

哈尔滨工业大学计算机科学与计算机学院
实验报告

课程名称: 计算建模

课程类型: 选修

实验题目: 多项式拟合正弦函数

学号: 1170300511

姓名: 易亚玲

一、实验目的

- 掌握最小二乘法求解（无惩罚项的损失函数）
- 掌握加惩罚项（2范数）的损失函数优化
- 掌握梯度下降法、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、实验要求及环境

实验要求：

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

实验环境：

- pycharm 2019.1
- python 3.7
- win10
- X86-64

三、设计思想

1. 算法原理

最小二乘法：又称最小平方法，是一种数学优化方法。它通过最小化误差的平方和寻找数据的最佳函数匹配。因此我们定义损失函数：

$$L(\omega) = \frac{1}{2N} \sum_{\omega} (y - X\omega)^2$$

其中

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} & 1 \\ x_{21} & x_{22} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{md} & 1 \end{pmatrix} = \begin{pmatrix} x_1^T & 1 \\ x_2^T & 1 \\ \vdots & \vdots \\ x_m^T & 1 \end{pmatrix}$$

于是求解 ω 使得

$$\hat{\omega}^* = \underset{\hat{\omega}}{\operatorname{argmin}} (y - X\hat{\omega})^T (y - X\hat{\omega})$$

令 $E_{\hat{\omega}} = (y - X\hat{\omega})^T (y - X\hat{\omega})$, 对 $\hat{\omega}$ 求导得到

$$\frac{\partial E_{\hat{\omega}}}{\partial \hat{\omega}} = \frac{1}{N} X^T (X\hat{\omega} - y)$$

令上式 (在梯度下降法中, 上面这个式子也是梯度) 等于0可得

$$(X^T X)\hat{\omega}^* = X^T y$$

如果 $X^T X$ 可逆, 则可以直接求解 $\omega = (X^T X)^{-1} X^T y$ 得到解析解;

或者直接运用共轭梯度法求解上面这个方程。

另外, 我们在求解析解时, 可以在损失函数后面加上 $\|\omega\|_2$ 作为惩罚项, 比例因子记为 λ , 防止模型太复杂而产生过拟合。加入惩罚项的损失函数为

$$L(\omega) = \frac{1}{2N} \sum_{\omega} (y - X\omega)^2 + \frac{\lambda}{2} \|\omega\|_2$$

利用解析解的方式可以求得

$$\omega = (X^T X + \lambda NI)^{-1} X^T y$$

因为 $X^T X + \lambda NI$ 一定可逆, 我们可以用这种方式求带惩罚项的解析解

2. 算法的实现

1. 生成噪声: 调用random库, 生成高斯噪声

```
x[i] += random.gauss(mu, sigma)
y[i] += random.gauss(mu, sigma)
```

2. 求解析解直接套用算法原理中的公式即可

3. 梯度下降法: 当梯度的2范数小于我们设置的阈值e时, 结束迭代 (其中lambda是梯度下降法的学习率)

```
while 1:
    g = gradient(X, w, y)
    if np.dot(g, g) < e:
        break
    w += (-lambda) * g
```

4.共轭梯度下降法：关键代码如下

```
while 1:  
    if np.dot(r.T, r) == 0 or np.dot(np.dot(p.T, A).T, p) == 0:  
        break  
    a = np.dot(r.T, r) / np.dot(np.dot(p.T, A).T, p)  
    w += a*p  
    r1 = r - np.dot(a*A, p)  
    beta = np.dot(r1.T, r1)/np.dot(r.T, r)  
    p = r1 + beta*p  
    r = r1  
loss = calLoss(w, y, X, dataAmount)      https://blog.csdn.net/qq_43162058
```

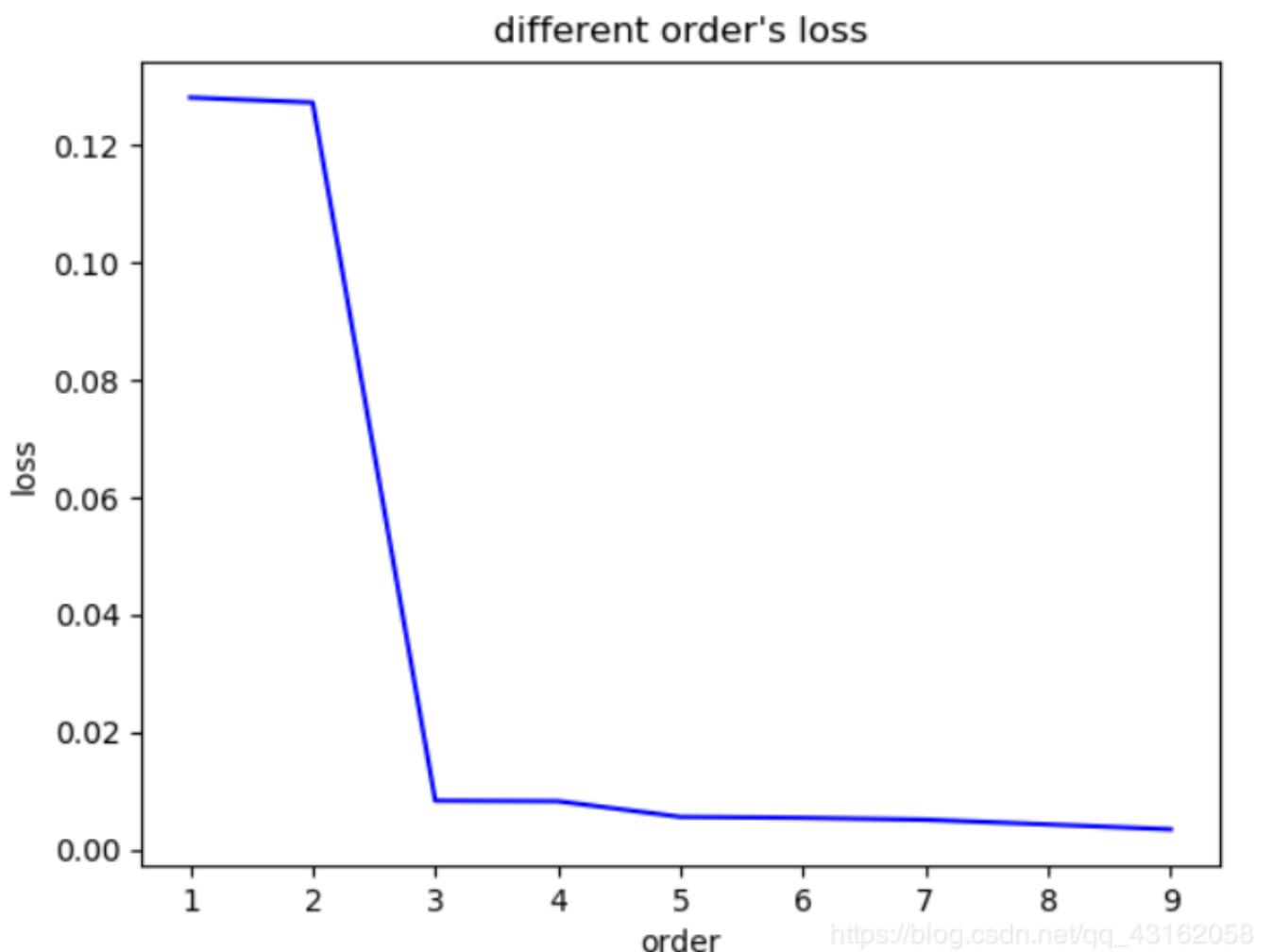
四、实验结果与分析

分析一：以20数据量为例，分析不同方法对于不同阶的拟合程度

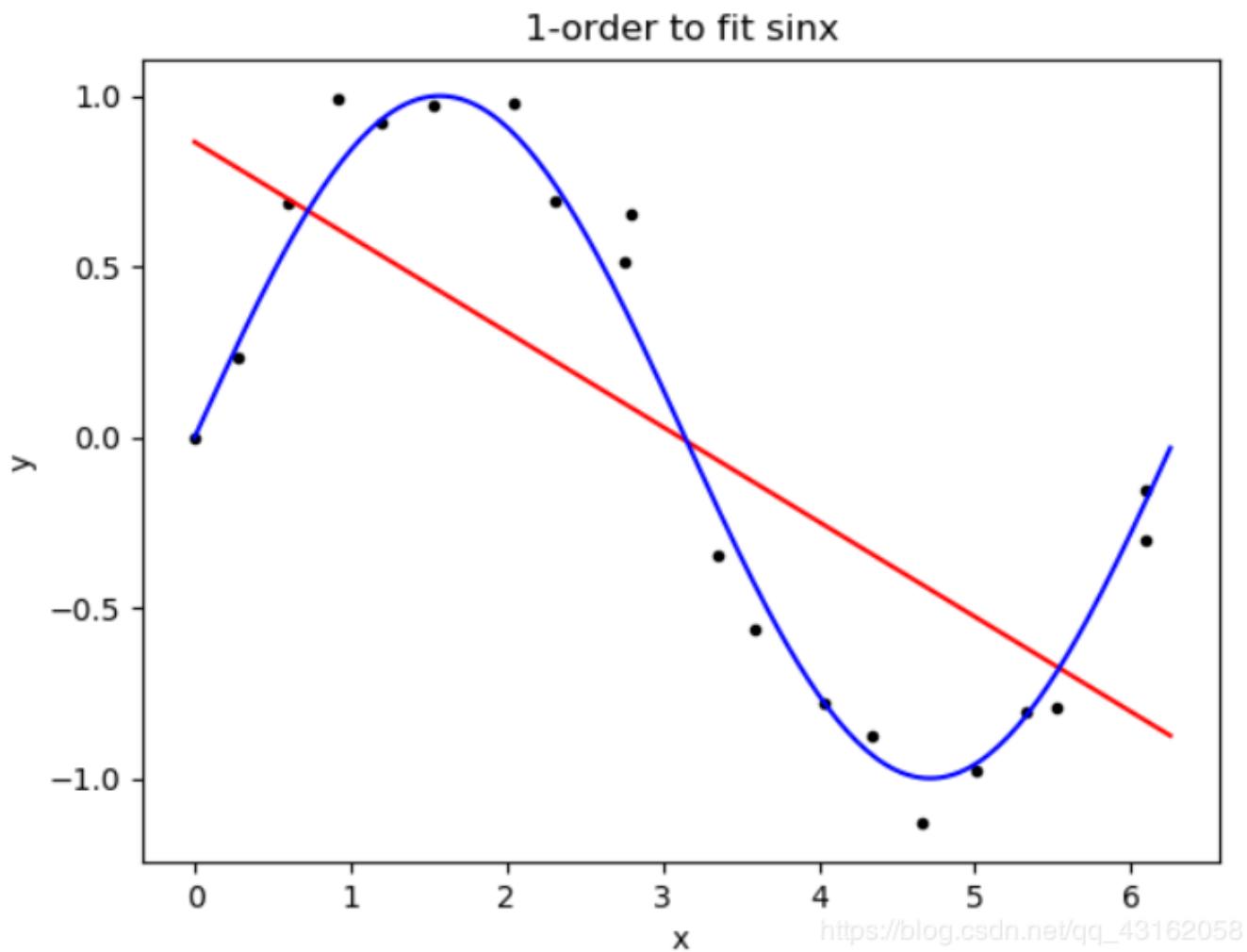
4.1解析式法（不带正则项）

4.1.1研究数据量为20时不同阶数的拟合

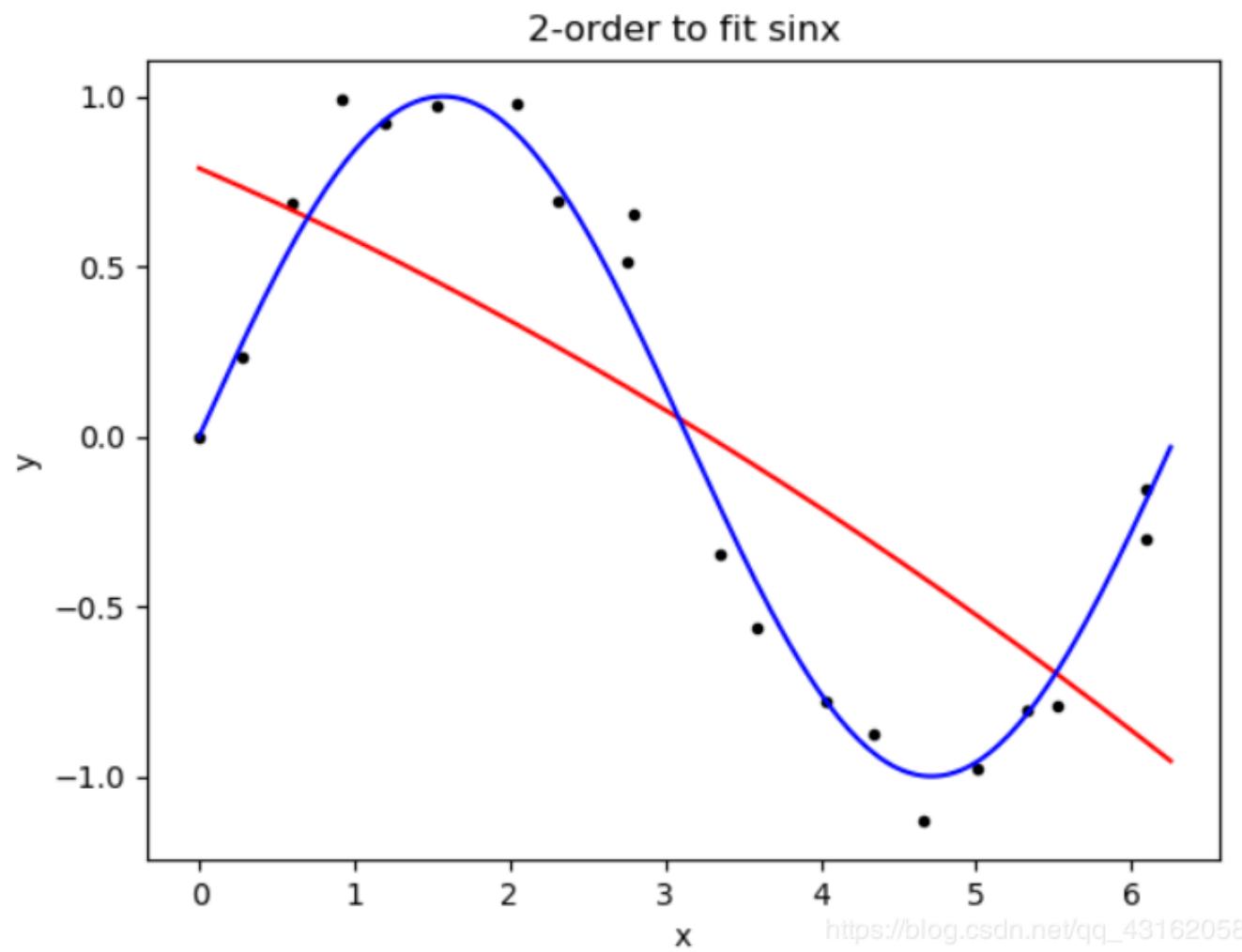
不同阶数的损失，可以看出3阶以上的损失已经很小了



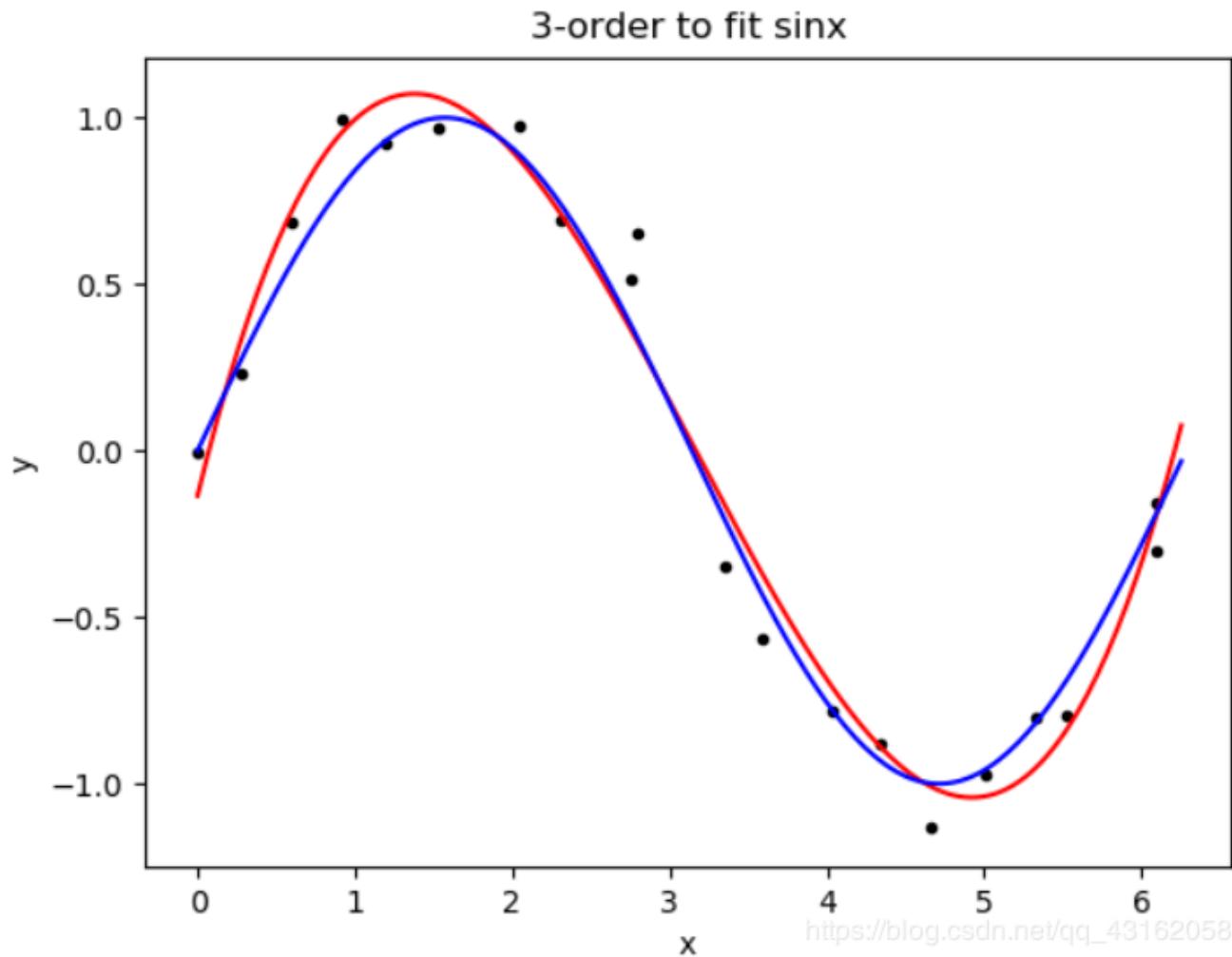
一阶



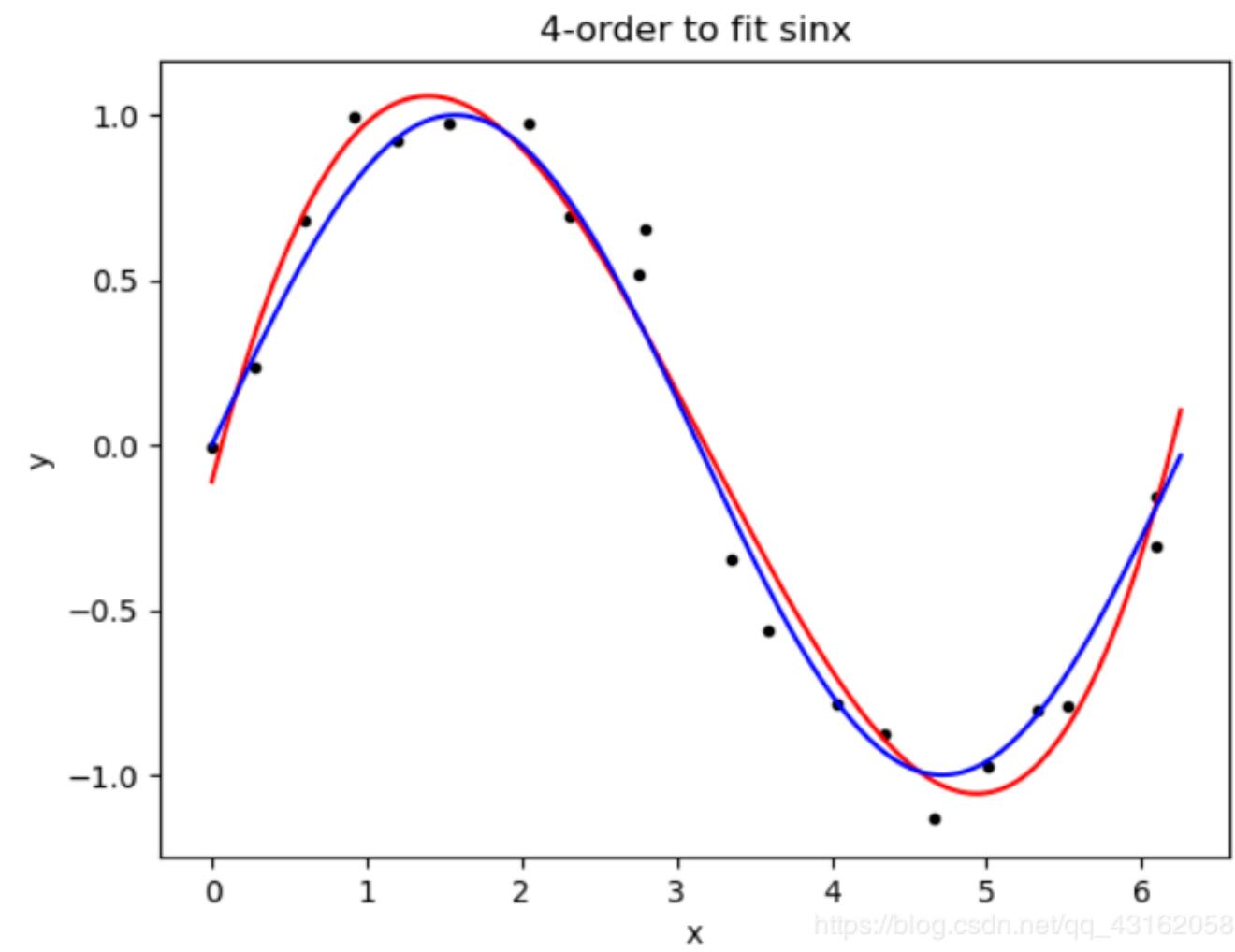
二阶



三阶

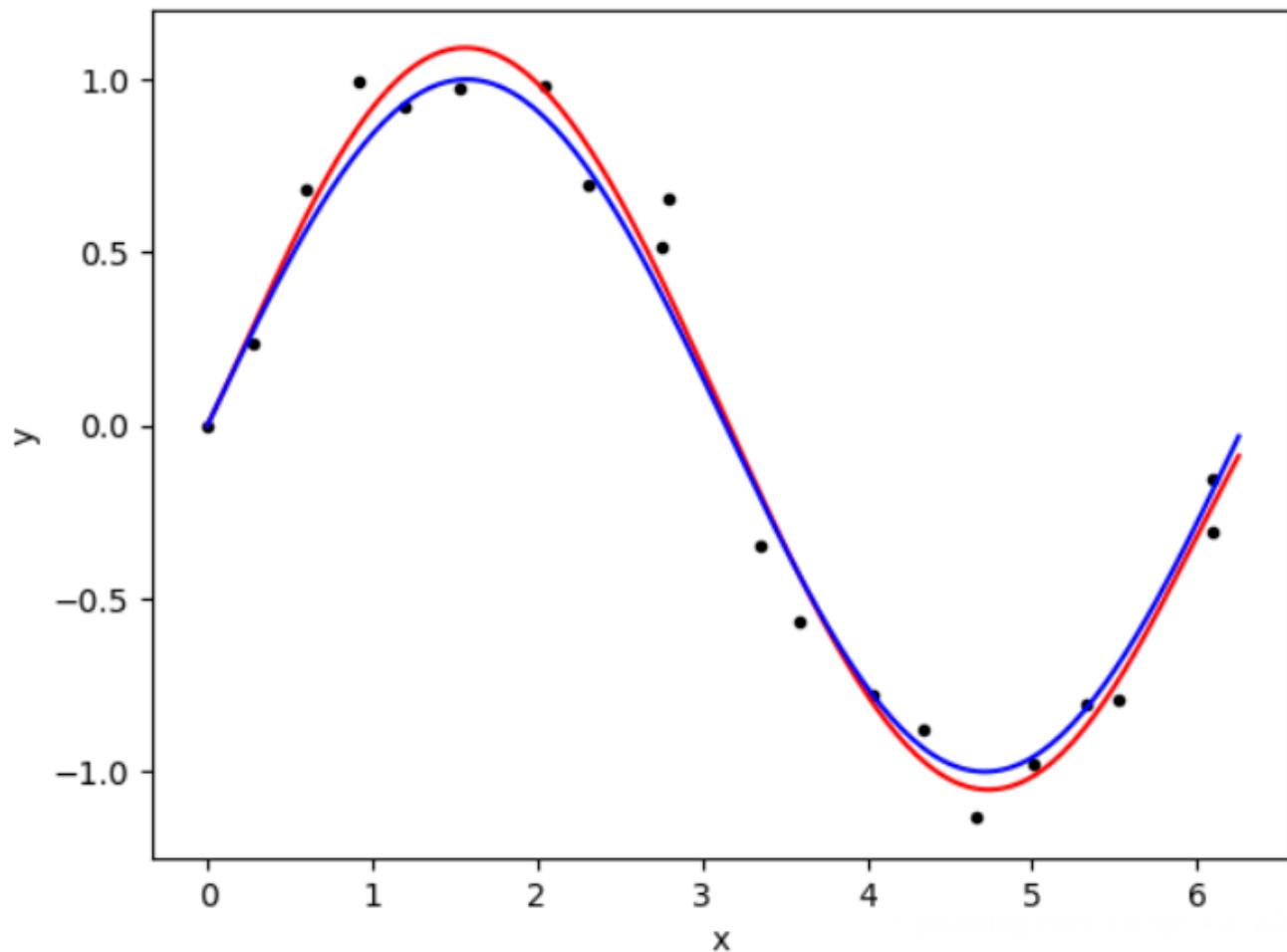


四阶



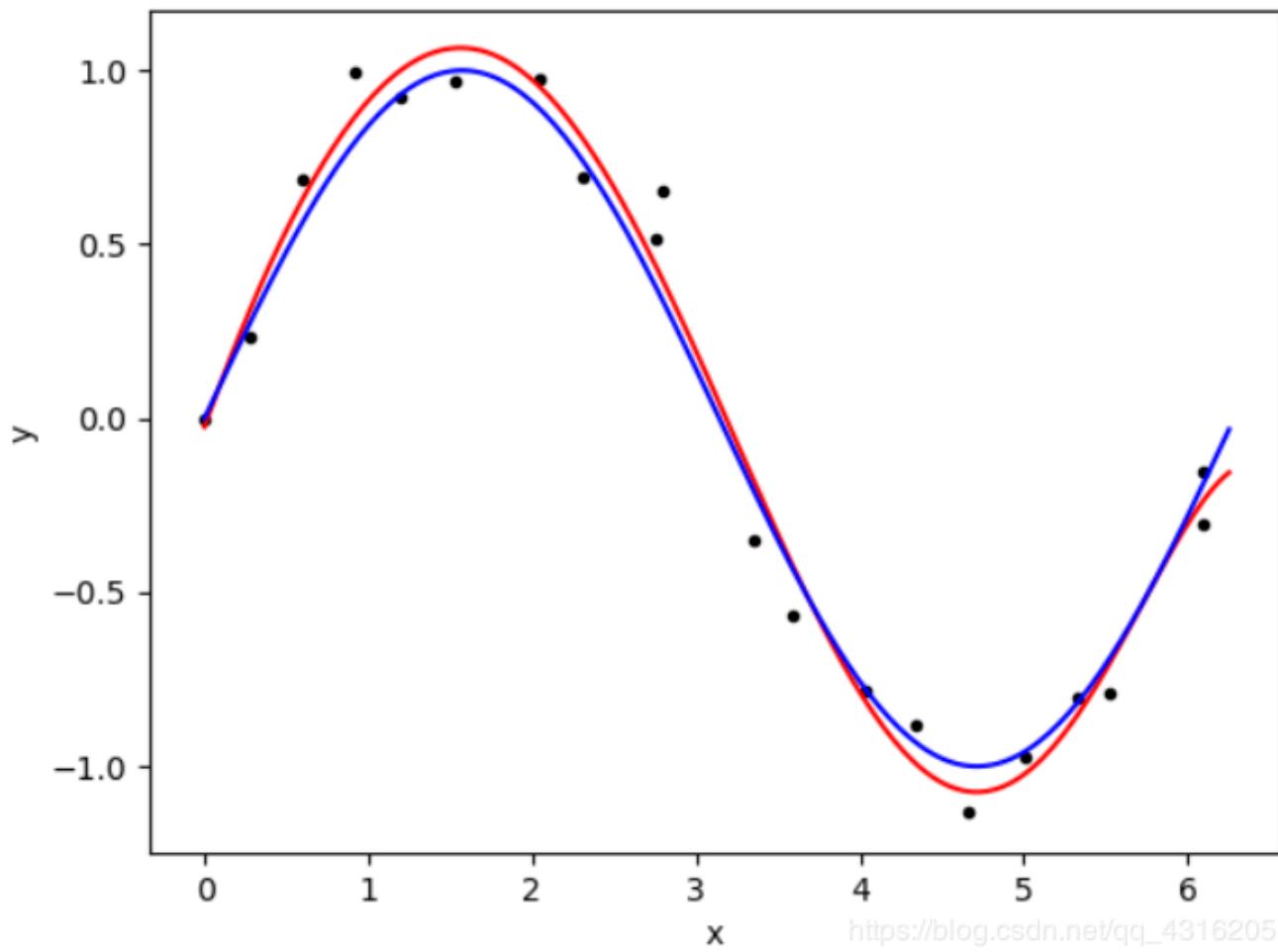
五阶

5-order to fit $\sin x$



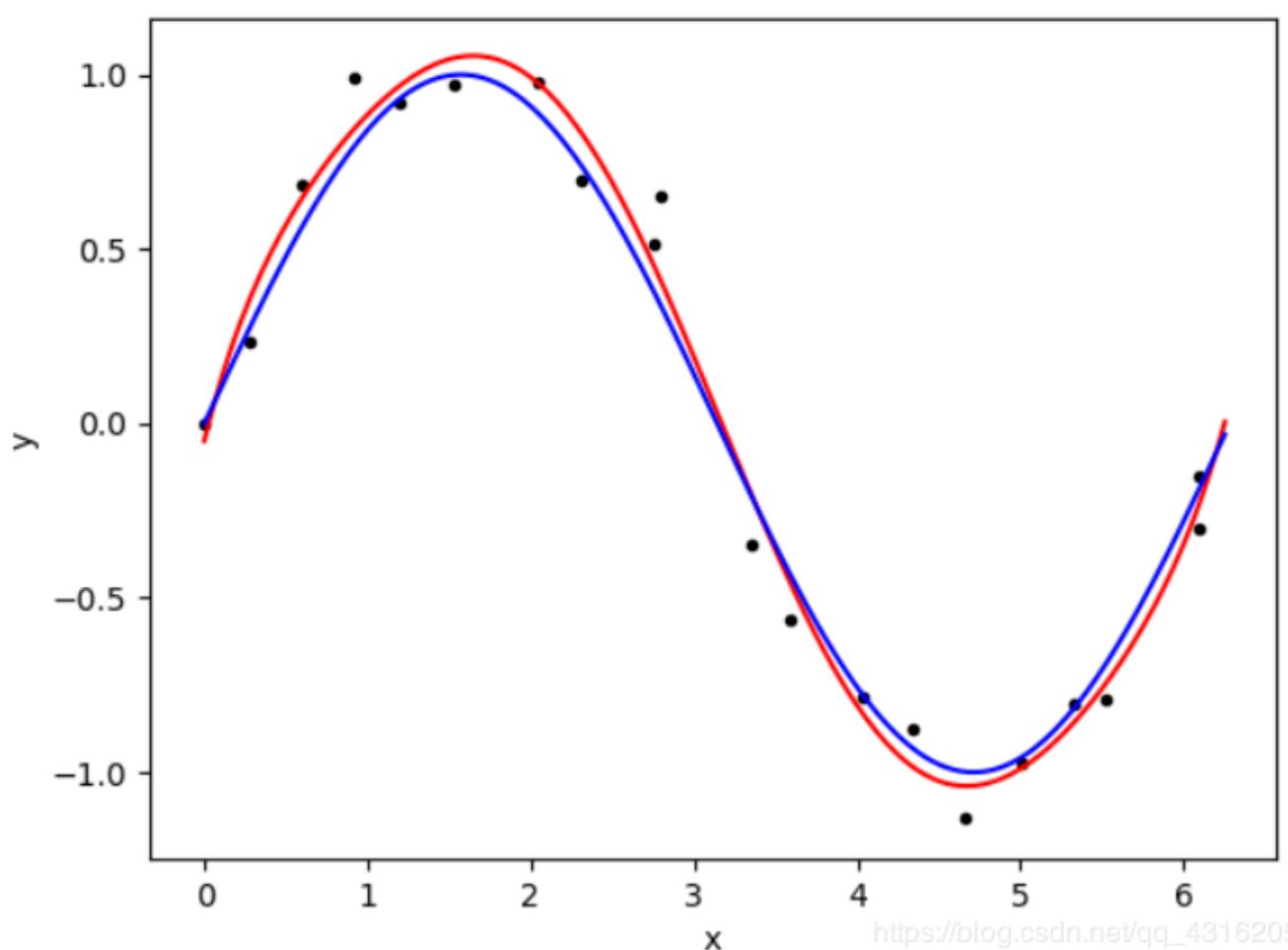
六阶

6-order to fit $\sin x$



七阶

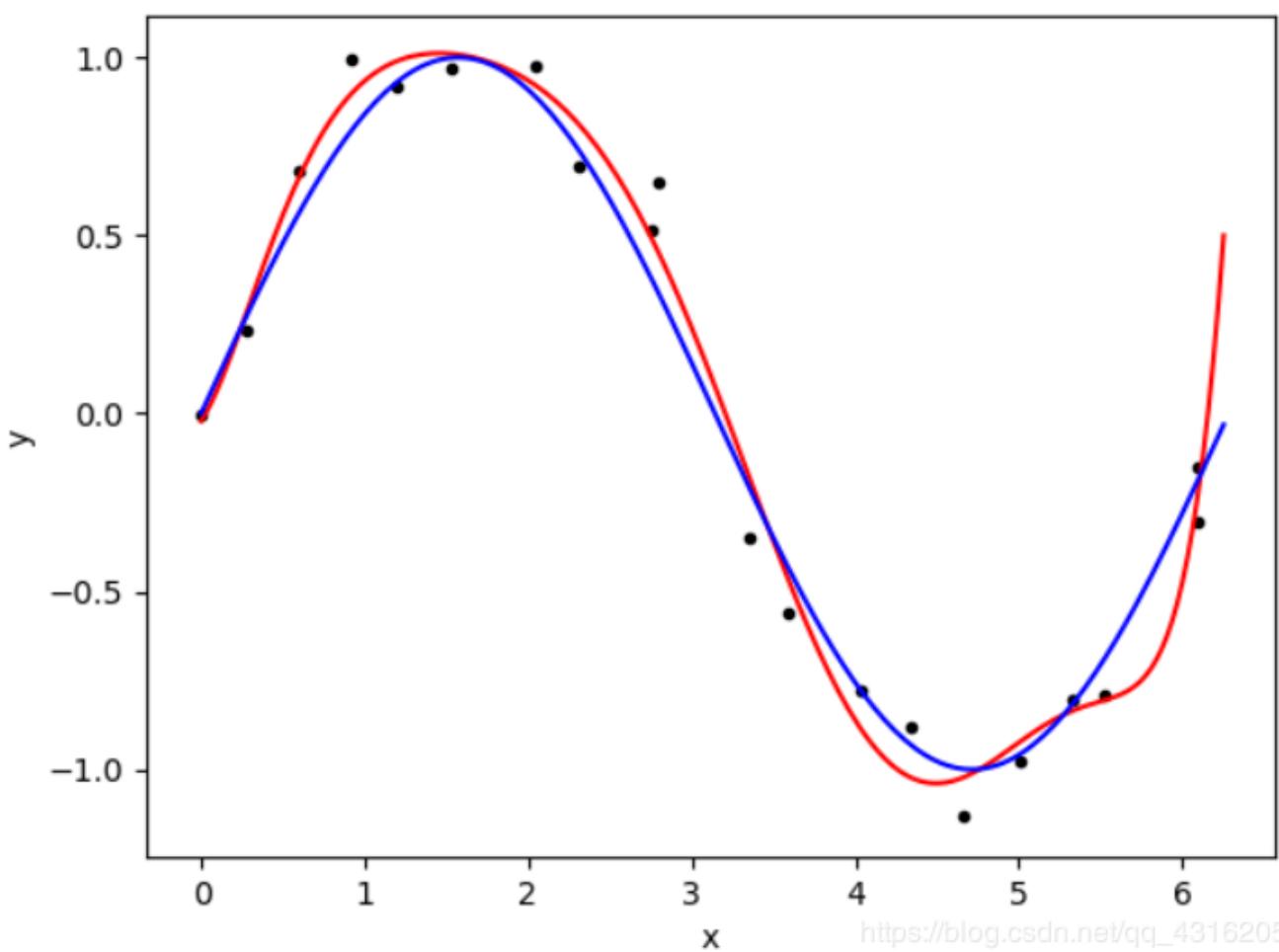
7-order to fit sinx



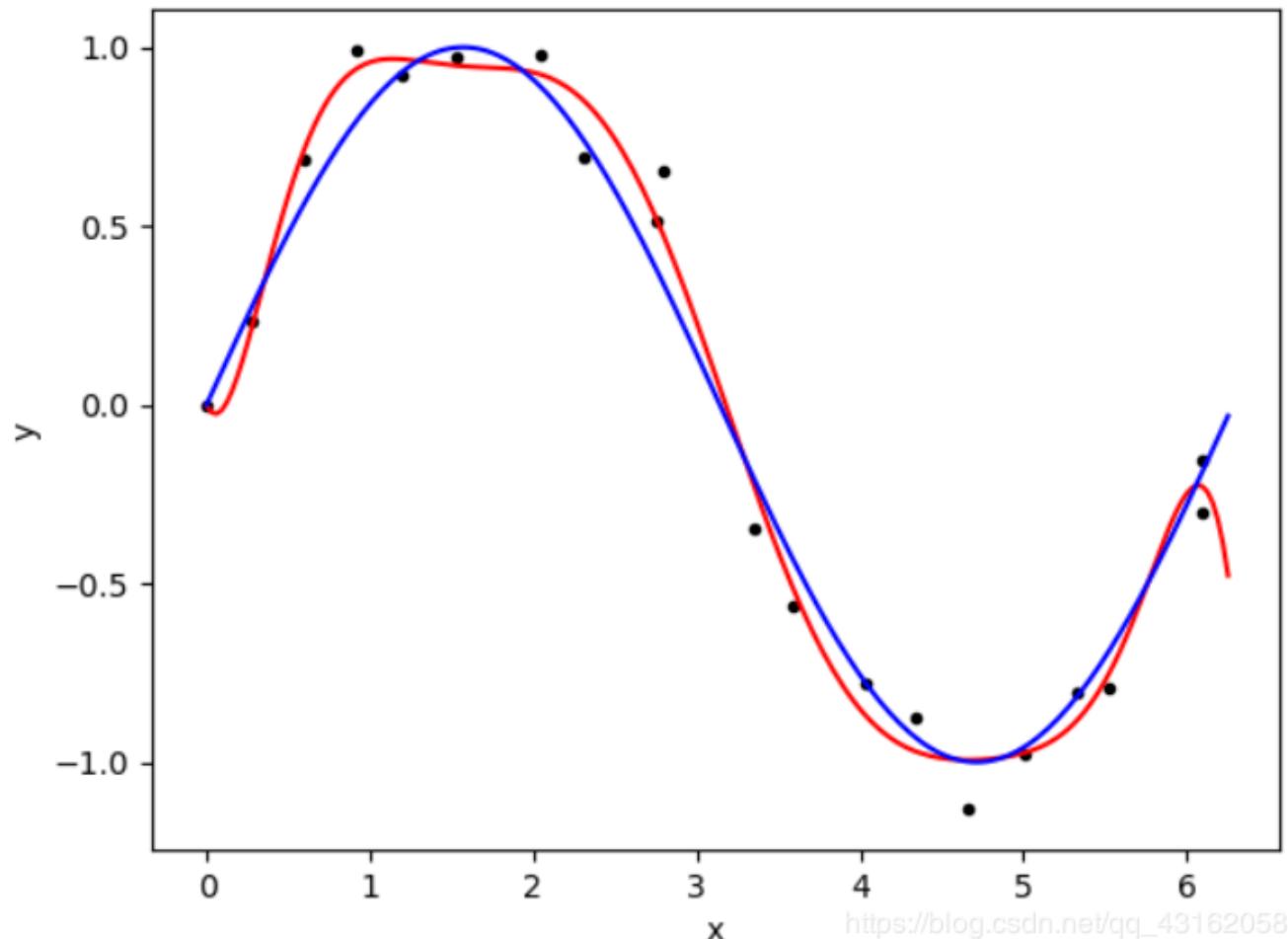
https://blog.csdn.net/qq_43162058

八阶 (明显过拟合了)

