

CLAWER Lecture3 编程作业 U202111505

1, 分别编写一个用广义逆和梯度下降法来求最小误差平方和最佳解的算法 分别见代码 LR 类中的 `generalized_inverse` 函数与 `gradient_descent_train` 函数

2, (a) 产生两个都具有 200 个二维向量的数据集 \mathbf{X}_1 和 \mathbf{X}_2 。数据集 \mathbf{X}_1 的样本来自均值向量 $\mathbf{m}_1 = [-5, 0]^T$ 、协方差矩阵 $\mathbf{s}_1 = \mathbf{I}$ 的正态分布, 属于 “+1” 类, 数据集 \mathbf{X}_2 的样本来自均值向量 $\mathbf{m}_2 = [0, 5]^T$ 、协方差矩阵 $\mathbf{s}_2 = \mathbf{I}$ 的正态分布, 属于 “-1” 类, 其中 \mathbf{I} 是一个 2*2 的单位矩阵。产生的数据中 80% 用于训练, 20% 用于测试。

与上一次实验一致见 `data_generator.py`

(b) 在上述数据集上分别第 1 题的两个算法, 利用产生的训练样本集得到分类面, 算法中用到的各类超参数自定。

(c) 分别在训练集和测试集上统计分类正确率。

(d) 画出数据集和分类面。

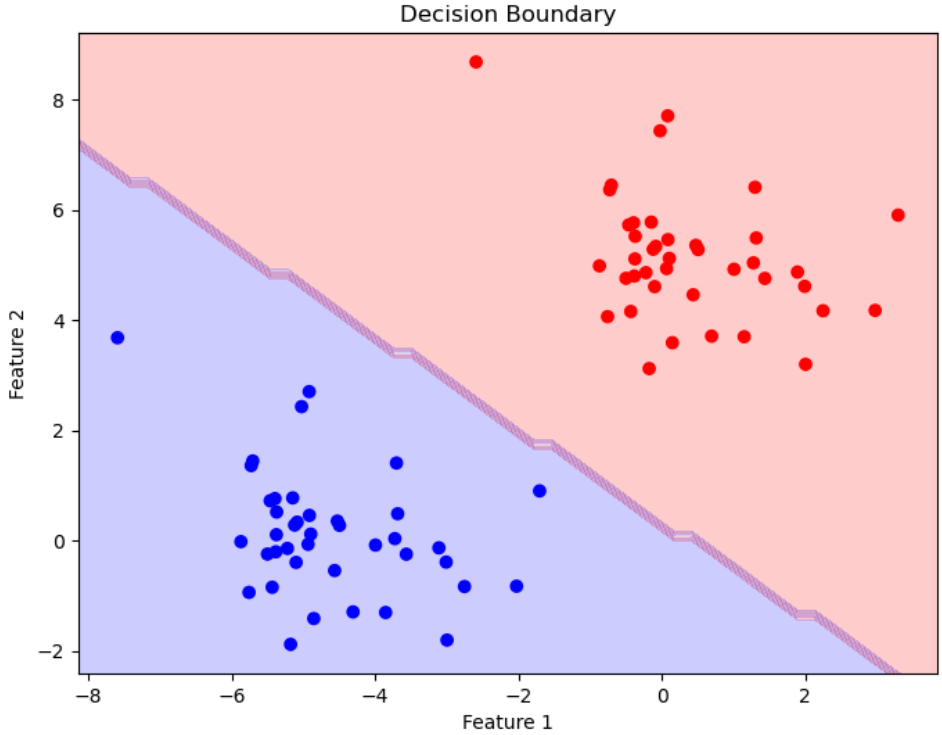
(e) 画出损失函数随 epoch 增加的变化曲线。

一、广义逆方法

该方法不需要训练迭代, 无损失函数曲线

结果如下:

```
最终权重 w (梯度下降法): [-0.16961808 -0.20139995]
最终偏置 b (梯度下降法): 0.06969891440039389
Accuracy: 1.0
F1 Score: 1.0
Precision: 1.0
Recall: 1.0
Confusion Matrix: {'True Positives': 40, 'False Positives': 0, 'True Negatives': 40, 'False Negatives': 0}
```

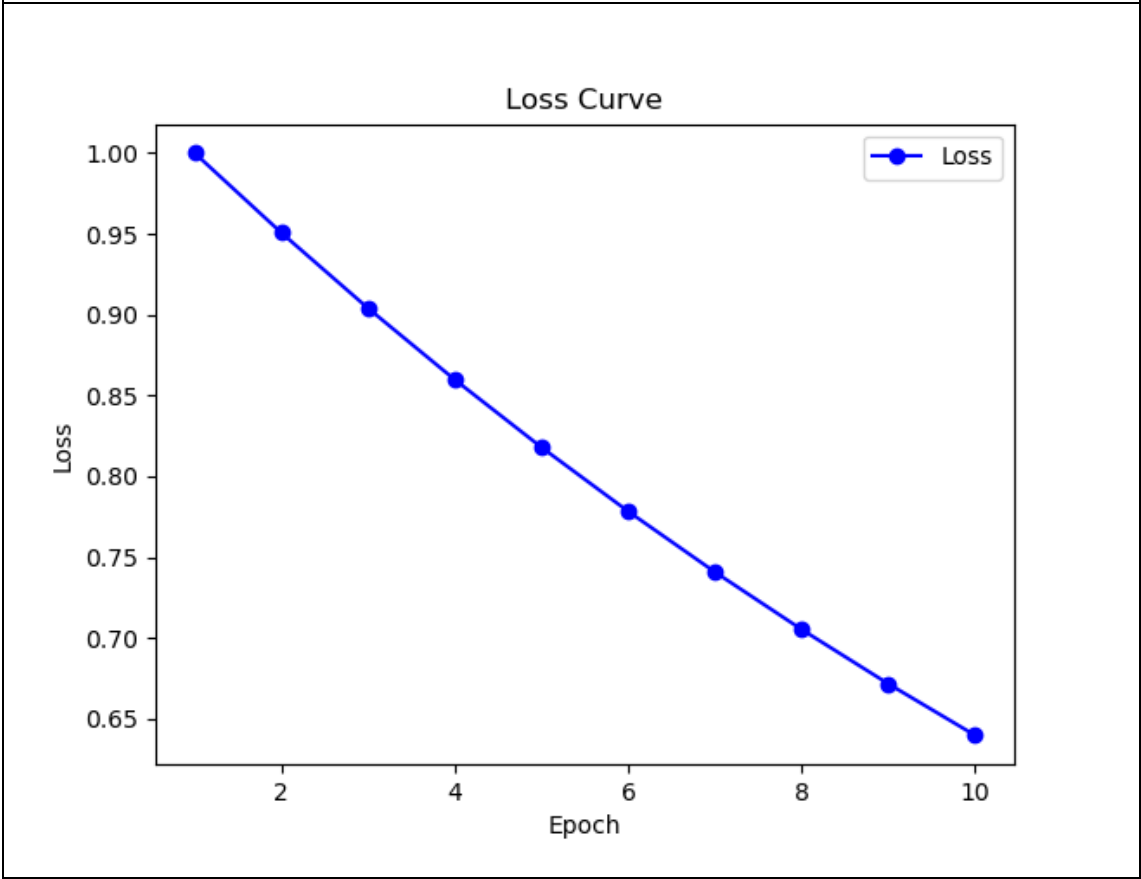
求解方法	广义逆
分类面	$0.0697 - 0.1696 * x + -0.20134 * y = 0$
测试集准确率	ACC=1
训练集准确率	ACC=1
损失函数曲线	广义逆法无
可视化结果	
	

二、梯度下降方法

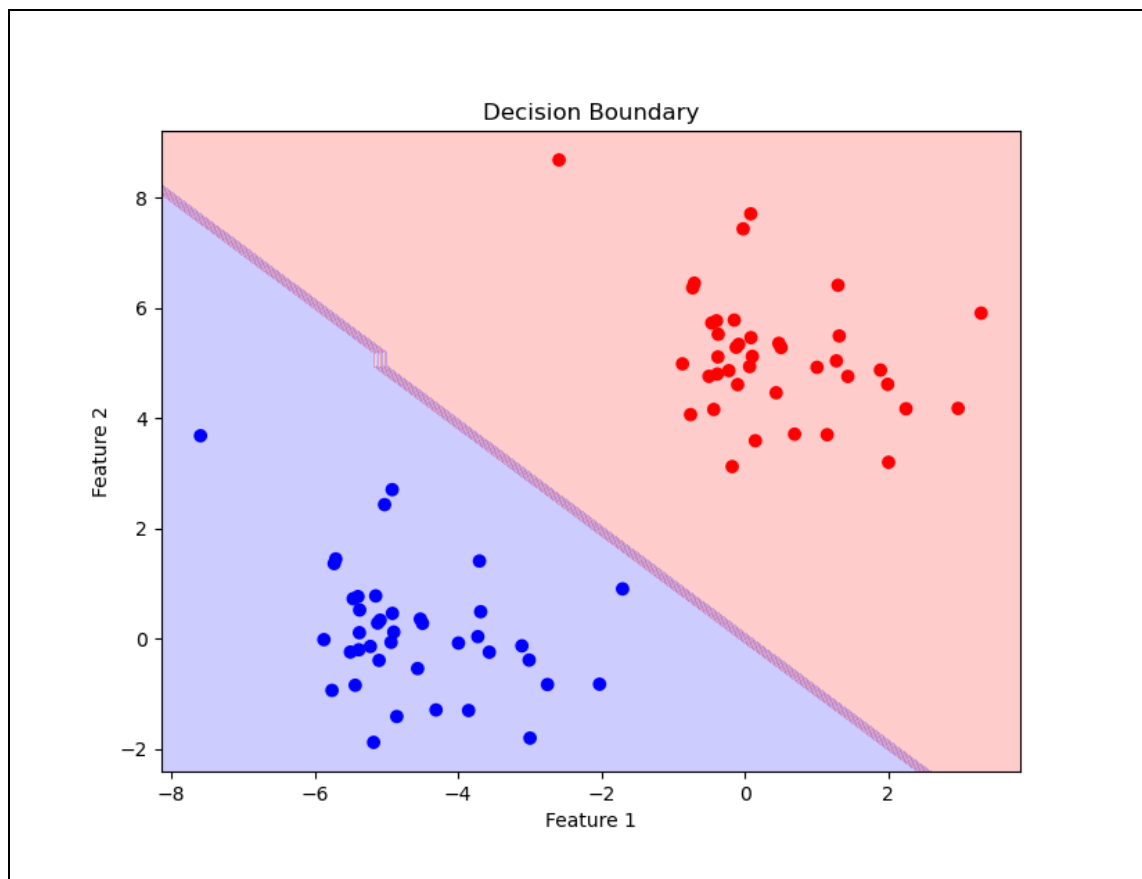
```
最终权重 w (梯度下降法): [-0.04426083 -0.0444356 ]
最终偏置 b (梯度下降法): -2.061622055012606e-05
Accuracy: 1.0
F1 Score: 1.0
Precision: 1.0
Recall: 1.0
Confusion Matrix: {'True Positives': 40, 'False Positives': 0, 'True Negatives': 40, 'False Negatives': 0}
```

