Fotran95/2003 科学计算与工程

（上）

宋叶志　编著

清华大学出版社

北京

内 容 简 介

科学计算方法是许多科研工作得以展开的前提。本书较为详细地介绍了科学计算与工程中的常用数值方法。全书以 Fortran95/2003 语言编写而成，全部程序在 Visual Studio 2008 集成 Intel 编译器环境下调试通过。

全书包括 12 章和 3 个附录。主要内容包括矩阵分解与线性方程组的直接方法、线性方程组的迭代方法、最小二乘法与数据拟合、特征值及特征向量、非线性方程求根、非线性方程组数值解法、插值法、数值微分、数值积分、常见的特殊函数计算、常微分方程（组）的数值方法及应用范例。

本书适合作为大学理工科非数学专业本科生或研究生计算方法、数值分析课程的教材或参考书。因为提供了全部的源代码，对于从事数值分析教学的教师也是一份难得的工具书。本书还可作为科研与工程技术人员的参考工具书。

图书在版编目（CIP）数据

Fotran95/2003 科学计算与工程/宋叶志编著.－北京：清华大学出版社，2010.09

ISBN 978-7-302-0
I. ①F… II. ①宋…②… III. ①－－－IV.①TP00
中国版本图书馆 CIP 数据核字（2010）第 000000 号

责任编辑：夏非彼　夏毓彦
责任校对：闫秀华
责任印制：
出版发行：清华大学出版社　　地　址：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　邮　编：100084
社 总 机：010-62770175　　邮　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：190×260　　印　张：33.75　　字　数：864 千字
版　次：　　印　次：
印　数：1～4000
定　价：

产品编号：

前　　言

科学计算是现代自然科学研究与工程技术的一个基石。可以认为科学计算方法属应用数学的一个分支，然而又不限于数学本身，其内容具有广泛性和一般性，在众多科学中都需要有科学计算方法的支持，而现代科学技术的进步及应用需求又反过来促进科学计算理论与方法的发展。可以这样说，就现今而言，科学计算能力的水平在一定程度上反应了一个国家或地区科学技术的发展水平。

抛开计算机硬件技术的发展不谈，就科学计算领域而言，曾经出现了较多计算机语言如 Fortran、Pascal、C、C++、Basic，及专门的数学软件 MATLAB、Mathimatica、Maple、SciLab 等。对于科学计算而言，建议读者优先学习 Fortran 及 MATLAB 语言，Fortran 在科学计算领域庞大的用户基础是其他语言所无法比拟的，尤其是对于老一辈科学家，对他们而言 Fortran 更是犹如母语一般。就作者经历而言，MATLAB 数值功能虽强大，但一般多用来仿真、模拟与快速分析。对于大型程序，主要是以 Fortran 居多，如数值气象预报程序、航天器轨道计算、许多有限元软件的编写程序等，这些大型程序一般都涉及非常复杂的科学计算方法。近些年，我国大范围普及 C、C++、Java、C# 等语言，造成介绍 Fortran 及其科学计算方面的书籍较少。

目前，我国大学理工科本科生、研究生及博士生普遍开设数值分析、计算方法或科学计算与工程等课程（计算数学系一般代之以函数逼近、数值代数等课程）。而这些教材中，往往没有提供让学生可以参考的程序，这对刚学习完高等数学与线性代数的同学学习科学计算方法造成了一定的困难。对于刚学习完高等数学与代数同学，往往一开始学习科学计算方法是较为困难的，这困难主要来源于两个方面，一是使用数学工具的手段还不是很成熟，二是刚开始面对数值问题编程时感到不习惯。鉴于此，本书提供了常用数值方法的算法及完整代码。

当然本书并不是仅仅为初学者准备的，有经验的编程用户，对于某些问题也可以直接使用书中的程序或者稍作改动即可使用书中的程序。

内容安排

数值分析本身的内容，如果广义地讲，可以包含很多内容，如偏微分方程、数值最优化等，而这些内容往往都可以单独成为一门学科或专题。通常的共识是数值算法所涉及的范围是直至偏微分方程但不包含偏微分方程的内容，而本书即主要介绍这部分内容。除了本书引子之外，全书共 12 章，其中前 11 章介绍计算方法的一般性内容，最后一章作为范例，仅仅起到示范性作用。

关于章节的安排，我们做了仔细的考虑，按照现在的顺序，保证了阅读时章节之间的依赖性不会对读者造成障碍，又使本书主要按照数值代数和函数逼近的两大知识块的顺序讲解。而如果先讲函数逼近的话，里面部分内容（如样条插值等），需要用到数值代数的知识。

在前 3 章分别介绍了线性方程组的直接方法、迭代方法、最小二次与数据拟合，其中最小二乘部分有些内容很自然地需要线性方程组的知识做基础。第四章特征值问题，不需要依赖于其他章节，因为其正交变换部分也可以放在该章单独讲解，因此可以放在全书的任何地方，但考虑到数值代数部分的整体性，把它放在了第 4 章。

第 5 章非线性方程的解法不需要依赖其他章节的内容，而第 6 章的非线性方程组的数值方法

则需要读者最好理解第五章的内容，同时计算非线性方程组必须具备线性方程组的知识，因此我们把第六章安排在线性方程组与非线性方程之后介绍。因为非线性方程组的应用非常广泛（如非线性回归等领域），我们介绍了较多的并且实用的数值方法，尤其是给出了多种变形的拟牛顿方法，如Broyden、DFP、BFS 等，还介绍了拓展收敛域的数值延拓法及参数微分法，这些算法涉及一定的编程技巧，在同类的书籍中较少有如此详细介绍的。我们把非线性最小二乘问题，放到应用范例一章以卫星导航原理为实例介绍。

接下来分别介绍了插值法（第 7 章）、数值微分（第 8 章）、数值积分（第 9 章）、常见的特殊函数计算（第 10 章）及常微分方程数值方法（第 11 章）。其中插值法介绍了最重要及最基础的 Lagrange、Newton、Hermite 插值方法，并把重点放到了样条插值方法。其中 B 样条因其理论优美、实用性强应该得到重视，而国内许多教材中较少有介绍，给出程序的则更少。关于特殊函数的计算，我们介绍了几种典型的特殊函数，并未求全而罗列更多的计算方法。常微分方程问题介绍了单步法、一般多步法及预测校正方法（PECE），作为基础，我们认为够了，当然在实践中，可能根据具体的工作需求而需要构造更复杂的算法，但基本原理大抵如此。

根据以往读者的需求，他们希望安排一章内容列举一些范例，以说明科学计算方法是有“用武之地”的，实际上我认为这个艰巨的任务，应该交给读者自己。作者知识面有限，只能举一两例做示范性的说明。

为了给编程经验不是很丰富的初学者提供更多的有效帮助，我们特地安排了附录部分讲解了，集成开发环境下编程的最基础知识及程序调试方法。

致谢：

本书的编写得到了单位领导及动力中心各位老师的帮助和支持，尤其感谢空间飞行器精密定轨课题组首席科学家胡小工研究员对我的帮助。在书的早期编写中，还得到了陈国平同志及北京航天飞控中心曹建峰博士的帮助。在编写过程中好友徐导，杨建平夫妇、复旦大学数学学院杨卫红副教授及陈清女士、王婷婷夫妇、Tiffany、Phebe Gray 博士等也给予作者许多鼓励。北京图格新知公司夏非彼老师为本书的出版付出了辛勤的劳动，作者致以诚挚的谢意。

囿于作者水平有限，加上时间仓促，书中肯定有不少疏漏，甚至谬误之处，还望读者不吝指教。作者的电子邮箱为：song.yz@163.com，也可以通过 QQ：1556207030 进行答疑。

如果有院校教师愿意采用本书作为教材或指定参考书的，可以和清华大学出版社联系或直接与作者联系，编者愿意提供 Fortran 程序设计或计算方法一定的技术支持。

宋叶志　中国科学院

茅永兴　中国卫星海上测控部

引　子

1. 科学计算方法有何应用？

至于科学计算法有何应用，这显然不是三言两语可以回答的。这里先对科学计算方法做一通俗介绍。在不严格的情况下，有时我们不去区分计算方法、科学计算、数值分析几个名词的差别。目前，我国理工科院校的计算数学、计算机等专业在本科生就开设了数值分析课程。而绝大部分院校理工科研究生和博士生都开设了计算方法的课程。科学计算对于个人而言，也是从事科研工作或工程技术的一项基本技能。

可以说从现在的航天器飞行、自动控制、大型桥梁设计等都离不开科学计算技术的支持。如果没有卫星飞行我们又如何能看上卫星电视？如何能使用卫星定位？可以说身边的许多事物都在直接或者间接的与科学计算有联系。现在许多社会学领域也离不开科学计算，比如医学统计方法、传染病动力学、证券金融时间序列分析等都需要依赖一定的数值方法实现。

当今社会科技日益发展，而许多高新技术都离不开科学计算的支持。上世纪中叶以来，计算机软硬件的发展为科学计算提供了有利条件。目前，各国科技领域都普遍非常重视科学计算理论与技术的发展。

科学计算具有一般性的理论意义，并不局限于某具体工程技术，而许多具体的科学实践领域反过来对进科学计算的发展提出了新的课题和需求，从而两者互相促进。

2. 该选择哪种编程语言？

现今，在使用的计算机高级语言种类繁多。对于编程经验不是很多的人而言，一个很自然的问题是：“我该学习哪种编程语言?”

我们给出直接的回答是，没有统一的答案。具体选择哪种语言是因人而异，因需求而异同，没必要厚此薄彼。

一般而言每种语言都有某一些领域比较突出。比如 C 语言编写操作系统很出名，现在的 C#编写界面非常方便，SQL 语言编写数据库程序，R 语言、SAS 编写统计程序等。

对于有意从事科研或工程技术的的读者，建议优先学习 Fortran 和 MATLAB，最好两种语言都能比较熟练的掌握。

Fortran 从诞生就一直是科学计算领域最重要的语言之一，拥有庞大的用户群体。第一版的 Fortran 是由 IBM 在上世纪 50 年代为自己的 704 计算机开发的。在那之前，基本所有的计算机程序都是通过手工编写的机器语言，非常容易出错。Fortran 的出现可以说是科学计算领域的一次革命，从此程序员能像写数学公式似的编写程序。

到目前为止 Fotran 已经经过数次较大的修改，不过由于良好的兼容性，一般不太会给程序员带来较大的麻烦。现在使用较多的是 Fortran90 之后的版本，包括 95 版和 2003 版。但是 Fortran77 依然有相当多的人在使用。目前，国内外众多科研机构的许多大型程序都是用 Fortran 编写的。

3.计算方法是否局限于编程实现？

一个简洁的回答：当然不是。

计算方法更重要之处，在于提供一整套能够武装人的大脑思维的分析理论与方法，使之可以

灵活运用于实践问题。而如果购买计算方法的书籍，仅仅是为了获得一套可执行的代码，这只实现了作者编写本书目的的一部分价值。为了表明算法的重要性，下面举一个简单的例子。

【问题】仅仅给你一张纸和笔，让你计算 $\sqrt{447}$ ，要求精确到小数点后 4 位有效数字。当然，如果你需要还可以提供给你只具有加、减、乘、除功能的小计算器。读者可以先不看解答，考虑一下这个问题，如果你仅仅有笔和纸，不一定要求你算出来，但是你要给出一个方法，要求是按照这个方法原则上是你可以算出来。

【解答】这是作者以前在某高校 BBS 上看到的一个问题，然后我给了提问者一个公式，即

$$x_1 = \frac{1}{2}\left(x_0 + \frac{447}{x_0}\right)$$

并给他介绍了公式使用方法，方法如下：

给定一个初始的你认为最接近的“猜想”值 x_0，然后把这个猜想的值带入等式右边，算出来 x_1。然后再把 x_1 当作 x_0，带入等式右边，算出新的 x_1，如此反复循环。可以把最后一次的 x_1，当作最终结果，即可以得到很高的精度。上面的公式加使用方法，已经构成了完整的计算方法。

实际上，可以把上述公式等价于下述迭代格式，即

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{447}{x_n}\right), n = 1, 2, \ldots$$

下面介绍一下以上公式的使用过程。

步骤一：$\sqrt{447}$ 等于多少，你不清楚。但是 $21^2=441$ 你应该知道。这样你就令 $x_1=21$。

步骤二：由上面的公式，可以算出

$$x_2 = \frac{1}{2}\left(21 + \frac{441}{21}\right) = 21.142857142857142$$

步骤三：再把 $x_2 = 21.142857142857142$，带入到公式中，可以求得

$$x_3 = \frac{1}{2}\left(x_2 + \frac{441}{x_2}\right) = 21.142374517374517$$

这时，可以就把 x_3 作为最后结果。$\sqrt{447}$ 的准确结果 21.142374511865974…。我们才作 2 次复合的四则运算，精度已经达到小数点后 8 位有效数字。

在以上计算过程中，仅仅是利用了加、减、乘、除法运算，并未采用更“高级”的运算方法。

有人疑问说：“我没想起来 $21^2=221$”。如此的话，$20^2=400$，这个总应该清楚的吧。在上面的公式中，就令 $x_1=20$。

$$x_2 = \left(20 + \frac{447}{20}\right) = 21.175$$

$$x_3 = \frac{1}{2}\left(x_2 + \frac{447}{x_2}\right) = 21.142399645808737$$

$$x_4 = \frac{1}{2}\left(x_3 + \frac{447}{x_3}\right) = 21.142374511880917$$

$$x_5 = \frac{1}{2}\left(x_4 + \frac{447}{x_4}\right) = 21.142374511865974$$

$$\vdots$$

可见，我们做了几次复合四则运算之后，精度已经达到 10 几位有效数字，远远超过我们的预期精度需求。

至于以上公式是怎么来的，这正是学习数值方法之后应该具备的最基本能力，说出来最简单不过了。既然是求 447 开根号的问题，就可以建立 x^2=447，这样一个方程（负根舍弃），剩下的工作，就是非线性方程一章中的内容了。

以上只是一个简单的范例，事实上关于数值算法的很多内容，早在计算机发明之前就已经被科学家所发现并运用了，从许多算法的名字也可以看出，牛顿迭代法、拉格朗日插值法、高斯积分法等等。高斯在曾就用手算实现最小二乘法而达到参数估计的目的。这些先贤，即便是没有数字计算机，依然是第一流的数值分析专家。

在数字计算机发明之前对于非线性微分方程有摄动分析解法，同样也有人用手工的方法进行微分方程的数值计算，当然这样的代价是相当高昂的。总之，算法是数值方法的灵魂，而编程仅是一个实现算法的过程。

这里没有意图要说明计算机编程不重要，而是同时强调编程方法和算法理论都是很重要的。

4. 学习计算方法需要准备哪些基础知识？

本书读者的起点是具备微积分（数学分析）、线性代数、矩阵论和概率统计部分知识、泛函分析少量知识。泛函分析的知识不是必须的，适当的时候我们会交代一下，当然如果是用泛函分析的语言去叙述，很多问题会更简洁明了，并更具备概括性。总之，要想学习好科学计算方法，需要对微积分、矩阵等知识比较熟悉。另外一点是，假定读者已经掌握了编程方法。

编写本书目的之一，是考虑到许多初学者，因为没有足够的数值程序范例，入门就很困难，更无从谈提高了。对于没有入门的读者，可以先拷贝光盘上的程序，在自己计算机上试运行，然后修改相关参数等看计算结果，逐步入门。当然这并不是建议大家都不需要编写程序，而直接拿书中的程序去应付自己的工作需求。要想真正学好科学计算方法，还是需要自己动手编写一定量的代码。这样才能对数学原理又更深刻的认识和体会。

5.书中程序是否有 Bug？

当然有 Bug，这是毫无疑问的。

Bug 本意是臭虫、缺陷、损坏等意思。现在在较大的计算机程序中，经常会隐藏着的一些可能是未被发现的缺陷、问题或漏洞统称为 Bug。早期的计算机是由许多庞大且昂贵的真空管组成，并利用大量的电力来使真空管发光。一次科研试验中，一只小虫子 Bug　钻进了一支真空管内，导致整个计算机无法工作。研究人员费了相当长时间，总算发现原因所在，把这只小虫子从真空管中取出后，计算机又恢复正常。后来，Bug 这个名词就沿用下来，表示电脑系统或程序中隐藏的错误、缺陷、漏洞或问题。

与之相对应，人们将发现 Bug 并加以纠正的过程叫做 Debug，有时候我们就称为调试，意思

是捉虫子、杀虫子。

一般而言，程序员并不试图一次性把程序写的完全正确，而是通过反复的“Debug”排除错误。关于如何 Debug 是一个技巧性很强的工作，也是编程所应该必须具备的技能，在附录部分，我们介绍了常见的 Debug 方法。

回到本段的问题，虽然我们对代码做了多次编译、测试，但是我相信里面一定还有不少 Bug。即便是目前世界上最优秀的数值方法库，他们也不敢说自己的程序没有 Bug。像 MATLAB 这样专业成熟的商业软件，现在每年发布两次软件中存在的 Bug。所以，虽然我们的程序读者可以直接使用，但是我们仍然希望读者了解计算方法。

在这一点上，许多人喜欢以“黑箱子”来形容这件事，即用户并不需要知道“黑箱子”是如何实现的，仅仅想知道进去什么，可以出来什么。我们的观点是，我们提供一套我们自己编写的“黑箱子”，声明一点，我们做的可能不一定太好，但是我们试图把“黑箱子”打开，让读者自己从中寻找需要的东西。

鉴于以上的考虑，在编程时，我们对代码的注释是比较详细的，这为读者理解程序提供了方便。

6. 什么是面向对象？如何使用书中的程序？

我们的程序主要以 Fortran90/95/2003（以下简称 F90/95/2003）编写，对于 F2003 我们持保留的态度，不过多的涉及其中的语法，因为其中的一些语法在部分编译器上还不支持。

如果是 Fortran 77（以下简称 F77）的用户，使用部分程序可能需要做较大的改动。F90/95 在 F77 一些容易犯错的地方，虽然允许使用，但是不建议使用，而我们编写时则回避了新版本建议舍弃的语法。另外新版本有一些新的语法，在老版本上是不支持的，如新版大大加强了数组（某种意义下相当于矩阵）的处理方式。

全书采用模块化方法编写，这里的模块化不仅仅是一种语法功能，且可以认为是一种面向对象的思想。F2003 则支持了更为丰富的面向对象功能，如前面所言，我们对这些语法持保留态度，不一味的求新。

面向对象的好处之一是可以进行数据封装，我们考虑到大部分用户编程习惯仍然是结构化程序设计方法，所以我们尽量只是采用面向对象的形式，而实际上我们的过程都是可以单独拿出来使用。当然，少部分程序，我们作为范例把变量放在模块中，如果有必要的话只需要把这些变量复制到对应的函数中即可。有些章节的程序，可能需要依赖其他章节的介绍的程序，我们把所依赖的函数拷贝到当前章节的程序中，这样做的一个坏处是增加了少量的篇幅，好处是保证每一章节的独立性与完整性。

因为目前国内大部分用户使用的系统依然是 Windows，我们主要的编程环境是采用 Visual studio2008+Intel 11 编译器。目前 Visual Studio 2010 还不支持 Intel 编译器的集成。当然我们的大部分程序在 Compaq Fortran 6.6 这个经典 IDE 上也能编译通过，只是有些少量语法 Compaq Fortran 还不支持，需要略做修改，在适当的地方我们做了介绍。部分程序在 Sun 公司的 Solaris Unix F90 编译器编译测试过。需要说明的是集成开发环境本身对 Fortran 语言是没有影响的，但是一个好的集成开发环境为程序员提供非常高效的工程条件。

目　录

第 1 章

◂矩阵分解与线性方程组的直接方法▸

自然现象大部分表现为非线性，而很多情况下，我们对于非线性问题是通过线性化来处理，故而在众多学科中，线性问题依然是最基础最重要的内容之一。本章将介绍线性方程组的数值方法，线性方程组的数值方法是数值计算的一个基础内容，在非线性方程组、微分方程边值问题、样条逼近等许多领域中，都需要用到线性方程组的处理方法。

线性方程组主要分为直接方法与迭代方法。本章介绍直接方法，同时介绍与之相应的常见的矩阵分解方法，关于迭代法将在下一章中介绍。

1.1　三角方程组

1. 实验基本原理

线性方程组的计算方法有多种，针对不同特点的方程采用不同的方法计算效率往往是不同的。线性方程组中最简单的是对角形方程与三角形方程，其中对角形方程几乎都不需要计算机，只要拿笔和纸进行简单的四则运算，就可以得出结果。三角形方程也比较简单，仅仅是一个回带的过程，即便是没有学过数值分析的读者，只要略具备代数知识即可自行给出算法。

然而很多解方程的方法都是先把一般性方程化成三角形方程，这样便可以方便地处理。因此，计算三角形方程便是数值线性代数的一个较简单而又基础的问题。

三角形方程分为上三角形和下三角形两种，其计算方法基本相同，仅仅是次序不同而已。

对于上三角形方程，计算步骤为：

$$\begin{cases} x_n = \dfrac{b_n}{a_{nn}} \\ x_i = \dfrac{b_i - \sum\limits_{k=i+1}^{n} a_{ik} x_k}{a_{ii}}, i = n-1, n-2, \cdots, 1 \end{cases}$$

对于下三角形方程，计算步骤为：

$$\begin{cases} x_1 = \dfrac{b_1}{a_{11}} \\ x_k = \dfrac{b_k - \sum\limits_{i=1}^{k-1} a_{ki} x_i}{a_{ii}}, \quad k = 2, \cdots, N \end{cases}$$

下面通过实验介绍上、下三角形矩阵的计算。

2. 实验目的与要求

- 理解上、下三角形方程组的计算流程。
- 能够编程实现三角形方程组的计算。

3. 实验内容与数据来源

计算上三角方程组

$$\begin{pmatrix} 2 & 1 & 3 & 4 \\ 0 & 3 & 2 & 5 \\ 0 & 0 & 7 & 3 \\ 0 & 0 & 0 & 2 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 50 \\ 49 \\ 53 \\ 12 \end{pmatrix}$$

计算下三角方程组

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 5 & 1 & 0 \\ 3 & 2 & 1 & 4 \end{pmatrix} \mathbf{y} = \begin{pmatrix} 10 \\ 23 \\ 39 \\ 43 \end{pmatrix}$$

4. 函数调用接口说明

uptri

输出参变量	数据类型	变量说明
X	REAL*8(N)	上三角方程的解
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	上三角系数矩阵
B	REAL*8(N)	右向量
N	INTEGER	方程组的维数

downtri

输出参变量	数据类型	变量说明
X	REAL*8(N)	下三角方程的解

输入参变量	数据类型	变量说明
A	REAL*8(N,N)	下三角系数矩阵
B	REAL*8(N)	右向量
N	INTEGER	方程的维数

5. 程序代码

```
module tri eq
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :  2010-4-8
!-----------------------------------------------------
!  Description : 用于解上、下三角形线性方程组的回带方法模块
!
!-----------------------------------------------------
!  Contains    :
!      1.   solve  方法函数
!      2.
!-----------------------------------------------------
contains
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,k,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
```

```
subroutine downtri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!----------------------------------------------------
!  Purpose   :  下三角方程组的回带方法
!               Ax=b
!----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!---------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(1)=b(1)/a(1,1)
do k=2,N
   x(k)=b(k)
   do i=1,k-1
      x(k)=x(k)-a(k,i)*x(i)
   end do
   x(k)=x(k)/a(k,k)
end do
end subroutine downtri
end module tri_eq
!###############################################################
module driver
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!----------------------------------------------------
!  Description :  驱动函数模块
!
!----------------------------------------------------
!  Parameters  :
!     1.
!     2.
!  Contains    :
!     1.       dir_main  驱动函数入口
!     2.       dri_up    当读到关键字 uptri 时启动该函数
!     3.       dri_down  当读到关键字 downtri 时启动该函数
!----------------------------------------------------
!  Post Script :
!     1.
!     2.
!----------------------------------------------------
contains
```

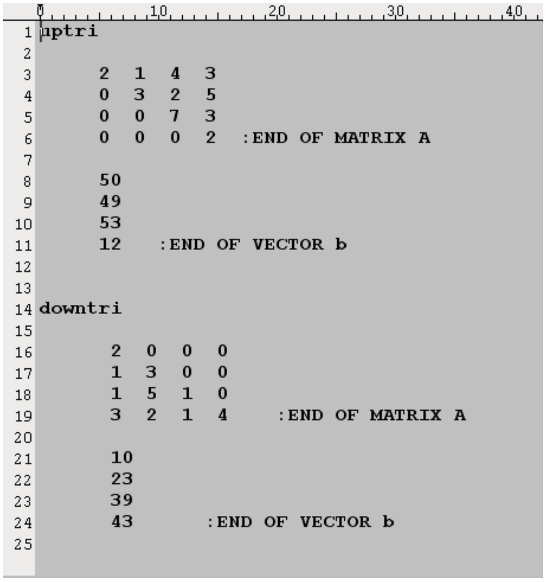
```
subroutine dri_main()
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!----------------------------------------------------
!  Purpose   :  驱动程序入口函数
!
!----------------------------------------------------
!  Input  files  :
!       1.   fin.txt  准备数据
!       2.
!  Output files  :
!       1.  fout.txt 输出结果文件
!       2.
!---------------------------------------------------
implicit real*8(a-z)
integer::ioerr
character(20)::upordown
open(unit=11,file='fin.txt',iostat=ioerr)
open(unit=12,file='fout.txt')
do
  read(11,*)upordown
  !  读输入文件
  !  当读到关键字 uptri 时启动上三角矩阵计算
  !  当读到关键字 downtri 时候启动下三角矩阵计算

  if ( upordown(1:5) == 'uptri' ) then
    call dri_up()
  else if (upordown(1:)=='downtri')then
    call dri_down()
  end if

  if (ioerr/=0) exit
  !读到文件结束时，退出读文件
end do
end subroutine dri_main
subroutine dri_up()
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!----------------------------------------------------
!  Purpose   : 启动上三角阵的计算
!
!----------------------------------------------------
use tri_eq
implicit real*8(a-z)
integer,parameter::N=4
integer::i,j
real*8::A(N,N),b(N),x(N)
read(11,*)((A(i,j),j=1,N),i=1,N)
!读入 B 向量
read(11,*) b
call uptri(A,b,x,N)
```

```
write(12,101)x
101 format(T5,'上三角形方程组的解',/,T4,'x=',4(/F12.8))
end subroutine dri_up
subroutine dri_down()
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!-----------------------------------------------------
!  Purpose   :  启动下三角阵的计算
!
!-----------------------------------------------------
use tri_eq
implicit real*8(a-z)
integer,parameter::N=4
integer::i,j
real*8::A(N,N),b(N),x(N)
read(11,*)((A(i,j),j=1,N),i=1,N)
!读入 B 向量
read(11,*) b
call downtri(A,b,x,N)
write(12,101)x
101 format(/,T5,'下三角形方程组的解',/,T4,'x=',4(/F12.8))
end subroutine dri_down
end module driver
!-----------------------------------------------------
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!-----------------------------------------------------
!  Purpose   :  计算上、下三角形方程组
!
!-----------------------------------------------------
!  In put data  files :
!      1.  fin.txt  输入方程系数
!      2.
!  Output data files  :
!      1. fout.txt  计算结果
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.    需要准备输入数据
!
!      2.    由主函数启动驱动程序进行计算
!-----------------------------------------------------
!use tri_eq
use driver
!调用驱动函数
call dri_main()
end program main
```

6. 实验结论

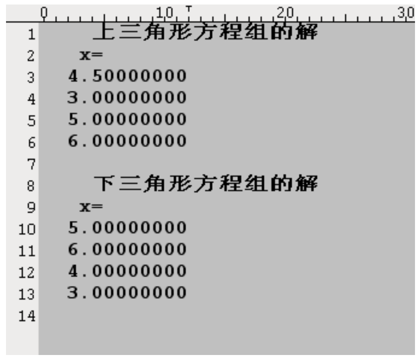
要使用该程序先要准备输入数据，输入数据格式如图 1-1 所示。

该文件需要以文件名 fin.txt 保存在可执行文件的当前目录下，执行程序后，输出结果以文件名 fout.txt 保存，如图 1-2 所示。



```
uptri

     2  1  4  3
     0  3  2  5
     0  0  7  3
     0  0  0  2   :END OF MATRIX A

     50
     49
     53
     12    :END OF VECTOR b


downtri

      2  0  0  0
      1  3  0  0
      1  5  1  0
      3  2  1  4     :END OF MATRIX A

      10
      23
      39
      43       :END OF VECTOR b

```

图 1-1　输入数据准备



```
    上三角形方程组的解
   x=
  4.50000000
  3.00000000
  5.00000000
  6.00000000

    下三角形方程组的解
   x=
  5.00000000
  6.00000000
  4.00000000
  3.00000000

```

图 1-2　三角形方程组计算结果

把计算结果回带到方程中，可以发现满足原方程。

为了方便读取文件，我们在文件中编写了驱动程序，虽然在小型的计算中似乎没有太大必要，但是建议读者养成用文件输入代替键盘屏幕输入的习惯，因为在大型的程序中，需要输入的量可能成百上千，如果使用键盘屏幕输入不仅效率低下，而且容易出错，使用文件输入只要把输入量按照一定的格式保存在文件中，然后由计算机读取效率会非常高。

