哈爾濱工業大學

机器学习实验报告

实 验（一）

题　　目　多项式拟合

专　　业　计算机科学与技术

学　　号　1161000309

班　　级　1603104

学生姓名　高靖龙

指导教师　刘扬

实验地点　G208

实验日期　2018-09-25

计算机科学与技术学院

目　录

第 1 章 实验基本信息

1.1 实验目的

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

1.2 实验环境与工具

1.2.1 硬件环境

- CPU:corei7

1.2.2 软件环境

- manjaro linux
- pycharm
- matlab
- python 3.7
  - numpy
  - matplotlib

1.3 实验要求

1. 用高阶多项式函数拟合曲线；
2. 生成数据，加入噪声；
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。

第 2 章 算法原理及实现

2.1 解析解法

算法思路：

使用多项式 $y(x,w)=\sum_{j=0}^{M} w_j x^j$ 来对曲线进行拟合，为了缓解过拟合，使用岭回归控制过拟合。针对超参数，需要进行模型选择。

loss 函数：

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

取 loss 函数最小时的 W 为估计值。

计算过程：

当不加惩罚项

计算梯度

$$\nabla \ln p(\mathbf{t}|\mathbf{w},\beta)=\sum_{n=1}^{N}\left\{t_n-\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\right\}\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}.$$

计算零点

$$0=\sum_{n=1}^{N}t_n\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}-\mathbf{w}^{\mathrm{T}}\left(\sum_{n=1}^{N}\boldsymbol{\phi}(\mathbf{x}_n)\boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}}\right)$$

得出结果

$$\mathbf{w}_{\mathrm{ML}}=\left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}$$

加入惩罚项只是增加 λI：

$$\mathbf{w} = \left(\lambda \mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \mathbf{t}.$$

进行矩阵的转置、求逆、乘法、加法，即可得出拟合结果。

算法实现：

```
lenW = n + 1
if lnLambada == None:

    lambada = 0
else:
    lambada = math.e ** lnLambada

XX = mat([[x ** i for i in range(lenW)] for x in X])
vectorT = mat(T).T
XXT = XX.T
return ((lambada * numpy.eye(lenW) + XXT * XX).I * XXT * vectorT).T.tolist()[0]
```

工具库：numpy

使用 numpy 来完成矩阵的运算，包括转置、求逆、矩阵乘法、矩阵加法

design matrix：

用 XX 形象的表示，它是一个 Vandermonde matrix，使用列表生成器实现。

Lambda：

惩罚项系数通过 lnlambada 控制，使用 ln 而不直接使用 lambda 是为更精确的控制 lambda 大小。这里不写成 lambda 是因为 lambda 是保留字。当不设置惩罚项设置 lnlambada 为 None。

2.2 梯度下降

算法思路：

梯度下降的思想很简单，由于 loss 是一个凸函数，我们可以计算样本点的梯度，并按逆梯度方向移动，不断迭代来接近目标。但是学习率的选择比较困难，需要不断尝试。结合 Bayesian 的观点，学习率可以归结为对概率模型的先验认识。

BGD、SGD、MBGD 的区别：

主要在于每次选取几个样本来进行 W 的更新。

SGD 相对而言更新速度快而且能够实现在线学习，但是学习方向的准确性下降，可能出现 zig-zag 型的趋近。而且其趋近也是螺旋式的，在趋近过程中可能出频繁

出现局部恶化。
BGD 更新速度慢，因为每次都要应用所有的数据。
MBGD 兼顾了两个的特点，但是实际效果还要在应用背景下考察，一般 batch 设置为 10。

计算过程：
算法上主要在于计算梯度来更新 W。

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta(t_n - \mathbf{w}^{(\tau)\mathrm{T}}\boldsymbol{\phi}_n)\boldsymbol{\phi}_n$$

$$\boldsymbol{\phi}_n = \boldsymbol{\phi}(\mathbf{x}_n)$$

利用新数据来修改先验知识。
算法实现：
实现了 MBGD，当 batch=1,MBGD=SGD（不考虑随机性，毕竟数据是随机的，如果想实现完全随机可以加入随机化函数），当 batch=len（sample），MBGD=BGD。

```
wSize = n + 1
count = 0
W = [5 for i in range(wSize)]
batchX = []
batchT = []

for i in range(maxItrTimes):
    for x, t in zip(X, T):
        batchX.append(x)
        batchT.append(t)
        count += 1
        if count % batch == 0:
            gradient = getGradient(batchT, W, batchX)
            W = [w + lr * g / batch for w, g in zip(W, gradient)]
            rss = RSS(T, X, W, range=10, isaverage=True)
            if rss <= targetAverageRSS :
                return rss, W
            print("%d %f %e" % (count, lr, rss))
            batchX = []
            batchT = []
return rss, W
```

就是每次选取 batch 个样本来进行梯度下降。
关于如何停止算法没有想到更好的方法，目前是计算最新 10 个样本关于 W 的平均 RSS，之后当平均 RSS 小于 targetAverageRSS 时停止迭代，targetAverageRSS 是函数参数之一，默认为 5*10^-4。

2. 3 共轭梯度法

算法思路：

共轭梯度也是一种梯度更新法，用来解决 AX=b 但是它将目标 W 分割成多个相互关于 A 共轭的基底。因此每次在某个方向上增量完全再更换方向，其更新也是基于梯度的，但是并不按照梯度方向来更新。

计算过程：

具体算法上，每次确认新的方向是使用旧的方向加上残差，使用类似 simit-regular 的方法计算出新的方向，但此时每次都要使用之前所有的方向，计算速度慢。

之后验证在每次转向时的残差之间也是正交的。由残差正交和方向共轭两个条件推导公式，并对新方向计算公式进行优化，使得其只使用残差来更新方向，优化算法速度。

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$$

$$\mathbf{p}_0 := \mathbf{r}_0$$

$$k := 0$$

repeat

$$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$$

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$$

if r_{k+1} is sufficiently small, then exit loop

$$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}$$

$$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

$$k := k + 1$$

end repeat

The result is $\mathbf{x}_{k+1}$

算法实现：

```
lenW = n + 1
if lnLambada == None:

    lambada = 0

else:
    lambada = math.e ** lnLambada
```

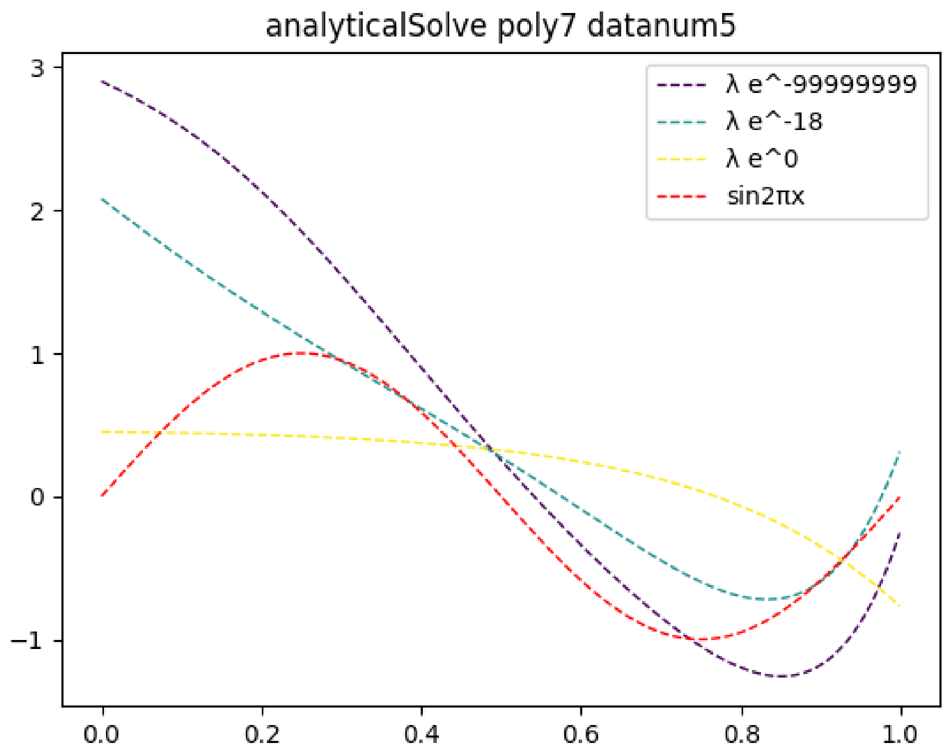
```
XX = mat([[x ** i for i in range(lenW)] for x in X])
vectorT = mat(T).T
A = XX.T * XX + lambada * numpy.eye(lenW)   # 带惩罚项
B = XX.T * vectorT
W = mat(zeros((lenW, 1)))
r = B - A * W
p = r.copy()
num = 0
while num < MaxIterationNum:
    num += 1
    alpha = (r.T * r / (p.T * A * p))[0, 0]
    W += alpha * p
    lastr = r.copy()
    r -= alpha * A * p
    if (r.T * r)[0, 0] < limit:
        break
    beta = (r.T * r / (lastr.T * lastr))[0, 0]
    p = r + beta * p
return W.T.tolist()[0], num
```