求解方法	梯度下降
超参数	Lr=1e-3,max_epochs=10,bs=320(full)
分类面	$-2.0616-0.0443*x-0.0444*y=0$
测试集准确率	ACC=1
训练集准确率	ACC=1

损失函数曲线



可视化结果



3, 重复第 2 题的内容, 但数据集 \mathcal{X}_1 和数据集 \mathcal{X}_2 的均值向量分别改为 $m_1 = [1, 0]^T$ 和 $m_2 = [0, 1]^T$, 其他不变。

一、广义逆方法

该方法不需要训练迭代, 无损失函数曲线

结果如下:

测试集

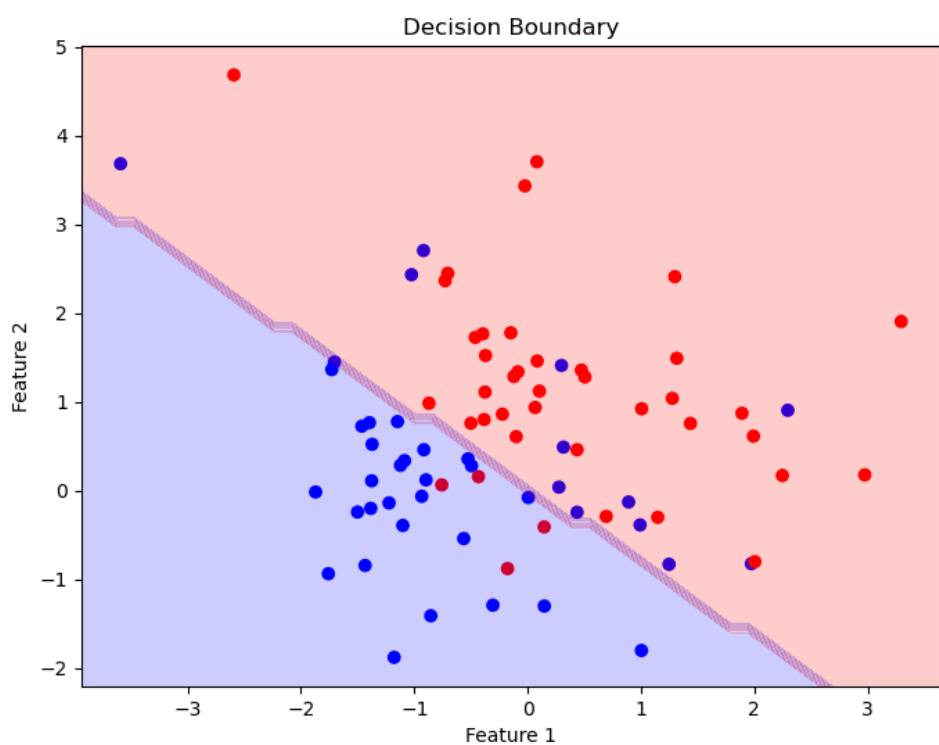
```
C:\ProgramData\Anaconda5\envs\python35\python.exe C:\Users\Think\Desktop\模式识别与机器学习\实验一：线性回归.py
最终权重 w (梯度下降法): [-0.30964433 -0.36766336]
最终偏置 b (梯度下降法): 0.011199981024385169
Accuracy: 0.8
F1 Score: 0.7777777777777777
Precision: 0.875
Recall: 0.7
Confusion Matrix: {'True Positives': 28, 'False Positives': 4, 'True Negatives': 36, 'False Negatives': 12}
```

训练集

```
最终权重 w (梯度下降法): [-0.30964433 -0.36766336]
最终偏置 b (梯度下降法): 0.011199981024385169
Accuracy: 0.76875
F1 Score: 0.76875
Precision: 0.76875
Recall: 0.76875
Confusion Matrix: {'True Positives': 123, 'False Positives': 37, 'True Negatives': 123, 'False Negatives': 37}
```

求解方法	广义逆
分类面	$0.0112-0.3096*x-0.3677*y=0$
测试集准确率	ACC=0.769
训练集准确率	ACC=0.8
损失函数曲线	广义逆法无

可视化结果



二、梯度下降方法

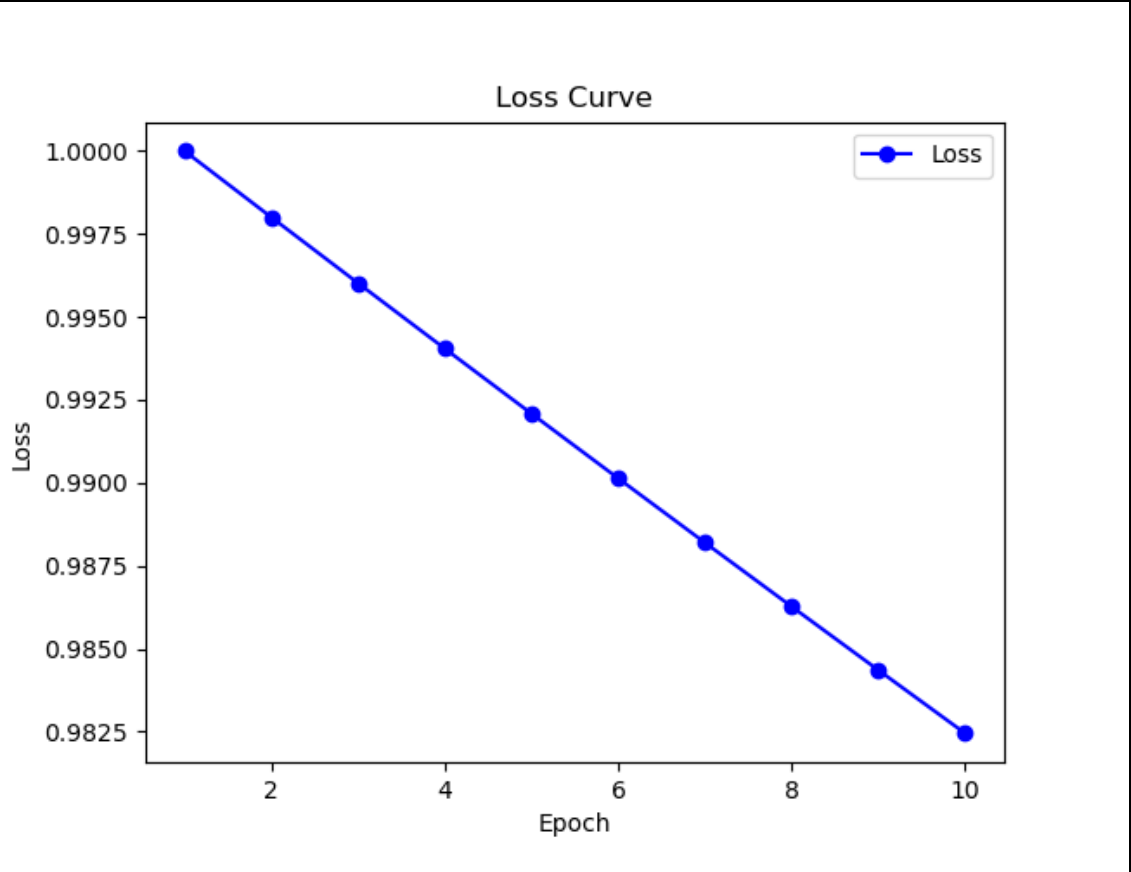
训练集

```
C:\ProgramData\Anaconda3\envs\python39\python.exe C:\Users\Think\Desktop\模式识别与机器学习\实验二：线性回归算法\mai
最终权重 w (梯度下降法): [-0.00985726 -0.00987751]
最终偏置 b (梯度下降法): -4.860897808623222e-06
Accuracy: 0.759375
F1 Score: 0.7616099071207431
Precision: 0.754601226993865
Recall: 0.76875
Confusion Matrix: {'True Positives': 123, 'False Positives': 40, 'True Negatives': 120, 'False Negatives': 37}
```