1.2　高斯消去法

1. 实验基本原理

高斯消去法的思想很朴素，方法是通过对增广矩阵实施消元变换，而在变换的过程中与原方程保持等价，直到矩阵变为上三角矩阵，这时候可以采用上一节介绍的方法进行回代，便得到方程的解。

对于方程组 $\mathbf{Ax}=\mathbf{b}$，增广矩阵为$\begin{bmatrix}\mathbf{A} & \mathbf{b}\end{bmatrix}$。第一次消元，使方矩阵变为：

$$\begin{bmatrix}\mathbf{A}^{(2)} & \mathbf{b}^{(2)}\end{bmatrix}=\begin{pmatrix}a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)}\\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)}\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} & b_n^{(2)}\end{pmatrix}$$

在第 k 次消元过程中，令 $l_{ik}=\dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$，则第 i 行的消去公式为“

$$\begin{cases}a_{ij}^{(k+1)}=a_{ij}^{(k)}-l_{ik}a_{ij}^{(k)} & i,j=k+1,k+2,\ldots,n\\ b_i^{k+1}=b_i^{(k)}-l_{ik}b_i^{(k)} & i=k+1,k+2,\ldots,n\end{cases}$$

如此循环，直到矩阵变为：

$$\begin{bmatrix}\mathbf{A}^{(n)} & \mathbf{b}^{(n)}\end{bmatrix}=\begin{pmatrix}a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)}\\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} & b_2^{(2)}\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & \cdots & a_{nn}^{(n)} & b_n^{(n)}\end{pmatrix}$$

当变为上三角阵之后，可以采用回代公式：

$$\begin{cases}x_n=\dfrac{b_n}{a_{nn}}\\ x_i=\dfrac{b_i-\sum\limits_{k=i+1}^{n}a_{ik}x_k}{a_{ii}},i=n-1,n-2,\cdots,1\end{cases}$$

就可以给出方程的解。

2. 实验目的与要求

- 了解高斯消去法的大致思路。
- 明白算法的计算步骤。
- 能编程实现高斯消去法。
- 知道高斯消去法的不足之处。

3. 实验内容与数据来源

计算线性方程组 $\mathbf{Ax}=\mathbf{b}$，其中

$$\mathbf{A}=\begin{pmatrix}-13.9211 & 21.7540 & -14.8743 & -7.9025\\ 18.3862 & -26.0893 & -5.6866 & 4.4451\\ -4.1683 & 3.9325 & -33.3169 & 41.7098\\ -6.0438 & 6.7018 & -32.9591 & -23.3378\end{pmatrix},\mathbf{b}=\begin{pmatrix}136.8721\\ -126.8849\\ 100.4524\\ 95.7019\end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	方程的解
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	右向量
N	INTEGER	方程的维数

5. 程序代码

```
module gauss
!-----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-----------------------------------------------------
!  Description : 高斯消去法模块
!
!-----------------------------------------------------
!  Contains    :
!      1.   solve  方法函数
!      2.
!-----------------------------------------------------
contains
subroutine solve(A,b,x,N)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵 [Ab]
real*8::Ab(N,N+1)
```

```
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!-------------------------------
!  这段是高斯消去法的核心部分
do k=1,N-1
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!            | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!            | 0  0  *  *  # |
!            | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine solve
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module gauss
```

```
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯消去法
!
!-----------------------------------------------------
!  In put data  files :
!      1.  fin.txt  输入方程系数
!      2.
!  Output data files  :
!      1. fout.txt  计算结果
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.    需要准备输入数据
!
!-----------------------------------------------------
use gauss
implicit real*8(a-z)
integer,parameter:: N=4
integer::i,j
real*8::A(N,N),b(N),x(N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
!读入 A 矩阵
read(11,*)((A(i,j),j=1,N),i=1,N)
!读入 B 向量
read(11,*) b
call solve(A,b,x,N)
write(12,101)x
101 format(T5,'高斯消去法计算结果',/,T4,'x=',4(/F12.8))
end program main
```

6. 实验结论

要执行程序，需要准备数据文件，其文件名为 fin.txt。输入格式如图 1-3 所示。

```
                    高斯消去法输入数据

  -13.9211   21.7540  -14.8743   -7.9025
   18.3862  -26.0893   -5.6866    4.4451
   -4.1683    3.9325  -33.3169   41.7098
   -6.0438    6.7018  -32.9591  -23.3378   :End of matrix A

  136.8721
 -126.8849
  100.4524
   95.7019                    :End of matrix B
```

图 1-3　输入数据文件

执行程序后计算结果保存在文件 fout.txt 中，如图 1-4 所示。

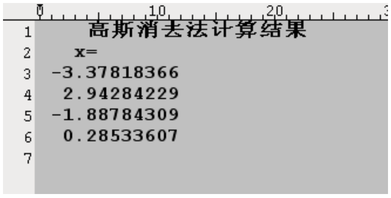

```
   高斯消去法计算结果
  x=
-3.37818366
 2.94284229
-1.88784309
 0.28533607
```

图 1-4　高斯消去法计算结果

把计算结果回代到原方程，可以验证计算结果是可靠的。

1.3　选主元消去法

1. 实验基本原理

上一节介绍的高斯消去法较为简单，然而如果在消去过程中出现 0 主元或者是主元非常小的话，消去法将失败或者数值不稳定。这时可以采用选主元的方法，进行处理。

下面给出列主元消去法的算法：

01 设置增广矩阵 $\mathbf{A}_p = [\mathbf{A}, \mathbf{b}]$。

02 对 k=1，N-1，做03~06处理。

03 设置一个元素 $a_{\max} = \left|\mathbf{A}_p(k,k)\right|$，及标号 $ID_{\max} = k$。

04 查找当列的最大元素，然后用标号 $ID_{\max}$ 记录下这个元素所在的行数。

05 交换第 k 行与第 $ID_{\max}$ 行的所有数据。注意其他元素不变。

06 完成05时已经完成了主元的选取，这时候可以按照上一节的消去法进行消去计算。

07 完成以上步骤之后，已经形成上三角矩阵。

08 对上三角矩阵进行回代，即可得到方程的解。

2. 实验目的与要求

- 复习高斯消去法。
- 了解选主元消去法的优点。
- 能够编程实现选主元消去法。

3. 实验内容与数据来源

用列主元消去法计算方程 $\mathbf{Ax} = \mathbf{b}$，其中

$$\mathbf{A} = \begin{pmatrix} -3.3435 & -0.4946 & 0.3834 & -4.9537 & -2.4013 & -3.5446 \\ 1.0198 & -4.1618 & 4.9613 & 2.7491 & 3.0007 & -3.6393 \\ -2.3703 & -2.7102 & -4.2182 & 3.1730 & -0.6859 & 3.6929 \\ 1.5408 & 4.1334 & -0.5732 & 3.6869 & 4.1065 & 0.7970 \\ 1.8921 & -3.4762 & -3.9335 & -4.1556 & -3.1815 & 0.4986 \\ 2.4815 & 3.2582 & 4.6190 & -1.0022 & -2.3620 & -3.5505 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 9.0537 \\ 0.0228 \\ -8.4177 \\ -4.6380 \\ 10.5575 \\ 9.8252 \end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	方程的解
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	右向量
N	INTEGER	方程维数

5. 程序代码

```
  module m_gauss
!---------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!----------------------------------------------------
!  Description : 高斯列主元消去法模块
!
!----------------------------------------------------
!  Contains    :
!      1.   solve  方法函数
!      2.
!----------------------------------------------------
contains
subroutine solve(A,b,x,N)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
```

```
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!############################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
   elmax=dabs(Ab(k,k))
   id_max=k
   !这段为查找主元素
   !这段程序的主要目的不是为了赋值最大元素给elmax，而是为了找出最大元素对应的标号
    do i=k+1,n
     if (dabs(Ab(i,k))>elmax) then
        elmax=Ab(i,k)
        id_max=i
     end if
   end do
!至此，已经完成查找最大元素，查找完成以后与 第k行交换
 !交换两行元素，其他不变
   vtemp1=Ab(k,:)
   vtemp2=Ab(id_max,:)
   Ab(k,:)=vtemp2
   Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!###########################################################
  do i=k+1,N
    temp=Ab(i,k)/Ab(k,k)
    Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
  end do
end do
!-----------------------------
! 经过上一步，Ab已经化为如下形式的矩阵
!           | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
```

```
!          | 0  0  *  *  # |
!          | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine solve
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1

   x(i)=b(i)
  do j=i+1,N
   x(i)=x(i)-a(i,j)*x(j)
  end do
   x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module m_gauss
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!
!-----------------------------------------------------
!  In put data  files :
!      1.  fin.txt  输入方程系数
!      2.
!  Output data files  :
```

```
!       1. fout.txt  计算结果
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.    需要准备输入数据
!
!-----------------------------------------------------
use m gauss
implicit real*8(a-z)
integer,parameter:: N=4
integer::i,j
real*8::A(N,N),b(N),x(N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
!读入 A 矩阵
read(11,*)((A(i,j),j=1,N),i=1,N)
!读入 B 向量
read(11,*) b
call solve(A,b,x,N)
write(12,101)x
101 format(T5,'高斯列主元消去法计算结果',/,T4,'x=',4(/F12.8))
end program main
```

6. 实验结论

程序运行需要准备文件名为 fin.txt 的输入文件，输入格式如图 1-5 所示。

```
                                列主元消去法

  -3.3435    -0.4946     0.3834    -4.9537    -2.4013    -3.5446
   1.0198    -4.1618     4.9613     2.7491     3.0007    -3.6393
  -2.3703    -2.7102    -4.2182     3.1730    -0.6859     3.6929
   1.5408     4.1334    -0.5732     3.6869     4.1065     0.7970
   1.8921    -3.4762    -3.9335    -4.1556    -3.1815     0.4986
   2.4815     3.2582     4.6190    -1.0022    -2.3620    -3.5505

   9.0537
   0.0228
  -8.4177
  -4.6380
  10.5575
   9.8252
```

图 1-5　数据输入文件

运行结果保存在文件 fout.txt 文件中，结果如图 1-6 所示。

```
    高斯列主元消去法计算结果
   x=
  1.26515941
  0.11026391
 -1.24520947
 -0.43379925
 -0.99093338
 -2.62011810
```

图 1-6　列主元消去法计算结果

同样可以把计算结果带入原方程进行检验。

1.4 Crout分解

1. 实验基本原理

如果一个方阵 **A** 可以分解为一个下三角矩阵 **L** 与一个上三角矩阵 **U** 的乘积，这种分解称为方阵 **A** 的三角分解或 LU 分解。特别的情况是，如果 **U** 为单位上三角矩阵时，称为 Crout 分解，如果 **L** 为单位下三角矩阵时称为 Doolittle 分解，Doolittle 分解将在下一节介绍。

下面给出 Crout 分解的算法：

01 计算 LU 分解中的 L 的第一列和 U 的第一行元素

$$\begin{cases} l_{i1} = a_{i1}, i = 1, 2, \ldots, n \\ u_{1j} = \dfrac{a_{1j}}{l_{11}}, j = 2, \ldots, n \quad (u_{11} = 1) \end{cases}$$

02 对 $k = 2, \ldots, n$ 计算 L 的第 k 列元素

$$l_{ik} = a_{ik} - \sum_{r=1}^{k-1} l_{ir} u_{rk}, i = k, \ldots, n$$

以及 U 的第 k 行元素

$$u_{kj} = \frac{a_{kj} - \sum_{r=1}^{k-1} l_{kr} u_{rj}}{l_{kk}}, i = k+1, \ldots, n$$

如此便完成了矩阵的 Crout 分解。

2. 实验目的与要求

- 了解 LU 分解的基本思想。
- 熟悉 Crout 分解的计算流程。
- 知道 Crout 分解与消去法的联系。
- 能编程实现对矩阵的 Crout 分解。

3. 实验内容与数据来源

已知矩阵

$$\mathbf{A}=\begin{pmatrix} 6 & 2 & 1 & -1 \\ 2 & 4 & 1 & 0 \\ 1 & 1 & 4 & -1 \\ -1 & 0 & -1 & 3 \end{pmatrix}$$

求矩阵 **A** 的 Crout 分解。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
L	REAL*8(N,N)	下三角矩阵
U	REAL*8(N,N)	单位上三角矩阵
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	输入矩阵
N	INTEGER	矩阵维数

5. 程序代码

```
module crout
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :  LU 分解模块
!
!-----------------------------------------------------
contains
subroutine solve(A,L,U,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  LU 之 Crout 分解
!              A=LU
!-----------------------------------------------------
!  Input  parameters  :
!      1.    A  方阵
!      2.    N  阶数
!  Output parameters  :
!      1.   L
!      2.   U
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
```

```
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::N,i,k,r
real*8::A(N,N),L(N,N),U(N,N)
!L 第一列
L(:,1)=a(:,1)
!U 第一行
U(1,:)=a(1,:)/L(1,1)
do k=2,N
   do i=k,n
       s=0
       do r=1,k-1
        s=s+l(i,r)*u(r,k)
       end do
       l(i,k)=a(i,k)-s
   end do
   do j=k+1,n
     s=0
     do r=1,k-1
      s=s+l(k,r)*u(r,j)
     end do
     u(k,j)=(a(k,j)-s)/l(k,k)

   end do
    u(k,k)=1
end do
end subroutine solve
end module crout
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :   Crot 分解
!
!-----------------------------------------------------
!  In put data  files :
!      1.    A,N
!      2.
!  Output data files  :
!      1.    L,U
!      2.
!-----------------------------------------------------
use crout
integer,parameter::N=4
real*8::A(n,n),L(N,N),U(N,N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
read(11,*)((A(i,j),j=1,N),i=1,N)
call solve(A,L,U,N)
write(12,21)
21 format(T10,'LU之Crout分解',/)
```

```
!输出 L 矩阵
write(12,*)'L='
do i=1,N
 write(12,22)L(i,:)
end do
22 format(4F10.6)
!输出 U 矩阵
write(12,*)'U='
do i=1,N
 write(12,22)U(i,:)
end do
23 format(4F10.6)
end program main
```

6. 实验结论

程序运行需要准备数据文件 fin.txt，输入格式如图 1-7 所示。

程序运行后，计算结果保存在文件 fout.txt 文件中，如图 1-8 所示。

```
      Crout  分解

  6   2   1   -1
  2   4   1    0
  1   1   4   -1
 -1   0  -1    3
```

图 1-7　Crout 分解的矩阵系数

```
          LU之Crout分解

L=
  6.000000  0.000000  0.000000  0.000000
  2.000000  3.333333  0.000000  0.000000
  1.000000  0.666667  3.700000  0.000000
 -1.000000  0.333333 -0.900000  2.581081
U=
  1.000000  0.333333  0.166667 -0.166667
  0.000000  1.000000  0.200000  0.100000
  0.000000  0.000000  1.000000 -0.243243
  0.000000  0.000000  0.000000  1.000000
```

图 1-8　Crout 分解

对计算结果可以进行矩阵相乘，看是否得到原先矩阵以检验结果是否有效。

1.5　Doolittle分解

1. 实验基本原理

另一种 LU 分解称为 Doolittle 分解，其 L、U 分解后的形式如下：

$$\mathbf{L}=\begin{pmatrix}1 & & & \\ l_{21} & 1 & & \\ \vdots & \cdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1\end{pmatrix},\mathbf{U}=\begin{pmatrix}u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn}\end{pmatrix}$$

计算步骤如下：

$$u_{1i}=a_{1i}, i=1,\cdots,n$$
$$l_{j1}=\frac{a_{j1}}{u_{11}}, j=1,\cdots,n$$

$$u_{kj}=a_{kj}-\sum_{m=1}^{k-1} l_{km}u_{mj},\quad j=k,k+1,\cdots,n$$
$$l_{ik}=\frac{a_{ik}-\sum_{m=1}^{k-1} l_{im}u_{mk}}{u_{kk}},\quad i=k+1,\cdots,n$$

通过 Doolittle 分解后也即把线性方程分为上三角与下三角型线性方程问题。

2. 实验目的与要求

- 知道 Doolittle 分解的基本思路。
- 熟悉计算流程。
- 能够编程实现矩阵的 Doolittle 分解。

3. 实验内容与数据来源

已知

$$\mathbf{A}=\begin{pmatrix} 6 & 3 & -8 \\ 15 & 5 & 2 \\ 2 & 2 & 7 \end{pmatrix}$$

要求对 **A** 矩阵进行 Doolittle 分解。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
L	REAL*8(N,N)	单位下三角矩阵
U	REAL*8(N,N)	上三角矩阵
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	输入矩阵
N	INTEGER	矩阵维数

5. 程序代码

```
module doolittle
!----------------------------------------module coment
!  Version     :  V1.0
```

```
!  Coded by    :  syz
!  Date       :
!-----------------------------------------------------
!  Description :  LU 分解之 doolittle 模块
!
!-----------------------------------------------------
contains
subroutine solve(A,L,U,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date     :
!-----------------------------------------------------
!  Purpose   :  LU 分解之 Doolittle 函数
!              A=LU
!-----------------------------------------------------
!  Input  parameters  :
!      1.    A  方阵
!      2.    N  阶数
!  Output parameters  :
!      1.   L
!      2.   U
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::N,i,k,r
real*8::A(N,N),L(N,N),U(N,N)
!U 的第一行
U(1,:)=A(1,:)
!L 的第一列
L(:,1)=a(:,1)/U(1,1)
do k=2,N
    l(k,k)=1
   do j=k,n
       s=0
       do m=1,k-1
        s=s+l(k,m)*u(m,j)
       end do
       u(k,j)=a(k,j)-s
   end do
   do i=k+1,n
     s=0
     do m=1,k-1
      s=s+l(i,m)*u(m,k)
     end do
     l(i,k)=(a(i,k)-s)/u(k,k)
   end do
end do
end subroutine solve
end module doolittle
```

```
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!----------------------------------------------------
!  Purpose   :  Doolittle 分解
!
!----------------------------------------------------
!  In put data  files :
!      1.    A,N
!      2.
!  Output data files  :
!      1.    L,U
!      2.
!----------------------------------------------------
use doolittle
integer,parameter::N=3
real*8::A(n,n),L(N,N),U(N,N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
read(11,*)((A(i,j),j=1,N),i=1,N)
call solve(A,L,U,N)
write(12,21)
21 format(T10,'LU之Doolittle分解',/)
!输出L矩阵
write(12,*)'L='
do i=1,N
 write(12,22)L(i,:)
end do
22 format(3F10.6)
!输出U矩阵
write(12,*)'U='
do i=1,N
 write(12,22)U(i,:)
end do
23 format(3F10.6)
end program main
```

6. 实验结论

程序运行需要准备数据文件让计算机读取，矩阵存放在文件 fin.txt 中，输入格式如图 1-9 所示。

计算结果存放在文件 fout.txt 文件中，如图 1-10 所示。

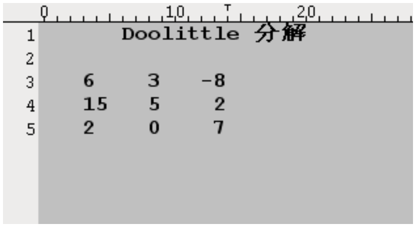


图 1-9 Doolittel 分解的输入数据

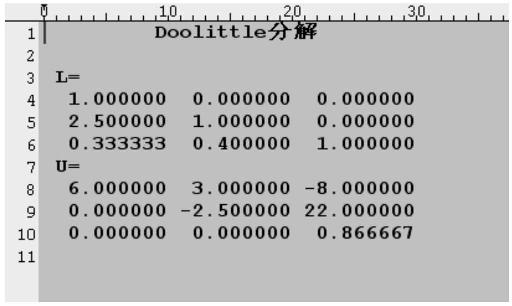


图 1-10 Doolittle 分解结果

可以看到分解后的矩阵满足 Doolittle 的形式要求，编程时可以通过两矩阵相乘验证程序是否正确。

1.6 LU分解法计算线性方程组

1. 实验基本原理

有了矩阵的 LU 分解继而求解方程是为水到渠成，这一小节即利用 Doolittle 分解计算线性方程组 $\mathbf{Ax}=\mathbf{b}$ 。对于 $\mathbf{Ax}=\mathbf{b}$ ，由 LU 分解有

$$\mathbf{LUx}=\mathbf{b}$$

进而求解方程分为两步：

$$\mathbf{Ly}=\mathbf{b}$$

$$\mathbf{Ux}=\mathbf{y}$$

其中这两个方程系数分别为下三角矩阵和上三角矩阵，可以由第一节介绍的方法进行回代计算。

2. 实验目的与要求

- 复习矩阵的 Doolittle 分解。
- 知道 LU 分解计算线性方程组的思路。
- 知道 LU 分解与消去法计算线性方程组的联系。
- 能够编程实现 LU 分解计算线性方程组。

3. 实验内容与数据来源

用 LU 分解方法计算线性方程组 $\mathbf{Ax}=\mathbf{b}$ ，其中：

$$\mathbf{A}=\begin{pmatrix} 6.5574 & 6.7874 & 6.5548 & 2.7692 \\ 0.3571 & 7.5774 & 1.7119 & 0.4617 \\ 8.4913 & 7.4313 & 7.0605 & 0.9713 \\ 9.3399 & 3.9223 & 0.3183 & 8.2346 \end{pmatrix},\mathbf{b}=\begin{pmatrix} 130.3242 \\ 42.9348 \\ 149.9893 \\ 83.1953 \end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	方程的解
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	右向量
N	INTEGER	方程维数

5. 程序代码

```
module LU
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :  LU 分解解方程
!
!-----------------------------------------------------
contains
subroutine solve(A,b,x,N)
implicit real*8(a-z)
integer::N
real*8::A(N,N),b(N),x(N)
real*8::L(N,N),U(N,N)
real*8::y(N)
 call doolittle(A,L,U,N)
call  downtri(L,b,y,N)
call uptri(U,y,x,N)
end subroutine solve
subroutine doolittle(A,L,U,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  LU 分解之 Doolittle 函数
!               A=LU
!-----------------------------------------------------
!  Input  parameters  :
!      1.    A  方阵
!      2.    N  阶数
```

```
! Output parameters :
!      1.  L
!      2.  U
! Common parameters :
!
!-----------------------------------------------------
! Post Script :
!      1.
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::N,i,k,r
real*8::A(N,N),L(N,N),U(N,N)
!U 的第一行
U(1,:)=A(1,:)
!L 的第一列
L(:,1)=a(:,1)/U(1,1)
do k=2,N
    l(k,k)=1
   do j=k,n
       s=0
       do m=1,k-1
        s=s+l(k,m)*u(m,j)
       end do
       u(k,j)=a(k,j)-s
   end do
   do i=k+1,n
     s=0
     do m=1,k-1
      s=s+l(i,m)*u(m,k)
     end do
     l(i,k)=(a(i,k)-s)/u(k,k)

   end do
end do
end subroutine doolittle
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
! Version   :  V1.0
! Coded by  :  syz
! Date      :  2010-4-8
!------------------------------------------------------
! Purpose   :  上三角方程组的回带方法
!              Ax=b
!------------------------------------------------------
! Input  parameters  :
!      1.  A(N,N)系数矩阵
!      2.  b(N)右向量
!      3.  N 方程维数
! Output parameters  :
!      1.  x  方程的根
!      2.
! Common parameters  :
!
!-----------------------------------------------------
```

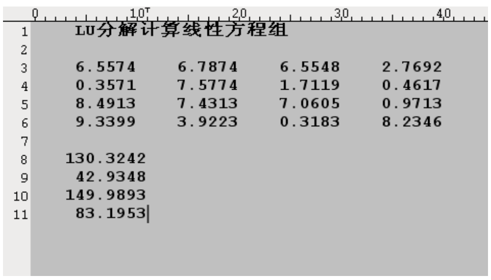
```
implicit real*8(a-z)
integer::i,j,k,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
subroutine downtri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!-----------------------------------------------------
!  Purpose   :  下三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(1)=b(1)/a(1,1)
do k=2,N
   x(k)=b(k)
   do i=1,k-1
     x(k)=x(k)-a(k,i)*x(i)
   end do
   x(k)=x(k)/a(k,k)
end do
end subroutine downtri
end module LU
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  LU 分解计算线性方程组
!             Ax=b
!-----------------------------------------------------
!  In put data  files :
```

```
!       1.    A,b
!       2.
!  Output data files  :
!       1.    x
!       2.
!-----------------------------------------------------
use LU
integer,parameter::N=4
real*8::A(n,n),L(N,N),U(N,N)
real*8::b(N),x(N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
read(11,*)((A(i,j),j=1,N),i=1,N)
read(11,*)b
call solve(A,b,x,N)
write(12,101)x
101 format(T5,'LU 分解计算线性方程组计算结果',//,4(/,F10.6))
end program main
```

6. 实验结论

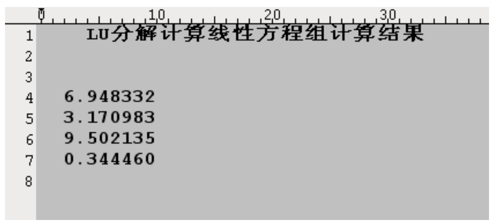
程序运行需要读取数据文件，文件格式如图 1-11 所示。

程序运行后计算结果保存在文件 fout.txt 文件中，如图 1-12 所示。



```
   LU分解计算线性方程组

   6.5574     6.7874     6.5548     2.7692
   0.3571     7.5774     1.7119     0.4617
   8.4913     7.4313     7.0605     0.9713
   9.3399     3.9223     0.3183     8.2346

  130.3242
   42.9348
  149.9893
   83.1953
```

图 1-11　输入数据格式



```
    LU分解计算线性方程组计算结果


   6.948332
   3.170983
   9.502135
   0.344460
```

图 1-12　方程的解

可以验证计算结果是方程的解。

1.7　追赶法计算三对角方程

1. 实验基本原理

在三次样条插值以及差分方法计算常微分方程边值问题，以及热传导定界问题等计算中，都会遇到如下形式的方程组 $\mathbf{Ax}=\mathbf{f}$，其中系数矩阵呈三对角形

$$
\mathbf{A}=\begin{pmatrix} b_1 & c_1 & & & \\ e_1 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & e_{n-1} & b_{n-1} & c_{n-1} \\ & & & e_n & b_n \end{pmatrix}
$$

往往这样的方程组都比较大，上千甚至上万个未知量。

如果系数矩阵满足消去法的条件，固然可以采用 Doolittle 分解法将系数矩阵分解为如下形式的三角阵的乘积，其中

$$
\mathbf{L}=\begin{pmatrix} 1 & & & & \\ l_2 & 1 & & & \\ & l_3 & \ddots & & \\ & & \ddots & 1 & \\ & & & l_n & 1 \end{pmatrix},\mathbf{U}=\begin{pmatrix} u_1 & d_1 & & & \\ & u_2 & d_2 & & \\ & & \ddots & \ddots & \\ & & & u_{n-1} & d_{n-1} \\ & & & & u_n \end{pmatrix}
$$

不过 L 和 U 矩阵非零元素相当稀疏，这种情况如果按照 Doolittle 分解将会照成严重的计算浪费，可以采用一种节省计算量的方法，公式如下：

$$
\begin{aligned} & d_i=c_i, i=1,2,\cdots,n-1 \\ & u_1=b_1 \end{aligned}
$$

$$
\left.\begin{aligned} & l_i=\frac{e_i}{u_{i-1}} \\ & u_i=b_i-l_i c_{i-1} \end{aligned}\right\} i=2,3,\cdots,n
$$

如此原方程变为下面两个方程

$$
\begin{cases} \mathbf{Ly}=\mathbf{f} \\ \mathbf{Ux}=\mathbf{y} \end{cases}
$$

进而

$$
\begin{cases} y_1=f_1 \\ y_i=f_i-l_i y_{i-1}, i=2,3,\cdots,n \end{cases}
$$

$$
\begin{cases} x_n=\dfrac{y_n}{u_n} \\ x_i=\dfrac{\left(y_i-c_i x_{i+1}\right)}{u_i}, i=n-1,n-2,\cdots,1 \end{cases}
$$

便得到原方程的解。

2. 实验目的与要求

- 了解三对角方程的适用范围。
- 理解计算基本原理。
- 熟悉计算流程。
- 能够编程实现该算法。

3. 实验内容与数据来源

编程计算三对角方程组

$$\begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	方程的解
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	三对角方程系数矩阵
F	REAL*8(N)	方程右向量
N	INTEGER	方程维数

5. 程序代码

```
module chase
!----------------------------------------module coment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date       :  2010-4-9
!-----------------------------------------------------
!  Description :  三对角线方程组方法模块
!
!-----------------------------------------------------
contains
 subroutine solve(A,f,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!-----------------------------------------------------
!  Purpose   :  追赶法计算三对角方程组
```