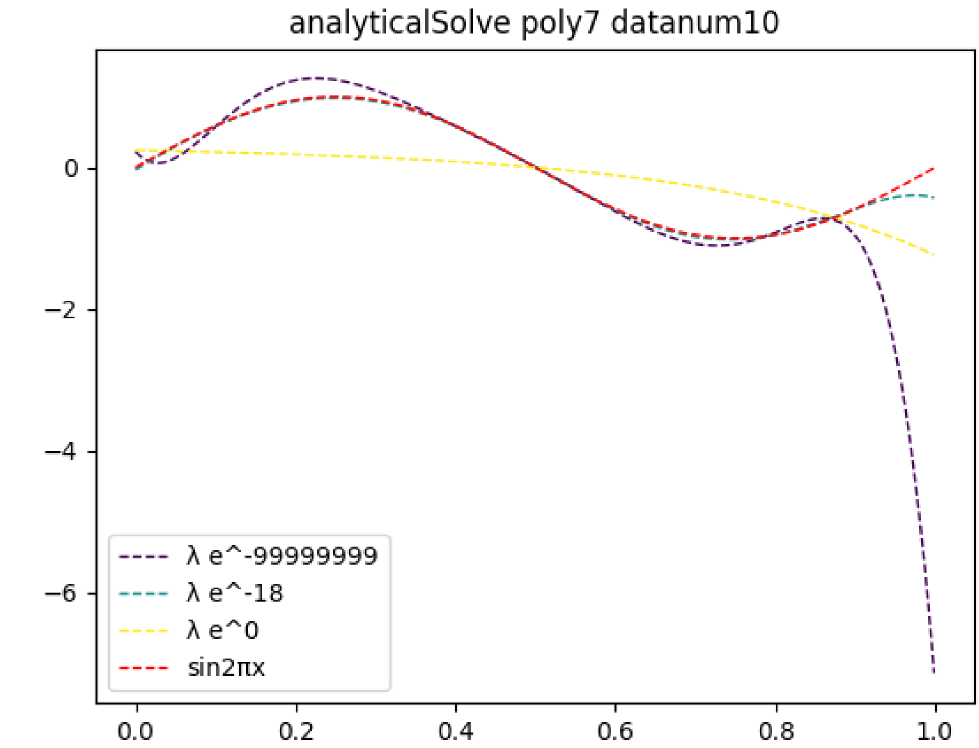
这完全是按照预处理共轭梯度法算法完成的，主要使用 numpy 完成矩阵运算，包括求逆。

