**中国矿业大学计算机学院**

**2021 级本科生课程报告**

课程名称： 最优化理论与方法

报告题目： 《共轭梯度法原理及实现》

报告时间： 2024.1.10

学生姓名： 杨学通

学 号： 08213129

专 业： 人工智能2021-1班

任课教师： 赵佳琦

**课程报告成绩评定及评分依据**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 评分依据 | 分值 | 实际得分 |
| 1 | 课程报告描述了最优化技术或算法的基本原理 | 20 |  |
| 2 | 课程报告使用python或C++程序设计语言（附源代码） | 40 |  |
| 3 | 给出至少2个测试用例，并且给出可视化的运行结果及分析 | 30 |  |
| 4 | 课程报告给出了个人学习体会 | 10 |  |
| 总分 | | |  |
| 教师评语：  教师签名：  年 月 日 | | | |

**摘 要**

共轭梯度法（Conjugate Gradient）是介于最速下降法与牛顿法之间的一个方法，它仅需利用一阶导数信息，但克服了最速下降法收敛慢的缺点，又避免了牛顿法需要存储和计算Hessian矩阵并求逆的缺点，共轭梯度法不仅是解决大型线性方程组最有用的方法之一，也是解大型非线性最优化最有效的算法之一。 在各种优化算法中，共轭梯度法是非常重要的一种。其优点是所需存储量小，具有步收敛性，稳定性高，而且不需要任何外来参数。

**关键词**：共轭梯度法；线性方程组；非线性最优化；步收敛性

目录

[1问题描述 1](#_Toc155817813)

[2算法原理及流程 1](#_Toc155817814)

[3运行代码 2](#_Toc155817815)

[4运行截图 3](#_Toc155817816)

[5总结与分析 4](#_Toc155817817)