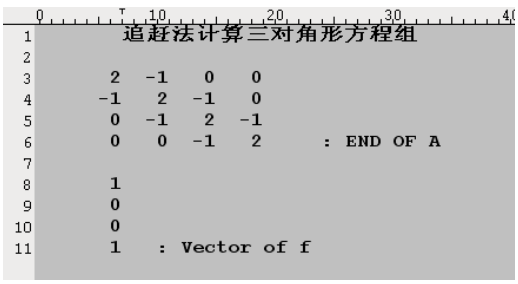
```
!            Ax=f
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A 系数矩阵
!       2.  f 右向量
!  Output parameters  :
!       1.  x 方程的解
!       2.  N 维数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.   注意：该方法仅适用于三对角方程组
!       2.
!---------------------------------------------------
implicit real*8(a-z)
 integer::N
 real*8::A(N,N),f(N),x(N)
 real*8::L(2:N),u(N),d(1:N-1)
real*8::c(1:N-1),b(N),e(2:N)
 integer::i
real*8::y(N)
!---------------把 A 矩阵复制给向量 e,b,c
do i=1,N
   b(i)=a(i,i)
end do
do i=1,N-1
  c(i)=a(i,i+1)
end do
do i=2,N
  e(i)=a(i,i-1)
end do
!------------------------
do i=1,N-1
 d(i)=c(i)
end do
u(1)=b(1)
do i=2,N
  L(i)=e(i)/u(i-1)
  u(i)=b(i)-L(i)*c(i-1)
end do
!------开始回带,求得 y
y(1)=f(1)
do i=2,N
  y(i)=f(i)-L(i)*y(i-1)
end do
!-----开始回带，求得 x
x(n)=y(n)/u(n)
do i=n-1,1,-1
x(i)=(y(i)-c(i)*x(i+1))/u(i)
end do
end subroutine solve
end module chase
program main
!---------------------------------------program comment
```

```
! Version   :  V1.0
! Coded by  :  syz
! Date      :
!-----------------------------------------------------
! Purpose   :  追赶法计算三对角方程
!
!-----------------------------------------------------
! In put data  files :
!       1.   fin.txt 输入数据
!       2.   fout.txt 输出数据
! Output data files  :
!       1.
!       2.
!-----------------------------------------------------
use chase
integer,parameter::N=4
real*8::A(N,N),f(N),x(N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
read(11,*)((A(i,j),j=1,N),i=1,N)
read(11,*)f
call solve(A,f,x,N)
write(12,101)x
101 format(T5,'追赶法计算结果',/,T4,'x=',4(/F12.8))
end program main
```

6. 实验结论

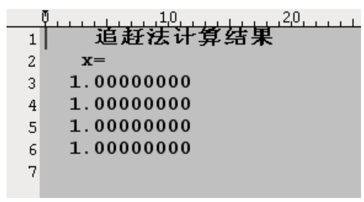
程序的运行需要准备输入数据文件 fin.txt，输入格式如图 1-13 所示。

计算结果保存在文件 fout.txt 中，如图 1-14 所示。



```
        追赶法计算三对角形方程组

    2  -1   0   0
   -1   2  -1   0
    0  -1   2  -1
    0   0  -1   2     : END OF A

    1
    0
    0
    1   : Vector of f
```

图 1-13　追赶法输入数据



```
    追赶法计算结果
   x=
  1.00000000
  1.00000000
  1.00000000
  1.00000000
```

图 1-14　追赶法计算结果

很容易验证，计算结果是原方程的解。

1.8 对称正定阵的Cholesky分解

1. 实验基本原理

由矩阵论可以知，如果矩阵 $\mathbf{A}$ 对称正定，则存在一个实的下三角矩阵 $\mathbf{L}$，使 $\mathbf{A}=\mathbf{L}\mathbf{L}^T$。若限定 $\mathbf{L}$ 的对角元素为正时，则这种分解是唯一的，这就是著名的 Cholesky 分解，即

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \cdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & \cdots & l_{n2} \\ & & \ddots & \\ & & & l_{nn} \end{pmatrix}$$

计算时先令 $\mathbf{L}=\mathbf{0}$，然后按一下步骤计算：

01 $l_{11}=\sqrt{a_{11}}$，

02 $l_{i1}=\dfrac{a_{i1}}{l_{11}}, i=2,N$

03 对 $j=2,N$，做 A~B 处理。

A：$l_{jj}=\sqrt{a_{jj}-\sum\limits_{k=1}^{j-1} l_{jk}^2}$

B：$l_{ij}=\dfrac{\left(a_{ij}-\sum\limits_{k=1}^{j-1} l_{ik}l_{jk}\right)}{l_{jj}}$，$i=j+1,N$

由以上几部就已经算出了 $\mathbf{L}$ 矩阵，Cholesky 分解经常用于对称正定方程的求解（将在下一节介绍），不过分解本身也比较重要，这里单独列出来满足需要对对称正定矩阵分解读者的需求。实际上，许多软件包如 MATLAB 等矩阵分析软件都提供了 Cholesky 分解的函数。

2. 实验目的与要求

- 了解 Cholesky 分解的适用范围。
- 熟悉分解的计算流程。
- 能够编程实现对对称正定矩阵的 Cholesky 分解。

3. 实验内容与数据来源

对矩阵

$$\mathbf{A}=\begin{pmatrix} 233.4615 & 113.8423 & 256.0623 & 145.0697 \\ 113.8423 & 78.6033 & 127.4298 & 95.3089 \\ 256.0623 & 127.4298 & 281.4721 & 164.8676 \\ 145.0697 & 95.3089 & 164.8676 & 181.2339 \end{pmatrix}$$

进行 Cholesky 分解。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
L	REAL*8(N,N)	输出矩阵，其中 A=L*L’
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	要分解的对称正定矩阵
N	INTEGER	矩阵维数

5. 程序代码

```
module cholesky
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   Cholesky 分解模块
!
!-----------------------------------------------------
contains
subroutine  solve(A,L,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  Cholesky 分解子程序
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.  A 对称正定矩阵
!      2.  N 矩阵阶数
!  Output parameters  :
!      1.  L 输出矩阵  A=L*L'
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.  Cholesky 分解只适用于对称正定矩阵
!      2.
!----------------------------------------------------
```

```
integer::N
real*8::A(N,N),L(N,N)
integer::i,j,k
L=0
L(1,1)=dsqrt(a(1,1))
L(2:,1)=a(2:,1)/L(1,1)
do j=2,N
   s=0
   do k=1,j-1
   s=s+L(j,k)**2
   end do
   L(j,j)=dsqrt(a(j,j)-s)
   !注意 i 范围
   do i=j+1,N
       s=0
       do k=1,j-1
         s=s+L(i,k)*L(j,k)
       end do
       L(i,j)=(a(i,j)-s)/L(j,j)
   end do
end do
end subroutine
end module cholesky
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :
!            Cholesky
!-----------------------------------------------------
!  Output data files  :
!       1.   fout.txt 输出文件
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.  注意：Cholesky 分解只适用于对称正定矩阵
!-----------------------------------------------------
use cholesky
integer,parameter::N=4
real*8::A(n,n),L(N,N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read
read(11,*)((A(i,j),j=1,N),i=1,N)
call solve(A,L,N)
write(12,21)
21 format(T5,'Cholesky 分解',/)
do i=1,N
 write(12,22)L(i,:)
end do
22 format(4F10.6)
end program main
```

6. 实验结论

程序运行需要准备输入数据 fin.txt，如图 1-15 所示。

程序运行后，计算结果保存在文件 fout.txt 中，如图 1-16 所示。

```
        Cholesky 分解输入数据

   233.4615  113.8423  256.0623  145.0697
   113.8423   78.6033  127.4298   95.3089
   256.0623  127.4298  281.4721  164.8676
   145.0697   95.3089  164.8676  181.2339
```

图 1-15　Cholesky 分解输入数据

```
    Cholesky分解之L矩阵

 15.279447  0.000000  0.000000  0.000000
  7.450682  4.805272  0.000000  0.000000
 16.758610  0.534148  0.579464  0.000000
  9.494434  5.112904  5.216939  6.142588

 其中  A=L*L'
```

图 1-16　Cholesky 分解计算结果

计算结束后，可以通过 L 矩阵与其转置矩阵相乘验证程序是否正确。

1.9　用Cholesky分解计算对称正定方程

1. 实验基本原理

有了 Cholesky 分解方法计算对称正定线性方程既成水到渠成之事，对于对称正定方程 $\mathbf{Ax}=\mathbf{b}$，利用 Cholsky 分解，方程化为

$$\mathbf{LL}^T\mathbf{x}=\mathbf{b}\text{，等价于}\begin{cases}\mathbf{Ly}=\mathbf{b}\\ \mathbf{L^Tx}=\mathbf{y}\end{cases}$$

而 $\mathbf{Ly}=\mathbf{b}$ 与 $\mathbf{L^Tx}=\mathbf{y}$ 都是三角方程，利用前面章节介绍的回带方法，即很容易计算出 $\mathbf{x}$，即得到原方程的解。

这一过程就是 Cholesky 方法或称为平方根法。

2. 实验目的与要求

- 理解 Cholesky 分解方法计算对称正定方程的基本原理。
- 复习三角方程的回带方法。
- 能够编程实现以上算法。

3. 实验内容与数据来源

用 Cholesky 分解方法计算方程组 $\mathbf{Ax}=\mathbf{b}$，其中

$$\mathbf{A}=\begin{pmatrix} 4 & -1 & 1 \\ -1 & 4.25 & 2.75 \\ 1 & 2.75 & 3.5 \end{pmatrix},\mathbf{b}=\begin{pmatrix} 4 \\ 6 \\ 7.25 \end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	方程的解
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	右向量
N	INTEGER	方程维数

5. 程序代码

```
module sym_p
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   Cholesky 分解计算对称正定方程模块
!
!-----------------------------------------------------
contains
subroutine solve(A,b,x,N)
implicit real*8(a-z)
integer::N
real*8::A(N,N),b(N),x(N)
real*8::L(N,N),y(N),LT(N,N)
!LT 为 L 的转置矩阵
integer::i,j
call  chol(A,L,N)
call  downtri(L,b,y,N)
do i=1,N
 do j=1,N
   LT(i,j)=L(j,i)
 end do
end do
call uptri(LT,y,x,N)  !这一步已经算出了 x
end subroutine solve
subroutine  chol(A,L,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  Cholesky 分解子程序
```

```
!------------------------------------------------------
integer::N
real*8::A(N,N),L(N,N)
integer::i,j,k
L=0
L(1,1)=dsqrt(a(1,1))
L(2:,1)=a(2:,1)/L(1,1)
do j=2,N
   s=0
   do k=1,j-1
   s=s+L(j,k)**2
   end do
   L(j,j)=dsqrt(a(j,j)-s)
   !注意 i 范围
   do i=j+1,N
       s=0
       do k=1,j-1
         s=s+L(i,k)*L(j,k)
       end do
       L(i,j)=(a(i,j)-s)/L(j,j)
   end do
end do
end subroutine
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!------------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!------------------------------------------------------
implicit real*8(a-z)
integer::i,j,k,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
subroutine downtri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!------------------------------------------------------
!  Purpose   :  下三角方程组的回带方法
!               Ax=b
!------------------------------------------------------
implicit real*8(a-z)
```

```
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(1)=b(1)/a(1,1)
do k=2,N
   x(k)=b(k)
   do i=1,k-1
      x(k)=x(k)-a(k,i)*x(i)
   end do
   x(k)=x(k)/a(k,k)
end do
end subroutine downtri
end module sym_p
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  对称正定方程组的计算（Cholesky 分解方法）
!
!-----------------------------------------------------
! data files  :
!      1.  fin.out  输入文件
!      2.  fout.txt 输出文件
!      2.
!-----------------------------------------------------
use sym_p
integer,parameter::N=3
real*8::A(N,N)
real*8::b(N),x(N)
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
read(11,*)((A(i,j),j=1,N),i=1,N)
read(11,*)b
call solve(A,b,x,N)
write(12,21)
21 format(T5,'对称正定方程组的解为',/)
 write(12,22)x
22 format(T3,3F10.6)
end program main
```

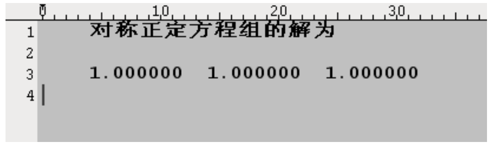
6. 实验结论

程序运行需要准备输入数据 fin.txt，格式如图 1-17 所示。

程序运行后计算结果保存在文件 fout.txt 中，如图 1-18 所示。

```
 对称正定矩阵的计算（Chol分解法）

  4     -1     1
 -1   4.25  2.75
  1   2.75   3.5

  4
  6
  7.25
```

图 1-17　对称正定方程组的输入数据



```
   对称正定方程组的解为

   1.000000   1.000000   1.000000
```

图 1-18　方程的解

把方程的解代入原方程可以验证满足原方程。

1.10　行列式的计算

1. 实验基本原理

计算矩阵的行列式可以按照高斯消去法直接进行计算，在已经完成 LU 分解之后也可以利用 LU 分解进行计算。这里采用 Crout 分解法把系数矩阵分解为

$$\mathbf{A}=\mathbf{LU}$$

其中 **L** 为下三角矩阵， **U** 为单位上三角矩阵，进而有

$$\det(\mathbf{A})=\det(\mathbf{L})\det(\mathbf{U})=\det(\mathbf{L})=\prod_{i=1}^{n} l_{ii}$$

如此便给出了行列式的计算方法。

2. 实验目的与要求

- 复习矩阵的 LU 分解。
- 能自行给出行列式计算的算法。
- 能编程实现对行列式的计算。

3. 实验内容与数据来源

已知

$$\mathbf{A}=\begin{pmatrix} 2.7486 & 1.9022 & 3.8958 & 0.0595 & 2.6427 \\ 4.5860 & 2.8391 & 4.6701 & 1.6856 & 0.8282 \\ 1.4292 & 0.3793 & 0.6495 & 0.8109 & 3.0099 \\ 3.7860 & 0.2698 & 2.8441 & 3.9714 & 1.3149 \\ 3.7686 & 2.6540 & 2.3470 & 1.5561 & 3.2704 \end{pmatrix}$$

要求计算矩阵 **A** 的行列式。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
D	REAL*8	矩阵的行列式
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	要计算的矩阵
N	INTEGER	矩阵维数

5. 程序代码

```
module det
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :  计算矩阵行列式模块
!
!-----------------------------------------------------
contains
subroutine  solve(A,d,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算矩阵行列式函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A 矩阵
!       2.  N 矩阵维数
!  Output parameters  :
!       1.    d  矩阵行列式
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N,i
real*8::A(N,N),L(N,N),U(N,N)
call crout(A,L,U,N)
d=1d0
do i=1,N
  d=d*L(i,i)
end do
end subroutine solve
subroutine crout(A,L,U,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
```

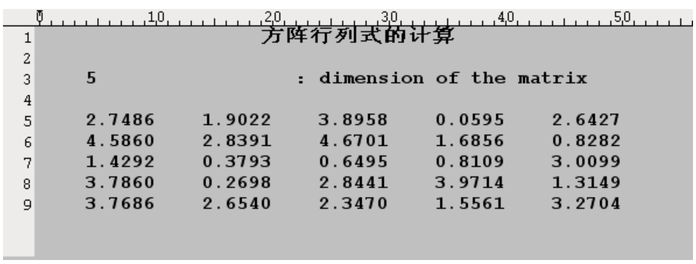
```
! Date      :
!---------------------------------------------------
! Purpose   : LU之Crout分解
!             A=LU
!---------------------------------------------------
implicit real*8(a-z)
integer::N,i,k,r
real*8::A(N,N),L(N,N),U(N,N)
!L第一列
L(:,1)=a(:,1)
!U第一行
U(1,:)=a(1,:)/L(1,1)
do k=2,N
   do i=k,n
       s=0
      do r=1,k-1
       s=s+l(i,r)*u(r,k)
      end do
      l(i,k)=a(i,k)-s
   end do
   do j=k+1,n
     s=0
     do r=1,k-1
      s=s+l(k,r)*u(r,j)
     end do
     u(k,j)=(a(k,j)-s)/l(k,k)

   end do
   u(k,k)=1
end do
end subroutine crout
end module det
module driver
!----------------------------------------module coment
! Version    : V1.0
! Coded by   : syz
! Date       :
!---------------------------------------------------
! Description :
!               驱动程序模块
!---------------------------------------------------
contains
subroutine dri_main(N)
use det
integer::N
real*8::A(N,N),d
!读入矩阵
read(11,*)((A(i,j),j=1,N),i=1,N)
call solve(A,d,N)
write(12,101)
101 format(T5,'该矩阵的行列式为',/)
write(12,*)d
end subroutine dri main
end module driver
program main
```

```
!------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算矩阵行列式
!
!-----------------------------------------------------
!  In put data  files :
!       1.   fin.txt   读入的数据
!       2.
!  Output data files  :
!       1.
!       2.   fout.txt 计算结果，矩阵的维数由输入卡片读取
!-----------------------------------------------------
use driver
integer::N
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
!读入 A 矩阵
!读入方程维数系数
read(11,*)N
!调用驱动函数
call dri_main(N)
end program main
```

6. 实验结论

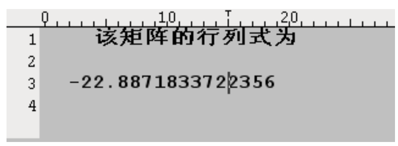
需要准备输入数据文件 fin.txt，格式如图 1-19 所示。

计算结果保存在文件 fou.txt 中，如图 1.20 所示。



```
方阵行列式的计算

5                   : dimension of the matrix

2.7486     1.9022     3.8958     0.0595     2.6427
4.5860     2.8391     4.6701     1.6856     0.8282
1.4292     0.3793     0.6495     0.8109     3.0099
3.7860     0.2698     2.8441     3.9714     1.3149
3.7686     2.6540     2.3470     1.5561     3.2704
```

图 1-19　输入数据格式



```
该矩阵的行列式为

-22.8871833722356
```

图 1-20　矩阵行列式计算结果

1.11　矩阵方程的计算

1. 实验基本原理

对于矩阵方程

$$\mathbf{A}_{[N,N]}\mathbf{X}_{[N,M]} = \mathbf{B}_{[N,M]}$$

记 $\mathbf{X}$ 的第 i 列向量为 $\mathbf{X}_i\left(i=1,m\right)$， $\mathbf{B}$ 的第 i 列向量为 $\mathbf{B}_i\left(i=1,m\right)$，则上述方程等价为

$$\begin{cases} \mathbf{AX}_1 = \mathbf{B}_1 \\ \mathbf{AX}_2 = \mathbf{B}_2 \\ \quad\vdots \\ \mathbf{AX}_m = \mathbf{B}_m \end{cases}$$

但在实际计算时不应该分别求解，如果是这样的话就造成计算机资源极大浪费，而应该是对所有向量一次选主元，然后分别回代。即可以得到方程的解矩阵 $\mathbf{X}_{[N,M]}$。

2. 实验目的与要求

- 了解矩阵方程的计算的基本原理。
- 能够编程对矩阵方程的数值计算。
- 注意不要对方程分别选主元解算。

3. 实验内容与数据来源

计算矩阵方程

$$\mathbf{AX} = \mathbf{B}$$

其中

$$\mathbf{A} = \begin{pmatrix} 1.80 & 2.88 & 2.05 & -0.89 \\ 525.00 & -295.00 & -95.00 & -380.00 \\ 1.58 & -2.69 & -2.90 & -1.04 \\ -1.11 & -0.66 & -0.59 & 0.80 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} 9.52 & 18.47 \\ 2435.00 & 225.00 \\ 0.77 & -13.28 \\ -6.22 & -6.21 \end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N,M)	矩阵方程的解矩阵
A	REAL*8(N,N)	矩阵方程系数矩阵
B	REAL*8(N,M)	右矩阵
N	INTEGER	方程维数
M	INTEGER	右矩阵的列数

5. 程序代码

```
module mat_eq
```

```
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-------------------------------------------------------
!  Description : 矩阵方程
!
!-------------------------------------------------------
!  Contains    :
!      1.   driver  驱动函数
!      2.   solve   方法函数
!-------------------------------------------------------
contains
subroutine solve(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-------------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程组
!               AX=B
!-------------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   B(N,M)右矩阵
!       3.   N 方程维数
!       4.   M 右矩阵的列数
!  Output parameters  :
!       1.  X 方程的解矩阵
!       2.
!  Common parameters  :
!
!------------------------------------------------------
implicit real*8(a-z)
integer::i,k,N,M
integer::id_max  !主元素标号
real*8::A(N,N),B(N,M),X(N,M)
real*8::Aup(N,N),Bup(N,M)
!Ab 为增广矩阵 [AB]
real*8::AB(N,N+M)
real*8::vtemp1(N+M),vtemp2(N+M)
real*8::vtmp(N),xtmp(N)
AB(1:N,1:N)=A
AB(:,N+1:N+M)=B
!##########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id_max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
      if (dabs(Ab(i,k))>elmax) then
         elmax=Ab(i,k)
         id_max=i
```

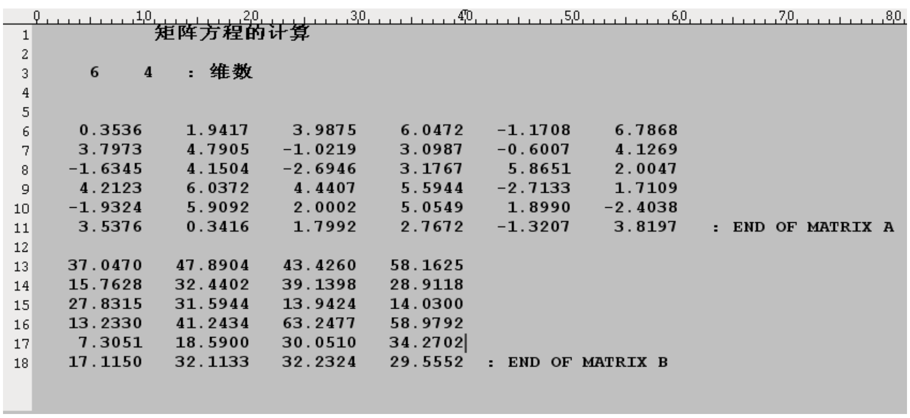
```
        end if
      end do
    !至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
     !交换两行元素，其他不变
      vtemp1=Ab(k,:)
      vtemp2=Ab(id_max,:)

      Ab(k,:)=vtemp2
      Ab(id_max,:)=vtemp1
    !
    !以上一大段是为交换两行元素，交换完成以后即按照消元法进行
    !##########################################################
     do i=k+1,N
       temp=Ab(i,k)/Ab(k,k)
       Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
     end do
    end do
    !-----------------------------
    ! 经过上一步，Ab 已经化为如下形式的矩阵
    !           | *  *  *  *  #  #|
    !     [A b]= | 0  *  *  *  #  #|
    !           | 0  0  *  *  #  #|
    !           | 0  0  0  *  #  #|
    !
    Aup(:,:)=AB(1:N,1:N)
    do i=1,m
    !调用用上三角方程组的回带方法
    vtmp=AB(:,N+i)
    call uptri(Aup,vtmp,xtmp,n)
    !把计算结果赋值给 X
    X(:,i)=xtmp
    end do
    end subroutine solve
    subroutine uptri(A,b,x,N)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :  2010-4-8
    !-----------------------------------------------------
    !  Purpose   :  上三角方程组的回带方法
    !               Ax=b
    !-----------------------------------------------------
    !  Input  parameters  :
    !      1.   A(N,N)系数矩阵
    !      2.   b(N)右向量
    !      3.   N 方程维数
    !  Output parameters  :
    !      1.  x  方程的根
    !      2.
    !  Common parameters  :
    !
    !----------------------------------------------------
    implicit real*8(a-z)
    integer::i,j,N
    real*8::A(N,N),b(N),x(N)
```

```
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
subroutine driver(N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   : 驱动程序
!-----------------------------------------------------
!  Input  parameters  :
!      1.   N     描述 A(N,N)
!      2.   M     描述方程  X(N,M), B(N,M)
!  Output parameters  :
!      1.
!      2.
!  P.S  :
!     N,M  从文件中读取
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N,M
integer::i,j
real*8::A(N,N),B(N,M),X(N,M)
!读入系数 A 矩阵
read(11,*)((A(i,j),j=1,N),i=1,N)
!读入 B 矩阵
read(11,*)((B(i,j),j=1,M),i=1,N)
call solve(A,B,X,N,M)
write(12,101)
write(12,102)((X(i,j),j=1,m),i=1,n)
!变量输出格式只针对 IVF 编译器，在 CVF 中不支持
102 format(<N>(<m>F16.10/))
!do i=1,m
!
!   do j=1,n
!     write(12,102)j,i,x(j,i)
!   end do
!
!end do
101 format(T4,'消去法计算矩阵方程，版本.1',/)
!102 format(T4,'x(',I2,',',I2,')=',f10.5)
write(12,103)
103 format(/'P.S:本软件既可以计算线性方程组 Ax=b,
! 也可以计算矩阵方程 AX=B')
end subroutine driver
end module mat_eq
```

```
program main
!----------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!--------------------------------------------------------
!  Purpose   :  消去法解矩阵方程
!
!--------------------------------------------------------
!  In put data  files :
!       1.  fin.txt  输入方程系数
!       2.
!  Output data files  :
!       1. fout.txt  计算结果
!       2.
!--------------------------------------------------------
!  Post Script :
!       1.    需要准备输入数据
!       2.    由于驱动函数调用方法函数
!--------------------------------------------------------
use mat_eq
integer::N,M
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
!读说明文字
!读入方程维数系数
read(11,*)N,M
!调用驱动函数
call driver(N,M)
end program main
```

6. 实验结论

程序要运行需要准备数据文件 fin.txt，输入格式如图 1-21 所示。



```
                矩阵方程的计算

      6     4    : 维数


    0.3536    1.9417    3.9875    6.0472   -1.1708    6.7868
    3.7973    4.7905   -1.0219    3.0987   -0.6007    4.1269
   -1.6345    4.1504   -2.6946    3.1767    5.8651    2.0047
    4.2123    6.0372    4.4407    5.5944   -2.7133    1.7109
   -1.9324    5.9092    2.0002    5.0549    1.8990   -2.4038
    3.5376    0.3416    1.7992    2.7672   -1.3207    3.8197    : END OF MATRIX A

   37.0470   47.8904   43.4260   58.1625
   15.7628   32.4402   39.1398   28.9118
   27.8315   31.5944   13.9424   14.0300
   13.2330   41.2434   63.2477   58.9792
    7.3051   18.5900   30.0510   34.2702
   17.1150   32.1133   32.2324   29.5552   : END OF MATRIX B
```

图 1-21 输入数据

计算结果保存在文件 fou.txt 中，如图 1-22 所示。

```
  消去法计算矩阵方程，版本1.1

   0.2121732635    3.6122334792    4.0016940917    0.8670272755
   0.3572632239    0.7493976331    2.2689460352    1.9546122268
   2.6082553783    3.2981298002    2.1619892132    4.1568916014
   0.4836318352    2.5928760331    4.1266154737    4.0169191135
   4.0907513473    4.8649561369    0.4173970857    0.3023852054
   4.0877427440    3.2449706750    0.6658036012    1.9962224714


P.S:本软件既可以计算线性方程组Ax=b，也可以计算矩阵方程AX=B
```

图 1-22　计算结果

从计算结果可以看到

$$
\mathbf{X}=\begin{pmatrix}
0.21217 & 3.61223 & 4.00169 & 0.86703\\
0.35726 & 0.74940 & 2.26895 & 1.95461\\
2.60826 & 3.29813 & 2.16199 & 4.15689\\
0.48363 & 2.59288 & 4.12662 & 4.01692\\
4.09075 & 4.86496 & 0.41740 & 0.30239\\
4.08774 & 3.24497 & 0.66580 & 1.99622
\end{pmatrix}
$$

程序编译并连接以后产生的可执行文件，可以计算一般的线性方程组和形如 $\mathbf{AX}=\mathbf{B}$ 的矩阵方程，只要自行修改输入数据即可。如要计算方程组

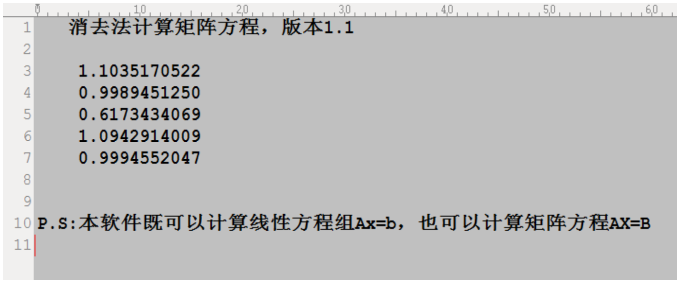
$$
\begin{pmatrix}
-0.7312 & -1.6835 & -2.2759 & 3.2033 & -0.5213\\
-1.8320 & -5.8451 & -4.0188 & -5.4732 & 3.4274\\
0.5686 & 3.8406 & -1.1031 & 1.3786 & -1.8226\\
0.2797 & -4.3283 & -2.6051 & -3.3088 & 3.8305\\
-3.0802 & -4.9378 & 3.5163 & -1.7716 & -2.9855
\end{pmatrix}\mathbf{Y}=\begin{pmatrix}
-0.9093\\
-12.9053\\
3.4700\\
-5.4157\\
-11.0834
\end{pmatrix}
$$

只要将输入卡片改成如图 1-23 所示的格式。

```
              矩阵方程的计算

     5     1    : 维数


   -0.7312   -1.6835   -2.2759    3.2033   -0.5213
   -1.8320   -5.8451   -4.0188   -5.4732    3.4274
    0.5686    3.8406   -1.1031    1.3786   -1.8226
    0.2797   -4.3283   -2.6051   -3.3088    3.8305
   -3.0802   -4.9378    3.5163   -1.7716   -2.9855    : END OF MATRIX A

   -0.9093
  -12.9053
    3.4700
   -5.4157
  -11.0834    : END OF MATRIX B
```