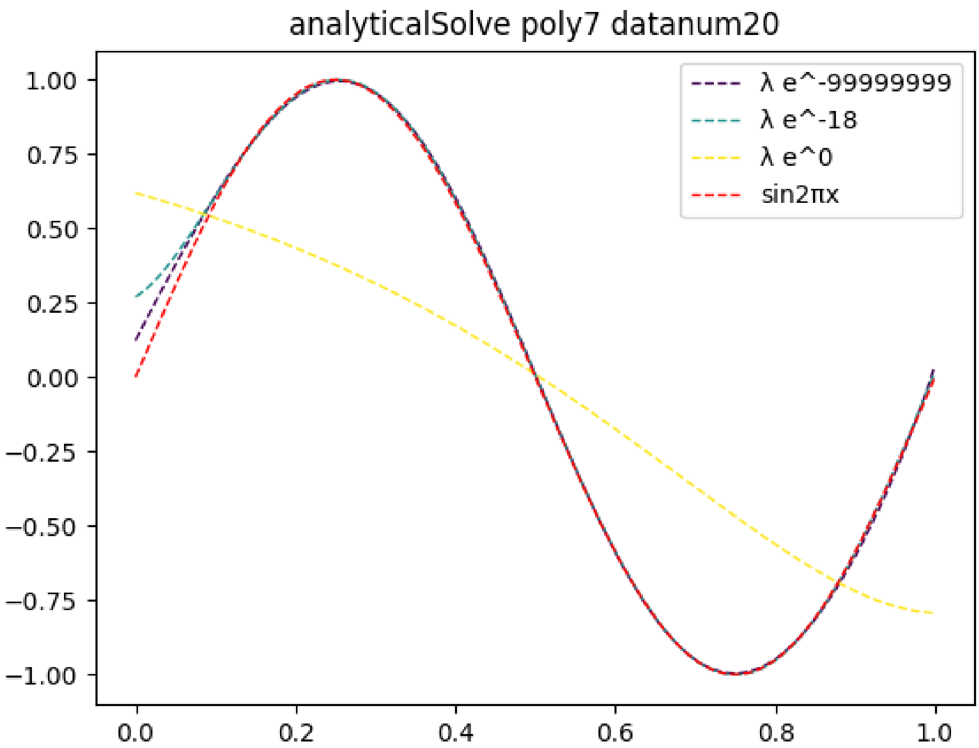
实现上和解析解法大同小异，没有特殊之处。

第 3 章 实验结果与分析

3.1 解析解

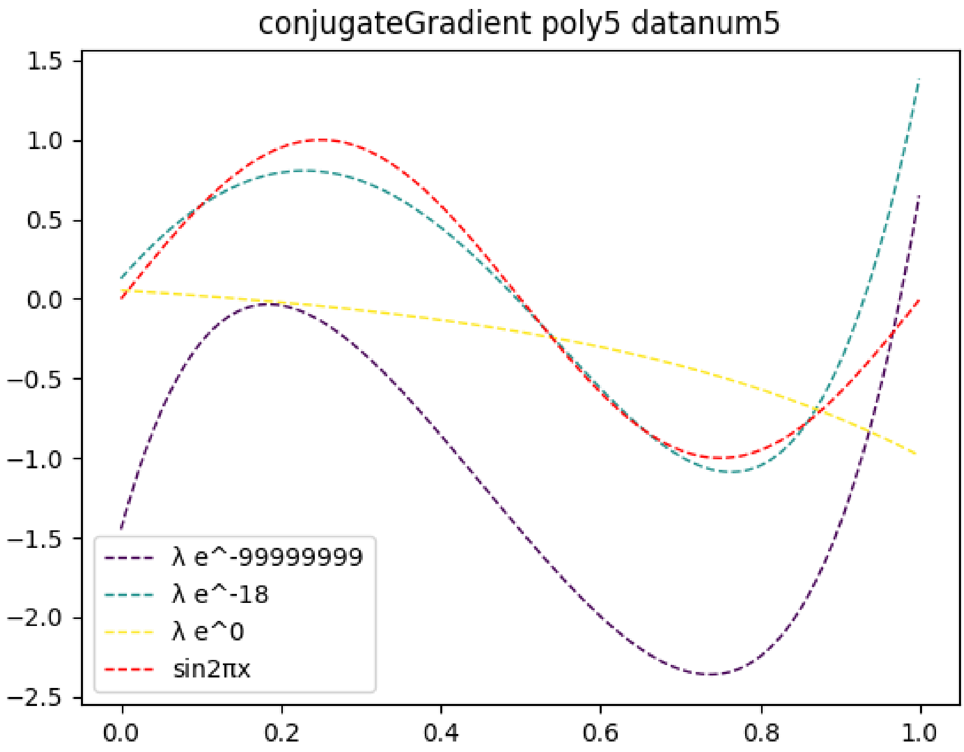


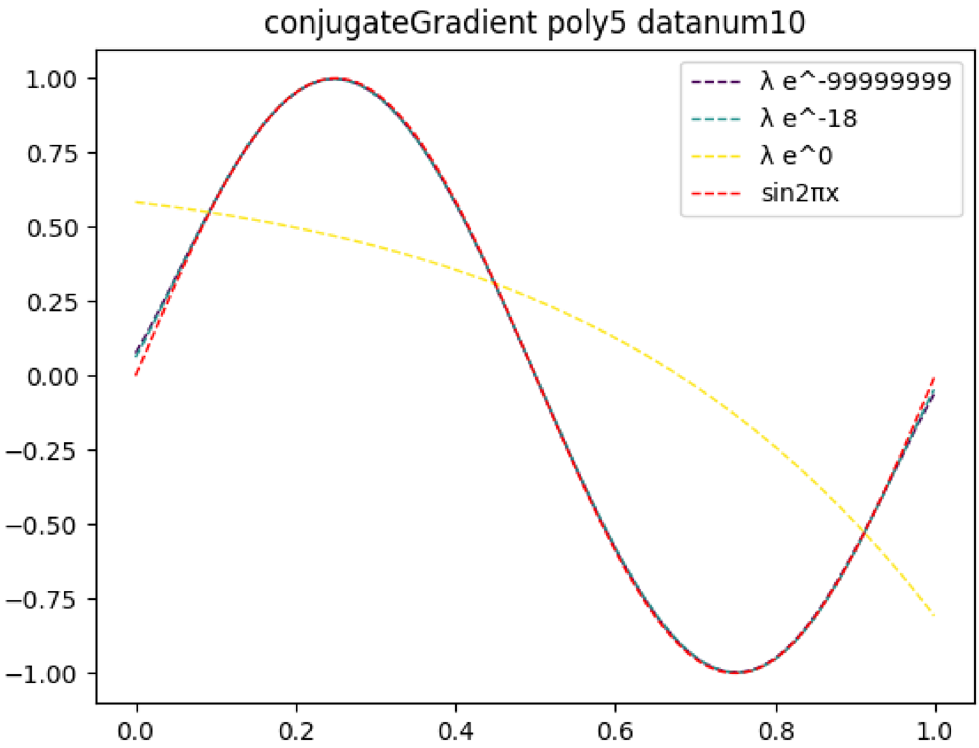

analyticalSolve poly7 datanum10
λ e^-99999999
λ e^-18
λ e^0
sin2πx
0
−2
−4
−6
0.0
0.2
0.4
0.6
0.8
1.0

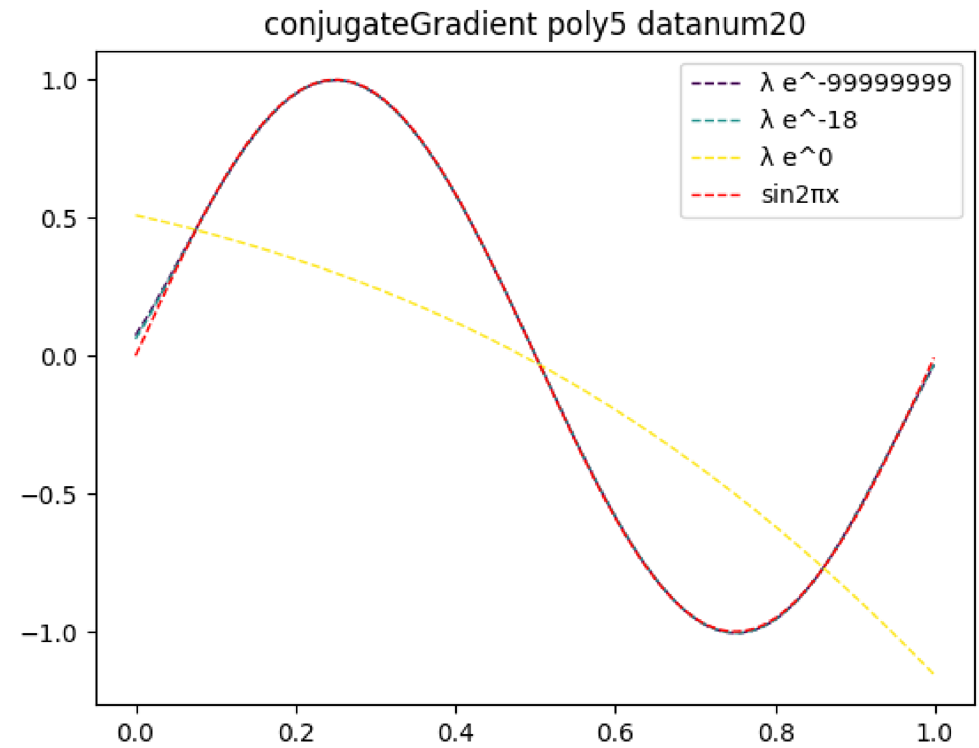

analyticalSolve poly7 datanum20
λ e^-99999999
λ e^-18
λ e^0
sin2πx
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

