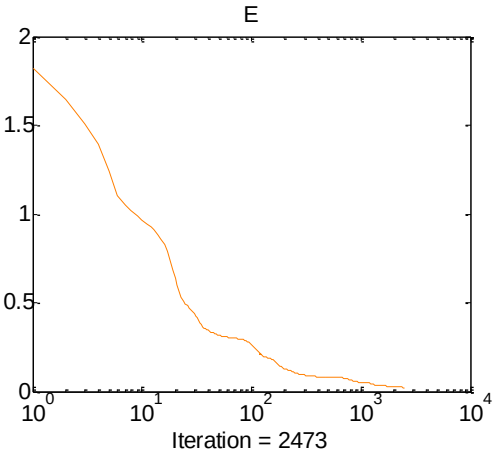

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.181007

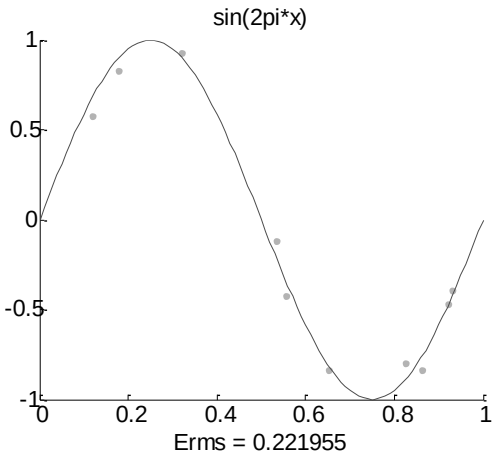

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.458120

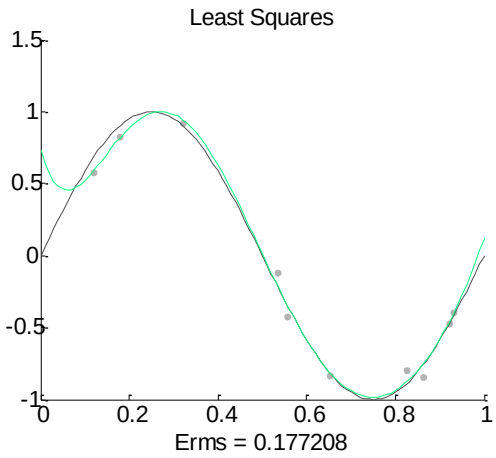

E
Iteration = 16248

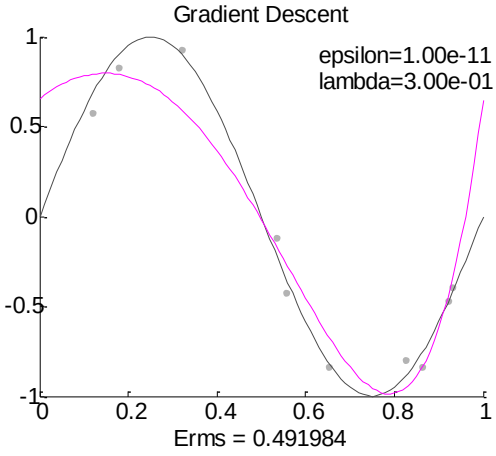

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.212746

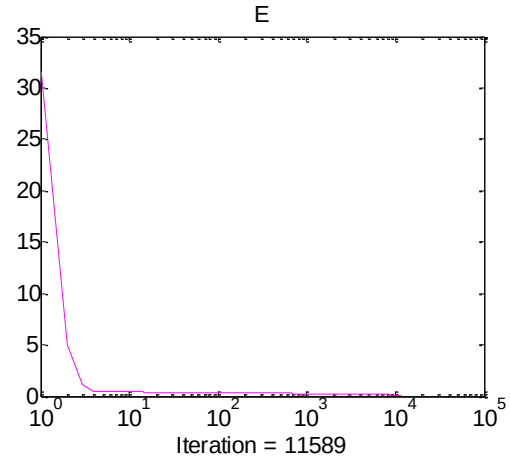

E
Iteration = 2473

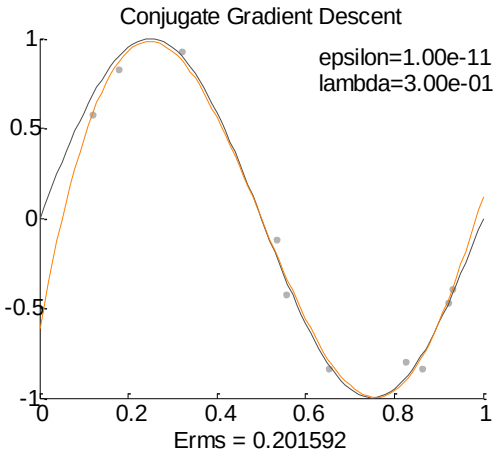
六阶

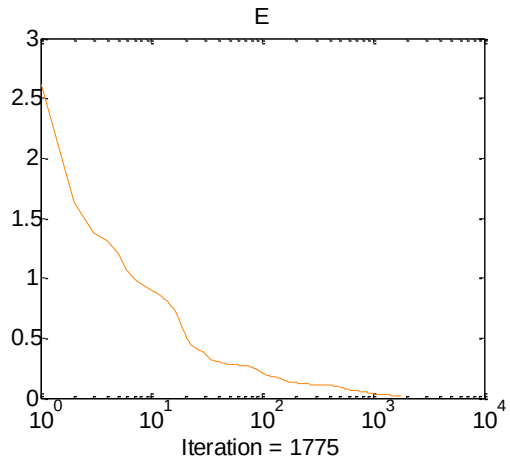

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.177208

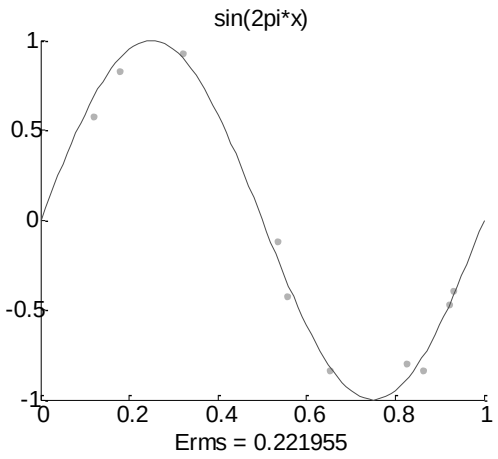

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.491984

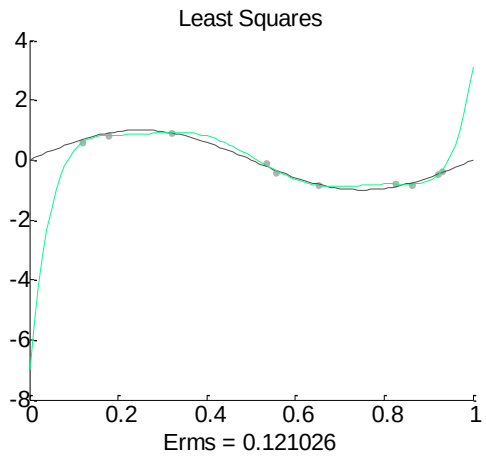

E
Iteration = 11589

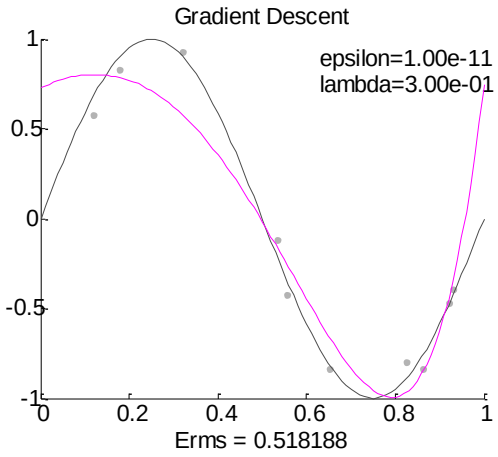

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.201592

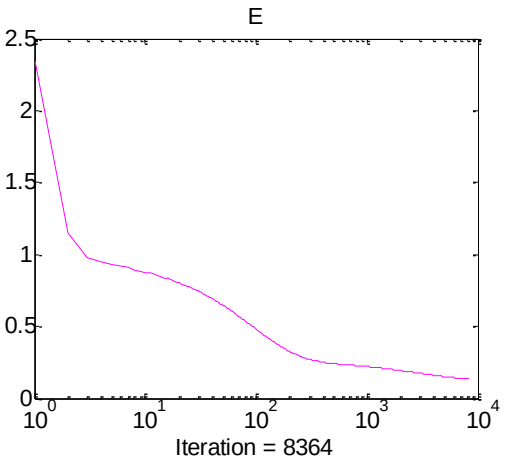

E
Iteration = 1775

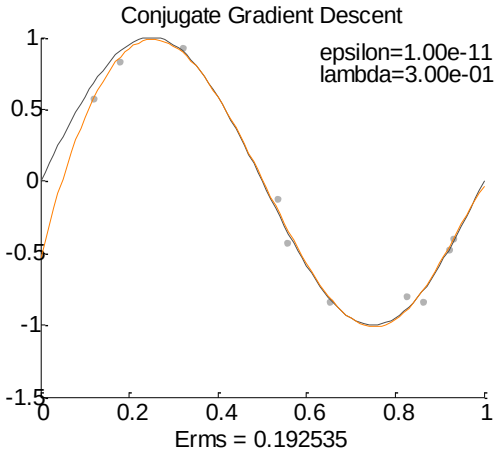
七阶

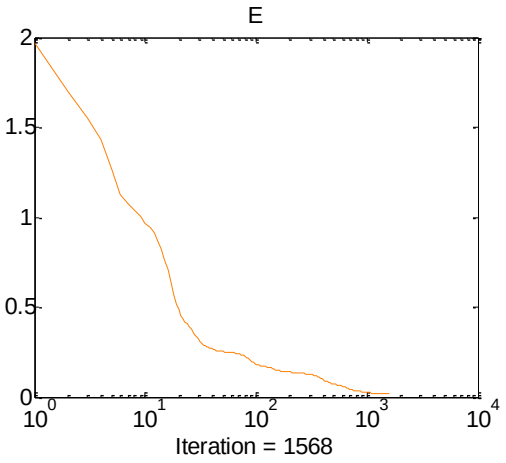

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.121026

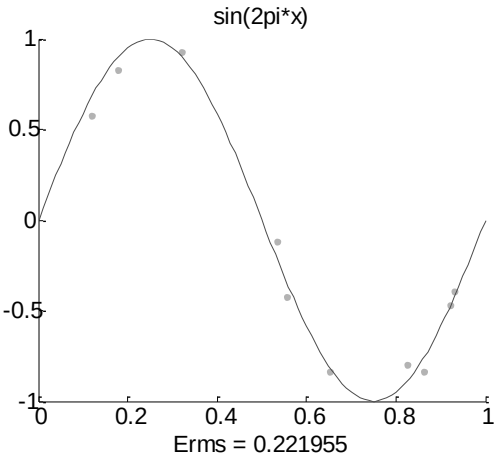

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.518188

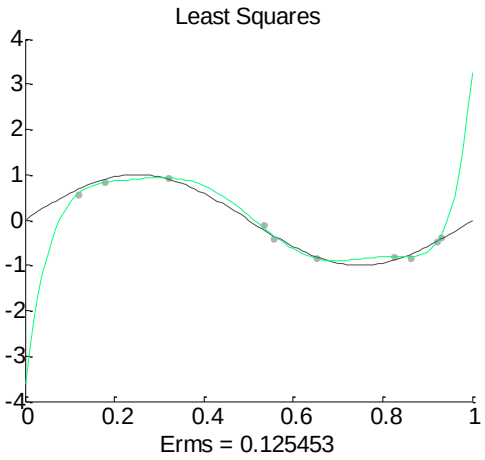

E
Iteration = 8364

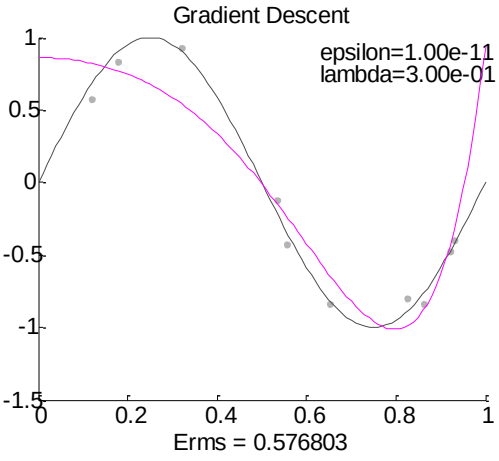

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.192535

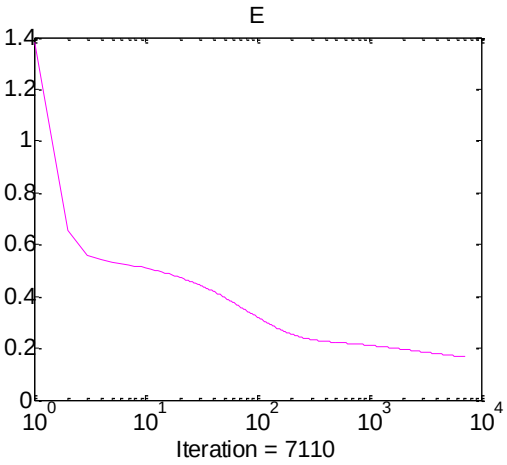

E
Iteration = 1568

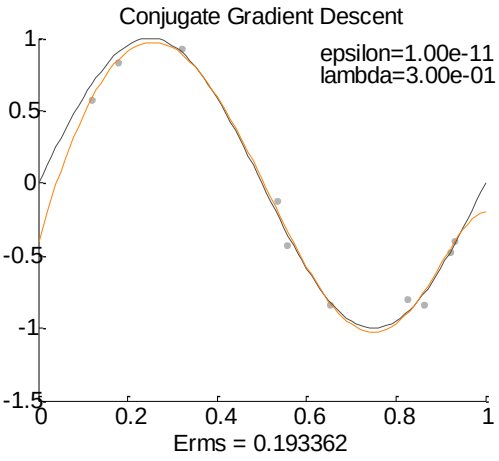
八阶

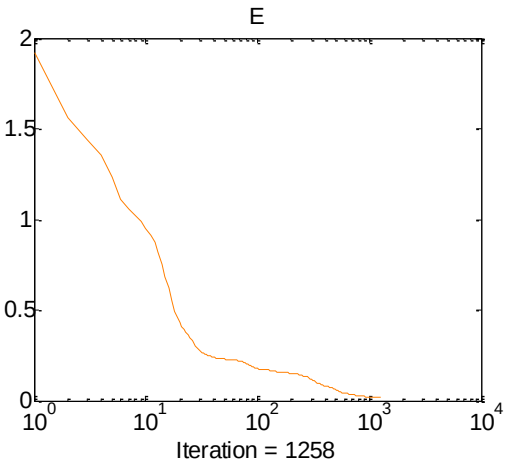

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.125453

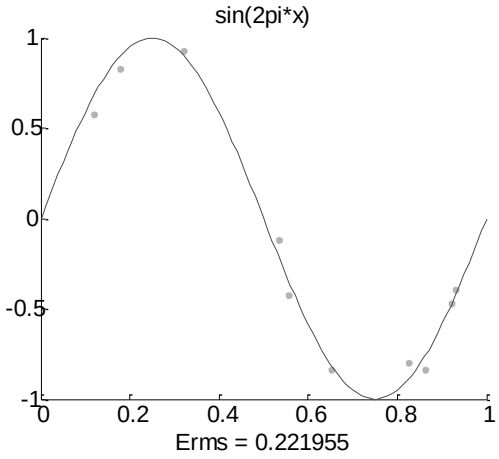