图 1-23　用本软件计算线性方程组输入文件

执行文件后，产生计算结果如图 1-24 所示。



```
   消去法计算矩阵方程，版本1.1

    1.1035170522
    0.9989451250
    0.6173434069
    1.0942914009
    0.9994552047


P.S:本软件既可以计算线性方程组Ax=b，也可以计算矩阵方程AX=B
```

图 1-24　线性方程计算结果

把计算结果代入原方程可以验证计算结果是正确的。

1.12　逆矩阵的计算

1. 实验基本原理

设矩阵 $\mathbf{A}_{[N,N]}$ 为非奇异矩阵，且其逆矩阵存在，记 $\mathbf{A}^{-1}=\mathbf{X}$，则

$$\mathbf{AX}=\mathbf{I}$$

由此，计算矩阵 A 的逆矩阵可以转化为计算矩阵方程问题，利用上一节介绍的计算方法很容易即可计算出逆矩阵。

实际上，如果没有计算矩阵方程的函数而直接利用消去法也是很容易给出逆矩阵的算法，这里就不多介绍，有兴趣的读者可以参考数值代数专著。

2. 实验目的与要求

- 复习矩阵方程的计算。
- 了解逆矩阵计算的基本原理。
- 能编程实现逆矩阵的计算方法。

3. 实验内容与数据来源

计算矩阵 **A** 的逆矩阵，其中

$$\mathbf{A}=\begin{pmatrix}1 & 2 & -1\\ 1 & 1 & 2\\ 1 & 2 & -1\end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
INVA	REAL*8(N,N)	逆矩阵
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	欲求逆的矩阵
N	INTEGER	矩阵维数

5. 程序代码

```
module inv mat
!----------------------------------------module coment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date       : 2010-4-8
!------------------------------------------------------
!  Description : 计算逆矩阵
!
!------------------------------------------------------
!  Contains    :
!      1.   driver  驱动函数
!      2.   solve   方法函数
!------------------------------------------------------
contains
subroutine solve(A,invA,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!------------------------------------------------------
!  Purpose   :  计算逆矩阵
!
!------------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i
real*8::A(n,n),invA(n,n),E(n,n)
E=0
```

```
!设置 E 为单位矩阵
do i=1,n
  E(i,i)=1
end do
call mateq(A,E,invA,N,N)
end subroutine solve
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程组
!               AX=B
!-----------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   B(N,M)右矩阵
!       3.   N 方程维数
!       4.   M 右矩阵的列数
!  Output parameters  :
!       1.  X 方程的解矩阵
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N,M
integer::id_max  !主元素标号
real*8::A(N,N),B(N,M),X(N,M)
real*8::Aup(N,N),Bup(N,M)
!Ab 为增广矩阵[AB]
real*8::AB(N,N+M)
real*8::vtemp1(N+M),vtemp2(N+M)
real*8::vtmp(N),xtmp(N)
AB(1:N,1:N)=A
AB(:,N+1:N+M)=B
!##########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
   elmax=dabs(Ab(k,k))
   id_max=k
   !这段为查找主元素
   !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
    do i=k+1,n
     if (dabs(Ab(i,k))>elmax) then
        elmax=Ab(i,k)
        id_max=i
     end if
   end do
!至此，已经完成查找最大元素，查找完成以后与第 k 行交换
 !交换两行元素，其他不变
   vtemp1=Ab(k,:)
   vtemp2=Ab(id_max,:)
   Ab(k,:)=vtemp2
```

```
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!##########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!           | *  *  *  *  #  #|
!     [A b]= | 0  *  *  *  #  #|
!           | 0  0  *  *  #  #|
!           | 0  0  0  *  #  #|
!
Aup(:,:)=AB(1:N,1:N)
do i=1,m
!调用用上三角方程组的回带方法
vtmp=AB(:,N+i)
call uptri(Aup,vtmp,xtmp,n)
!把计算结果赋值给 x
X(:,i)=xtmp
end do
end subroutine mateq
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
```

```
end do
end subroutine uptri
subroutine drive(N)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  驱动程序
!               用以读取数据，计算结果并把结果保存于文件中
!
!----------------------------------------------------
!  Post Script :
!       1.     fin   输入文件
!       2.     fout  输出文件
!----------------------------------------------------
implicit real*8(a-z)
integer::N,i,j
real*8::A(N,N),invA(N,N)
read(11,*)((A(i,j),j=1,N),i=1,N)
write(12,101)
101 format(/,T12,'逆矩阵为',/)
call solve(A,invA,N)
write(12,102)((invA(i,j),j=1,n),i=1,n)
!变量输出格式只针对 IVF 编译器，在 CVF 中不支持
102 format(<N>(<N>F16.10/))
end subroutine drive
end module inv_mat
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  逆矩阵的计算
!
!-----------------------------------------------------
!  In put data  files :
!       1.  fin.txt  输入方程系数
!       2.
!  Output data files  :
!       1. fout.txt  计算结果
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.    需要准备输入数据
!       2.    由于驱动函数调用方法函数
!-----------------------------------------------------
use inv_mat
integer::N
!N 表示矩阵的维数，由文件读入
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
!读说明文字
```

```
    !读入方程维数系数
    read(11,*)N
    call drive(N)
    end program main
```

6. 实验结论

计算结果保存于文件 fout.txt 中，打开该文件可以看到结果如图 1-25 所示。

使用本程序时要先准备输入文件，输入文件需要注意格式，输入格式如图 1-26 所示。

```

                        逆矩阵为

    0.1764705882   -0.0588235294    0.2941176471
    0.2941176471    0.2352941176   -0.1764705882
   -0.2352941176    0.4117647059   -0.0588235294

```

图 1-25　逆矩阵计算结果

```
           输入矩阵

    3    :  维数

   1   2  -1
   1   1   2
   3  -1   1        : END OF MATRIX A

```

图 1-26　数据输入卡片

1.13　线性方程组解的迭代改进

1. 实验基本原理

一般说来，很难使方程组的解的精度超过计算机浮点数的精度，尤其是对于大型线性方程组，甚至可以说解接近计算机精度都是相当困难的。对于维数较高的方程，在计算中不断有误差累积，造成解丢失两、三位有效数字的情况是时有发生的。

对于这种丢失精度的情况，一个简单可靠的方法可以使解能迅速恢复到计算机精度，即采用解的迭代改进方法，其原理实际上很简单，下面叙述之。

设线性方程组

$$\mathbf{Ax} = \mathbf{b}$$

的精确解为 $\mathbf{x}$，然而这个精确的 $\mathbf{x}$ 我们却无从知晓。对于上述方程，我们可以采用直接方法计算出带有误差的解 $\mathbf{x} + \boldsymbol{\delta}\mathbf{x}$。现在我们想办法求出误差，从而使精度得到回复。既然方程的解带有误差，如果把这个带误差的解带入原方程，从而造成右向量 $\mathbf{b}$ 的摄动，即

$$\mathbf{A}(\mathbf{x} + \boldsymbol{\delta}\mathbf{x}) = \mathbf{b} + \boldsymbol{\delta}\mathbf{b}$$

从而我们可以建立残差方程

$$\mathbf{A}\boldsymbol{\delta}\mathbf{x} = \mathbf{A}(\mathbf{x} + \boldsymbol{\delta}\mathbf{x}) - \mathbf{b}$$

我们可以通过求解以上方程从而得到改正量 $\boldsymbol{\delta}\mathbf{x}$，继而我们可以改正方程的解

$$\mathbf{x}_{compute} = (\mathbf{x} + \boldsymbol{\delta}\mathbf{x}) - \boldsymbol{\delta}\mathbf{x}$$

当然，实际上亦不可能求得完全精确的改正量 **δx**，但是经过一次迭代改进量就会非常小，进而可以循环改进。一般而言，改进一、两次就可以获得比较理想的结果。

2. 实验目的与要求

- 了解解的迭代改进原理。
- 能够建立残差方程并求解之。
- 能通过迭代实现对直接法的解的改进。

3. 实验内容与数据来源

用直接法计算方程 $\mathbf{Ax}=\mathbf{b}$，其中

$$\mathbf{A}=\begin{pmatrix} 3.3435 & 1.4946 & 0.3834 & -1.9537 & -2.4013 & -3.5446 \\ 1.0198 & 6.1618 & 4.9613 & 2.7491 & 3.0007 & -3.6393 \\ 2.3703 & 2.7102 & 4.2182 & 1.1730 & -0.6859 & 3.6929 \\ 1.5408 & 4.1334 & 0.5732 & 5.6869 & 3.1065 & 0.7970 \\ 3.8921 & 3.4762 & 3.9335 & -2.1556 & -6.1815 & 0.4986 \\ 2.4815 & 8.2582 & 4.6190 & -0.0022 & -0.3620 & -8.5505 \end{pmatrix}$$

$$\mathbf{b}=\begin{pmatrix} 5.0537 \\ 0.0228 \\ -3.4177 \\ 7.6380 \\ 0.5575 \\ 9.8252 \end{pmatrix}$$

然后进行解的迭代改进，输出改进序列。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	方程的解
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	右向量
N	INTEGER	方程维数

5. 程序代码

```
module iterprove
```

```
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-----------------------------------------------------
!  Description : 迭代改进模块
!
!-----------------------------------------------------
!  Contains    :
!      1.   solve  方法函数
!      2.
!-----------------------------------------------------
contains
subroutine solve(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.08.10
!-----------------------------------------------------
!  Purpose     :  线性方程组的迭代改进
!
!  Post Script :
!       1.    注意是先调用选主元消去法计算
!       2.    然后计算残差方程
!       3.    再迭代改进
!-----------------------------------------------------
!  Input  parameters  :
!       1.       A 系数矩阵
!       2.       b 右向量
!       3.       N 方程维数
!  Output parameters  :
!       1.       x 方程的解
!       2.
!  Common parameters  :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::i,N
integer::itmax=5
!默认迭代次
real*8::A(n,n),b(n),x(N),x1(n),x2(n)
real*8::db(N),dx(N)
!先调用一次计算，得到初值
call gauss(A,b,x1,N)
!*****************************************
!x1=1d0
! 该语句可以测试在 X1 非常差的情况下计算结果
! 如果要查看可以去掉这 x1=1d0 的注释，
! 而把 call gauss(A,b,x1,N)注释掉
!*****************************************
!输出初值
write(12,101)0,x1
do i=1,itmax
  !db 为残差
```

```
    db=matmul(A,x1)-b
    call gauss(A,db,dx,N)
    !由此得到改进量 dx
    !x2 为改进后的新解
    x2=x1-dx
    !更新变量
    x1=x2
    !输出改进结果
    write(12,102)i,x1
  end do
  x=x1
  101 format(/,T25,'迭代改进序列',//,I3,<n>(F10.5))
  102 format(I3,<N>(F10.5))
  end subroutine solve
  subroutine gauss(A,b,x,N)
  !--------------------------------subroutine  comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      : 2010-4-8
  !-----------------------------------------------------
  !  Purpose   :  高斯列主元消去法
  !               Ax=b
  !-----------------------------------------------------
  implicit real*8(a-z)
  integer::i,k,N
  integer::id_max  !主元素标号
  real*8::A(N,N),b(N),x(N)
  real*8::Aup(N,N),bup(N)
  !Ab 为增广矩阵 [Ab]
  real*8::Ab(N,N+1)
  real*8::vtemp1(N+1),vtemp2(N+1)
  Ab(1:N,1:N)=A
  Ab(:,N+1)=b
  do k=1,N-1
     elmax=dabs(Ab(k,k))
     id_max=k
      do i=k+1,n
       if (dabs(Ab(i,k))>elmax) then
          elmax=Ab(i,k)
          id_max=i
       end if
     end do
     vtemp1=Ab(k,:)
     vtemp2=Ab(id_max,:)
     Ab(k,:)=vtemp2
     Ab(id_max,:)=vtemp1
  !
    do i=k+1,N
      temp=Ab(i,k)/Ab(k,k)
      Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
    end do
  end do
  !-----------------------------
  ! 经过上一步，Ab 已经化为如下形式的矩阵
  !           | *  *  *  *  # |
```

```
!     [A b]= | 0  *  *  *  # |
!            | 0  0  *  *  # |
!            | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine gauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
   x(i)=b(i)
  do j=i+1,N
   x(i)=x(i)-a(i,j)*x(j)
  end do
   x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module iterprove
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!
!-----------------------------------------------------
!  In put data  files :
!       1.  fin.txt  输入方程系数
!       2.
!  Output data files  :
!       1. fout.txt  计算结果
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.    需要准备输入数据
!
!-----------------------------------------------------
use iterprove
implicit real*8(a-z)
integer,parameter:: N=6
```

```
    integer::i,j
    real*8::A(N,N),b(N),x(N)
    open(unit=11,file='fin.txt')
    open(unit=12,file='fout.txt')
    read(11,*)
    !读入 A 矩阵
    read(11,*)((A(i,j),j=1,N),i=1,N)
    !读入 B 向量
    read(11,*) b
    !调用迭代模块
    call solve(A,b,x,N)
    end program main
```

6. 实验结论

程序编译链接之后，需要准备数据输入文件，格式如图 1-27 所示。

```
                        迭代改进

   3.3435    1.4946    0.3834   -1.9537   -2.4013   -3.5446
   1.0198    6.1618    4.9613    2.7491    3.0007   -3.6393
   2.3703    2.7102    4.2182    1.1730   -0.6859    3.6929
   1.5408    4.1334    0.5732    5.6869    3.1065    0.7970
   3.8921    3.4762    3.9335   -2.1556   -6.1815    0.4986
   2.4815    8.2582    4.6190   -0.0022   -0.3620   -8.5505

   5.0537
   0.0228
  -3.4177
   7.6380
   0.5575
```

图 1-27 数据输入文件

计算结果保存在文件 fout.txt 中，如图 1-28 所示。

在图 1-28 中，似乎计算结果一直没有变化，实际上是因为第一次结果已经比较好，后面的改正量已经较小，为了排版需要这里输出了小数点后 5 位，如果取小数点位数更多，则可以看到每次都有一定的改进。

```

                       迭代改进序列

 0   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
 1   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
 2   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
 3   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
 4   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
 5   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
```

图 1-28 迭代改进序列

如果第一次计算结果较差，则改进效果更明显，如果强制第一次计算结果为

$$x=\begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1\end{pmatrix}^T$$

则迭代一次后，计算结果就会得到明显的改善，结果如图 1-29 所示。

```
                     迭代改进序列

0   1.00000   1.00000   1.00000   1.00000   1.00000   1.00000
1   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
2   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
3   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
4   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
5   0.46133   5.21617  -4.62817  -3.06304   1.37487   1.46509
```

图 1-29　第一次解算较差情况下的迭代改进

从计算结果可以看出，迭代改进是相当有效的。

一般而言，迭代改进 1 到 2 次就可以有明显的效果，再进行迭代改进量就已经很小了。迭代改进还有一个好处是可以检验方程解的收敛性。

注意，这里的迭代改进和下一章介绍的线性方程组的迭代法不可混淆。这里依然是采用直接方法计算线性方程，只是如果方程解不是太理想，可以通过求解残差方程以改进原先的计算结果，或者检验方程解的收敛性。

本章小结

本章介绍了常见的线性方程组的计算方法，如选主元消去法，三对角矩阵的追赶法以及矩阵的常见分解方法。

阅读本章内容需要具备线性代数的一些基础知识，如果读者有所遗忘，可以翻阅相关教材。实际上第 3 章将要介绍的正交变换方法也可以用于特定方程组的计算，不过鉴于最小二乘有更多的统计学意义，故而留在第 3 章介绍。

第 2 章

◂解线性方程组的迭代方法方法▸

对于与中小型方程组，一般采用直接法，如果不计舍入误差，直接法能在预定的计算步数内求得方程精确解，计算效率高且很可靠。而对于维数比较高，尤其是大型的稀疏方程组，由于直接法存储量与计算量太大，通常迭代法更有优势。本章将展开常用迭代算法介绍。

2.1　Jacobi迭代法

1. 实验基本原理

设线性方程组

$$\mathbf{AX}=\mathbf{b}$$

的系数矩阵非奇异，且主对角元素 $a_{ii}\neq 0$。令

$$\begin{cases} b_{ij}=-a_{ij}/a_{ii} & i\neq j \\ g_i=b_i/a_{ii} & \mathrm{i=1,2,\cdots,n} \end{cases}$$

原方程可以改写为

$$\begin{cases} x_1=b_{12}x_2+b_{13}x_3+\cdots+b_{1n}x_n+g_1 \\ x_2=b_{21}x_x+b_{23}x_3+\cdots+b_{2n}x_n+g_2 \\ \vdots \\ x_1=b_{n1}x_1+b_{n2}x_2+\cdots+b_{n,n-1}x_{n-1}+g_n \end{cases}$$

令 $\mathbf{D}=diag(a_{11},a_{22},\cdots,a_{nn})$， $\mathbf{g}=(g_1,g_2,\cdots g_n)^T$, $\mathbf{B=D^{-1}(D-A)=E-D^{-1}A}$

则上述方程用矩阵符号表示既为

$$\mathbf{X}=\mathbf{BX}+\mathbf{g}$$

由此如果我们得到迭代公式 $\mathbf{X}^{(k)}=\mathbf{BX}^{(k+1)}+\mathbf{g}$，此即 Jacobi 迭代。

2. 实验目的与要求

- 对于给定的线性方程组，能判断是否合适使用 Jacobi 迭代。
- 能够构造出迭代公式。
- 能编程实现 Jacobi 迭代，并处理相关问题。

3. 实验内容与数据来源

应用 Jacobi 迭代法求解如下方程组：

$$\begin{pmatrix} 10 & -1 & 0 \\ -1 & 10 & -2 \\ 0 & -4 & 10 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 9 \\ 7 \\ 6 \end{pmatrix}$$

要求计算精度为10^{-7}。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

```
module jacobi
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!-----------------------------------------------------
!  Description : jacobi 迭代法模块
!
!-----------------------------------------------------
!  Parameters   :
!      1.    IMAX 最大允许迭代次数
!      2.    tol 误差容限
!  Contains     :
!      1.    solve 雅克比迭代法方法函数
!      2.
!-----------------------------------------------------
!  Post Script :
```

```
!     1.
!     2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
contains
subroutine solve(A,b,x,x0,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  雅克比迭代法函数
!              用于计算方程 AX=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.  A,b 意义即 AX=b
!      2.  x0 迭代初值
!      3.  N 方程的维数
!  Output parameters  :
!      1. x 方程的解
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::x1(N),x2(N)
x1=x0
!写入标题
  write(102,501)
  501 format(//,26x,'Jacobi 迭代法',//)
do k=1,IMAX
   do i=1,N
        s=0
        do j=1,N
          if (j==i) cycle
          s=s+A(i,j)*x1(j)
        end do
       x2(i)=(b(i)-s)/A(i,i)
   end do
! 这段程序用于判断精度，满足精度时退出循环
   dx2=0
   do i=1,N
    dx2=dx2+(x1(i)-x2(i))**2
   end do
   dx2=dsqrt(dx2)
   if (dx2<tol)  exit
!---------------------------------------
   x1=x2
  !记录迭代中间值
     write(102,*)x1
```

```
    !----
  end do
  x=x2
  end subroutine solve
  end module jacobi
  program  main
  !-------------------------------------program comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      :  2010-4-5
  !-----------------------------------------------------
  !  Purpose   :  采用雅克比迭代法计算线性方程
  !
  !-----------------------------------------------------
  !  In put data  files :
  !      1.
  !      2.
  !  Output data files  :
  !      1. Im_result.txt 计算的中间数据
  !      2.  result.txt 计算结果
  !-----------------------------------------------------
  use jacobi
  implicit real*8(a-z)
  integer,parameter::N=3
  real*8 ::A(N,N),b(N),x(N),x0(N)
    open(unit=101,file='result.txt')
    open(unit=102,file='Im_result.txt')
    x0=(/0d0,0d0,0d0/)
    b=(/9d0,7d0,6d0/)
    A=reshape((/10,-1,0,-1,10,-4,0,-2,10 /),(/3,3/))
    call solve(A,b,x,x0,N)
    write(101,501)x
    501 format(/,T20,'jacobi 迭代法',/,T10,'x(1)',T30,'x(2)',T50,
          'x(3)',/,3F20.15)
  end program main
```

6. 实验结论

程序运行后，产生两个数据文件，其中 result.txt 记录了计算结果，如图 2-1 所示。

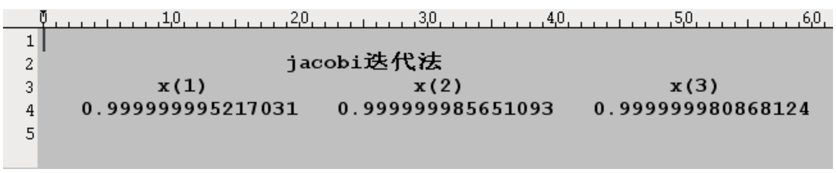


图 2-1　Jacobi 迭代法计算结果

文件 Im_result.txt 记录了中间数据，如表 2-1 所示。

表 2-1 Jacobi 迭代中间结果

迭代序列	x_1	x_2	x_3
1	0.90000000	0.70000000	0.60000000
2	0.97000000	0.91000000	0.88000000
3	0.99100000	0.97300000	0.96400000
4	0.99730000	0.99190000	0.98920000
5	0.99919000	0.99757000	0.99676000
6	0.99975700	0.99927100	0.99902800
7	0.99992710	0.99978130	0.99970840
8	0.99997813	0.99993439	0.99991252
9	0.99999344	0.99998032	0.99997376
10	0.99999803	0.99999410	0.99999213
11	0.99999941	0.99999823	0.99999764
12	0.99999982	0.99999947	0.99999929
13	0.99999995	0.99999984	0.99999979
14	0.99999998	0.99999995	0.99999994

2.2 Gauss-Seidel迭代法

1. 实验基本原理

雅可比迭代法较为简单，方便编程。在雅可比迭代过程中，每次计算出的最新结果，没有用在当次迭代中，而是在一次迭代循环结束后，下一次重新全部使用上次迭代结果。而高斯-赛德尔迭代则把每次迭代的最新结果更新到当前迭代步骤中。

对于方程 $\mathbf{Ax}=\mathbf{b}$，把系数矩阵做如下分裂

$$\mathbf{A}=\mathbf{D}(\mathbf{I}-\mathbf{L})-\mathbf{DU}$$

其中

$$\mathbf{D}=\mathrm{diag}\begin{pmatrix}a_{11} & a_{22} & \cdots & a_{nn}\end{pmatrix}$$

$$\mathbf{L}=\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ -\dfrac{a_{21}}{a_{22}} & 0 & 0 & \cdots & 0 & 0 \\ -\dfrac{a_{31}}{a_{33}} & -\dfrac{a_{32}}{a_{33}} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -\dfrac{a_{n1}}{a_{nn}} & -\dfrac{a_{n2}}{a_{nn}} & \cdots & \cdots & -\dfrac{a_{n,n-1}}{a_{nn}} & 0 \end{pmatrix}$$

$$
\mathbf{U}=\begin{pmatrix}
0 & -\dfrac{a_{12}}{a_{11}} & -\dfrac{a_{13}}{a_{11}} & \cdots & -\dfrac{a_{1,n-1}}{a_{11}} & -\dfrac{a_{1n}}{a_{11}} \\
0 & 0 & -\dfrac{a_{23}}{a_{22}} & \cdots & -\dfrac{a_{2,n-1}}{a_{22}} & -\dfrac{a_{2n}}{a_{22}} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & -\dfrac{a_{n-1,n}}{a_{n-1,n-1}} \\
0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
$$

高斯-赛德尔迭代格式为

$$\mathbf{x}_k=\mathbf{L}\mathbf{x}_k+\mathbf{U}\mathbf{x}_{k-1}+\mathbf{D}^{-1}\mathbf{b}$$

分量形式为

$$x_i^{(k)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k-1)}\right)$$

$$i=1,\cdots,n，\quad k=1,2,\cdots$$

上标 k 表示第 k 次迭代，下标 i 表示方程的第 i 个分量。从分量形式很容易看出计算原理。

2. 实验目的与要求

- 了解算法中各表达式的意义。
- 了解高斯-赛德尔迭代法的构造步骤。
- 熟悉高斯-赛德尔迭代法的矩阵形式与分量形式。
- 能够编程实现该算法。

3. 实验内容与数据来源

用高斯-赛德尔迭代法计算上一个实验中的方程，要求精度控制在 10^{-7} 。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

```
module gs
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!-----------------------------------------------------
!  Description : GS迭代法模块
!
!-----------------------------------------------------
!  Parameters  :
!      1.    IMAX最大允许迭代次数
!      2.    tol误差容限
!  Contains    :
!      1.    solve  GS迭代法方法函数
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
contains
subroutine solve(A,b,x,x0,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  GS迭代法函数
!              用于计算方程AX=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A,b 意义即 AX=b
!       2.  x0 迭代初值
!       3.  N 方程的维数
!  Output parameters  :
!       1. x 方程的解
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::x1(N),x2(N)
!写入标题
  write(102,501)
  501 format(//,18x,'G-S迭代法',//)
```

```
!迭代之前两值都设为初值
x1=x0
x2=x1
do k=1,IMAX
   do i=1,N
        s=0
        do j=1,N
       !-----------------------
       !这段为 GS 迭代法的核心部分
       !如果 j<i 则表示这些量已经更新过了，则下一个元素就用最新的量计算
       !如果 j>i 则还没有计算到这些量，所以就用上一次迭代的结果
        if (j<i) then
         s=s+A(i,j)*x2(j)
        else if (j>i) then
        s=s+A(i,j)*x1(j)
        end if
        end do
       !------------------------
       x2(i)=(b(i)-s)/A(i,i)
   end do
!这段程序用于判断精度，满足精度时退出循环
   dx2=0
   do i=1,N
    dx2=dx2+(x1(i)-x2(i))**2
   end do
   dx2=dsqrt(dx2)
   if (dx2<tol)  exit
!-----------------------------------------
   x1=x2
  !记录迭代中间值
     write(102,502)k,x1
     502 format(I3,3F12.8)
  !----
end do
x=x2
end subroutine solve
end module gs
program  main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-5
!-----------------------------------------------------
!  Purpose   :  采用 G-S 迭代法计算线性方程
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1. Im_result.txt 计算的中间数据
!      2.  result.txt 计算结果
!-----------------------------------------------------
use gs
implicit real*8(a-z)
```

```
    integer,parameter::N=3
    real*8 ::A(N,N),b(N),x(N),x0(N)
      open(unit=101,file='result.txt')
      open(unit=102,file='Im_result.txt')
      x0=(/0d0,0d0,0d0/)
      b=(/9d0,7d0,6d0/)
      A=reshape((/10,-1,0,-1,10,-4,0,-2,10 /),(/3,3/))
      call solve(A,b,x,x0,N)
      write(101,501)x
      501 format(/,T20,'G-S迭代法',/,T10,'x(1)',T30,'x(2)',T50,'x(3)',/,3F20.15)
    end program main
```

6. 实验结论

程序运行后生成两个数据文件，其中 result.txt 记录了计算结果，如图 2-2 所示。

```

                   G-S迭代法
         x(1)                x(2)                x(3)
   0.999999998995576   0.999999999096019   0.999999999638407

```

图 2-2　高斯-赛的人迭代法计算结果

文件 Im_result.txt 记录了详细的计算过程，如表 2-2 所示。

表 2-2　高斯-赛德尔迭代法计算的中间结果

迭代序列	x_1	x_2	x_3
1	0.90000000	0.79000000	0.91600000
2	0.97900000	0.98110000	0.99244000
3	0.99811000	0.99829900	0.99931960
4	0.99982990	0.99984691	0.99993876
5	0.99998469	0.99998622	0.99999449
6	0.99999862	0.99999876	0.99999950
7	0.99999988	0.99999989	0.99999996
8	0.99999999	0.99999999	1.00000000

因为在计算中，对每个分量的计算都用了最近更新的值，所以一般而言高斯-赛德尔迭代法要比雅克比迭代法快一些，不过这也不是绝对的。

2.3　逐次超松弛迭代法

1. 实验基本原理

有时候雅可比迭代或者高斯-赛德尔迭代收敛速度不够快，于是发展了逐次松弛迭代法，迭代格式如下

$$x_i^{k+1}=\frac{1}{a_{ii}}\left[(1-\omega)x_i^k+\omega\left(-\sum_{j=1}^{i-1}a_{ij}x_j^{k+1}-\sum_{j=i+1}^{n}a_{ij}x_j^k+b_i\right)\right]$$

$$i=1,\cdots,n\text{，}\quad k=1,2,\cdots$$

其中ω称为松弛因子。可以看出当$\omega=1$时即为高斯-赛德尔迭代，当松弛因子大于 1 时称为逐次超松弛迭代，松弛因子小于 1 时称为逐次低松弛迭代。不过，有时都统称为逐次超松弛迭代，或简称 SOR 迭代方法。

不过在实际应用时，松弛因子的选择往往是比较麻烦的事，对于一般情形，目前并未见有绝对好的松弛因子选择方法。这有时需要经验，甚至是反复试验。

2. 实验目的与要求

- 了解 SOR 迭代法的大致含义。
- 知道高斯-赛德尔迭代是 SOR 的特殊情况。
- 能够编程实现 SOR 迭代法。
- 分析计算的中间结果。

3. 实验内容与数据来源

采用逐次超松弛迭代法计算方程组

$$\begin{pmatrix}5&-1&-1&-1\\-1&10&-1&-1\\-1&-1&5&-1\\-1&-1&-1&10\end{pmatrix}\mathbf{x}=\begin{pmatrix}-4\\12\\8\\34\end{pmatrix}$$