我们发现数据量相对多项式阶数越小，过拟合越严重。
Lambda 越大，曲线越平缓，但是过大的 lambda 会导致欠拟合。

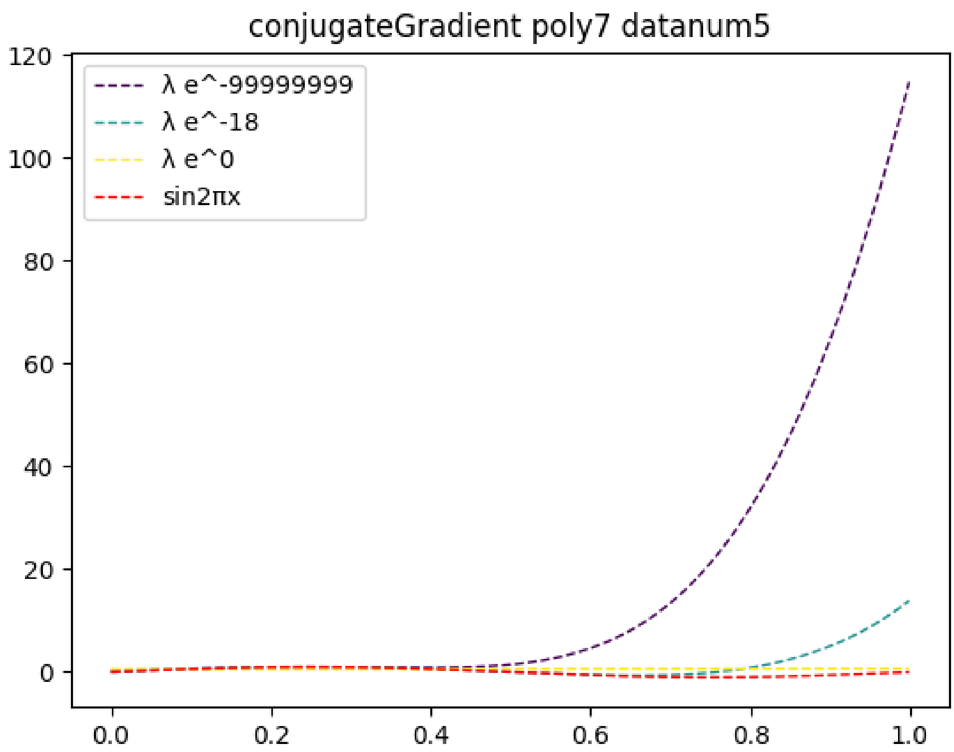
3.2 共轭梯度

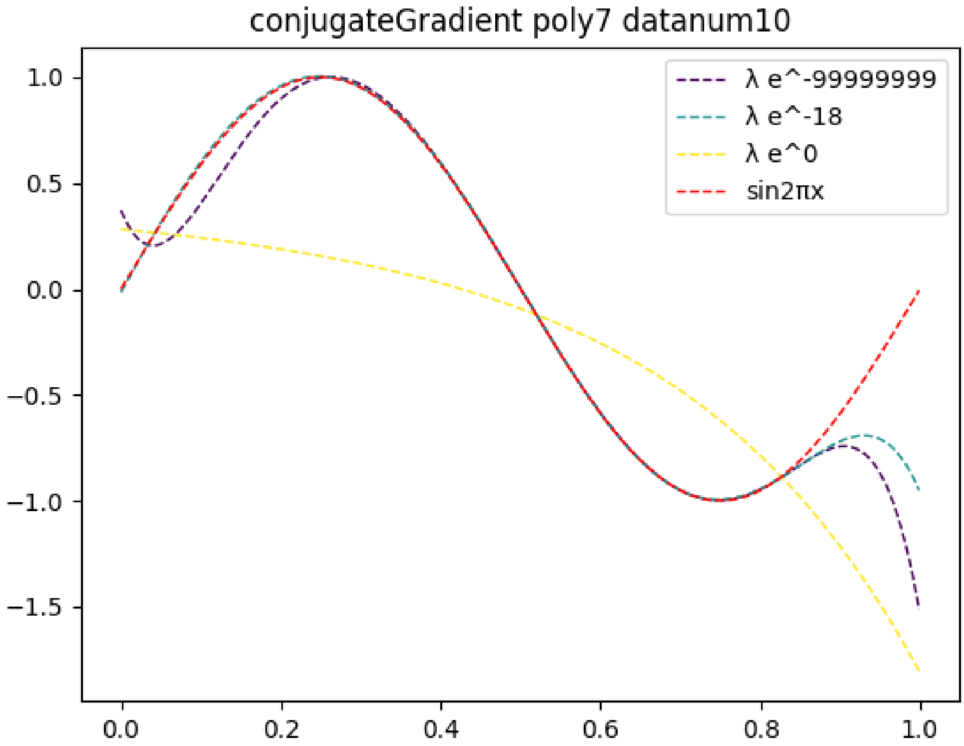


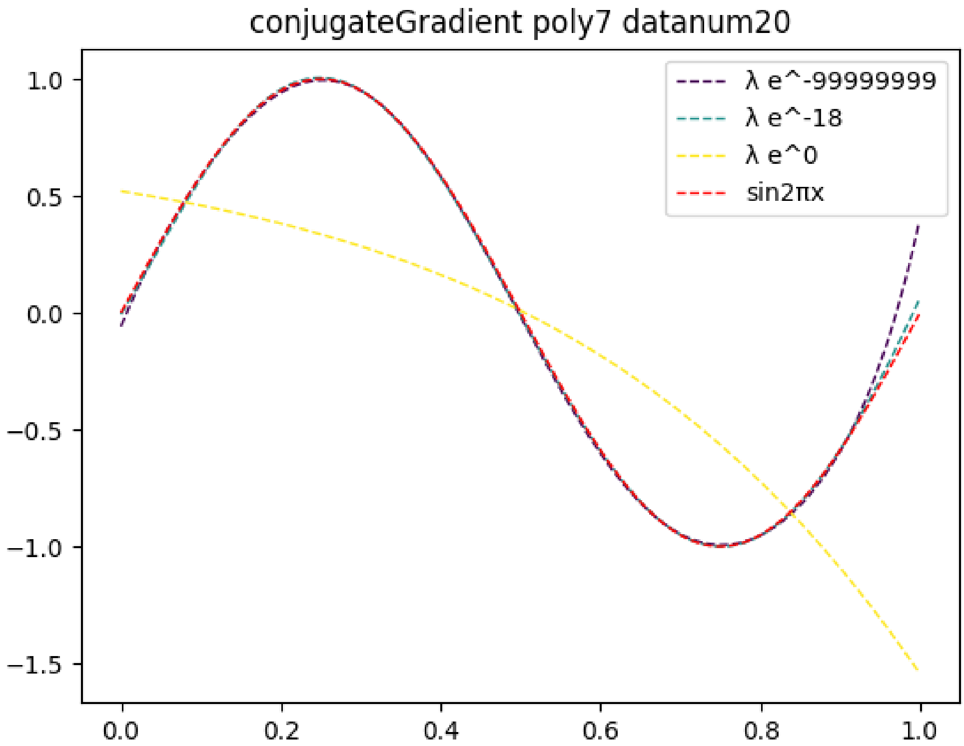

conjugateGradient poly5 datanum10
λ e^-99999999
λ e^-18
λ e^0
sin2πx
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0


conjugateGradient poly5 datanum20
λ e^-99999999
λ e^-18
λ e^0
sin2πx
1.0
0.5
0.0
−0.5
−1.0
0.0
0.2
0.4
0.6
0.8
1.0