8-order to fit sinx



https://blog.csdn.net/qq_43162058

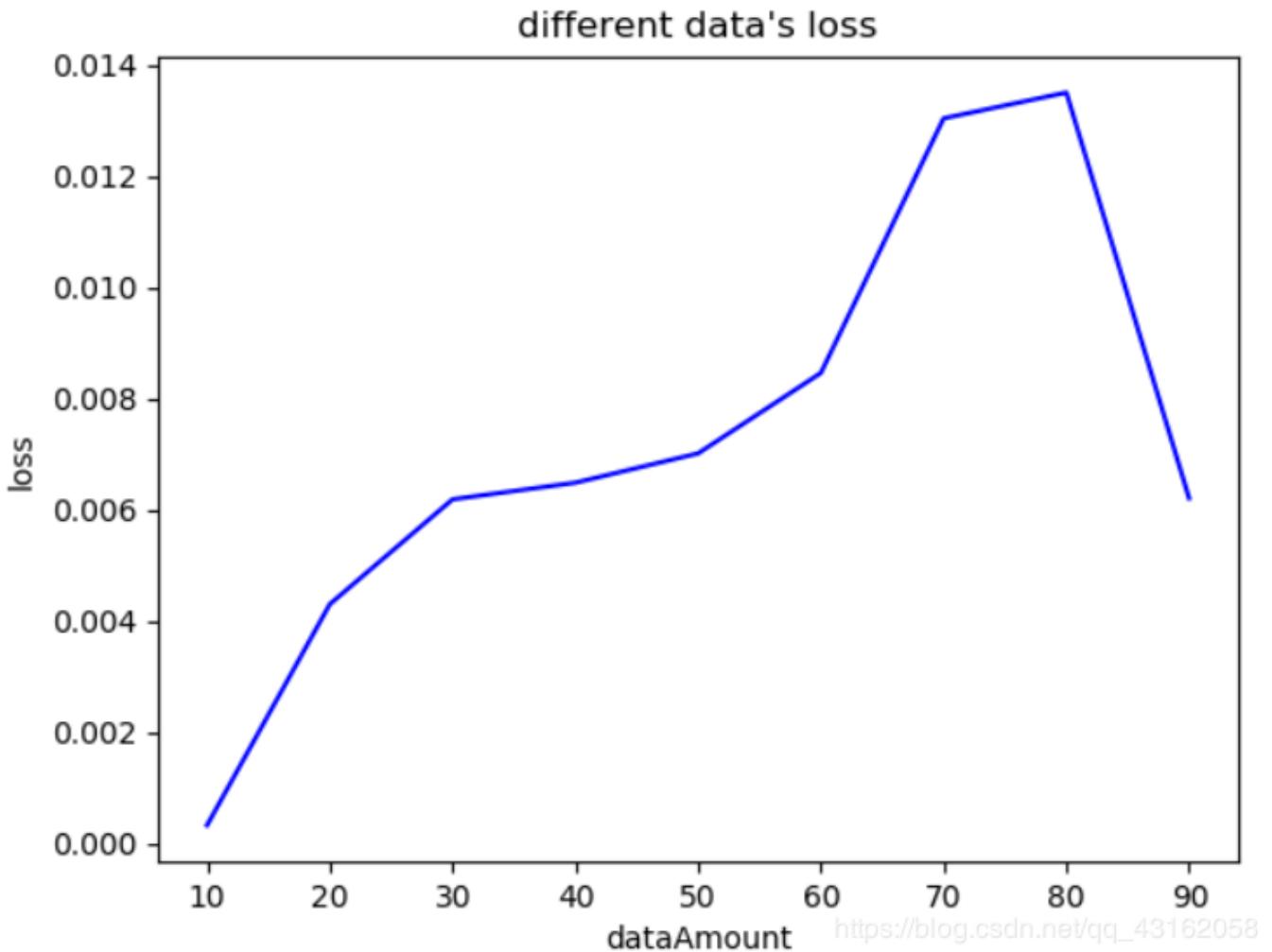
9-order to fit $\sin x$ 

https://blog.csdn.net/qq_43162058

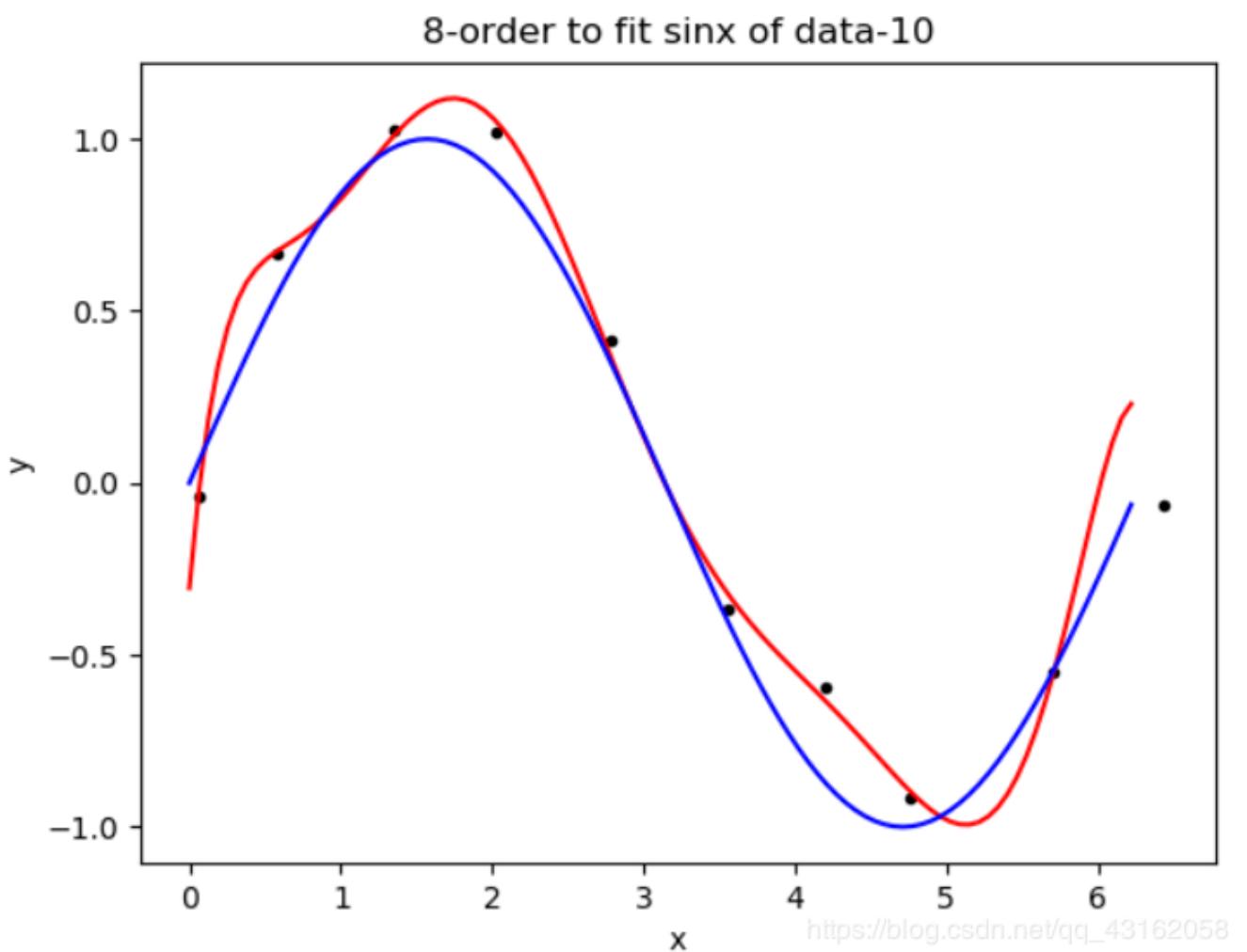
4.1.2 针对会出现过拟合的八阶拟合，再选用不同的数据量进行比较

不同数据量的平均loss（由于数据量较小时会出现过拟合，所以损失会比较小；当数据规模持续增大，过拟合会减弱，因此损失会增大；当数据量超过90时，曲线已经几乎不出现过拟合的现象，且loss相比6, 7阶

下降了许多。增大数据量也是克服过拟合的一种手段)