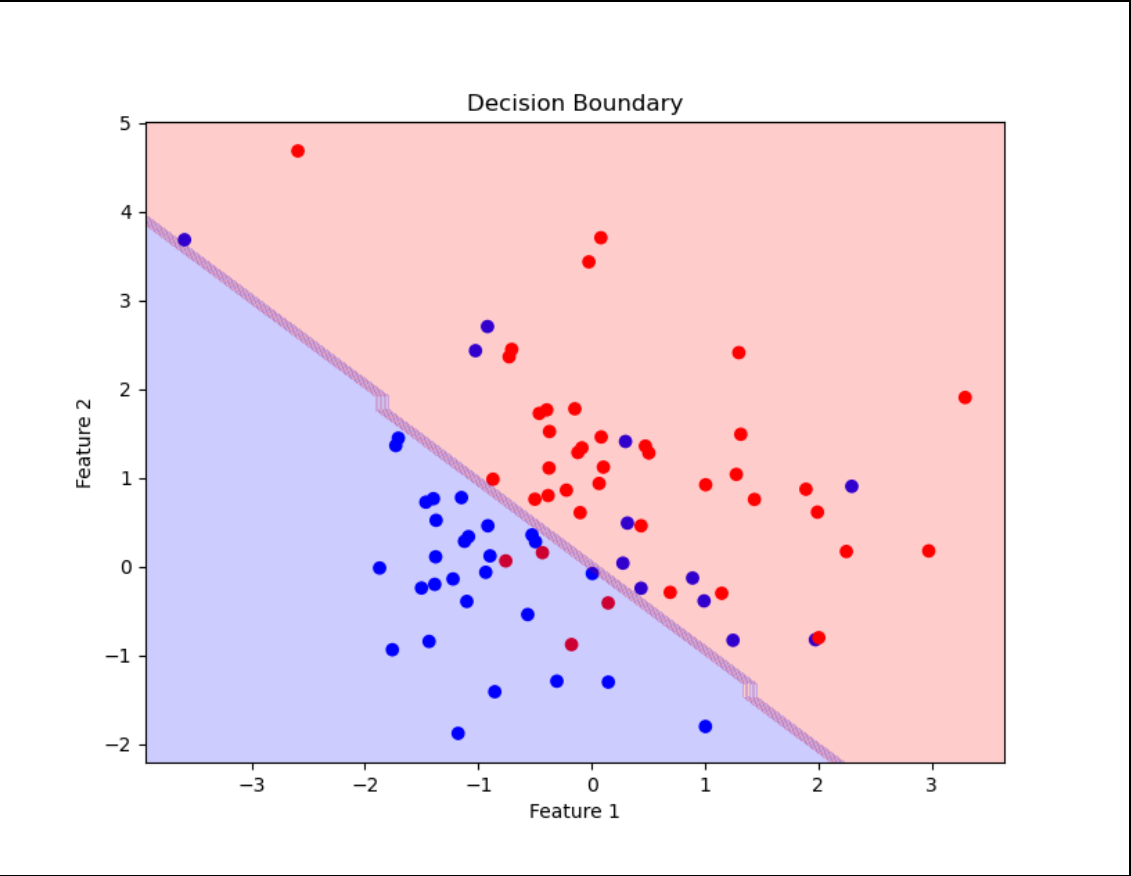
测试集

```
最终权重 w (梯度下降法): [-0.00985726 -0.00987751]
最终偏置 b (梯度下降法): -4.860897808623222e-06
Accuracy: 0.8
F1 Score: 0.7777777777777777
Precision: 0.875
Recall: 0.7
Confusion Matrix: {'True Positives': 28, 'False Positives': 4, 'True Negatives': 36, 'False Negatives': 12}
```

求解方法	梯度下降
超参数	Lr=1e-3,max_epochs=10,bs=320(full)
分类面	-4.86+-0.0099*x-0.00988*y=0
测试集准确率	ACC=0.7593
训练集准确率	ACC=0.8
损失函数曲线	



可视化结果



4, 改变算法中的各类超参数、样本数量、样本分布等, 对于梯度下降法还要改变不同的学习率以及不同的 **batch size** 和不同 **epoch** 次数, 讨论实验结果。

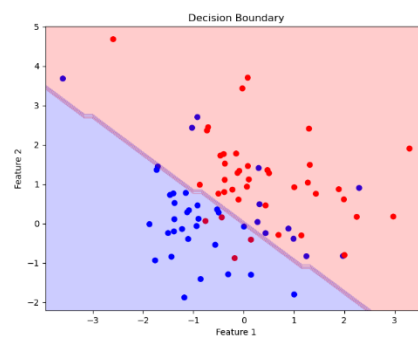
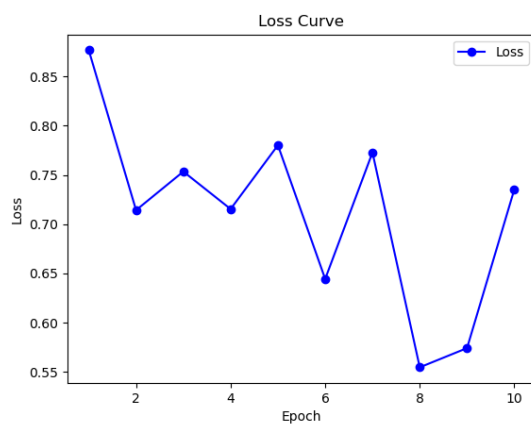
改变 **batch_size** 为 $0.1 \times \text{dataset}$ 采用随机梯度下降法对于第三题中的结果获得了相同解, 损失函数更为波动。下面分别展示了 10 个 epoch 和 100 个 epoch 的结果

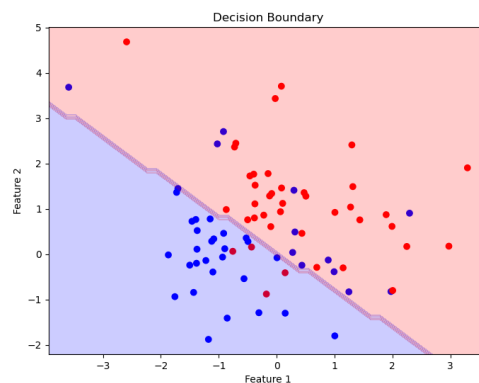
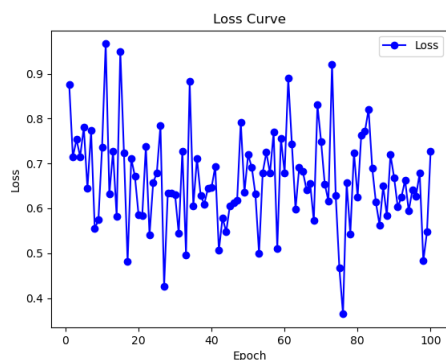
10epoch

```
最终权重 w (梯度下降法): [-0.30086748 -0.34003265]
最终偏置 b (梯度下降法): -0.000902320054584918
accuracy: 0.8
F1 Score: 0.7777777777777777
precision: 0.875
recall: 0.7
Confusion Matrix: {'True Positives': 28, 'False Positives': 4, 'True Negatives': 36, 'False Negatives': 12}
```

100epoch

```
最终权重 w (梯度下降法): [-0.30956921 -0.36829177]
最终偏置 b (梯度下降法): 0.011241025577772722
Accuracy: 0.8
F1 Score: 0.7777777777777777
Precision: 0.875
Recall: 0.7
Confusion Matrix: {'True Positives': 28, 'False Positives': 4, 'True Negatives': 36, 'False Negatives': 12}
```

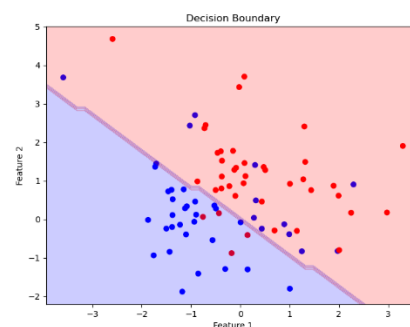
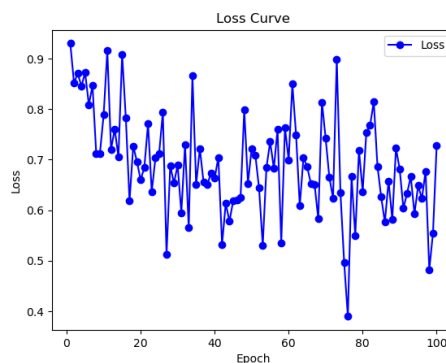




对于随机梯度下降维持其他超参不变，分别采用 Adagrad、RMSProp、Adam 的更新方式结果如下，可以看到损失函数的不同：

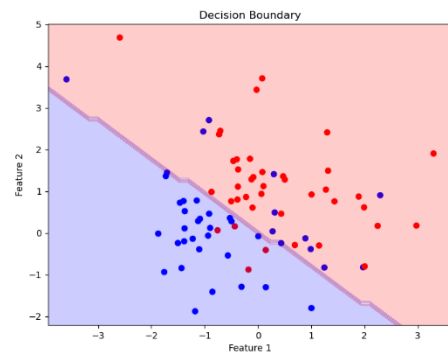
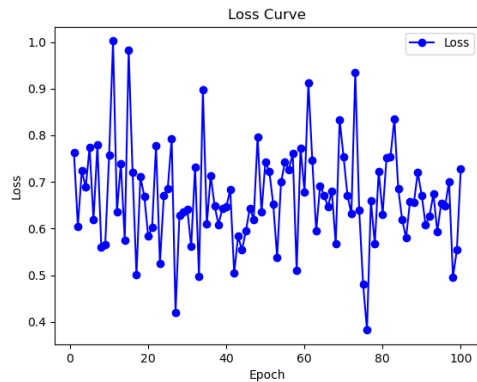
Adagrad:

```
最终权重 w (梯度下降法): [-0.30053677 -0.33826908]
最终偏置 b (梯度下降法): -0.0009617335204278667
Accuracy: 0.8
F1 Score: 0.7777777777777777
Precision: 0.875
Recall: 0.7
Confusion Matrix: {'True Positives': 28, 'False Positives': 4, 'True Negatives': 36, 'False Negatives': 12}
```