取松弛因子$\omega=1.2$，迭代初值从$\mathbf{x}=\begin{pmatrix}0\\0\\0\\0\end{pmatrix}$出发。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

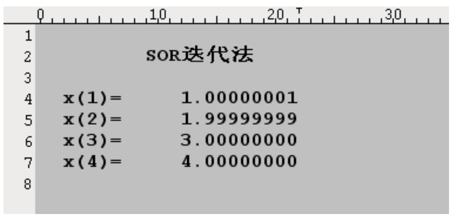
```
module sor
!---------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!----------------------------------------------------
!  Description : SOR 迭代法模块
!
!----------------------------------------------------
!  Parameters  :
!      1.    IMAX 最大允许迭代次数
!      2.    tol 误差容限
!      3.    omiga 松弛因子
!  Contains    :
!      1.    solve  SOR 迭代法方法函数
!      2.
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
real*8::omiga=1.2
contains
subroutine solve(A,b,x,x0,N)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!----------------------------------------------------
!  Purpose   :  SOR 迭代法函数
!               用于计算方程 AX=b
!----------------------------------------------------
!  Input  parameters  :
!       1.  A,b 意义即 AX=b
!       2.  x0 迭代初值
!       3.  N 方程的维数
!  Output parameters  :
!       1. x 方程的解
!       2.
!  Common parameters  :
!
!---------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::x1(N),x2(N)
!写入标题
```

```
   write(102,501)
   501 format(//,18x,'SOR 迭代法',//)
  !迭代之前两值都设为初值
  x1=x0
  x2=x1
  do k=1,IMAX
    do i=1,N
        s=0
        do j=1,N
       !-----------------------
       !如果 j<i 则表示这些量已经更新过了，则下一个元素就用最新的量计算
       !如果 j>i 则还没有计算到这些量，所以就用上一次迭代的结果
        if (j<i) then
         s=s+A(i,j)*x2(j)
        else if (j>i) then
        s=s+A(i,j)*x1(j)
        end if
        end do
       !------------------------
       x2(i)=(b(i)-s)*omiga/A(i,i)+(1-omiga)*x1(i)
    end do
  !这段程序用于判断精度，满足精度时退出循环
    dx2=0
    do i=1,N
     dx2=dx2+(x1(i)-x2(i))**2
    end do
    dx2=dsqrt(dx2)

    if (dx2<tol) exit
  !----------------------------------------
    x1=x2
   !记录迭代中间值
     write(102,502)k,x1
     502 format(I3,4F12.8)
   !----
  end do
  x=x2
  end subroutine solve
  end module sor
  program main
  !--------------------------------------program comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      :  2010-4-5
  !-----------------------------------------------------
  !  Purpose   :  采用 SOR 迭代法计算线性方程
  !
  !-----------------------------------------------------
  !  In put data  files :
  !      1.
  !      2.
  !  Output data files  :
  !      1. Im result.txt 计算的中间数据
  !      2.  result.txt 计算结果
  !-----------------------------------------------------
```

```
use sor
implicit real*8(a-z)
integer,parameter::N=4
real*8 ::A(N,N),b(N),x(N),x0(N)
  open(unit=101,file='result.txt')
  open(unit=102,file='Im_result.txt')
  x0=(/0d0,0d0,0d0,0d0/)
  b=(/-4,12,8,34/)
  A=reshape((/5,-1,-1,-1,&
              -1,10,-1,-1,&
              -1,-1,5,-1,&
              -1,-1,-1,10 /),(/4,4/))
  call solve(A,b,x,x0,N)
  write(101,501)x
  501 format(/,T10,'SOR 迭代法',//,&
                2x,'x(1)=',F15.8,/,&
                2x,'x(2)=',F15.8,/,&
                2x,'x(3)=',F15.8,/,&
                2x,'x(4)=',F15.8,/)
end program main
```

6. 实验结论

程序运行后生成两个数据文件，其中 result.txt 存放计算结果，如图 2-3 所示。



```
        SOR迭代法

x(1)=      1.00000001
x(2)=      1.99999999
x(3)=      3.00000000
x(4)=      4.00000000
```

图 2-3　逐次超松弛迭代法计算结果

文件 Im_result.txt 记录了计算的中间结果，如表 2-3 所示。

表 2-3　逐次超松弛迭代计算的中间结果

迭代序列	x_1	x_2	x_3	x_4
1	-0.96000004	1.32480005	2.00755208	4.36468242
2	1.07928838	2.06922277	3.32165609	3.98348357
3	1.07398931	2.03165092	2.95705848	4.01082713
4	0.98509090	1.98802700	3.00473511	3.99517693
5	1.00008719	2.00239451	2.99849105	4.00108134
6	1.00045462	1.99952434	3.00055626	3.99984796
7	0.99989193	2.00013067	2.99985768	4.00001604
8	1.00002267	1.99996143	3.00002850	3.99999830
9	0.99999264	2.00001005	2.99999454	4.00000001
10	1.00000257	1.99999764	3.00000115	4.00000016
11	0.99999923	2.00000054	2.99999975	3.99999991

（续表）

迭代序列	x_1	x_2	x_3	x_4
12	1.00000020	1.99999988	3.00000005	4.00000003
13	0.99999995	2.00000003	2.99999999	3.99999999

2.4　Richardson同步迭代法

1. 实验基本原理

Richardson 迭代法也是带参数的迭代方法，计算过程较为简单，迭代格式为

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \omega(\mathbf{Ax} - \mathbf{b}), k = 1, 2, \cdots$$

与松弛迭代法一样在 Richardson 迭代中，因子 ω 的选择也是较为麻烦的，未有一般方法给出最优的因子。

2. 实验目的与要求

- 了解 Richardson 迭代法的矩阵与分量形式。
- 能够编程实现 Richardson 迭代方法。
- 分析计算的中间结果。

3. 实验内容与数据来源

采用 Ricardson 同步迭代法计算方程组

$$\begin{pmatrix} 0.8 & -0.01 & 0.12 \\ -0.01 & 0.84 & 0.05 \\ 0.12 & 0.05 & 0.88 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

迭代初值选择 $\mathbf{x}_0 = (0,0,0)^T$， $\omega = 1.2$ 。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

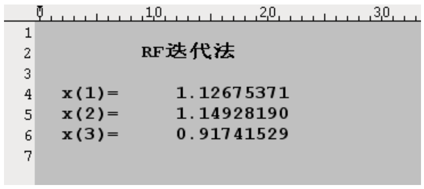
```
module RF
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!-----------------------------------------------------
!  Description : RF 迭代法模块
!
!-----------------------------------------------------
!  Parameters  :
!      1.    IMAX--最大允许迭代次数
!      2.    tol--误差容限
!      3.    omiga --RF 因子
!  Contains    :
!      1.    solve RF 迭代法方法函数
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
real*8::omiga=1.5
contains
subroutine solve(A,b,x,x0,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  RF 迭代法函数
!               用于计算方程 AX=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A,b 意义即 AX=b
!       2.  x0 迭代初值
!       3.  N 方程的维数
!  Output parameters  :
!       1. x 方程的解
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::x1(N),x2(N)
!写入标题
```

```
   write(102,501)
   501 format(//,18x,'RF 迭代法',//)
 x1=x0
 do k=1,IMAX
    do i=1,N
         s=0
         do j=1,N
         s=s+A(i,j)*x1(j)
         end do
        !------------------------
   !RF 迭代更新
        x2(i)=x1(i)-omiga*(s-b(i))
    end do
  !这段程序用于判断精度，满足精度时退出循环
    dx2=0
    do i=1,N
     dx2=dx2+(x1(i)-x2(i))**2
    end do
    dx2=dsqrt(dx2)
    if (dx2<tol)  exit
 !----------------------------------------
    x1=x2
   !记录迭代中间值
      write(102,502)k,x1
      502 format(I3,3F12.8)
   !----
 end do
 x=x2
 end subroutine solve
 end module RF
 program  main
 !-------------------------------------program comment
 !  Version   :  V1.0
 !  Coded by  :  syz
 !  Date     :  2010-4-5
 !----------------------------------------------------
 !  Purpose   :  采用 RF 迭代法计算线性方程
 !
 !----------------------------------------------------
 !  In put data  files :
 !      1.
 !      2.
 !  Output data files  :
 !      1. Im_result.txt 计算的中间数据
 !      2.  result.txt 计算结果
 !----------------------------------------------------
 use RF
 implicit real*8(a-z)
 integer,parameter::N=3
 real*8 ::A(N,N),b(N),x(N),x0(N)
   open(unit=101,file='result.txt')
   open(unit=102,file='Im_result.txt')
   x0=(/0d0,0d0,0d0/)
   b=(/1,1,1/)
   A=reshape((/0.8,-0.01,0.12,&
```

```
            -0.01,0.84,0.05,&
            0.12,0.05,0.88/),(/3,3/))
  call solve(A,b,x,x0,N)
  write(101,501)x
  501 format(/,T10,'RF迭代法',//,&
             2x,'x(1)=',F15.8,/,&
             2x,'x(2)=',F15.8,/,&
             2x,'x(3)=',F15.8,/)
end program main
```

6. 实验结论

程序运行后产生两个数据文件，其中 result.txt 记录了计算结果，如图 2-4 所示。



```
         RF迭代法

  x(1)=      1.12675371
  x(2)=      1.14928190
  x(3)=      0.91741529
```

图 2-4　Richardson 迭代法的计算结果

文件 Im_result.txt 记录了计算的中间结果，如表 2-4 所示。

表 2-4　迭代序列

迭代序列	x_1	x_2	x_3
1	1.50000000	1.50000000	1.50000000
2	0.95249998	1.02000006	0.63750002
3	1.21004999	1.20127502	1.04805000
4	1.08736011	1.12721554	0.85671939
5	1.14522671	1.15898045	0.94558382
6	1.11813426	1.14492474	0.90434885
7	1.13076421	1.15126546	0.92347486
8	1.12489065	1.14837187	0.91460559
9	1.12761843	1.14970130	0.91871801
10	1.12635257	1.14908813	0.91681133
11	1.12693975	1.14937157	0.91769531
12	1.12666745	1.14924038	0.91728549
13	1.12679371	1.14930114	0.91747548
14	1.12673517	1.14927299	0.91738740
15	1.12676231	1.14928604	0.91742824
16	1.12674973	1.14927999	0.91740931
17	1.12675556	1.14928279	0.91741808
18	1.12675286	1.14928149	0.91741401
19	1.12675411	1.14928210	0.91741590
20	1.12675353	1.14928182	0.91741502
21	1.12675380	1.14928195	0.91741543

（续表）

迭代序列	x_1	x_2	x_3
22	1.12675367	1.14928189	0.91741524
23	1.12675373	1.14928192	0.91741533

2.5　广义Richardson迭代法

1. 实验基本原理

广义的 Rrichardson 迭代法与 Rrichardson 迭代法类似，仅仅是每一个分量允许有独自的因子参数。迭代格式为

$$\mathbf{x}^{k+1}=\mathbf{x}^{k}-\mathbf{\Omega}\left(\mathbf{Ax}-\mathbf{b}\right),k=1,2,\cdots$$

其中

$$\mathbf{\Omega}=\begin{pmatrix}\omega_1 & 0 & \cdots & 0\\ 0 & \omega_2 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & \omega_n\end{pmatrix}$$

可以看到因子矩阵为对角矩阵，因子的选择较 Richardson 迭代有更大的自由度。

2. 实验目的与要求

- 了解广义 Richardson 迭代格式。
- 能够编程实现该算法。
- 分析迭代中间结果。

3. 实验内容与数据来源

采用广义的 Richardson 迭代法计算上一个实验中的方程，迭代初值选 $\mathbf{x}_0=\left(0,0,0\right)^T$，$\omega=\left(1.18,1.17,1.23\right)^T$，要求精度达到$10^{-7}$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	系数矩阵

（续表）

输入参变量	数据类型	变量说明
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

```
module GRF
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!-----------------------------------------------------
!  Description : 广义 RF 迭代法模块
!
!-----------------------------------------------------
!  Parameters  :
!      1.    IMAX--最大允许迭代次数
!      2.    tol--误差容限
!      3.
!  Contains    :
!      1.    solve RF 迭代法方法函数
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
contains
subroutine solve(A,b,x,x0,omiga,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  广义 RF 迭代法函数
!              用于计算方程 AX=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A,b 意义即 AX=b
!       2.  x0 迭代初值
!       3.  N 方程的维数
!       4. omiga 这里 omiga 为向量是 GRF 迭代因子（对角阵，简化为向量）
!  Output parameters  :
!       1. x 方程的解
!       2.
!  Common parameters  :
!
```

```
!-----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::x1(N),x2(N),omiga(N)
!写入标题
  write(102,501)
  501 format(//,18x,'GRF 迭代法',//)
x1=x0
do k=1,IMAX
   do i=1,N
        s=0
        do j=1,N
        s=s+A(i,j)*x1(j)
        end do
       !-----------------------
  !GRF 迭代更新
       x2(i)=x1(i)-omiga(i)*(s-b(i))
   end do
 !这段程序用于判断精度，满足精度时退出循环
   dx2=0
   do i=1,N
    dx2=dx2+(x1(i)-x2(i))**2
   end do
   dx2=dsqrt(dx2)
   if (dx2<tol)  exit
!---------------------------------------
   x1=x2

  !记录迭代中间值
     write(102,502)k,x1
     502 format(I3,3F12.8)
  !----
end do
x=x2
end subroutine solve
end module GRF
program  main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-5
!-----------------------------------------------------
!  Purpose   :  采用 GRF 迭代法计算线性方程
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1. Im_result.txt 计算的中间数据
!       2.  result.txt 计算结果
!-----------------------------------------------------
use GRF
```

```
    implicit real*8(a-z)
    integer,parameter::N=3
    real*8 ::A(N,N),b(N),x(N),x0(N),omiga(N)
      open(unit=101,file='result.txt')
      open(unit=102,file='Im result.txt')
      x0=(/0d0,0d0,0d0/)
      b=(/1,1,1/)
      A=reshape((/0.8,-0.01,0.12,&
                 -0.01,0.84,0.05,&
                 0.12,0.05,0.88/),(/3,3/))
      omiga=(/1.18,1.17,1.23/)
      call solve(A,b,x,x0,omiga,N)
      write(101,501)x
      501 format(/,T10,'GRF 迭代法',//,&
                    2x,'x(1)=',F15.8,/,&
                    2x,'x(2)=',F15.8,/,&
                    2x,'x(3)=',F15.8,/)
    end program main
```

6. 实验结论

程序运行后产生两个数据文件，其中 result.txt 记录了计算结果，如图 2-5 所示。

```
         GRF迭代法

  x(1)=      1.12675372
  x(2)=      1.14928191
  x(3)=      0.91741531
```

图 2-5　广义 Richardson 迭代法计算结果

文件 Im_result.txt 记录了计算的中间结果，如表 2-5 所示。

表 2-5　广义 Richardson 迭代的中间结果

迭代序列	x_1	x_2	x_3
1	1.17999995	1.16999996	1.23000002
2	1.08571799	1.13197503	0.88252501
3	1.12919196	1.15054519	0.92741151
4	1.12548970	1.14874738	0.91615404
5	1.12685522	1.14933171	0.91773867
6	1.12671420	1.14926503	0.91737061
7	1.12675763	1.14928377	0.91742585
8	1.12675246	1.14928137	0.91741374
9	1.12675386	1.14928197	0.91741565
10	1.12675367	1.14928189	0.91741525

2.6　acobi超松弛迭代法

1. 实验基本原理

雅可比超松弛迭代格式为

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \omega \mathbf{D}^{-1}\left(\mathbf{Ax} - \mathbf{b}\right), k = 1, 2, \cdots$$

其中

$$\mathbf{D} = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}$$

D 为原方程系数矩阵对角元素组成的对角阵。实际计算编程时是按照分量编写，而不是对对角阵求逆（对角阵的逆实为对角元素各自倒数）。可以看到该算法与广义的 Richardson 迭代法非常类似。甚至于，如果广义的 Richardson 迭代法每次的因子如果都相同的话，则两者可以互相转化。

2. 实验目的与要求

- 了解雅可比超松弛迭代法迭代格式。
- 比较与广义 Richardson 迭代法的异同。
- 能编程实现该算法。
- 分析计算的中间结果。

3. 实验内容与数据来源

采用雅克比超松弛迭代法计算方程组

$$\begin{pmatrix} 7 & 2 & 1 & -2 \\ 9 & 15 & 3 & -2 \\ -2 & -2 & 11 & 5 \\ 1 & 3 & 2 & 13 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 4 \\ 7 \\ -1 \\ 0 \end{pmatrix}$$

迭代初值取

$$\mathbf{x}_0 = \left(0, 0, 0, 0\right)^T$$

松弛因子取 $\omega = 1.04$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	数据类型	变量说明
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

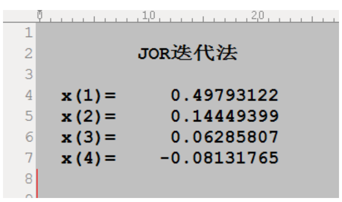
```
module JOR
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!-----------------------------------------------------
!  Description : 雅克比超松弛迭代法
!
!-----------------------------------------------------
!  Parameters  :
!      1.    IMAX--最大允许迭代次数
!      2.    tol--误差容限
!      3.    omiga --因子
!  Contains    :
!      1.    solve 迭代法方法函数
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
real*8::omiga=1.04
contains
subroutine solve(A,b,x,x0,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  雅克比超松弛迭代法函数
!              用于计算方程 AX=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A,b 意义即 AX=b
!       2.  x0 迭代初值
```

```
!       3.  N 方程的维数
!  Output parameters  :
!       1. x 方程的解
!       2.
!  Common parameters  :
!
!-----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::x1(N),x2(N)
!写入标题
  write(102,501)
  501 format(//,18x,'JOR 迭代法',//)
x1=x0
do k=1,IMAX
   do i=1,N
        s=0
        do j=1,N
        s=s+A(i,j)*x1(j)
        end do
       !-------------------------
  !迭代更新
       x2(i)=x1(i)-omiga*(s-b(i))/a(i,i)
   end do
!这段程序用于判断精度，满足精度时退出循环
   dx2=0
   do i=1,N
    dx2=dx2+(x1(i)-x2(i))**2
   end do
   dx2=dsqrt(dx2)

   if (dx2<tol)  exit
!----------------------------------------
   x1=x2
  !记录迭代中间值
     write(102,502)k,x1
     502 format(I3,4F12.8)
  !----
end do
x=x2
end subroutine solve
end module JOR
program  main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-5
!-----------------------------------------------------
!  Purpose   :  采用雅克比超松弛迭代法计算线性方程
!
!-----------------------------------------------------
!  In put data  files :
!       1.
```

```
!       2.
!  Output data files  :
!       1. Im_result.txt 计算的中间数据
!       2.  result.txt 计算结果
!-----------------------------------------------------
use JOR
implicit real*8(a-z)
integer,parameter::N=4
real*8 ::A(N,N),b(N),x(N),x0(N)
  open(unit=101,file='result.txt')
  open(unit=102,file='Im result.txt')
  x0=(/0d0,0d0,0d0,0d0/)
  b=(/4,7,-1,0/)
  A=reshape((/7,9,-2,1,&
             2,15,-2,3,&
             1,3,11,2,&
             -2,-2,5,13/),(/4,4/))
  call solve(A,b,x,x0,N)
  write(101,501)x
  501 format(/,T10,'JOR 迭代法',//,&
                2x,'x(1)=',F15.8,/,&
                2x,'x(2)=',F15.8,/,&
                2x,'x(3)=',F15.8,/,&
                2x,'x(4)=',F15.8)
end program main
```

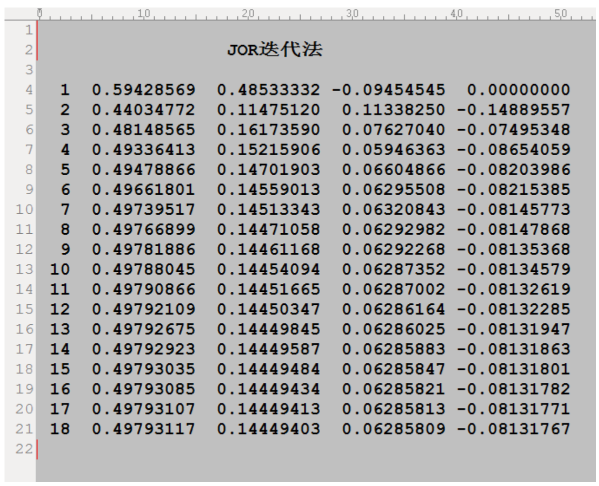
6. 实验结论

计算结果保存在文件 result.txt 中，如图 2-6 所示。
中间结果保存在文件中，如图 2-7 所示。



```
        JOR迭代法

  x(1)=      0.49793122
  x(2)=      0.14449399
  x(3)=      0.06285807
  x(4)=     -0.08131765
```

图 2-6　雅可比松弛迭代法



JOR迭代法

1	0.59428569	0.48533332	-0.09454545	0.00000000
2	0.44034772	0.11475120	0.11338250	-0.14889557
3	0.48148565	0.16173590	0.07627040	-0.07495348
4	0.49336413	0.15215906	0.05946363	-0.08654059
5	0.49478866	0.14701903	0.06604866	-0.08203986
6	0.49661801	0.14559013	0.06295508	-0.08215385
7	0.49739517	0.14513343	0.06320843	-0.08145773
8	0.49766899	0.14471058	0.06292982	-0.08147868
9	0.49781886	0.14461168	0.06292268	-0.08135368
10	0.49788045	0.14454094	0.06287352	-0.08134579
11	0.49790866	0.14451665	0.06287002	-0.08132619
12	0.49792109	0.14450347	0.06286164	-0.08132285
13	0.49792675	0.14449845	0.06286025	-0.08131947
14	0.49792923	0.14449587	0.06285883	-0.08131863
15	0.49793035	0.14449484	0.06285847	-0.08131801
16	0.49793085	0.14449434	0.06285821	-0.08131782
17	0.49793107	0.14449413	0.06285813	-0.08131771
18	0.49793117	0.14449403	0.06285809	-0.08131767

图 2-7　计算的中间结果

可以看到迭代 18 次满足设定的精度需求。

2.7 最速下降法

1. 实验基本原理

除了前面介绍的迭代方法之外，还可以从变分的角度给出一些列迭代算法，这一小节的最速下降法与下一节的共轭梯度法即属于此类算法。这里给出最速下降法的算法描述。

输入：方程系数矩阵 $\mathbf{A}$，方程右向量 $\mathbf{b}$，最大允许的迭代次数 IMAX，误差容限 TOL，迭代初值 $\mathbf{x}_0$。

输出：方程的根 $\mathbf{x}$，以及迭代次数等信息。

01 $\mathbf{x}_1 = \mathbf{x}_0$。
02 对 $k = 1, IMAX$，做 03~07 处理。
03 $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}_0$。
04 $\alpha = \dfrac{(\mathbf{r},\mathbf{r})}{(\mathbf{Ar},\mathbf{r})}$。
05 $\mathbf{x}_2 = \mathbf{x}_1 + \alpha\mathbf{r}$。
06 如果 $|\alpha\mathbf{r}| < TOL$ 则退出循环。
07 $\mathbf{x}_1 = \mathbf{x}_2$。
08 $\mathbf{x} = \mathbf{x}_2$。
09 结束。

2. 实验目的与要求

- 了解最速下降法的意义。
- 从最优化的角度考虑最速下降法。
- 能够编程实现该方法，并计算相关方程。

3. 实验内容与数据来源

采用最速下降算法计算上一实验中的方程组，即

$$\begin{pmatrix} 7 & 2 & 1 & -2 \\ 9 & 15 & 3 & -2 \\ -2 & -2 & 11 & 5 \\ 1 & 3 & 2 & 13 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 4 \\ 7 \\ -1 \\ 0 \end{pmatrix}$$

迭代初值取

$$\mathbf{x}_0 = (0,0,0,0)^T$$

要求精度达到10^{-7}。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

```
module sd
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!-----------------------------------------------------
!  Description : 最速下降法
!
!-----------------------------------------------------
!  Parameters  :
!      1.    IMAX--最大允许迭代次数
!      2.    tol--误差容限
!
!  Contains    :
!      1.    solve 迭代法方法函数
!      2.
!      3.    dr(r,N) 计算向量长度平方函数
!      4.     Ar(A,r,N)  计算矩阵乘以向量函数，返回向量
!      5.    rAr(A,r,N)   计算（Ar,r）函数
!-----------------------------------------------------
!  Post Script :
!      1.    里面的三个函数，可以简化程序，同时可以用在其他地方
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
contains
subroutine solve(A,b,x,x0,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  最速下降法函数
!                用于计算方程 AX=b
```

```
!-----------------------------------------------------
!  Input  parameters  :
!      1.  A,b 意义即 AX=b
!      2.  x0 迭代初值
!      3.  N 方程的维数
!  Output parameters  :
!      1. x 方程的解
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::r(N)
real*8::x1(N),x2(N)
!写入标题
  write(102,501)
  501 format(//,18x,'最速下降法',//)
x1=x0
do k=1,IMAX
    r=b-Ar(A,x1,N)
    temp1=dr(r,N)
    temp2=rAr(A,r,N)
    afa=temp1/temp2
    x2=x1+afa*r
     !记录迭代中间值
     write(102,502)k,x2
     502 format(I3,4F12.8)
    !----
    ! 判断迭代停止标准，实际上 x1-x2 就直接等于 afa*r
    dx2=dr(afa*r,N)
    dx=dsqrt(dx2)
    if (dx<tol) exit
    !---------
    !更新结果
    x1=x2
end do
     x=x2
end subroutine solve
function dr(r,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算向量长度平方 (r,r)
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.     r 向量
!      2.     N 维数
!  Output parameters  :
!      1.     dr 长度平方
```

```
!       2.
!  Common parameters  :
!
!-----------------------------------------------------
implicit real*8(a-z)
integer::N,i
real*8::r(N),dr
s=0
do i=1,N
  s=s+r(i)**2
end do
dr=s
end function dr
function Ar(A,r,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  !计算 A*r,返回 N 维向量
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.     r 向量
!       2.     N 维数
!       3.     A 矩阵
!  Output parameters  :
!       1.     Ar 返回向量
!       2.
!  Common parameters  :
!
!-----------------------------------------------------
implicit real*8(a-z)
integer::i,N
real*8::A(N,N),r(N),temp(N),Ar(N)
temp=0
do i=1,N
  do j=1,n
    temp(i)=temp(i)+A(i,j)*r(j)
  end do
end do
Ar=temp
end function ar
function v1v2(v1,v2,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-7-29
!-----------------------------------------------------
!  Purpose     : 向量点乘  v1v2=v1(1)*v2(1)+v1(2)*v(2)+...
!
!  Post Script :
!       1.
!       2.
!       3.
```

```
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.   v1,v2 向量
!      2.   N 向量维数
!  Output parameters  :
!      1.   v1,v2 向量点乘值
!      2.
!  Common parameters  :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n
real*8::v1(n),v2(n)
integer::i
v1v2=0
do i=1,n
 v1v2=v1v2+v1(i)*v2(i)
end do
end function v1v2
function rAr(A,r,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  !计算（Ar,r）,返回标量
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.     r 向量
!      2.     N 维数
!      3.     A 矩阵
!  Output parameters  :
!      1.     Ar 返回标量
!      2.
!  Common parameters  :
!----------------------------------------------------
implicit real*8(a-z)
integer::i,N
real*8::A(N,N),r(N),temp(N)
temp=Ar(A,r,N)
rAr=v1v2(r,temp,N)
end function rAr
end module sd
program  main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-5
!-----------------------------------------------------
!  Purpose   :  采用最速下降法计算线性方程
!
!-----------------------------------------------------
```

```
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1. Im result.txt 计算的中间数据
!       2.  result.txt 计算结果
!------------------------------------------------------
use sd
implicit real*8(a-z)
integer,parameter::N=4
real*8  ::A(N,N),b(N),x(N),x0(N)
  open(unit=101,file='result.txt')
  open(unit=102,file='Im_result.txt')
  x0=(/0d0,0d0,0d0,0d0/)
  b=(/4,7,-1,0/)
  A=reshape((/7,9,-2,1,&
           2,15,-2,3,&
           1,3,11,2,&
           -2,-2,5,13/),(/4,4/))
  call solve(A,b,x,x0,N)
  write(101,501)x
  501 format(/,T10,'最速下降法',//,&
              2x,'x(1)=',F15.8,/,&
              2x,'x(2)=',F15.8,/,&
              2x,'x(3)=',F15.8,/,&
              2x,'x(4)=',F15.8)
end program main
```

6. 实验结论

程序运行后生成两个数据文件，result.txt 记录了计算结果，如图 2-8 所示。Im_result.txt 记录了中间数据，如图 2-9 所示。

```

        最速下降法

  x(1)=      0.49793098
  x(2)=      0.14449418
  x(3)=      0.06285808
  x(4)=     -0.08131769