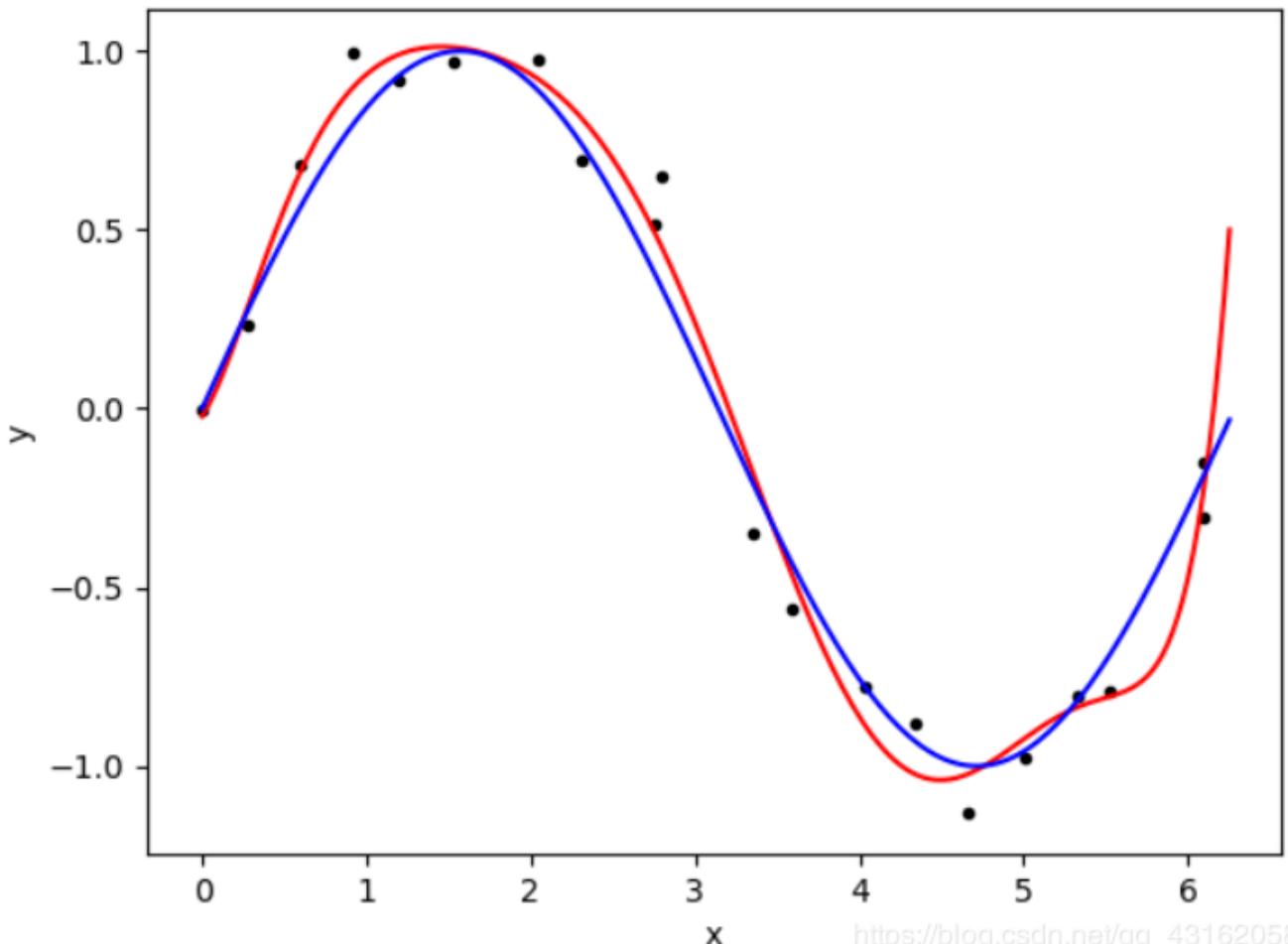


数据量为10



数据量为20

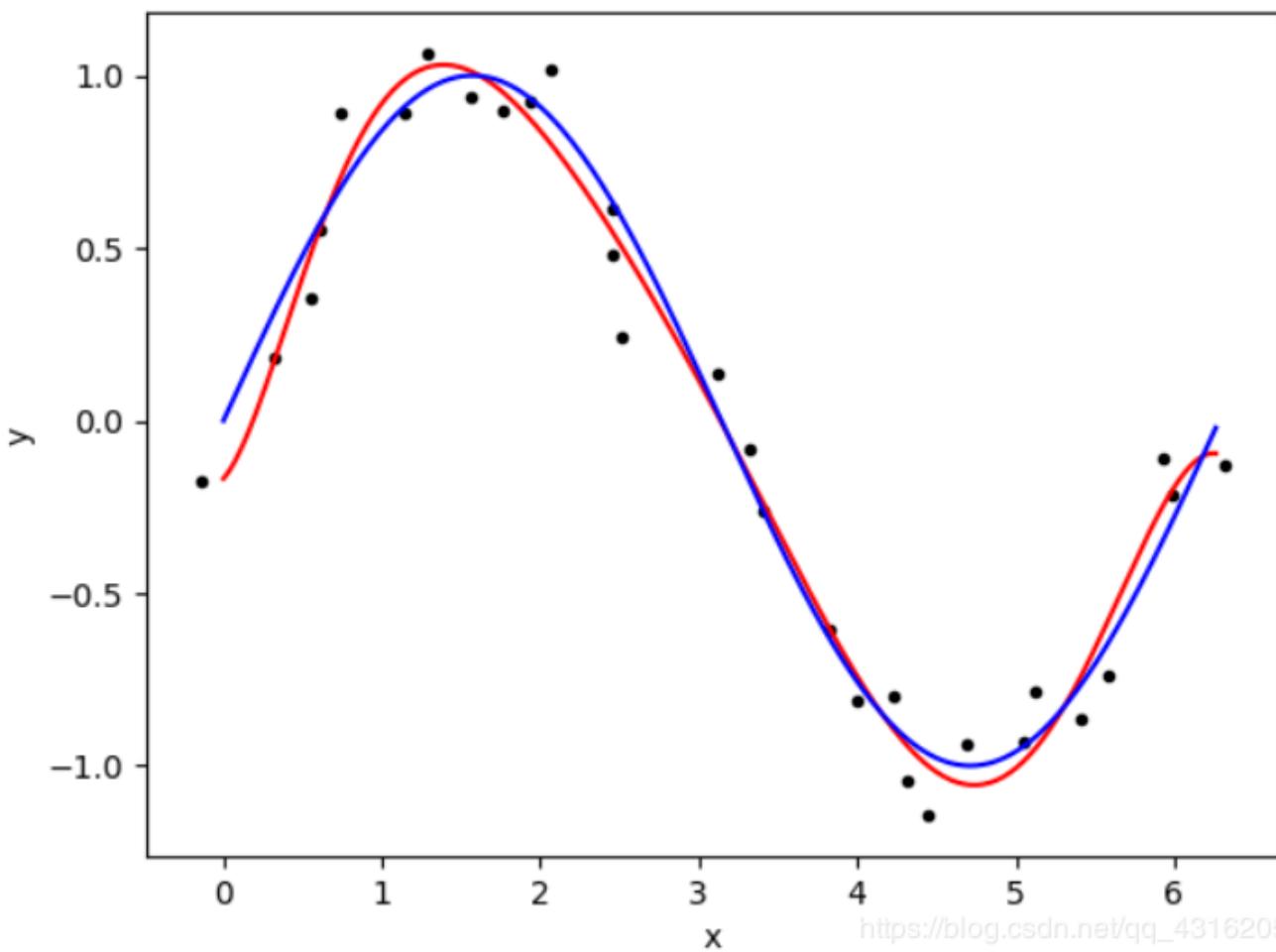
8-order to fit $\sin x$ of data-20



https://blog.csdn.net/qq_43162058

数据量为30

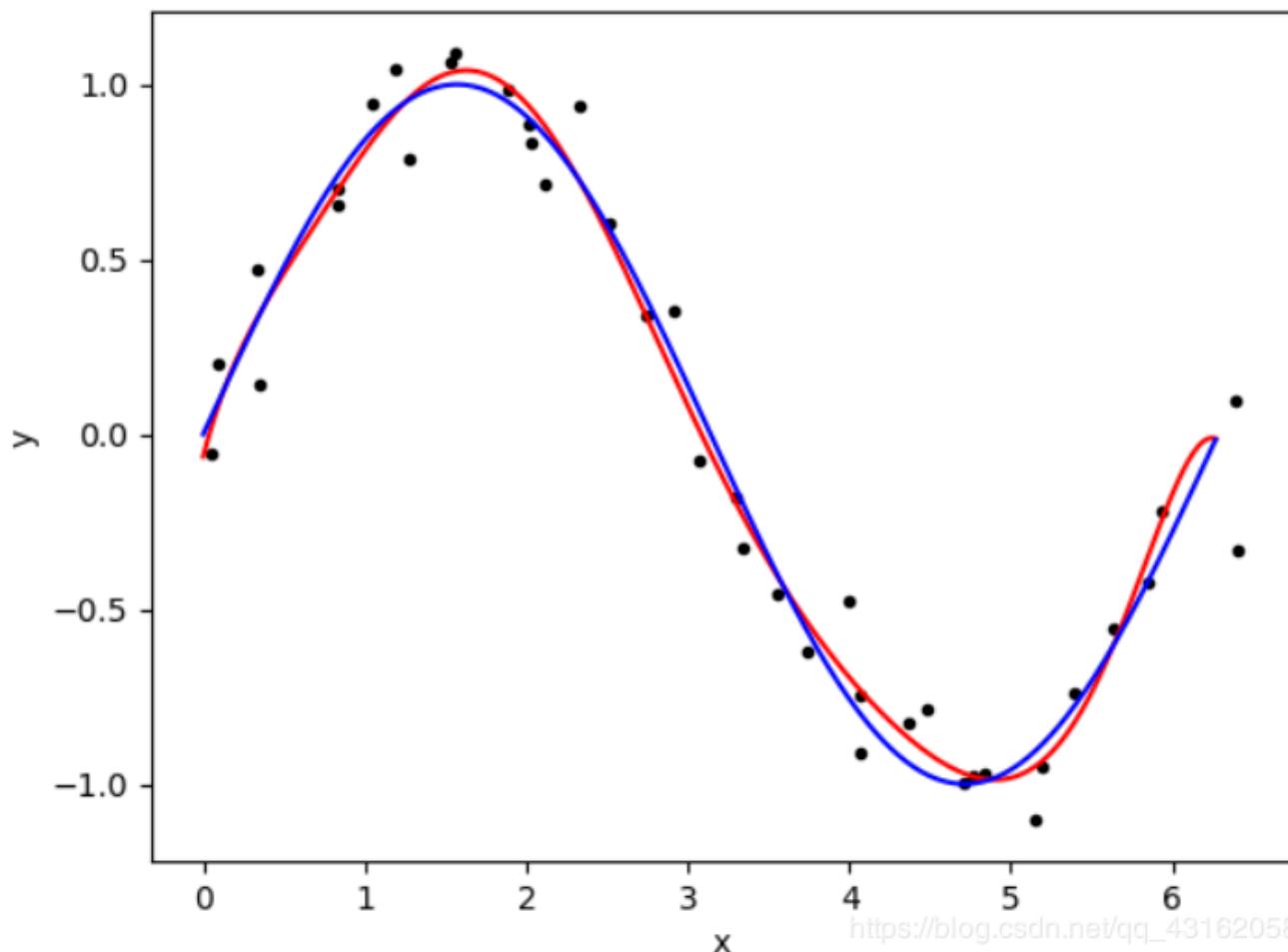
8-order to fit $\sin x$ of data-30



https://blog.csdn.net/qq_43162058

数据量为40

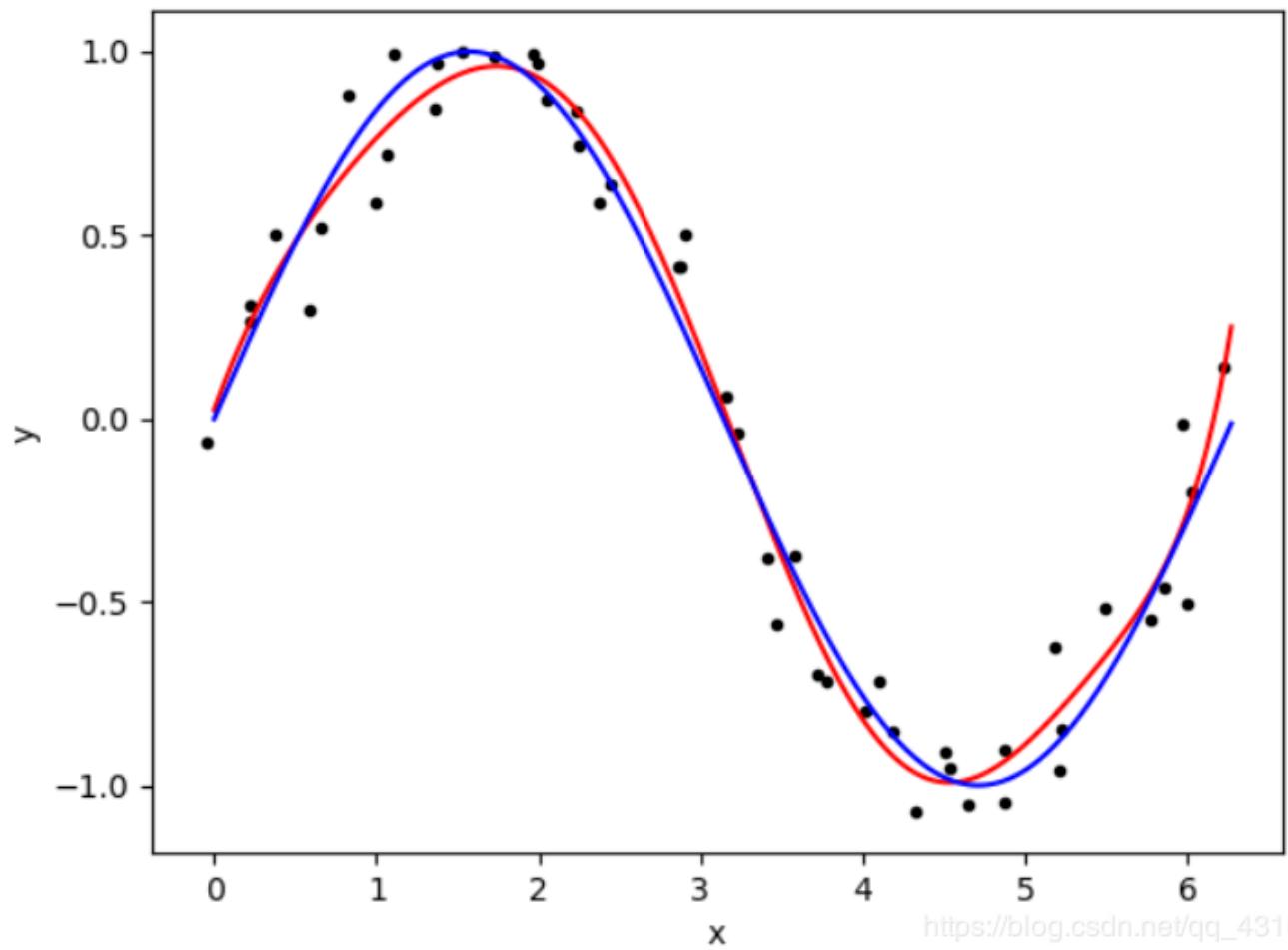
8-order to fit $\sin x$ of data-40



https://blog.csdn.net/qq_43162058

数据量为50

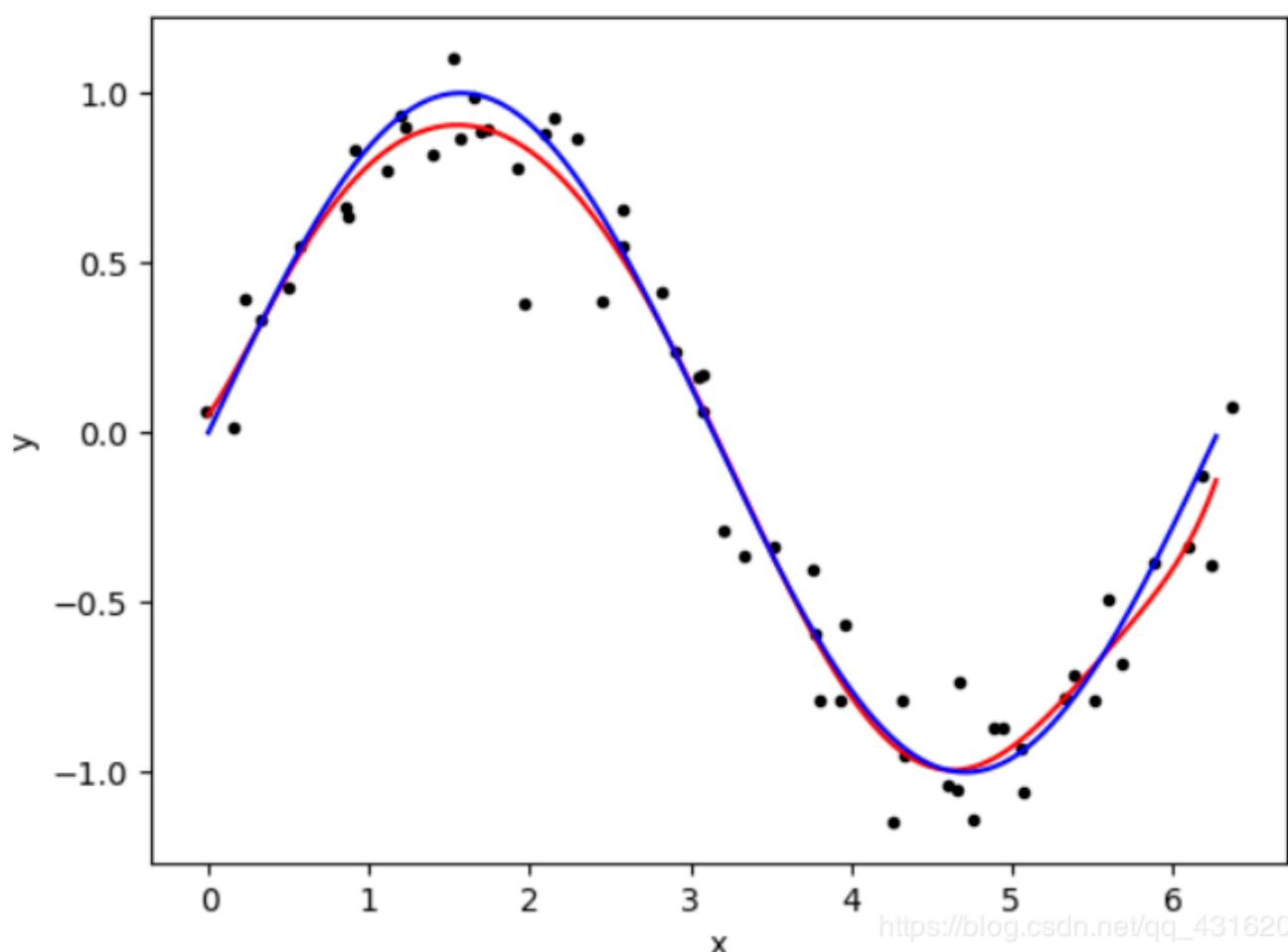
8-order to fit $\sin x$ of data-50



https://blog.csdn.net/qq_43162058

数据量为60

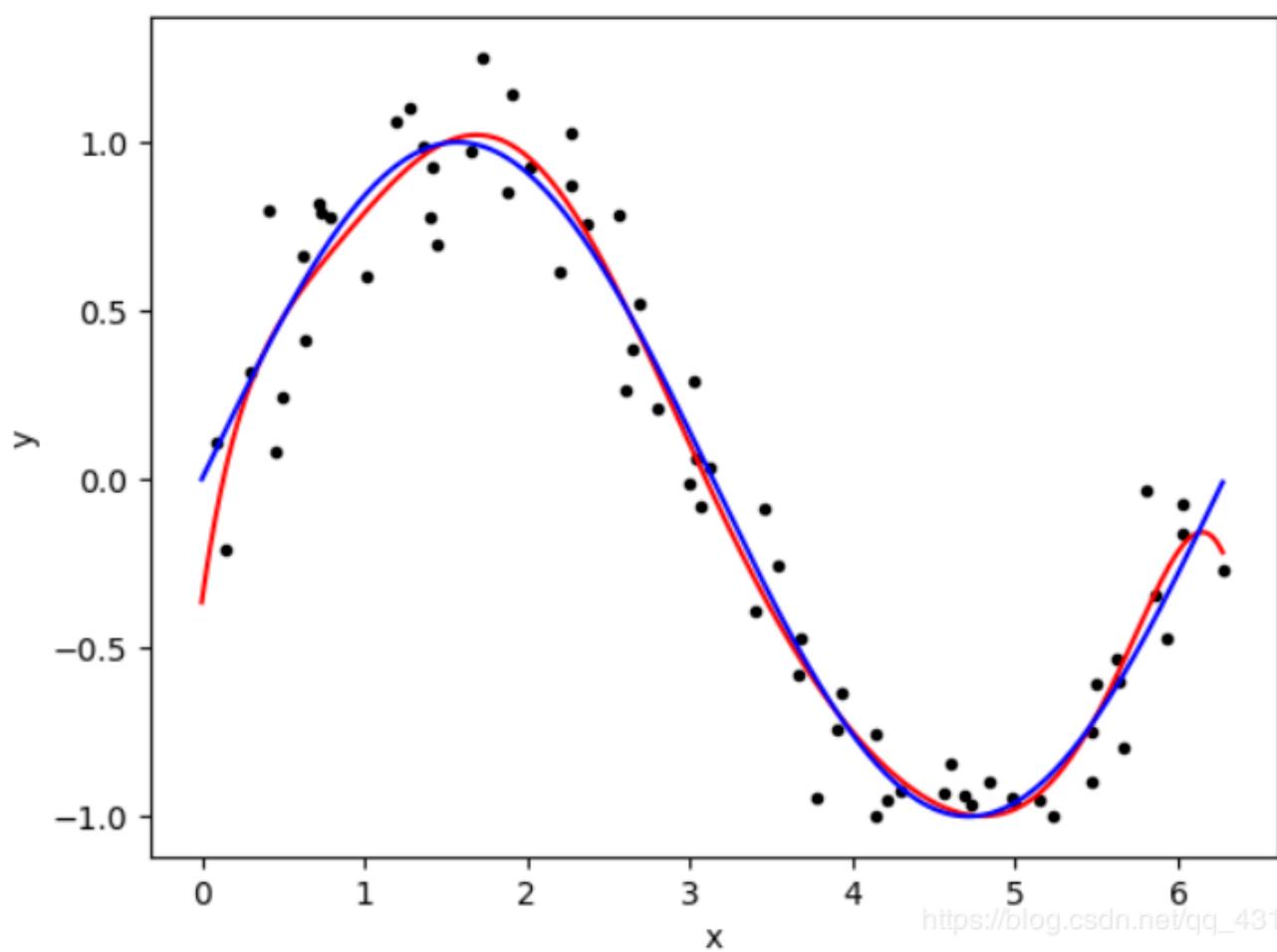
8-order to fit $\sin x$ of data-60



https://blog.csdn.net/qq_43162058

数据量为70

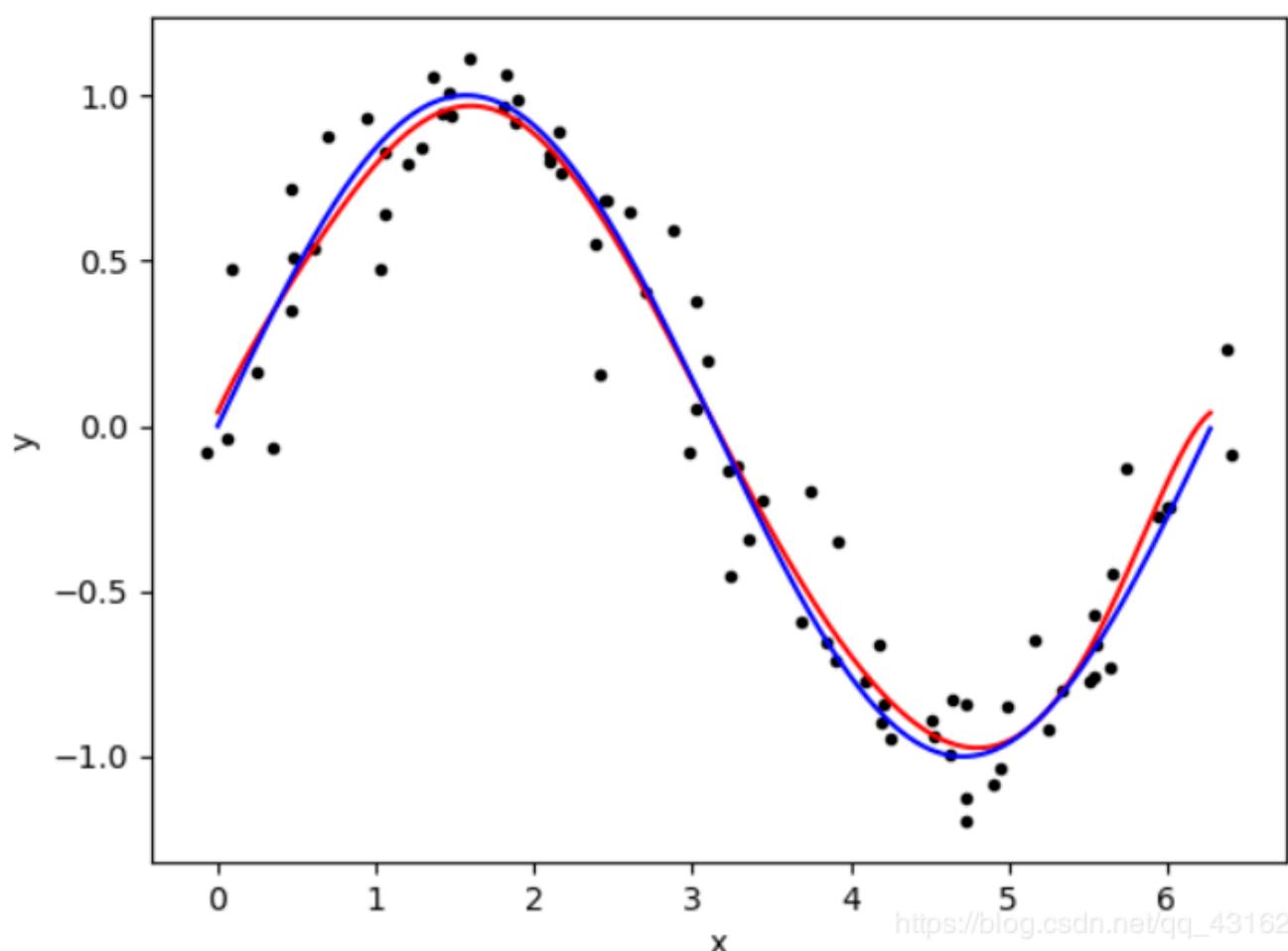
8-order to fit $\sin x$ of data-70



https://blog.csdn.net/qq_43162058

数据量为80

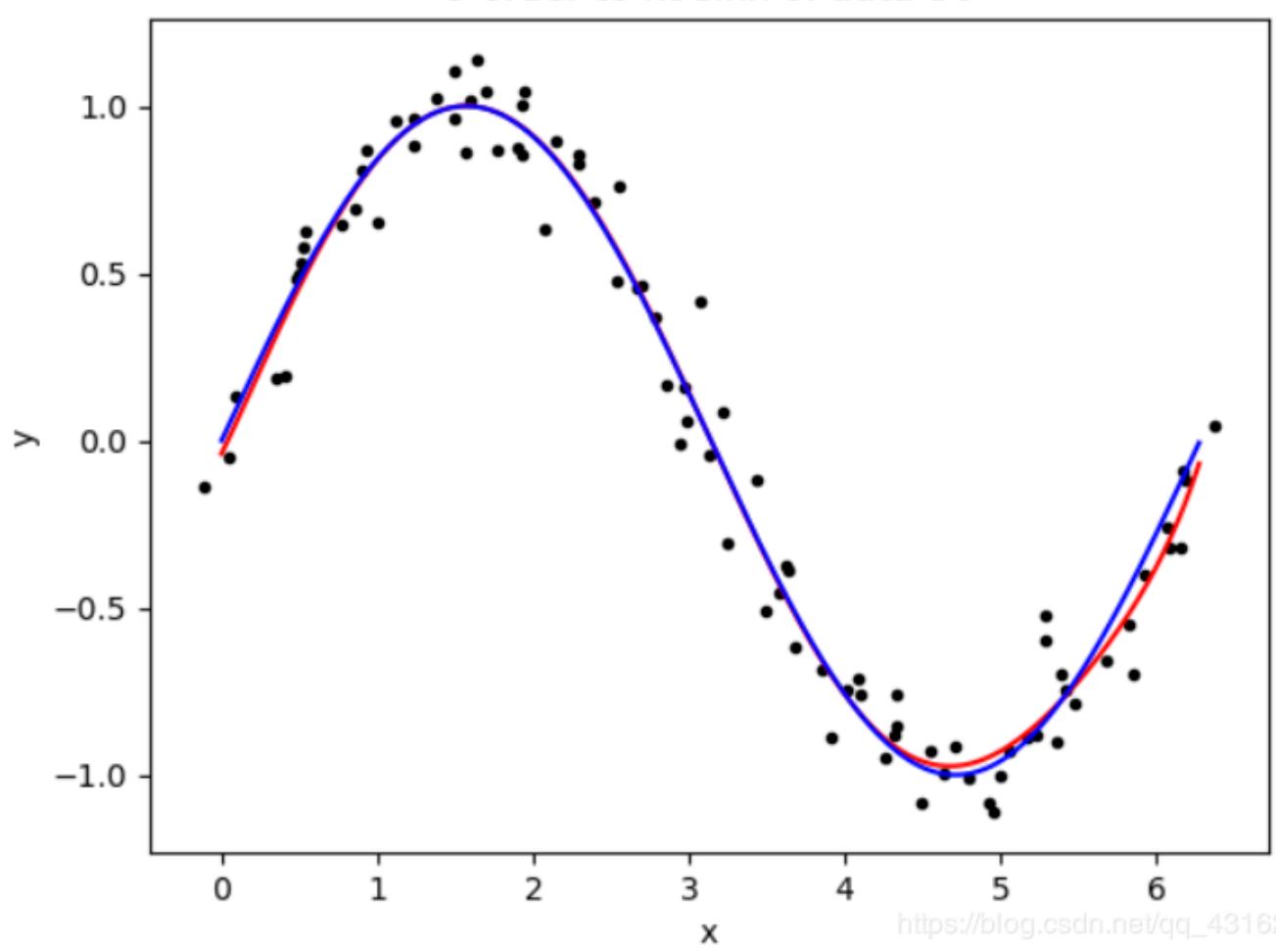
8-order to fit $\sin x$ of data-80



https://blog.csdn.net/qq_43162058

数据量为90

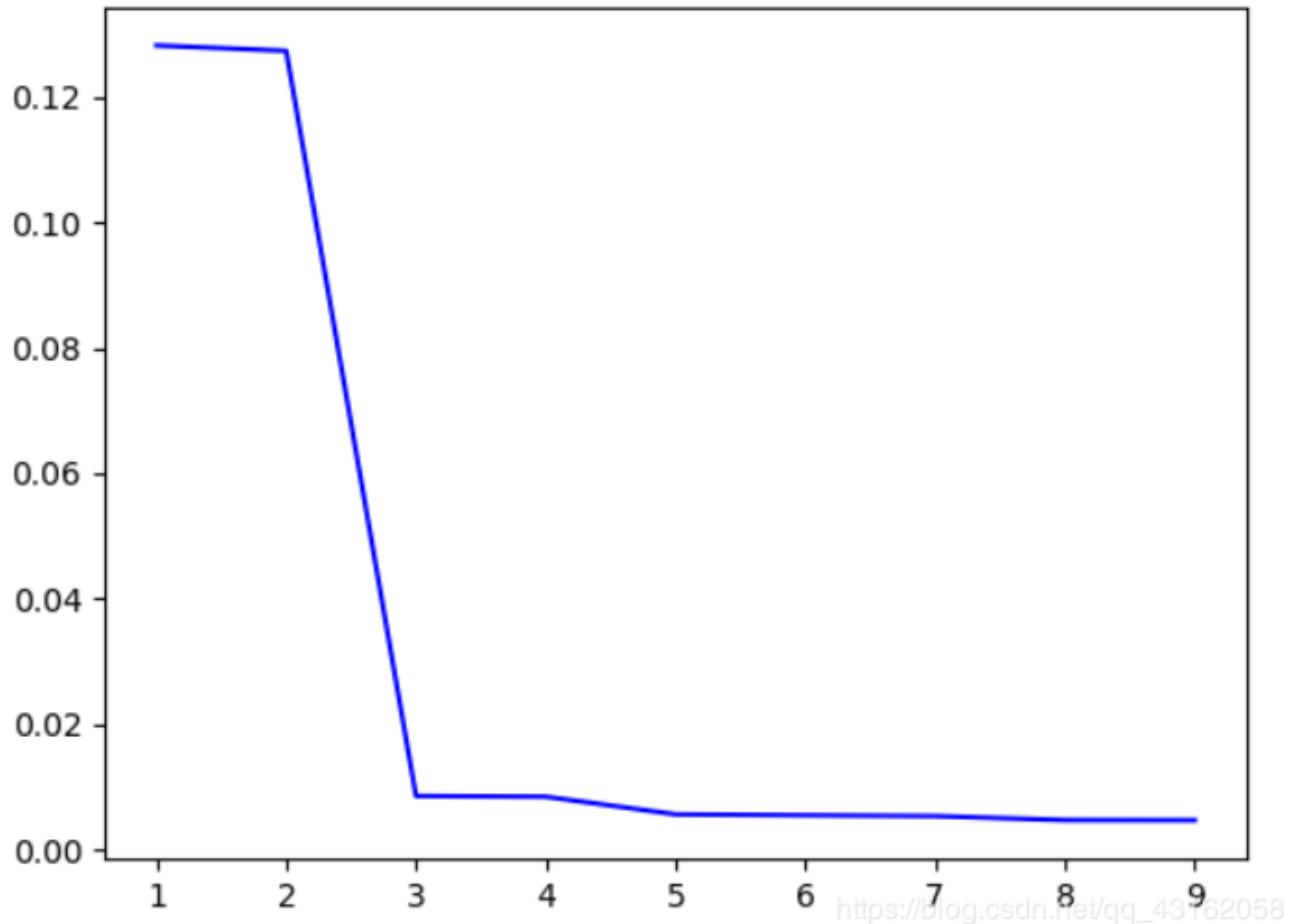
8-order to fit $\sin x$ of data-90



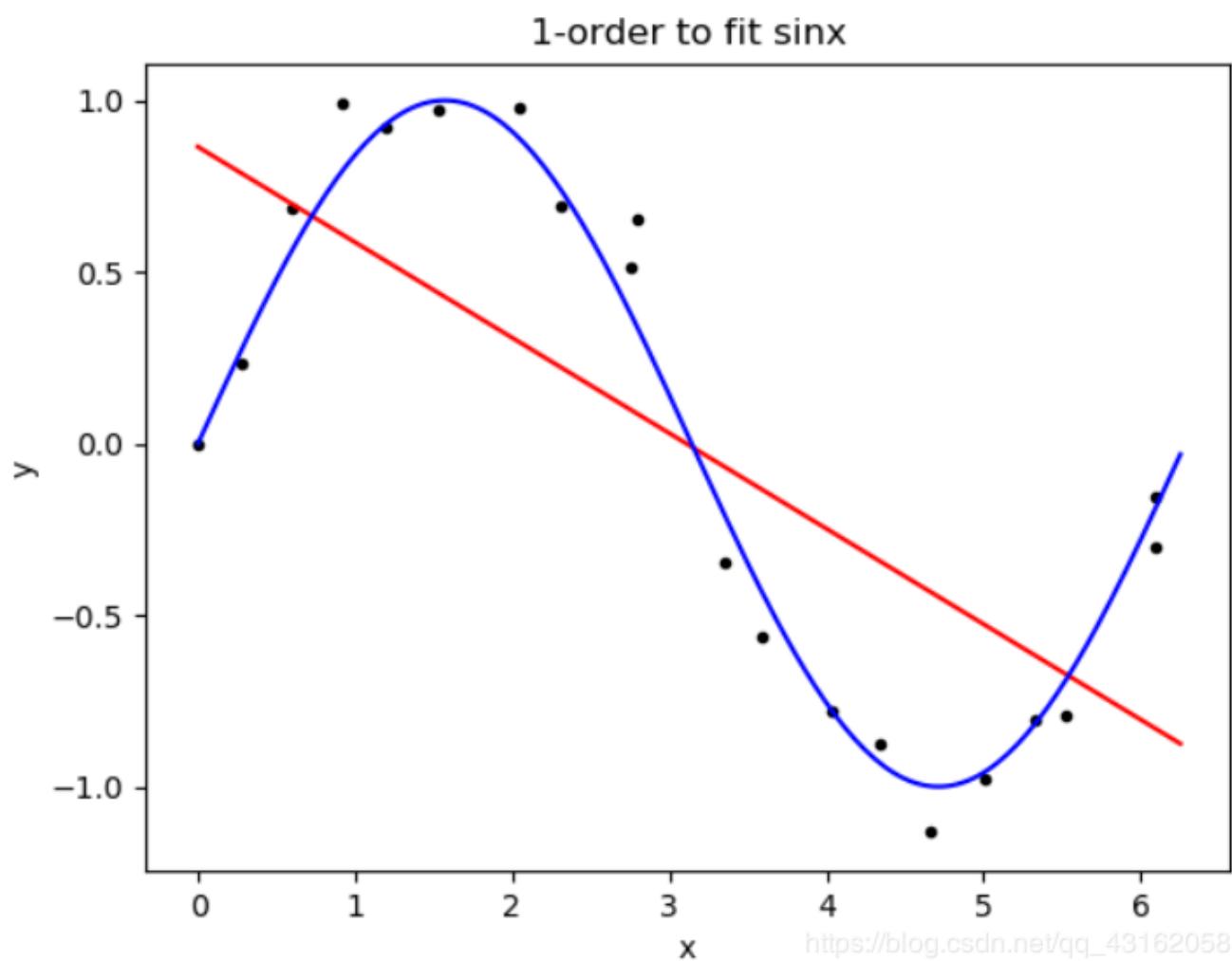
https://blog.csdn.net/qq_43162058

4.2.1当数据量为20时，研究不同的阶数的拟合效果

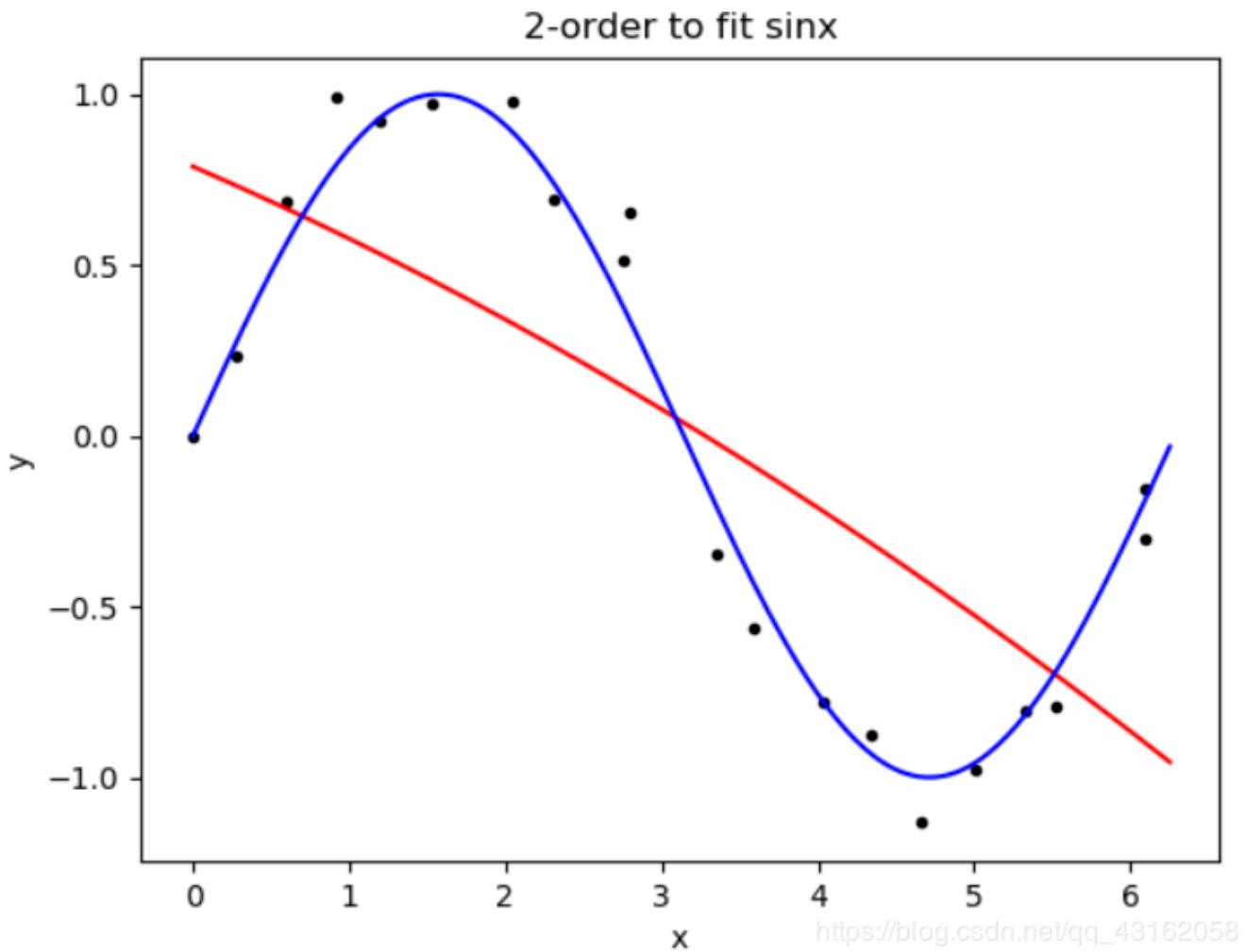
不同阶的损失如下图，其中 $\lambda = 0.0001$



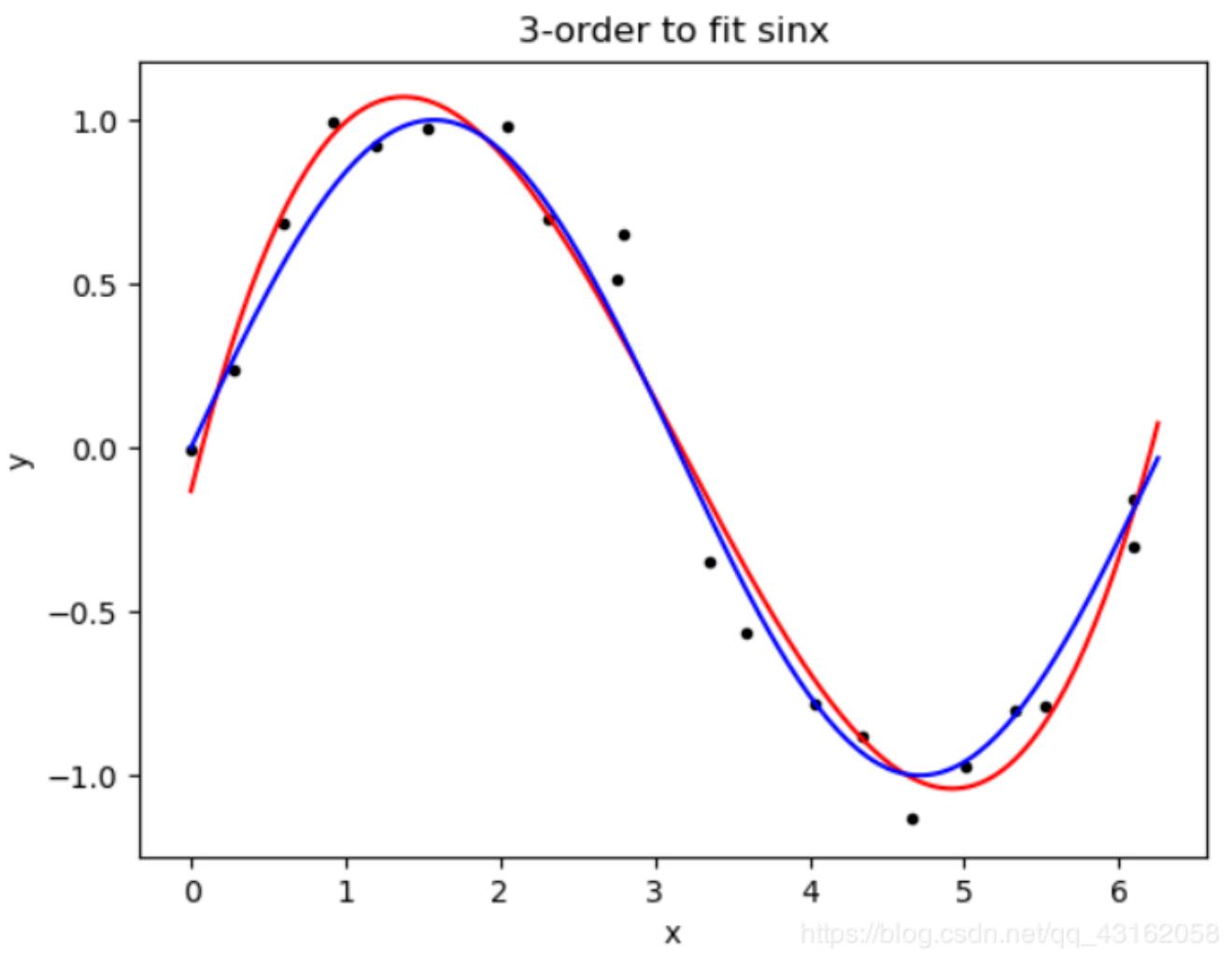
一阶



二阶

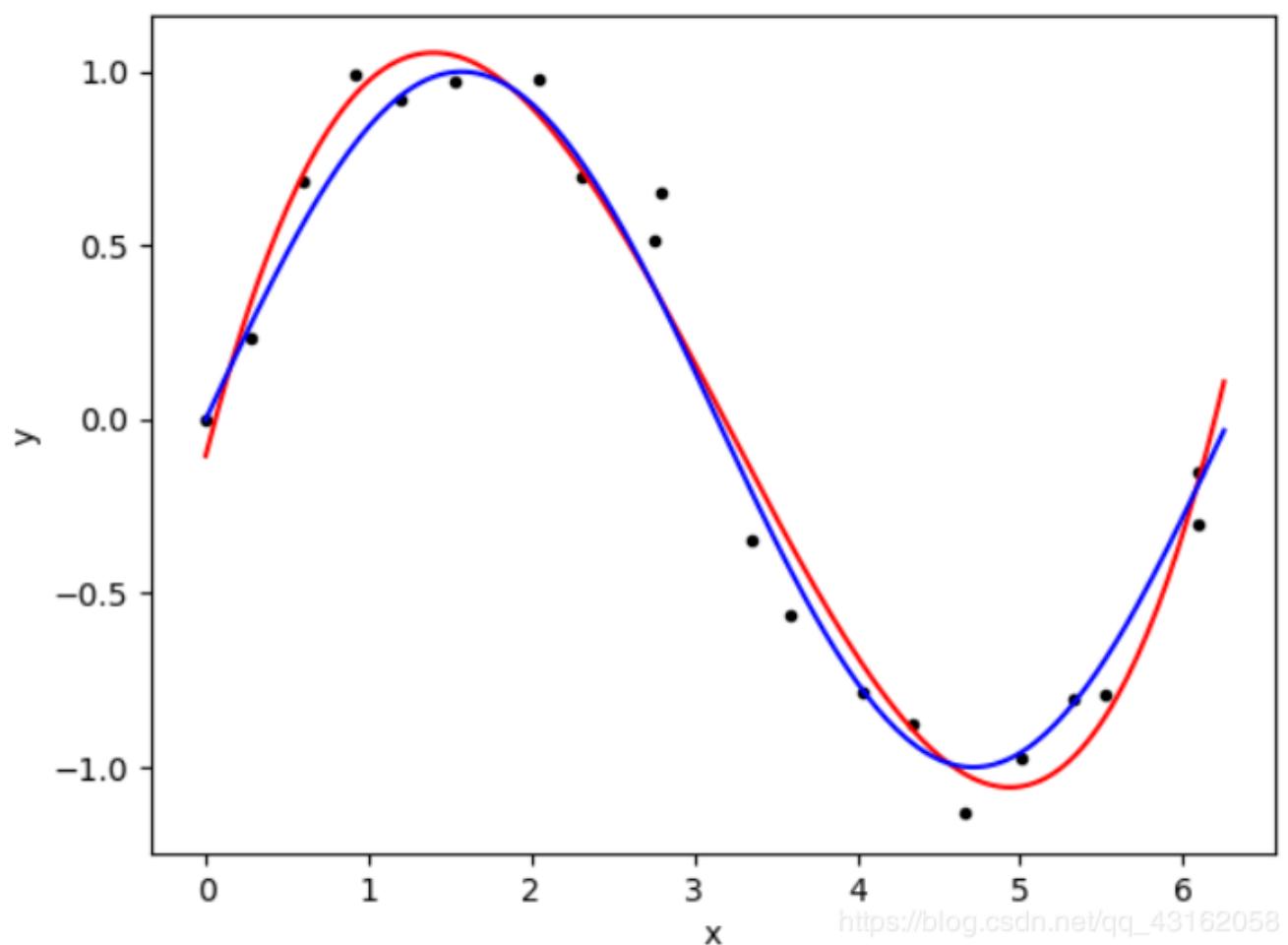


三阶



四阶

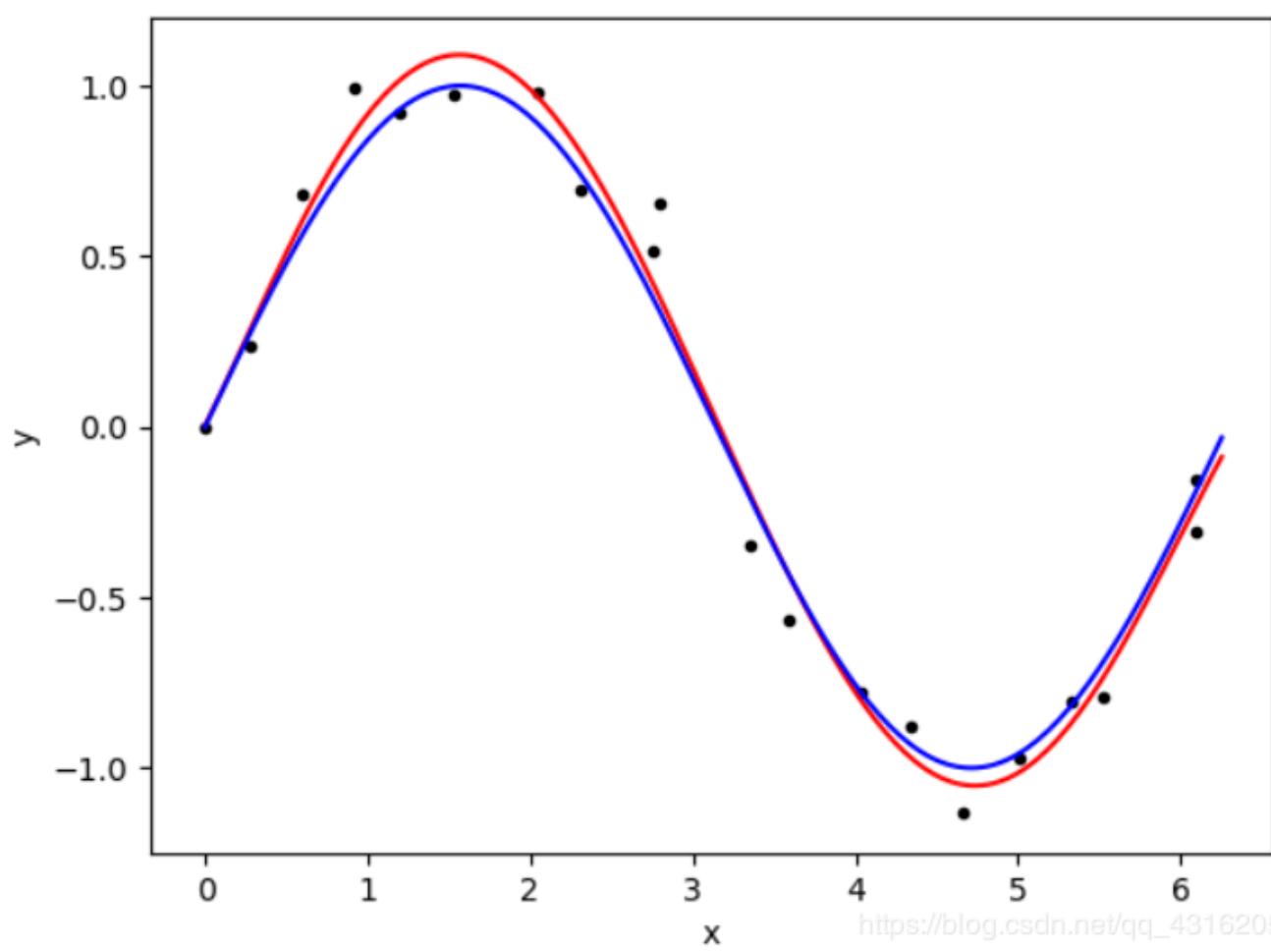
4-order to fit $\sin x$



https://blog.csdn.net/qq_43162058

五阶

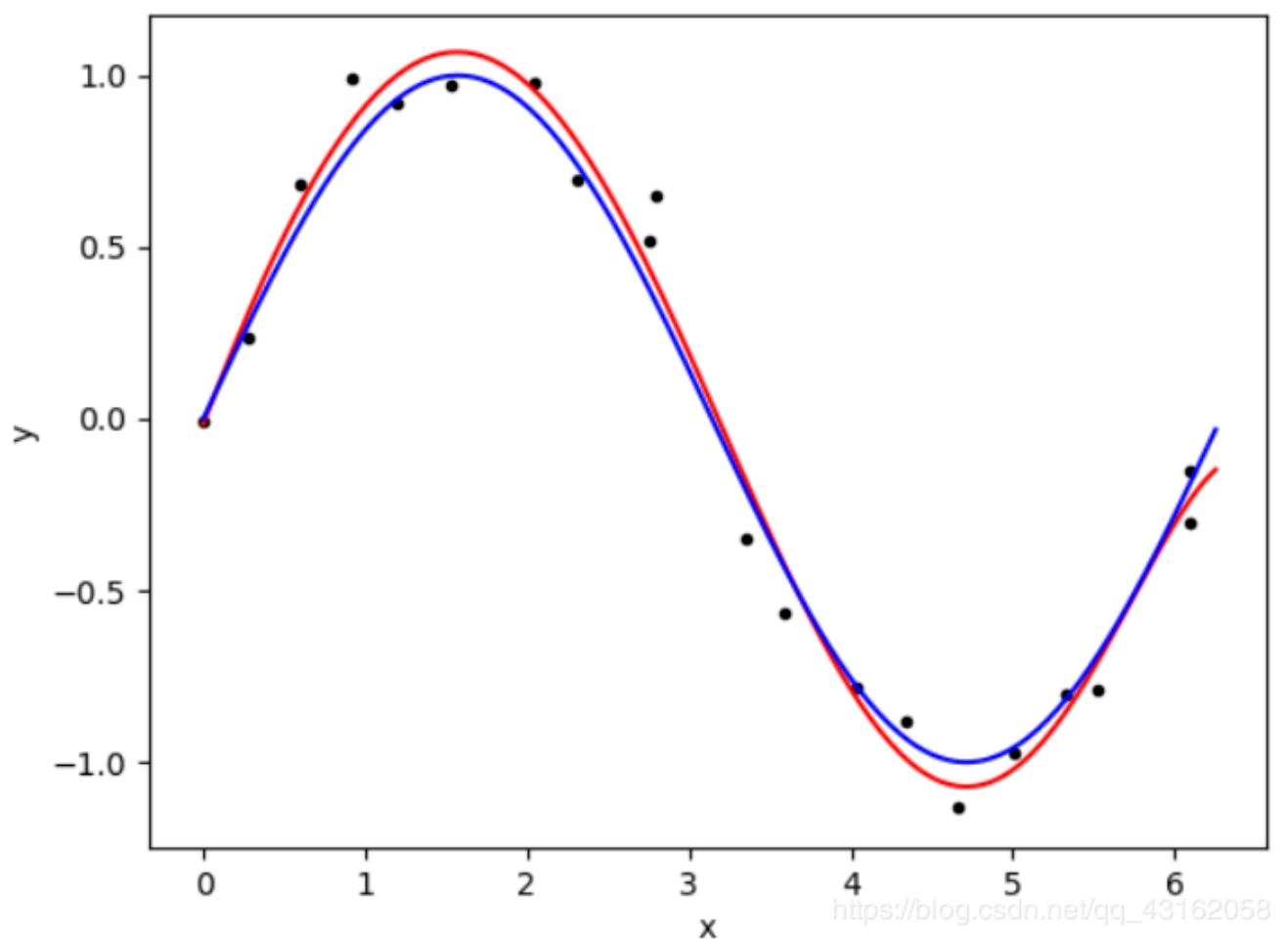
5-order to fit $\sin x$



https://blog.csdn.net/qq_43162058

六阶

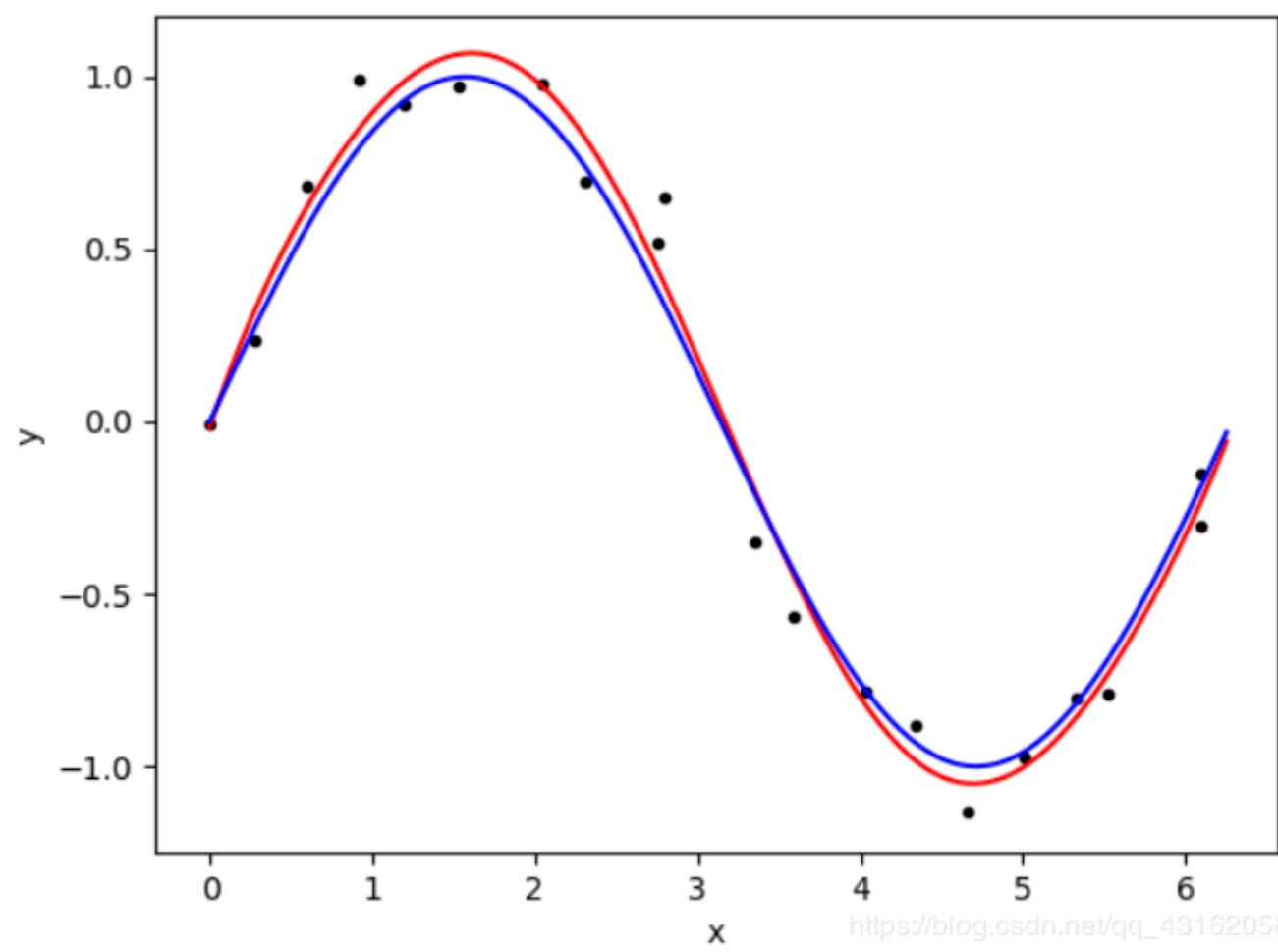
6-order to fit sinx



https://blog.csdn.net/qq_43162058

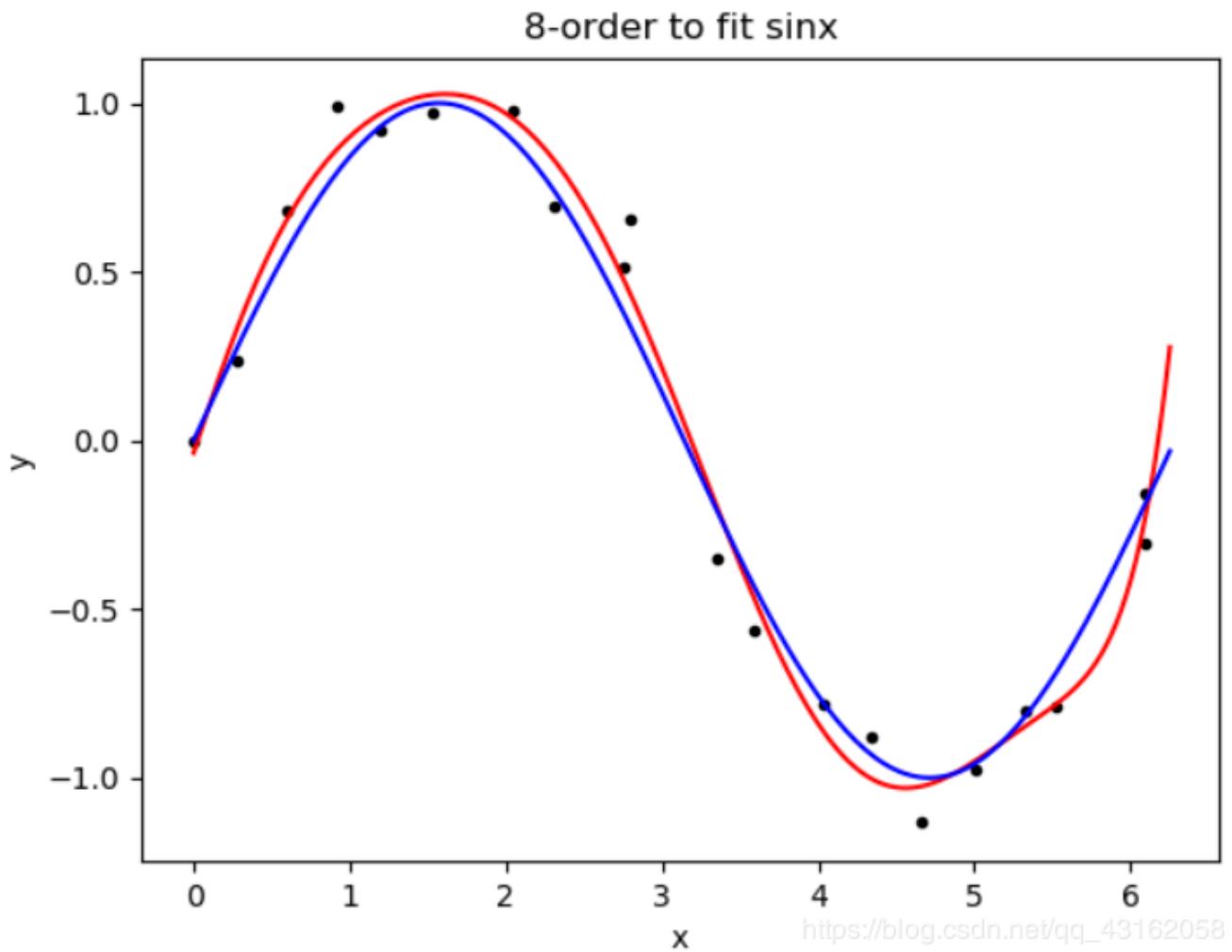
七阶