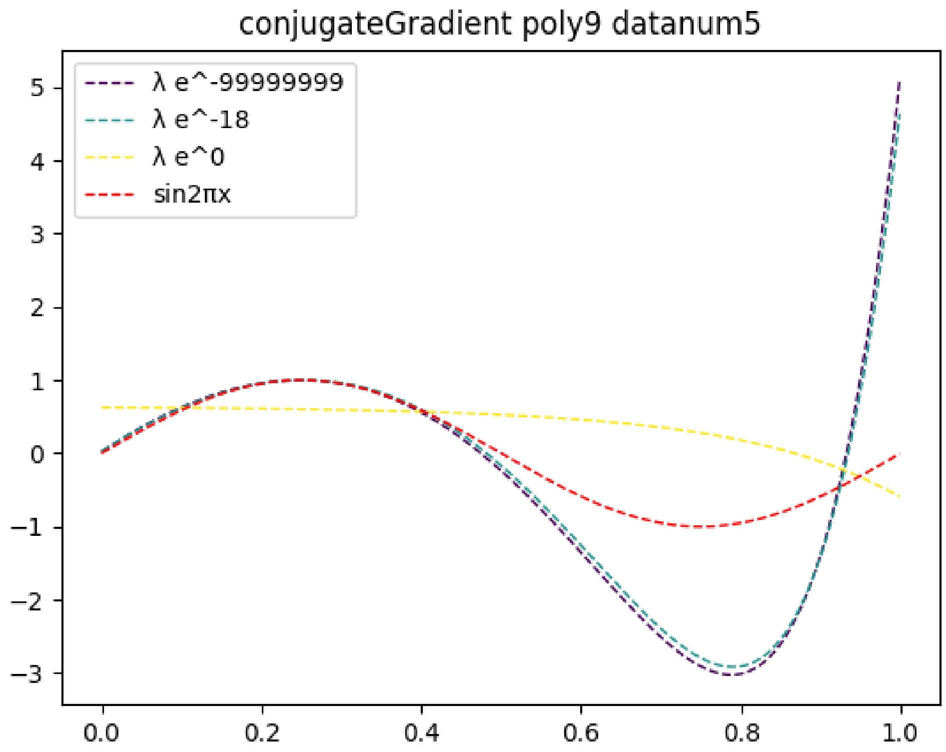
针对次数较低的情况，我们发现共轭梯度拟合的非常好，即使加上正则项影响也不大。

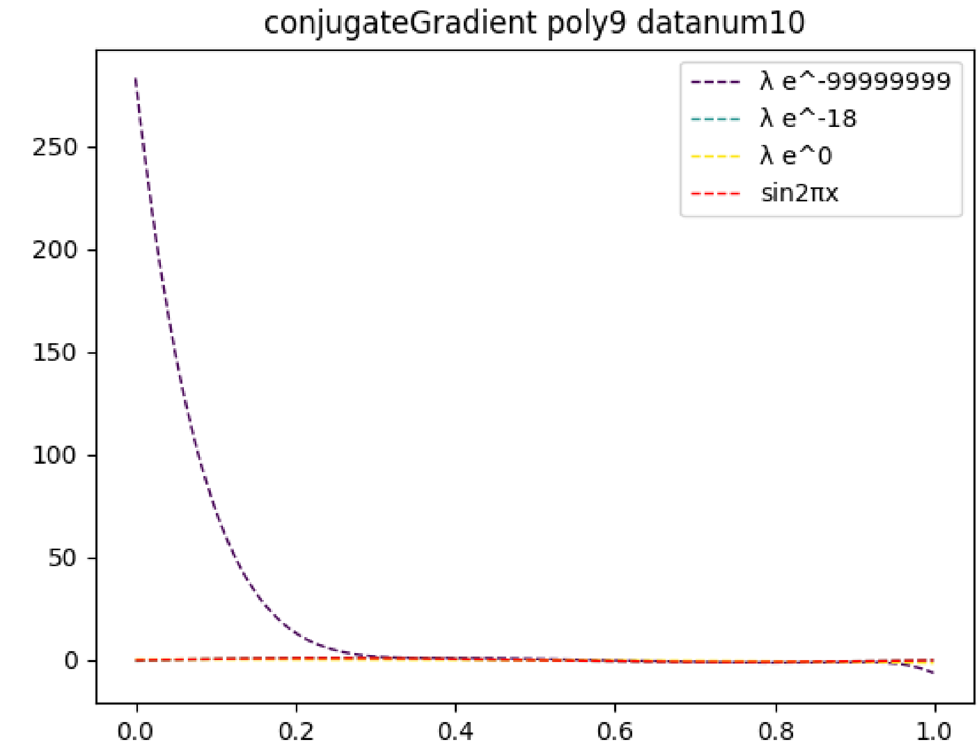


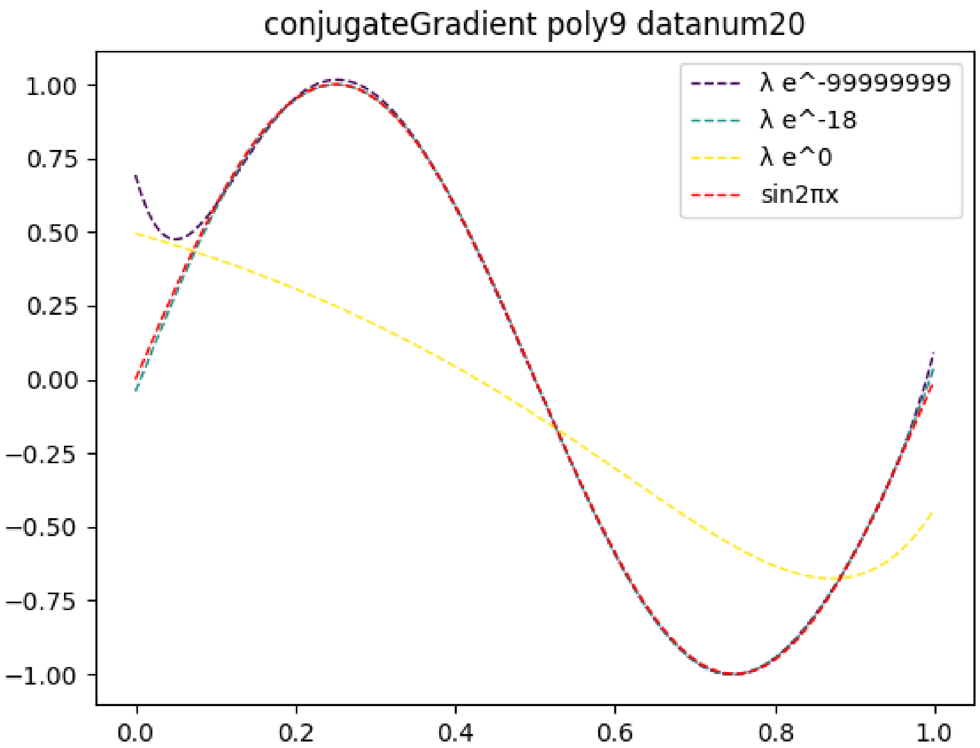

conjugateGradient poly7 datanum10
λ e^-99999999
λ e^-18
λ e^0
sin2πx
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.2
0.4
0.6
0.8
1.0


conjugateGradient poly7 datanum20
λ e^-99999999
λ e^-18
λ e^0
sin2πx
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.2
0.4
0.6
0.8
1.0