```

图 2-8　最速下降法计算结果

```

              最速下降法中间结果

  1  0.22699914  0.39724850 -0.05674979  0.00000000
  2  0.48722582  0.26793750  0.07897996 -0.20299286
  3  0.46024344  0.13991072  0.12004014 -0.12857240
  4  0.47343110  0.15533749  0.06814703 -0.06862095
  5  0.48506856  0.15995189  0.05794393 -0.08119991
  6  0.49346858  0.14673524  0.06561006 -0.08449512
  7  0.49560735  0.14575776  0.06332851 -0.08043040
  8  0.49672885  0.14593080  0.06245394 -0.08146980
  9  0.49741529  0.14473586  0.06314428 -0.08150894
 10  0.49773394  0.14470655  0.06278921 -0.08128803
 11  0.49790042  0.14453033  0.06288747 -0.08139363
 12  0.49789762  0.14449320  0.06289240 -0.08133149
 13  0.49791341  0.14450975  0.06285833 -0.08131818
 14  0.49792990  0.14449571  0.06285754 -0.08132230
 15  0.49792971  0.14449426  0.06285950 -0.08131849
 16  0.49793048  0.14449461  0.06285797 -0.08131753
 17  0.49793119  0.14449423  0.06285799 -0.08131791
 18  0.49793115  0.14449397  0.06285815 -0.08131772
 19  0.49793120  0.14449399  0.06285806 -0.08131761
 20  0.49793123  0.14449399  0.06285804 -0.08131764

```

图 2-9　最速下降法计算的迭代序列

迭代次数与方程系数有关，也与设置的精度有关系，同等情况下，一般设置精度越高，迭代次数会更多。

2.8 共轭梯度法

1. 实验基本原理

共轭梯度法又称共轭斜量法，对于系数矩阵是对称正定的矩阵线性方程组，共轭梯度法算法如下：

01 设置初值 $\mathbf{x}^0$。

02 $\mathbf{r}^0 = \mathbf{b} - \mathbf{A}\mathbf{x}^0, \mathbf{p}^0 = \mathbf{r}^0$

03 对于 $k = 1, 2, 3 \ldots$

$$\alpha_k = \frac{\left(\mathbf{r}^k, \mathbf{r}^k\right)}{\left(\mathbf{p}^k, \mathbf{A}\mathbf{p}^k\right)}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}^k$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha_k \mathbf{A}\mathbf{p}^k$$

$$\beta_k = \frac{\left(\mathbf{r}^{k+1}, \mathbf{r}^{k+1}\right)}{\left(\mathbf{r}^k, \mathbf{r}^k\right)}$$

$$\mathbf{p}^{k+1} = \mathbf{r}^{k+1} + \beta_k \mathbf{p}^k$$

在03中，可以设置循环 k 的最大允许迭代次数，然后在03内部，给出精度判断，当满足精度时，退出循环。

2. 实验目的与要求

- 了解共轭梯度法适用范围。
- 了解共轭梯度法的基本原理。
- 能够理解算法中公式的含义。
- 能够编程实现共轭梯度法计算相关线性方程问题。

3. 实验内容与数据来源

采用共轭梯度法计算以下方程

$$\begin{pmatrix} 5 & 7 & 6 & 5 \\ 7 & 10 & 8 & 7 \\ 6 & 8 & 10 & 9 \\ 5 & 7 & 9 & 10 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 62 \\ 87 \\ 91 \\ 90 \end{pmatrix}$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	计算结果
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	系数矩阵
B	REAL*8(N)	A*X=B 中的右向量
N	INTEGER	方程维数
X0	REAL*8(N)	迭代初值

5. 程序代码

```
module conj
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-5
!-----------------------------------------------------
!  Description : 共轭梯度法
!
!-----------------------------------------------------
!  Parameters  :
!      1.    IMAX--最大允许迭代次数
!      2.    tol--误差容限
!
!  Contains    :
!      1.    solve 迭代法方法函数
!      2.
!      3.    dr(r,N) 计算向量长度平方函数
!      4.     Ar(A,r,N)  计算矩阵乘以向量函数，返回向量
!      5.    rAr(A,r,N)   计算（Ar,r）函数
!-----------------------------------------------------
!  Post Script :
!      1.    里面的三个函数，可以简化程序，同时可以用在其他地方
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::IMAX=200
real*8::tol=1d-7
contains
subroutine solve(A,b,x,x0,N)
!----------------------------------subroutine  comment
!  Version   :  V1.0
```

```
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  共轭梯度法
!              用于计算方程 AX=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A,b 意义即 AX=b
!       2.  x0 迭代初值
!       3.  N 方程的维数
!  Output parameters  :
!       1. x 方程的解
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j,k
real*8::A(N,N),b(N),x(N),x0(N)
real*8::r0(N),r1(N),p0(N),p1(N)
real*8::x1(N),x2(N)
!写入标题
  write(102,501)
  501 format(//,18x,'共轭梯度法中间结果',//)
x1=x0
r0=b-Ar(A,x1,N)
p0=r0
do k=1,IMAX
    tmp1=dr(r0,N)
    tmp2=rAr(A,p0,N)
    afa=tmp1/tmp2
    x2=x1+afa*p0
     !记录迭代中间值
     write(102,502)k,x2
     502 format(I3,4F12.8)
    !如果 r0 接近于，则停止迭代
    !该部分算法在《数值分析原理》（李庆扬、关治、
    !白峰彬编）中叙述较为详细
    dr_s=dsqrt(dr(r0,N))
    if (dr_s<tol) exit
    r1=r0-afa*Ar(A,p0,N)
    tmp3=dr(r1,N)
    beta=tmp3/tmp1
    p1=r1+beta*p0
    !全部更新
    r0=r1
    p0=p1
    x1=x2
end do
     x=x2
end subroutine solve
function dr(r,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
```

```
! Coded by  :  syz
! Date      :
!--------------------------------------------------------
! Purpose   :  计算向量长度平方 (r,r)
!
!--------------------------------------------------------
! Input  parameters  :
!       1.     r 向量
!       2.     N 维数
! Output parameters  :
!       1.     dr 长度平方
!       2.
! Common parameters  :
!
!-------------------------------------------------------
implicit real*8(a-z)
integer::N,i
real*8::r(N),dr
s=0
do i=1,N
  s=s+r(i)**2
end do
dr=s
end function dr
function Ar(A,r,N)
!---------------------------------subroutine  comment
! Version   :  V1.0
! Coded by  :  syz
! Date      :
!--------------------------------------------------------
! Purpose   :  !计算 A*r,返回 N 维向量
!
!--------------------------------------------------------
! Input  parameters  :
!       1.     r 向量
!       2.     N 维数
!       3.     A 矩阵
! Output parameters  :
!       1.     Ar 返回向量
!       2.
! Common parameters  :
!
!-------------------------------------------------------
implicit real*8(a-z)
integer::i,N
real*8::A(N,N),r(N),temp(N),Ar(N)
temp=0
do i=1,N
  do j=1,n
    temp(i)=temp(i)+A(i,j)*r(j)
  end do
end do
Ar=temp
end function ar
function v1v2(v1,v2,N)
```

```
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-7-29
!-----------------------------------------------------
!  Purpose     : 向量点乘  v1v2=v1(1)*v2(1)+v1(2)*v(2)+...
!
!  Post Script :
!      1.
!      2.
!      3.
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.   v1,v2 向量
!      2.   N 向量维数
!  Output parameters  :
!      1.   v1,v2 向量点乘值
!      2.
!  Common parameters  :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n
real*8::v1(n),v2(n)
integer::i
v1v2=0
do i=1,n
 v1v2=v1v2+v1(i)*v2(i)
end do
end function
function rAr(A,r,N)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  !计算（Ar,r）,返回标量
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.     r 向量
!      2.     N 维数
!      3.     A 矩阵
!  Output parameters  :
!      1.     Ar 返回标量
!      2.
!  Common parameters  :
!----------------------------------------------------
implicit real*8(a-z)
integer::i,N
real*8::A(N,N),r(N),temp(N)
temp=Ar(A,r,N)
rAr=v1v2(r,temp,N)
```

```
end function rAr
end module conj
program  main
!------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-5
!-----------------------------------------------------
!  Purpose   :  采用共轭梯度计算线性方程
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1. Im_result.txt 计算的中间数据
!       2.  result.txt 计算结果
!-----------------------------------------------------
use conj
implicit real*8(a-z)
integer,parameter::N=4
real*8 ::A(N,N),b(N),x(N),x0(N)
  open(unit=101,file='result.txt')
  open(unit=102,file='Im_result.txt')
 !迭代初值
  x0=(/0d0,0d0,0d0,0d0/)
!系数
  b=(/62d0,87d0,91d0,90d0/)
  A=reshape((/5d0,7d0,6d0,5d0,&
             7d0,10d0,8d0,7d0,&
             6d0,8d0,10d0,9d0,&
             5d0,7d0,9d0,10d0/),(/4,4/))
! 调用函数
  call solve(A,b,x,x0,N)
  write(101,501)x
  501 format(/,T10,'共轭梯度法',//,&
               2x,'x(1)=',F15.8,/,&
               2x,'x(2)=',F15.8,/,&
               2x,'x(3)=',F15.8,/,&
               2x,'x(4)=',F15.8)
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，如图 2-10 所示。

在文件 Im_result.txt 中还记录了计算的中间结果，如图 2-11 所示。

```
        共轭梯度法

x(1)=      2.00000000
x(2)=      3.00000000
x(3)=      1.00000000
x(4)=      5.00000000
```

图 2-10　共轭梯度法计算结果

```
                 共轭梯度法中间结果

1  2.04792193  2.87369690  3.00582089  2.97278989
2  1.56058255  2.51989545  2.58844050  4.12469697
3  1.53689971  3.27970581  1.11628376  4.93103148
4  2.00000000  3.00000000  1.00000000  5.00000000
5  2.00000000  3.00000000  1.00000000  5.00000000
```

图 2-11　共轭梯度法中间结果

从中间结果可以看到收敛速度还是较快的。

本章小结

本章介绍了常用的线性方程组的迭代方法。其中主要包括雅可比迭代、高斯赛-德尔迭代、逐次超松弛迭代等。还介绍了基于变分原理的最速下降与共轭梯度迭代方法。这些方法各有特点，实际应用时需要视情况而定。需要说明的是，一般在采用迭代法进行数值运算时，我们首先要分析迭代法是否收敛，否则计算可能是没有意义的。就线性方程组迭代法而言，每种方法都有自己的收敛条件，关于这些条件读者可以参阅相关数学手册，这里从略。

第 3 章

◂最小二乘与数据拟合▸

最小二乘法广泛出现在工程技术实践中，因为实际应用中，相当多的问题都是有足够冗余的测量数据，这就需要采用最小二乘估计方法。最小二乘法是最基础、也最重要的参数估计方法。即便是现代出现了名目繁多的参数估计方法，很多方法都和最小二乘有密切联系，如现在比较流行的多传感器数据融合估计，实际上都可以通过最小二乘法得到比较不错的解释。数值代数以及统计学的发展赋予了最小二乘许多新的意义和内涵。

QR 分解是本章的一个重点，除了计算最小二乘之外，QR 分解也是特征值问题的重要手段，其中主要介绍了 Houlseholder 变换和修正的 Gram-Shimdt 正交化方法。关于 Givens 变换因其计算效率较低，故而这里不作介绍。

3.1　Cholesky分解法计算最小二乘

1. 实验基本原理

对于带误差的超定线性方程组

$$\begin{cases} y_1 = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n + \eta_1 \\ y_2 = a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n + \eta_2 \\ \quad\vdots \\ y_m = a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n + \eta_m \end{cases}$$

可以把上述方程用矩阵形式写为

$$\mathbf{Ax = b}$$

是实数域上（m，n）阶的方程。由线性方程组的数值方法一章中知道，上述方程组有解的充分必要条件是方程组的系数矩阵的秩与增广矩阵的秩相同，即

$$\mathbf{rank}(\mathbf{A}) = \mathbf{rank}([\mathbf{A,b}])$$

但有时候方程组不满足上述条件，于是就没有通常意义下的解，这样的方程组称为矛盾方程组。

如果存在 $\mathbf{x}_s$，使

$$f(\mathbf{x}) = \|\mathbf{Ax} - \mathbf{b}\|_2$$

在 $\mathbf{x} = \mathbf{x}_s$ 时达到最小，则 $\mathbf{x}_s$ 称为方程 $\mathbf{Ax} = \mathbf{b}$ 的最小二乘解。$\mathbf{x}_s$ 是方程组最小二乘解的充分必要条件是 $\mathbf{x}_s$ 是方程组

$$\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$$

的解。方程 $\mathbf{A}^T\mathbf{Ax} = \mathbf{A}^T\mathbf{b}$ 即称为法方程，通过求解法方程来计算最小二乘解是一种可行的方法。在线性方程组直接方法中介绍的 Cholesky 分解法是比较适用于解法方程的。

2. 实验目的与要求

- 了解最小二乘的基本原理
- 知道常用的最小二乘解法
- 对于超定方程能够写出法方程
- 能够编程求出法方程给出最小二乘解

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(M)	方程的解
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,M)	系数矩阵
b	REAL*8(N)	右向量
N	INTEGER	A 的行数
M	INTEGER	A 的列数

3. 实验内容与数据来源

计算超定方程 $\mathbf{Ax} = \mathbf{b}$ 在最小二乘意义下的解。其中

$$\mathbf{A} = \begin{pmatrix} -4.6429 & -1.0777 & -4.5383 \\ 3.4913 & 1.5548 & -4.0287 \\ 4.3399 & -3.2881 & 3.2346 \\ 1.7874 & 2.0605 & 1.9483 \\ 2.5774 & -4.6817 & -1.8290 \\ 2.4313 & -2.2308 & 4.5022 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} -7.0294 \\ -2.1417 \\ -0.3499 \\ 5.1209 \\ -8.0224 \\ 2.5942 \end{pmatrix}$$

5. 程序代码

```
module  normal
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   超定方程的最小二乘计算模块
!
!-----------------------------------------------------
!  Post Script :
!      1.       模块主函数 solve
!      2.
!-----------------------------------------------------
contains
subroutine  solve(A,b,x,N,M)
!---------------------------------subroutine  comment
!  Version    :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose    :  计算超定方程的最小二乘问题
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.       A    (N,M)      N>M
!       2.       b    (N)
!  Output parameters  :
!       1.       x   (M)
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.      方法是通过求解法方程
!       2.      而法方程的求解基于 Cholesky 分解
!       3.    程序中调用了个基础函数
!
!        transpose 矩阵转置，这个是非 IMSL 函数，Fortran 自带的，
!         可以直接使用
!        Matmul    矩阵相乘，注意矩阵的维数
!----------------------------------------------------
implicit real*8(a-z)
integer::N,M,P
real*8::A(N,M),b(N),x(M)
real*8::AT(M,N)
real*8::ATA(M,M),ATb(M)
integer::i,j,k
! 转置 系统函数
AT=TRANSPOSE(A)
! 矩阵相乘，系统函数，注意维数匹配
ATA=MATMUL(AT,A)
```

```
ATb=MATMUL(AT,b)
!调用 Cholesky 分解方法计算法方程
call chol_eq(ATA,ATb,x,M)
end subroutine solve
subroutine chol eq(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  用 cholesky 分解方法解方程
!
!-----------------------------------------------------
implicit real*8(a-z)
integer::N
real*8::A(N,N),b(N),x(N)
real*8::L(N,N),y(N),LT(N,N)
!LT 为 L 的转置矩阵
integer::i,j
call  chol(A,L,N)
call  downtri(L,b,y,N)
do i=1,N
 do j=1,N
   LT(i,j)=L(j,i)
 end do
end do
call uptri(LT,y,x,N)  !这一步已经算出了 x
end subroutine chol_eq
subroutine  chol(A,L,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  Cholesky 分解子程序
!-----------------------------------------------------
integer::N
real*8::A(N,N),L(N,N)
integer::i,j,k
L=0
L(1,1)=dsqrt(a(1,1))
L(2:,1)=a(2:,1)/L(1,1)
do j=2,N
   s=0
   do k=1,j-1
   s=s+L(j,k)**2
   end do
   L(j,j)=dsqrt(a(j,j)-s)
   !注意 i 范围
   do i=j+1,N
       s=0
       do k=1,j-1
         s=s+L(i,k)*L(j,k)
       end do
       L(i,j)=(a(i,j)-s)/L(j,j)
```

```
    end do
  end do
  end subroutine chol
  subroutine uptri(A,b,x,N)
  !---------------------------------subroutine  comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      :  2010-4-8
  !-----------------------------------------------------
  !  Purpose   :  上三角方程组的回带方法
  !               Ax=b
  !-----------------------------------------------------
  implicit real*8(a-z)
  integer::i,j,k,N
  real*8::A(N,N),b(N),x(N)
  x(N)=b(N)/A(N,N)
  !回带部分
  do i=n-1,1,-1
      x(i)=b(i)
     do j=i+1,N
      x(i)=x(i)-a(i,j)*x(j)
     end do
      x(i)=x(i)/A(i,i)
  end do
  end subroutine uptri
  subroutine downtri(A,b,x,N)
  !---------------------------------subroutine  comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      :  2010-4-9
  !-----------------------------------------------------
  !  Purpose   :  下三角方程组的回带方法
  !               Ax=b
  !-----------------------------------------------------
  implicit real*8(a-z)
  integer::i,j,N
  real*8::A(N,N),b(N),x(N)
  x(1)=b(1)/a(1,1)
  do k=2,N
     x(k)=b(k)
     do i=1,k-1
        x(k)=x(k)-a(k,i)*x(i)
     end do
     x(k)=x(k)/a(k,k)
  end do
  end subroutine downtri
  end module normal
  module driver
  !----------------------------------------module coment
  !  Version     :  V1.0
  !  Coded by    :  syz
  !  Date        :
  !-----------------------------------------------------
  !  Description :   驱动程序模块
  !
```

```
!--------------------------------------------------------
!  Contains    :
!      1.    dri_main  读文件，并调用方法函数
!      2.
!--------------------------------------------------------
!  Post Script :
!      1.
!      2.
!--------------------------------------------------------
contains
subroutine dri main(N,M)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!--------------------------------------------------------
!  Purpose   :  驱动程序模块
!
!-------------------------------------------------------
!  Post Script :
!       1.      可以计算任意 N,M 阶最小二乘 N>M  由用户输入
!       2.
!-------------------------------------------------------
!引用方法模块
use normal
implicit real*8(a-z)
integer::N,M,i
real*8::A(N,M),b(N),x(M)
!读入系数 A 矩阵
read(11,*)((A(i,j),j=1,M),i=1,N)
!读入向量 b
read(11,*)b
!调用方法函数
 call solve(A,b,x,N,M)
  write(12,101)
101 format(T5,'最小二乘解为：',/)
do i=1,m
 write(12,'(T5,F10.6)')x(i)
end do
end subroutine dri_main
end  module driver
program main
!---------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-10
!--------------------------------------------------------
!  Purpose   :  生成计算最小二乘的软件，方法是采用解法方程
!               法方程的计算是基于 cholesky 分解
!--------------------------------------------------------
!  Post Script :
!       1.       软件适用于 超定方程，系数由文件读入
!       2.       由主函数启动驱动程序模块，驱动程序调用方法函数
!--------------------------------------------------------
use driver
```

```
    integer::N,M
    open(unit=11,file='fin.txt')
    open(unit=12,file='fout.txt')
    read(11,*)
    !读入 A 矩阵
    !读入方程维数系数
    read(11,*)N,M
    !调用驱动函数
    call dri_main(N,M)
    end program main
```

6. 实验结论

先准备数据文件 fin.txt，格式如图 3-1 所示。

程序运行后，计算结果保存在文件 fout.txt 中，如图 3-2 所示。

```
           法方程解最小二乘    版本: 1.1

  6    3        : 矩阵维数信息   Ax=b  A(N,M)

  -4.6429   -1.0777   -4.5383
   3.4913    1.5548   -4.0287
   4.3399   -3.2881    3.2346
   1.7874    2.0605    1.9483
   2.5774   -4.6817   -1.8290
   2.4313   -2.2308    4.5022      :END MATRIX A


  -7.0294
  -2.1417
  -0.3499
   5.1209
  -8.0224
   2.5942    : END VECTOR b

  P.S: 可以处理一般最小二乘，作为验证程序
```

图 3-1　输入数据文件格式

```
     最小二乘解为:

       0.097598
       1.323411
       1.141427
```

图 3-2　计算结果

这里补充说一下文中数据来源，系数矩阵 $\mathbf{A}$ 由计算机产生 6×3 的在 $[-5,5]$ 区间的正态分布随机数，预给定向量 $\mathbf{x}$ 一个值，然后令 $\mathbf{b}=\mathbf{Ax}+0.2\times\text{randn}(6,1)$，其中 randn(6,1) 表示 6×1 维的标准正态分布。

3.2　Householder镜像变换之QR分解

1. 实验基本原理

对于单个向量 $\mathbf{x}$，欲用镜像变换使之成为

$$\mathbf{Hx}=\mathbf{w}=\|\mathbf{x}\|\mathbf{e}_1,\mathbf{e}_1=(1,0,\cdots)^T$$

可以令

$$\mathbf{u}=\mathbf{w}-\mathbf{x}$$

继而可以构造镜像变换矩阵

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{\|\mathbf{u}\|_2^2}$$

$\|\mathbf{u}\|_2^2$ 表示向量之 2 范数的平方，无泛函基础的读者可直接理解为向量之长度平方。

对于矩阵 **A** 做 QR 分解，即连续使用 Houlseholder 变换。如第一次使用正交变换后，使之成为如下形式：

$$\mathbf{H}_1\mathbf{A} = \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \cdots & * \end{pmatrix}$$

第二次变换使之成为如下形式：

$$\mathbf{H}_2\mathbf{H}_1\mathbf{A} = \begin{pmatrix} * & * & * & \cdots & * \\ 0 & * & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ 0 & 0 & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & * & \cdots & * \end{pmatrix}$$

如此反复。这一过程看上去似乎比较简单，实际编程时需要较多的技巧。

2. 实验目的与要求

- 了解 Householder 变换的意义。
- 能给出针对两个矢量做镜像变换时候的变换矩阵。
- 能够会利用作者编写的程序对一般矩阵做 QR 分解。
- 最好会自己编写程序实现 QR 分解。

3. 实验内容与数据来源

采用 Householder 变换求矩阵

$$\mathbf{A} = \begin{pmatrix} 4 & 4 & 0 \\ 3 & 3 & -1 \\ 0 & 1 & 1 \end{pmatrix}$$

的 QR 分解。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
Q	REAL*8(M,M)	变换矩阵
R	REAL*8(M,N)	上三角矩阵
输入参变量	**数据类型**	**变量说明**
A	REAL*8(M,N)	欲求分解之矩阵
M	INTEGER	A 的行数
N	INTEGER	A 的列数

5. 程序代码

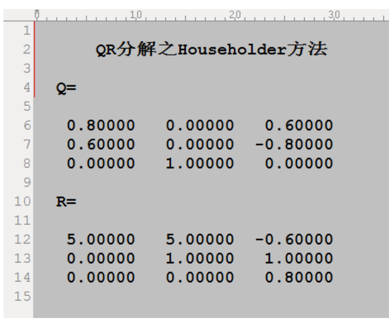
```
module Hou qr
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :  2009.07.30
!-----------------------------------------------------
!  Description :   Houlseholder 求 QR 分解之模块
!
!  Post Script :
!     1.
!     2.
!
!-----------------------------------------------------
contains
subroutine solve(A,Q,R,M,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2009.07.30
!-----------------------------------------------------
!  Purpose     : 采用 Householder 做 QR 分解
!
!  Post Script :
!       1.     A=QR
!       2.     Q 为正交矩阵
!       3.
!-----------------------------------------------------
!  Input  parameters  :
!       1.   A 要求分解之矩阵
!       2.   M,N 矩阵维数
!  Output parameters  :
!       1.
!       2.   R  上三角矩阵
!  Common parameters  :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
```

```
integer::M,N
real*8::A(M,N),Q(M,M),R(M,N)
real*8::H0(M,M),H1(M,M),H2(M,M),qt(M,M)
real*8::A1(M,N),A2(M,N),u(M)
integer::i,k,j
A1=A
H1=0d0
do j=1,M
 H1(j,j)=1d0
end do
do k=1,n-1
!设置 H 矩阵初值，这里设置为单位矩阵
  H0=0d0
  do i=1,M
  H0(i,i)=1d0
  end do
  s=0d0
  do i=k,M
  s=s+a1(i,k)*a1(i,k)
  end do
  s=dsqrt(s)
  u=0d0
  u(k)=a1(k,k)-s
  do i=k+1,m
  u(i)=a1(i,k)
  end do
  du=0
  do i=k,m
  !求的单位 u 长度平方
  du=du+u(i)*u(i)
  end do
  do i=k,m
    do j=k,m
        H0(i,j)=-2d0*u(i)*u(j)/du
        if (i==j) then
        H0(i,j)=1+H0(i,j)
        end if
    end do
  end do
H1=matmul(H1,H0)
A2=matmul(H1,A1)
A1=A2
end do
Q=H1
!QT 为 Q 的转置矩阵
do i=1,m
  do j=1,m
  QT(i,j)=Q(j,i)
  end do
end do
R=matmul(QT,A)
end subroutine solve
end module hou qr
program main
!--------------------------------------program comment
```

```
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2009.07.30
!-----------------------------------------------------
!  Purpose    : Householder镜像变换求QR分解主函数
!
!  Post Script :
!      1.
!      2.
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.  result.txt  输出结果
!      2.
!
!-----------------------------------------------------
use hou qr
implicit real*8(a-z)
integer,parameter::m=3,n=3
real*8::A(m,n),Q(m,m),R(m,n)
open(unit=11,file='result.txt')
A=reshape((/4,3,0,4,3,1,0,-1,1/),(/3,3/))
call solve(A,Q,R,m,n)
write(11,101)
101 format(T10,'QR分解之Householder方法',/,T3,'Q=',/)
write(11,102)((Q(i,j),j=1,M),i=1,M)
!变量输出格式只针对IVF编译器，在CVF中不支持
102 format(<M>(<M>F10.5/))
write(11,103)
103 format(T3,'R=',/)
write(11,104)((R(i,j),j=1,n),i=1,m)
!变量输出格式只针对IVF编译器，在CVF中不支持
104 format(<m>(<N>F10.5/))
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，如图 3-3 所示。



```
        QR分解之Householder方法

  Q=

   0.80000   0.00000   0.60000
   0.60000   0.00000  -0.80000
   0.00000   1.00000   0.00000

  R=

   5.00000   5.00000  -0.60000
   0.00000   1.00000   1.00000
   0.00000   0.00000   0.80000
```

图 3-3　Householder 方法 QR 分解结果

在编程时候，如果不确定分解结果是否正确，可以把自己的结果回代，看是否满足 $\mathbf{QR}=\mathbf{A}$ 。

3.3 修正的Gram-Schimdt正交化方法的QR分解

1. 实验基本原理

Gram-Schimdt 线性代数或矩阵论中大家都熟悉的正交化方法，令 $\{\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n\}$ 是 p 维向量子空间 W 的任意一组基，则子空间 W 的标准正交基 $\{\mathbf{u}_1,\mathbf{u}_2,\cdots,\mathbf{u}_n\}$ 可以通过 Gram-Schimdt 正交化构造，这个方法是大家都很熟悉的，即

$$\mathbf{p}_1=\mathbf{x}_1,\mathbf{u}_1=\frac{\mathbf{p}_1}{\|\mathbf{p}\|}=\frac{\mathbf{x}_1}{\|\mathbf{x}_1\|}$$

$$\mathbf{p}_k=\mathbf{x}_k-\sum_{i=1}^{k=1}\left(\mathbf{v}_i^{\ H}\mathbf{x}_k\right)\mathbf{u}_i,\mathbf{u}_k=\frac{\mathbf{p}_k}{\|\mathbf{p}_k\|}$$

对于超定的线性方程系数矩阵的 QR 分解可以通过 Gram-Schimdt 正交化方法来实现，然而采用 Gram-Schimdt 正交化方法求解列正交矩阵 $\mathbf{Q}$ 时，舍入误差较大，这在求解最小二乘法时候，有时会不稳定。针对 Gram-Schimdt 正交化的缺点，下面给出修正的 Gram-Schimdt 正交化算法。

对于 n 个向量 $\mathbf{a}_1,\mathbf{a}_2,\cdots,\mathbf{a}_n$ 构造标准正交基 $\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_n$ 方法如下

$$R_{11}=\|\mathbf{a}_1\|$$

$$\mathbf{q}_1=\frac{\mathbf{q}_1}{R_{11}}$$

对于 $k=2,\cdots,n$ 。

$$R_{jk}=\mathbf{q}_j^H\mathbf{a}_k,\ j=1,\cdots,k-1$$

$$R_{kk}=\left\|\mathbf{a}_k-\sum_{j=1}^{k-1}\mathbf{q}R_{jk}\right\|$$

$$\mathbf{q}=\frac{\mathbf{a}_k-\sum_{j=1}^{k-1}\mathbf{q}R_{jk}}{R_{kk}}$$

可以看出，此方法比上一节介绍的 Holuseholder 要更复杂一些，虽然公式看上去不是很繁，但程序设计时需要较多的编程技巧，而在这一过程中对向量与矩阵的维数分析至关重要，做到这一点的前提是真正理解了算法的实现过程。

2. 实验目的与要求

- 了解 Gram-Schmidt 正交化的几何与代数意义。
- 了解修正的 Gram-Schmidt 正交化的计算步骤。
- 熟悉修正 Gram-Schmidt 正交化的计算流程。
- 最好能自己编写修正 Gram-Schmidt 正交化方法。