7-order to fit sinx



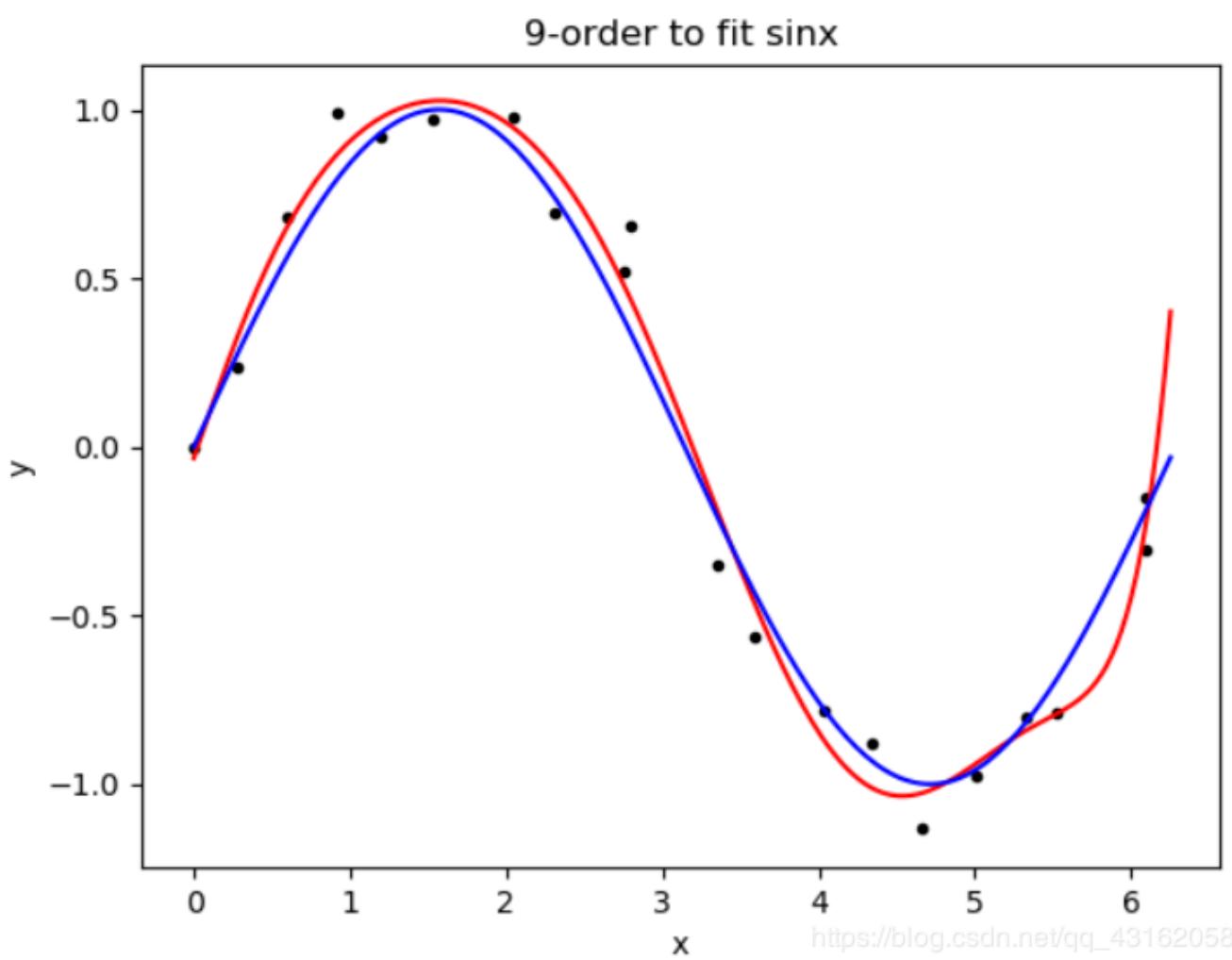
https://blog.csdn.net/qq_43162058

八阶



https://blog.csdn.net/qq_43162058

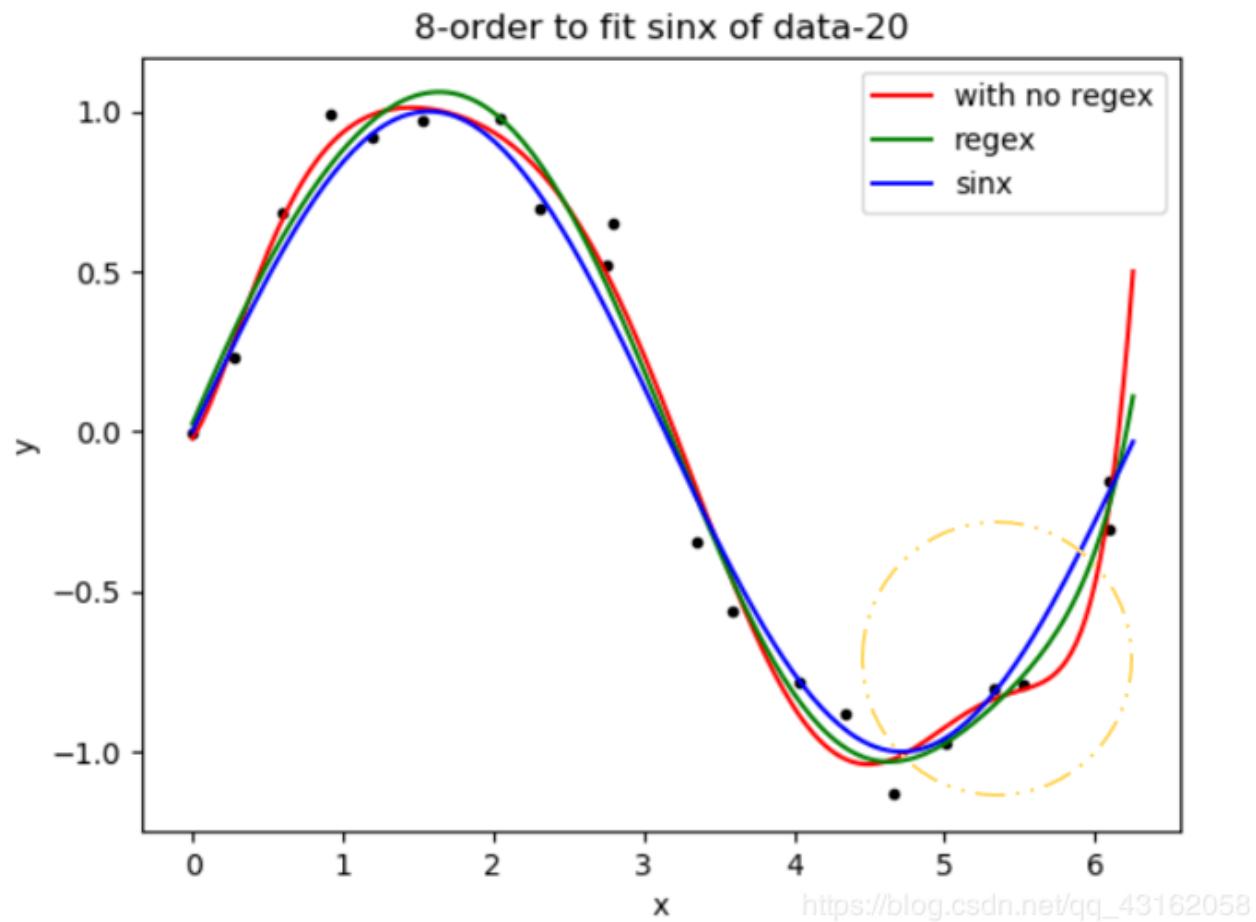
九阶



https://blog.csdn.net/qq_43162058

可以看出，相比不带正则项的解析式解法，8, 9阶的过拟合有所好转，（下图单独对这两个进行比较）仍

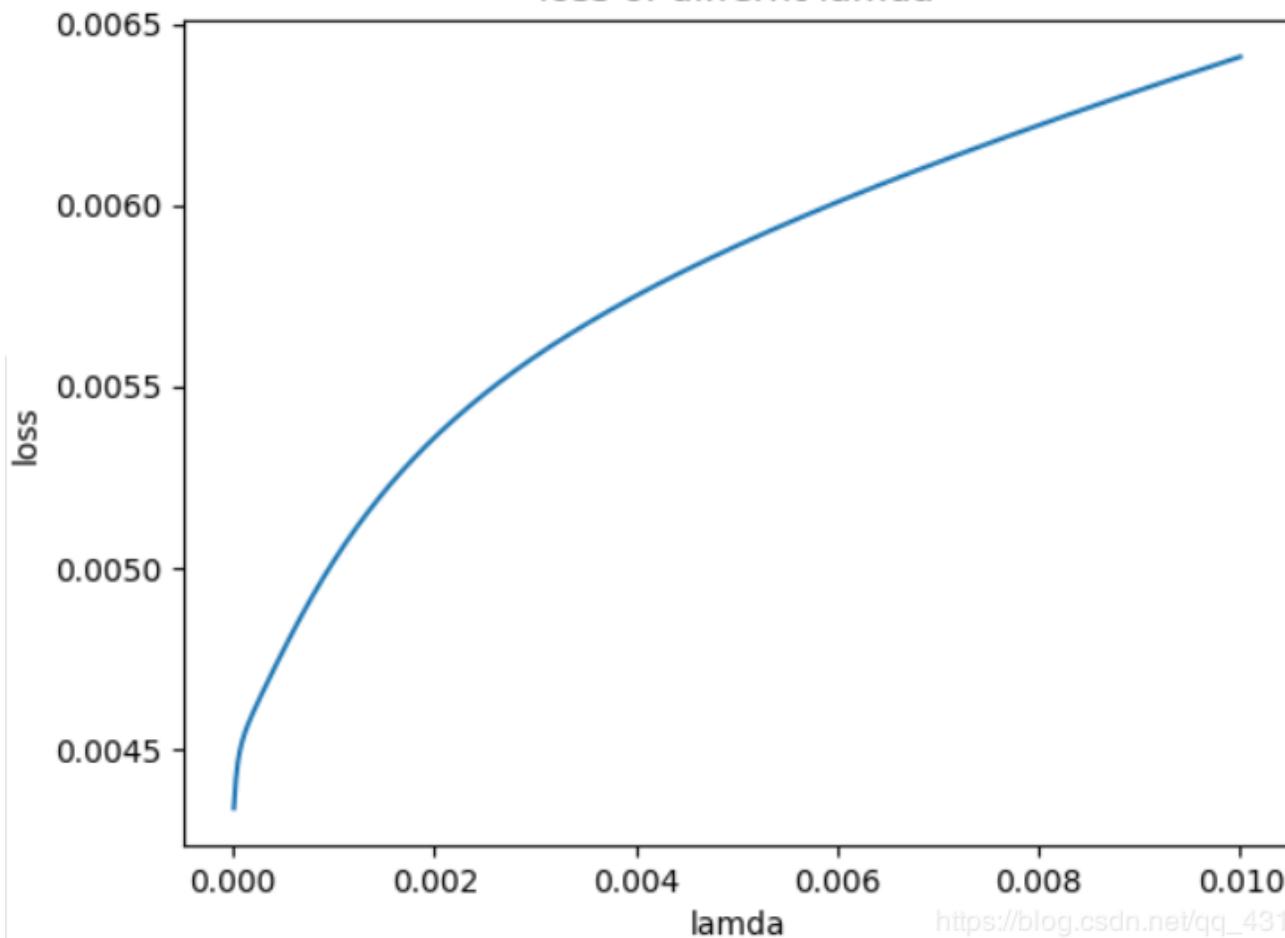
然出现过拟合可能是前面的参数不够好的缘故，接下来会讨论不同的 λ 取值的影响



4.2.1 数据量为20时，不同 λ 取值对曲线过拟合的影响

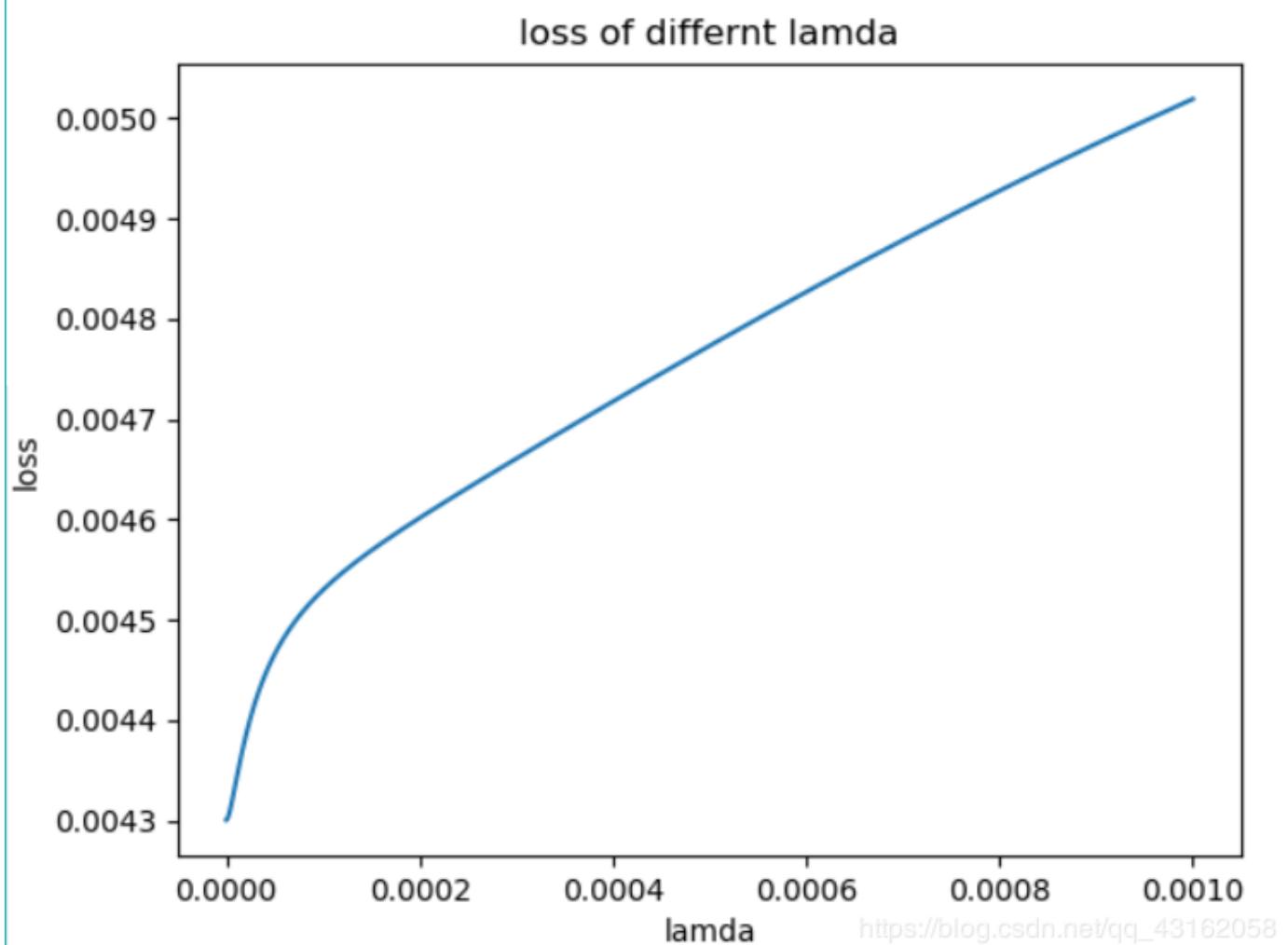
取从0至0.01的，绘制出了如下曲线，可以看出当 λ 较小时，loss比较小(不含正则项的loss，因为正则项的loss太大了)，即使取到0.01，loss值仍然在可接受范围

loss of differnt lamda



https://blog.csdn.net/qq_43162058

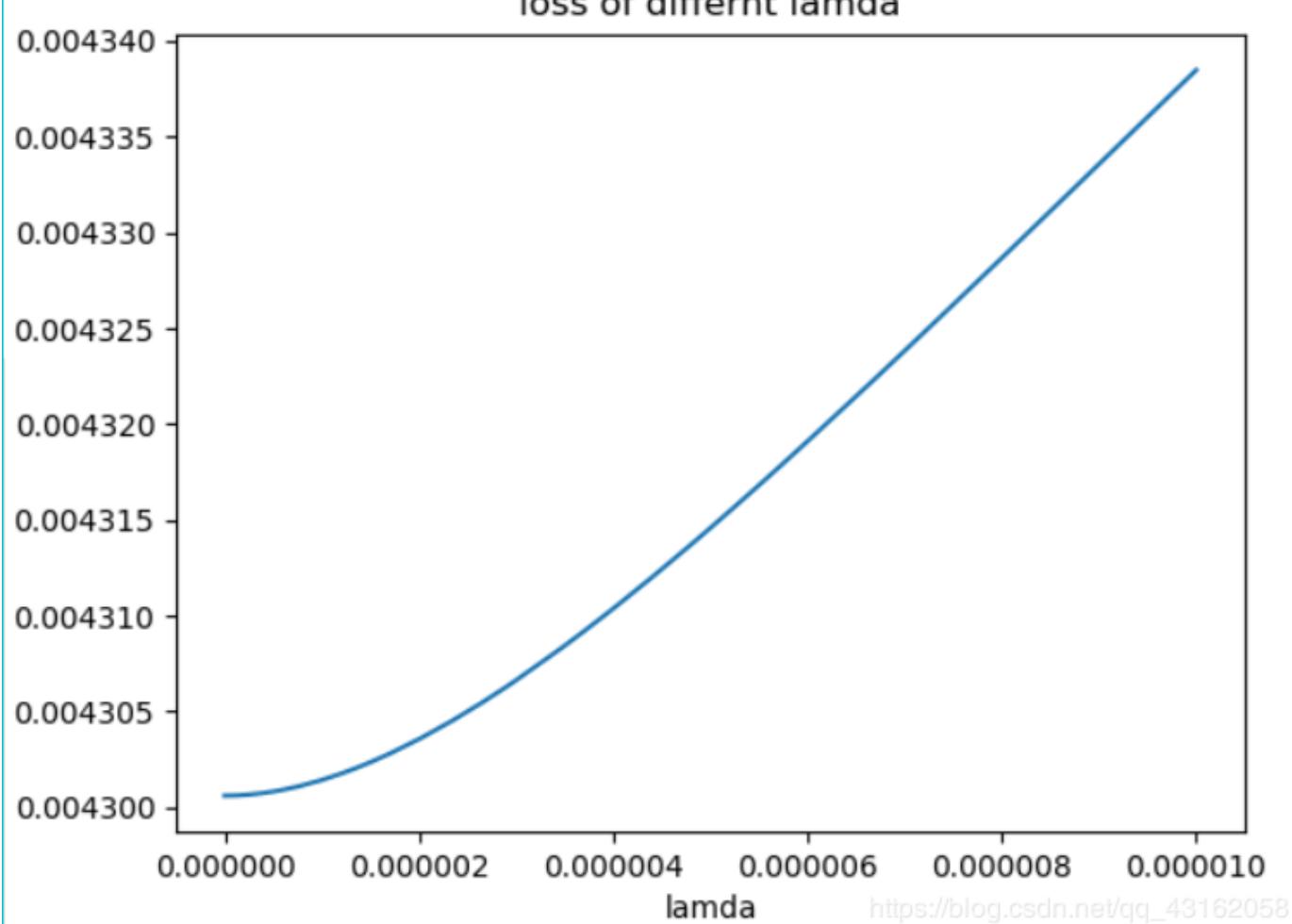
我们再取一个更小的区间



https://blog.csdn.net/qq_43162058

再缩小

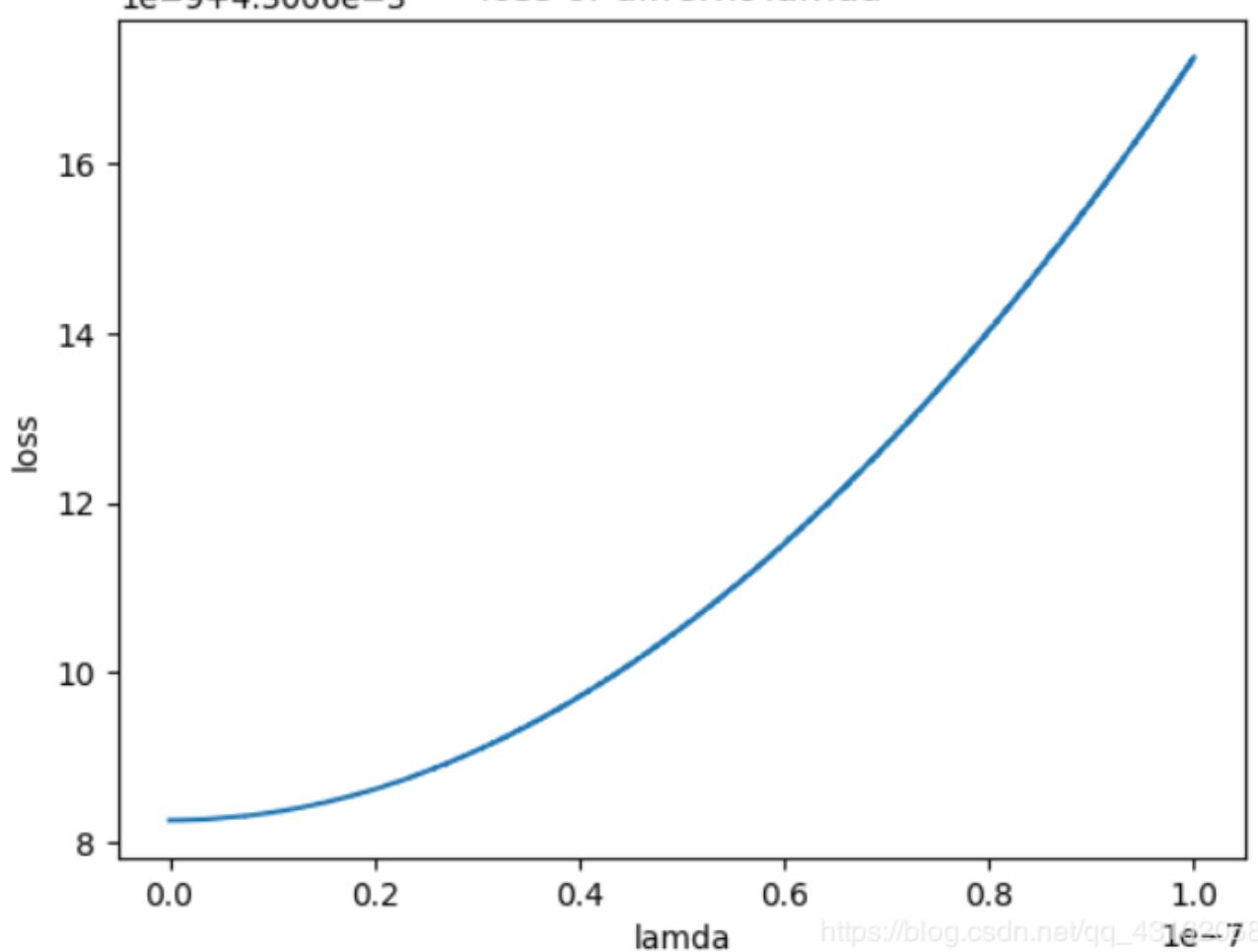
loss of differnt lamda



https://blog.csdn.net/qq_43162058

继续缩小

$1e-9+4.3006e-3$ loss of differnt lamda

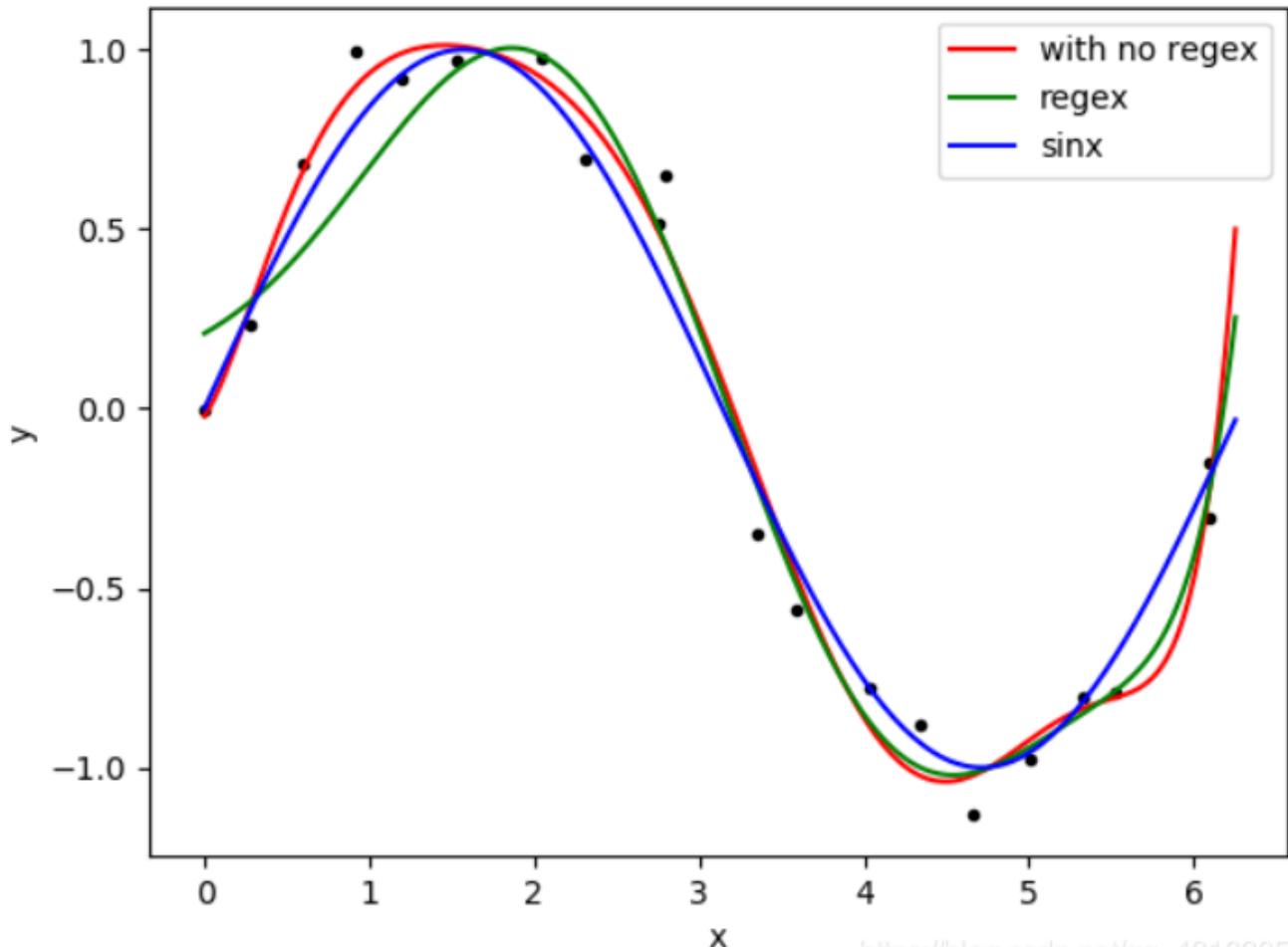


https://blog.csdn.net/qq_43162058

根据上面的loss图，我们抽取 $\lambda = 1e - 1, 1e - 2, 1e - 3, 1e - 4, 1e - 5, 1e - 6$ 分布和不带正则项的解析式法去比较。

当 $\lambda = 1e - 1$ 时

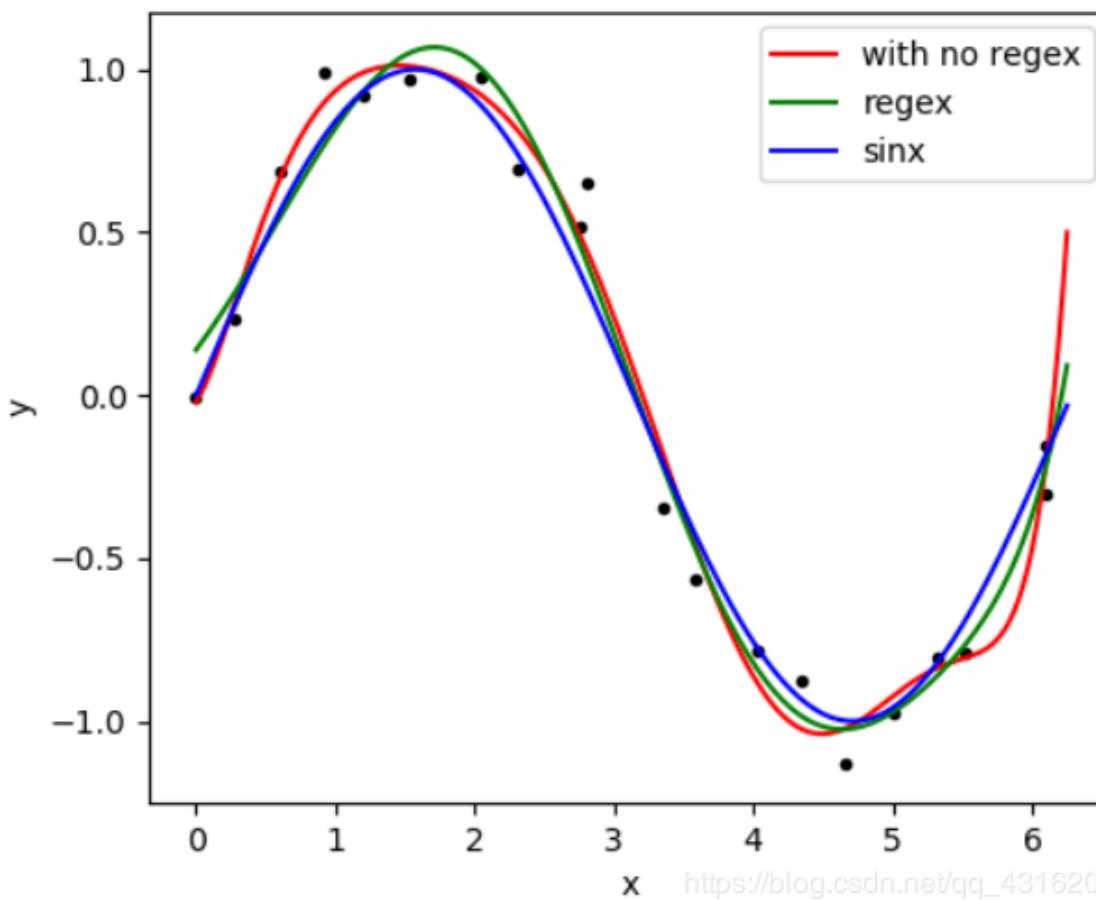
8-order to fit sinx of data-20, lamda = 0.1



https://blog.csdn.net/qq_43162058

当 $\lambda = 1e - 2$ 时

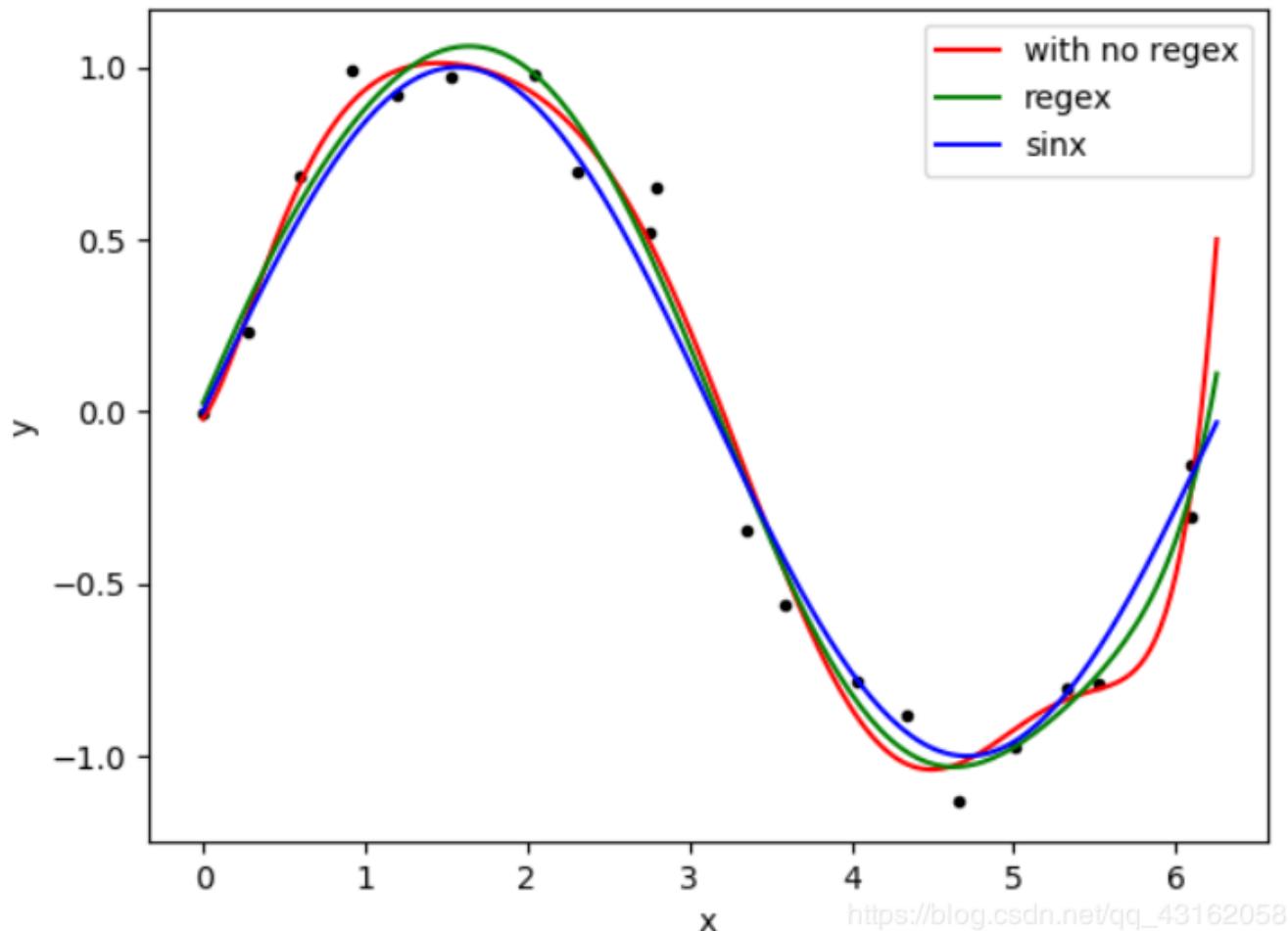
8-order to fit sinx of data-20, lamda = 0.01



https://blog.csdn.net/qq_43162058

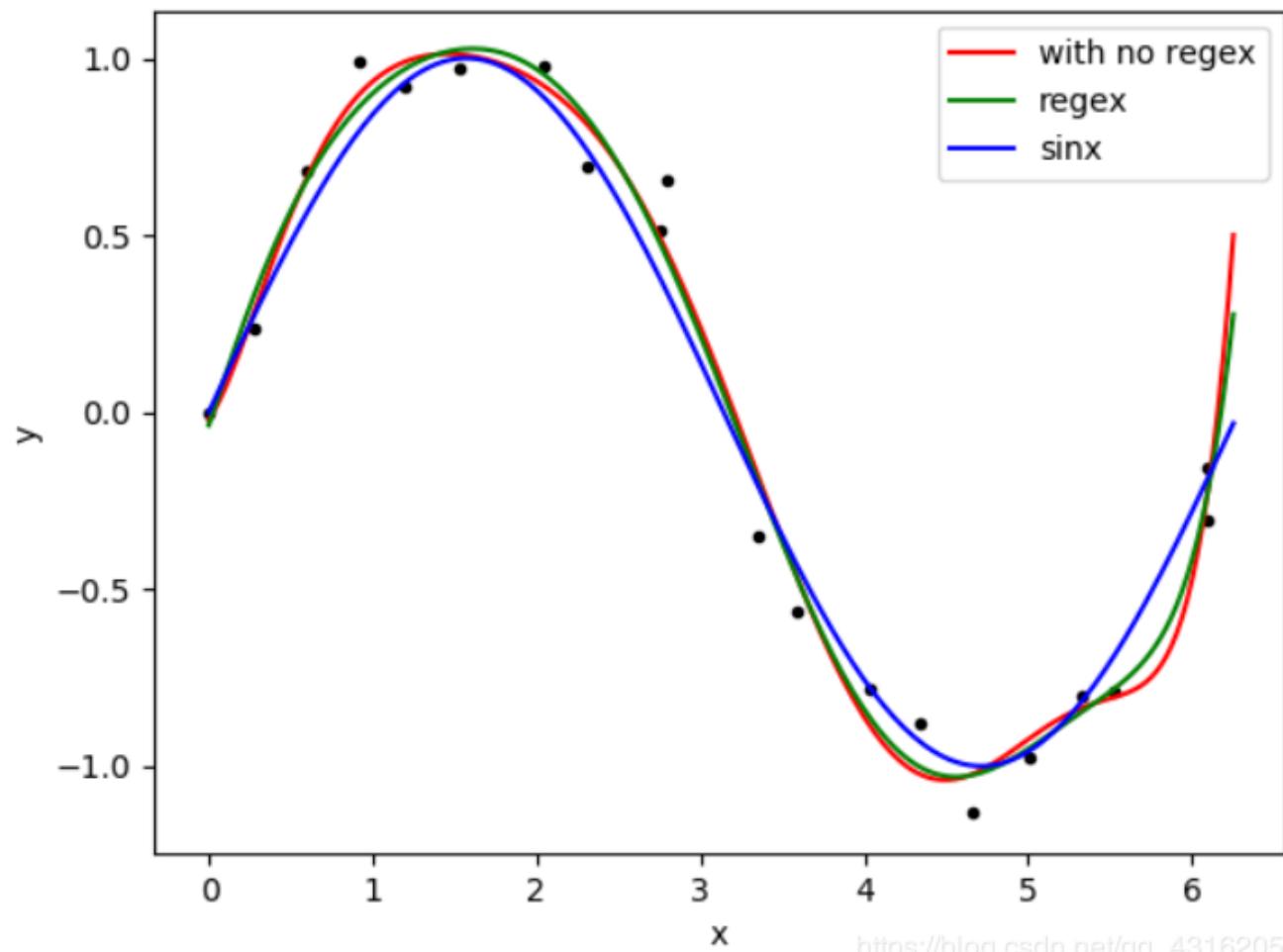
当 $\lambda = 1e - 3$ 时

8-order to fit sinx of data-20, lamda = 0.001



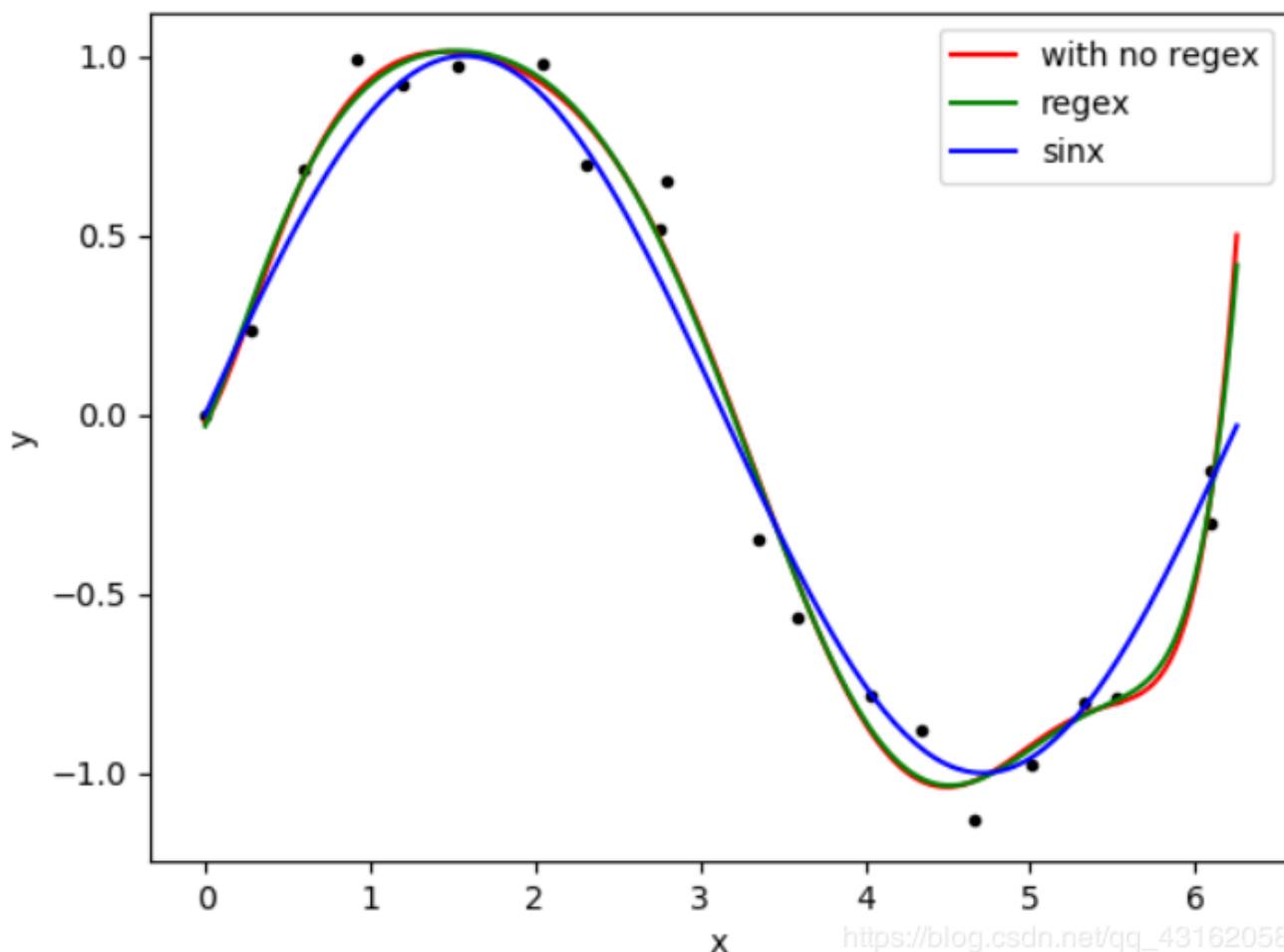
当 $\lambda = 1e - 4$ 时

8-order to fit sinx of data-20, lamda = 0.0001



当 $\lambda = 1e - 5$ 时，带正则和不带正则的曲线几乎重合，此时正则系数过小

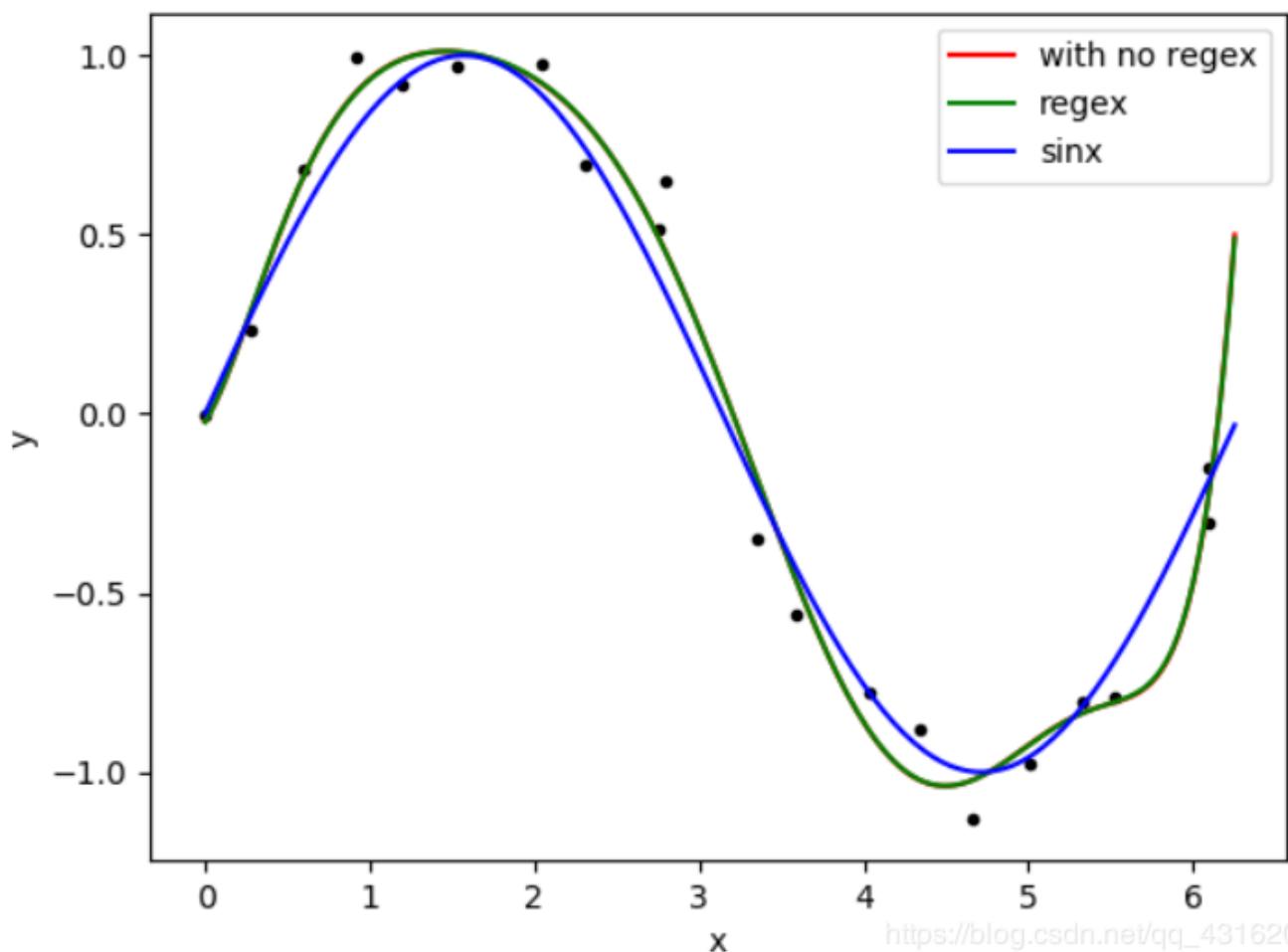
8-order to fit sinx of data-20, lamda = 1e-05



