



计算机与软件工程学院

上机报告

**（ 2021/2022 学年 第 1 学期 ）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | **数值计算（C++）** | | | | | |
| 课程代码 | **190901319** | | | | | |
| 上机时间 | 2021 | 年 | 11 | 月 | 20 | 日 |
| 指导单位 | 物联网工程系 | | | | | |
| 任课教师 | 李显勇 | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 | 李子涵 | | |
| 学 号 | 3120190971401 | | |
| 成 绩 |  | 年级专业 | 计算机科学与技术 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | **【实验4】Dolittle分解法和Crout分解法** | | | **实验地点** | 8-314 |
| **实验类型** | **验证** | **实验学时** | **2** | **实验日期** | 2021.11.20 |
| **实验目的和要求**   1. 了解**Dolittle分解法和Crout分解法**基本理论和算法； 2. 提交以下文档（用**学号+姓名**作为文件夹名，将所有内容放于该文件夹中）：   1）实验报告  2）程序代码 | | | | | |
| **实验环境(实验设备)**  windows XP及以上版本；PC；Matlab7.0及以上版本；其它高级语言 | | | | | |
| **实验原理及内容**  **一、实验原理**  根据**求解线性方程组的Dolittle分解法和Crout分解法等**相关知识和算法编程完成本实验  **二、实验内容**  已知线性方程组     1. 用Doolitle分解法求解线性方程组的解； 2. 用Crout分解法求解线性方程组的解。   **三、实验过程（可以文字说明+运行结果截图）**  '''  文件名：西华大学数值计算C++实验报告——实验四  内容：求解线性方程组的Dolittle分解法和Crout分解法  当前版本：1.0  完成作者：李子涵  学号：3120190971401  完成日期：2021.11.20  '''  import numpy as np  from numpy.core.fromnumeric import transpose  import math  from numpy.matrixlib.defmatrix import matrix  def Doolittle(coef,b):      """      用Doolittle求解方程式(系数矩阵为方阵)      """      def add\_1(i,j):          """          传入第i行第j列，求解由r=0到i-1求和的值：l\_kr\*u\_rk          """          sum=0          for r in range(i):              sum+=matrix\_L[i,r]\*matrix\_U[r,j]          return sum      def add\_2(i):          """          求和          """          sum=0          for j in range(i):              sum+=matrix\_L[i,j]\*y[j]          return sum      def add\_3(i):          """          求解求和从j=i+1到shape :u\_ij\*x\_j          """          sum=0          for j in range(i,shape):              sum+=matrix\_U[i,j]\*x[j]          return sum        matrix\_a=coef#系数矩阵      matrix\_U=np.zeros(matrix\_a.shape)#U矩阵      matrix\_L=np.eye(matrix\_a.shape[0])#L矩阵      if(matrix\_a.shape[0]!=matrix\_a.shape[1]):          print("非方阵")          return#若不为方阵打印信息并退出      shape=matrix\_a.shape[0]#shape为方阵的行（列）数        for i in range(shape):          #求解U中第i行          for j in range(i,shape):              matrix\_U[i,j]=matrix\_a[i,j]-add\_1(i,j)          #求解L中第i列          for k in range(i+1,shape):              matrix\_L[k,i]=(matrix\_a[k,i]-add\_1(i,j))/matrix\_U[i,i]        #求解方程组LY=b      y=[0]\*shape#创建y初始值为0      for i in range(shape):          y[i]=b[i]-add\_2(i)#第二个求和      #求解方程组UX=Yx      x=[0]\*shape#创建x初始值为0      for i in range(shape-1,-1,-1):#range最后一个参数表示倒着取值          x[i]=(y[i]-add\_3(i))/matrix\_U[i,i]      print("Doolittle求解方程答案为：")      print(x)  def Crout(coef,b):      """      用Crout法求解线性方程组（系数矩阵为方阵）      """      def add\_1(i,k):          """          计算r=0到k-1对L\_ik\*U\_rk求和          """          sum=0          for r in range(k):              sum+=matrix\_L[i,r]\*matrix\_U[r,k]          return sum      def add\_2(j,k):          """          计算r=0到k-1对L\_kr\*U\_rj求和          """          sum=0          for r in range(k):              sum+=matrix\_L[k,r]\*matrix\_U[r,j]          return sum      def add\_3(i):          """          求和          """          sum=0          for j in range(i+1,shape):              sum+=matrix\_U[i,j]\*x[j]          return sum      def add\_4(i):          """          求解求和从j=i+1到shape :u\_ij\*x\_j          """          sum=0          for j in range(i):              sum+=matrix\_L[i,j]\*y[j]          return sum        matrix\_a=coef#系数矩阵      matrix\_U=np.eye(matrix\_a.shape[0])#U矩阵(单位上三角矩阵)      matrix\_L=np.zeros(matrix\_a.shape)#L矩阵(下三角)      if(matrix\_a.shape[0]!=matrix\_a.shape[1]):          print("非方阵")          return#若不为方阵打印信息并退出      shape=matrix\_a.shape[0]#shape为方阵的行（列）数        for k in range(shape):#计算L与U          for i in range(k,shape):#先算L中第K列              matrix\_L[i,k]=matrix\_a[i,k]-add\_1(i,k)          for j in range(k+1,shape):#再算U中第K行              matrix\_U[k,j]=(matrix\_a[k,j]-add\_2(j,k))/matrix\_L[k,k]        #求解方程组LY=b      y=[0]\*shape#创建y初始值为0      for i in range(shape):          y[i]=(b[i]-add\_4(i))/matrix\_L[i,i]      #求解方程组UX=Yx      x=[0]\*shape#创建x初始值为0      for i in range(shape-1,-1,-1):#range最后一个参数表示倒着取值          x[i]=y[i]-add\_3(i)      print("Crout求解方程答案为：")      print(x)  if \_\_name\_\_=='\_\_main\_\_':      #给出系数矩阵      coef=np.matrix([(2,1,1),(1,3,2),(1,2,2)])      y=[4,6,5]      Doolittle(coef,y)      Crout(coef,y)  实验结果： | | | | | |

|  |
| --- |
| **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  两种解决方法的差别虽然很小，但是计算顺序的不同使编程的代码需要改变很多，尤其是求和运算的代码。此次实验所用的代码大多是循环，容易出错且不好调试，健壮性也不强，后续我应该继续从这些方面完善改进代码。 |