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.576803

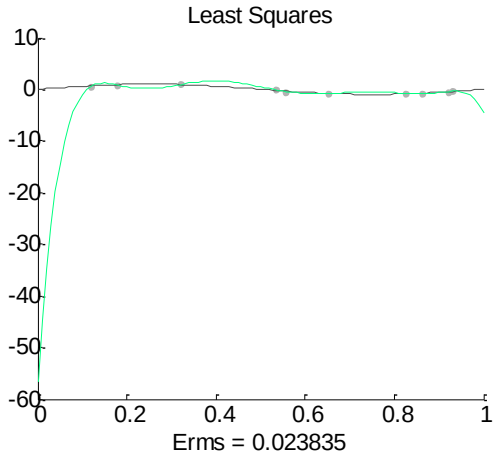

E
Iteration = 7110

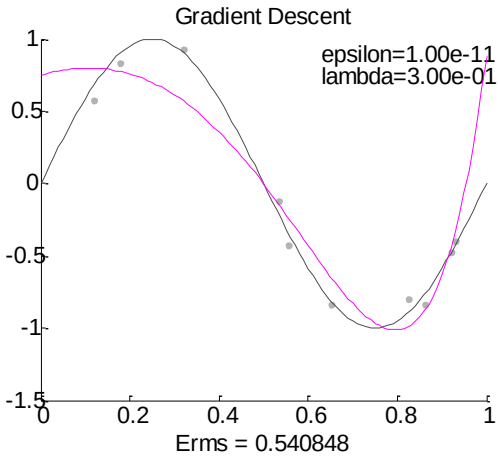

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.193362

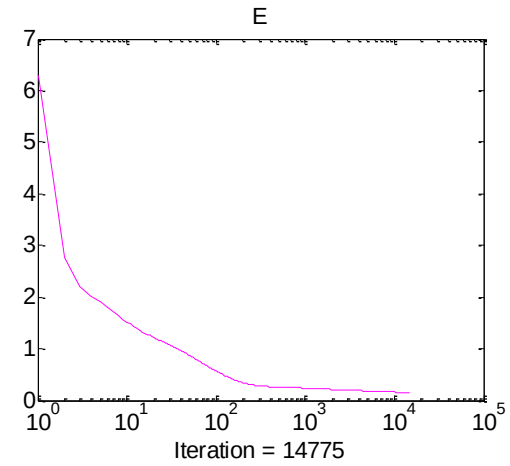

E
Iteration = 1258

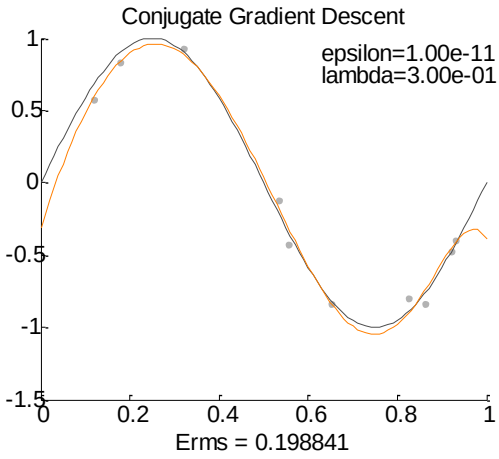
九阶

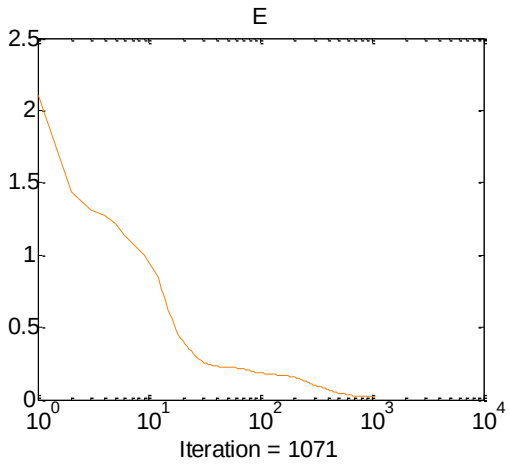

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.023835


Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.540848

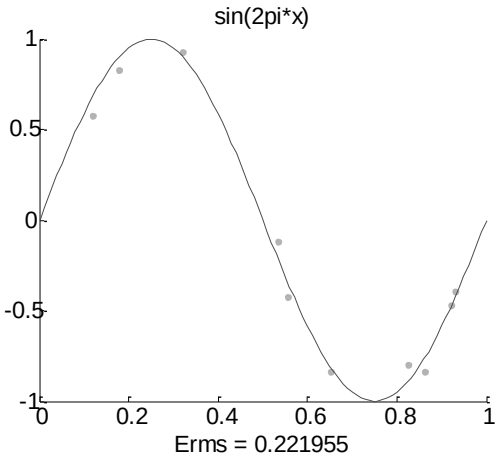

E
Iteration = 14775

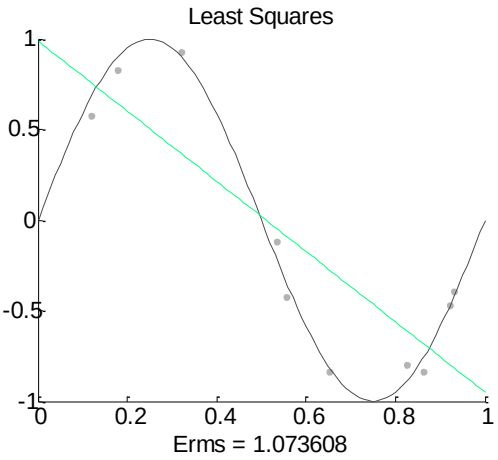

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.198841

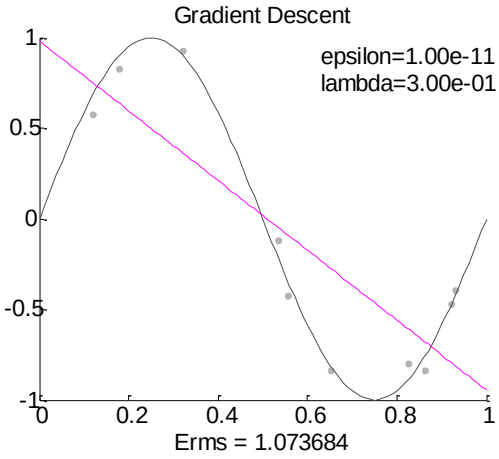

E
Iteration = 1071

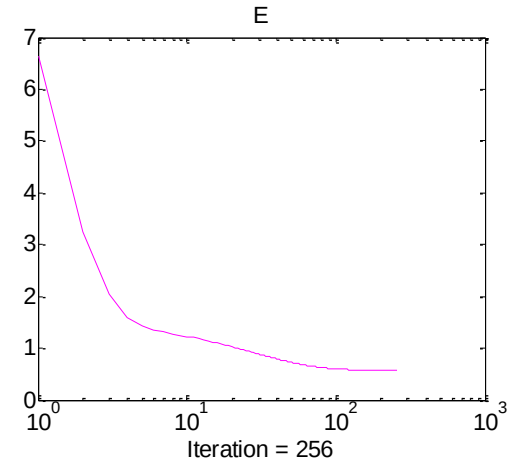
首先，比较不同的算法。
最小二乘法最终的均方差永远小于梯度下降法和共轭梯度法，这是显然的。因为最小二乘法是用数学的方法直接计算得出的最优解，其对应的均方差一定是最小的。（但对应的曲线未必是最优的，后文会提到）
梯度下降法能够在一定程度上拟合曲线，但在三种方法中效果较差，耗时（迭代次数）也较多，虽然其思想十分经典，实现也非常简单，但不是一种十分推荐的算法。
共轭梯度法明显迭代次数少于梯度下降法，均方差也相对较小，而且拟合效果也一直都不差，可以说是这三个算法中最优的，是一个比较推荐的算法。
其次，分析阶数对实验结果的影响。
由实验的结果可以看出，当阶数比较小时，多项式曲线缺乏变化，模型的表达能力有限，没有能力拟合正弦函数曲线，比如一阶曲线只能是一条直线，而二阶曲线是一个凸函数，都不可能拟合正弦函数。
从三阶开始，拟合情况开始好转，多项式曲线比较接近正弦曲线，均方差也相对减小，这是期望中想要的到的结果。
阶数继续增高，从七阶开始，最小二乘法得到的曲线已经开始出现异常，而到了九阶最为严重。不难发现，九阶的曲线可以拟合任意十个横坐标不等的点，因此，最小二乘法在数学上完全拟合了这十个点，但由于计算的精度有限，实际上仍然有一定误差，但明显小于之前的误差。至于另外的梯度下降法和共轭梯度法，他们没能完美的拟合十个点，主要原因是有停止条件的存在，迭代在完全拟合之前就停止了。
另外需要提到的是，随着阶数的增高，Erms 是呈不断减小的趋势，这是因为阶数越高的曲线表达能力越强，拟合的能力也越强，知道 N-1 阶时，可以完全拟合。Erms 在不断减小，但看起来拟合曲线却在阶数过高时偏离了原本的正弦曲线，这就是过拟合现象。
如果将原本的曲线当作测试集，拟合用的数据当作训练集，过拟合指的就是随着训练模型越来越复杂，模型对于训练集的描述（拟合）越来越好，但对于测试集的描述（拟合）却开始慢慢变差。
克服过拟合的方法有多种，首先采用增加惩罚项的方式进行同样的实验（取$\lambda = 0.0000001$）。