https://blog.csdn.net/qq_43162058

当 $\lambda = 1e - 6$ 时，不带正则的曲线和带正则的曲线完全重合

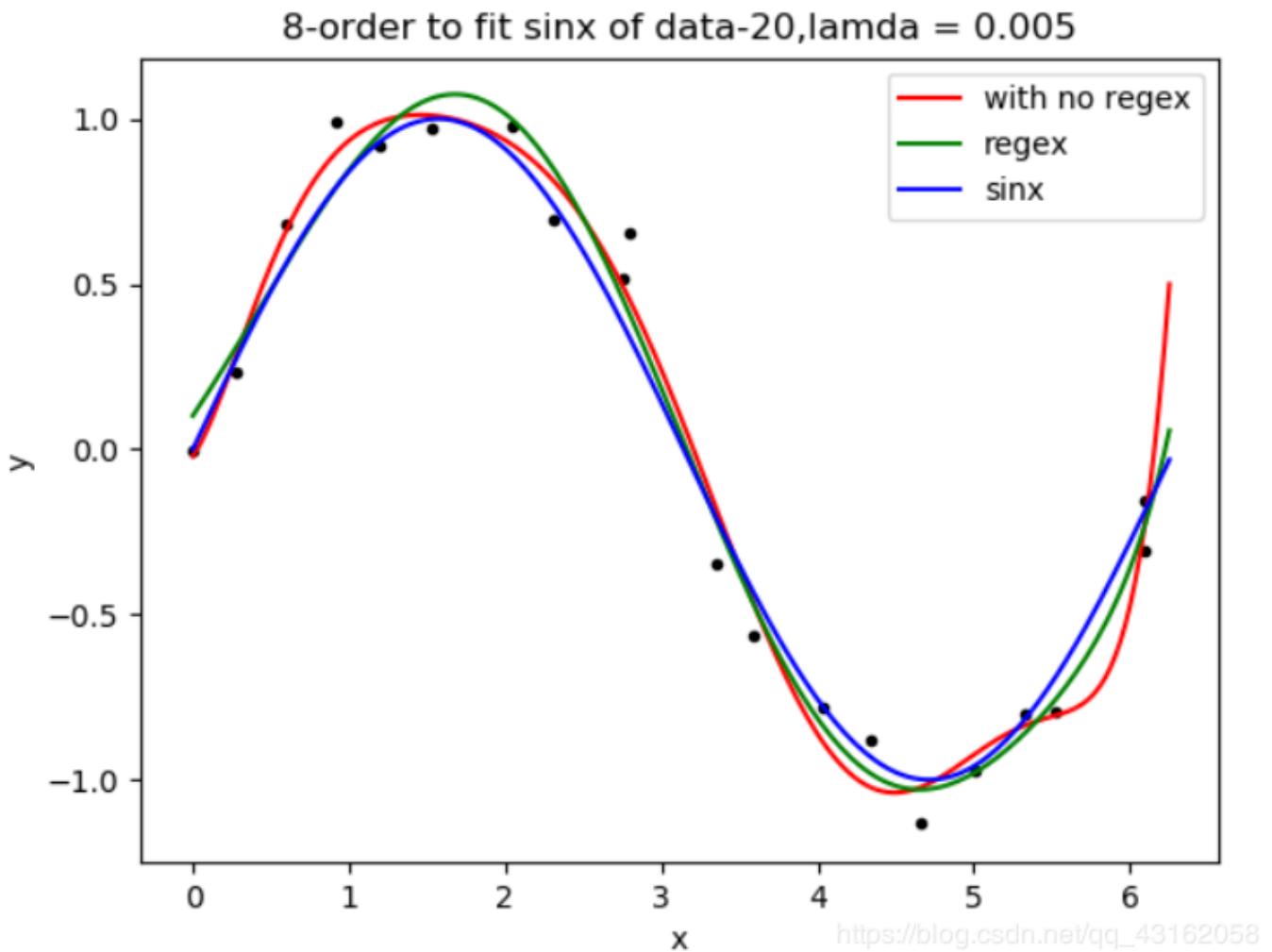
8-order to fit sinx of data-20, lamda = 1e-06



https://blog.csdn.net/qq_43162058

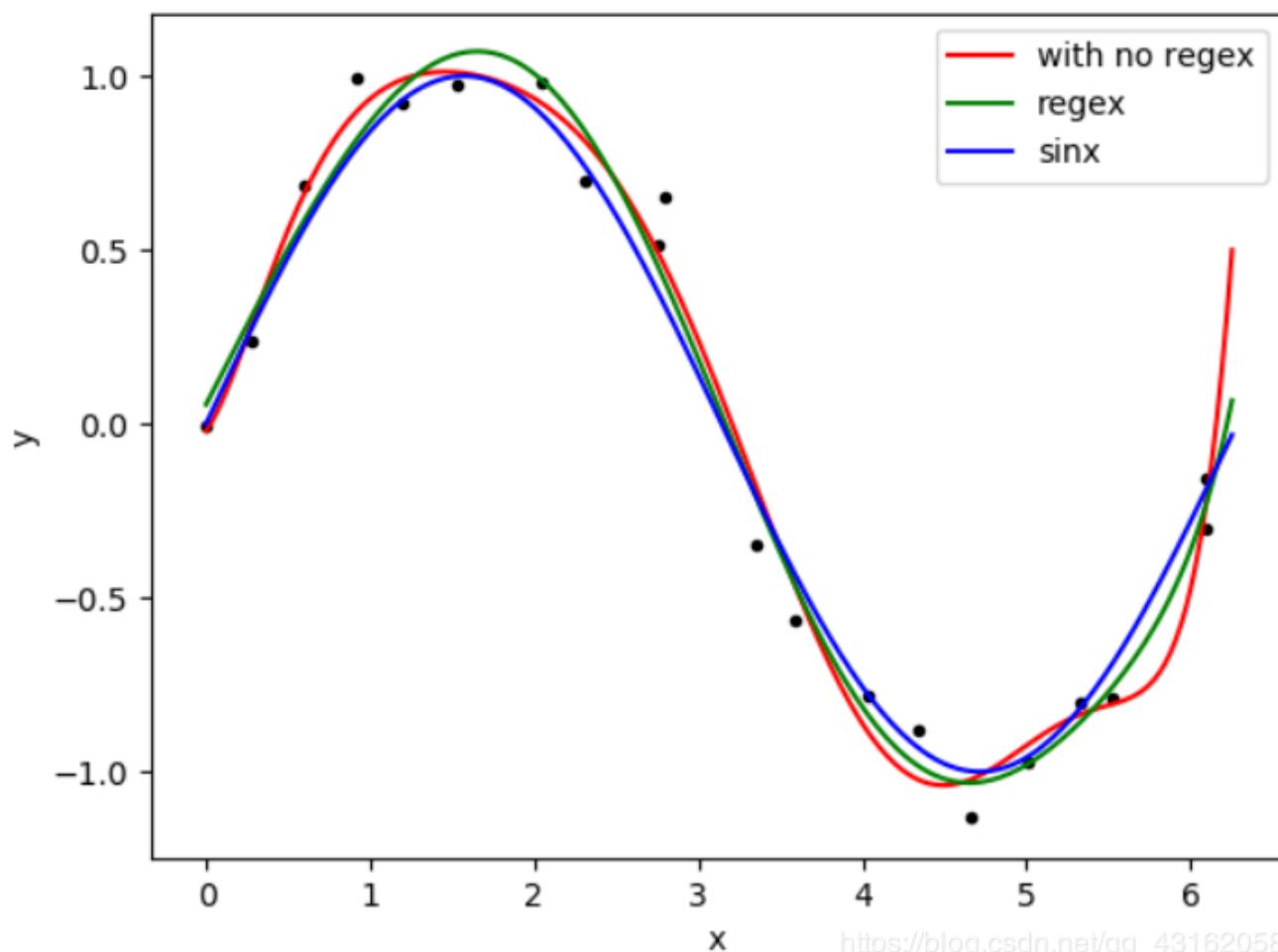
由以上信息可以得出， λ 在从0.1到1e-6变化的过程中，曲线有一个拟合先变好再过拟合的过程，我们对lambda取0.01至0.001之间的数进行分析，寻找比较好的值。

先取0.005



再取一下0.002,可以看出和0.001, 0.005以及0.01区别不大, 可以猜测应该是受数据量的影响, 所以拟合得不好。接下来我们减少数据量为10

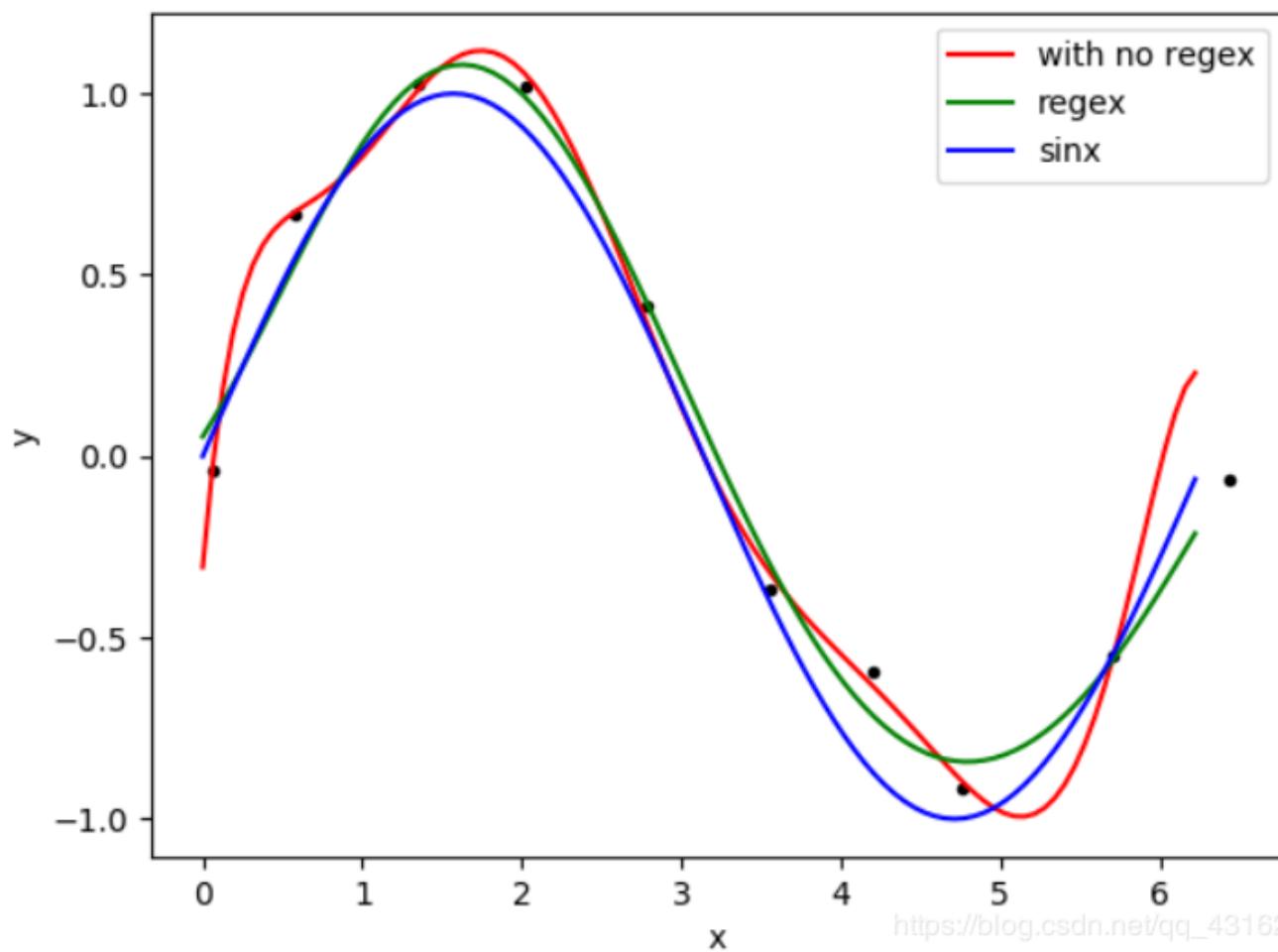
8-order to fit sinx of data-20, lamda = 0.002



https://blog.csdn.net/qq_43162058

$\lambda = 0.01$

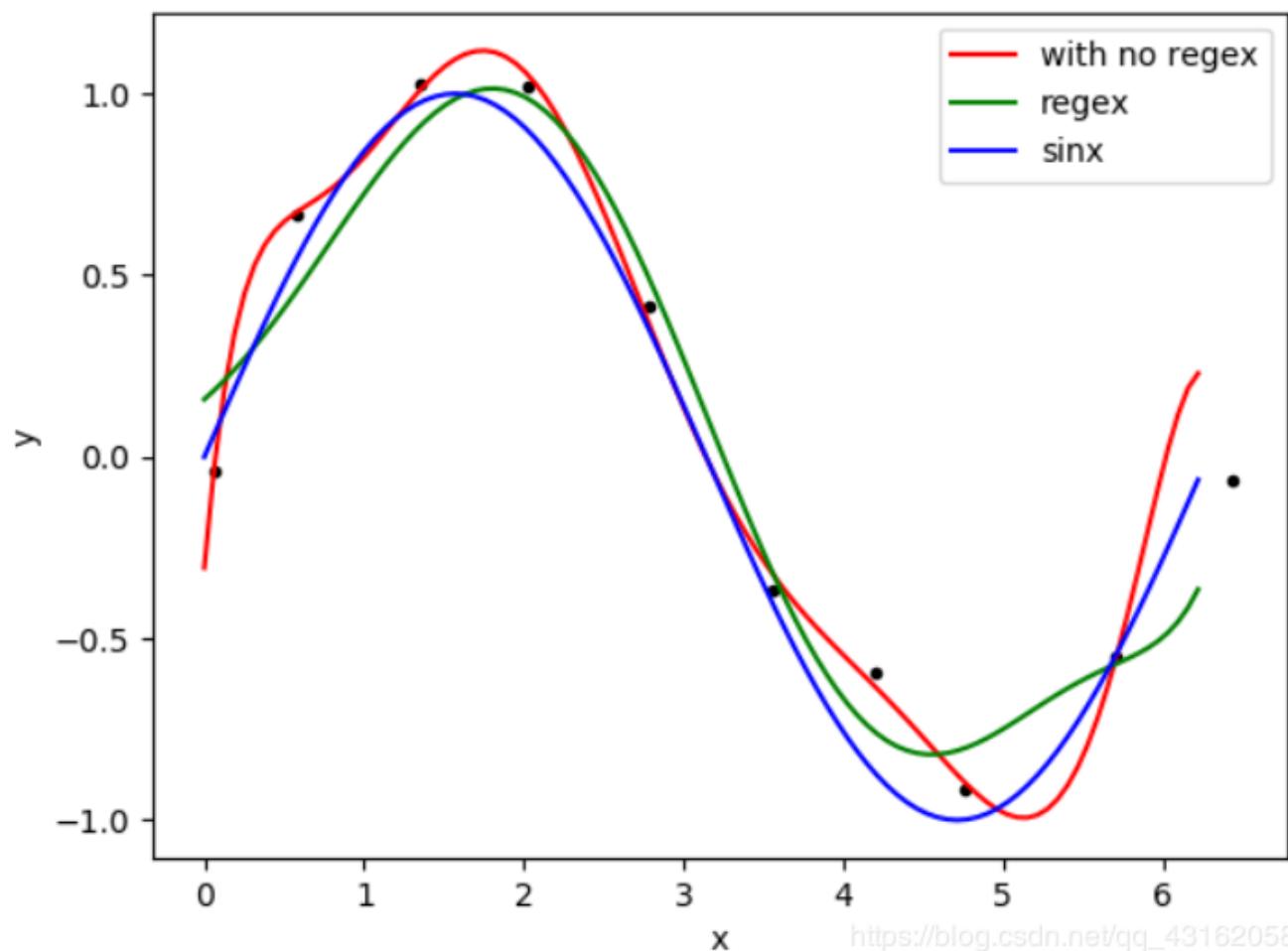
8-order to fit sinx of data-10, lamda = 0.01



https://blog.csdn.net/qq_43162058

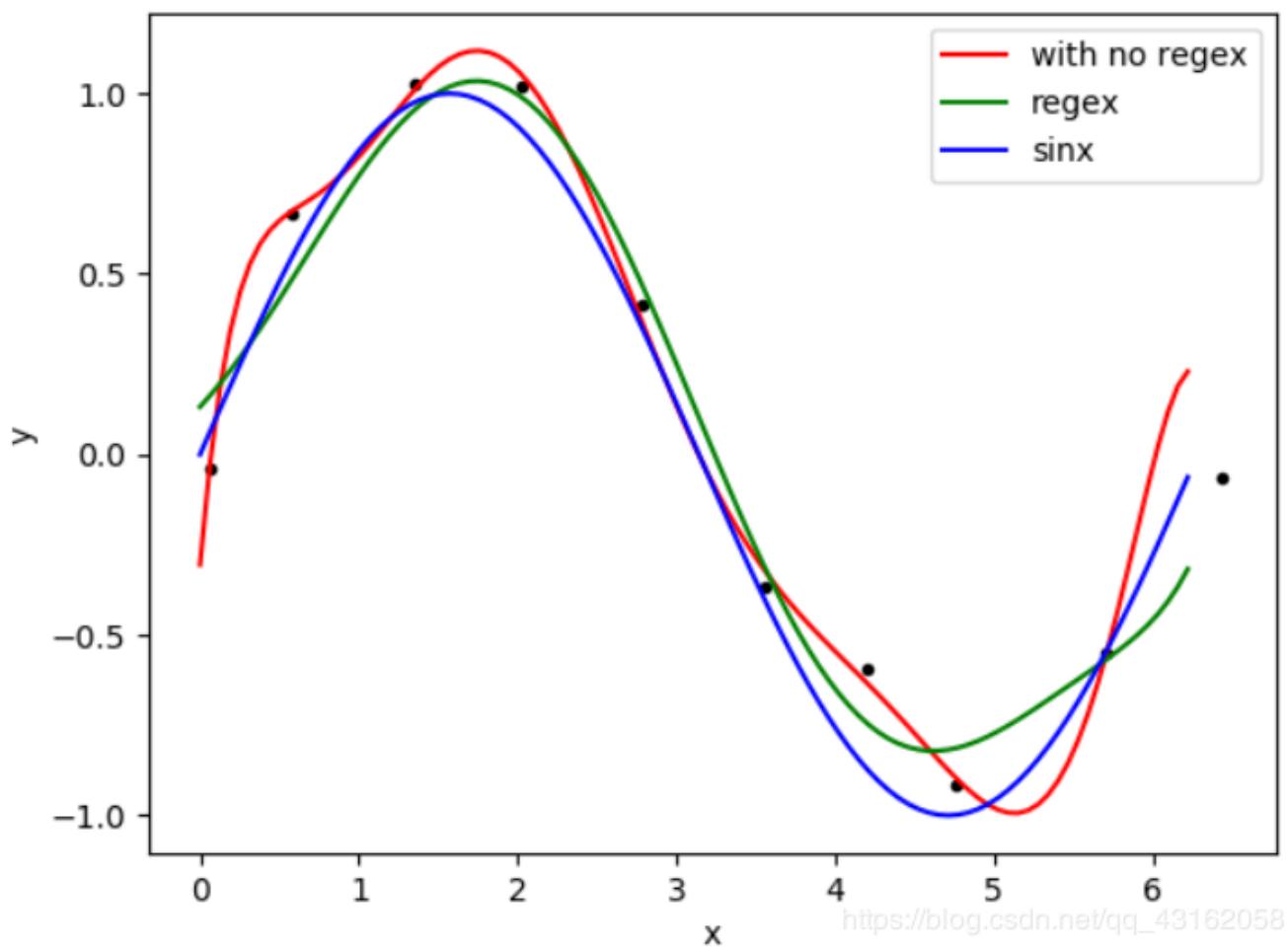
$\lambda = 0.05$ 可以看出效果仍然不太好，调小一点

8-order to fit sinx of data-10, lamda = 0.05



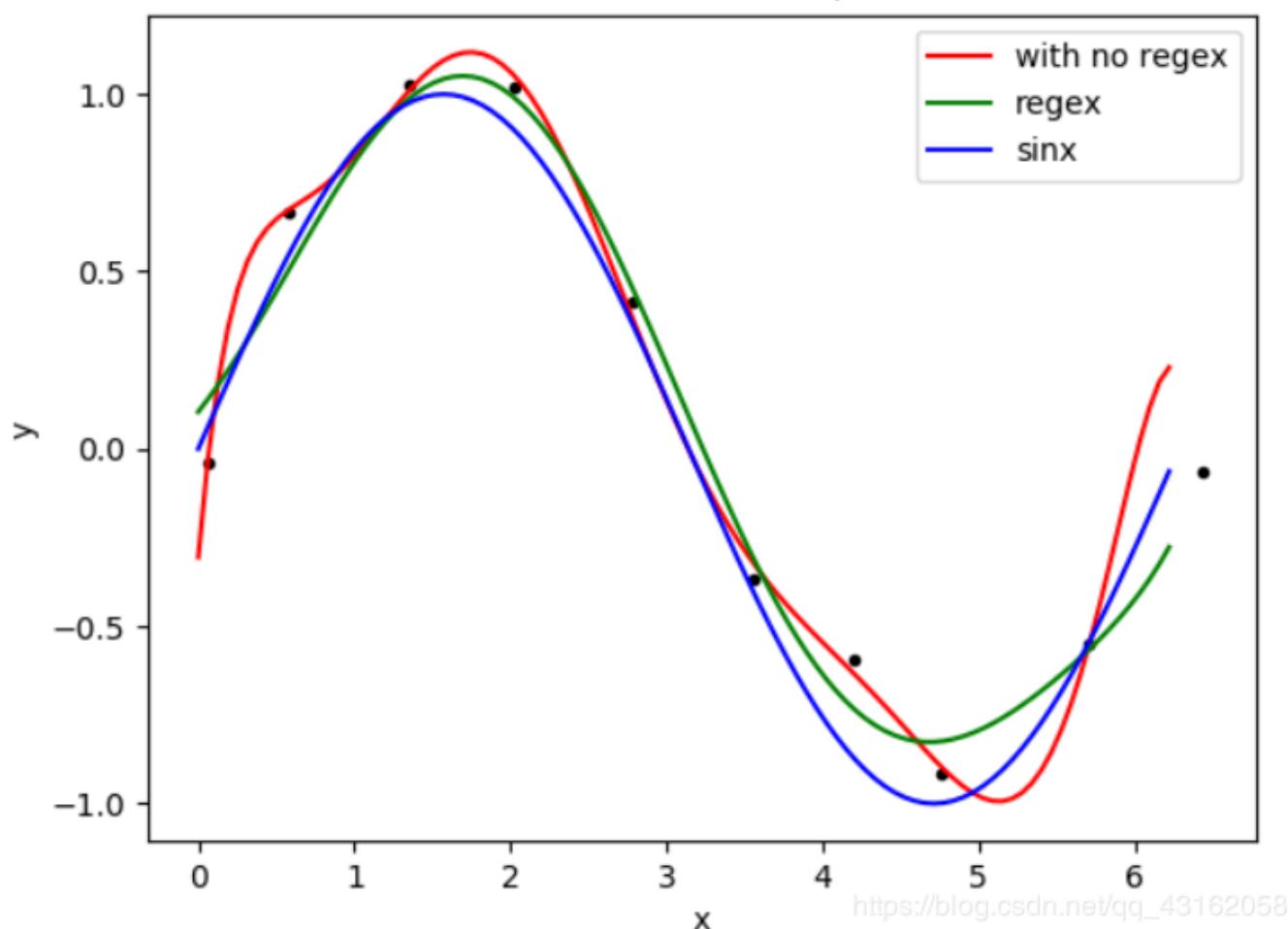
$\lambda = 0.03$ 如下图，比0.05要好一些，但依然不太理想

8-order to fit sinx of data-10, lamda = 0.03



再取0.02，不如0.01效果好

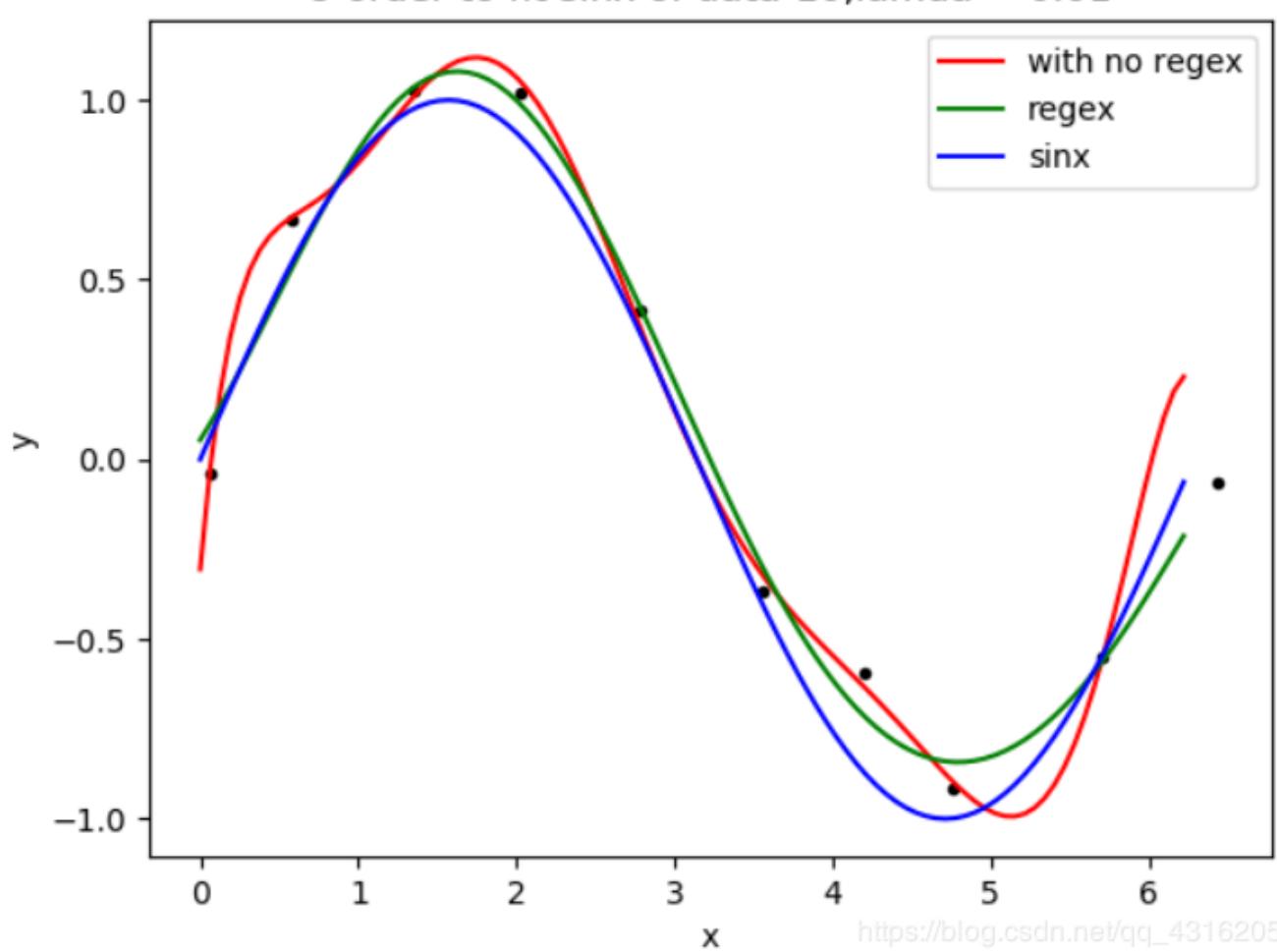
8-order to fit sinx of data-10, lamda = 0.02



https://blog.csdn.net/qq_43162058

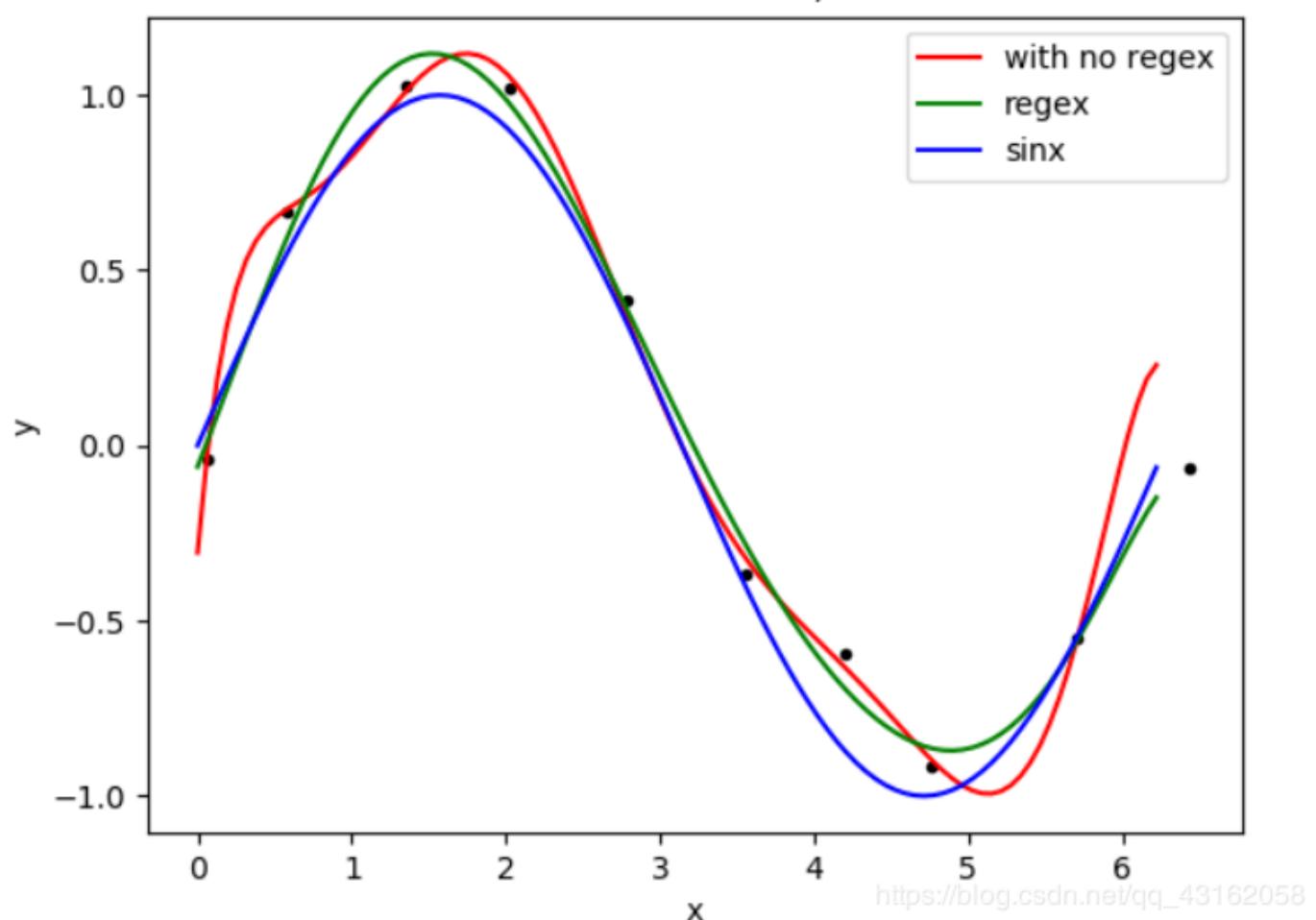
又继续减小，当 λ 在0.0001至0.01这个区间的拟合都是比较好的，已经不会发生过拟合了

8-order to fit sinx of data-10, lamda = 0.01



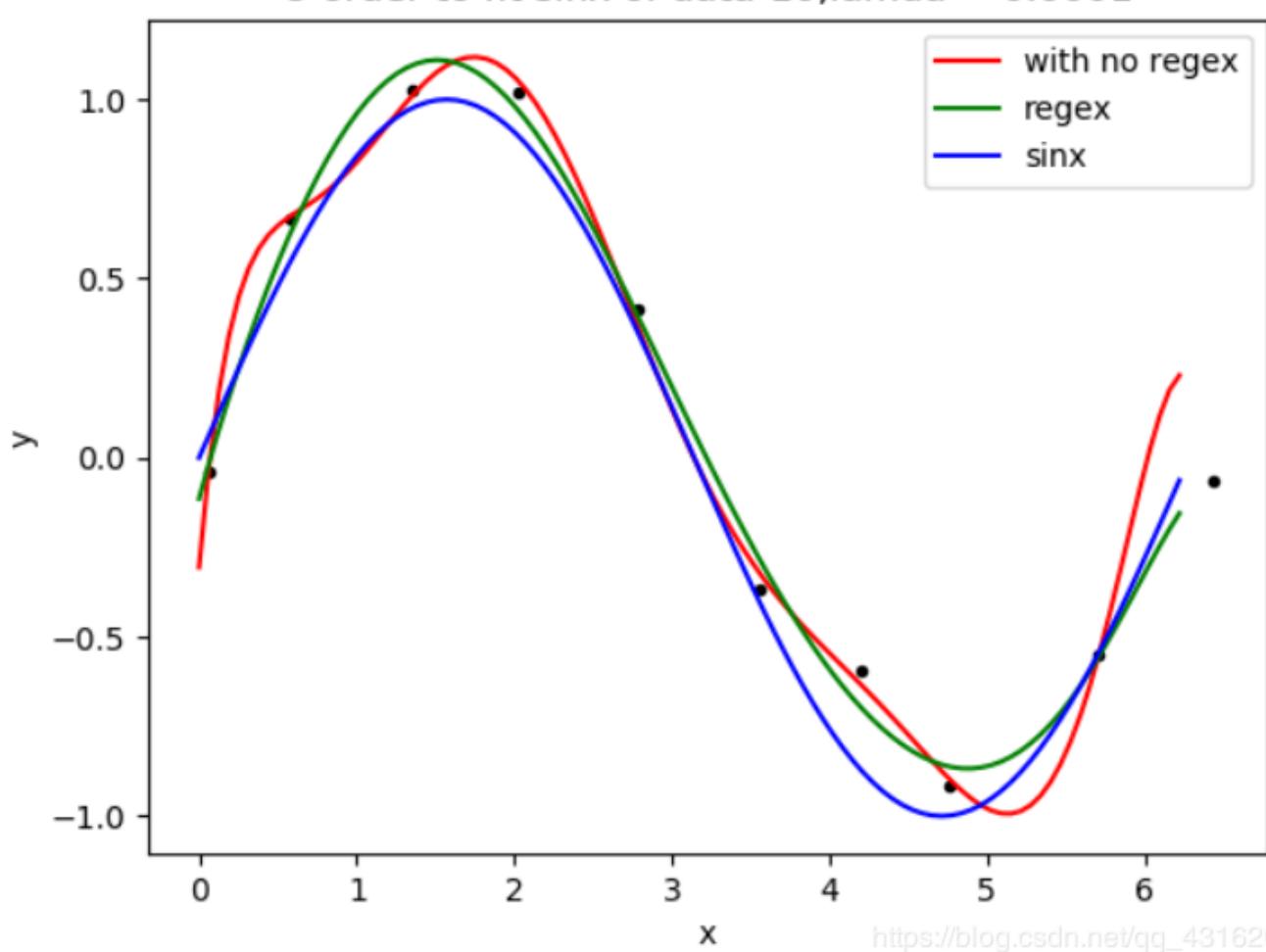
https://blog.csdn.net/qq_43162058

8-order to fit sinx of data-10, lamda = 0.001



https://blog.csdn.net/qq_43162058

8-order to fit sinx of data-10, lamda = 0.0001



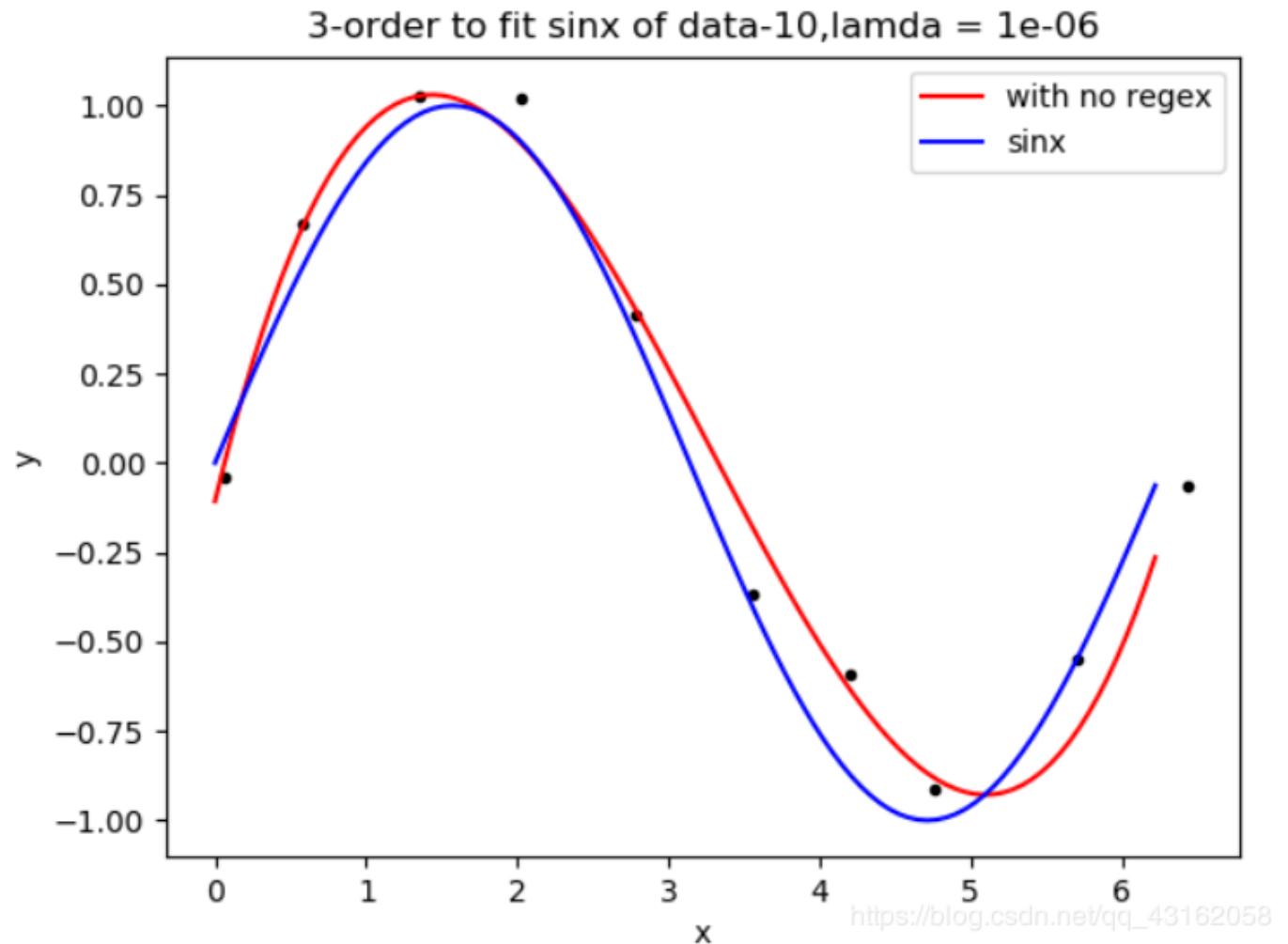
https://blog.csdn.net/qq_43162058

从上面的分析过程还可以得出，不同阶数不同数据量的入取值各不相同，当阶数为8，数据量在50以上时，基本不需要正则化也可以拟合得很好

4.2.3 关于不同阶数的拟合效果 (研究时我们将lambda调到最佳, 以数据量为10作为研究)