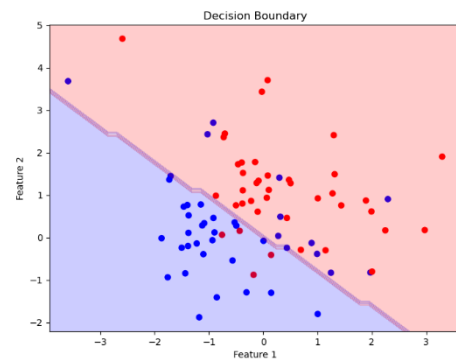
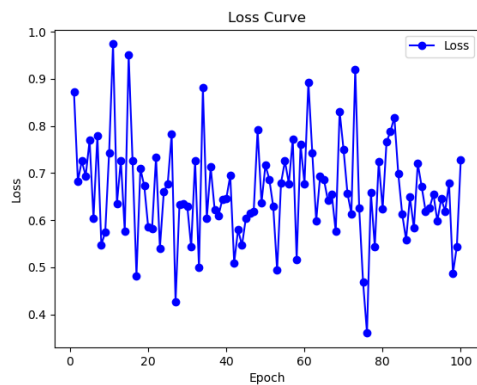
RMSProp:

```
最终权重 w (梯度下降法): [-0.31933278 -0.36957036]
最终偏置 b (梯度下降法): 0.020223538310825087
Accuracy: 0.8
F1 Score: 0.7777777777777777
Precision: 0.875
Recall: 0.7
Confusion Matrix: {'True Positives': 28, 'False Positives': 4, 'True Negatives': 36, 'False Negatives': 12}
```



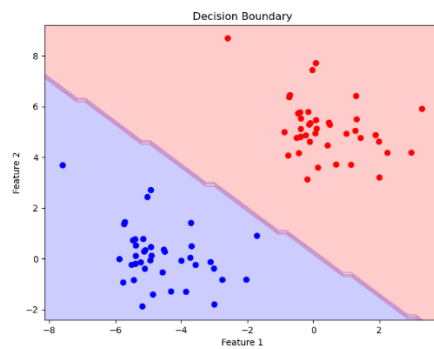
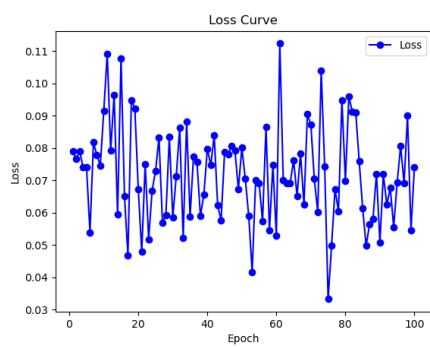
Adam

```
最终权重 w (梯度下降法): [-0.313023 -0.36552259]
最终偏置 b (梯度下降法): 0.018277344132065863
Accuracy: 0.8
F1 Score: 0.7777777777777777
Precision: 0.875
Recall: 0.7
Confusion Matrix: {'True Positives': 28, 'False Positives': 4, 'True Negatives': 36, 'False Negatives': 12}
```

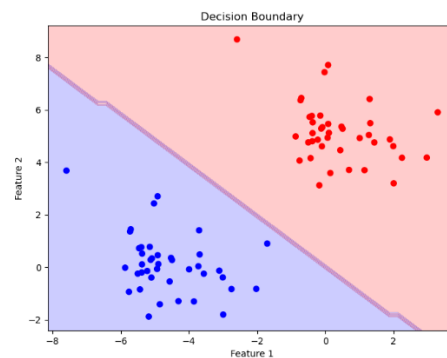
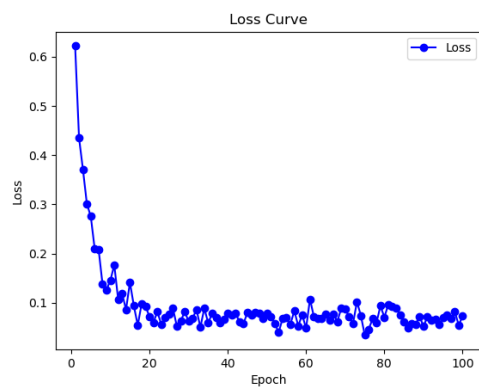


因为数据点太接近了，线性不可分，下面用第二题的分布，更为直观的
比较几种参数更新方法的区别：

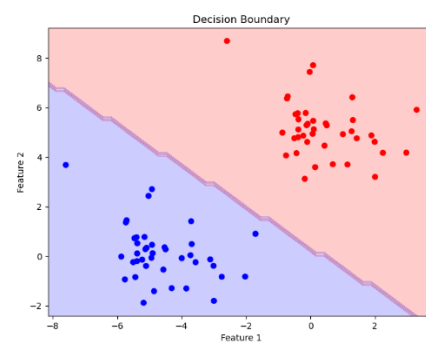
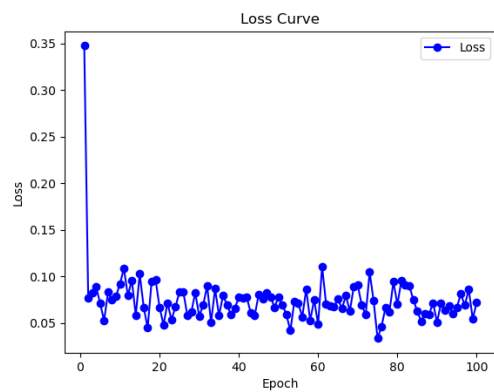
SGD:



Adagrad:



Adam:

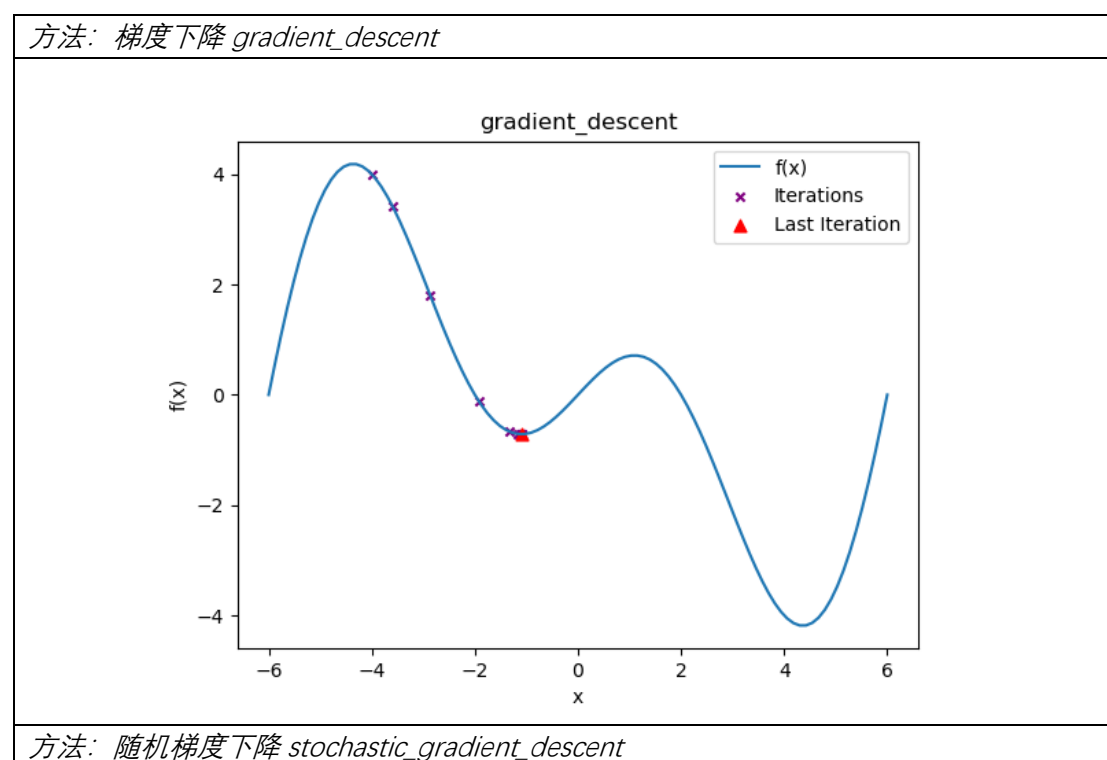


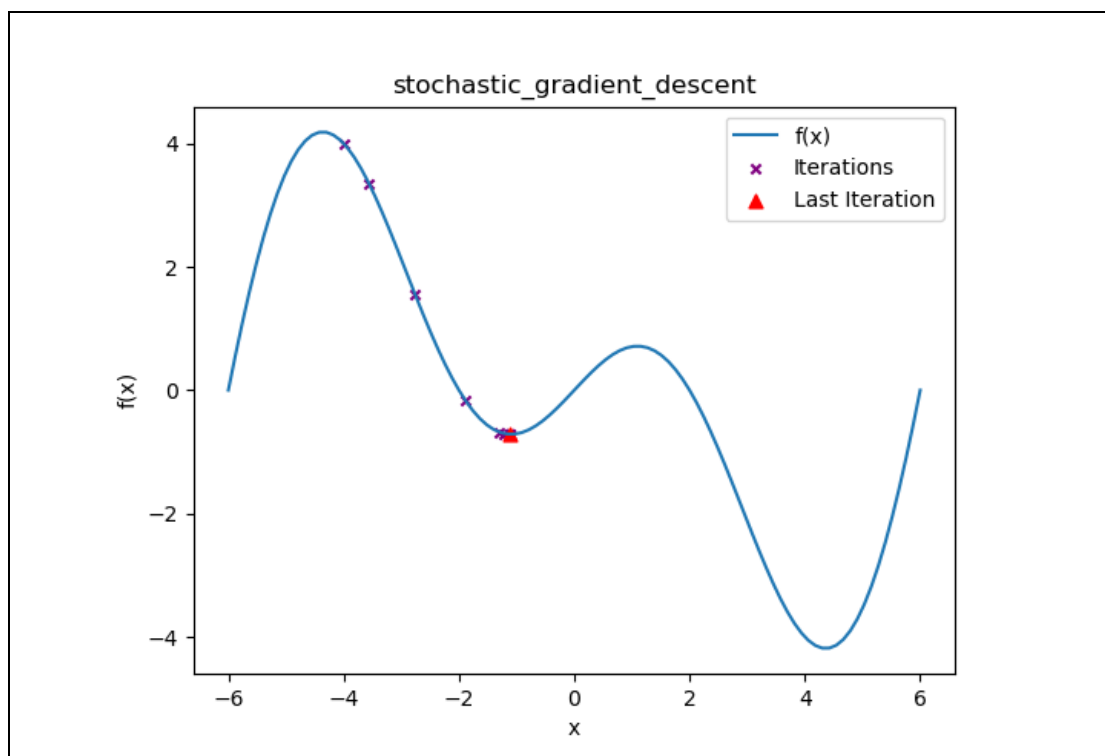
5, 单变量函数为 $f(x) = x * \cos(0.25\pi * x)$, 分别用梯度下降法、随机梯度下降法、Adagrad、RMSProp、动量法 (Momentum) 和 Adam

共 6 种方法，编写程序画图呈现 x 从初始值为-4、迭代 10 次时 x 及 $f(x)$ 的每次变化情况，这里对所有算法学习率（或初始学习率）均为 0.4, 为防止分母为 0 时给的最小量为 $\varepsilon = 1e-6$, RMSProp 算法的 $\alpha=0.9$, 动量法的 $\lambda=0.9$, Adam 的 $\text{beta1}=0.9$, $\text{beta2}=0.999$, 观察不同算法的变化情况体会各自的差异。如果迭代 50 次，并将 Adam 的 beta1 改成 0.99，其他参数不变，观察不同算法的变化结果。尝试调整上述算法的各种参数，体会上述不同方法的特点。