3. 实验内容与数据来源

编写修正的 Gram-Schimdit 方法函数，计算矩阵

$$\mathbf{A}=\begin{pmatrix}1 & 2 & -3\\ 1 & 3 & 10\\ 1 & -1 & 6\end{pmatrix}$$

的 QR 分解。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
Q	REAL*8(M,N)	变换矩阵
R	REAL*8(N,N)	上三角矩阵
输入参变量	**数据类型**	**变量说明**
A	REAL*8(M,N)	欲求分解之矩阵
M	INTEGER	A 的行数
N	INTEGER	A 的列数

5. 程序代码

```
module gram_sch
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   修正的 Gram-Schimdt 正交化求 QR 分解
!
!-----------------------------------------------------
contains
subroutine solve(A,Q,R,M,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
```

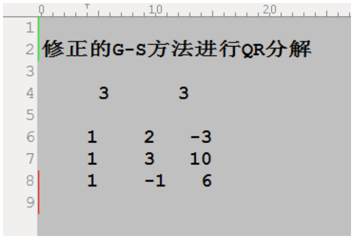
```
!  Purpose   :  采用修正的 Gram-Schmidt 分解求矩阵的 QR 分解
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    A 原始矩阵
!      2.    A(M,N)
!  Output parameters  :
!      1.    分解结果为  Q(M,N):注意 Q 不是方阵，Q 列向量为标准正交基
!      2.              R(N,N): R 是方阵
!      3.
!----------------------------------------------------
!  Post Script :
!      1.  注意矩阵的维数，分解后 Q 列向量是正交的
!      2.  关于编程方法可以参看《矩阵分析与应用》张贤达编著
!      3.  详细的数学解释，可以参看麻省理工学院的
!          线性代数教材《Linear Algebra with Application》
!----------------------------------------------------
implicit real*8(a-z)
integer::M,N
integer::i,j,k
real*8::A(M,N),Q(M,N),R(N,N)
real*8::vec_temp(M)
R(1,1)=dsqrt(dot_product(a(:,1),a(:,1)))
Q(:,1)=a(:,1)/R(1,1)

do k=2,N
     do j=1,k-1
       R(j,k)=dot_product(Q(:,j),A(:,k))
     end do
     vec_temp=A(:,k)
     do j=1,k-1
       vec_temp=vec_temp-Q(:,j)*R(j,k)
     end do
   R(k,k)=dsqrt(dot_product(vec_temp,vec_temp))
   Q(:,k)=vec_temp/R(k,k)
end do
end subroutine solve
subroutine dri_main(M,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  驱动程序
!
!-----------------------------------------------------
integer::M,N
real*8::A(m,n),Q(m,n),R(n,n)
real*8::b(M),x(N)
!读入矩阵
read(11,*)((A(i,j),j=1,n),i=1,m)
call solve(A,Q,R,m,n)
!-------------------这段程序用于输出 QR 分解
write(12,101)
101 format(T10,'修正的 Gram-Schmidt 方法 QR 分解',/,T3,'Q=',/)
```

```
write(12,102)((Q(i,j),j=1,N),i=1,M)
!变量输出格式只针对 IVF 编译器，在 CVF 中不支持
102 format(<M>(<N>F10.5/))
write(12,103)
103 format(T3,'R=',/)
write(12,104)((R(i,j),j=1,n),i=1,n)
!变量输出格式只针对 IVF 编译器，在 CVF 中不支持
104 format(<n>(<N>F10.5/))
!------------------------------
end subroutine dri_main
end module gram_sch
program main
!--------------------------------------program comment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date       :  2010-4-11
!-----------------------------------------------------
!  Purpose    :  1. 采用修正的 Gram-Schimidt 方法进行 QR 分解
!                2. 的主函数
!-----------------------------------------------------
!  In put data  files :
!      1.        fin.txt 输入文件
!      2.
!  Output data files  :
!      1.        fou.txt 输出文件
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.        由主函数引导驱动程序
!      2.
!-----------------------------------------------------
use gram_sch
integer::M,N
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
!读入 A 矩阵
!读入方程维数系数
read(11,*)M,N
!调用驱动函数
call dri_main(M,N)
end program main
```

6. 实验结论

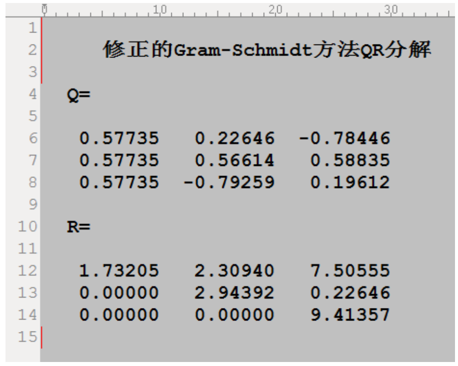
计算时需要先准备输入卡片，输入格式如图 3-4 所示。

计算结果保存在文件 result.txt 中，如图 3-5 所示。



```
修正的G-S方法进行QR分解

    3      3

  1    2   -3
  1    3   10
  1   -1    6
```

图 3-4　输入数据



```
   修正的Gram-Schmidt方法QR分解

Q=

 0.57735   0.22646  -0.78446
 0.57735   0.56614   0.58835
 0.57735  -0.79259   0.19612

R=

 1.73205   2.30940   7.50555
 0.00000   2.94392   0.22646
 0.00000   0.00000   9.41357
```

图 3-5　QR 分解结果

可以验证计算结果是符合要求的。

3.4　QR分解法计算最小二乘问题

1. 实验基本原理

前面两节分别介绍了两种 QR 分解方法，有了 QR 分解方法就可以方便的实现对超定或者适定方程的最小二乘法计算。

对于超定方程 $\mathbf{Ax}=\mathbf{b}$，矩阵 $\mathbf{A}$ 是 $m\times n$ 维，对 $\mathbf{A}$ 的列向量采用上述方法正交化得到 $\mathbf{A}=\mathbf{QR}$，进而可以得到 $\mathbf{Rx}=\mathbf{Q}^{\mathrm{T}}\mathbf{b}$，其中 $\mathbf{R}$ 为上三角矩阵，回带即可得到原超定方程最小二乘意义下的解。

QR 分解方法可以选择 Househoulder 及修正的 G-S 方法或者快速 Givens 变换等，这里选择修正的 G-S 方法。

在编程时，读者需要仔细考虑各矩阵的维数。

2. 实验目的与要求

- 了解最小二乘算法 QR 实现的基本原理
- 能用作者编写的程序计算最小二乘问题。
- 能对最小二乘结果做出解释。

3. 实验内容与数据来源

计算超定方程 $\mathbf{Ax}=\mathbf{b}$，在最小二乘意义下的解。其中

$$
\mathbf{A}=\begin{pmatrix} 1.4775 & -1.3152 & -0.5322 \\ -0.4908 & 1.2562 & -1.9365 \\ 0.4701 & 2.8023 & 0.0851 \\ -2.0368 & -4.1887 & 0.1077 \\ 2.4469 & 4.2939 & 3.1763 \\ -3.1104 & 2.7571 & 2.9483 \\ 1.8678 & -0.1321 & 1.4432 \\ -3.1649 & -0.6414 & -1.2139 \end{pmatrix},\quad \mathbf{b}=\begin{pmatrix} 2.1960 \\ -1.3255 \\ 7.7531 \\ -14.8732 \\ 21.8125 \\ -1.0265 \\ 7.6441 \\ -13.4903 \end{pmatrix}
$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8(N)	方程的解
输入参变量	**数据类型**	**变量说明**
A	REAL*8(M,N)	系数矩阵
b	INTEGER	右向量
M	INTEGER	A 的行数
N	INTEGER	A 的列数

5. 程序代码

```
module gram_sch
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   修正的 Gram-Schimdt 正交化求最小二
!                  乘问题模块
!
!-----------------------------------------------------
!  Parameters  :
!      1.    solve     解超定方程方法函数
!      2.    gram_dec  G-S ,QR 分解
!      3.    uptri     上三角方程回带函数
!      4.
!-----------------------------------------------------
!  Post Script :
!      1.       即可以单独调用 QR 分解函数
!      2.       也可以调用解方程函数
!-----------------------------------------------------
contains
subroutine  solve(A,b,x,M,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
```

```
! Date     :
!---------------------------------------------------
! Purpose  :  通过修正的 Gram-Schmidt 正交化求最小二问题
!             方法函数
!---------------------------------------------------
! Method   :
!             对超定方程 A 进行 QR 分解后  方程变为
!                QR x=b
!              => Rx=Q'b   R 为上三角阵
!              => 回带，可以求得最小二乘意义下的解
!---------------------------------------------------
! Post Script :
!     1.      即求解超定方程组 Ax=b    其中 A(M,N)  M>N
!     2.
!---------------------------------------------------
implicit real*8(a-z)
integer::M,N
real*8::A(M,N),Q(M,N),R(N,N)
real*8::b(M)
real*8::QT(N,M)  !Q 的转置矩阵
real*8::QTb(N)   !Q'b
real*8::x(N)
call gram dec(A,Q,R,M,N)
QT=transpose(Q)
QTb=matmul(QT,b)  !  Rx=Q'b
call uptri(R,QTb,x,N) !回带
end subroutine
subroutine gram_dec(A,Q,R,M,N)
!---------------------------------subroutine  comment
! Version  :  V1.0
! Coded by :  syz
! Date     :
!---------------------------------------------------
! Purpose  :   采用修正的 Gram-Schmidt 分解求矩阵的 QR 分解
!
!---------------------------------------------------
! Input  parameters  :
!     1.    A 原始矩阵
!     2.    A(M,N)
! Output parameters  :
!     1.    分解结果为  Q(M,N):注意 Q 不是方阵，Q 列向量为标准正交基
!     2.                R(N,N): R 是方阵
!     3.
!---------------------------------------------------
implicit real*8(a-z)
integer::M,N
integer::i,j,k
real*8::A(M,N),Q(M,N),R(N,N)
real*8::vec_temp(M)
R(1,1)=dsqrt(dot_product(a(:,1),a(:,1)))
Q(:,1)=a(:,1)/R(1,1)
do k=2,N
     do j=1,k-1
       R(j,k)=dot_product(Q(:,j),A(:,k))
     end do
```

```
        vec_temp=A(:,k)
        do j=1,k-1
          vec_temp=vec_temp-Q(:,j)*R(j,k)
        end do
      R(k,k)=dsqrt(dot_product(vec_temp,vec_temp))
      Q(:,k)=vec_temp/R(k,k)
end do
end subroutine gram_dec
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!
implicit real*8(a-z)
integer::i,j,k,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module gram_sch
module driver
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :
!               驱动程序模块
!               分别调用 QR 分解，以及用之解决最小二乘问题
!-----------------------------------------------------
contains
subroutine dri_main(M,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  驱动模块主函数
!
!-----------------------------------------------------
use gram_sch
integer::M,N
real*8::A(m,n),Q(m,n),R(n,n)
```

```
real*8::b(M),x(N)
!读入矩阵
read(11,*)((A(i,j),j=1,n),i=1,m)
!读入 b 向量
read(11,*)b
call gram_dec(A,Q,R,m,n)
!-------------------这段程序用于输出 QR 分解
write(12,101)
101 format(T10,'修正的 Gram-Schmidt 方法 QR 分解计算最小二乘问题',/,T3,'Q=',/)
write(12,102)((Q(i,j),j=1,N),i=1,M)
!变量输出格式只针对 IVF 编译器，在 CVF 中不支持
102 format(<M>(<N>F10.5/))
write(12,103)
103 format(T3,'R=',/)
write(12,104)((R(i,j),j=1,n),i=1,n)
!变量输出格式只针对 IVF 编译器，在 CVF 中不支持
104 format(<N>(<N>F10.5/))
!------------------------------
!调用求解函数
call solve(A,b,x,M,N)
write(12,105)x
105 format(T3,'最小二乘意义下的解为',//,(<N>F10.5))
end subroutine dri_main
end module driver
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-11
!-----------------------------------------------------
!  Purpose   :  1. 采用修正的 Gram-Schmidt 方法进行 QR 分解
!               2. 分解后进行超定方程的最小二乘求解
!-----------------------------------------------------
!  In put data  files :
!      1.        fin.txt 输入文件
!      2.
!  Output data files  :
!      1.        fou.txt 输出文件
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.         由主函数引导驱动程序
!      2.         该程序可以处以一般的最小二乘问题
!                 只要按照输入卡片输入数据，即可以计算
!-----------------------------------------------------
use driver
integer::M,N
open(unit=11,file='fin.txt')
open(unit=12,file='fout.txt')
read(11,*)
!读入 A 矩阵
!读入方程维数系数
read(11,*)M,N
!调用驱动函数
call dri_main(M,N)
```

```
end program main
```

6. 实验结论

程序运行需要准备输入数据 fin.txt，格式如图 3-6 所示。

```
采用修正的Gram-Schmidt方法进行QR分解，并计算最小二乘

    8       3

   1.4775   -1.3152   -0.5322
  -0.4908    1.2562   -1.9365
   0.4701    2.8023    0.0851
  -2.0368   -4.1887    0.1077
   2.4469    4.2939    3.1763
  -3.1104    2.7571    2.9483
   1.8678   -0.1321    1.4432
  -3.1649   -0.6414   -1.2139      : End of Matrix A

   2.1960
  -1.3255
   7.7531
 -14.8732
  21.8125
  -1.0265
   7.6441
 -13.4903   :End of Vector b
```

图 3-6　采用修正的 Gram-Schimdt 方法进行 QR 分解输入数据

计算结果保存在文件 fout.txt 中，如图 3-7 所示。

```
   修正的Gram-Schmidt方法QR分解计算最小二乘问题

 Q=

   0.24638  -0.24554  -0.02305
  -0.08184   0.19539  -0.54836
   0.07839   0.36935  -0.21878
  -0.33965  -0.49536   0.39078
   0.40804   0.49254   0.35471
  -0.51868   0.51528   0.47100
   0.31147  -0.09776   0.33043
  -0.52777   0.04543  -0.20335

 R=

   5.99672   1.83491   0.85446
   0.00000   7.19771   2.61781
   0.00000   0.00000   4.33668

 最小二乘意义下的解为

   3.39103   2.06349   1.31222

```

图 3-7　QR 分解结果及最小二乘解

程序不仅给出了最小二乘的计算结果，还给出了 QR 分解情况，实验中的软件生成以后，可以计算一般的超定方程的最小二乘问题，只要修改输入卡片即可。

3.5　最小二乘曲线拟合

1. 实验基本原理

对于给定的一组数据$\{(x_i,y_i)\}_{i=1}^m$，假设拟合函数

$$\varphi(x)=a_0\varphi_0(x)+a_1\varphi_1(x)+\cdots+a_n\varphi_n(x)$$

其中基函数为线性无关的函数系。最小二乘拟合问题就是求系数，使下面的表达式取极小值。

$$E=\sum_{i=1}^{m}\left[\sum_{j=0}^{n}a_j\varphi_j(x)-y_i\right]^2$$

根据多元函数极值问题，E 取极值的必要条件是

$$\frac{\partial E(a_0,a_1,\cdots,a_n)}{\partial a_i}=0$$

设

$$\mathbf{a}=\begin{bmatrix}a_0\\a_1\\\vdots\\a_n\end{bmatrix},\qquad \mathbf{y}=\begin{bmatrix}y_1\\y_2\\\vdots\\y_m\end{bmatrix}$$

以及 $A=\left[a_{ij}\right]_{m\times(n+1)},a_{ij}=\varphi_j(x_i)$，

那么

$$E=\left\|A\mathbf{a}-\mathbf{y}\right\|_2^2$$

如此最小二乘拟合问题就转化为超定线性方程组

$$A\mathbf{a}=\mathbf{y}$$

的最小二乘解问题。

2. 实验目的与要求

- 了解曲线拟合的基本含义
- 能够自行推导出曲线拟合最小二乘方法的计算步骤
- 会使用作者编写的多项式拟合程序
- 最好能自行编写程序解决相关问题

3. 实验内容与数据来源

求数据

x	-3	-2	-1	0	1	2	3
y	4	2	3	0	-1	-2	-5

的二阶与三阶多项式最小二乘拟合。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
C	REAL*8(M)	系数向量
输入参变量	**数据类型**	**变量说明**
X	REAL*8(N)	输入的自变量
Y	REAL*8(N)	输入的变量
N	INTEGER	输入数据的维数
M	INTEGER	欲采用的多项式阶数

5. 程序代码

```
module lsqcurvefit
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :  2010.07.09
!-----------------------------------------------------
!  Description :   任意阶多项式曲线拟合模块
!
!  Post Script :
!      1.
!      2.
!
!-----------------------------------------------------
!  Contains     :
!      1.    拟合函数，基函数
!      2.    最小二乘法函数
!-----------------------------------------------------
!  Parameters   :
!      1.
!      2.
!-----------------------------------------------------
contains
subroutine solve(x,y,N,c,m)
!---------------------------------subroutine  comment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date      :
!-----------------------------------------------------
```

```
!  Purpose     :  任意阶多项式拟合函数
!
!  Post Script :
!       1.
!       2.
!       3.
!-----------------------------------------------------
!  Input  parameters  :
!       1.  x,y --- 输入数据
!       2.  N --- x,y 向量的维数
!       3.  m----希望用 m 阶多项式拟合
!  Output parameters  :
!       1. c 系数向量
!       2.
!  Common parameters  :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::N,m
real*8::x(n),y(n),c(M)
real*8::bv(M)
integer::i
!A 系数矩阵
real*8::A(N,M)
!如果多项式阶数加高于实测数据个数则报错
!-------------
if (m>n) then
write(11,101)
stop
end if
101 format(/,'warning:the order of polynomial+1 > the ',/,&
            'number of date,that is forbidden')
!-----------------------------
do i=1,n
   call basefunc(bv,x(i),m)
   A(i,:)=bv
end do
call leas_eq(A,y,c,N,m)
end subroutine solve
subroutine basefunc(bv,x,m)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :2010.07.09
!-----------------------------------------------------
!  Purpose     :  任意阶多项式基函数
!
!  Post Script :
!       1.
!       2.
!       3.
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 标量
```

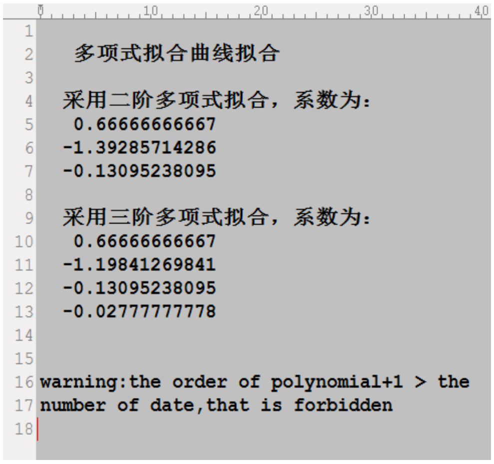
```
!      2.    m 多项式阶数
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::M
real*8::bv(m)
integer::i
bv(1)=1d0
do  i=2,m
   bv(i)=bv(i-1)*x
end do
end subroutine basefunc
subroutine  leas_eq(A,b,x,M,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  通过修正的 Gram-Schmidt 正交化求最小二问题
!              方法函数
!-----------------------------------------------------
!  Post Script :
!      1.      即求解超定方程组 Ax=b    其中 A(M,N)  M>N
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::M,N
real*8::A(M,N),Q(M,N),R(N,N)
real*8::b(M)
real*8::QT(N,M)  !Q 的转置矩阵
real*8::QTb(N)   !Q'b
real*8::x(N)
call gram_dec(A,Q,R,M,N)
QT=transpose(Q)
QTb=matmul(QT,b)  !  Rx=Q'b
call uptri(R,QTb,x,N) !回带
end subroutine leas_eq
subroutine gram_dec(A,Q,R,M,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  采用修正的 Gram-Schmidt 分解求矩阵的 QR 分解
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    A 原始矩阵
!      2.    A(M,N)
!  Output parameters  :
```

```
!      1.    分解结果为 Q(M,N):注意 Q 不是方阵，Q 列向量为标准正交基
!      2.              R(N,N): R 是方阵
!      3.
!----------------------------------------------------
implicit real*8(a-z)
integer::M,N
integer::i,j,k
real*8::A(M,N),Q(M,N),R(N,N)
real*8::vec_temp(M)
R(1,1)=dsqrt(dot_product(a(:,1),a(:,1)))
Q(:,1)=a(:,1)/R(1,1)
do k=2,N
     do j=1,k-1
       R(j,k)=dot_product(Q(:,j),A(:,k))
     end do
     vec_temp=A(:,k)
     do j=1,k-1
       vec_temp=vec_temp-Q(:,j)*R(j,k)
     end do
   R(k,k)=dsqrt(dot_product(vec_temp,vec_temp))
   Q(:,k)=vec_temp/R(k,k)
end do
end subroutine gram_dec
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
implicit real*8(a-z)
integer::i,j,k,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
   x(i)=b(i)
  do j=i+1,N
   x(i)=x(i)-a(i,j)*x(j)
  end do
   x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module lsqcurvefit
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.07.09
!-----------------------------------------------------
!  Purpose     : 多项式拟合主函数
!
!  Post Script :
```

```
!       1.
!       2.
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.
!       2.
!
!-----------------------------------------------------
use  lsqcurvefit
implicit real*8(a-z)
real*8::x(7),y(7)
real*8::c1(3),c2(4),c3(9)
open(unit=11,file='result.txt')
x=(/-3d0,-2d0,-1d0,0d0,1d0,2d0,3d0/)
y=(/4d0,2d0,3d0,0d0,-1d0,-2d0,-5d0/)
call solve(x,y,7,c1,3)
call solve(x,y,7,c2,4)
write(11,101)c1,c2
101 format(/T4,'多项式拟合曲线拟合',//,&
            T3,'采用二阶多项式拟合，系数为：',3(/,F16.11),//,&
            T3,'采用三阶多项式拟合，系数为：',4(/,F16.11),/)
call solve(x,y,7,c3,9)
end program main
```

6. 实验结论

计算结果保持着文件 result.txt 中，如图 3-8 所示。



```
  多项式拟合曲线拟合

 采用二阶多项式拟合，系数为：
  0.66666666667
 -1.39285714286
 -0.13095238095

 采用三阶多项式拟合，系数为：
  0.66666666667
 -1.19841269841
 -0.13095238095
 -0.02777777778


warning:the order of polynomial+1 > the
number of date,that is forbidden
```

图 3-8　多项式曲线拟合

即如果采用二次多项式拟合函数为

$$f\left(x\right)=0.66666666667\text{-}1.39285714286x--0.13095238095x^2$$

三阶多项式拟合函数为

$$f(x)=0.66666666667\text{-}1.19841269841x\text{-}0.13095238095x^2\\ \text{-}0.02777777778x^3$$

从计算结果可以看出采用二阶多项式已经很好的对数据做了拟合，而高次项则已经很小。在最小二乘曲线拟合时，多项式的阶数加 1 不能高于实测数据对的个数。在主函数中，我们故意用 10 次多项式（加常数-零次项）11 个系数，系统则提示次数过高，这是因为我们在程序设计时候增加了这样的语句，防止调用时不小心输入错误。

需要说明的是数据拟合并不一定需要基是多项式，也可以是其他函数。实际上较高阶多项式拟合并不常用，因为多项式拟合的方程系数矩阵是 Hilbert 矩阵，是一个严重病态矩阵，所以有时候经常采用 Chebyshev 多项式拟合方法，这里就不多做介绍。

本章小结

本章介绍了最小二乘的法方程解法和基于 QR 分解的方法。一般而言在实际计算中，多采用 QR 分解的方法而不是解算法方程，这已经是数值代数里一个基本常识了。也就是说，如果无特殊情况，QR 分解应该是作为最常用的方法。当系数矩阵接近奇异时，可以采用奇异值分解法，不过奇异值分解需要更大的运算量。

对于 QR 分解，本章介绍了 Houlseholder 镜像变换和修正的 Gram-Schimidt 正交化方法，除此之外比较常用的方法还有 Givens 正交变换及其变形算法。相比之下 Givens 变换因计算效率较低，所以有些现代数值分析已经不在介绍。即便是改进的 Givens 变换不需要开平方，部分计算数学家认为，这依然不具备和 Householder 竞争的能力。然而在实际应用时候，使用改进的 Givens 变换（简称为 G-G 变换）可以对数据逐行处理，而不需要存储数据，这是比较有利的一面，不过作者认为实际上 Houlseholder 加以改进也可以做到这一点。

关于非线性最小二乘问题，需要用到多元微分知识，这在非线性方程组一章有比较充分的体现。我们把非线性最小二乘问题留在应用范例一章中介绍。

关于数据拟合部分，这里介绍了一个常用的多项式拟合方法。多项式拟合因为方便简单且适用，所以常常被用来吸收实测数据中的系统差。如果是知道函数性态的，一般直接以该函数类型作为基函数。另外，近几十年发展起来的人工神经网络对函数具有良好的逼近效果。关于这方面的材料读者可以阅读德克萨斯大学数学系 Ward Cheney、Will Light 教授编写的《A Course in Approximation Theory》一书。

第 4 章

矩阵特征值及特征向量

特征值问题是数值代数基本问题之一，无论在理论上还是在工程技术上都非常重要。特征值与特征向量理论发展也是相当丰富的。在这一章节中，主要介绍标准的特征值问题 $\mathbf{Ax}=\lambda\mathbf{x}$ 的数值方法。

关于数值特征值与特征向量问题，需要一些准备知识，读者可以翻阅线性代数与矩阵分析理论。

4.1 幂法计算主特征值及其特征向量

1. 实验基本原理

在许多工程与物理应用问题中，往往并不需要计算出矩阵的全部特征值与特征向量，而只需要计算出矩阵的按模最大的特征值及其特征向量，通常按模最大的特征值称为主特征值。对于这一类的问题采用乘幂法通常比较合适，乘幂法计算简单而且对于稀疏矩阵比较有效，但有时候收敛速度不是太快。

设 $A\in R^{n\times n}$ 有 n 个线性无关的特征向量，主特征值满足 $|\lambda_1|>|\lambda_2|\ge|\lambda_3|\ge\cdots\ge|\lambda_n|$，其对应的特征向量为 $\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_n$，那么对于任意的非零初始向量 $\mathbf{v}_0=\mathbf{u}_0$，按照下面的方法构造向量序列

$$\begin{aligned}\mathbf{v}_0 &= \mathbf{u}_0\ne\mathbf{0}\\ \mathbf{v}_k &= \mathbf{A}\mathbf{u}_{k-1}\\ \mathbf{u}_k &= \frac{\mathbf{v}_k}{\max(\mathbf{v}_k)}\end{aligned}$$

则有

$$\lim_{k\to\infty}\mathbf{u}_k=\frac{\mathbf{x}_1}{\max(\mathbf{x}_1)}$$

$$\lim_{k\to\infty}\max(\mathbf{v}_k)=\lambda_1$$

$\max(\mathbf{v})$ 表示向量 $\mathbf{v}$ 中绝对值最大的分量。这个收敛定理，就已经包含了乘幂法计算主特征值的计算方法。

2. 实验目的与要求

- 理解乘幂法的基本思想。
- 熟悉计算流程，了解幂法的一些变形形式。
- 在程序设计中灵活处理自己需求的变量。
- 程序设计时应该考虑的计算精度需求，在满足一定精度下即停止迭代，给出允许迭代的最大次数，超过最大迭代次数也停止迭代。

3. 实验内容与数据来源

假设

$$A=\begin{bmatrix}-1 & 2 & 1\\ 2 & -4 & 1\\ 1 & 1 & -6\end{bmatrix}$$

计算 A 的主特征值与主特征向量。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
NAMDA	REAL*8	主特征值
U	REAL*8(N)	主特征向量
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	要计算的方阵
N	INTEGER	矩阵维数
TOL	REAL*8	误差容限