由前面的可以看出一二阶拟合效果并不好, 因此不作进一步观察

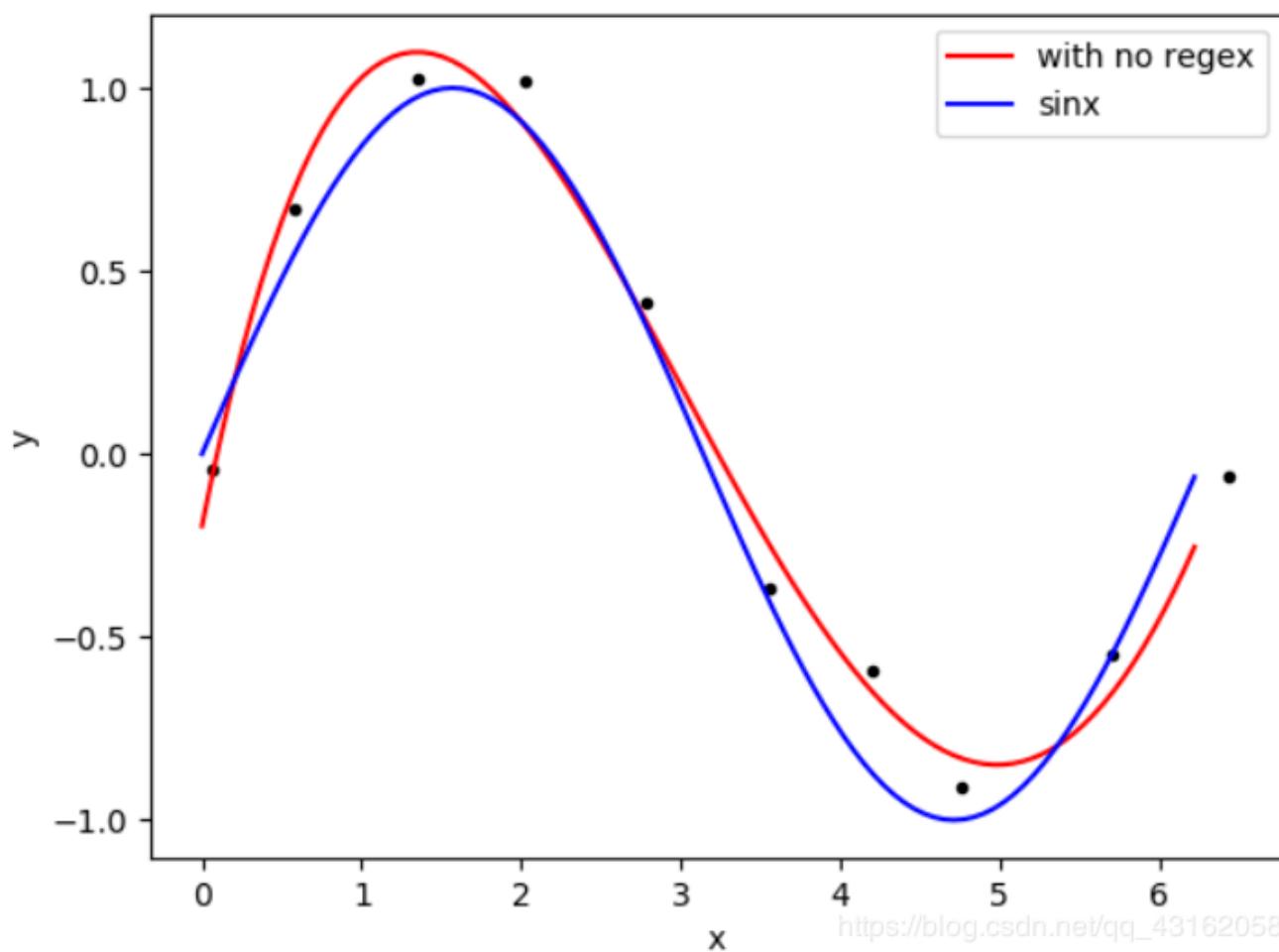
三阶



四阶

https://blog.csdn.net/qq_43162058

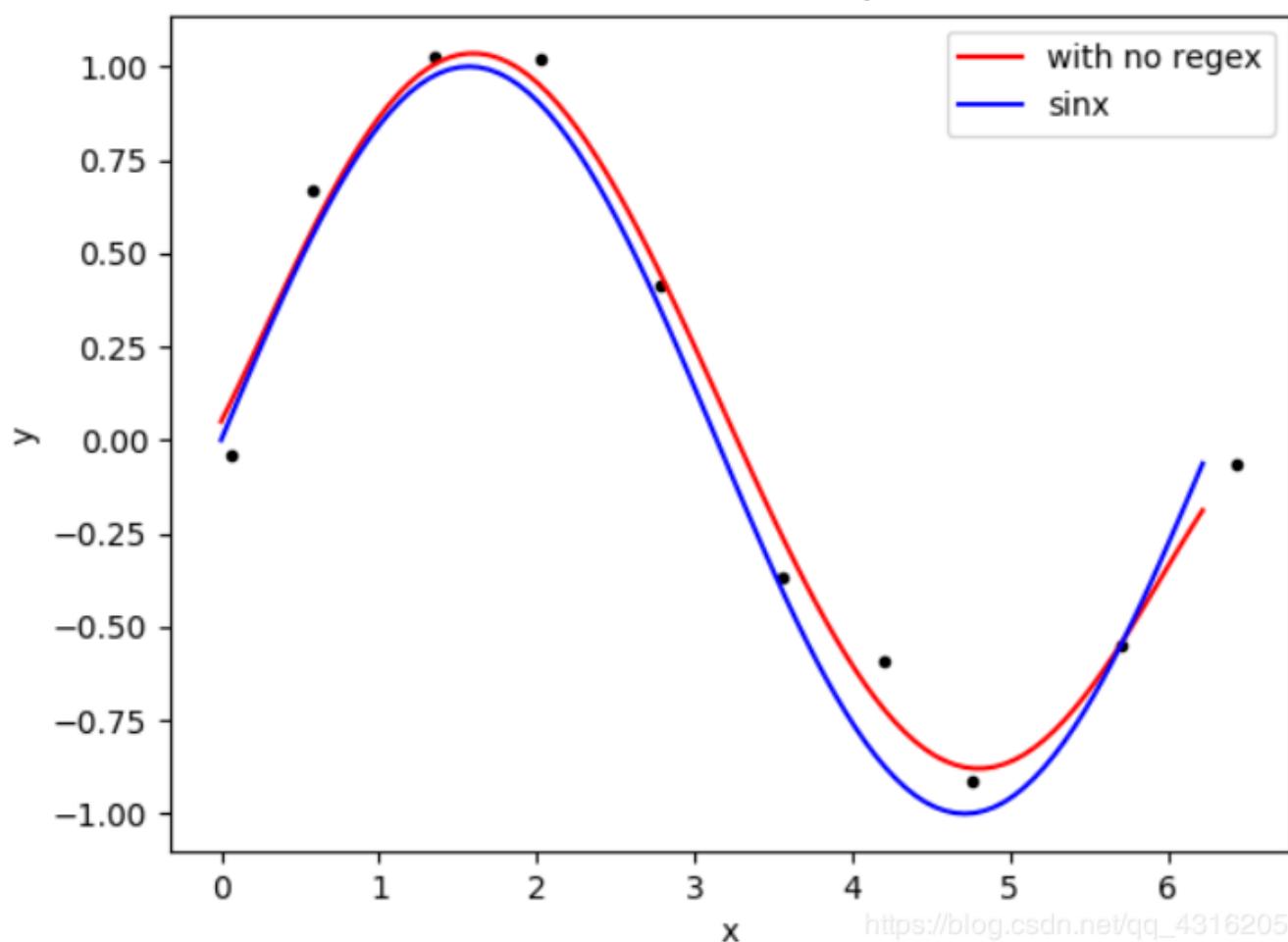
4-order to fit sinx of data-10, lamda = 0.0002



https://blog.csdn.net/qq_43162058

五阶

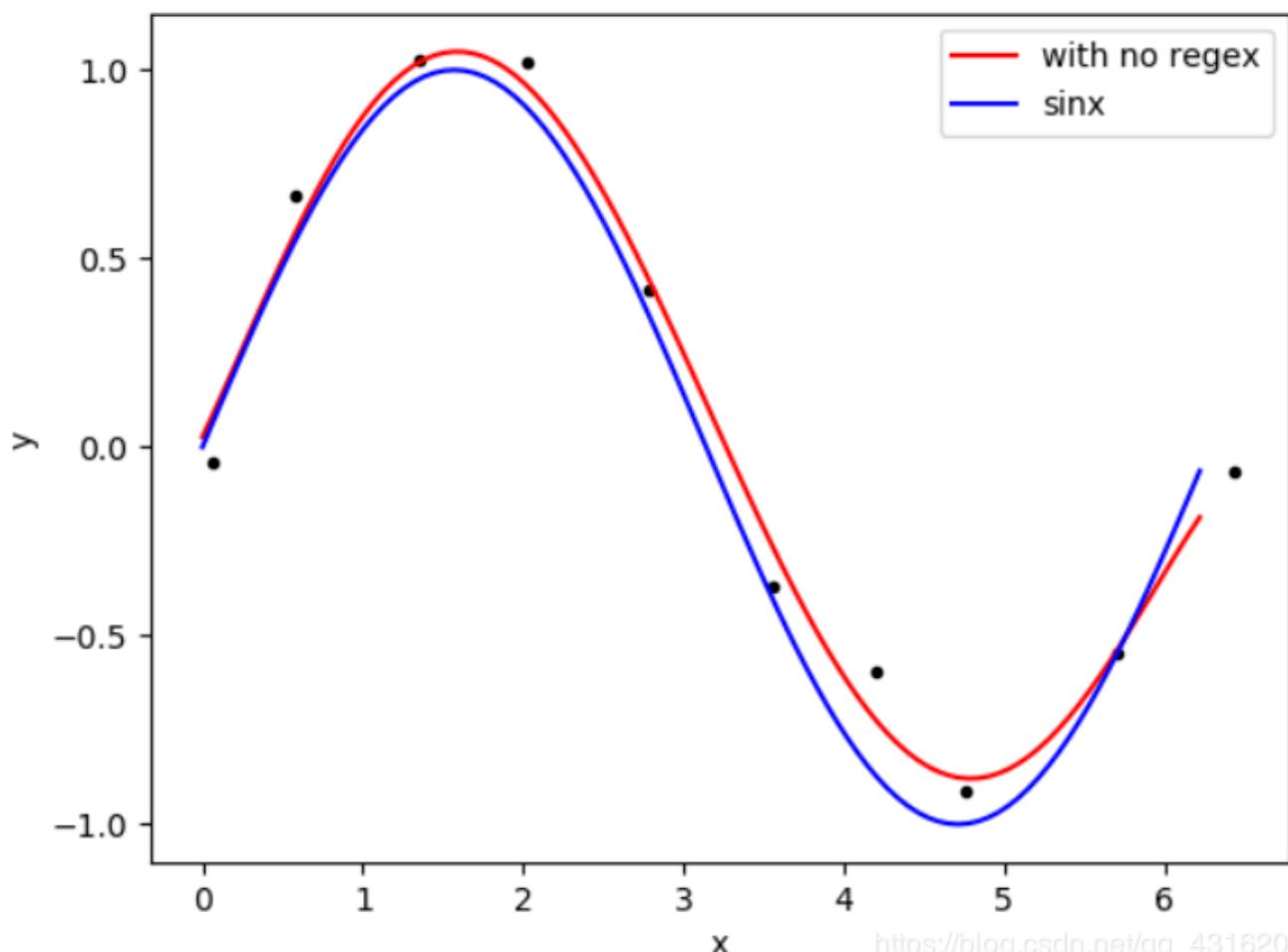
5-order to fit sinx of data-10, lamda = 0.01



https://blog.csdn.net/qq_43162058

六阶

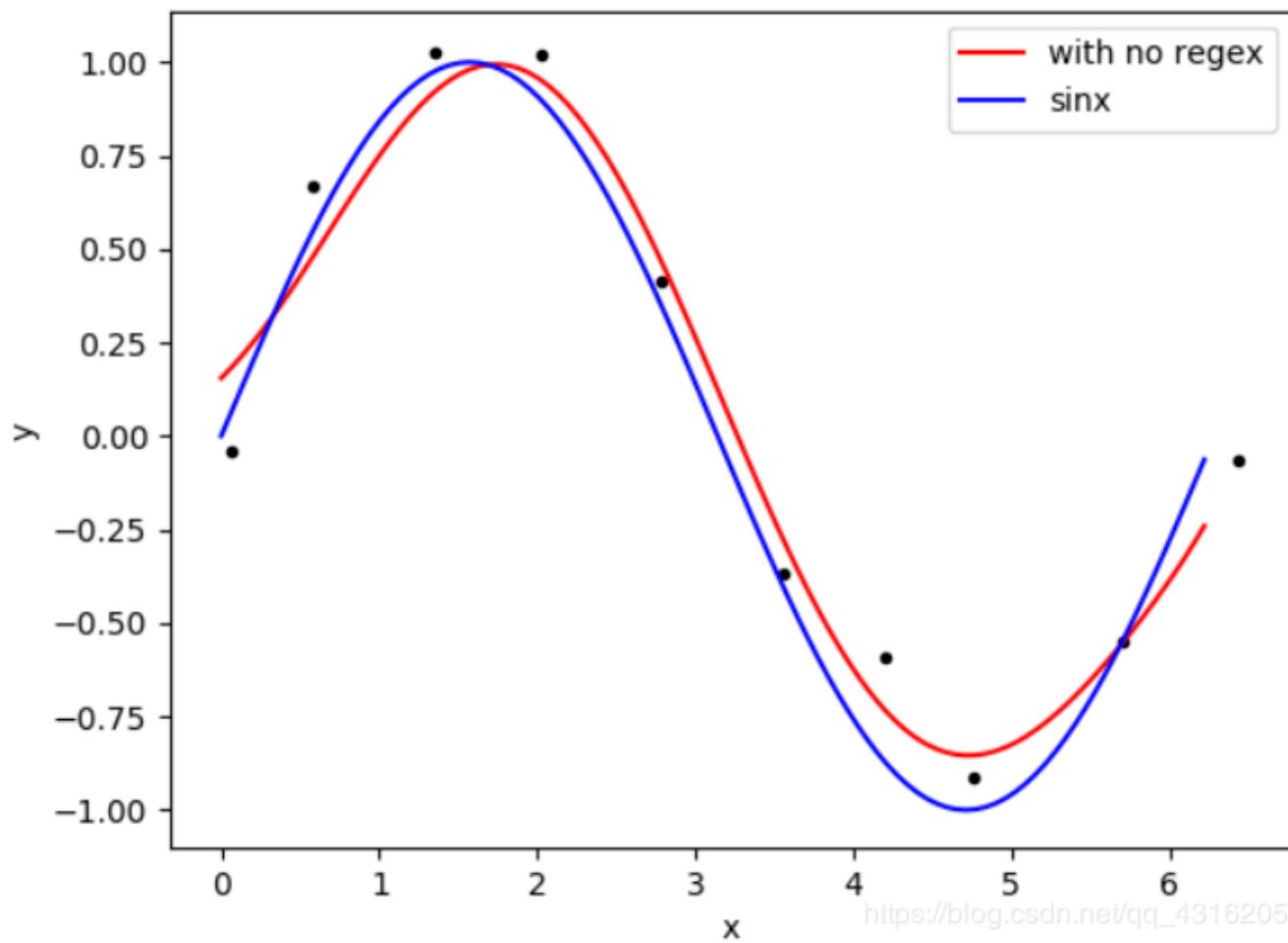
6-order to fit sinx of data-10, lamda = 0.007



https://blog.csdn.net/qq_43162058

七阶

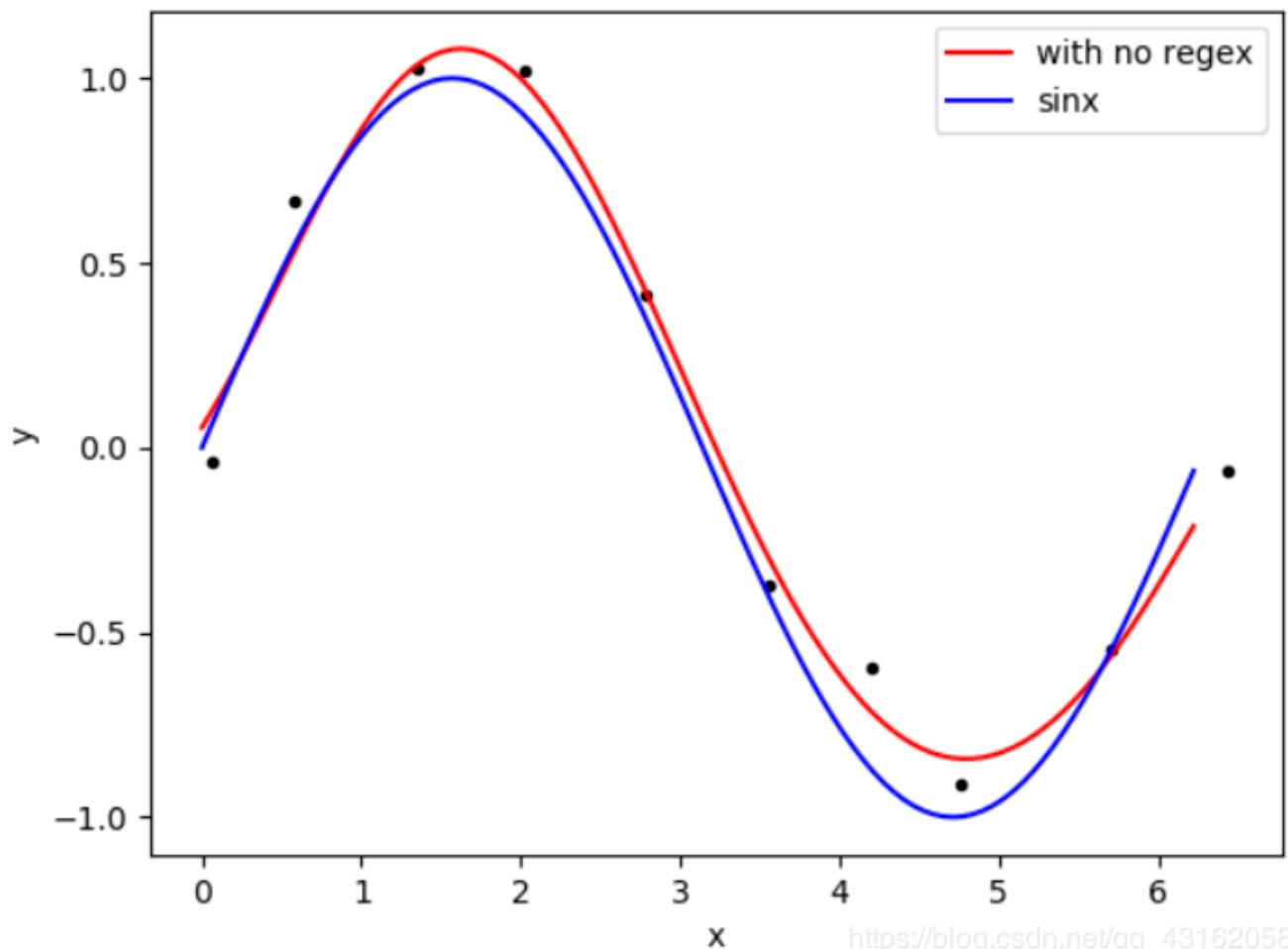
7-order to fit sinx of data-10, lamda = 0.05



https://blog.csdn.net/qq_43162058

八阶

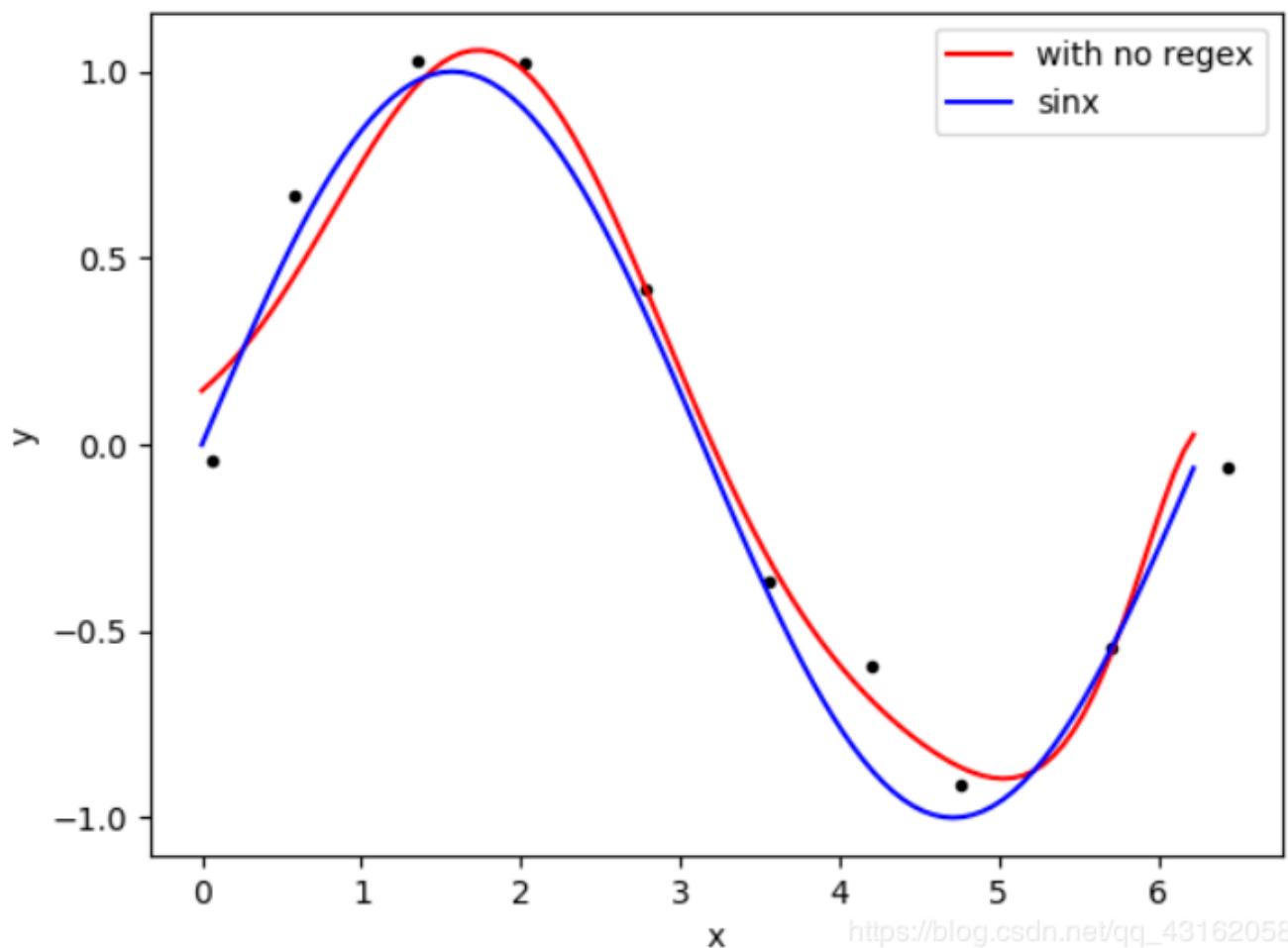
8-order to fit sinx of data-10, lamda = 0.01



https://blog.csdn.net/jq_43162058

九阶

9-order to fit sinx of data-10, lamda = 0.05



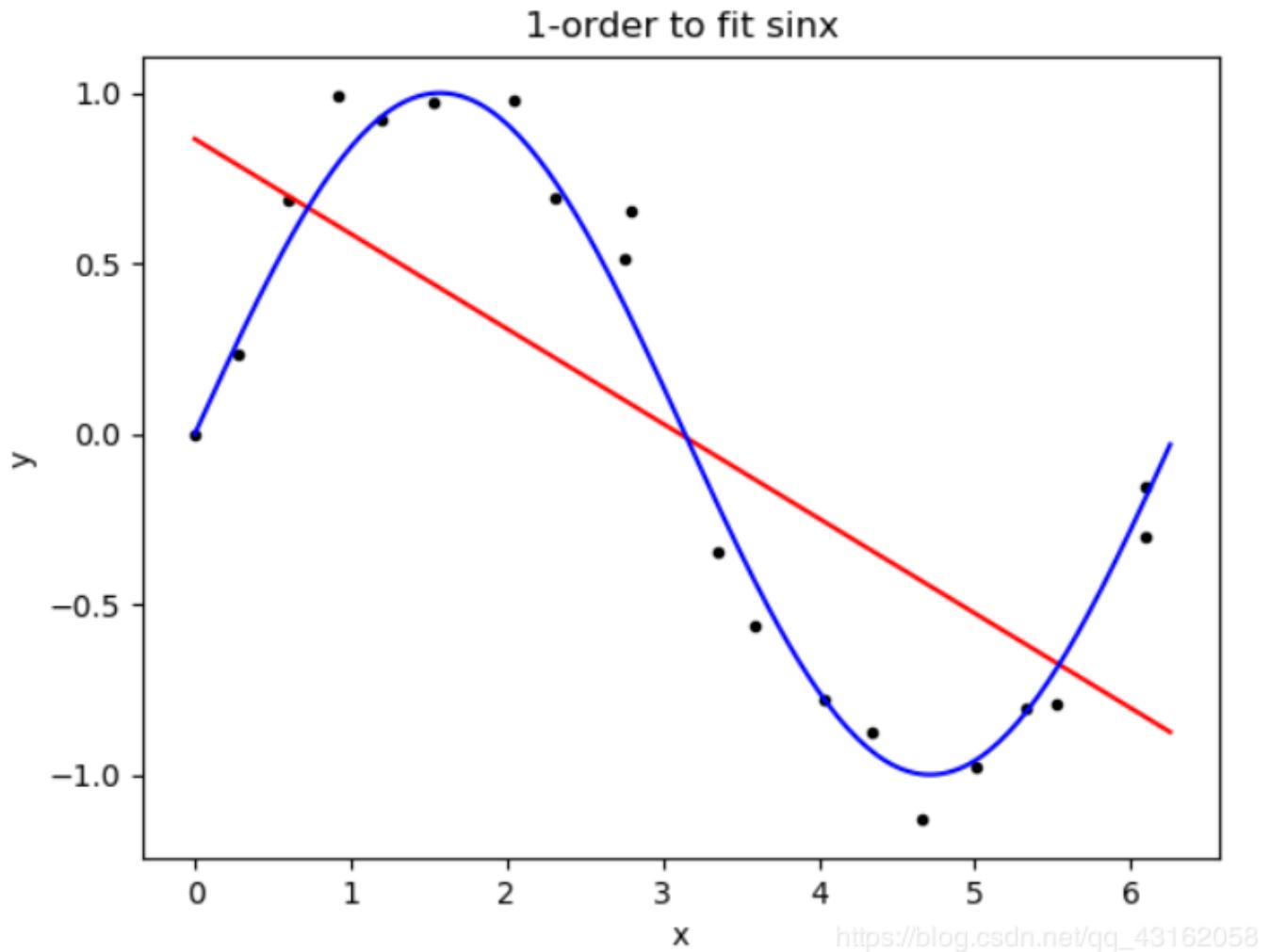
https://blog.csdn.net/jq_43162058

由上可以看出，三阶已经拟合得可以接受了，但是还不够，从5阶到8阶都可以拟合得很接近正弦曲线了，但是九阶的拟合依然有点过拟合（也可能是对于参数的寻找要求更高一些）。