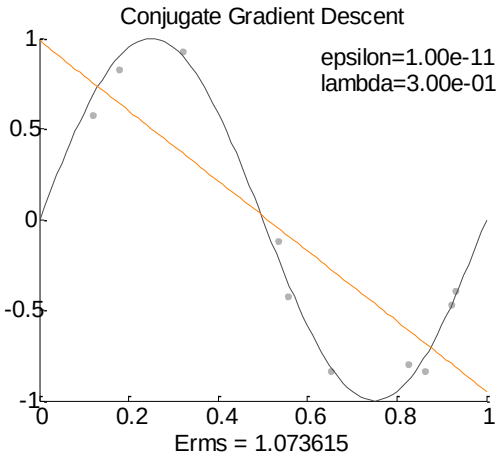
一阶

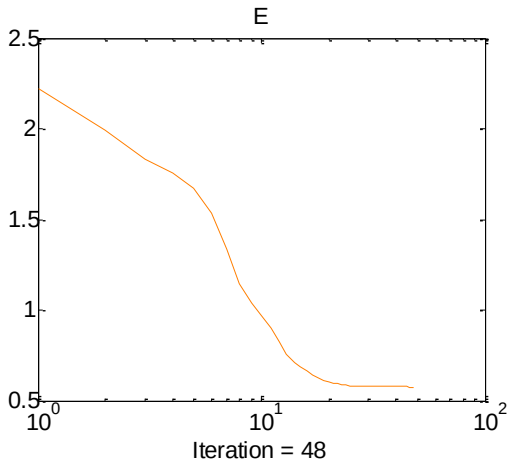

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 1.073608

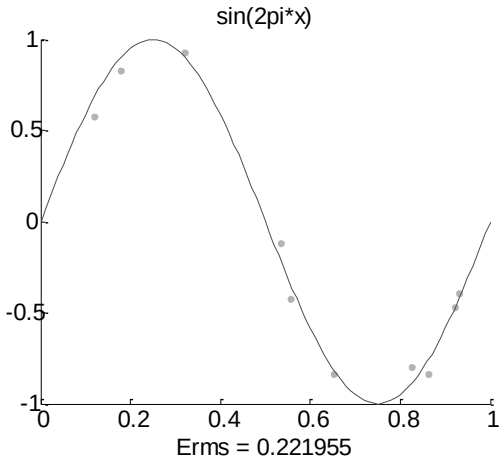

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 1.073684

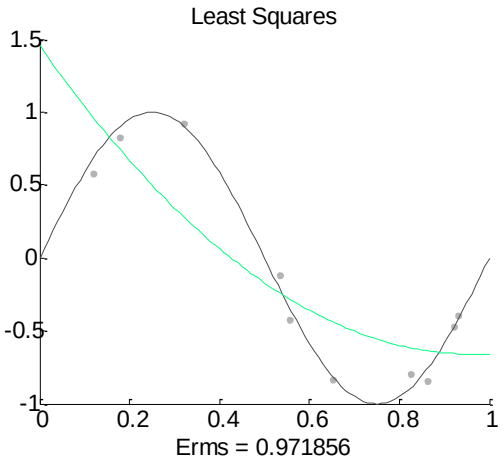

E
Iteration = 256

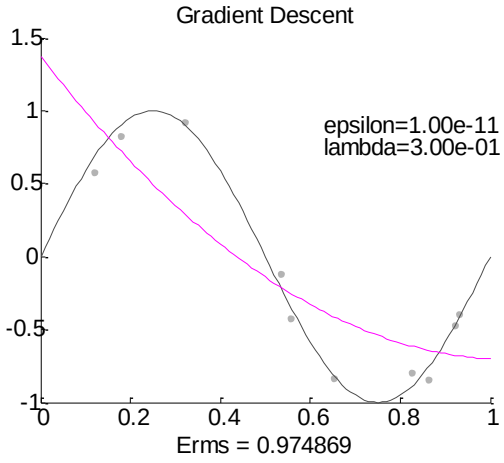

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 1.073615

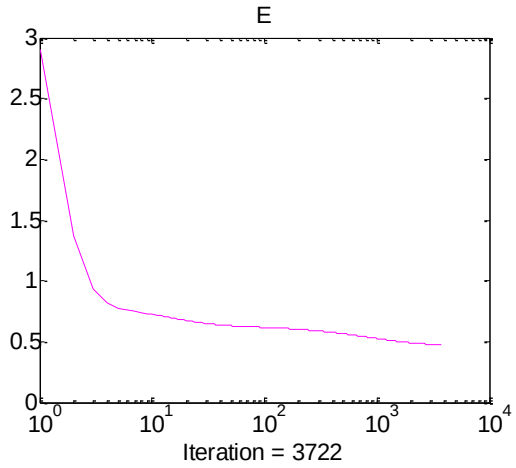

E
Iteration = 48

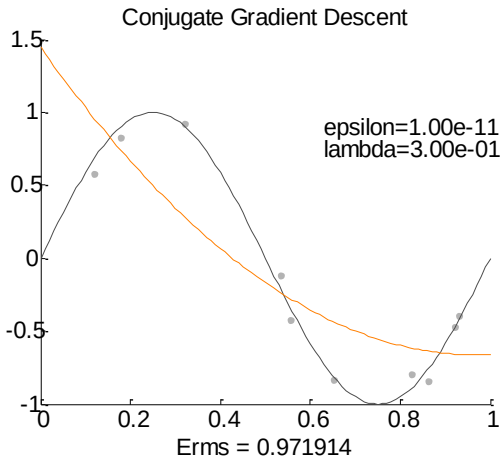
二阶

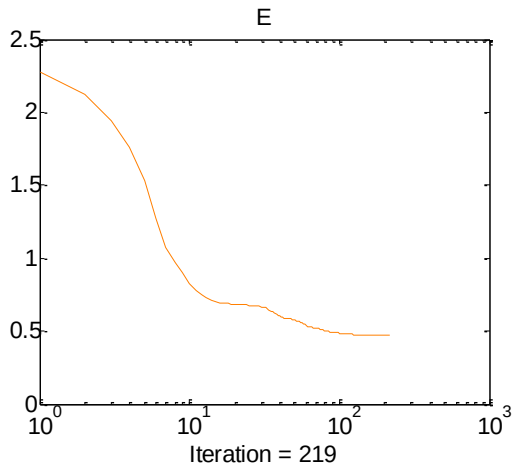

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.971856

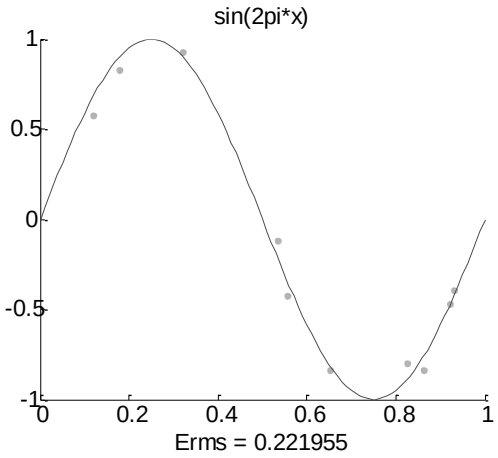

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.974869

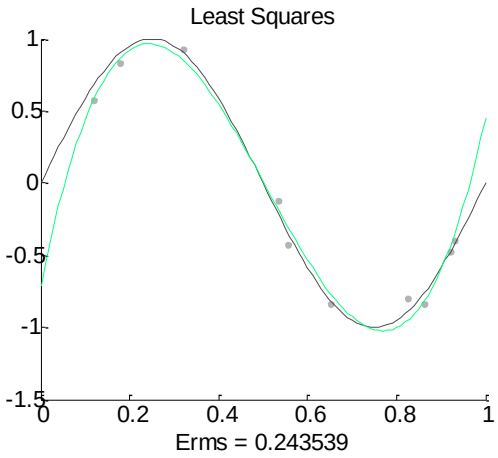

E
Iteration = 3722

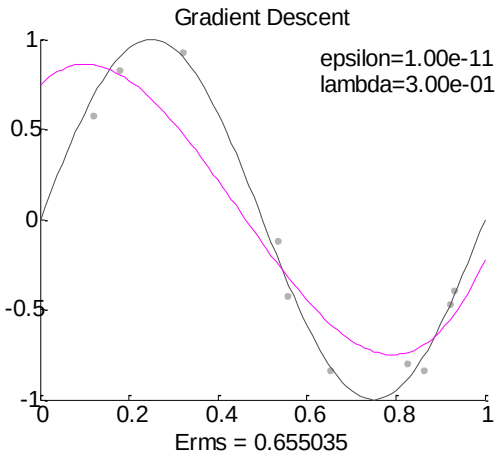

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.971914

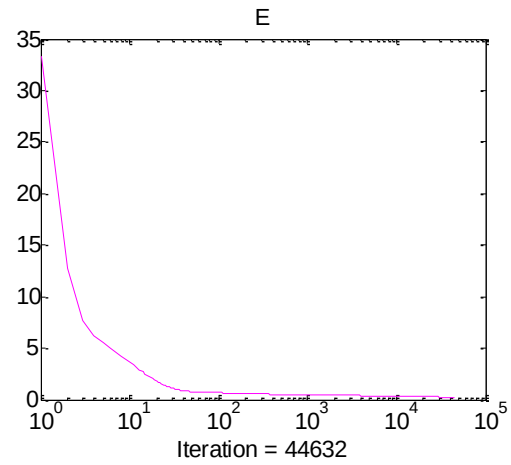

E
Iteration = 219

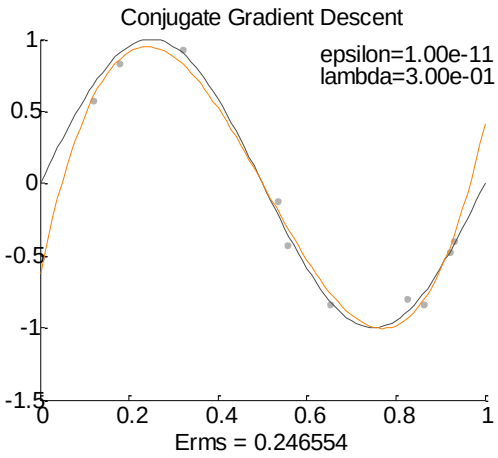
三阶

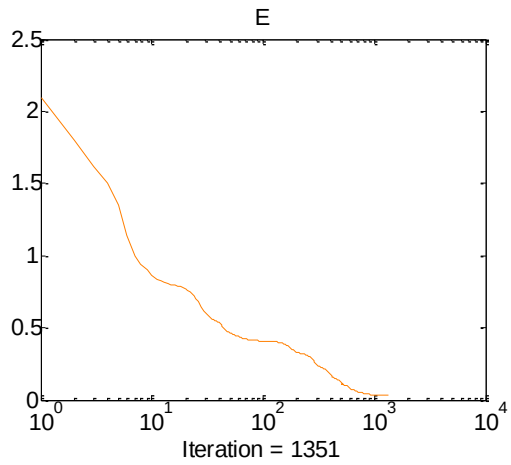

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.243539

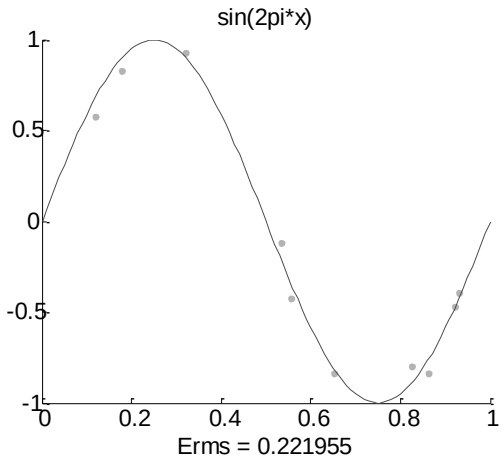

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.655035

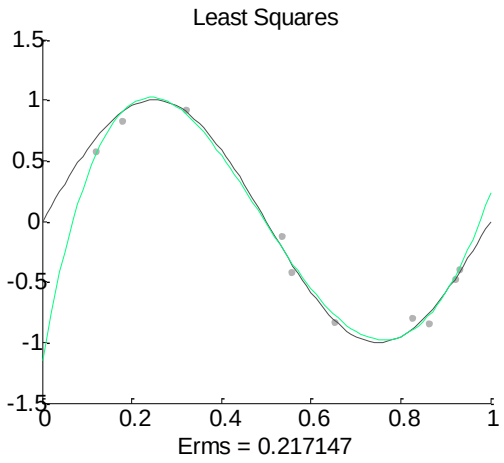

E
Iteration = 44632

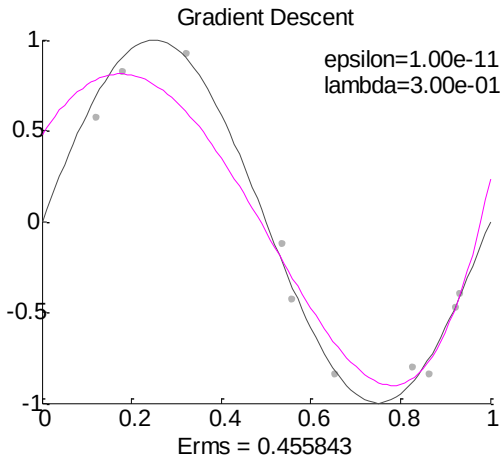

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.246554

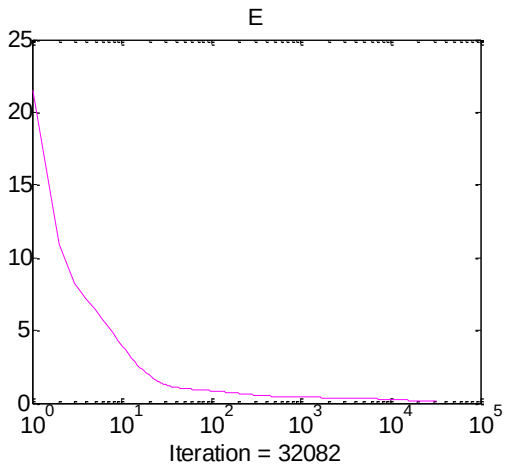

E
Iteration = 1351

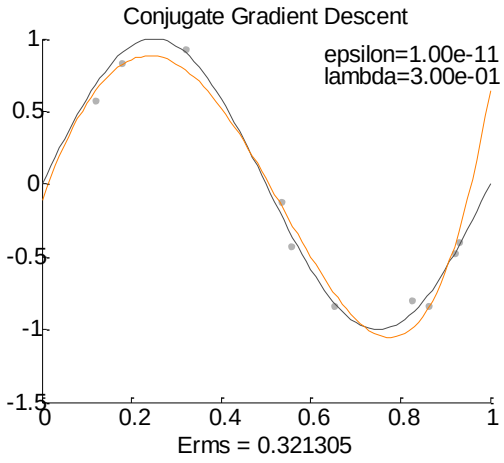
四阶

