**最优化理论与算法 作业 1**

**23020211153893 华浩宇**

1. 编程实现割线法。

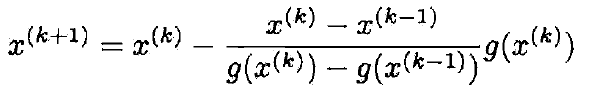
在牛顿法中，每一步都需要计算当前点的导数值，需要手动求导。如果不想人工求导，可以使用两点割线来代替切线，从而得到割线法， 割线法是一维搜索方法的一种，割线法使用的是第k-1个和第k个迭代点之间的割线确定第k+1个迭代点，并且两个位置不能太远，否则会超过零点的位置。

使用割线法求解方程𝑔(𝑥)=0，已知初始值为𝑥(−1)=0，𝑥(0)=1，𝜀=10−5

1：首先我们将初始点带入𝑔(𝑥)：

由：𝑔(𝑥) = (2𝑥 − 1)3 + 4(4− 1024𝑥)

2：利用公式：

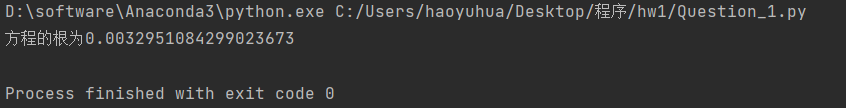


求解：𝑥(1)

3：再利用𝑥(0)和𝑥(1) 进行新一轮的迭代。

4：其中收敛条件为 |𝑥(𝑘+1)−𝑥(𝑘)|<𝜀|𝑥(𝑘)|，𝜀>0

import numpy as np  
  
  
def f(x):  
 return np.array((2 \* x - 1) \*\* 3 + 4 \* (4 - 1024 \* x) \*\* 3)  
  
  
def secant(f, x0, e=0.00001):  
 assert (len(x0) == 2)  
 x1, x0 = x0[1], x0[0]  
 x = []  
 iter = 1  
 while True:  
 x2 = x1 - (x1 - x0) \* f(x1) / (f(x1) - f(x0))  
 print("第{}次迭代，x值为{}".format(iter, x2))  
 x.append(x2)  
 iter = iter + 1  
 tol = abs(x2 - x1)  
 if tol < e \* abs(x1):  
 return x2, x  
 else:  
 x0 = x1  
 x1 = x2  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 x0 = [0, 1]  
 x, \_ = secant(f, x0)  
 print("方程的根为{}".format(x))



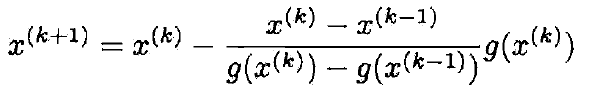
1. 编写程序，利用割线法实现多维优化问题中的一维搜索算法。

利用割线法实现多维优化问题中的一维搜索算法：

设计函数，应用最速下降法，让目标数值能够找到最大程度的减少。

1：实现求函数梯度方法

2：牛顿割线法求解步长带入公式：



3：退出条件：

|𝒅T∇𝑓(𝒙 + 𝛼𝒅)| ≤ 𝜀|𝒅T

其中𝜀为预设精度，∇𝑓表示梯度，𝒙为当前迭代点，𝒅为搜索方向。该停止规则的含义为按照比例𝜀来降低目标函数𝑓在方向𝒅。否则迭代搜索。

def linesearch\_secant(f, df, d, x, alpham, rho, t):  
 *'''  
 线性搜索子函数，函数f，导数df，当前迭代点x和当前搜索方向d  
 '''* flag = 0  
  
 a = 0  
 b = alpham  
 fk = f(x)  
 gk = df(x)  
  
 phi0 = fk  
 dphi0 = np.dot(gk, d)  
 alpha = b \* random.uniform(0, 1)  
  
 while (flag == 0):  
 newfk = f(x + alpha \* d)  
 phi = newfk  
 if (phi - phi0) <= (rho \* alpha \* dphi0):  
 if (phi - phi0) >= ((1 - rho) \* alpha \* dphi0):  
 flag = 1  
 else:  
 a = alpha  
 b = b  
 if (b < alpham):  
 alpha = (a + b) / 2  
 else:  
 alpha = t \* alpha  
 else:  
 a = a  
 b = alpha  
 alpha = (a + b) / 2  
 return alpha