多项式阶数升高，过拟合逐渐严重，并且惩罚项影响力增大，出现欠拟合。

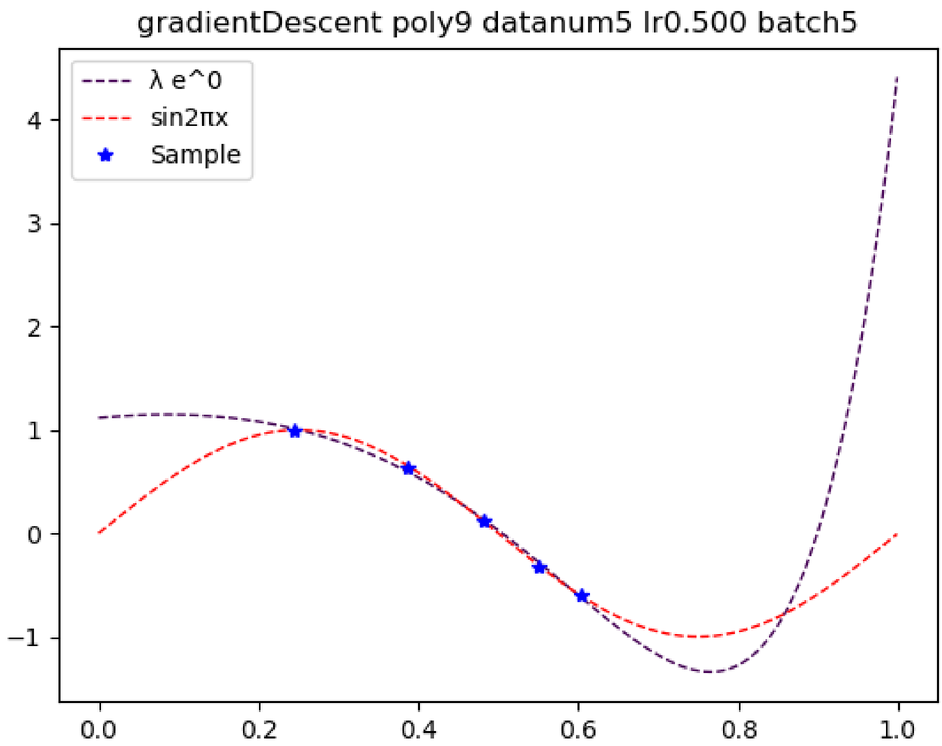


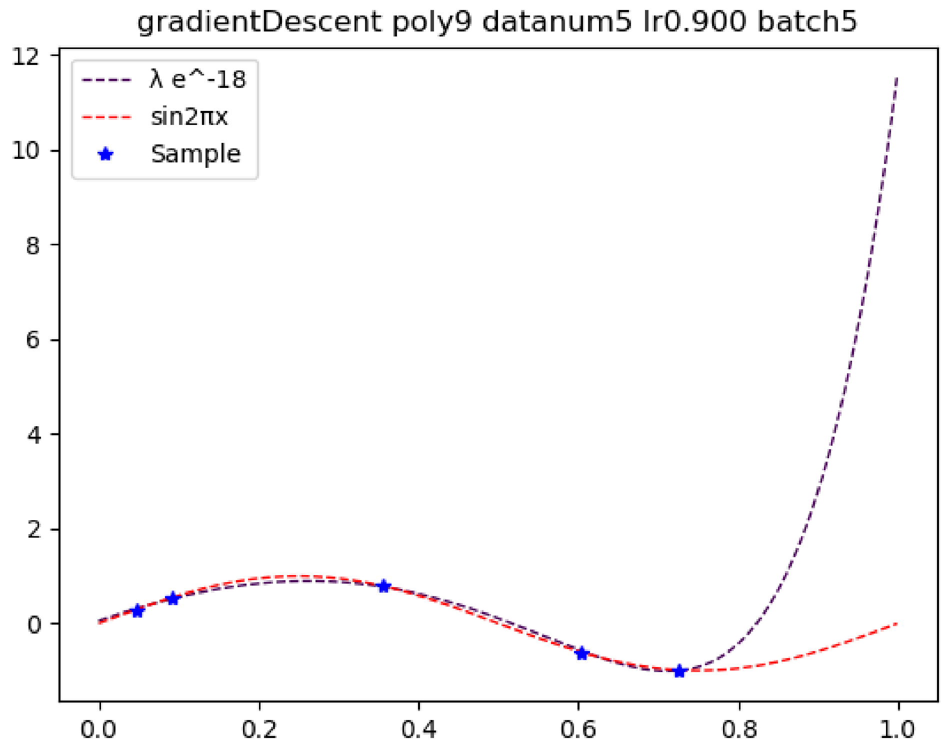

conjugateGradient poly9 datanum10
λ e^-99999999
λ e^-18
λ e^0
sin2πx
250
200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0

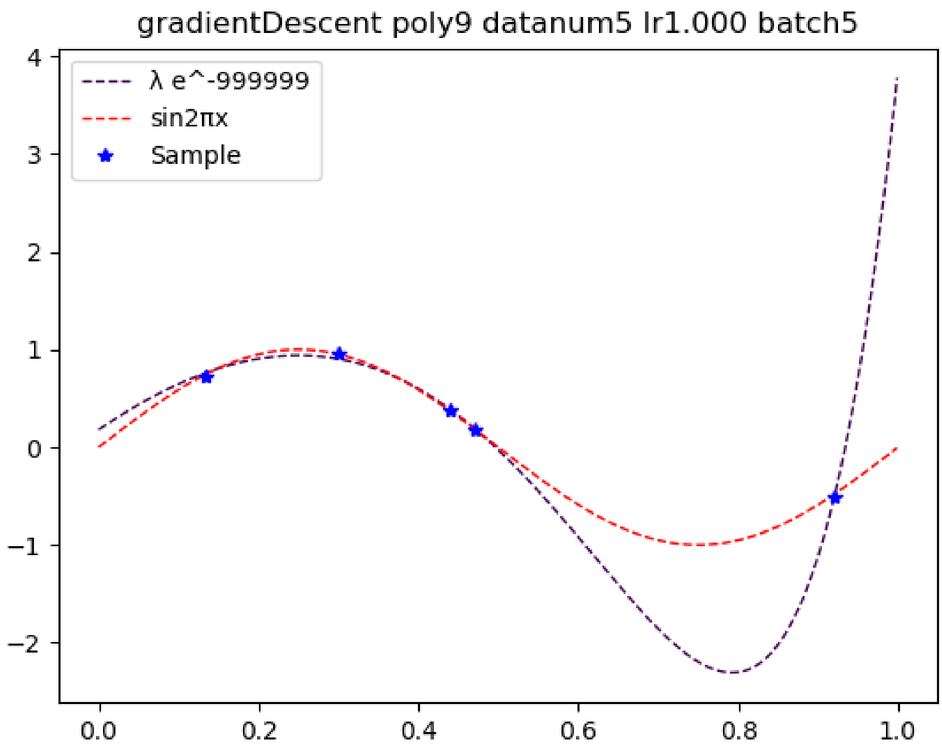

conjugateGradient poly9 datanum20
λ e^-99999999
λ e^-18
λ e^0
sin2πx
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0.0
0.2
0.4
0.6
0.8
1.0