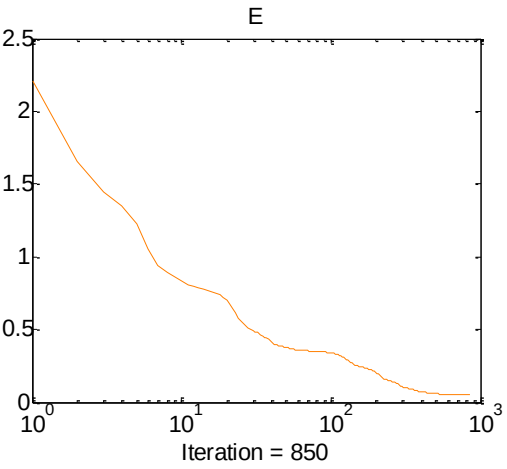

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.217147

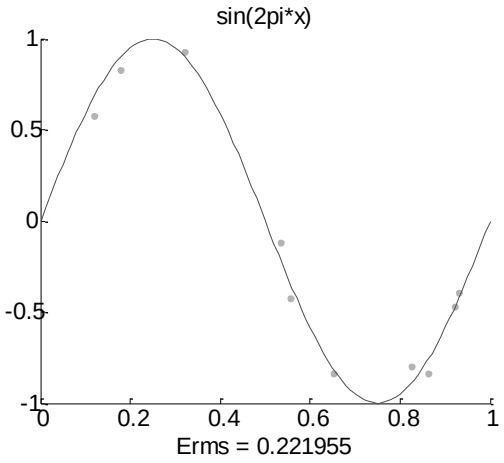

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.455843

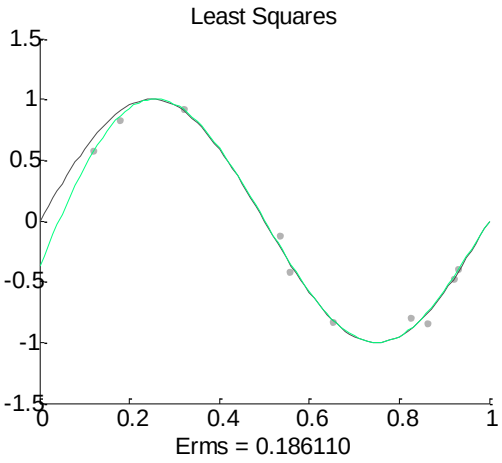

E
Iteration = 32082

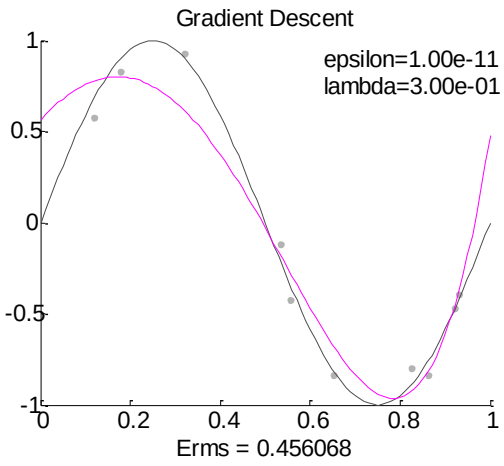

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.321305

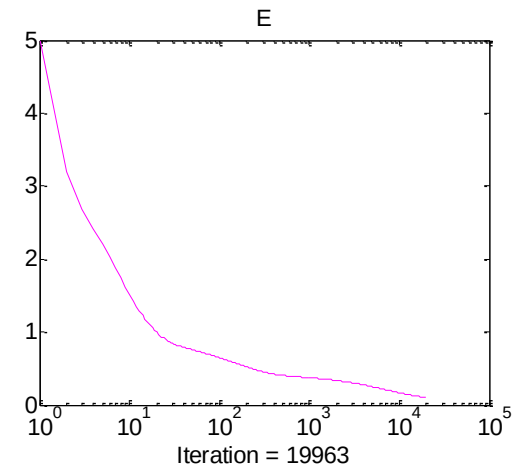

E
Iteration = 850

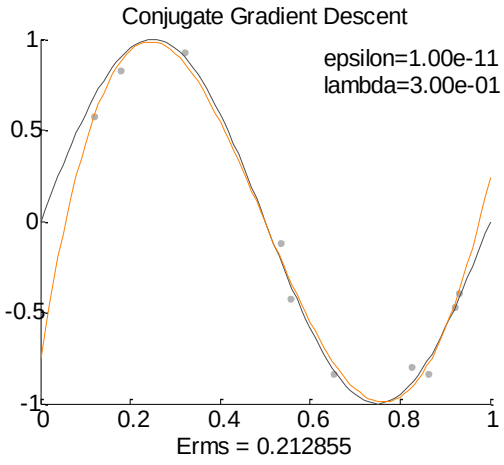
五阶

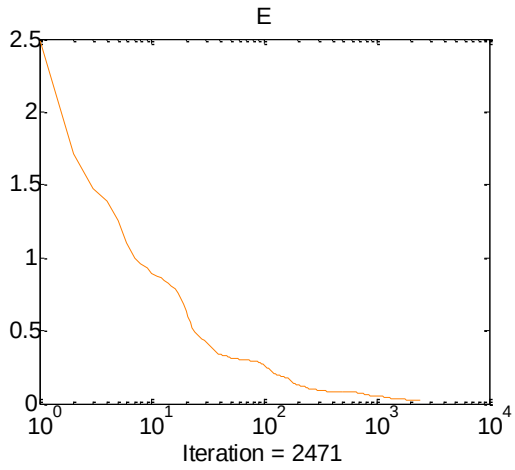

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.186110

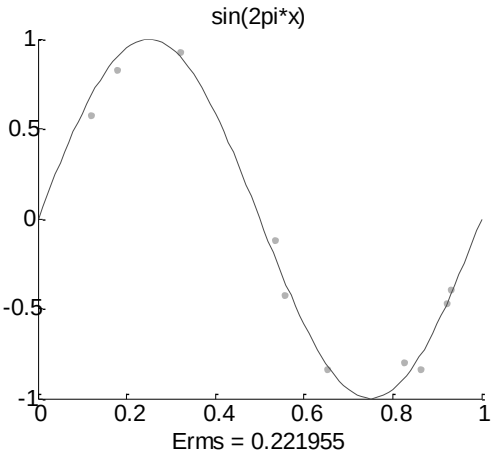

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.456068

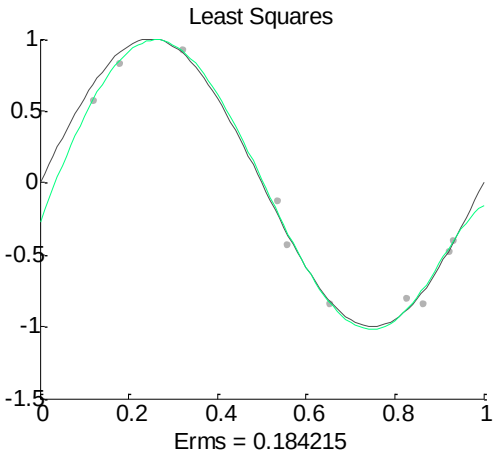

E
Iteration = 19963

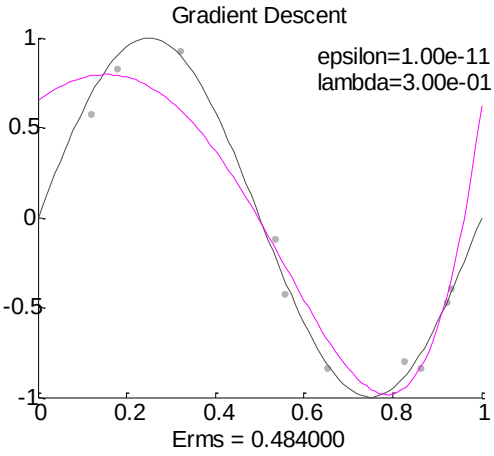

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.212855

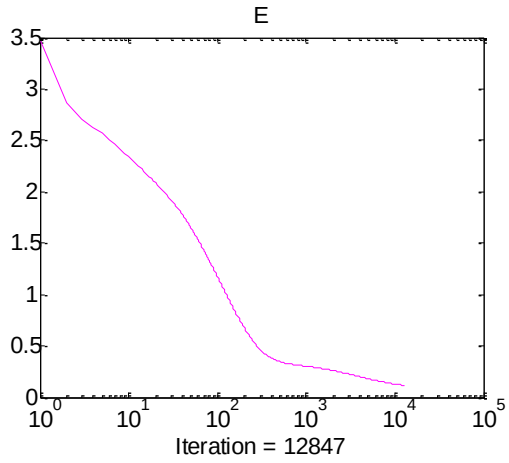

E
Iteration = 2471

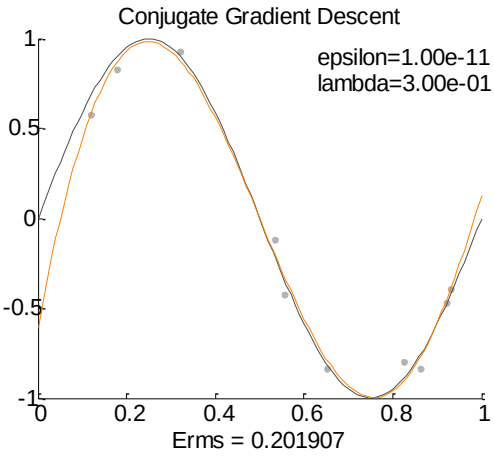
六阶

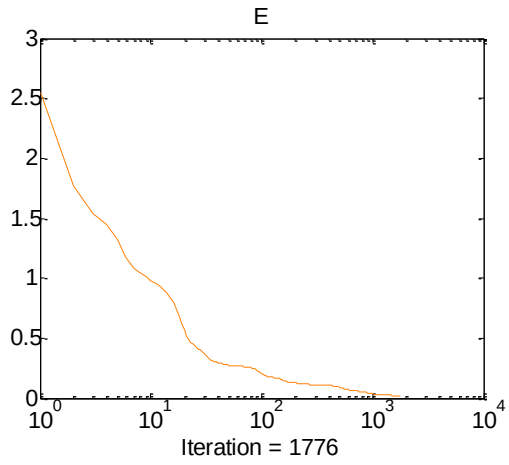

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.184215

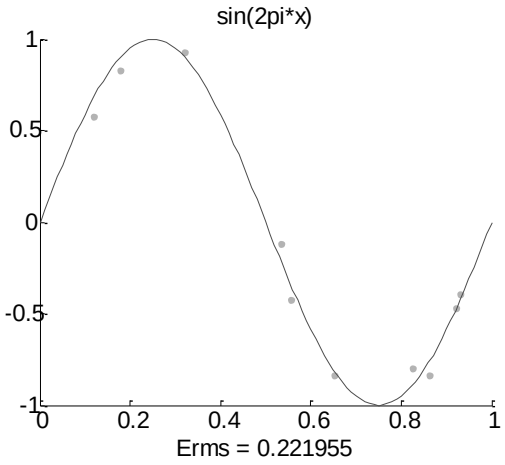

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.484000

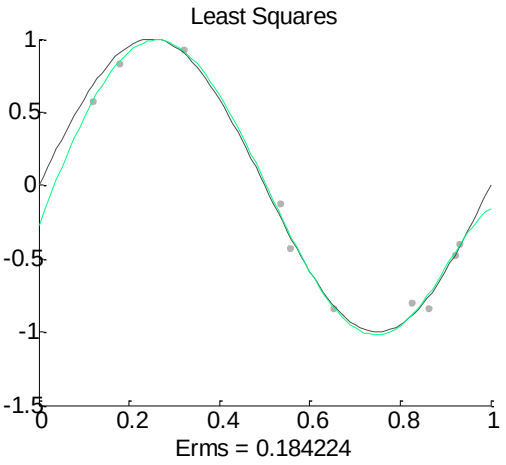

E
Iteration = 12847

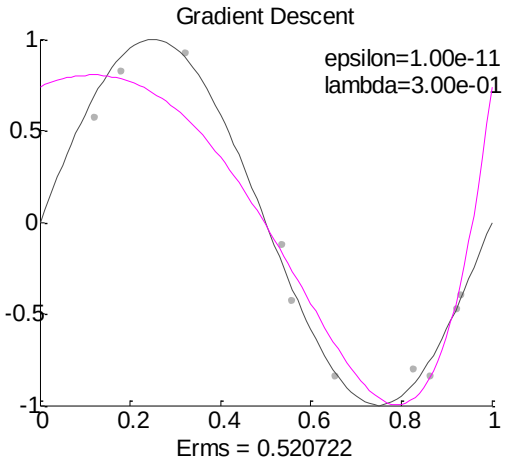

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.201907

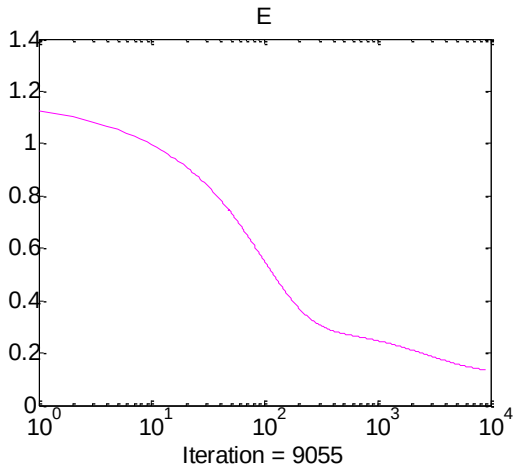

E
Iteration = 1776

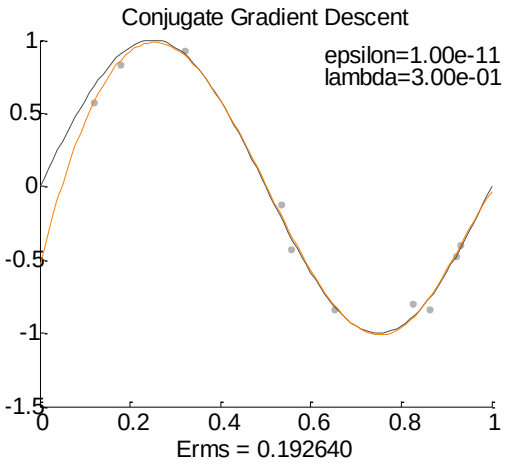
七阶

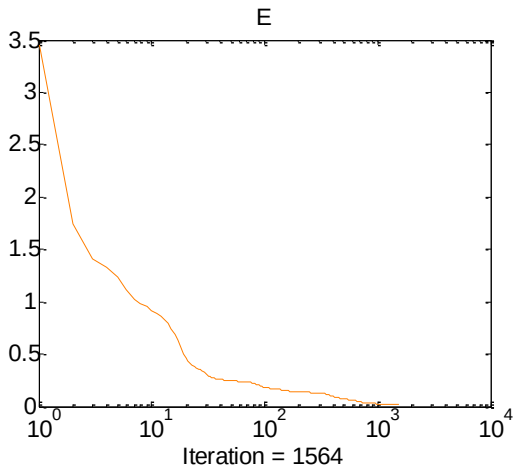