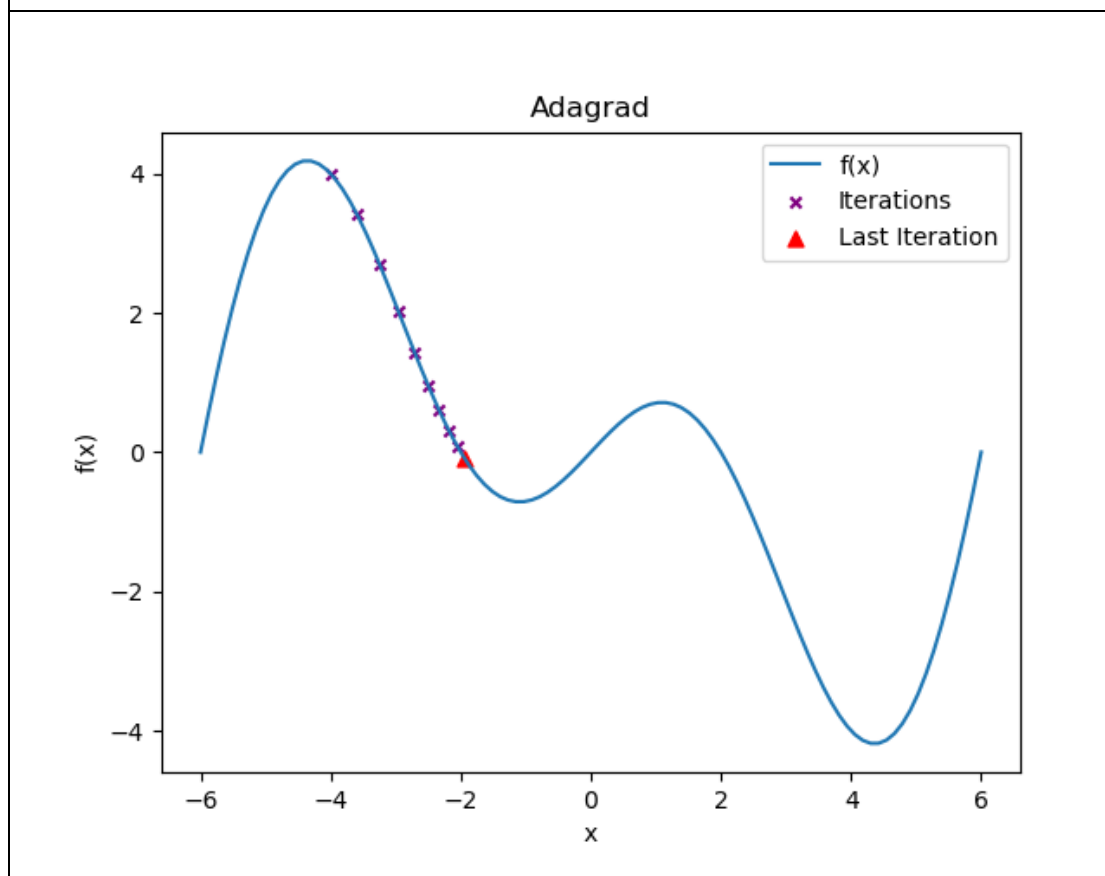
这一题的 SGD 使用高斯噪声

首先是 iteration=10 的结果：

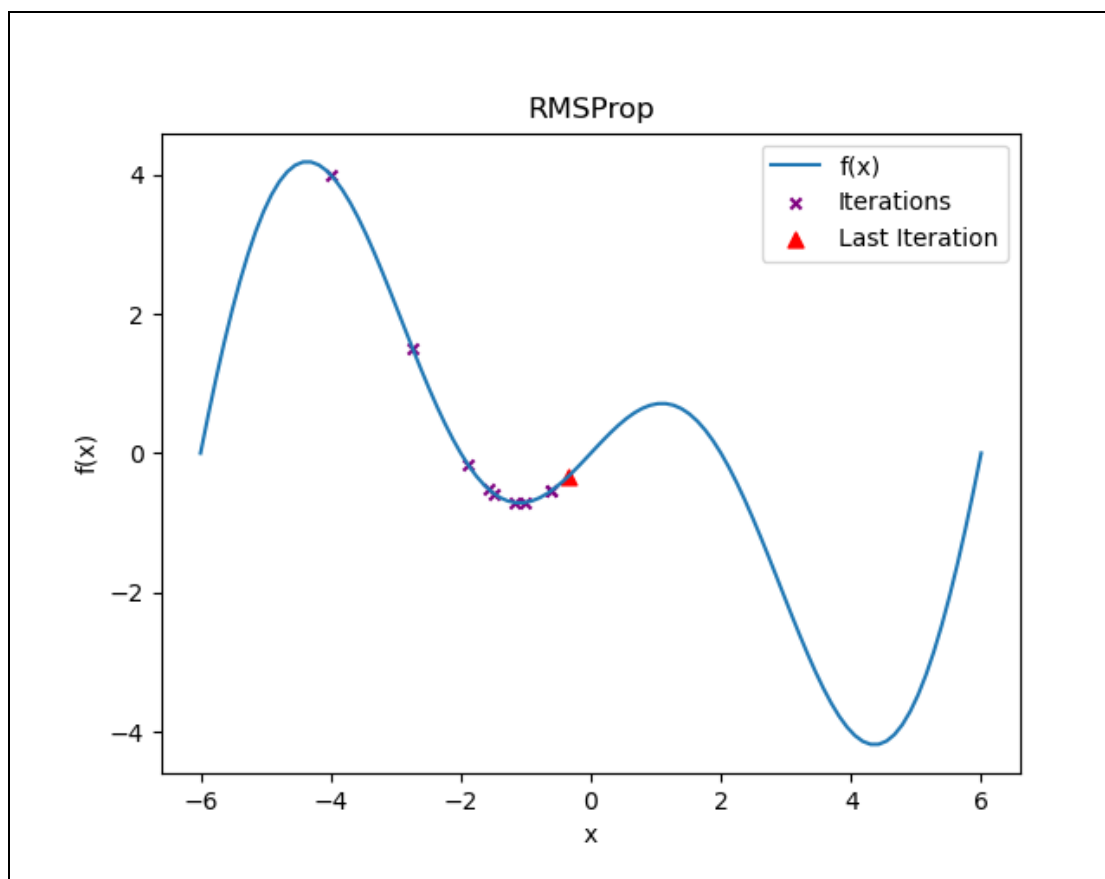




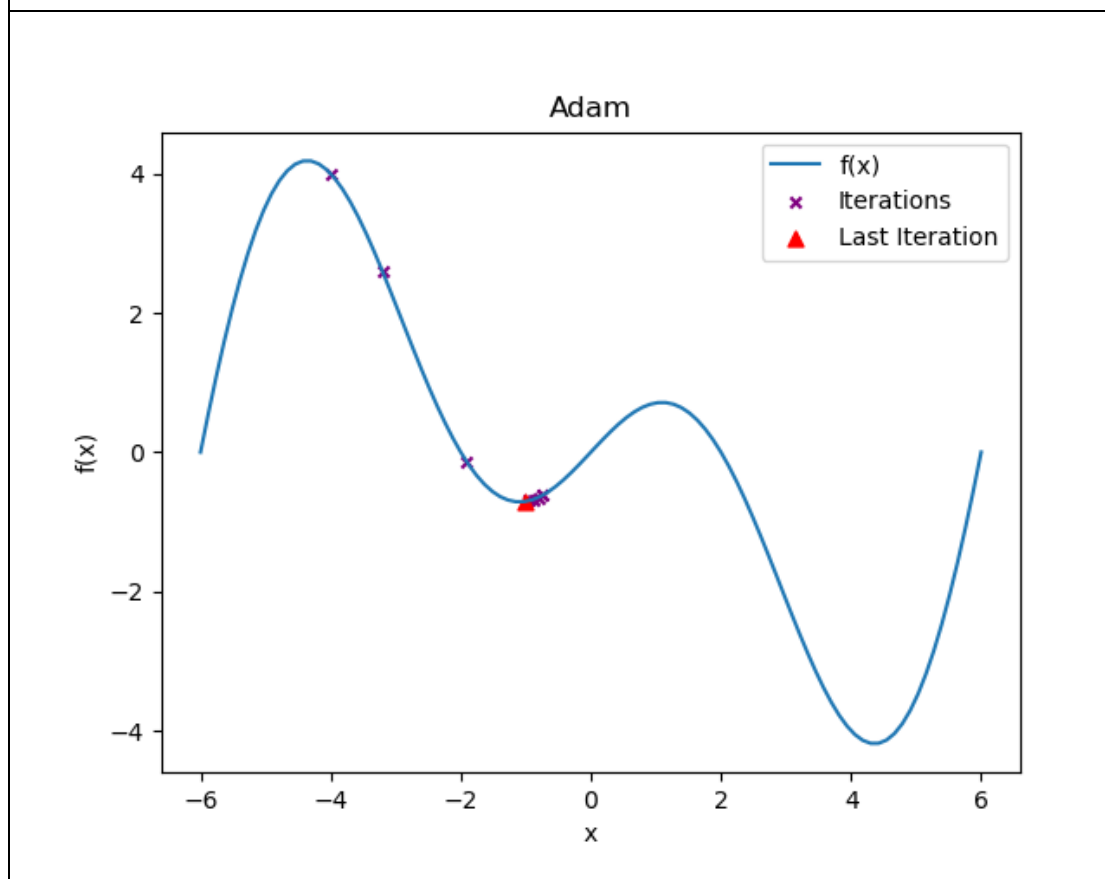
方法：自适应梯度下降 *Adagrad*



方法： *RMSProp*

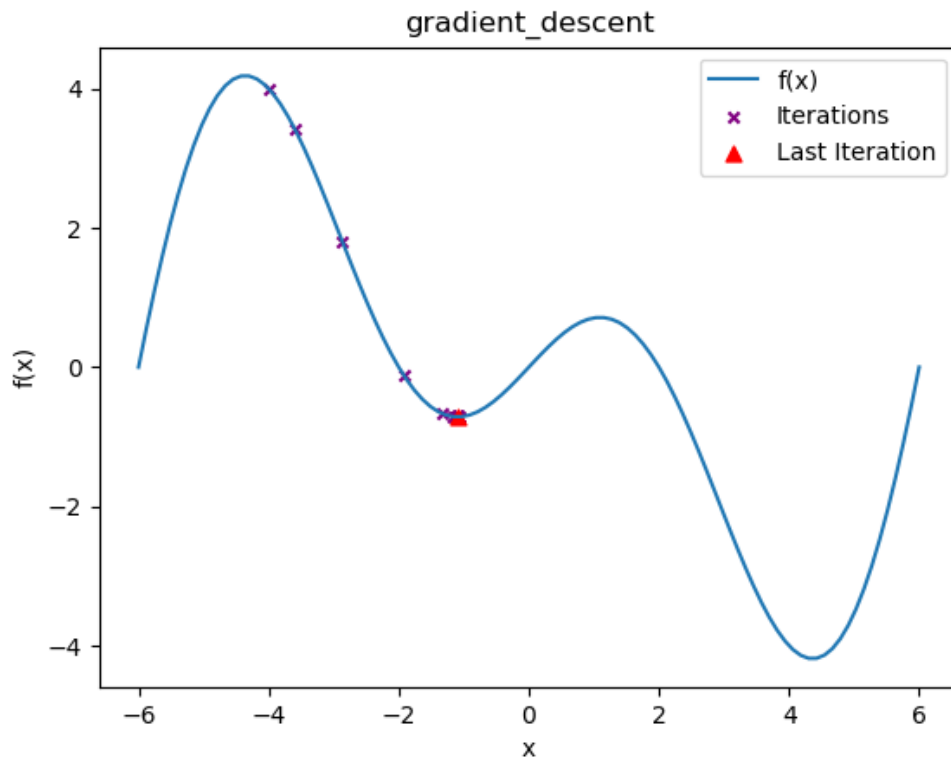


方法: Adam

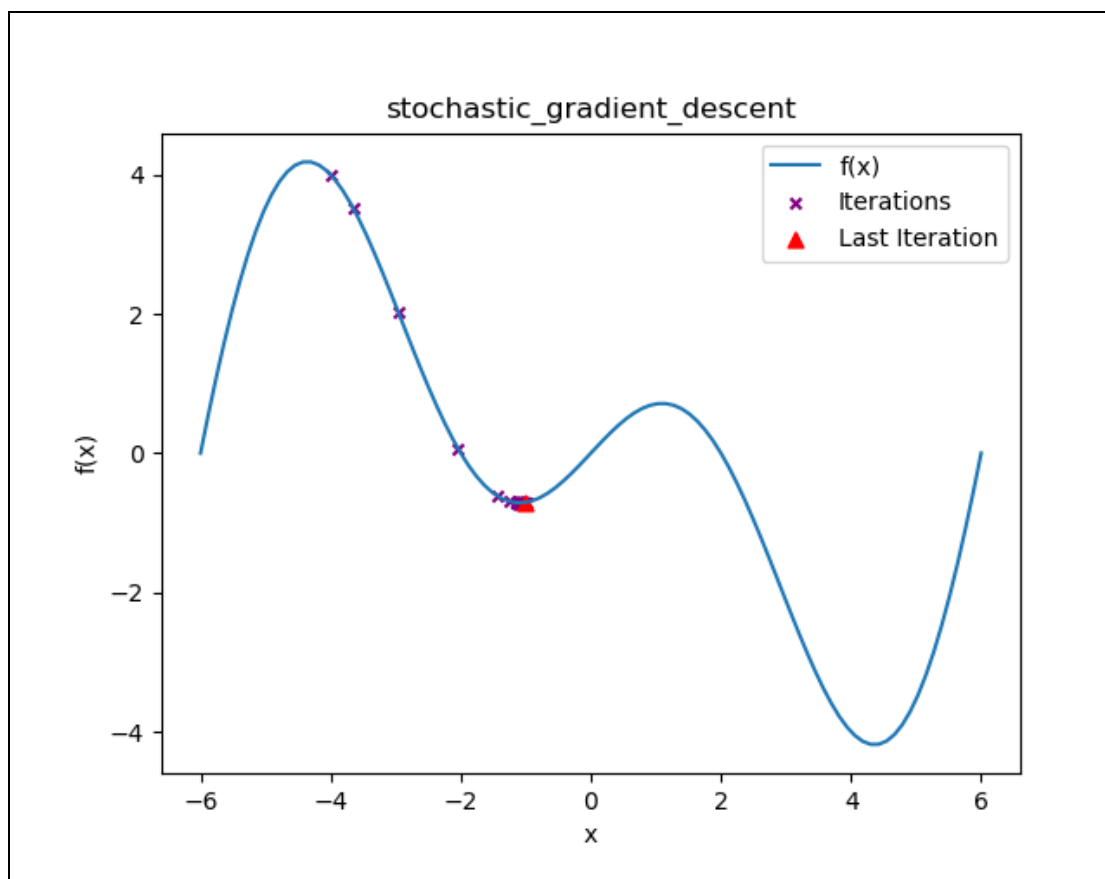


iteration=50,beta1=0.99 的结果:

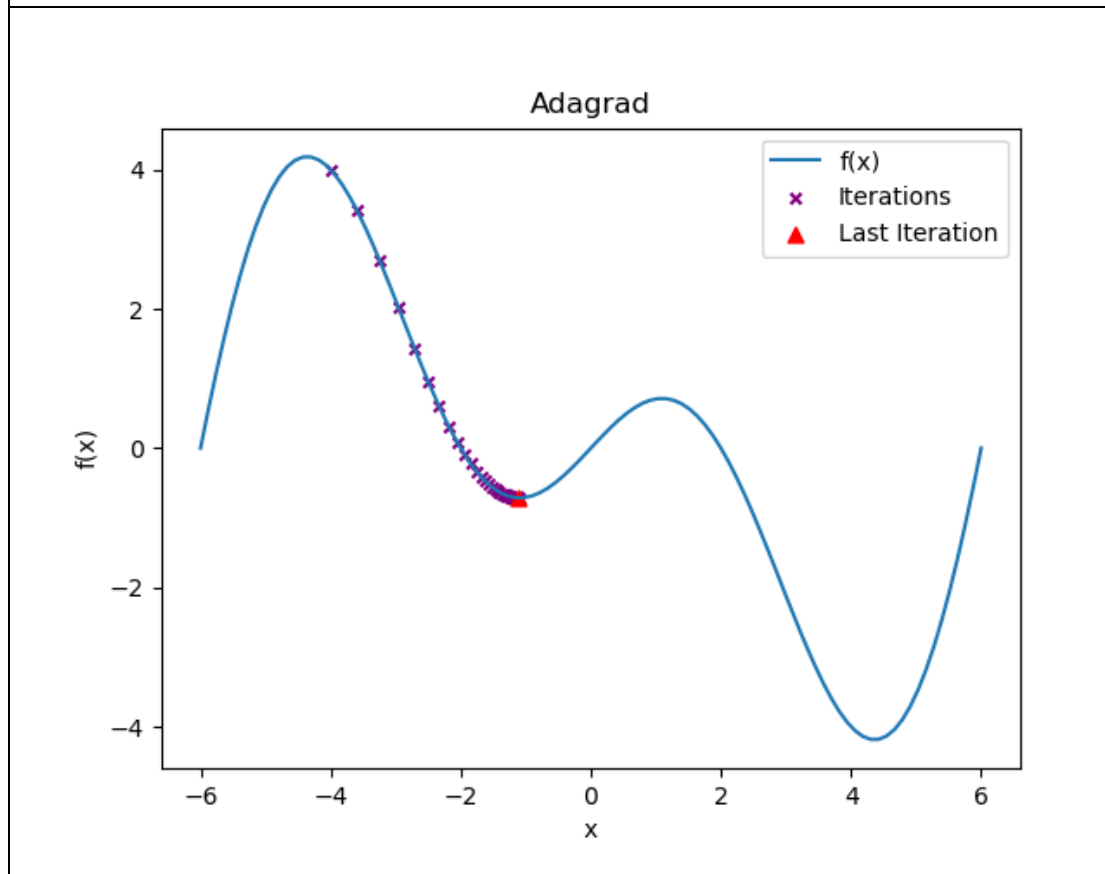
方法: 梯度下降 *gradient_descent*



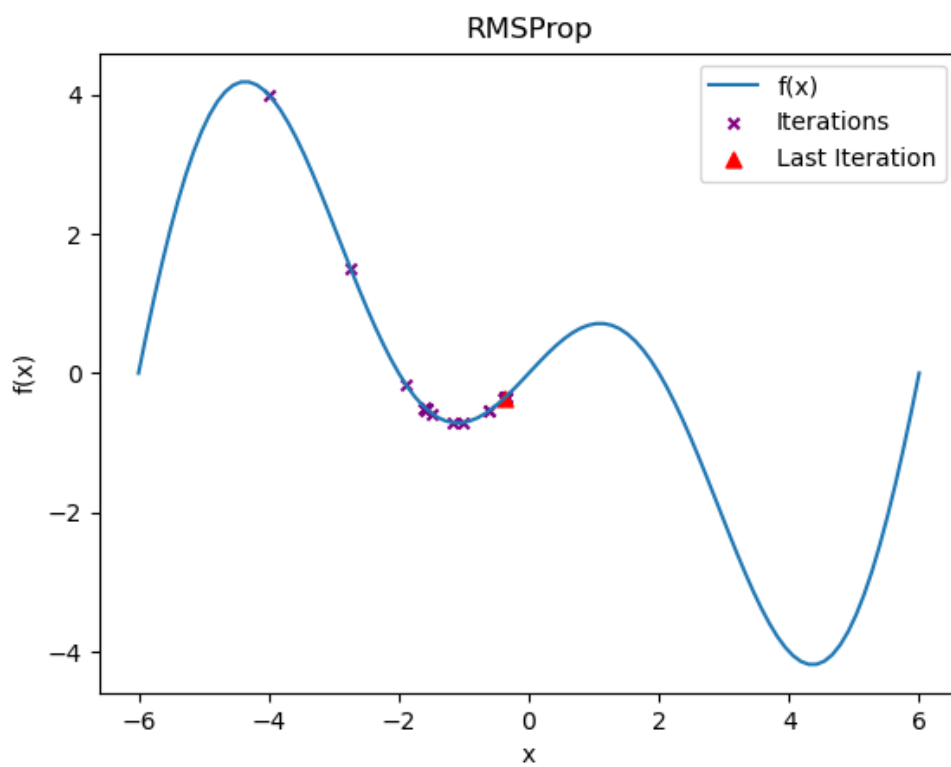
方法: 随机梯度下降 *stochastic_gradient_descent*