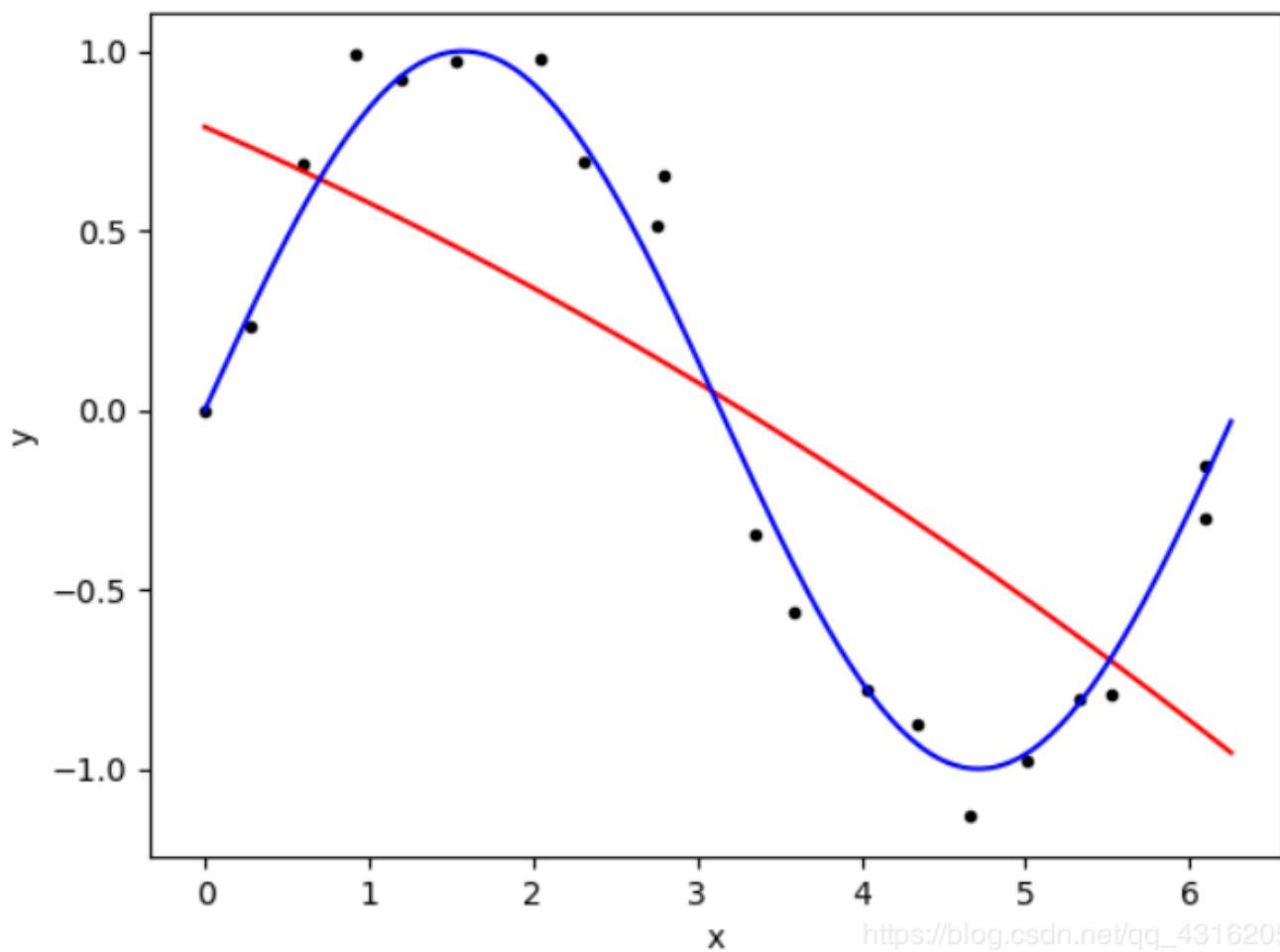
4.3梯度下降法

4.3.1不同阶数的梯度下降法

lamda = 1e-7 ,e =1e-7

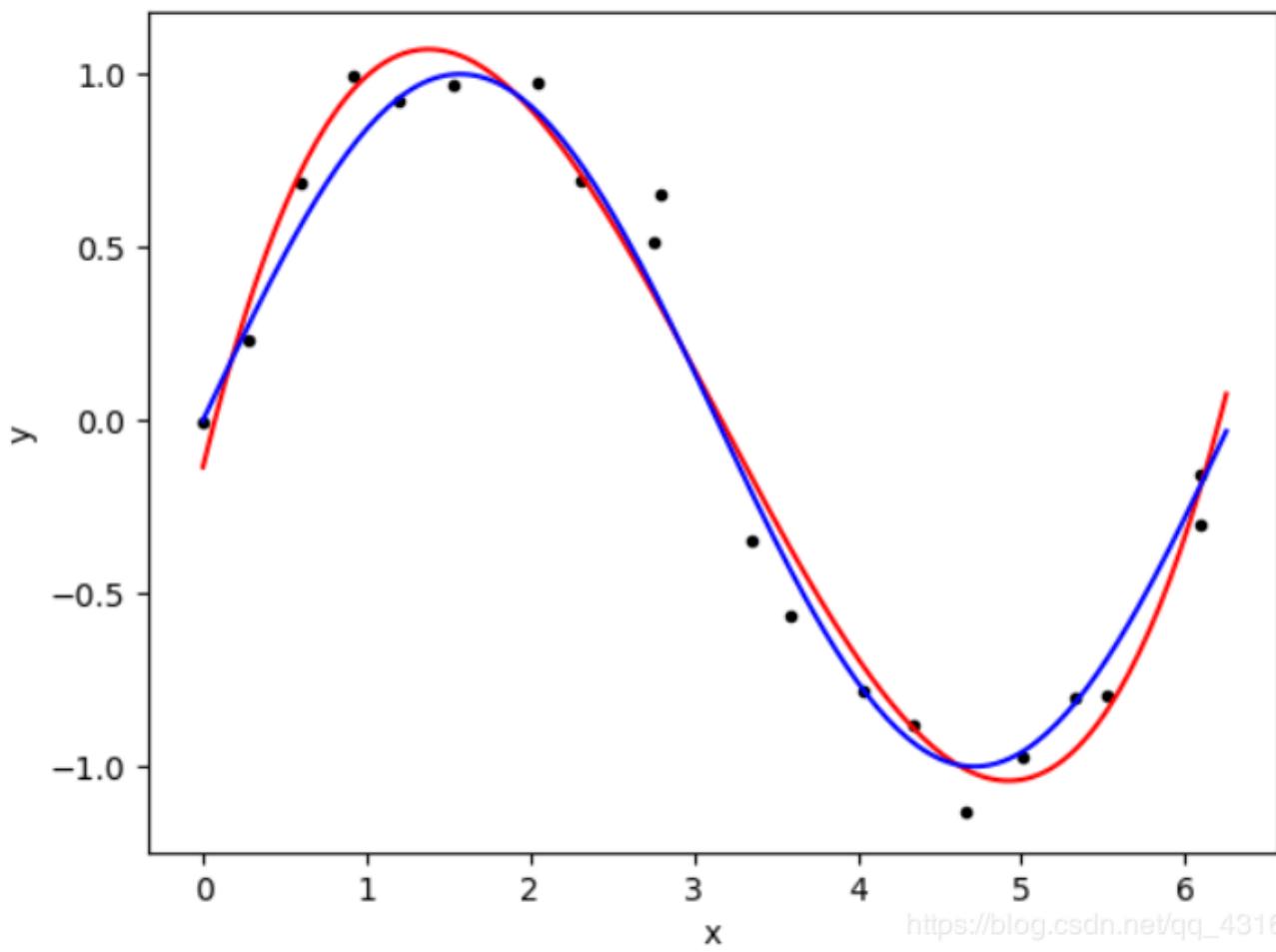


2-order to fit sinx



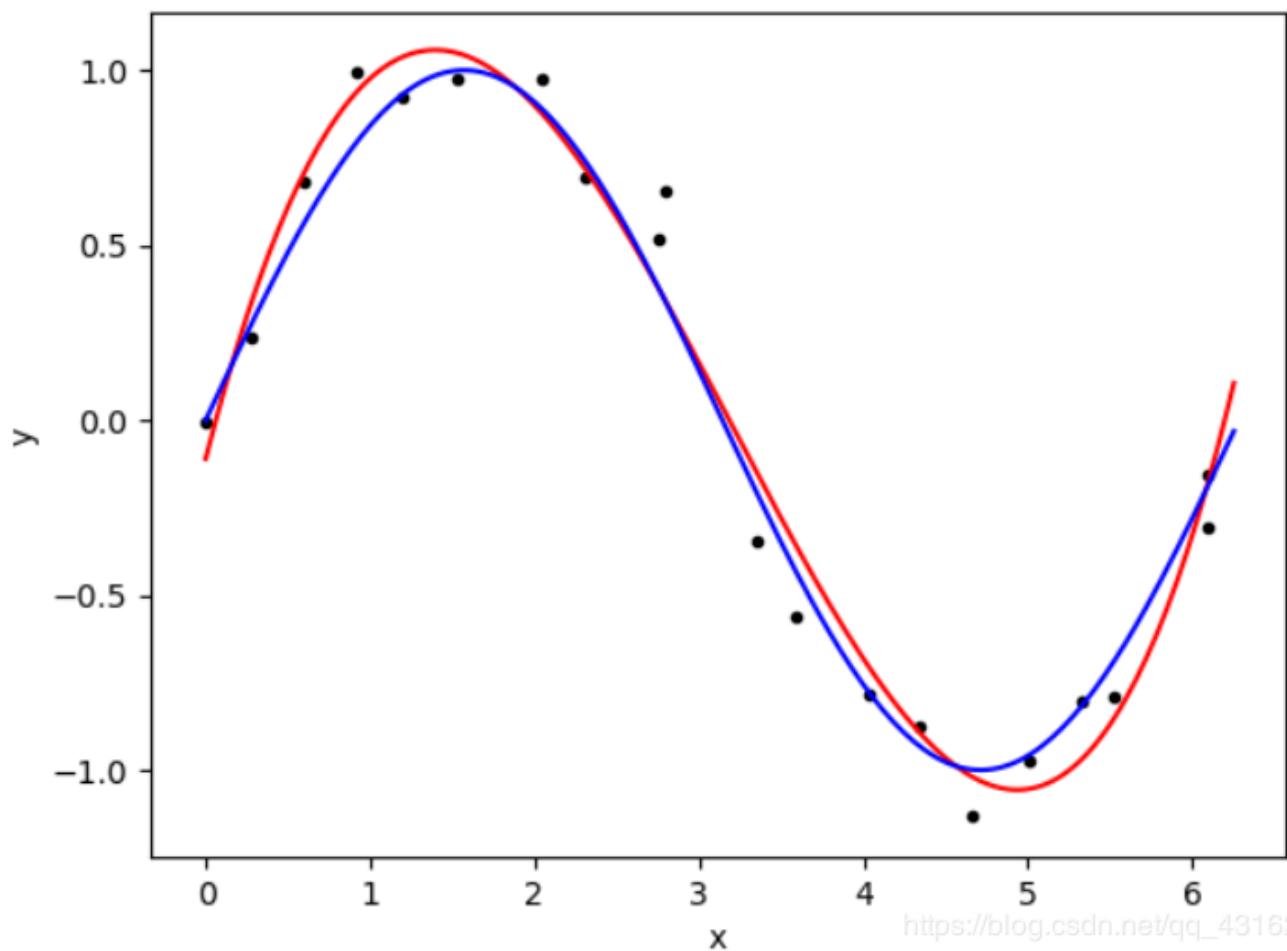
https://blog.csdn.net/qq_43162058

3-order to fit sinx



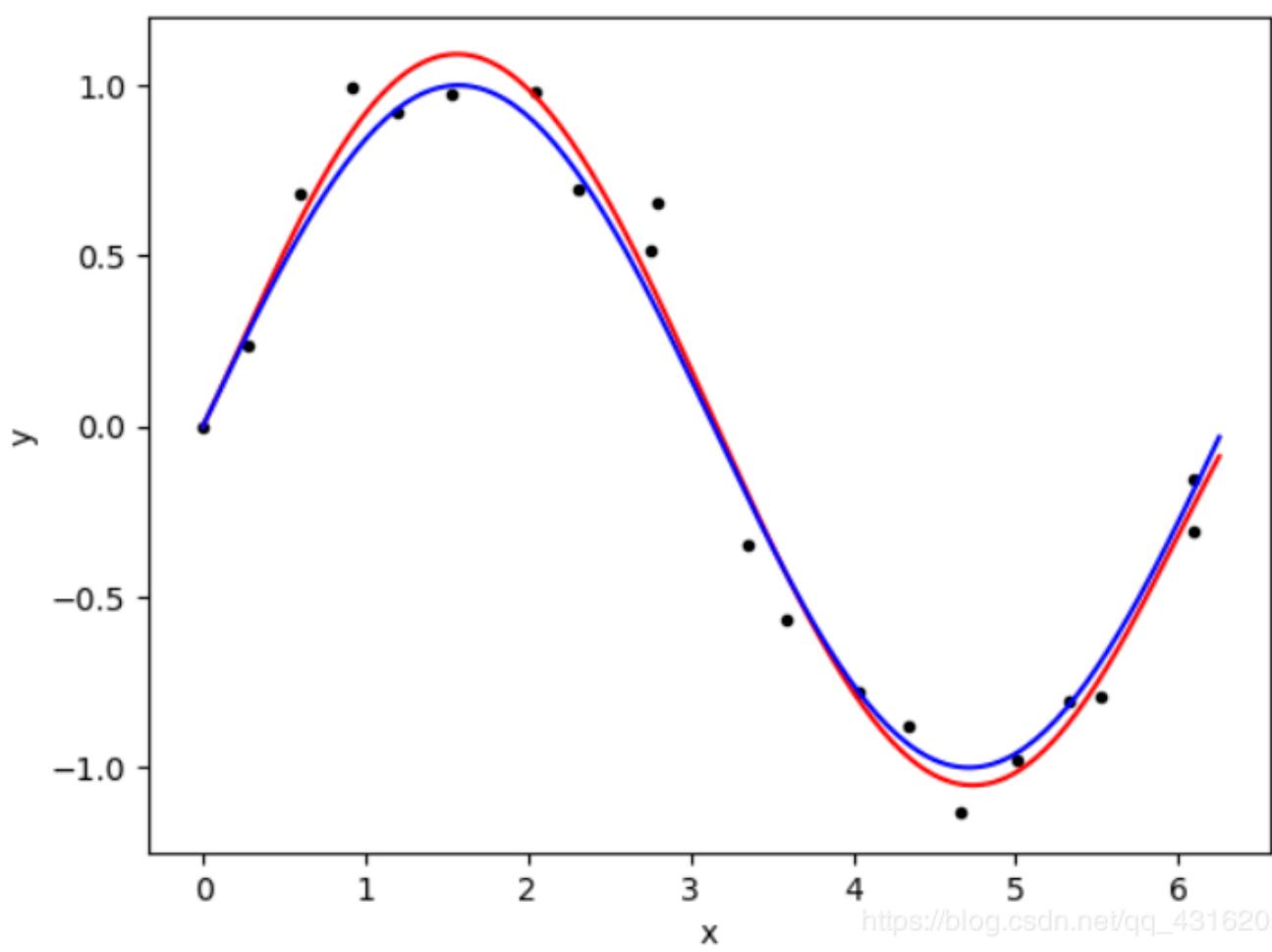
https://blog.csdn.net/qq_43162058

4-order to fit sinx



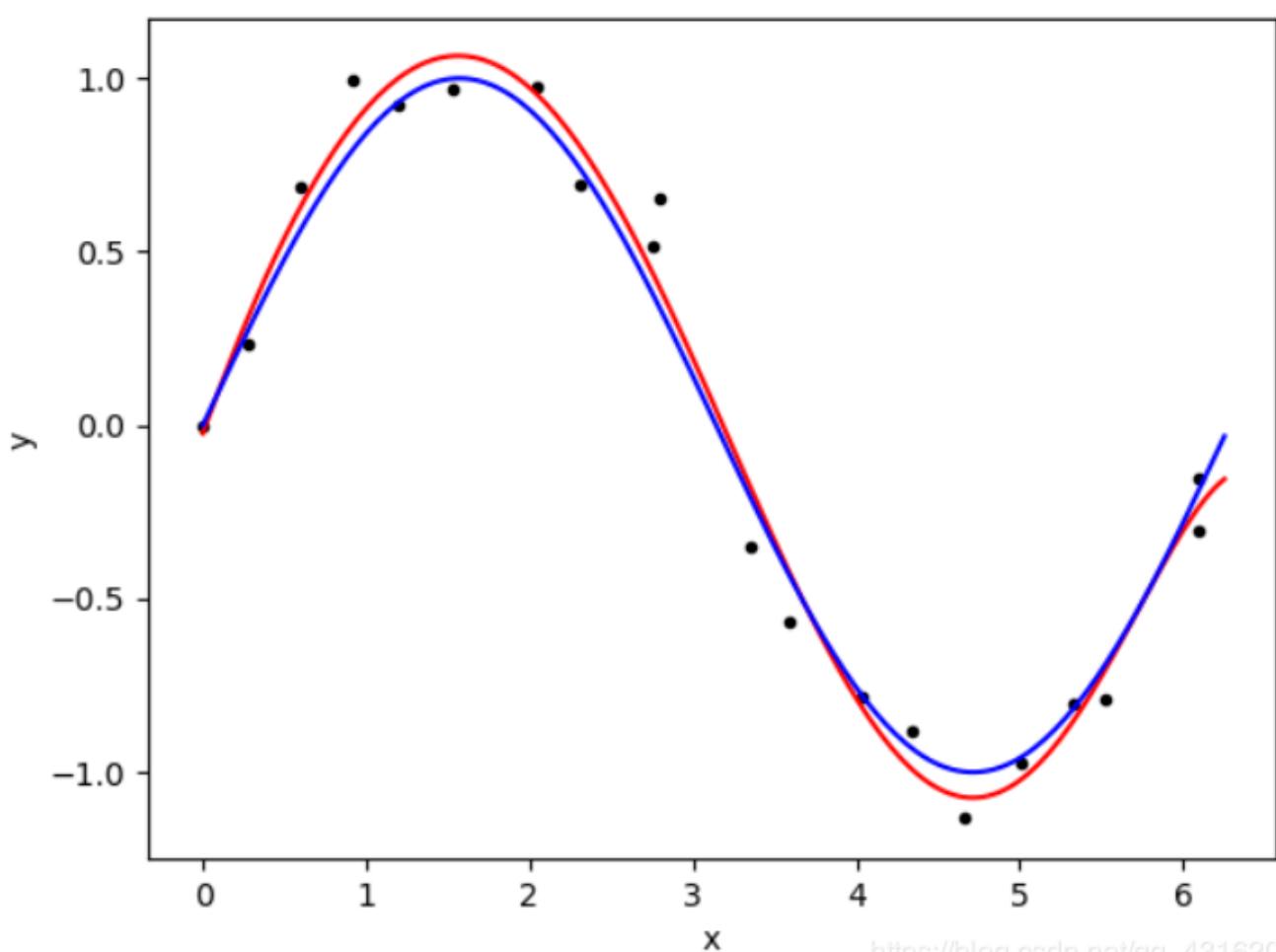
https://blog.csdn.net/qq_43162058

5-order to fit sinx



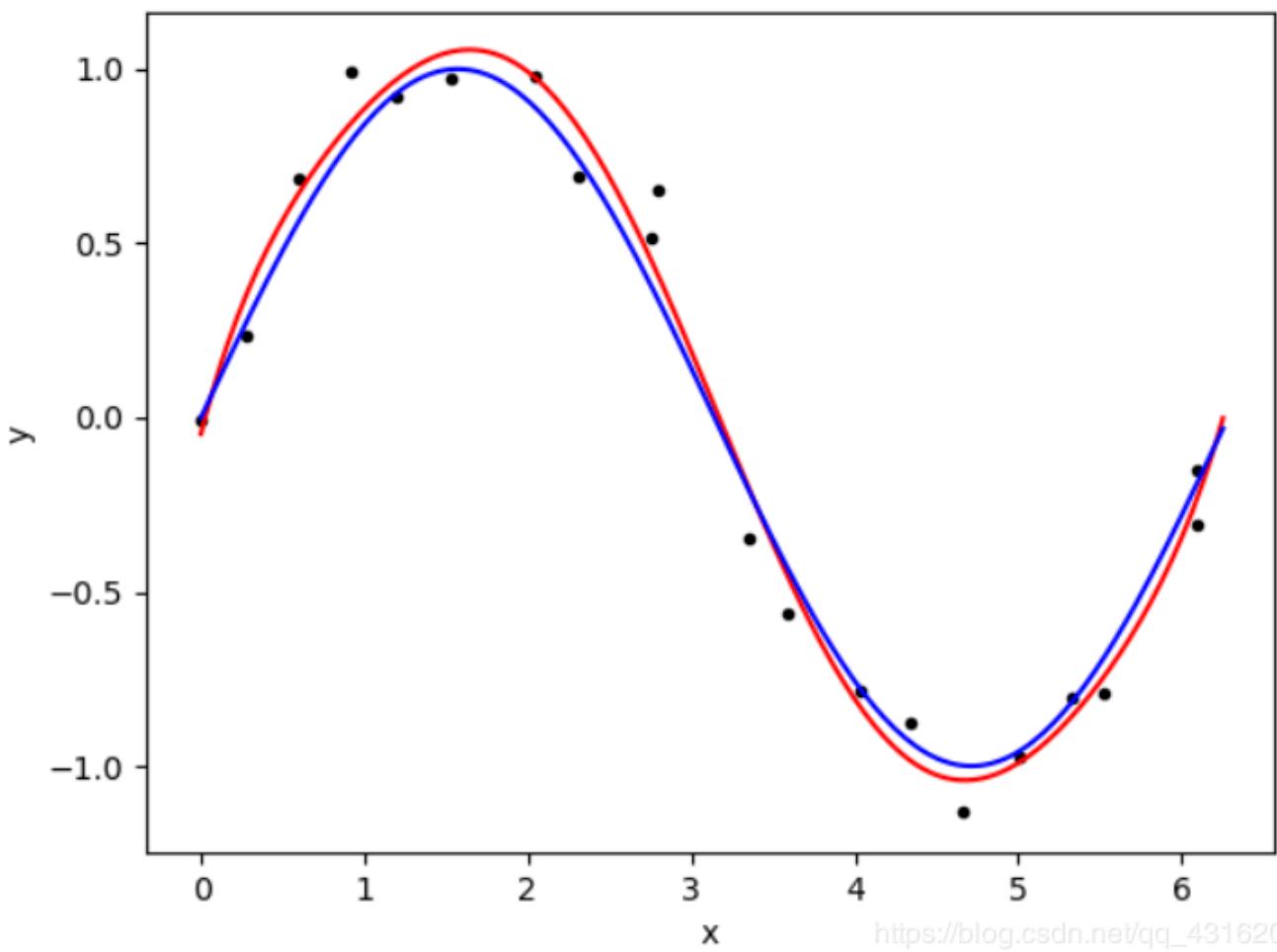
https://blog.csdn.net/qq_43162058

6-order to fit sinx



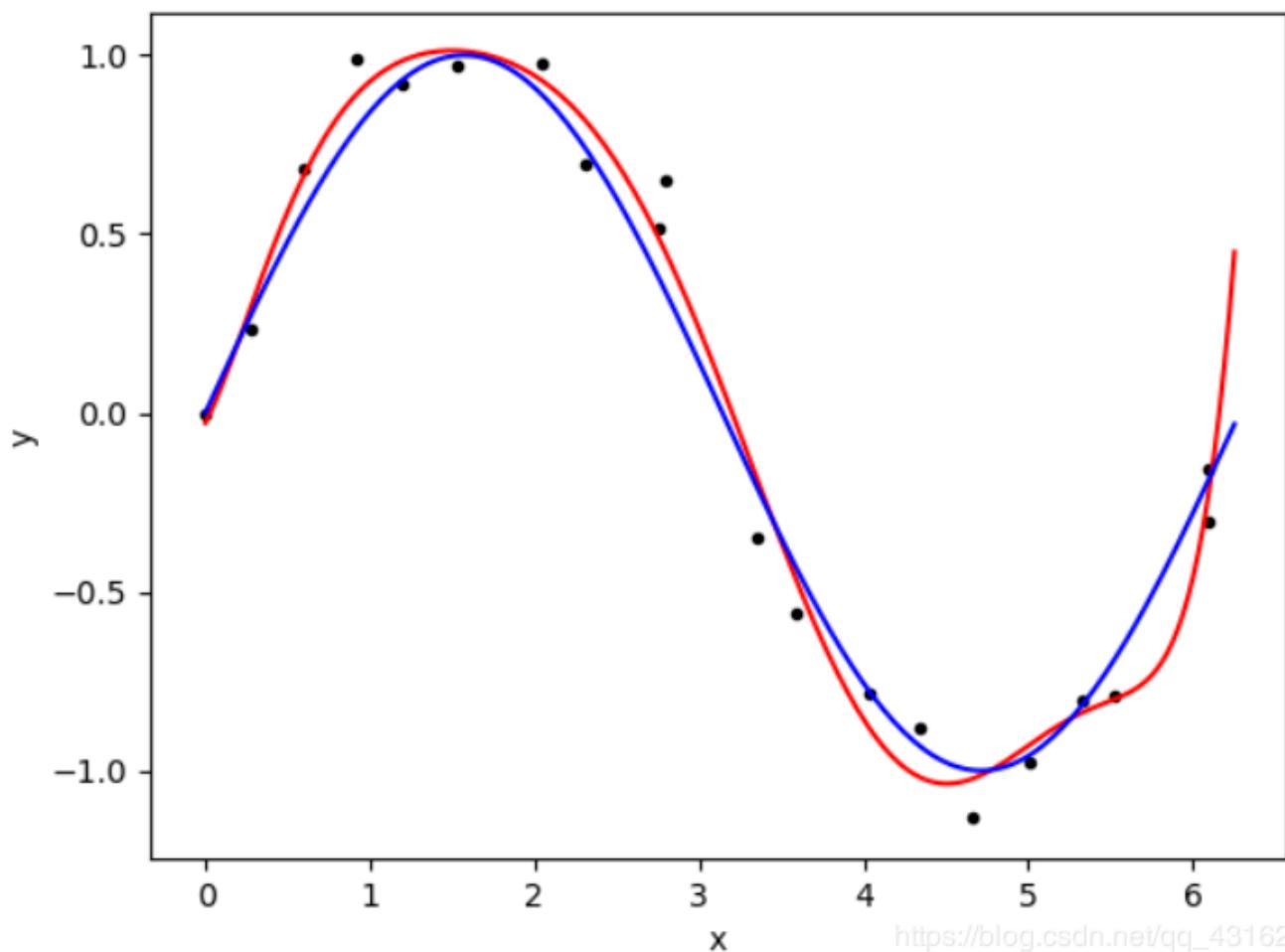
https://blog.csdn.net/qq_43162058

7-order to fit sinx

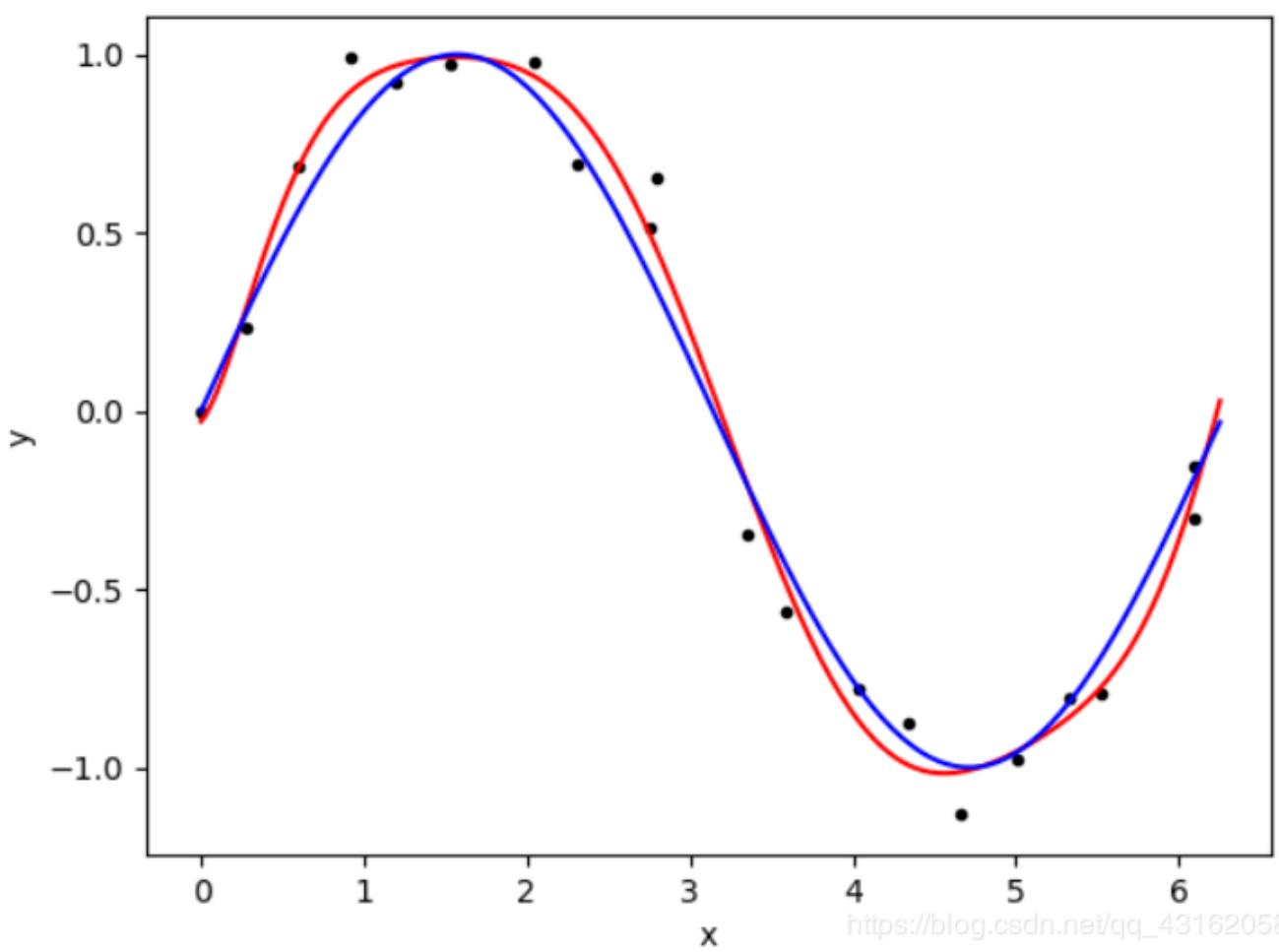


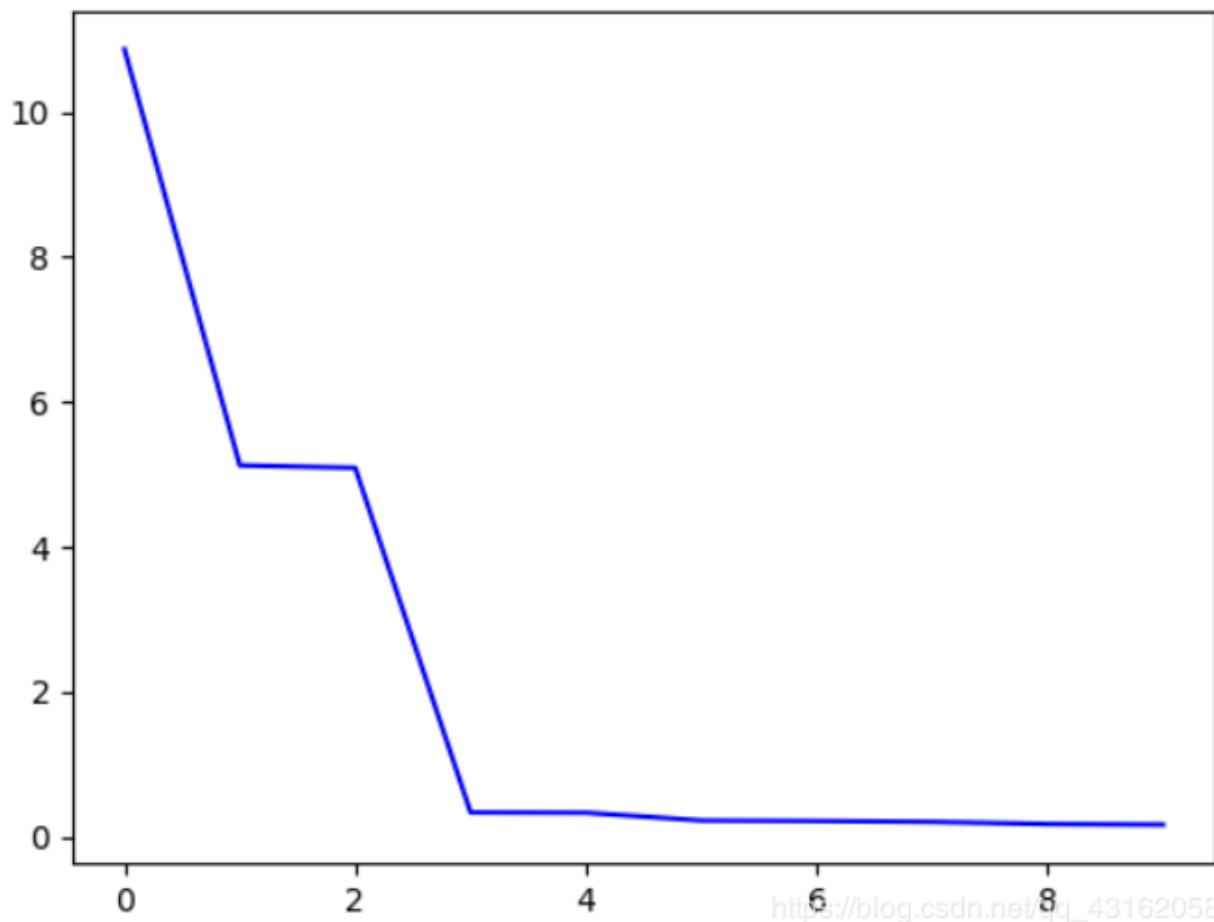
https://blog.csdn.net/qq_43162058

8-order to fit sinx



9-order to fit sinx

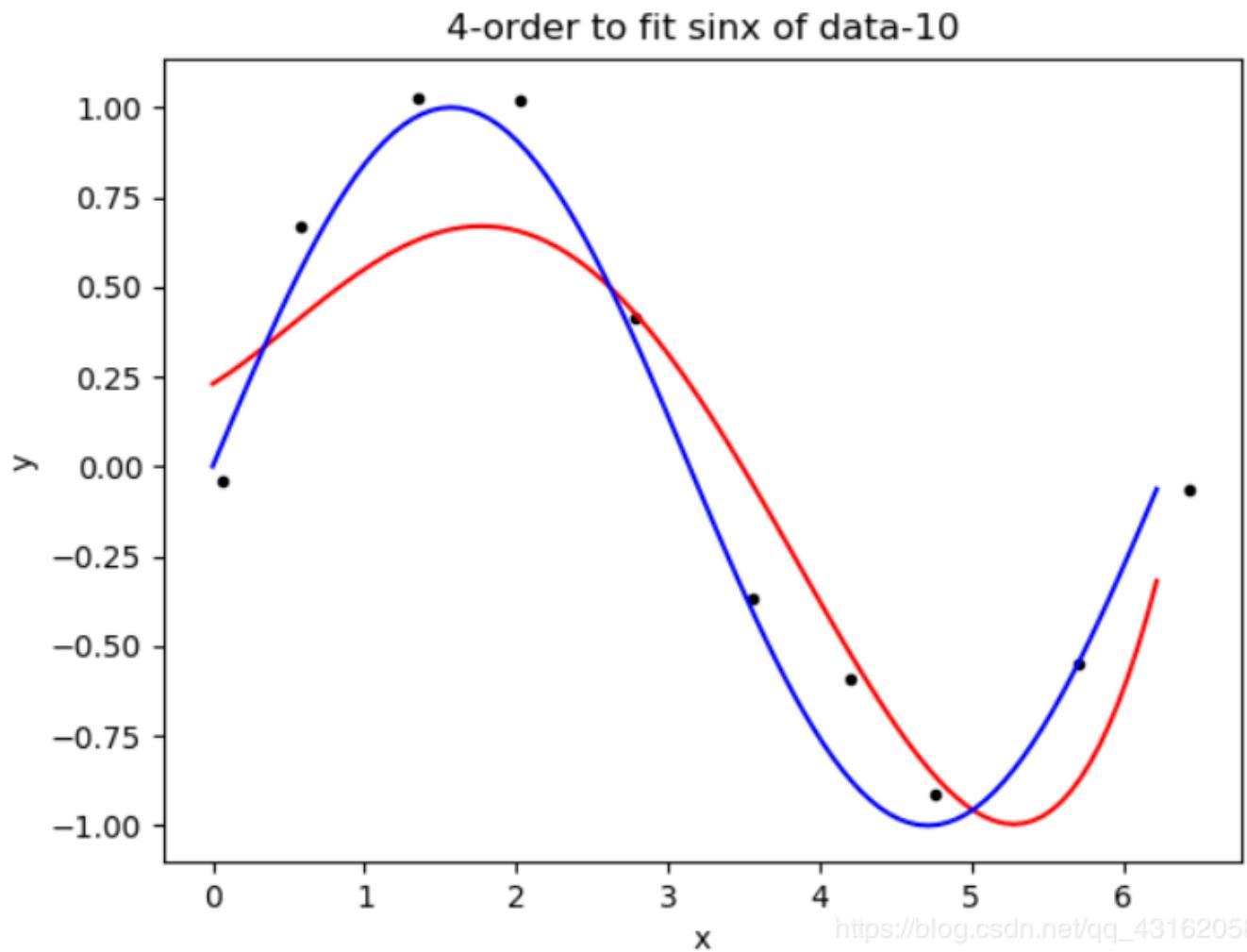




https://blog.csdn.net/qq_43162058

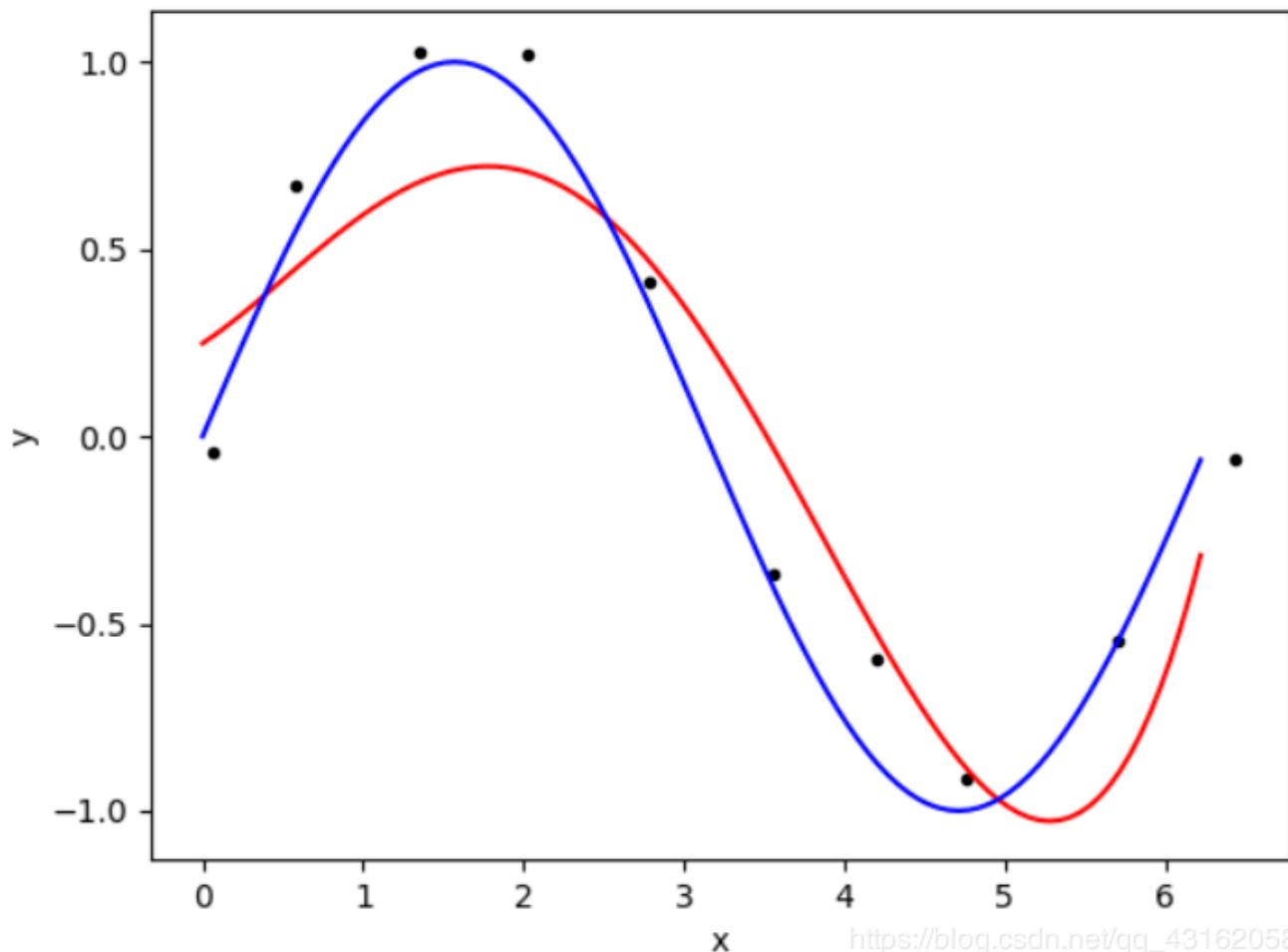
4.3.2 关于步长的选取 (以5阶, 数据量为10为例)

步长为 $1e-6$



步长为 $2e-6$

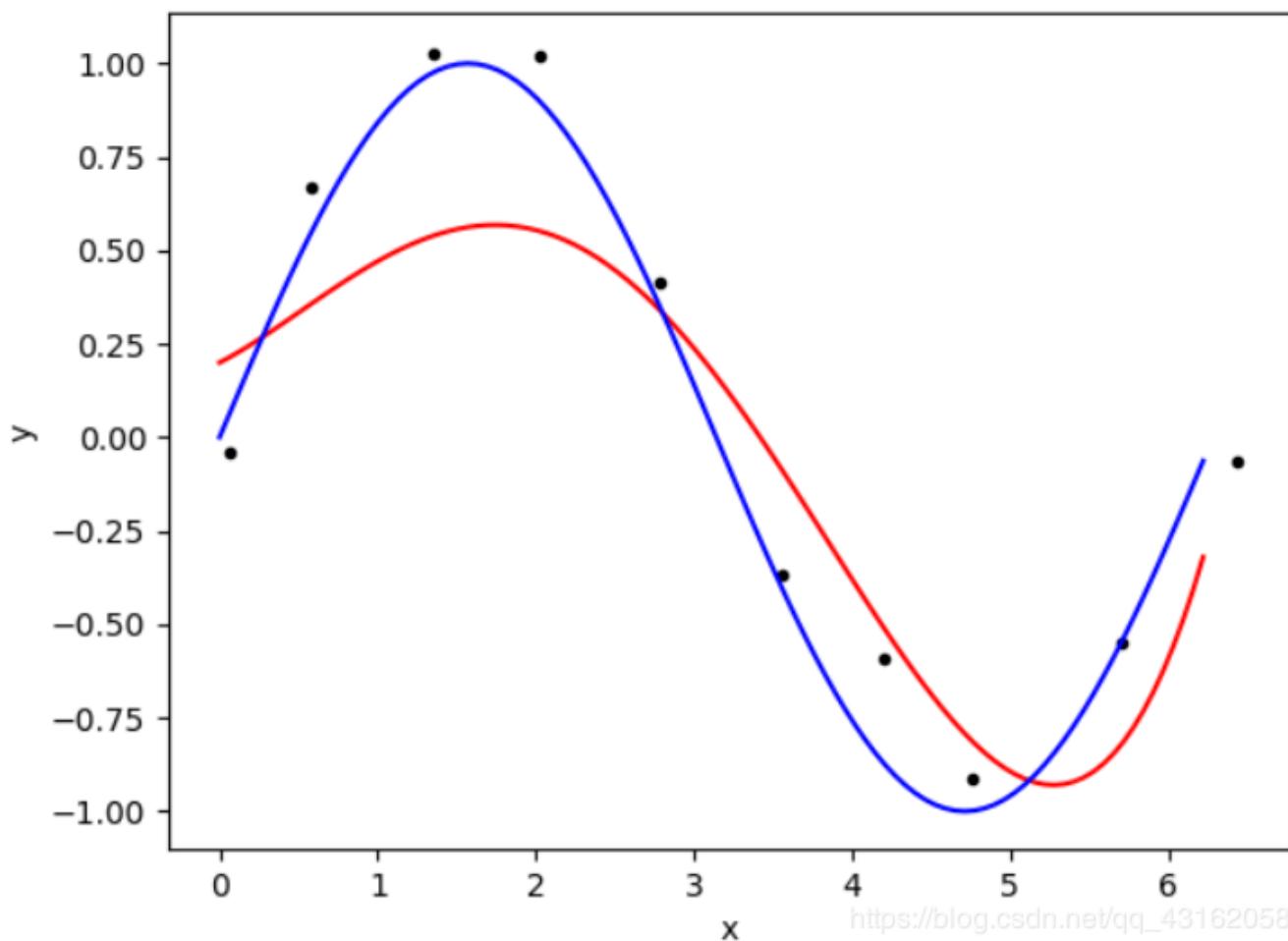
4-order to fit $\sin x$ of data-10



https://blog.csdn.net/qq_43162058

步长为 $5e-7$ 时，运行时间比较长，拟合效果也不太好

4-order to fit $\sin x$ of data-10

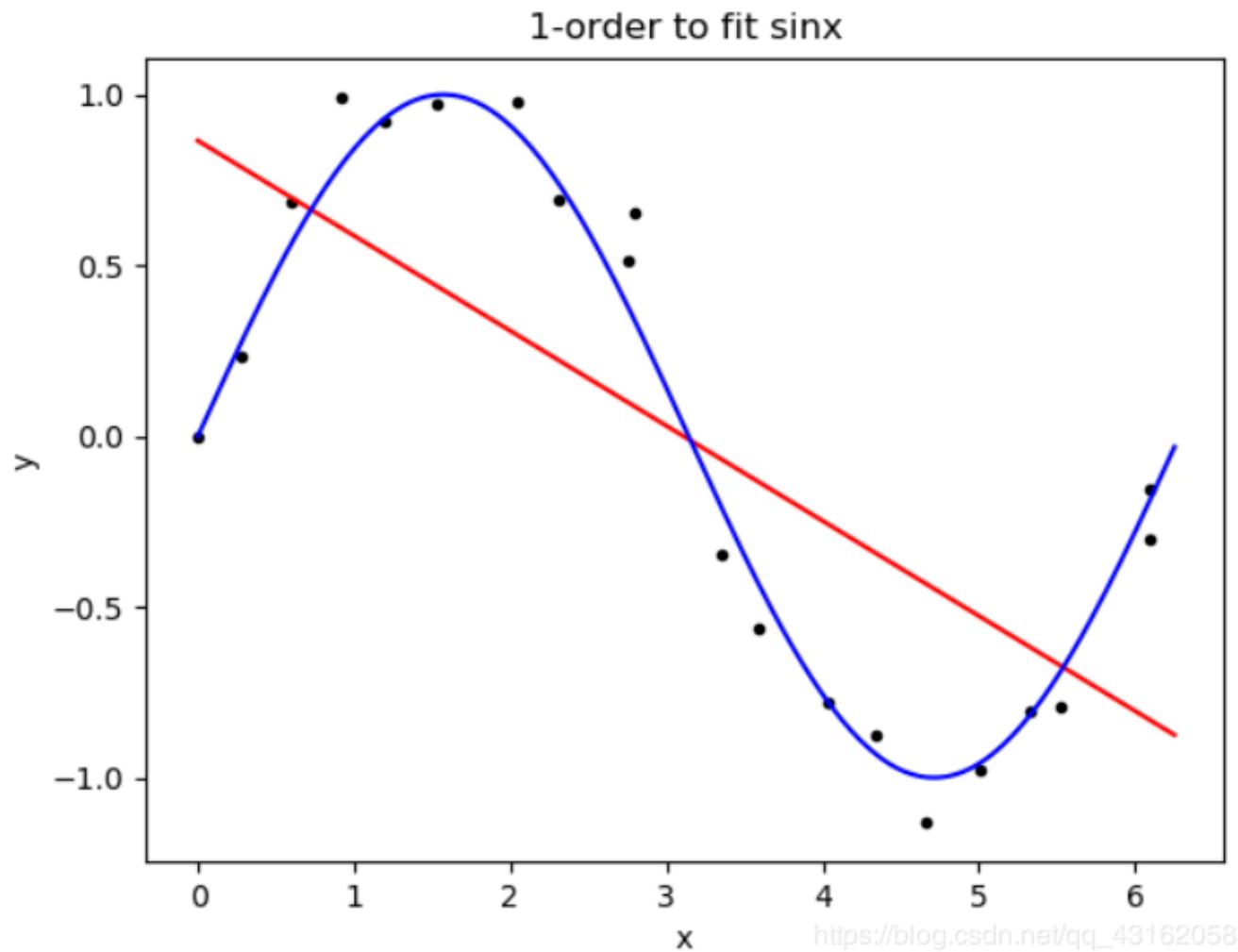


https://blog.csdn.net/qq_43162058

从这几组数据来看， $1e-6$ 拟合得最好，当增大步长时，矩阵运算过程中会发生溢出，当继续减小步长时，由于步长太小对精度非常敏感，但是如果继续下调精度收敛速度会很慢。