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.184224

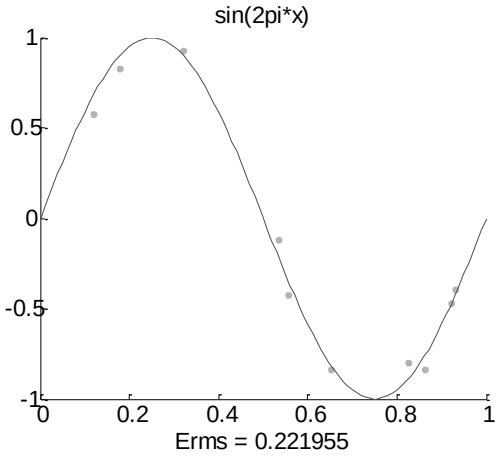

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.520722

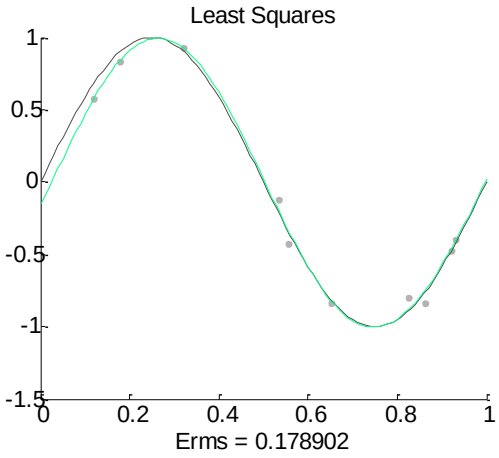

E
Iteration = 9055

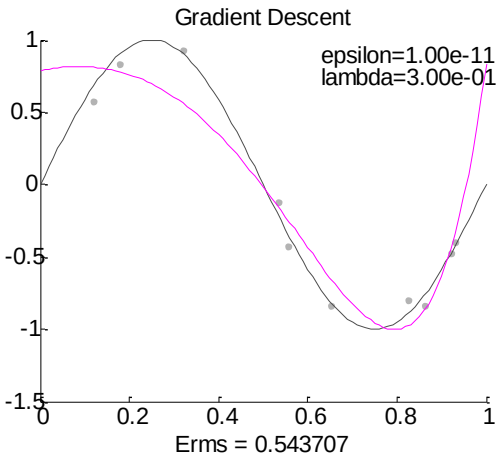

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.192640

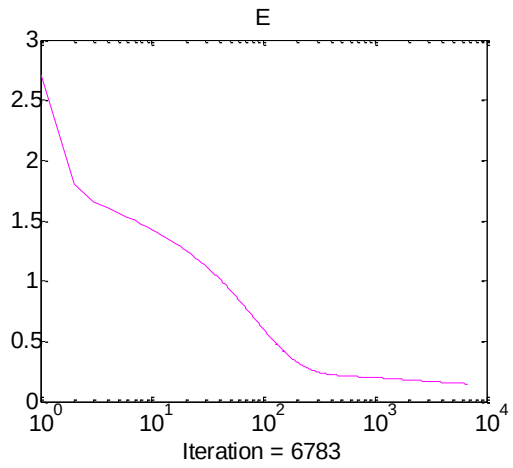

E
Iteration = 1564

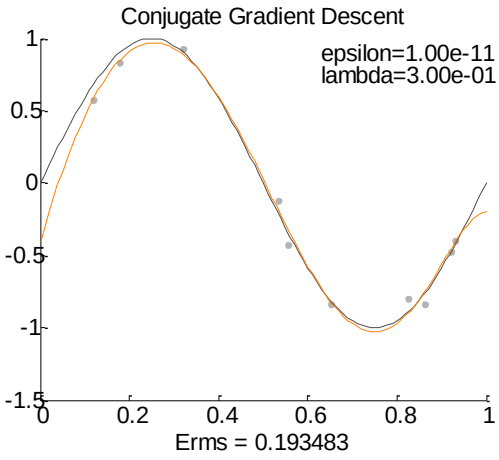
八阶

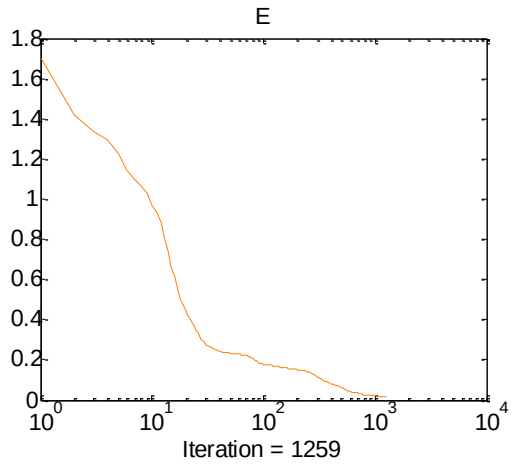

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.178902

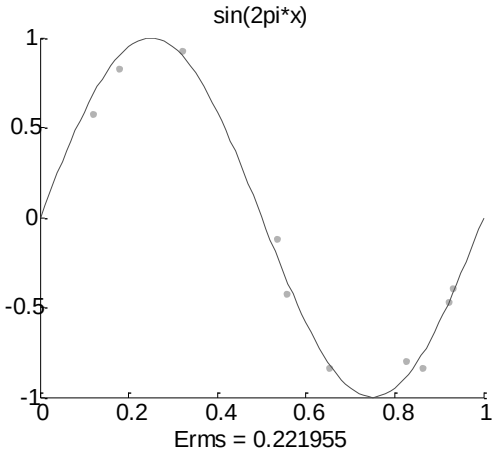

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.543707

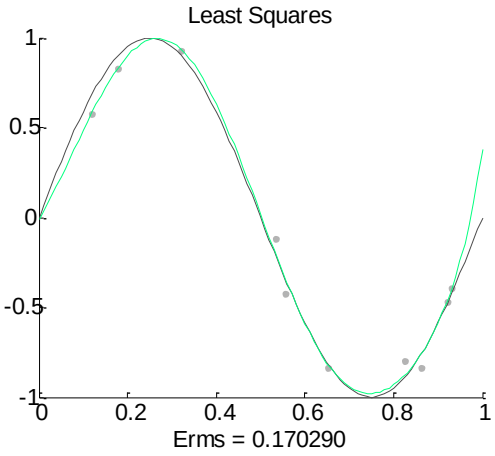

E
Iteration = 6783

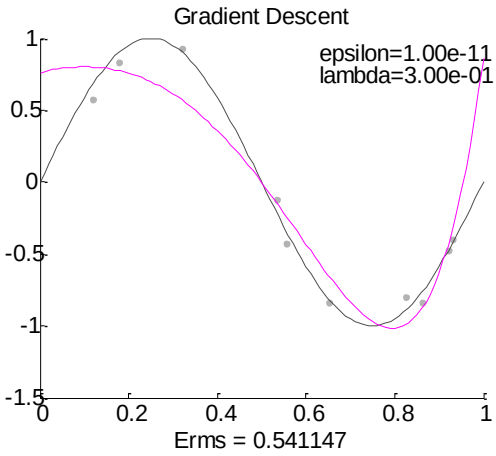

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.193483

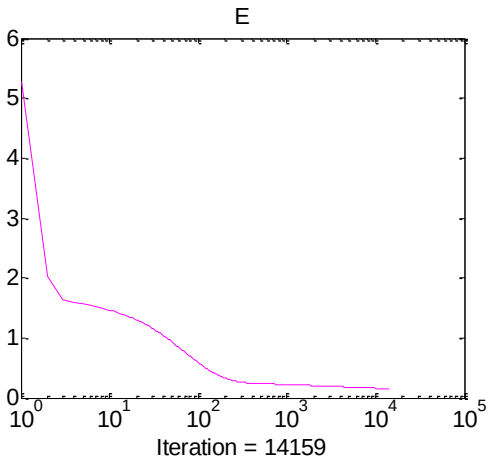

E
Iteration = 1259

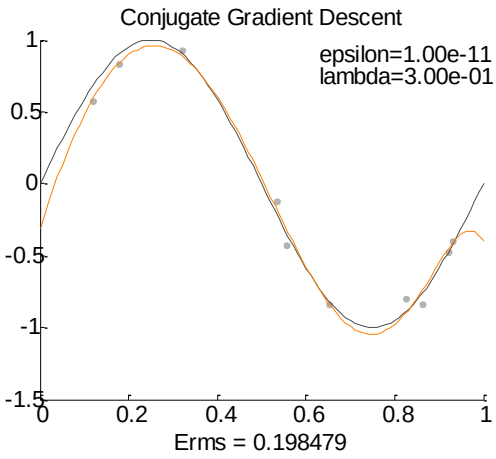
九阶

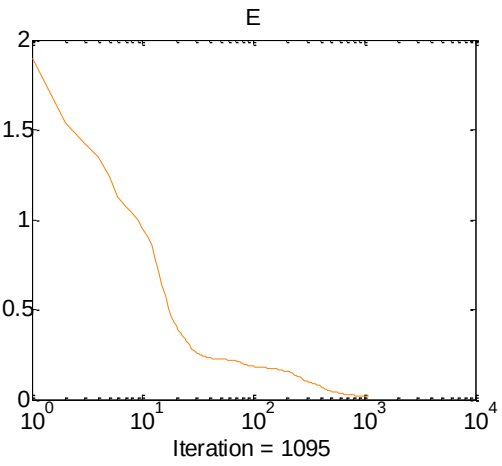

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.170290


Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.541147


E
Iteration = 14159

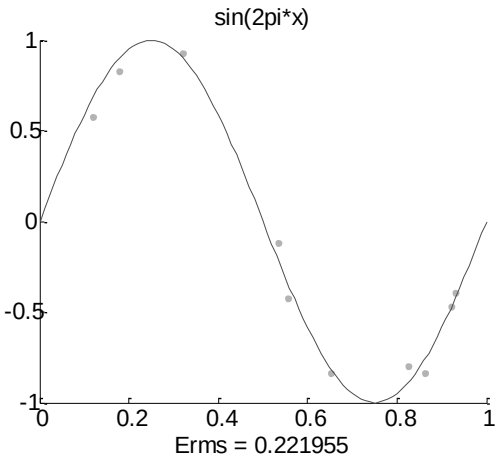

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.198479

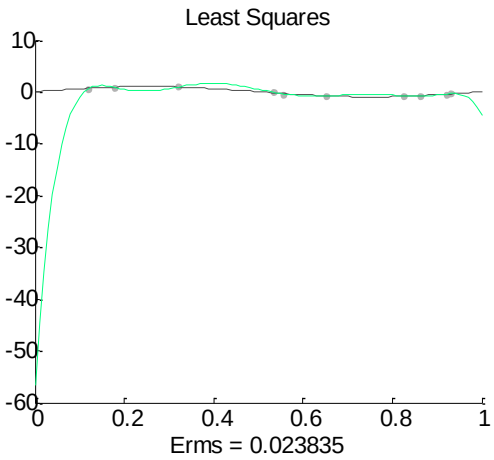

E
Iteration = 1095

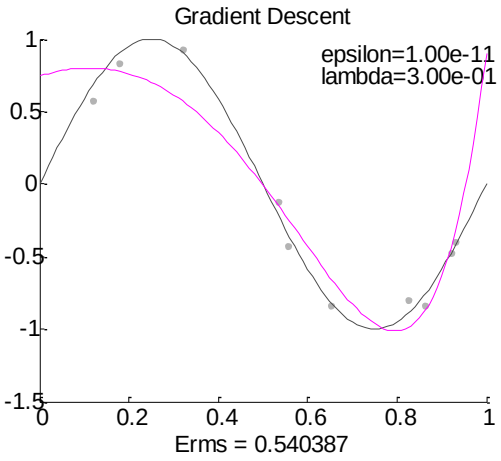
通过实验的结果可以看出，惩罚项的作用几乎是立竿见影，即使到了九阶，也没有出现（明显的）过拟合现象。惩罚项的存在也没有干扰阶数较低时拟合的效果。因此，惩罚项是一个非常有效，实用的克服过拟合的方法。

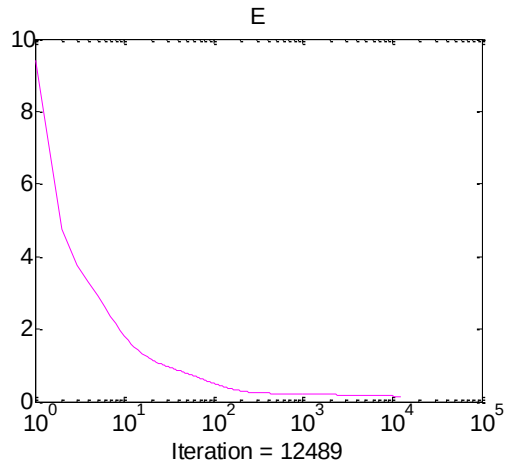
除了增加惩罚项，还有一种克服过拟合的方法是增加样本，接下来将样本数 N 依次增加，阶数直接取最容易过拟合的九阶,进行同样的实验。