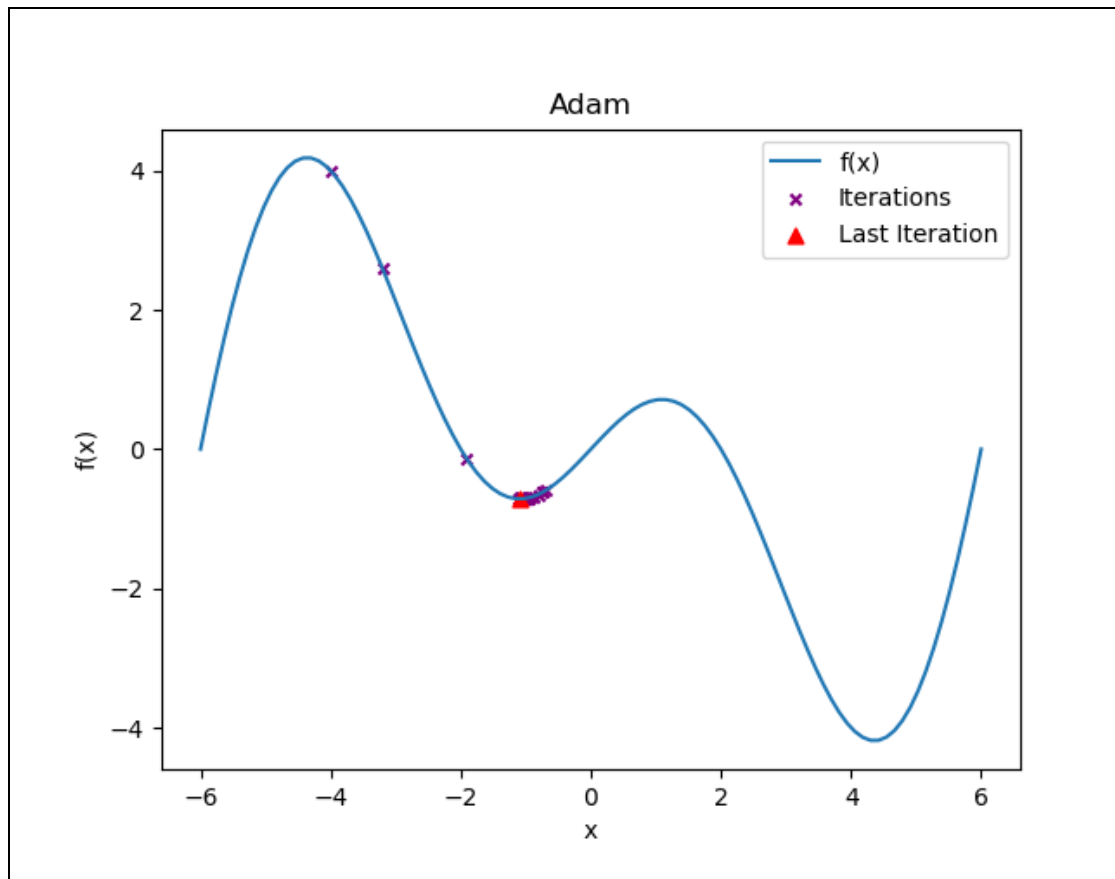
方法：自适应梯度下降 *Adagrad*



方法: *RMSProp*

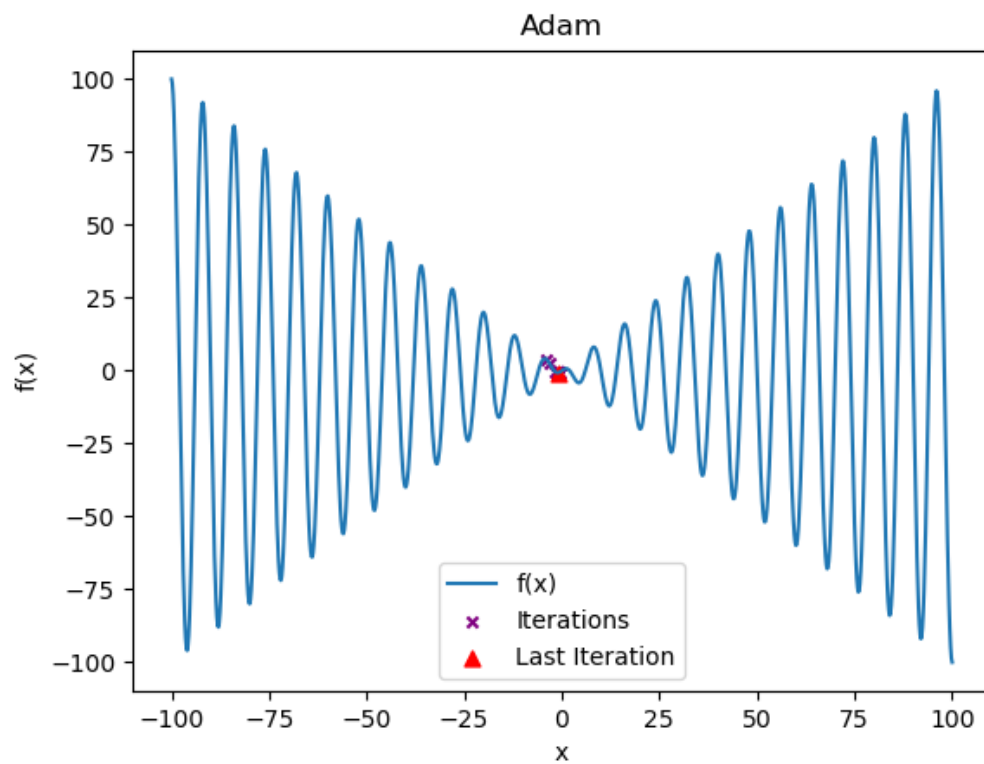


方法: *Adam*



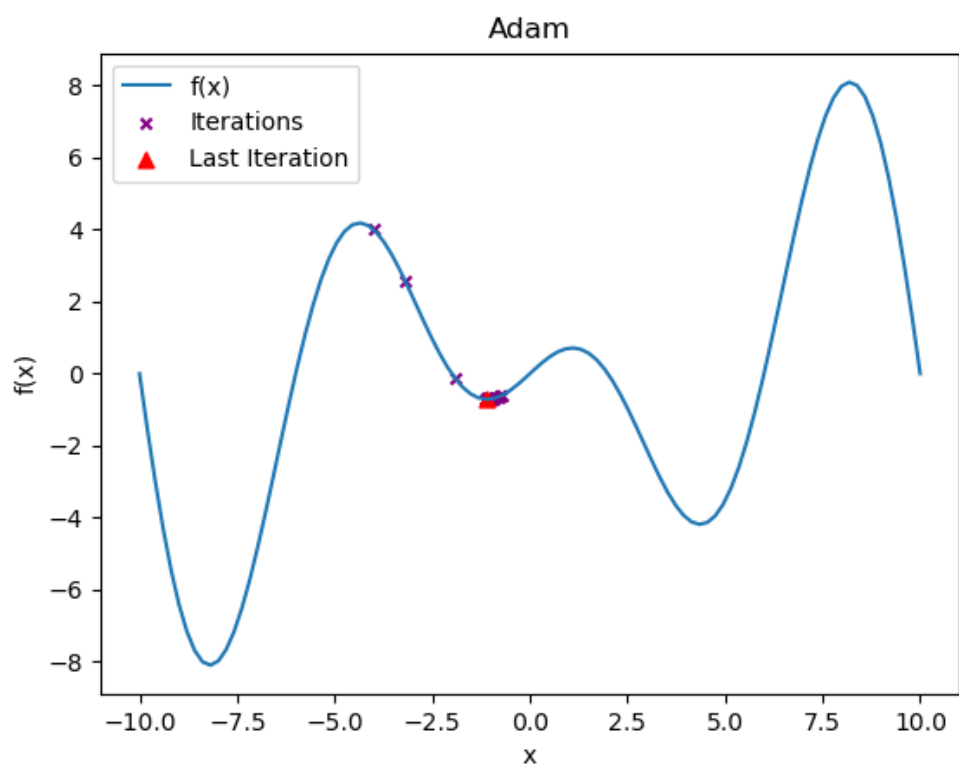
按照要求完成了仿真实验，问题是结果似乎都陷入了局部最优，能不能冲出局部最优解呢？

首先需要了解函数本身有没有最小值

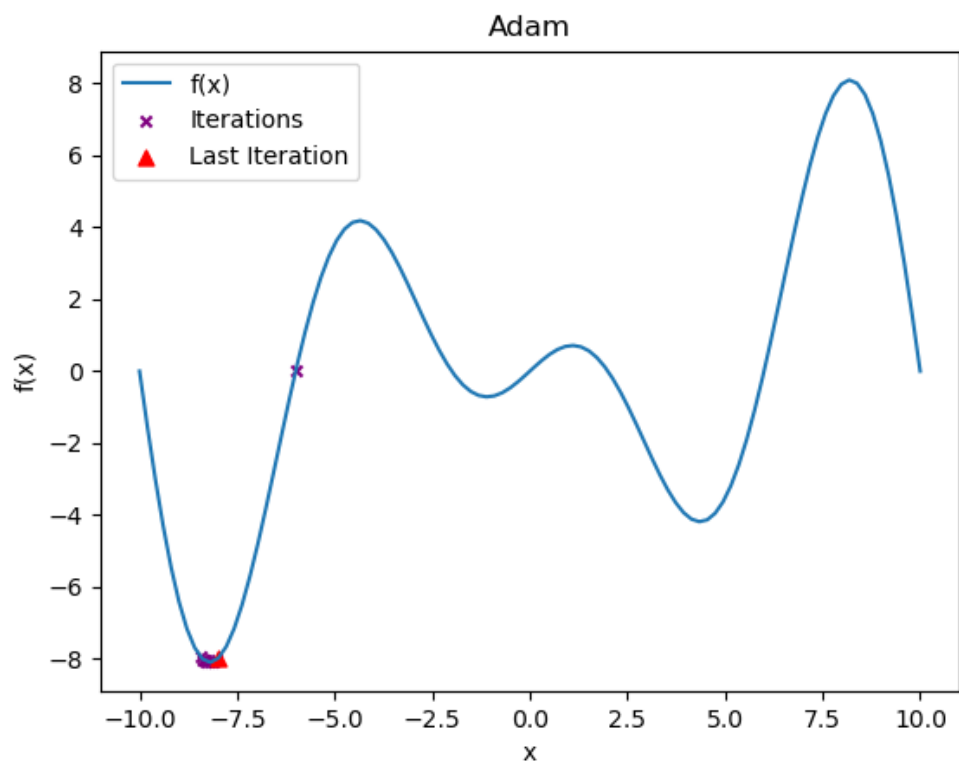


可以看到，函数发散，所谓的最小值，只能按照一定区间去选取，才有意义。

那么选取-10，10，如何获得最优解呢？：

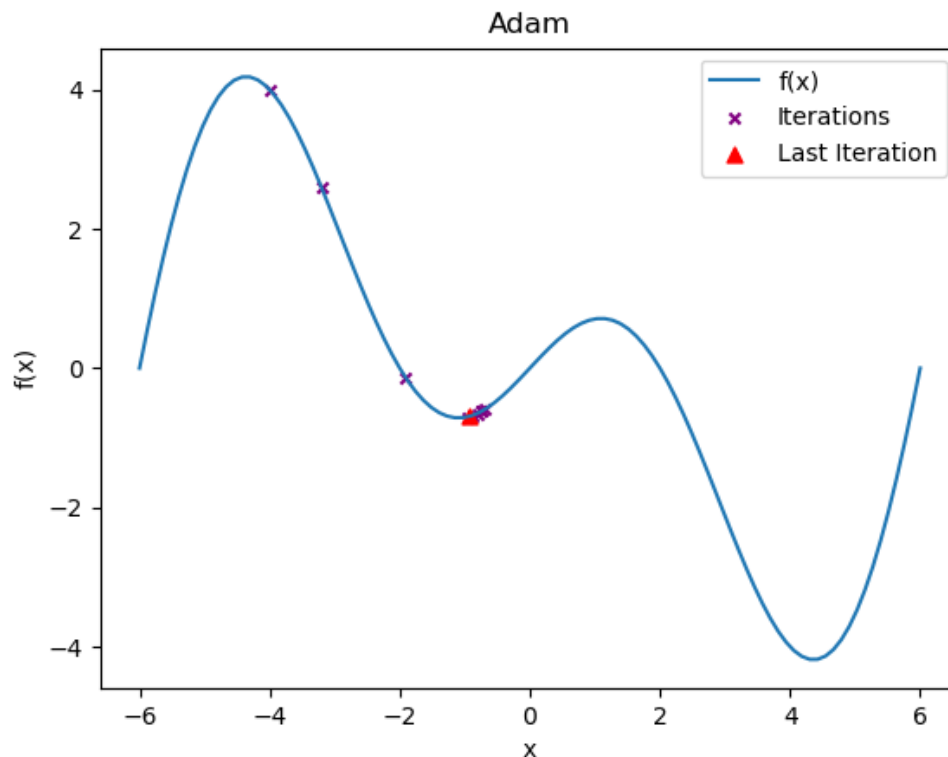


尝试改变初始化的值,从-5 开始,用第一问相同的配置,迭代 10 次:



发现实现了相对最优解的获取。

下一步，如果规定-6 到 6 的范围，如何冲出局部最优呢？

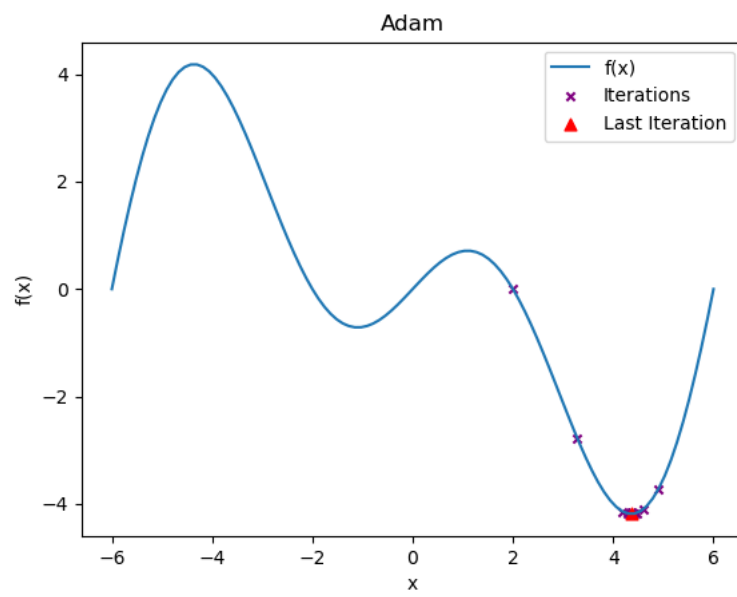


尝试调大初始学习率。和减少 beta 的衰减值。发现无论如何调参，无论使用何种算法，都无法让最后的点稳定的收敛在-4 附近的那个点。就算偶然有几次能到达 4 的位置，冲出这个破但也会无法停留。

唯有改变初始位置

最后我尝试改变初始位置：

终于在初始位置改为 2 以后，达到了右边最优解



结论：可见权重初始化对于模型本身有多重要！