# 

# 1问题描述

共轭梯度法是一种常用的优化算法，特别适用于解决大规模线性方程组和非线性优化问题。以下是共轭梯度法的主要特点和总结：

收敛速度快：相对于最速下降法，共轭梯度法通常具有更快的收敛速度，尤其是对于二次型优化问题。

内存消耗低：共轭梯度法通常只需要存储少量的历史信息，因此在内存消耗方面较为节省，适合处理大规模问题。

适用范围广泛：共轭梯度法不仅适用于解决凸优化问题，还可以用于非凸优化问题，尤其是在求解大规模问题时表现优异。

具有自动搜索方向的特性：共轭梯度法能够利用先前搜索方向的信息，避免了在搜索过程中出现重复的搜索方向，从而提高了搜索效率。

对称正定矩阵的优势：对于二次型优化问题，共轭梯度法可以更快地收敛到最优解，尤其是对称正定矩阵的情况下。

本次实验要求python实现共轭梯度法，并通过测试样例，验证共轭梯度法的正确性，体会共轭梯度法进行多维搜索的过程。

# 2算法原理及流程

Step1:给定初始点，允许误差，置

Step2:计算

若，停止计算，得点，否则，进行下一步

Step3:构造搜索方向，

当时，

当时，

Step4:令，计算步长

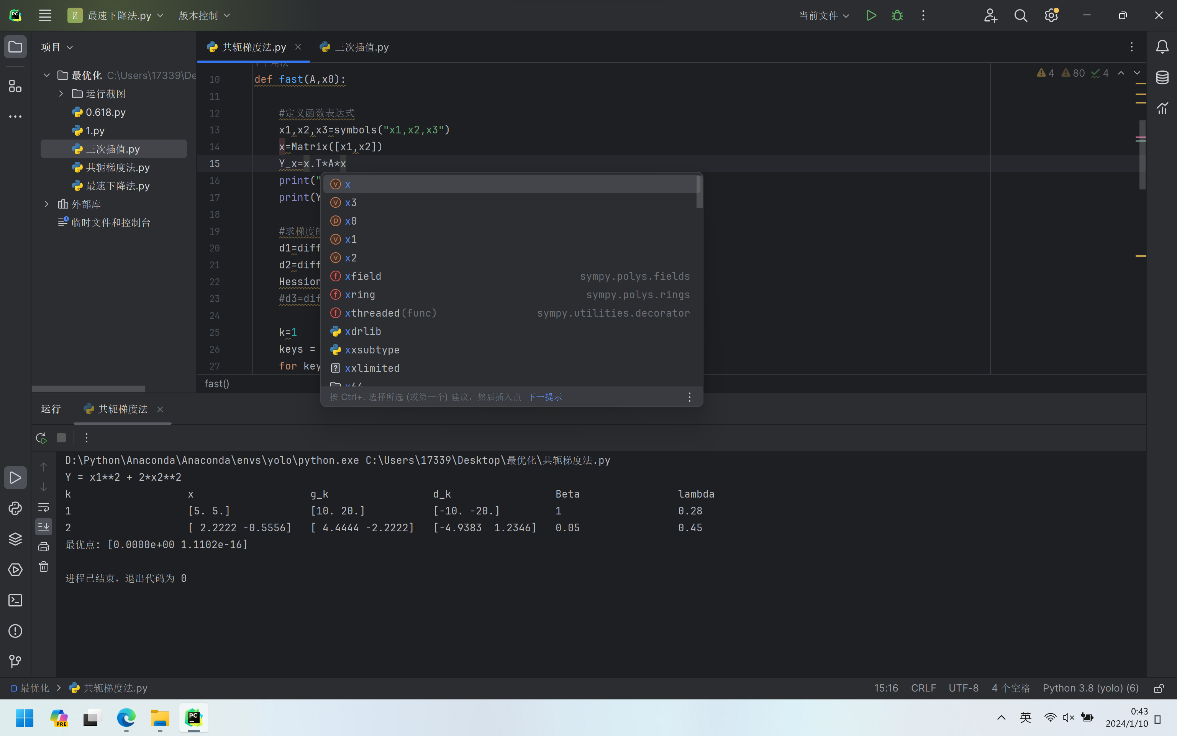
Step5:若，则停止计算得点，否则，置，转Step2

# 3运行代码

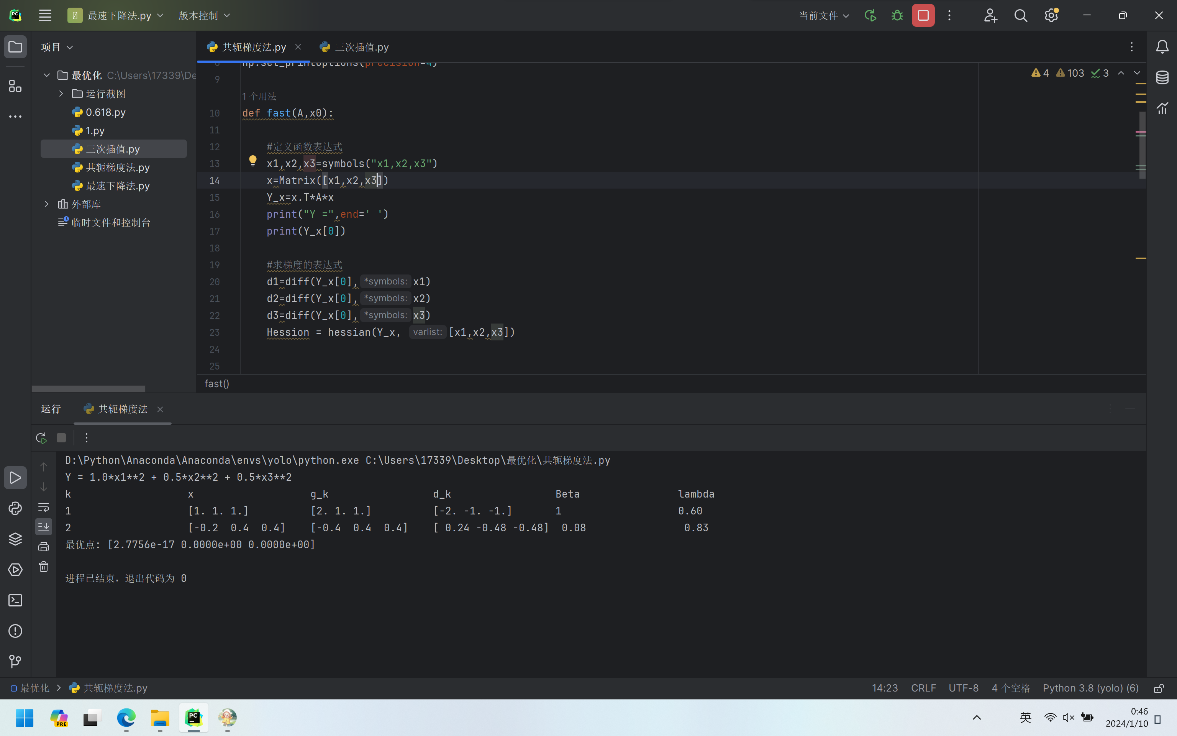
import math  
  
import numpy as np  
import sympy  
from sympy import \*  
import mpmath  
  
np.set\_printoptions(precision=4)  
  
def fast(A,x0):  
  
 *#定义函数表达式* x1,x2,x3=symbols("x1,x2,x3")  
 x=Matrix([x1,x2,x3])  
 Y\_x=x.T\*A\*x  
 print("Y =",end=' ')  
 print(Y\_x[0])  
  
 *#求梯度的表达式* d1=diff(Y\_x[0],x1)  
 d2=diff(Y\_x[0],x2)  
 d3=diff(Y\_x[0],x3)  
 Hession = hessian(Y\_x, [x1,x2,x3])  
  
  
 k=1  
 keys = ["k", "x", "g\_k","d\_k", "Beta", "lambda"]  
 for key in keys:  
 print('%-18s' % key, end=" ")  
 print()  
 while 1:  
  
 *#求x[0]点的负梯度* df1=float(d1.evalf(subs ={'x1':x0[0],'x2':x0[1],'x3':x0[2]}))  