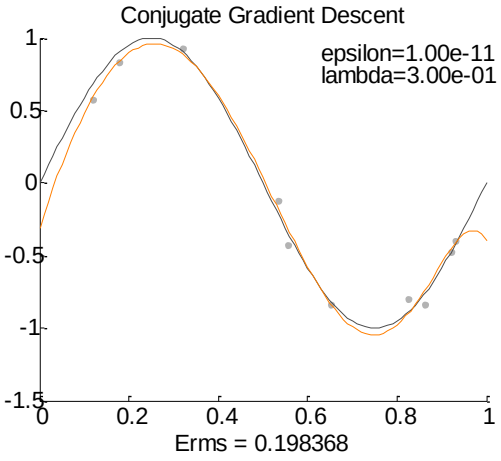
N=10

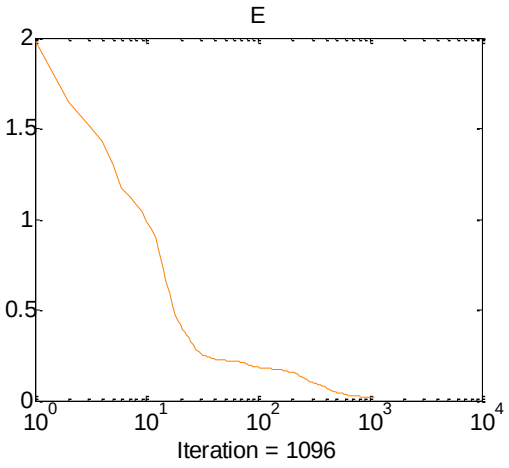

sin(2pi*x)
Erms = 0.221955


Least Squares
Erms = 0.023835

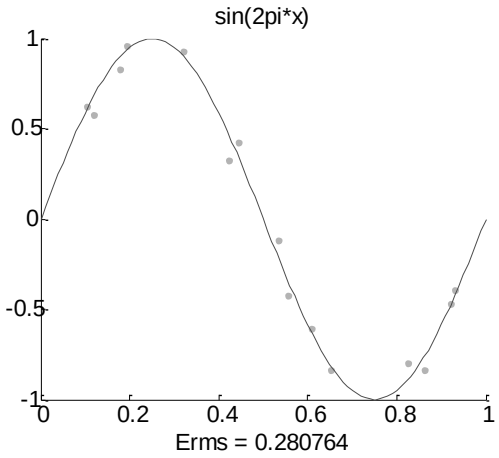

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.540387

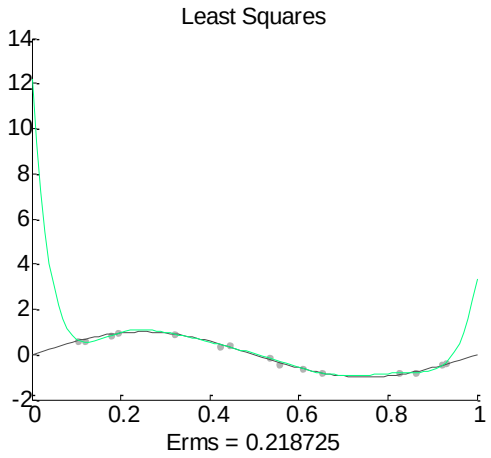

E
Iteration = 12489

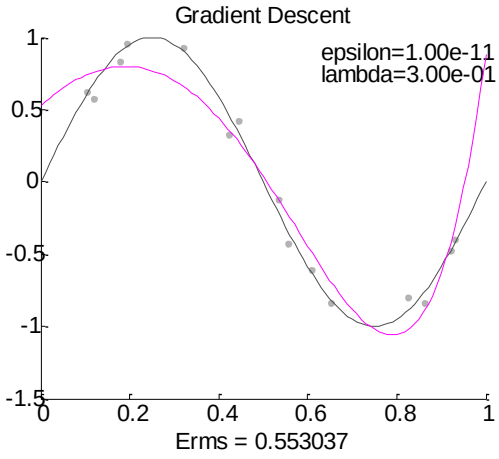

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.198368

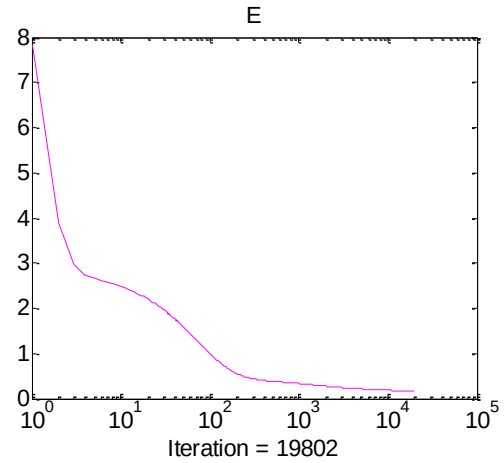

E
Iteration = 1096

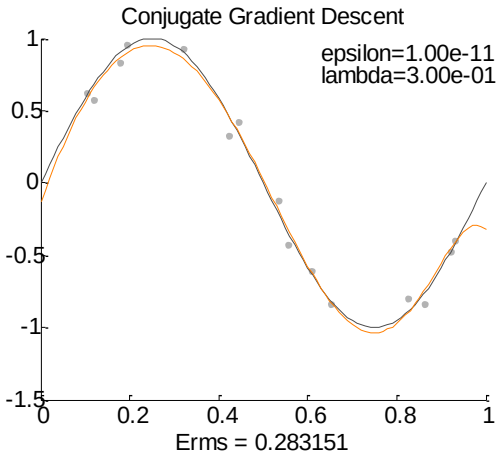
N=15

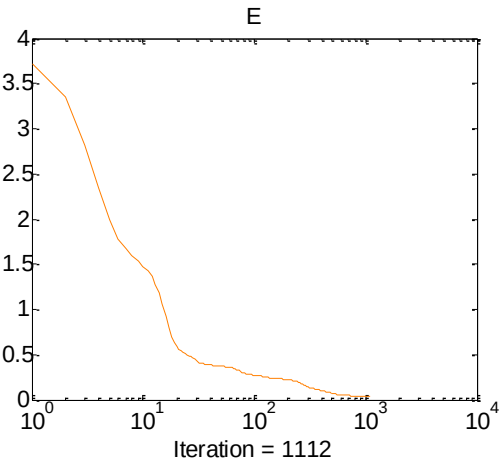

sin(2pi*x)
Erms = 0.280764


Least Squares
Erms = 0.218725

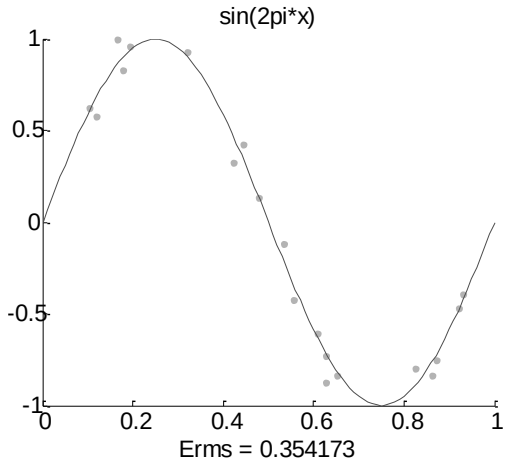

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.553037

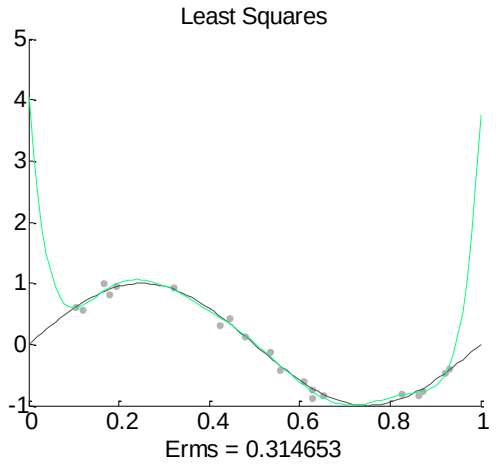

E
Iteration = 19802

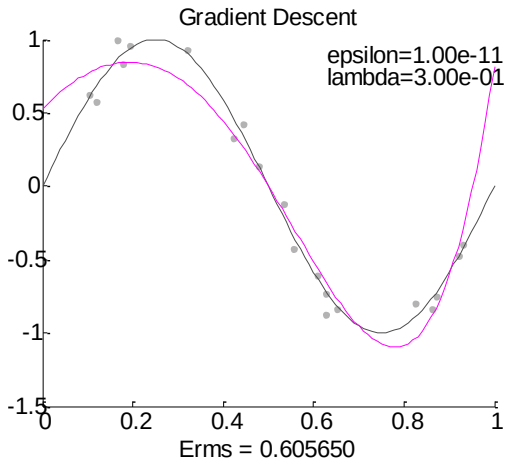

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.283151

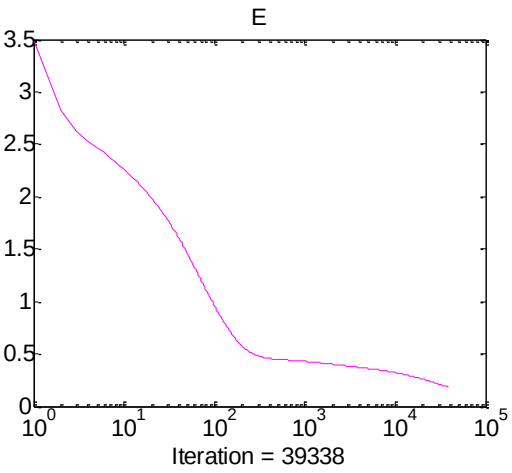

E
Iteration = 1112

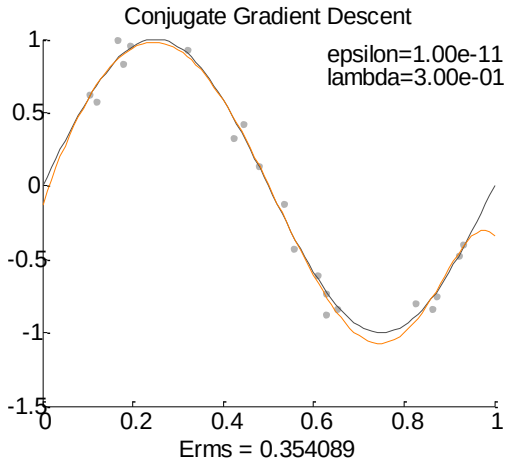
N=20

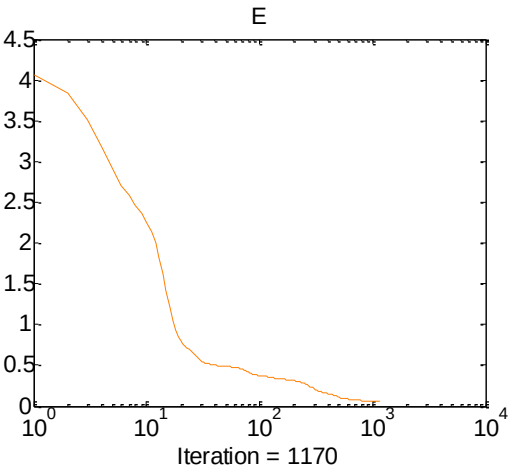

sin(2pi*x)
Erms = 0.354173


Least Squares
Erms = 0.314653

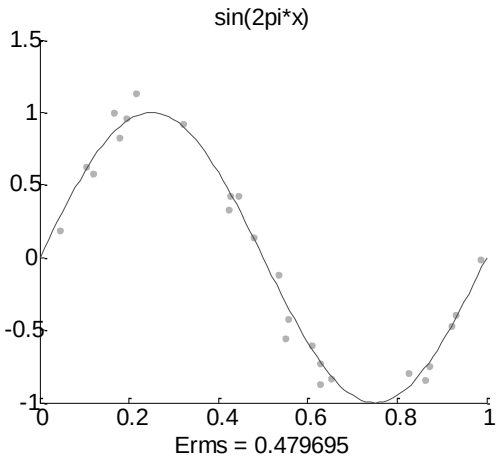

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.605650

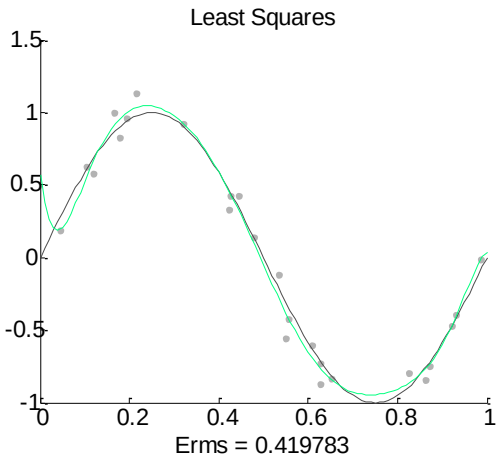

E
Iteration = 39338

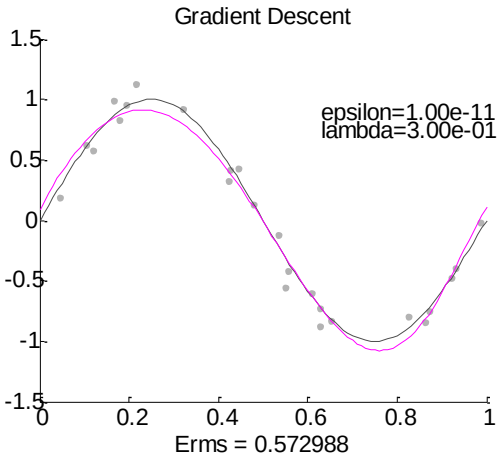

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.354089

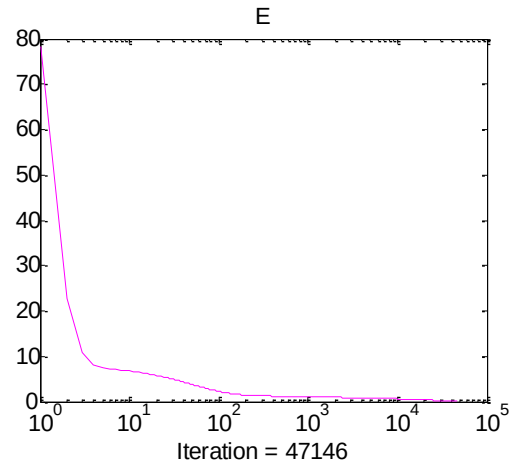

E
Iteration = 1170

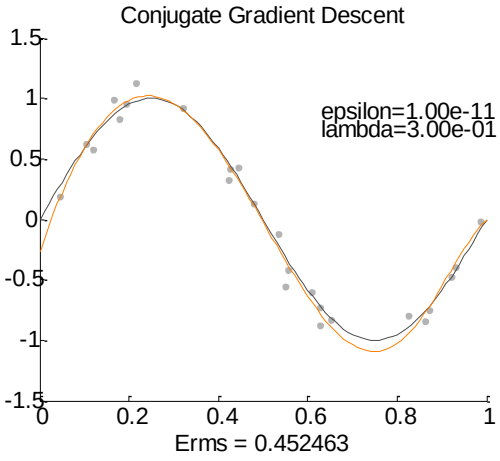
N=25

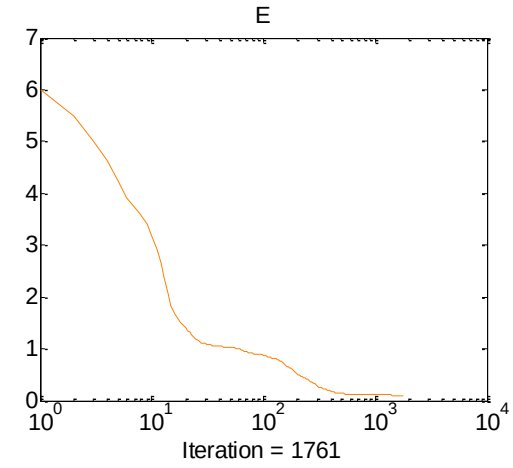

sin(2pi*x)
Erms = 0.479695


Least Squares
Erms = 0.419783

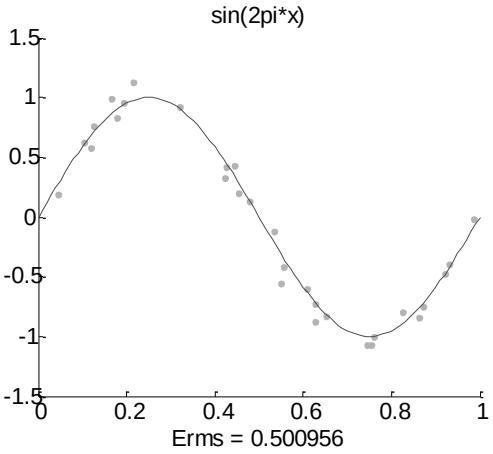

Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.572988

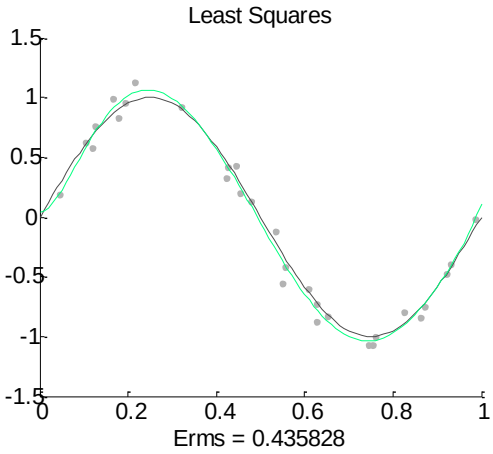

E
Iteration = 47146

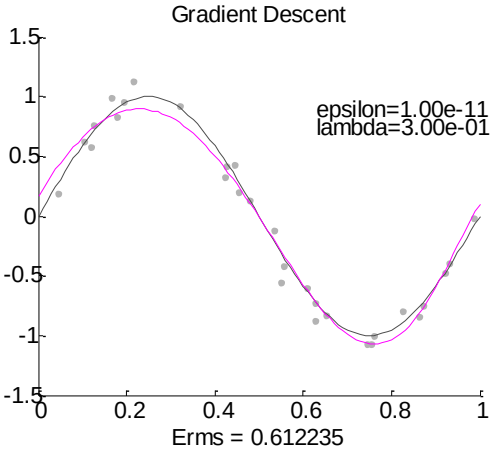

Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.452463

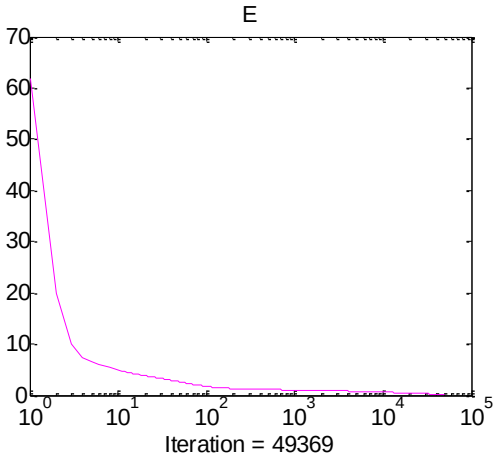

E
Iteration = 1761

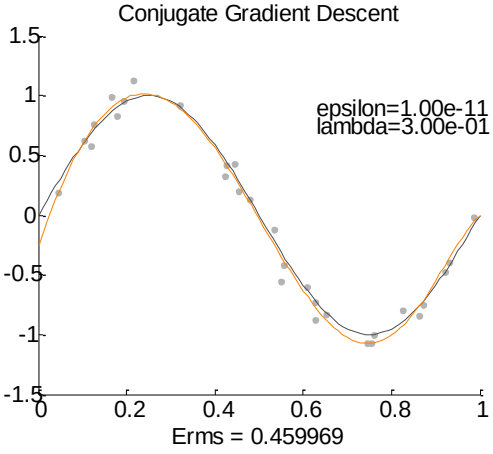
N=30

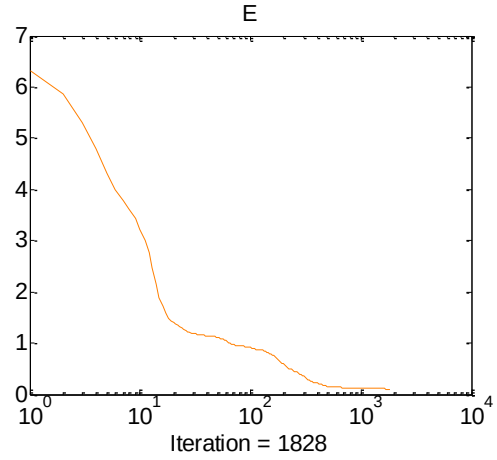

sin(2pi*x)
Erms = 0.500956


Least Squares
Erms = 0.435828


Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.612235


E
Iteration = 49369


Conjugate Gradient Descent
epsilon=1.00e-11
lambda=3.00e-01
Erms = 0.459969


E
Iteration = 1828

从实验结果来看，增加数据显然是能够克服过拟合的，但更加重要的一点是，增加数据不仅克服了过拟合，也使拟合曲线更加靠近原本的正弦曲线了。这个现象与直观的猜测是一致的，即数据越多拟合越准。
增加数据显然是一个理论上最好的选择，但在实际中可能有很大局限性，因为增加惩罚项只是简单的调整一下算式，而获取新数据可能在实际问题中十分困难，甚至有时候根本不能得到更多的数据。因此，虽然增加数据是一个很好的方法，但在实际使用中，其他的方法也是十分必要的。

五、系统不足与经验体会

本次实验主要是进行了一些验证性的实验，相对来说实验数据较少，实验的设计也比较简单，因此得到的结论不是很具有说服力。另外，在计算方面，都是采用了 matlab 进行计算，计算过程中没有考虑过精度的问题，导致结果有误差，且误差比较明显。还有，在实际的梯度下降和共轭梯度中，都有寻找最优步长的步骤，由于这一步骤没有十分简便的方法，在本次实验中都直接赋予了定值，这显然是会影响最终结果的。
通过本次实验，我更加深刻的理解了最小二乘法，梯度下降法和共轭梯度法的具体过程，也对他们的差异有了大致的了解。最小二乘法虽然误差最小，但特别容易过拟合，梯度下降法几乎没有能够胜过共轭梯度法的优势，而共轭梯度法收敛快，相对不容易过拟合，是一个很实用的方法。另外，我初步掌握了一些克服过拟合的技巧，例如增加惩罚项，可以显著的改善过拟合的情况，而增加数据不仅能够克服过拟合，还能使拟合效果更好。其次，在实验过程中需要多次进行调参，积累了一些调参的经验和技巧，调参的速度有所提高。最后，我也更加熟悉了 matlab 的使用技巧以及一些编程思想，由于 matlab 是基于矩阵的，因此将各种运算转化为矩阵运算能够提高运算效率，这也是线性代数的一种应用。

六、附录：源代码（带注释）

```
clear;clf;clc;

%==============调整参数==============
N = 30;%样本数量
M = 9;%阶数
lmd = 0.000000;%正则项的系数，没有正则项时为0

%==============获取数据==============
[data,y] = DataGenerator(N);

%==============显示数据==============
subplot(3, 2, 1);
hold on;
title('sin(2pi*x)');
sample_x = 0:0.01:1;
plot(data, y, '.', 'color', [0.7 0.7 0.7]);
```

```