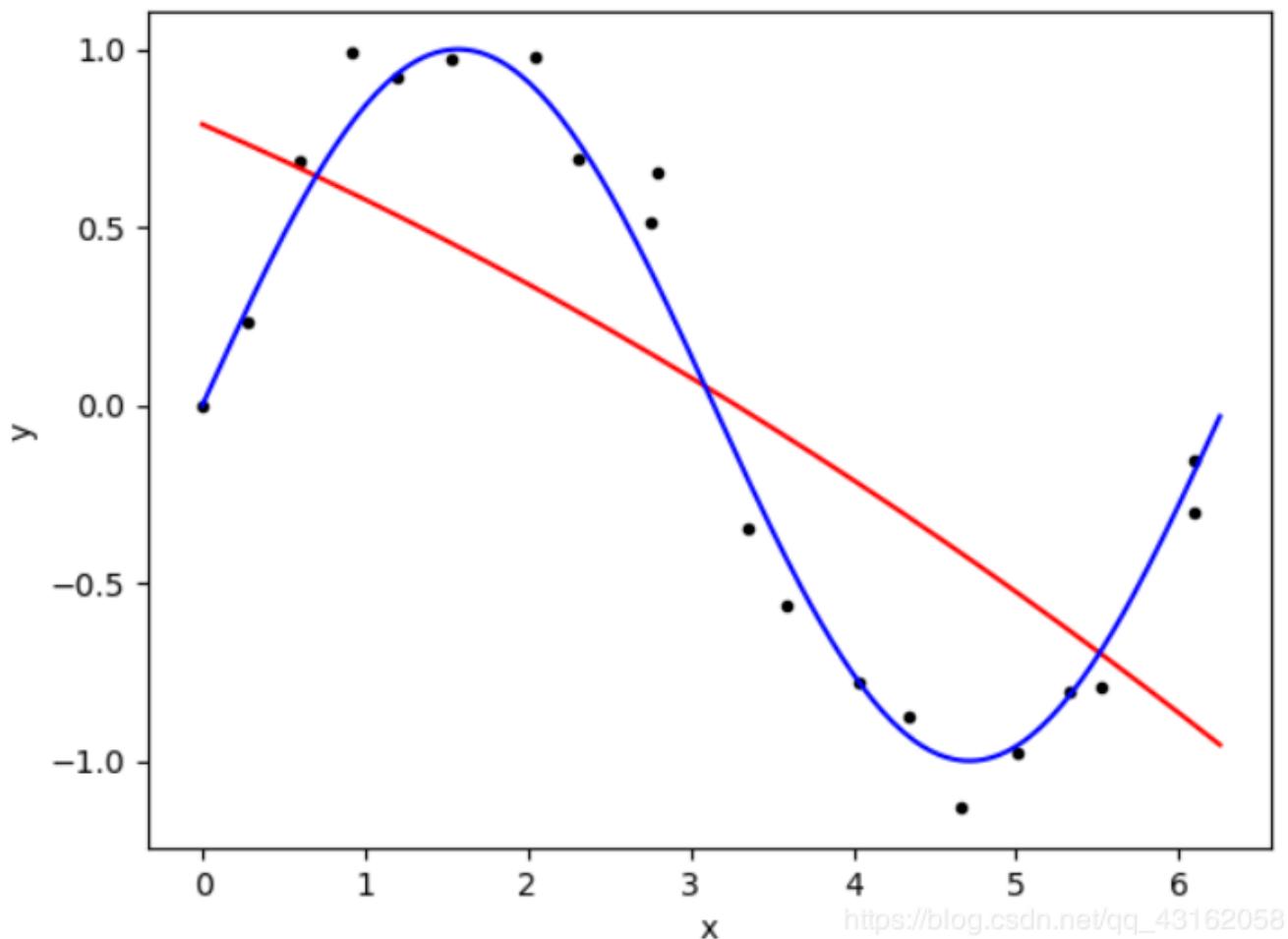
数据量为20时不同阶数共轭梯度法的损失如下图所示，从三阶开始损失以及比较小了

一阶



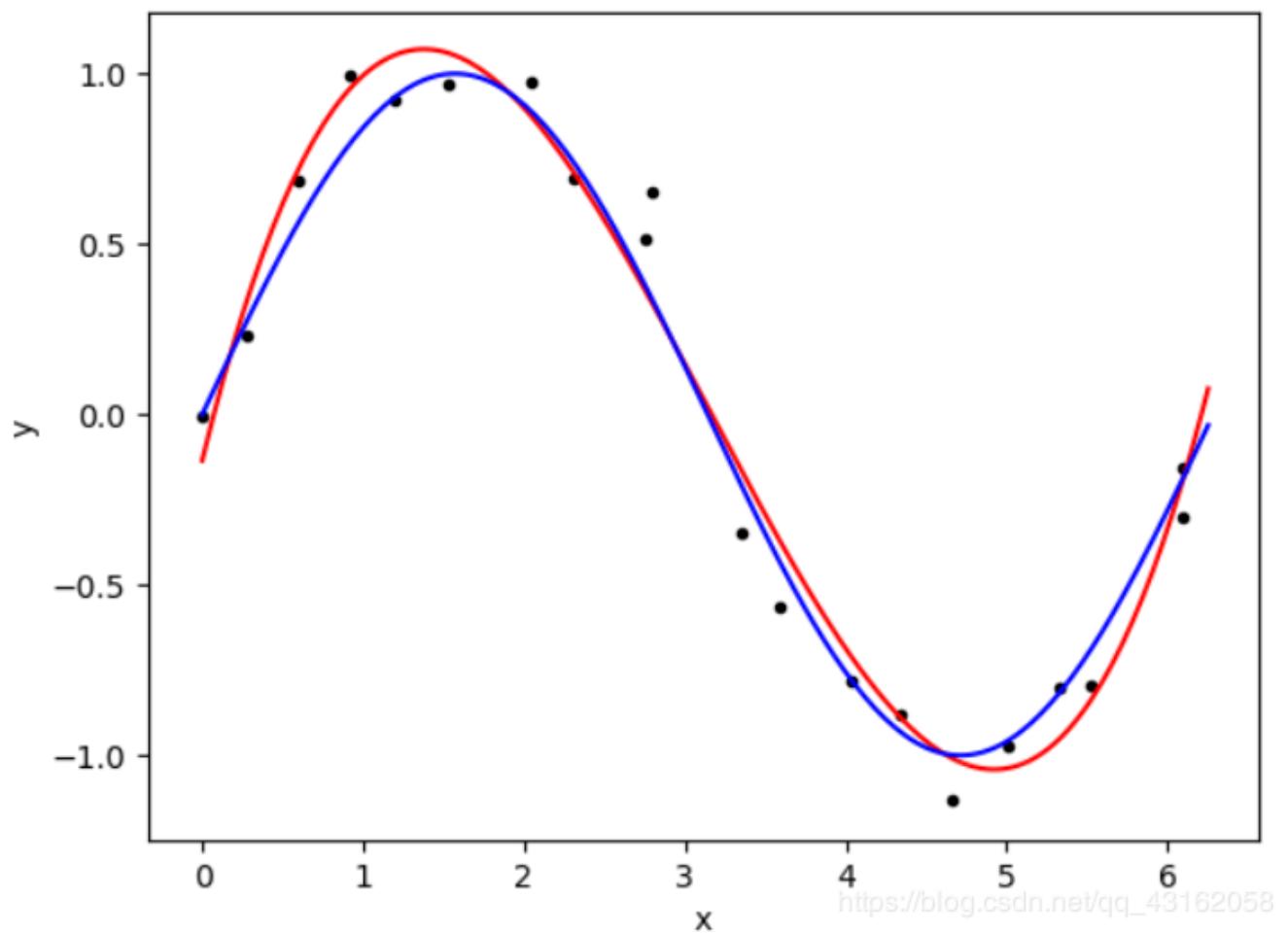
二阶

2-order to fit $\sin x$

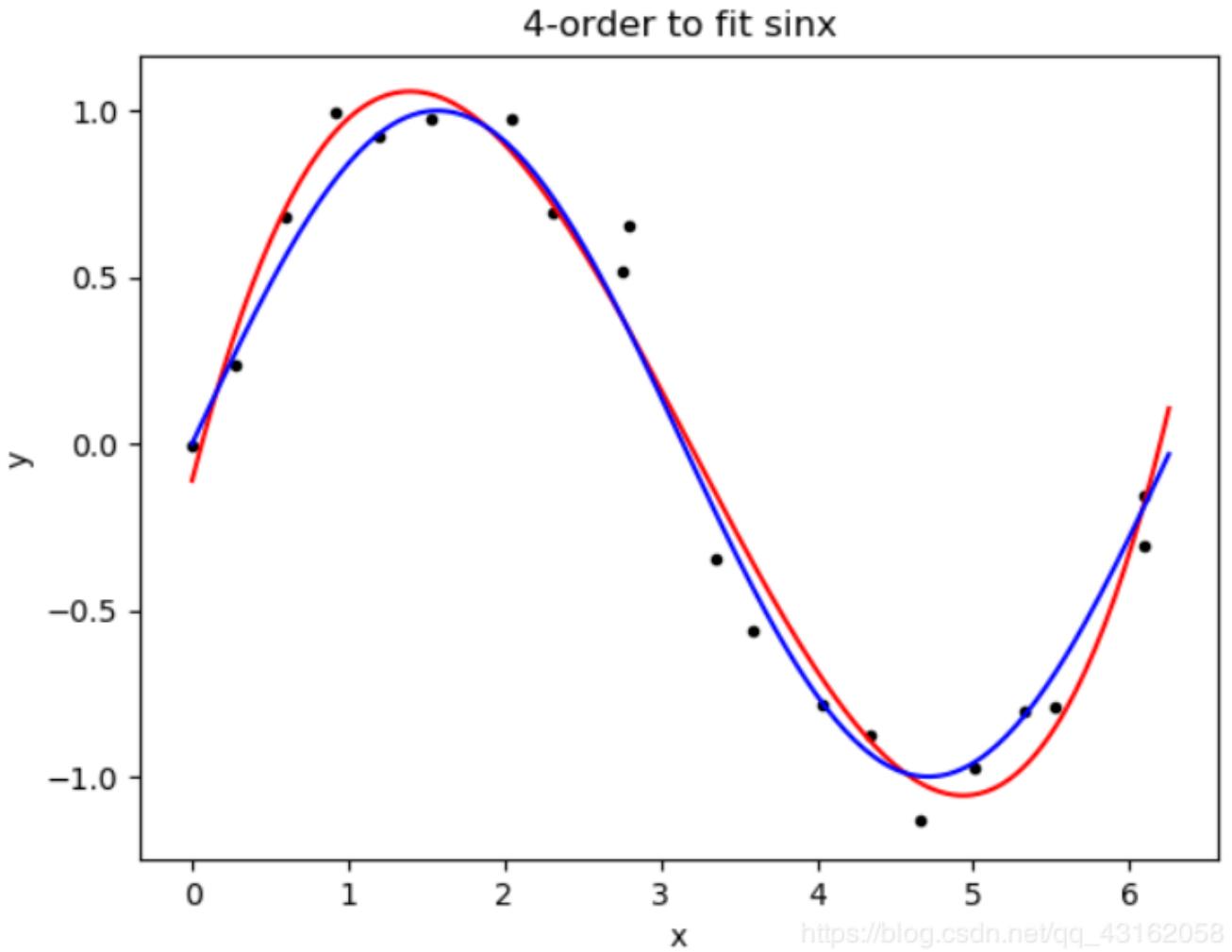


三阶

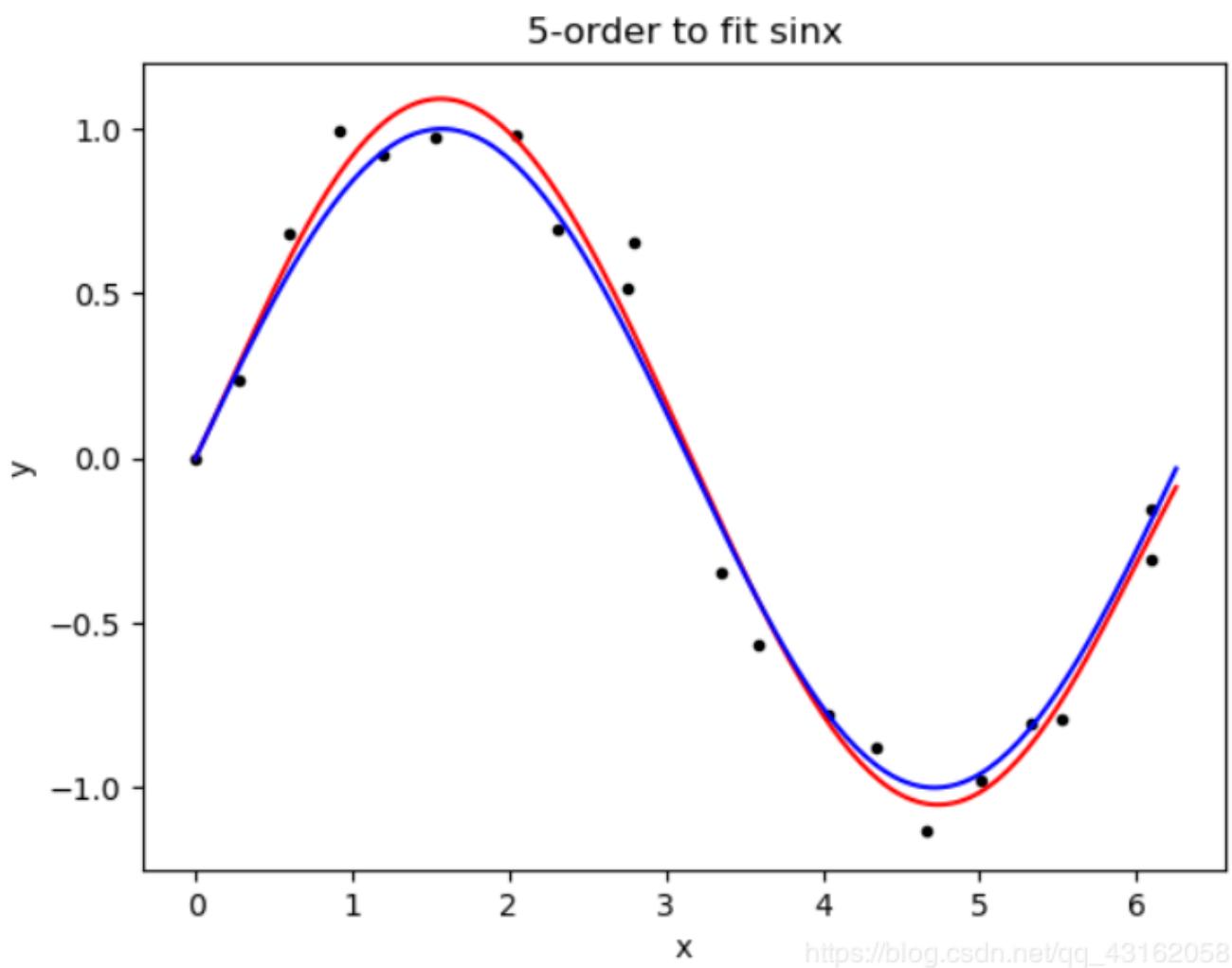
3-order to fit $\sin x$



四阶

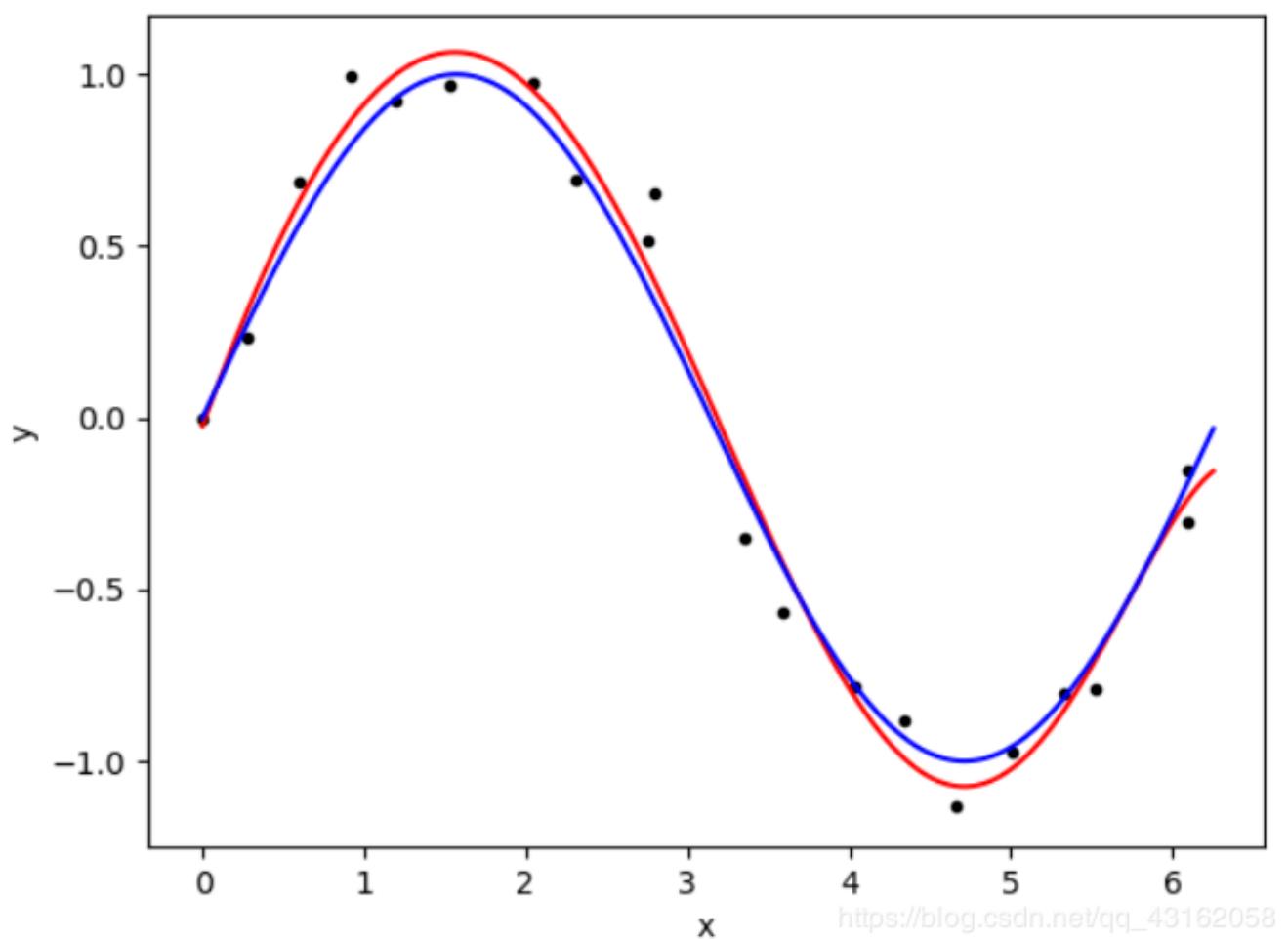


五阶



六阶

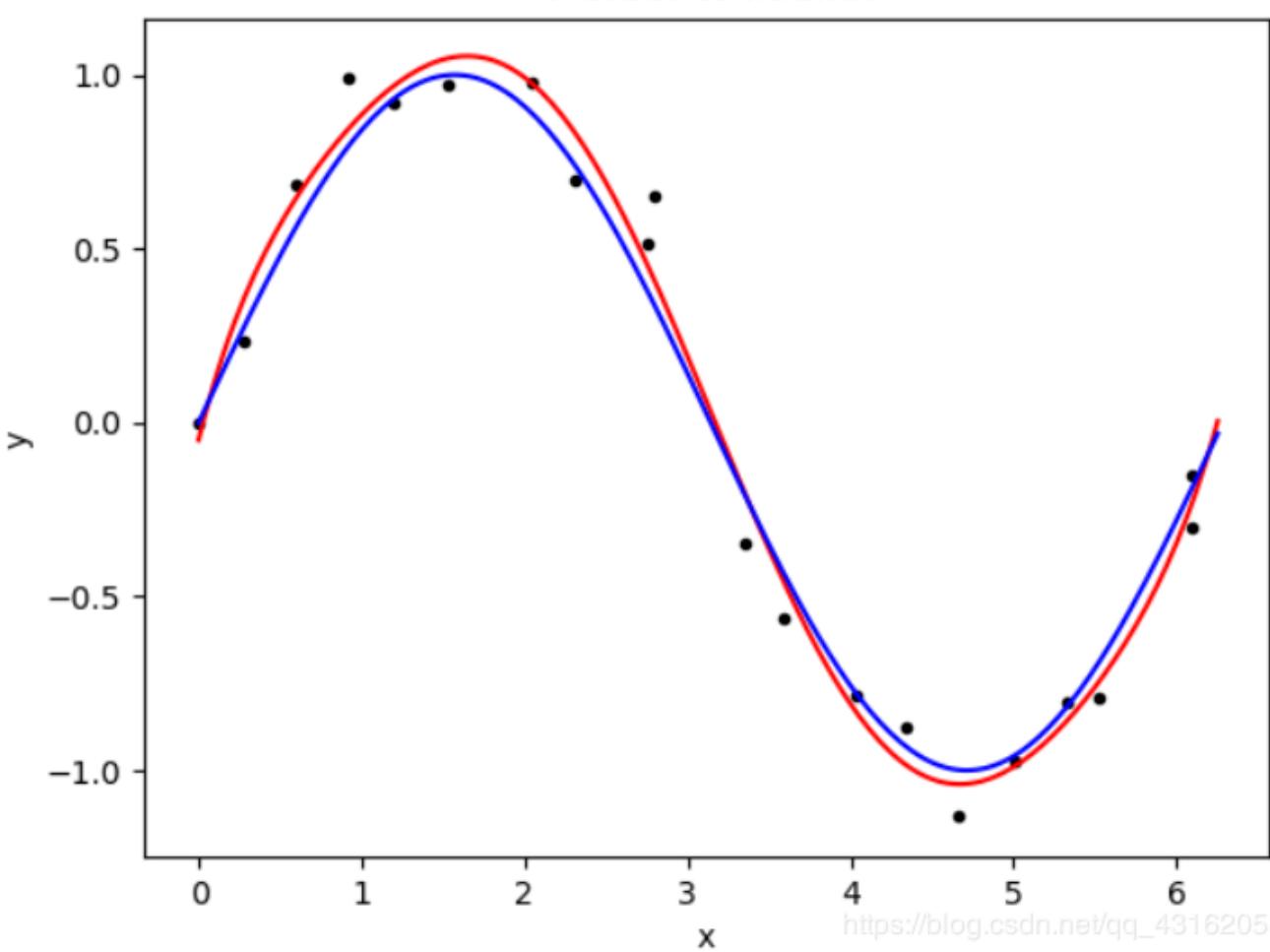
6-order to fit sinx



https://blog.csdn.net/qq_43162058

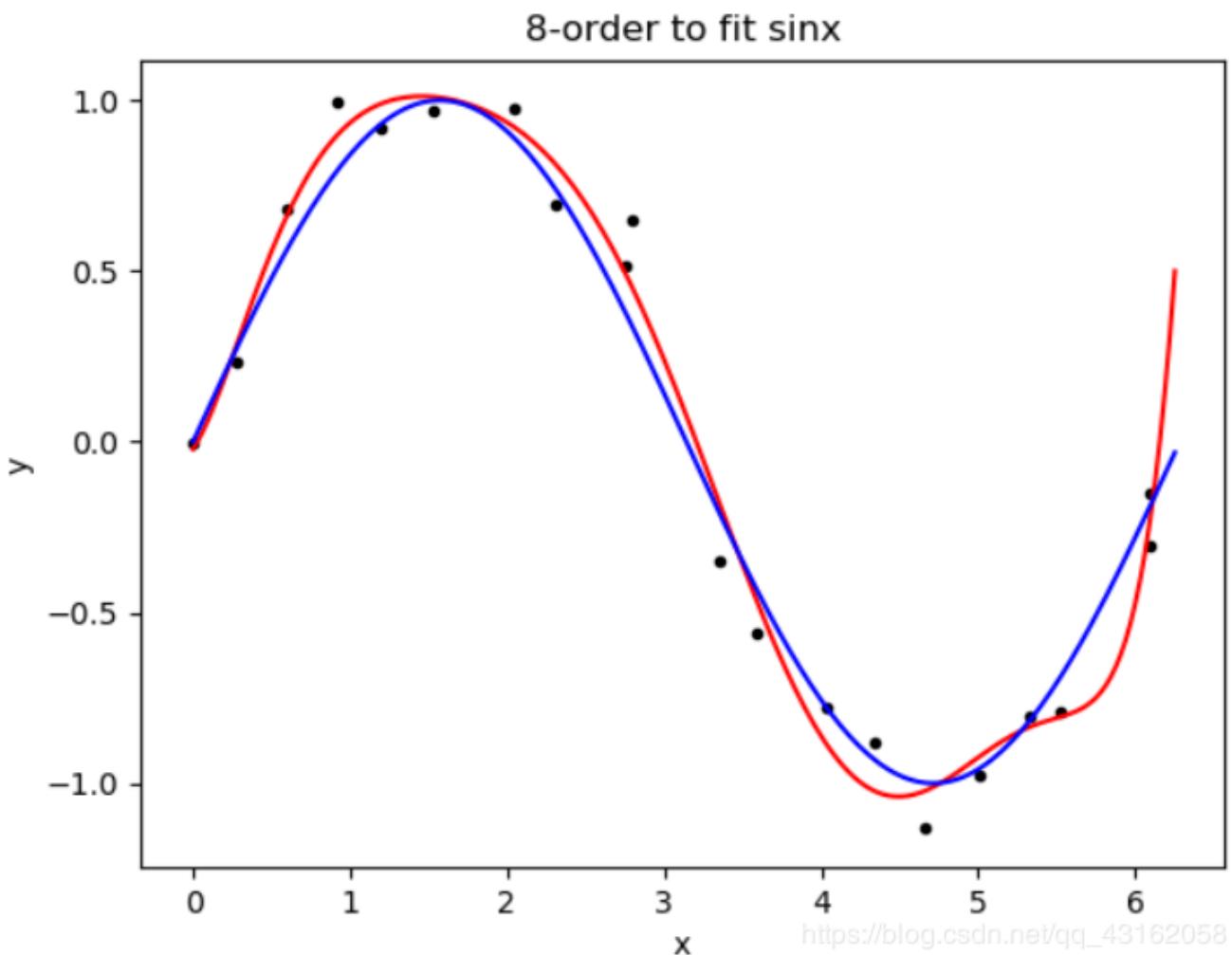
七阶

7-order to fit sinx

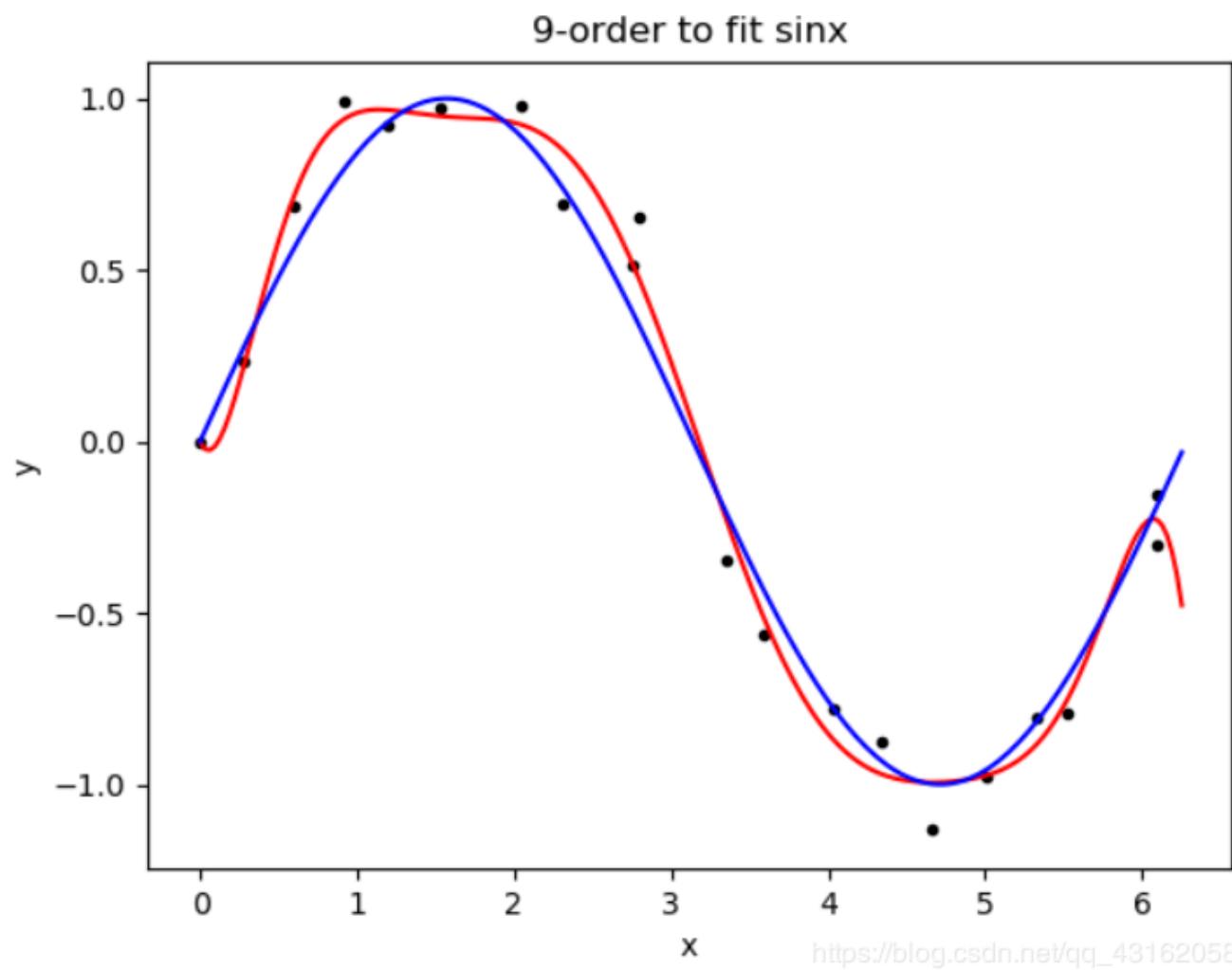


https://blog.csdn.net/qq_43162058

八阶 (开始出现明显的过拟合)

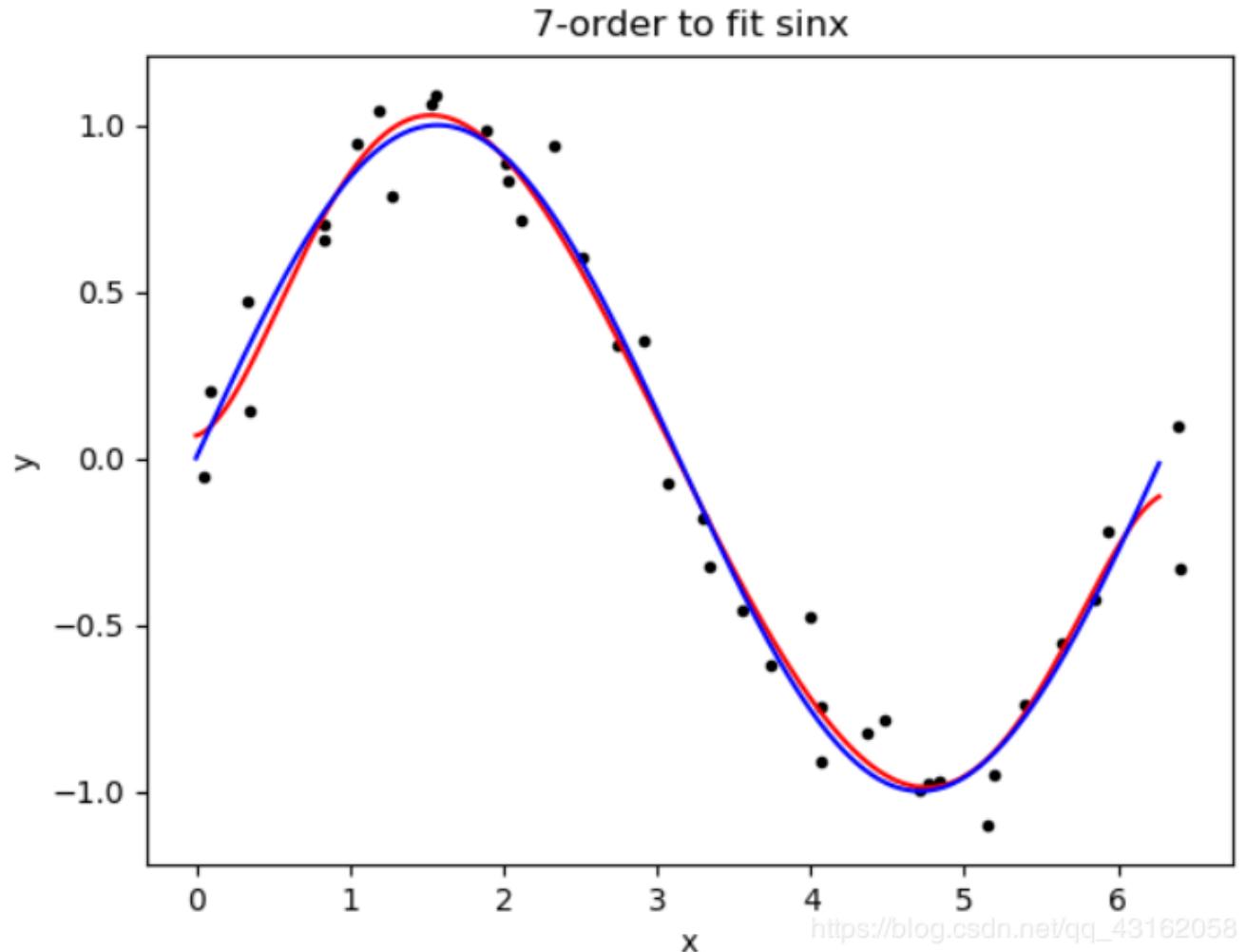


九阶



由此可以得出结论，当阶数为1, 2时，损失特别大，可能原因是矩阵运算过程中产生了一些误差，并且阶数太小精度不够；3-9阶的损失都比较小，但是阶数为8, 9时出现了明显的过拟合。随后我开始增大数据规

模，当数据量增大到为40，7阶也出现了过拟合现象



当数据量大于50时，7-9阶基本没有出现过拟合现象。

综上，当数据量为10-100时，使用5, 6阶拟合sinx损失小，且不会出现过拟合

五、结论

- 通常情况下解析解能解决大多数拟合问题，但是 $(X^T X)$ 不一定可逆。
- 当数据量和阶数相当时，直接使用最小二乘法容易过拟合，可以通过加入正则项解决。关于正则因子的选择，它与数据量和阶数都相关。通常情况下，数据量越大，正则因子越大；阶数越大，正则因子越大。正则+解析式方法是最简便同时又不容易过拟合的方法，但是正则因子的选择很困难。
- 为了避免谈论 $(X^T X)$ 是否可导的问题，我们可以采取梯度下降法或者共轭梯度法来替代。梯度下降法很直接，但是下降速度比较慢，而且关于步长的选择及其严苛。共轭梯度法下降速度快，且能得到几乎和解析式重合的拟合曲线，在应对高维数据时具有明显优势。但是共轭梯度法理解起来不够直观，想要改进也比较困难。
- 本实验讨论的损失函数是凸函数，对于非凸函数问题不适用（主要体现在梯度下降法需要给出跳出局部最优的条件）。
- 多项式拟合函数能力很强，但是阶数不宜太高，否则在运算过程中容易溢出（当样例中的x比较大，对x乘方很容易溢出）

六、参考文献

机器学习【周志华】
统计学习方法【李航】
数值分析【李庆扬，王能超，易大义】

七、附录：源代码（带注释）

```
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
np.seterr(divide="ignore", invalid='ignore')

#求解矩阵X
def calX(dataAmount, n, dataMatrix):
    X = ones((dataAmount, n+1)) #初始化
    for i in range(dataAmount):
        for j in range(n):
            X[i][j] = dataMatrix[i][0]**(j+1)
        X[i][n] = 1
    return X

def cal(x, n, w):
    rel = 0
    for j in range(n):
        rel += x**(j+1)*w[j]
    rel += w[n]
    return rel

#计算loss(不带惩罚项)
def calLoss(w, y, X, dataAmount):
    rel = y - np.dot(X, w)
    return np.dot(rel.T, rel)/(2*dataAmount)

#不带正则项的解析解
def analysis(dataAmount, n):
    doc = str(dataAmount) + ".txt"
    dataMatrix = np.loadtxt(doc, dtype=float)
    cols = dataMatrix.shape[-1]
    y = dataMatrix[:, cols-1:cols]
    X = calX(dataAmount, n, dataMatrix)
    w = np.dot(np.linalg.inv(np.dot(X.T, X)), X.T), y)
    loss = calLoss(w, y, X, dataAmount)

    dx = []
    dx1 = []
    d = 2 * np.pi / (dataAmount * 10)
    cnt = 0
    for i in range(dataAmount):
        dx1.append(dataMatrix[i][0])
    for i in range(dataAmount * 10):
        dx.append(cnt)
        cnt += d
    dy = []
    dy1 = []
    for i in range(dataAmount):
        dy1.append(dataMatrix[i][1])
    for i in range(len(dx)):
        dy.append(cal(dx[i], n, w))
    plt.plot(dx, dy, 'r')
```

```

plt.plot(dx, sin(dx), 'b')
plt.scatter(dx1, dy1, c="#000000", marker='.')
plt.xlabel("x")
plt.ylabel("y")
plt.title(str(n) + "-order to fit sinx of data-" + str(dataAmount))
plt.show()

return loss[0][0]
#analysis(40,7)

#带正则项的解析解
def analysis_regex(dataAmount, n, lamda):
    doc = str(dataAmount) + ".txt"
    dataMatrix = np.loadtxt(doc, dtype=float)
    cols = dataMatrix.shape[-1]
    y = dataMatrix[:, cols-1:cols]
    X = calX(dataAmount, n, dataMatrix)
    I = eye(n+1)
    w = np.dot(np.dot(np.linalg.inv(np.dot(X.T, X) + lamda*dataAmount*I), X.T), y)
    loss1 = calLoss(w, y, X, dataAmount) #对比不同lamda的损失
    #loss = calLoss(w, y, X, dataAmount) + lamda*np.dot(w.T, w)/2
    """
    dx = []
    dx1 = []
    d = 2 * np.pi / (dataAmount * 10)
    cnt = 0
    for i in range(dataAmount):
        dx1.append(dataMatrix[i][0])
    for i in range(dataAmount * 10):
        dx.append(cnt)
        cnt += d
    dy = []
    dy1 = []
    for i in range(dataAmount):
        dy1.append(dataMatrix[i][1])
    for i in range(len(dx)):
        dy.append(cal(dx[i], n, w))
    plt.plot(dx, dy, 'r')
    plt.plot(dx, sin(dx), 'b')
    plt.scatter(dx1, dy1, c="#000000", marker='.')
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(str(n) + "-order to fit sinx of data-" + str(dataAmount))
    plt.show()
    """

    return loss1[0][0]
#analysis_regex(20,8,0.001)

#求梯度
def gradient(X, w, y, dataAmount):
    return np.dot(X.T, np.dot(X, w)-y)/dataAmount

#梯度下降法
def de_gradient(dataAmount, n, lamda):
    doc = str(dataAmount) + ".txt"

```

```

dataMatrix = np.loadtxt(doc, dtype=float)
cols = dataMatrix.shape[-1]
y = dataMatrix[:, cols-1:cols]
X = calX(dataAmount, n, dataMatrix)
w = 0.1*ones((n+1, 1))
on = 0.000001
g0 = gradient(X, w, y, dataAmount)
#cnt = 5000000
while 1:
    #loss = calLoss(w, y, X, dataAmount)
    w += (-lamda) * g0*1000
    print(g0)
    #loss1 = calLoss(w, y, X, dataAmount)
    g = gradient(X, w, y, dataAmount)
    #cnt -= 1
    if np.linalg.norm(g-g0) < on:
        break
    g0 = g
dx = []
dx1 = []
d = 2 * np.pi / (dataAmount * 10)
cnt = 0
for i in range(dataAmount):
    dx1.append(dataMatrix[i][0])
for i in range(dataAmount * 10):
    dx.append(cnt)
    cnt += d
dy = []
dy1 = []
for i in range(dataAmount):
    dy1.append(dataMatrix[i][1])
for i in range(len(dx)):
    dy.append(cal(dx[i], n, w))
plt.plot(dx, dy, 'r')
plt.plot(dx, sin(dx), 'b')
plt.scatter(dx1, dy1, c="#000000", marker='.')
plt.xlabel("x")
plt.ylabel("y")
plt.title(str(n) + "-order to fit sinx of data-" + str(dataAmount))
plt.show()

loss = calLoss(w, y, X, dataAmount)
return loss[0][0]

de_gradient(10,4,0.000000001)

#共轭梯度法
def cj_gradient(dataAmount,n):
    doc = str(dataAmount) + ".txt"
    dataMatrix = np.loadtxt(doc, dtype=float)
    cols = dataMatrix.shape[-1]
    y = dataMatrix[:, cols - 1:cols]
    X = calX(dataAmount, n, dataMatrix)
    A = np.dot(X.T, X)
    w = zeros((n+1, 1))

```

```

b = np.dot(X.T, y)
r = b - np.dot(A, w)
p = r
while 1:
    if np.dot(r.T, r) == 0 or np.dot(np.dot(p.T, A), p) == 0:
        break
    a = np.dot(r.T, r) / np.dot(np.dot(p.T, A), p)
    w += a*p
    r1 = r - np.dot(a*A, p)
    beta = np.dot(r1.T, r1)/np.dot(r.T, r)
    p = r1 + beta*p
    r = r1
loss = calLoss(w, y, X, dataAmount)

dx = []
dx1 = []
d = 2*np.pi/(dataAmount*10)
cnt = 0
for i in range(dataAmount):
    dx1.append(dataMatrix[i][0])
for i in range(dataAmount*10):
    dx.append(cnt)
    cnt += d
dy = []
dy1 = []
for i in range(dataAmount):
    dy1.append(dataMatrix[i][1])
for i in range(len(dx)):
    dy.append(cal(dx[i], n, w))

plt.plot(dx, dy, 'r')
plt.plot(dx, sin(dx), 'b')
plt.scatter(dx1, dy1, c="#000000", marker='.')
plt.xlabel("x")
plt.ylabel("y")
plt.title(str(n)+"-order to fit sinx of data-"+str(dataAmount))
plt.show()

return loss[0][0]

#cj_gradient(40,7)

#比较数据量为20时不同阶不带正则项解析解的损失
def main1(max_n):
    dataAmount = 20
    x = []
    for i in range(1,max_n):
        x.append(i)
    y = []
    for i in range(1,max_n):
        y.append(analysus(dataAmount, x[i-1]))
    plt.plot(x, y, 'b')
    plt.title("different order's loss")
    plt.xlabel("order")

```

```
plt.ylabel("loss")
plt.show()
```