 df2=float(d2.evalf(subs ={'x1':x0[0],'x2':x0[1],'x3':x0[2]}))  
 df3=float(d3.evalf(subs ={'x1':x0[0],'x2':x0[1],'x3':x0[2]}))  
 Hession\_k=Hession.evalf(subs ={'x1':x0[0],'x2':x0[1],'x3':x0[2]})  
 Hession\_k = np.array(Hession\_k).astype(float)  
  
 g\_k=np.array([df1,df2,df3])  
 *#梯度的二范数* magnitude=g\_k.dot(g\_k)\*\*0.5  
 if(magnitude<=1e-6 or k>100):*#满足精度要求，返回点x,退出循环* break  
 Beta=1  
 if k!=1:  
 a = d\_k.dot(Hession\_k).dot(g\_k.T)  
 b= d\_k.dot(Hession\_k).dot(d\_k.T)  
 Beta=a/b  
 d\_k=-g\_k+Beta\*d\_k  
 else:  
 d\_k=-g\_k  
  
 a=g\_k.dot(d\_k)  
 b=d\_k.dot(Hession\_k)  
 b=b.dot(d\_k.T)  
 lamb=(-a/b)  
 values = [k, x0, g\_k,d\_k, Beta, lamb]  
 for value in values:  
 if isinstance(value, float):  
 print("{:<18.2f}".format(value),end=' ')  
 else:  
 print('%-18s' % value, end=" ")  
 print()  
  
 x0 =x0+lamb \* d\_k  
 k+=1  
 return x0  
A=np.array([[1,0,0],[0,0.5,0],[0,0,0.5]])  
x0=np.array([1.0,1.0,1.0])  
print("最优点:",fast(A,x0))

# 4运行截图

实例1：，初始点



实例2：，初始点，



# 5总结与分析

通过本次实验，我用python实现了共轭梯度法。通过使用不同的函数测试，取得很好的效果。最速下降法内存消耗小，收敛速度快，具有自动搜索的特性。其本身存在一定的缺陷：对非二次型问题的适应性较差，共轭梯度法在处理非二次型优化问题时，可能会出现收敛速度慢或者收敛到局部最优解的情况。对非光滑函数的处理能力有限，因此在进行凸优化分析时，应当兼顾其缺点，防止造成太大的误差，得到错误结果。