plot(sample_x,sin(sample_x*2*pi), 'color', [0.3 0.3 0.3]);
xlabel(sprintf('Erms = %f', (sum((sin(data*2*pi)-y).^2))^0.5));
hold off;

%==============调整数据==============
%（将数据调整为多次的）
X = zeros(N, M);
X(:,1) = ones(N,1);
for i = 2:M+1,
    X(:,i) = X(:,i-1).*data;
end

%==============最小二乘==============
W = Normal(X, y, lmd);
sample_y = polyval(W(end:-1:1)', sample_x);
subplot(3, 2, 2);
hold on;
title('Least Squares');
plot(data, y, '.', 'color', [0.7 0.7 0.7]);
plot(sample_x,sin(sample_x*2*pi), 'color', [0.3 0.3 0.3]);
plot(sample_x,sample_y, 'color', [0 1 0.5]);
xlabel(sprintf('Erms = %f', (sum((X*W-y).^2))^0.5));
hold off;

%==============梯度下降==============
alpha = 0.3;
epsilon = 0.00000000001;
W = randn(M+1, 1);%随机初值
J = [];%记录每次迭代的均方差
[J(end+1), grad] = costFunction(W, X, y, lmd);
W = W - alpha.*grad;
[J(end+1), grad] = costFunction(W, X, y, lmd);
W = W - alpha.*grad;
while (J(end)-J(end-1))^2>epsilon,
    [J(end+1), grad] = costFunction(W, X, y, lmd);
    W = W - alpha.*grad;
end

%绘图
subplot(3, 2, 4);
semilogx(1:size(J,2),J, 'color', [1 0 1]);
hold on;
title('E');
xlabel(sprintf('Iteration = %d', size(J,2)));
```

```
hold off;
subplot(3, 2, 3);
hold on;
title('Gradient Descent');
text(0.63, 0.9, sprintf('epsilon=%.2e', epsilon));
text(0.63, 0.75, sprintf('lambda=%.2e', alpha));
plot(data, y, '.', 'color', [0.7 0.7 0.7]);
sample_y = polyval(W(end:-1:1)', sample_x);
plot(sample_x,sin(sample_x*2*pi), 'color', [0.3 0.3 0.3]);
plot(sample_x,sample_y, 'color', [1 0 1]);
xlabel(sprintf('Erms = %f', (sum((X*W-y).^2))^0.5));
hold off;

%==============共轭梯度==============
epsilon2 = epsilon;
lambda = 0.3;
W = randn(M+1, 1)/10;
J = [];
[J(end+1), gradk] = costFunction(W, X, y, lmd);
pk = -gradk;
W = W + lambda*pk;
[J(end+1), gradk1] = costFunction(W, X, y, lmd);
beta = sum(gradk1.^2)/sum(gradk.^2);
pk1 = beta*pk-gradk1;
W = W + lambda*pk1;
gradk = gradk1;
pk = pk1;
while (J(end)-J(end-1))^2>epsilon2,
    [J(end+1), gradk1] = costFunction(W, X, y, lmd);
    beta = sum(gradk1.^2)/sum(gradk.^2);
    pk1 = beta*pk-gradk1;
    W = W + lambda*pk1;
    gradk = gradk1;
    pk = pk1;
end

%绘图
subplot(3, 2, 6);
semilogx(1:size(J,2),J, 'color', [1 0.5 0]);
hold on;
title('E');
xlabel(sprintf('Iteration = %d', size(J,2)));
sample_y = polyval(W(end:-1:1)', sample_x);
hold off;
```

```
subplot(3, 2, 5);
hold on;
title('Conjugate Gradient Descent');
text(0.63, 0.9, sprintf('epsilon=%.2e', epsilon2));
text(0.63, 0.75, sprintf('lambda=%.2e', lambda));
plot(data, y, '.', 'color', [0.7 0.7 0.7]);
plot(sample_x,sin(sample_x*2*pi), 'color', [0.3 0.3 0.3]);
plot(sample_x,sample_y, 'color', [1 0.5 0]);
xlabel(sprintf('Erms = %f', (sum((X*W-y).^2))^0.5));
hold off;


function [ W ] = Normal( X, y, lmd )
%计算最小二乘法的解

lambda = lmd;
W = pinv(X'*X + lambda*eye(size(X,2)))*X'*y;

end


function [ X, y ] = DataGenerator( N )

%数据可以是随机产生的，也可以是用下方矩阵截取的
%这样设计是为了保证多次实验使用同样的数据

%X = rand(N,1);
X =
[0.118907558219845;0.650181148301737;0.178899912608483;0.919854108058750;0.555616728
397525;0.929961851168603;0.321445798171207;0.535497951929982;0.861541592008529;0.824
039042920457;0.444894999943138;0.422216499686049;0.104086602945463;0.60878137563743
0;0.193630622678006;0.626097512939333;0.165131817294095;0.625434000097254;0.47931907
8563128;0.872373759739693;0.216272892753600;0.548489755587899;0.044276181783365 2;0.9