1. 编写程序，实现最速下降法, 其中一维搜索过程采用割线法。

根据梯度函数求出初始点为𝑥(0) = [−2,2]T

1：根据初始点，求出当前函数，在初始点的梯度∇g（xk）。

2：一维搜索的方向为-∇g（xk）

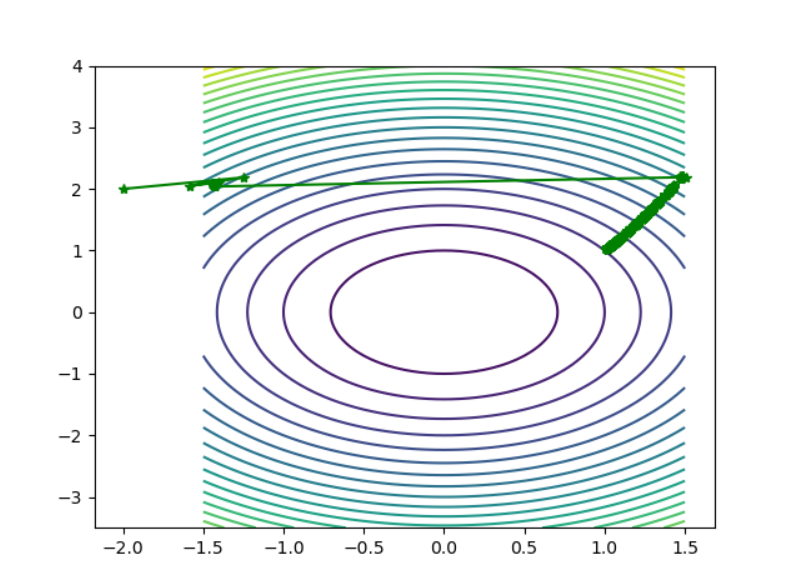
3：利用2题中算出最优步长。

4：利用步长和公式算出下一个迭代点：



5：重复过程直到达到条件：‖𝒈(𝑘)‖ < 𝜀，其中𝜀 = 10−4

结果如下：





import random  
import numpy as np  
import matplotlib.pyplot as plt  
  
  
def linesearch\_secant(f, df, d, x, alpham, rho, t):  
 *'''  
 线性搜索子函数，函数f，导数df，当前迭代点x和当前搜索方向d  
 '''* flag = 0  
  
 a = 0  
 b = alpham  
 fk = f(x)  
 gk = df(x)  
  
 phi0 = fk  
 dphi0 = np.dot(gk, d)  
 alpha = b \* random.uniform(0, 1)  
  
 while (flag == 0):  
 newfk = f(x + alpha \* d)  
 phi = newfk  
 if (phi - phi0) <= (rho \* alpha \* dphi0):  
 if (phi - phi0) >= ((1 - rho) \* alpha \* dphi0):  
 flag = 1  
 else:  
 a = alpha  
 b = b  
 if (b < alpham):  
 alpha = (a + b) / 2  
 else:  
 alpha = t \* alpha  
 else:  
 a = a  
 b = alpha  
 alpha = (a + b) / 2  
 return alpha  
  
  
def rosenbrock(x):  
 return 100 \* (x[1] - x[0] \*\* 2) \*\* 2 + (1 - x[0]) \*\* 2  
  
  
def jacobian(x):  
 return np.array([-400 \* x[0] \* (x[1] - x[0] \*\* 2) - 2 \* (1 - x[0]), 200 \* (x[1] - x[0] \*\* 2)])  
  
  
def steepest(x0):  
 print('初始点为:')  
 print(x0, '\n')  
 imax = 20000  
 W = np.zeros((2, imax))  
 epo = np.zeros((2, imax))  
 W[:, 0] = x0  
 i = 1  
 x = x0  
 grad = jacobian(x)  
 delta = sum(grad \*\* 2) # 初始误差  
  
 f = open("最速.txt", 'w')  
  
 while i < imax and delta > 10 \*\* (-4):  
 p = -jacobian(x)  
 x0 = x  
 alpha = linesearch\_secant(rosenbrock, jacobian, p, x, 1, 0.1, 2)  
 x = x + alpha \* p  
 W[:, i] = x  
  
 epo[:, i] = np.array((i, delta))  
 f.write(str(i) + " " + str(delta) + "\n") #  
 print(i, np.array((i, delta)))  
 grad = jacobian(x)  
 delta = sum(grad \*\* 2)  
 i = i + 1  
 print("迭代次数为:", i)  
 print("近似最优解为:")  
 print(x, '\n')  
 W = W[:, 0:i] # 记录迭代点  
 return [W, epo]  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 X1 = np.arange(-1.5, 1.5 + 0.05, 0.05)  
 X2 = np.arange(-3.5, 4 + 0.05, 0.05)  
 [x1, x2] = np.meshgrid(X1, X2)  
  
 f = 2 \* x1 \*\* 2 + x2 \*\* 2  
 plt.contour(x1, x2, f, 20) # 画出函数的20条轮廓线  
 x0 = np.array([-2, 2])  
 list\_out = steepest(x0)  
 W = list\_out[0]  
 epo = list\_out[1]  
 plt.plot(W[0, :], W[1, :], 'g\*-') # 画出迭代点收敛的轨迹  
 plt.show()