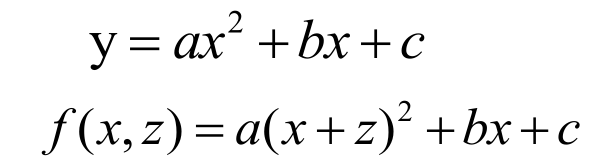
# 大数据第三次作业

## 题目

定义一个一元或者多元函数，例如

  

利用梯度下降算法求解最小值，并分析不同学习率（learning rate ）对于算法收敛

速度的影响。

## 分析问题

梯度下降算法是最常用的优化算法，在线性回归，神经网络训练等方面有重要应用。基本思想是自变量沿着梯度方向逐步移动没移动一次更新自变量的值，并计算新的损失，当损失降到我们可以接受的位置时，停止迭代。其核心公式为：



X的更新方向即为与梯度相反的方向。其中学习速率对结果有重要影响，另外梯度下降是典型的贪心算法，容易陷入局部最优。

## 算法实现与分析

我们的目标函数是：



利用梯度下降求其最小值，观察学习速率lr对梯度下降结果的影响。针对上述函数我们知道x=-1时函数取得最小值。

代码如下：

import numpy as np

import matplotlib.pyplot as plt

# 目标函数:y=x^2+2\*x+1

def func(x):

return x\*\*2 + 2 \* x + 1

# 目标函数一阶导数也即是偏导数:dy/dx=2\*x+2

def dfunc(x):

return 2 \* (x+1)

def cal\_loss(x): # 损失函数定义当前的x与最优解之间的距离

return (x + 1) \*\* 2

def GD(x\_start, df, epochs, lr):

xs = []

x = x\_start

xs.append(x\_start)

min\_loss = 0.01

i = 0

loss = cal\_loss(x)

while i <= epochs and loss > min\_loss:

print("迭代次数：" + str(i + 1))

dx = df(x)

v = - dx \* lr

x += v

i += 1

xs.append(x)

loss = cal\_loss(x)

return xs

def demo\_GD():

# 演示如何使用梯度下降法GD()

line\_x = np.linspace(-5, 5, 100)

line\_y = func(line\_x)

x\_start = -5 # 从-5开始进行梯度下降

epochs = 10000000 # 最大迭代次数

lr = 0.1 # 学习速率

x = GD(x\_start, dfunc, epochs, lr=lr)

x = np.array(x)

print(x)

print(x[-1])

color = 'r'

plt.plot(line\_x, line\_y, c='b')

plt.plot(x, func(x), c=color, label='lr={}'.format(lr))

plt.scatter(x, func(x), c=color)

plt.legend()

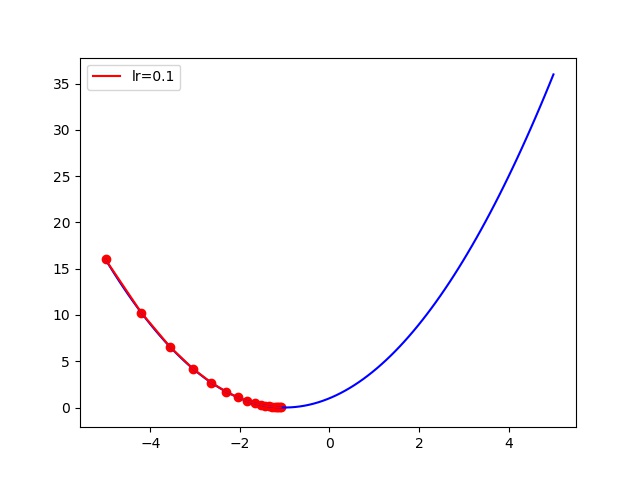
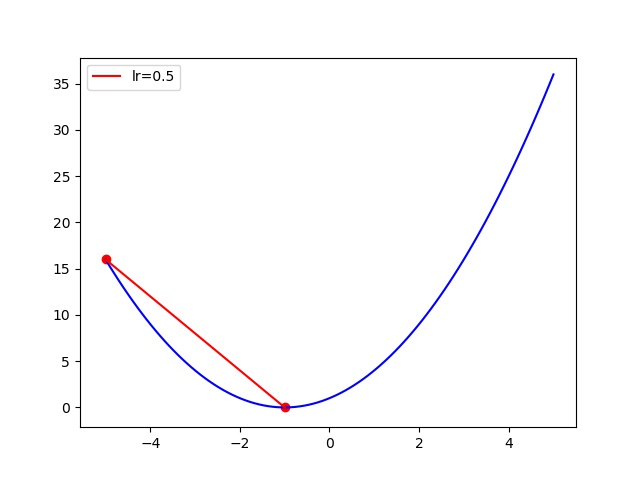
plt.savefig('b.jpg')

plt.show()

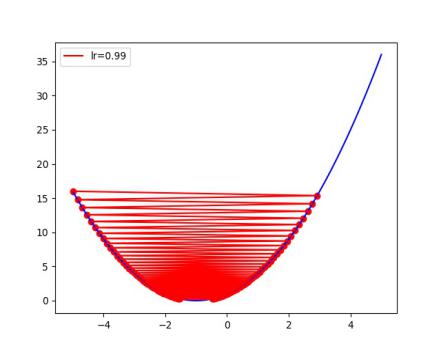
if \_\_name\_\_ == '\_\_main\_\_':

demo\_GD()

运行结果：



图a 图b



图c

实验结果表明学习速率lr取值不同时算法的收敛速度不同，当lr=0.1时迭代了17次才达到最低点，当lr提高到0.5时，值迭代了1次便到达最低点，表明适当提高lr可以加快算法的收敛速度，但图c中lr取为0.99时算法一直震荡无法收敛到最低点。说明若lr取值过大梯度下降法有可能不能收敛而一直震荡下去。