我们看到不同的样本分布对于曲线拟合的巨大影响。

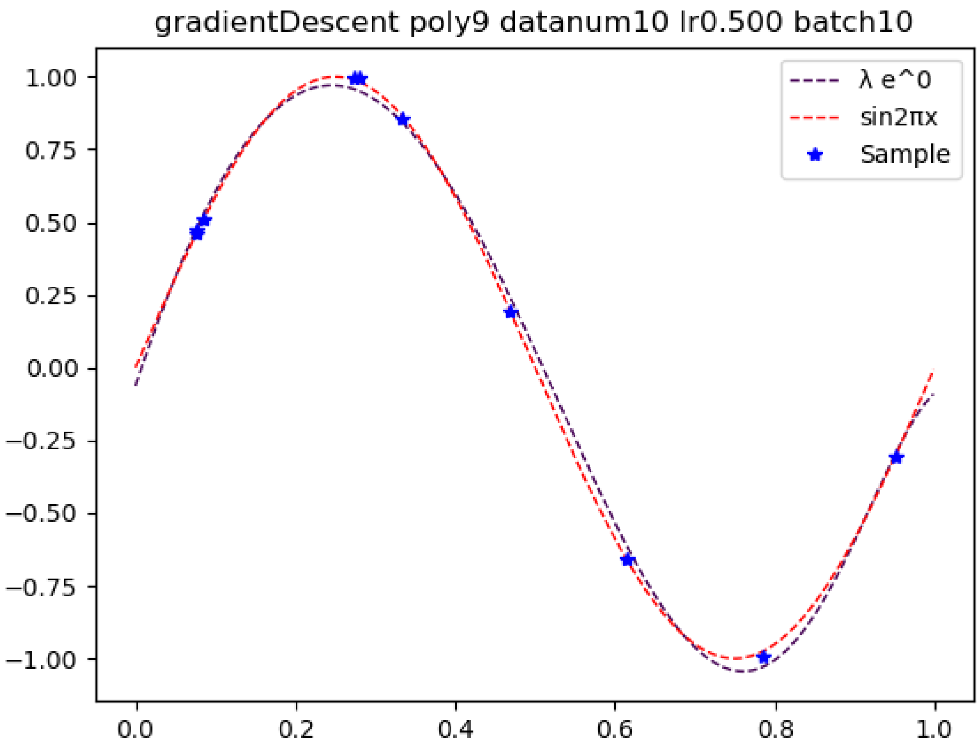
3.3 MBGD 梯度下降

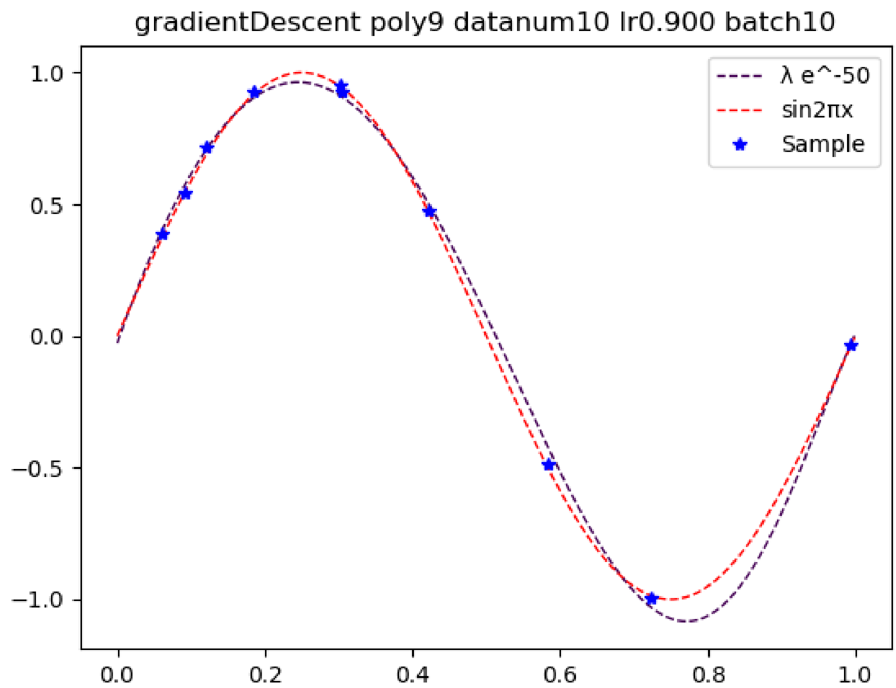


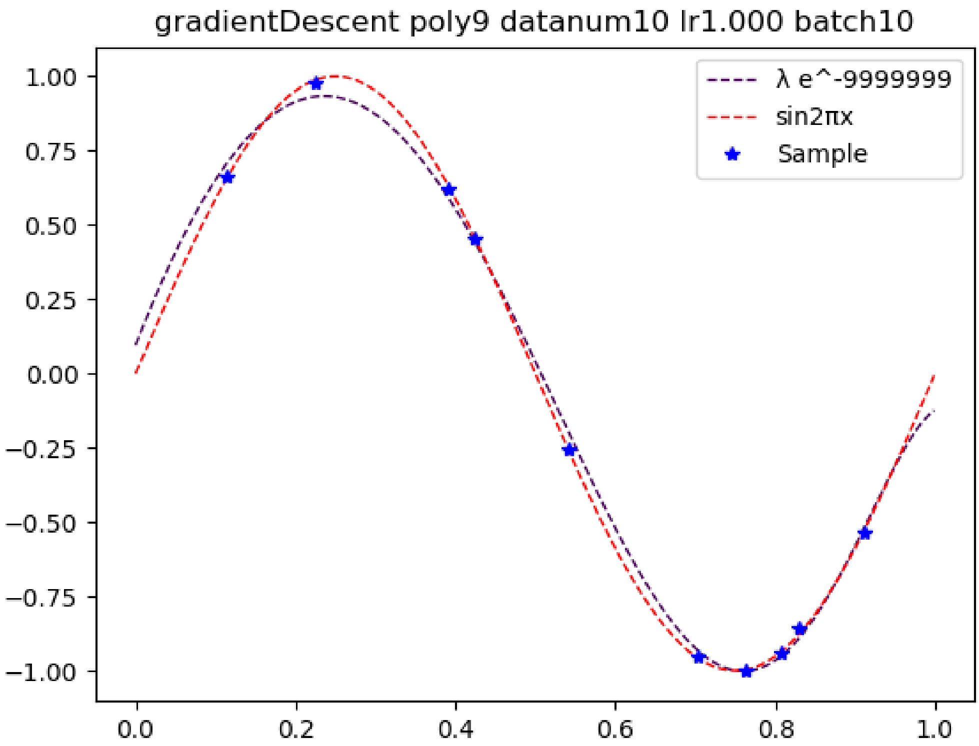

gradientDescent poly9 datanum5 lr0.900 batch5
λ e^-18
sin2πx
Sample
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0


gradientDescent poly9 datanum5 lr1.000 batch5
λ e^-999999
sin2πx
Sample
4
3
2
1
0
−1
−2
0.0
0.2
0.4
0.6
0.8
1.0

我们看到梯度下降法对于不同的惩罚项非常敏感，需要不断调节学习率。
数据量低的时候即使使用了较高的惩罚项系数，仍然局部存在过拟合现象。







我们看到对于大数据量，样本的超参数起的作用变小了。

第 4 章 结论

数据量和多项式阶数对过拟合的影响：

数据量越大，过拟合越低，数据量小越可能出现过拟合，按 bayesian 的观点来看，就是数据占比增大，先验知识占比减小。

惩罚项对过拟合的影响：

惩罚项本身就是为了抑制过拟合而采取的手段，但是它也可能出现欠拟合，往往需要进行模型选择来确定最终选择的惩罚项系数。

梯度下降学习率：

影响学习率的因素有很多，batch、多项式次数、惩罚项系数。所以学习率的选择是 trick 的。实验中有存在 loss 随着学习过程反而爆炸式增长的情况，这是因为学习率过高，导致不断上滑。还有 loss 不改变的情况，是因为 lr 设置过低。也有 lr 特别低的情况，要求到 10**-50，这是因为在计算惩罚项引入的梯度时，没有除 batch。总之 lr 的选择比较“不智能”，目前已经有更进一步的方法来自适应学习率，在实验中也进行了尝试，效果不是特别稳定。

第 5 章 参考文献

References

Avriel, M. (2012). *Nonlinear Programming*. Dover Publications.

Avriel, M. (2012). *Nonlinear Programming*. Dover Publications.

BISHOP, C. (2016). *PATTERN RECOGNITION AND MACHINE LEARNING*. [S.l.]: SPRINGER-VERLAG NEW YORK.

BISHOP, C. (2016). *PATTERN RECOGNITION AND MACHINE LEARNING*. [S.l.]: SPRINGER-VERLAG NEW YORK.

En.wikipedia.org. (2018). *Conjugate gradient method*. [online] Available at: https://en.wikipedia.org/wiki/Conjugate_gradient_method [Accessed 28 Sep. 2018].

En.wikipedia.org. (2018). *Conjugate gradient method*. [online] Available at: https://en.wikipedia.org/wiki/Conjugate_gradient_method [Accessed 28 Sep. 2018].

En.wikipedia.org. (2018). *Conjugate residual method*. [online] Available at: https://en.wikipedia.org/wiki/Conjugate_residual_method [Accessed 28 Sep. 2018].

En.wikipedia.org. (2018). *Conjugate residual method*. [online] Available at: https://en.wikipedia.org/wiki/Conjugate_residual_method [Accessed 28 Sep. 2018].

En.wikipedia.org. (2018). *Gradient descent*. [online] Available at: https://en.wikipedia.org/wiki/Gradient_descent [Accessed 28 Sep. 2018].

En.wikipedia.org. (2018). *Gradient descent*. [online] Available at: https://en.wikipedia.org/wiki/Gradient_descent [Accessed 28 Sep. 2018].

En.wikipedia.org. (2018). *Krylov subspace*. [online] Available at: https://en.wikipedia.org/wiki/Krylov_subspace [Accessed 28 Sep. 2018].

En.wikipedia.org. (2018). *Krylov subspace*. [online] Available at: https://en.wikipedia.org/wiki/Krylov_subspace [Accessed 28 Sep. 2018].