87227638559375;0.425965392531060;0.753573244213155;0.761604945840599;0.452745688158
287;0.744019979809978;0.126219916256357;0.108712423939115;0.525033797162904;0.432186
974392176;0.802605862524551;0.329732115283355;0.164594887779254;0.972260853874559;0.
802881709879834;0.782435635746238;0.332209065255501;0.154880226460795;0.43368877925
6687;0.947035539268100;0.414700356721901;0.637469592629435;0.043888020369806;0.5973
38982587062;0.322391138768930;0.475942960960871;0.595089561155079;0.367866755487244
;0.568352173753013;0.388651371031664;0.948424189082716;0.560824624796265;0.356498289
834951;0.862535443725515;0.861336389662243;0.561735645110092;0.672420589638419;0.668
032517312457;0.717494507379800;0.375039556022439;0.934318731858537;0.09221050512048
14;0.593853088208639;0.059906180196 1192;0.880515583091734;0.797508073232580;0.641348
602185854;0.803732464929721;0.194418811371919;0.720922072233337;0.397139620638693;0.
```

100593409602541;0.964909587781395;0.534326005250434;0.974501656792706;0.88887606703
1278;0.211449713799404;0.804247393241236;0.765976703953730;0.154595830216135;0.87544
0143354608;0.00636129033144661;0.482103228259781;0.865389698561101;0.77686170465188
9;0.341554634336814;0.174007179148808;0.096997241999549 8;0.565396659912707;0.6082911
38312068;0.209512967857529;0.556809772134110;0.038124347 2819210;0.791008027270643;0.
861328079724631;0.657344377894258;0.999933861494954];
%y = sin(X.*2*pi) + randn(N,1)*0.1;
y = [0.579858169047580;-0.832389984792784;0.828732384243244;-0.470452808844018;-
0.422479951003150;-0.395291397387722;0.926172671173564;-0.116969106550776;-
0.838193778149256;-
0.798857553998058;0.427392355106608;0.326342391112487;0.623633197889601;-
0.606290250783126;0.959145994836184;-0.730385609614730;0.996010867037011;-
0.873865262123308;0.136668664759480;-0.748119876316139;1.13355328590518;-
0.560322344133264;0.188326491315289;-0.013709589149278 8;0.421847348029066;-
1.07389620502478;-1.00475656414532;0.205246471502390;-
1.07491173330466;0.756716184504968;0.765189287861904;-
0.272803706863018;0.346936195657969;-
1.01286556684942;0.906040324704663;0.861091089951148;-0.205881616925362;-
0.984223832558263;-
0.891923606227507;0.884336906701667;0.999803568106895;0.498530103361023;-
0.220030520253484;0.487978202011596;-0.730341921389512;0.376385885469887;-
0.489349980347442;0.836205022446378;0.00217347382035846;-
0.632733163823629;0.907144975188520;-0.659730287323253;0.600777382893008;-
0.370340120578385;-0.386216360856734;0.842749931181913;-0.742004096365696;-
0.506058179612786;-0.373387404188194;-0.902069651430828;-0.788008998097991;-
0.976766242550903;0.782470917674058;-0.456995506555782;0.612076209133347;-
0.427183163805453;0.368228472448296;-0.711454151113390;-0.812980942488454;-
0.815307765746932;-0.889324673753956;0.692399821371964;-
0.966462579989446;0.605049964521341;0.605808544454485;-0.136288287798405;-
0.292111017887689;-0.102001427497861;-0.759262647454141;0.808867392487452;-
1.00802294274761;-0.992235741449020;0.968841985643549;-
0.763801411329639;0.063785642502176 4;0.248457367785236;-0.746859976128228;-
0.815836448373036;0.766937621787460;0.944755877865963;0.596979913860633;-
0.603756093890094;-0.725540524596933;0.982250795406439;-
0.328304241767545;0.087362513533432 8;-1.05599729860724;-0.662860226320163;-
0.933677996642019;0.025101098103084 9];

X = X(1:N);
y = y(1:N);
end