附录 源代码（带注释）

GradientDescent.py

```
import sys

import numpy
from numpy import logspace, math
from numpy import mat, polyval

from DataGenerator import generateData
from Visualization import visualResultAndSampleAndTarget, visualPoly


def gradientDescent(n, X, T, lr, batch=1, lnLambada=None, maxItrTimes=10,
targetAverageRSS=5 * 10 ** -4):
    """
    MBGD 梯度下降法多项式拟合
    :param n:
    :param X:
    :param T:
    :param lr:
    :param batch:
    :param maxItrTimes: 最大迭代次数，默认为 sys.maxsize
    :param targetAverageRSS: 目标 RSS 均值，到达时停止迭代
    :return: W，次数由低到高
    """
    wSize = n + 1
    if lnLambada == None:

        lambada = 0
    else:
        lambada = math.e ** lnLambada
    count = 0
    W = [5 for i in range(wSize)]
    mins = 1000
    batchX = []
    batchT = []
```

```
    for i in range(maxItrTimes):
        for x, t in zip(X, T):
            batchX.append(x)
            batchT.append(t)
            count += 1
            if count % batch == 0:
                gradient = getGradient(batchT, W, batchX,lambada)
                W = [w + lr * g / batch for w, g in zip(W, gradient)]
                rss = RSS(T, X, W, range=10, isaverage=True)

                if rss <= targetAverageRSS:
                    return rss, W

                print("%d %f %e" % (count, lr, rss))
                batchX = []
                batchT = []
        # visualPoly(*[W, type], isShow=True)
    return rss, W


def getGradient(T, W, X, lambada):
    W.reverse()
    rW = [0 for w in W]
    for t, x in zip(T, X):
        val = polyval(W, x)
        k = t - val
        rW = [rw + k * m for rw, m in zip(rW, logspace(0, len(W) - 1, len(W),
base=x))]
    W.reverse()
    rW = [rw + rw * lambada for rw in rW]
    return rW


def standgetGradient(T, W, X):
    """
    样例梯度下降，测试代码正确性用，不属于实验内容
    :param T:
    :param W:
    :param X:
    :return:
    """
```

```
    XX = mat([[x ** i for i in range(len(W))] for x in X])
    XXT = XX.T
    vectorW = mat(W).T
    vectorT = mat(T).T
    hypothesis = XX * vectorW
    loss = vectorT - hypothesis
    gradient = XXT * loss
    for i in range(len(W)):
        assert abs(getGradient(T, W, X)[i] - 
standgetGradient.T.tolist()[0][i]) < 0.0001

    return gradient.T.tolist()[0]


def RSS(T, X, W, range=-1, isaverage=False):
    sum = 0
    W.reverse()
    count = 0
    for t, x in zip(T, X):
        sum += numpy.square((t - polyval(W, x)))
        count += 1
        if range > 0 and count == range:
            break
    sum /= 2
    W.reverse()
    if isaverage:
        sum /= count
    return sum
```

AnalyticalSolution.py

```
import math

import numpy
from numpy import mat

from DataGenerator import generateData
from Visualization import visualPoly, visualResultAndSampleAndTarget
```

```
def analyticalSolve(n, X, T, lnLambada=None):
    '''
    解析解法
    :param n:多项式次数
    :param X:样本自变量 x
    :param T:样本因变量 t
    :param lnLambada:  lambada 的以自然数为底的对数，如果设为 None 则表示
lambada=0
    :return: 解析解拟合的次数由低到高的权重向量
    '''
    lenW = n + 1
    if lnLambada == None:

        lambada = 0
    else:
        lambada = math.e ** lnLambada

    XX = mat([[x ** i for i in range(lenW)] for x in X])
    vectorT = mat(T).T
    # print(XX,XX.max())

    XXT = XX.T
    return ((lambada * numpy.eye(lenW) + XXT * XX).I * XXT *
vectorT).T.tolist()[0]
```

ConjugateGradient.py

```
import math
from Visualization import visualResultAndSampleAndTarget
import matplotlib.pyplot as plt
import numpy
from numpy import mat, polyval, zeros

from DataGenerator import generateData
```

```python
def conjugateGradient(n, X, T, lnLambada=None, limit=0.00000001,
MaxIterationNum=100):
    """
    共轭梯度下降
    :param n 多项式次数
    :param X: 训练数据 x
    :param T: 训练数据 t
    :param limit:  当残差 r=B-AW 的内积 rT*r 小于 limit 时停止迭代
    :param lnLambada 惩罚参数的以 e 为底的对数，控制岭回归惩罚力度，若取
None 则不带惩罚项
    :param MaxIterationNum: 最大迭代次数
    :return: 次数由低到高的权重向量，迭代次数
    """
    lenW = n + 1
    if lnLambada == None:

        lambada = 0

    else:
        lambada = math.e ** lnLambada

    XX = mat([[x ** i for i in range(lenW)] for x in X])
    vectorT = mat(T).T
    A = XX.T * XX + lambada * numpy.eye(lenW)  # 带惩罚项
    B = XX.T * vectorT
    W = mat(zeros((lenW, 1)))
    r = B - A * W
    p = r.copy()
    num = 0
    while num < MaxIterationNum:
        num += 1
        alpha = (r.T * r / (p.T * A * p))[0, 0]
        W += alpha * p
        lastr = r.copy()
        r -= alpha * A * p
        if (r.T * r)[0, 0] < limit:
            break
        beta = (r.T * r / (lastr.T * lastr))[0, 0]
        p = r + beta * p
    return W.T.tolist()[0], num
```

DataGenerator.py

```
from math import sin, pi

import matplotlib.pyplot as plt
import numpy as np


def generateData(num):
    X=np.random.rand(num)

    S=np.random.randn(num)
    tmp=[sin( 2*pi * m) for m in X]
    T=[]
    for i in range(num):
        T.append(tmp[i]+S[i]/100)

    assert len(X)==len(T)
    return X,T
```

Visualization.py

```
"""
图形可视化。不属于实验内容
"""
def visualPoly(*WsandLabels,X=[],T=[],title="Data
Graph",savePath="None",isShow=False):
    """
    多曲线展示，默认增加sin2pix曲线。可添加样本点
    :param WsandLabels: 按次数由低到高排列，先W，全部W输入后接入按顺序
接入labels
    :param X,T: 样本
    :param title: 图像标题
    :param savePath: 保存路径
    :param  isShow: 是否展示图像
    """
    t=len(WsandLabels)//2
    print(len(WsandLabels))
    assert t*2==len(WsandLabels)
```

```
    cmap = plt.get_cmap('viridis')
    colors = cmap(numpy.linspace(0, 1, t))
    x = numpy.arange(0, 1, 0.001)
    for i,color in zip(range(t),colors):
        W=WsandLabels[i]
        label=WsandLabels[i+t]
        s = list(W)
        s.reverse()
        y = polyval(s, x)
        plt.plot(x, y, "--",c=color, linewidth=1,label=label)
    plt.plot(x, sin(2*math.pi*x), "r--", linewidth=1, label="sin2πx")
    if len(X)!=0 and len(T)!=0:
        plt.plot(X,T,"b*",linewidth=10,label="Sample")
    plt.title(title)  # 添加图形标题

    plt.legend()  # 展示图例
    if savePath !="None":
        plt.savefig(savePath+title+".png")
    if isShow:
        plt.show()
    plt.clf()


def visualResultAndSampleAndTarget(W,X,T):
    """
    适用于单曲线的简单可视化，功能不如 visualPoly 强大
    :param W:
    :param X:
    :param T:
    :return:
    """
    x = numpy.arange(0, 1, 0.001)
    s = list(W)
    s.reverse()
    y = polyval(s, x)
    plt.plot(x, y, 'r--', linewidth=1,label="W")
    plt.plot(X, T, 'g*', linewidth=10,label="Sample")
    plt.plot(x, [math.sin(2 * math.pi * x) for x in x], 'b--',
linewidth=1,label="sin2πx")
    plt.legend()
    plt.show()
```