5. 程序代码

```
module power
!--------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-5-31
```

```
!----------------------------------------------------
!  Description :   幂法模块
!
!----------------------------------------------------
!  Parameters  :
!      1.
!      2.
!----------------------------------------------------
!  Contains    :
!      1.   方法函数
!      2.   取模最大分量函数
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
contains
subroutine solve(A,N,namda,u,tol)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-5-31
!----------------------------------------------------
!  Purpose   : 幂法计算主特征值及主特征向量
!
!----------------------------------------------------
!  Input  parameters  :
!       1.  N 矩阵维数
!       2.  A 输入矩阵 N*N 维
!       3.  tol 控制精度
!  Output parameters  :
!       1.  namda 主特征值
!       2.  u     主特征向量
!  Common parameters  :
!
!---------------------------------------------------
!  Post Script :
!       1.
!       2.
!---------------------------------------------------
implicit real*8(a-z)
integer::n,i,k
real*8::A(n,n)
real*8::u(n),u0(n),v(n)
!设置迭代初值向量
do i=1,n
  u0(n)=1d0
end do
u=u0
!设置模最大分量初值为，使之进入循环
m0=0
do k=1,500
v=matmul(A,u)
call max_rou(v,n,m1)
u=v/m1
```

```
!判断迭代停止标准
if (dabs(m1-m0)<tol) exit
!更新 m 值
m0=m1
end do
namda=m1
end subroutine solve
subroutine max rou(r,n,ma)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  给出向量模最大的分量
!            *是给出分量本身，而非分量的绝对值
!-----------------------------------------------------
!  Input  parameters  :
!      1.    r 输入向量
!      2.    n 输入向量的维数
!  Output parameters  :
!      1.    ma 输出结果
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.    本函数取模最大分量本身，而非取其绝对值
!      2.    例如 r=(1,0,-4,3),则结果取 ma= -4
!----------------------------------------------------
implicit real*8(a-z)
integer::n,i,k
!n 为向量维数
!i 为循环指标
!k 为第 k 个分量为模最大的分量，这里作为标记
real*8::r(n)
ma=dabs(r(1))
do i=2,n
  if (dabs(r(i))>ma) then
   ma=dabs(r(i))
   k=i   !用 k 记录指标，但是 ma 不取 r(i)
  end if
end do
ma=r(k)
end subroutine max_rou
end module power
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-5-31
!-----------------------------------------------------
!  Purpose   :  主函数
!
!-----------------------------------------------------
!  In put data  files :
```

```
!       1.
!       2.
!  Output data files  :
!       1.  result.txt 计算结果文件
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
use power
implicit real*8(a-z)
real*8::A(3,3),u(3)
open(unit=11,file='result.txt')
a=reshape((/-1d0,2d0,1d0,&
         2d0,-4d0,1d0,&
         1d0,1d0,-6d0/),(/3,3/))
call solve(A,3,namda,u,1d-7)
write(11,101)namda,u
101 format(T5,'幂法计算主特征值及特征向量',//,&
         T3,'主特征值为：',/,F12.7,//,&
         T3,'主特征向量为：',3(/F12.7))
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，如图 4-1 所示。

关于迭代条件的误差判断问题，可以选择相对误差方法，也可以选择绝对误差，在程序设计时我们采用了选择绝对误差方法，这样便直接知道计算结果与要求的误差范围有多大。当然这一条件可以根据需要而定。判断标准还可以以特征值对应的特征向量作为依据，即以两次迭代前后特征向量之间的误差向量范数作为与误差容限比较，如果误差向量范数小于给定的误差容限，则停止迭代。

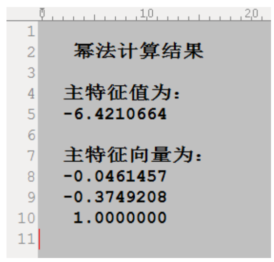


图 4-1　幂法计算结果

4.2　幂法2范数单位化方法

1. 实验基本原理

在用幂法计算时候迭代过程中

$$\begin{cases} \mathbf{u}_k = \mathbf{A}\mathbf{v}_{k-1} \\ \mathbf{v}_k = \dfrac{\mathbf{u}_k}{m_k} \end{cases}$$

m_k 是防止 $|\lambda_1|>1$ 时，$\mathbf{v}_k$ 分量会趋于无穷，而 $|\lambda_1|<1$ 时，$\mathbf{v}_k$ 分量会趋于零，这个过程叫规

范化或者单位化，实际上这个过程也可以取 2 范数。

下面给出伪代码算法：

for　k=1,2,⋯

$$z^{(k)} = Aq^{(k-1)}$$

$$q^{(k)} = \frac{z^{(k)}}{\left\| z^{(k)} \right\|_2}$$

$$\lambda^{(k)} = \left[q^{(k)} \right]^H Aq^{(k)}$$

end

当然在实际的编程时候，我们不会简单机械的运算一定的次数，而是给出满足精度的判断标准。当运算满足一定精度以后迭代停止，同时如果迭代足够多的次数以后还没收敛，则要给出警告信息。

2. 实验目的与要求

- 理解本实验基本原理部分伪代码的含义。
- 能够准确的把伪代码编程实现。
- 能够给出合理判断迭代停止条件。
- 如果迭代超过一定次数还没收敛，要停止迭代，防止进入死循环。
- 对计算结果做出恰当的分析。

3. 实验内容与数据来源

计算矩阵

$$A = \begin{bmatrix} 2 & 3 & 2 \\ 10 & 3 & 4 \\ 3 & 6 & 1 \end{bmatrix}$$

的主特征值及其对应的特征向量。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
NAMDA	REAL*8	主特征值
U	REAL*8(N)	主特征向量
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	要计算的方阵
N	INTEGER	矩阵维数

TOL	REAL*8	误差容限

5. 程序代码

```
module power
!------------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :  2010-5-31
!-------------------------------------------------------
!  Description :   幂法范数单位化方法模块
!
!-------------------------------------------------------
!  Parameters  :
!      1.
!      2.
!-------------------------------------------------------
!  Contains    :
!      1.   方法函数
!      2.   取模最大分量函数
!-------------------------------------------------------
!  Post Script :
!      1.
!      2.
!-------------------------------------------------------
contains
subroutine solve(A,N,namda,u,tol)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date     :  2010-5-31
!-------------------------------------------------------
!  Purpose   : 幂法范数单位化方法函数
!
!-------------------------------------------------------
!  Input  parameters  :
!       1.  N 矩阵维数
!       2.  A 输入矩阵 N*N 维
!       3.  tol 控制精度
!  Output parameters  :
!       1.  namda 主特征值
!       2.  u     主特征向量
!  Common parameters  :
!
!------------------------------------------------------
!  Post Script :
!       1.
!       2.
!------------------------------------------------------
implicit real*8(a-z)
integer::n,i,k
real*8::A(n,n)
real*8::u(n),u0(n),v(n),vt(n)
```

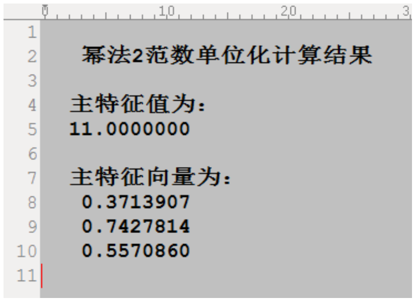
```
!设置迭代初值向量
do i=1,n
  u0(n)=1d0
end do
u=u0
!设置模最大分量初值为，使之进入循环
m0=0
do k=1,500
v=matmul(A,u)
call norm(v,rou,n)
u=v/rou
vt=matmul(A,u)
call vvdot(u,vt,m1,n)
!判断迭代停止标准
if (dabs(m1-m0)<tol) exit
!更新 m 值
m0=m1
end do
namda=m1
end subroutine solve
subroutine norm(r,rou,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-6-1
!-----------------------------------------------------
!  Purpose   :  计算向量范数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.  r 输入向量
!      2.  n 向量维数
!  Output parameters  :
!      1.  rou 向量长度（范数）
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,i
real*8::r(n)
rou=0d0
do i=1,n
rou=rou+r(i)**2
end do
rou=dsqrt(rou)
end subroutine norm
subroutine vvdot(r1,r2,dot,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
```

```
! Date      : 2010-6-1
!----------------------------------------------------
! Purpose   :  向量内积
!              r1(1)*r2(1)+r1(2)*r2(2)+...
!----------------------------------------------------
! Input parameters  :
!      1.  r1,r2 欲求之向量
!      2.  n 维数
! Output parameters  :
!      1.  dot 向量内积
!      2.
! Common parameters  :
!
!---------------------------------------------------
! Post Script :
!      1.
!      2.
!---------------------------------------------------
implicit real*8(a-z)
integer::n,i
real*8::r1(n),r2(n)
dot=0d0
do i=1,n
dot=dot+r1(i)*r2(i)
end do
end subroutine vvdot
end module power
program main
!-------------------------------------program comment
! Version   :  V1.0
! Coded by  :  syz
! Date      : 2010-5-31
!----------------------------------------------------
! Purpose   :  主函数
!
!----------------------------------------------------
! In put data  files :
!      1.
!      2.
! Output data files  :
!      1.  result.txt 计算结果文件
!      2.
!----------------------------------------------------
! Post Script :
!      1.
!      2.
!----------------------------------------------------
use power
implicit real*8(a-z)
real*8::A(3,3),u(3)
open(unit=11,file='result.txt')
a=reshape((/2d0,10d0,3d0,&
         3d0,3d0,6d0,&
         2d0,4d0,1d0/),(/3,3/))
call solve(A,3,namda,u,1d-7)
```

```
    write(11,101)namda,u
    101 format(/,T4,'幂法范数单位化计算结果',//,&
             T3,'主特征值为：',/,F12.7,//,&
             T3,'主特征向量为：',3(/F12.7))
    end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，如图 4-2 所示。



幂法2范数单位化计算结果

主特征值为：
11.0000000

主特征向量为：
0.3713907
0.7427814
0.5570860

图 4-2　幂法 2 范数单位化方法

可以验证以上计算结果即原矩阵的主特征值及主特征向量。

4.3　Rayleigh加速方法

1. 实验基本原理

上一个实验指出乘幂法在最大特征值与次最大特征值的绝对值比较接近时往往收敛速度会不是很快，针对这样的情况计算数学家发展了一些加速方法，本实验介绍其中比较典型的一种加速方法，Rayleigh 加速方法。

Hermitian 矩阵 $A\in C^{n\times n}$ 的 Rayleigh 商（也叫 Rayleigh-Ritz 比）是一个标量，定义为

$$R(\mathbf{x})=R(\mathbf{x},\mathbf{A})=\frac{(\mathbf{Ax},\mathbf{x})}{(\mathbf{x},\mathbf{x})}=\frac{\mathbf{x}^H\mathbf{Ax}}{\mathbf{x}^H\mathbf{x}}$$

通常 $\mathbf{x}$ 是待选择的向量，目的是使用 Rayleigh 商达到极值。

由此，我们可以把 Rayleigh 商应用到主特征值计算的乘幂法当中，提高乘幂法的收敛速度。

这里直接给出 Rayleigh 加速方法的算法：

任意给定 $\mathbf{x}^{(0)}\in R^n$，给定误差容限 TOL（小量）。

对于 $k=1,2,\cdots$ 做（3）到（6）

$$\mathbf{y}^{(k)}=\mathbf{Ax}^{(k-1)}$$

$$m_k = \frac{\left(\mathbf{A}\mathbf{x}^{(k-1)}, \mathbf{x}^{(k-1)}\right)}{\left(\mathbf{x}^{(k-1)}, \mathbf{x}^{(k-1)}\right)}$$

$$\mathbf{x}^{(k)} = \frac{\mathbf{y}^{(k)}}{m_k}$$

如果$|m_k - m_{k-1}| < TOL$，跳出循环。

结束。

上面的算法已经详细的描述了计算流程，只要把这个算法翻译成程序设计语言就可以。在第 6 步循环判断时，可以选择相对误差

$$\frac{|m_k - m_{k-1}|}{|m_k|} < TOL$$

当然也可以用迭代前后两次特征向量的误差向量范数作为迭代停止条件。

2. 实验目的与要求

- 知道 Rayleigh 商的相关概念。
- 明白 Rayleigh 加速原理。
- 能够正确的编程实现加速算法，并处理相关特征值计算问题。
- 处理好迭代次数与迭代停止条件，注意不能进入死循环。
- 对结果作出合理分析。

3. 实验内容与数据来源

已知矩阵

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$$

计算 A 的主特征值及其对应的特征向量。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
NAMDA	REAL*8	主特征值
U	REAL*8(N)	主特征向量
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	要计算的方阵

（续表）

输入参变量	数据类型	变量说明
N	INTEGER	矩阵维数
TOL	REAL*8	误差容限

5. 程序代码

```
module Rayleigh
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-5-31
!-----------------------------------------------------
!  Description :   Rayleigh 加速计算对称矩阵特征值模块
!
!-----------------------------------------------------
!  Parameters  :
!      1.
!      2.
!-----------------------------------------------------
!  Contains    :
!      1.   方法函数
!      2.   内积函数
!      3.   范数函数
!-----------------------------------------------------
contains
subroutine solve(A,N,namda,u,tol)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-5-31
!-----------------------------------------------------
!  Purpose   : Rayleigh 加速计算对称矩阵特征值函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.  N 矩阵维数
!       2.  A 输入矩阵 N*N 维
!       3.  tol 控制精度
!  Output parameters  :
!       1.  namda 主特征值
!       2.  u     主特征向量
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,i,k
real*8::A(n,n)
real*8::u(n),u0(n),v(n),vt(n)
```

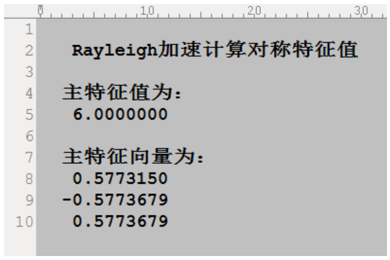
```
!设置迭代初值向量
do i=1,n
  u0(n)=1d0
end do
u=u0
m0=0
do k=1,500
v=matmul(A,u)
call vvdot(v,u,temp1,n)
call vvdot(u,u,temp2,n)
!Rayleigh 商
m1=temp1/temp2
u=v/m1
!判断迭代停止标准
if (dabs(m1-m0)<tol) exit
!更新 m 值
m0=m1
end do
namda=m1
!特征向量归一化
call norm(u,rou,n)
u=u/rou
end subroutine solve
subroutine norm(r,rou,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-6-1
!-----------------------------------------------------
!  Purpose   :  计算向量范数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.  r 输入向量
!       2.  n 向量维数
!  Output parameters  :
!       1.  rou 向量长度（范数）
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,i
real*8::r(n)
rou=0d0
do i=1,n
rou=rou+r(i)**2
end do
rou=dsqrt(rou)
end subroutine norm
subroutine vvdot(r1,r2,dot,n)
```

```
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-6-1
!-----------------------------------------------------
!  Purpose   :  向量内积
!              r1(1)*r2(1)+r1(2)*r2(2)+...
!-----------------------------------------------------
!  Input  parameters  :
!      1.  r1,r2 欲求之向量
!      2.  n 维数
!  Output parameters  :
!      1.  dot 向量内积
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,i
real*8::r1(n),r2(n)
dot=0d0
do i=1,n
dot=dot+r1(i)*r2(i)
end do
end subroutine vvdot
end module Rayleigh
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-5-31
!-----------------------------------------------------
!  Purpose   :  主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.  result.txt 计算结果文件
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use Rayleigh
implicit real*8(a-z)
real*8::A(3,3),u(3)
open(unit=11,file='result.txt')
a=reshape((/4d0,-1d0,1d0,&
```

```
            -1d0,3d0,-2d0,&
            1d0,-2d0,3d0/),(/3,3/))
    call solve(A,3,namda,u,1d-7)
    write(11,101)namda,u
    101 format(/,T4,'Rayleigh 加速计算对称特征值',//,&
            T3,'主特征值为: ',/,F12.7,//,&
            T3,'主特征向量为: ',3(/F12.7))
    end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，打开该文件如图 4-3 所示。



```
 Rayleigh加速计算对称特征值

主特征值为:
 6.0000000

主特征向量为:
 0.5773150
-0.5773679
 0.5773679
```

图 4-3　Rayleigh 加速计算对称矩阵特征值

若要判断计算结果是否正确，可以把特征值及特征向量带入原矩阵看矩阵方程两边是否相等。

4.4　修正的Rayleigh加速方法

1. 实验基本原理

修正的 Rayleigh 加速方法与 Rayleigh 加速方法差别不是很大，主要差别在于迭代初值的选择，当然由此引起 Rayleigh 商计算上的一些简化，当然客观的说这减少了计算机的运算量，在迭代过程中减少了向量点乘与与代数除法运算，从这个角度而言修正的算法也是有意义的。

在 Rayleigh 加速方法中，当 $(\mathbf{x},\mathbf{x})=\|\mathbf{x}\|_2^{\ 2}=1$ 时，$R(\mathbf{x})=(\mathbf{Ax},\mathbf{x})$。如果在迭代过程中中，$\mathbf{x}^{(k)}$ 用欧氏范数规范化，就是修正的 Rayleigh 加速方法，下面给出具体的算法步骤。

给定误差容限 TOL>0 和 $\mathbf{x}^{(0)}\in R$ 使 $\left\|\mathbf{x}^{(0)}\right\|_2=1$，如 $\mathbf{x}^{(0)}=\dfrac{1}{\sqrt{n}}(1,\cdots,1)^T$。

对于 $k=1,2,\cdots$ 做（3）到（6）

$$\mathbf{y}^{(k)}=\mathbf{Ax}^{(k-1)}$$

$$m_k=\left(\mathbf{Ax}^{(k-1)},\mathbf{x}^{(k-1)}\right)=\left(\mathbf{y}^{(k)}\right)^T\mathbf{x}^{(k-1)}$$

$$\mathbf{x}^{(k)} = \mathbf{y}^{(k)} / \left\| \mathbf{y}^{(k)} \right\|_2$$

如果$|m_k - m_{k-1}| < TOL$，跳出迭代循环。

结束。

可以看到这几个算法差别不是特别大，当然从算法构造的出发点的思想是有意义的，

本实验中可以把可以把迭代初值直接用程序方式生成。在迭代过程中对 Rayleigh 商做一些简化。

2. 实验目的与要求

- 明白修正的 Rayleigh 加速方法的原理。
- 能够快速的把基本原理部分的算法用计算机编程语言实现。
- 能用编写好的程序处理相关特征值计算问题。
- 对计算结果作出合理分析。

3. 实验内容与数据来源

用修正的 Rayleigh 加速方法计算矩阵

$$\mathbf{A} = \begin{bmatrix} 1.867310185256756 & 0.676506748114136 & 1.832446222432665 \\ 0.676506748114136 & 0.438313983411573 & 0.927636560784582 \\ 1.832446222432665 & 0.927636560784582 & 2.421070728983985 \end{bmatrix}$$

的主特征值及对应的特征向量。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
NAMDA	REAL*8	主特征值
U	REAL*8(N)	主特征向量
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	要计算的方阵
N	INTEGER	矩阵维数
TOL	REAL*8	误差容限

5. 程序代码

```
module mody_Rayleigh
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :  2010-6-1
```

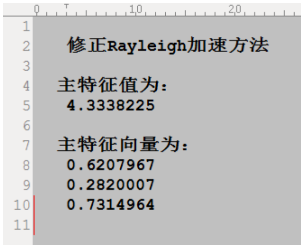
```
!--------------------------------------------------------
!  Description :   修正 Rayleigh 加速方法
!
!--------------------------------------------------------
!  Parameters  :
!      1.
!      2.
!--------------------------------------------------------
!  Contains    :
!      1.   方法函数
!      2.   内积函数
!      3.   范数函数
!--------------------------------------------------------
contains
subroutine solve(A,N,namda,u,tol)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-6-1
!--------------------------------------------------------
!  Purpose   : Rayleigh 加速计算对称矩阵特征值函数
!
!--------------------------------------------------------
!  Input  parameters  :
!       1.  N 矩阵维数
!       2.  A 输入矩阵 N*N 维
!       3.  tol 控制精度
!  Output parameters  :
!       1.  namda 主特征值
!       2.  u     主特征向量
!  Common parameters  :
!
!-------------------------------------------------------
!  Post Script :
!       1.
!       2.
!-------------------------------------------------------
implicit real*8(a-z)
integer::n,i,k
real*8::A(n,n)
real*8::u(n),u0(n),v(n)
!设置迭代初值向量
do i=1,n
  u0(n)=1d0
end do
call norm(u0,rou,n)
!u 已经单位化
u=u0
m0=0
do k=1,500
v=matmul(A,u)
call vvdot(v,u,m1,n)
call norm(v,rou,n)
u=v/rou
!判断迭代停止标准
```

```
    if (dabs(m1-m0)<tol) exit
    !更新 m 值
    m0=m1
    end do
    namda=m1
    !特征向量归一化
    call norm(u,rou,n)
    u=u/rou
    end subroutine solve
    subroutine norm(r,rou,n)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :  2010-6-1
    !-----------------------------------------------------
    !  Purpose   :  计算向量范数
    !
    !-----------------------------------------------------
    !  Input  parameters  :
    !      1.  r 输入向量
    !      2.  n 向量维数
    !  Output parameters  :
    !      1.  rou 向量长度（范数）
    !      2.
    !  Common parameters  :
    !
    !----------------------------------------------------
    !  Post Script :
    !      1.
    !      2.
    !----------------------------------------------------
    implicit real*8(a-z)
    integer::n,i
    real*8::r(n)
    rou=0d0
    do i=1,n
    rou=rou+r(i)**2
    end do
    rou=dsqrt(rou)
    end subroutine norm
    subroutine vvdot(r1,r2,dot,n)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :  2010-6-1
    !-----------------------------------------------------
    !  Purpose   :  向量内积
    !              r1(1)*r2(1)+r1(2)*r2(2)+...
    !-----------------------------------------------------
    !  Input  parameters  :
    !      1.  r1,r2 欲求之向量
    !      2.  n 维数
    !  Output parameters  :
    !      1.  dot 向量内积
    !      2.
```

```
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,i
real*8::r1(n),r2(n)
dot=0d0
do i=1,n
dot=dot+r1(i)*r2(i)
end do
end subroutine vvdot
end module mody_Rayleigh
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-5-31
!-----------------------------------------------------
!  Purpose   :  主函数
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.  result.txt 计算结果文件
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
use mody_Rayleigh
implicit real*8(a-z)
real*8::A(3,3),u(3)
open(unit=11,file='result.txt')
a=reshape((/1.867310185256756d0,0.676506748114136d0,&
1.832446222432665d0,&
   0.676506748114136d0, 0.438313983411573d0, &,
0.927636560784582d0,&
   1.832446222432665d0,0.927636560784582d0,& 2.421070728983985d0/),(/3,3/))
call solve(A,3,namda,u,1d-7)
write(11,101)namda,u
101 format(/,T4,'修正 Rayleigh 加速方法',//,&
         T3,'主特征值为：',/,F12.7,//,&
         T3,'主特征向量为：',3(/F12.7))
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，如图 4-4 所示。



```
修正Rayleigh加速方法

主特征值为:
 4.3338225

主特征向量为:
 0.6207967
 0.2820007
 0.7314964
```

图 4-4　修正 Rayleigh 商加速方法

同样，可以把计算结果带入原矩阵，看矩阵方程两边是否相等，如果相等表示计算是可靠的，如果差别比较大，则有待于分析查找原因。

4.5　QR分解方法求全部特征值

1. 实验基本原理

QR 算法是计算矩阵特征值问题最有效的方法之一，也是普遍被用于工程实践中的一种方法。QR 方法基于对于任何实的非奇异矩阵都可以分解为正交矩阵 Q 和上三角矩阵 R 乘积，而且当 R 的对角元素符号取定时，分解是唯一的。

QR 方法的基本计算步骤如下，令 $A = A_1$，对 A_1 正交分解，分解为正交矩阵 Q_1 和上三角矩阵 R_1 的乘积

$$A_1 = Q_1 R_1$$

然后将得到的因式矩阵 Q_1 与 R_1 反序相乘，得

$$A_2 = R_1 Q_1$$

以 A_2 代替 A_1，重复以上步骤得到 A_3，所以得到 QR 算法的计算公式为

$$\begin{cases} A_k = Q_k R_k \\ A_{k+1} = R_k Q_k \end{cases}$$

2. 实验目的与要求

- 熟悉 QR 方法的基本思想。

- 理解 QR 方法的计算流程。
- 能够编程实现 QR 方法。
- 给出合理的迭代停止条件，防止进入死循环，同时设置最大允许迭代次数，如果超过次数还没收敛，则停止迭代。

3. 实验内容与数据来源

已知矩阵

$$A=\begin{bmatrix}2 & 1 & 0\\ 1 & 3 & 1\\ 0 & 1 & 4\end{bmatrix}$$

采用 QR 方法计算 A 的全部特征值。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
NAMDA	REAL*8(N)	全部的特征值
输入参变量	**数据类型**	**变量说明**
A	REAL*8(N,N)	要计算的方阵
N	INTEGER	矩阵维数
TOL	REAL*8	误差容限

5 . 实验操作指导

```
module eig qr
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :
!-----------------------------------------------------
!  Description :   QR 分解计算全部特征值模块
!
!-----------------------------------------------------
!  Contains    :
!     1.       方法函数
!     2.       QR 分解函数（采用 G-S 正交化方法）
!-----------------------------------------------------
contains
subroutine solve(A,N,namda,tol)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
```

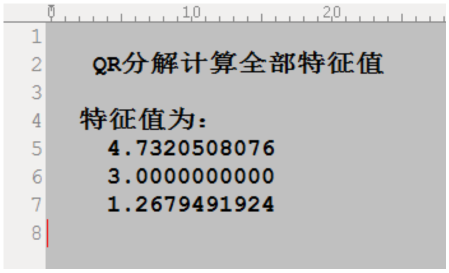
```
!  Purpose    :  QR 分解计算全部特征值
!
!  Post Script :
!      1.      QR 分解采用修正的 G-S 分解方法，当然也可以
!      2.       使用 Householder 或者 Givens 变换方法
!      3.
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A 需要计算的矩阵
!      2.   N 矩阵的维数
!  Output parameters  :
!      1.   namda  特征值组成的向量 N 维
!      2.   tol    用户指定的误差容限
!  Common parameters  :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::N
real*8::A(N,N),namda(N)
real*8::A1(N,N),Q(N,N),R(N,N)
integer::i,j,k
A1=A
!循环迭代,最大允许迭代次
do i=1,200
   call gram dec(A1,Q,R,N,N)
   A1=matmul(R,Q)
! 判断迭代停止标准
   do k=1,N
    ds=0d0
    ds=ds+A1(k,k)**2
   end do
   do j=1,N
   namda(j)=A1(j,j)
   end do
    if (ds<tol) exit
end do
end subroutine solve
subroutine gram_dec(A,Q,R,M,N)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   采用修正的 Gram-Schmidt 分解求矩阵的 QR 分解
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    A 原始矩阵
!      2.    A(M,N)
!  Output parameters  :
!      1.    分解结果为 Q(M,N):注意 Q 不是方阵，Q 列向量为标准
!            正交基
!      2.              R(N,N): R 是方阵
!      3.
```

```
!-----------------------------------------------------
!  Post Script :
!       1.  注意矩阵的维数，分解后 Q 列向量是正交的
!       2.  关于编程方法可以参看《矩阵分析与应用》张贤达编著
!       3.  详细的数学解释，可以参看麻省理工学院的
!           线性代数教材《Linear Algebra with Application》
!-----------------------------------------------------
implicit real*8(a-z)
integer::M,N
integer::i,j,k
real*8::A(M,N),Q(M,N),R(N,N)
real*8::vec_temp(M)
R(1,1)=dsqrt(dot_product(a(:,1),a(:,1)))
Q(:,1)=a(:,1)/R(1,1)
do k=2,N
     do j=1,k-1
       R(j,k)=dot product(Q(:,j),A(:,k))
     end do
     vec_temp=A(:,k)
     do j=1,k-1
       vec_temp=vec_temp-Q(:,j)*R(j,k)
     end do
   R(k,k)=dsqrt(dot product(vec temp,vec temp))
   Q(:,k)=vec_temp/R(k,k)
end do
end subroutine gram dec
end module eig_qr
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.07.06
!------------------------------------------------------
!  Purpose     :  QR 分解计算全部特征值
!
!  Post Script :
!       1.      QR 分解采用的修正的 G-S 正交化方法
!       2.
!
!------------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.    result.txt  结果输出文件
!       2.
!
!------------------------------------------------------
use eig_qr
implicit real*8(a-z)
real*8::A(3,3),namda(3)
open(unit=11,file='result.txt')
A=reshape((/2d0,1d0,0d0,&
         1d0,3d0,1d0,&
         0d0,1d0,4d0/),(/3,3/))
```

```
!调用方法函数
call solve(A,3,namda,1d-7)
write(11,101)namda
101 format(/,T4,'QR 分解计算全部特征值',//,&
         T3,'特征值为：',3(/F16.10))
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，如图 4-5 所示。



```
  QR分解计算全部特征值

 特征值为：
   4.7320508076
   3.0000000000
   1.2679491924
```

图 4-5　QR 分解解算矩阵的特征值

计算结果已经给出了该矩阵的全部特征值。在实际应用时，有时为了减少计算机运算量，常常先通过正交变化把矩阵化为 Hessenberg 矩阵 **H**，然后再对 **H** 作 QR 分解。关于矩阵特征值及特征向量是数值代数一个比较活跃的领域，还有诸多内容没有介绍，有兴趣的读者可以翻阅一些专著，如 Golub 的《MATRIX COMPUTATION》（1996 年 The Johns Hopkins University Press 出版）等。另外一些数值代数包如 LAPACK 等，都提供了相当丰富的特征值问题的处理方法，有兴趣的读者也可以从网上下载并参照说明自行调用。

本章小结

在这一章中介绍了矩阵特征值与特征向量的常用方法，鉴于本书不是一本矩阵分析教材，因此本章省略了一些基础内容，同时也省略了一些数学理论。矩阵特征值问题内容是很丰富的，所以在这一章中介绍的也只是一些比较常用而且实用的算法，对于一些计算效率比较低下的方法，比如 Jacobi 方法这里不再介绍。

计算特征值与特征向量主要有两类方法，一是从原始矩阵出发，求出其多项式，再求出多项式的根。这个方法虽然理论上可行，但实际上效率一般，并不高效，所以基本上不被采用。比较实用的是把特征值和特征向量作为一个序列极限来求得，了解这一思想可以有助于理解本章中各种算法。

第 5 章

非线性方程求根

非线性方程的数值方法是数值计算课程的基本内容之一，广泛存在于科研问题中。对于一个比较大的科研项目，在相当多的时候我们都要面对非线性方程的计算。在本章中，我们按照计算的基本思想把计算方法分为二分法、不动点迭代、牛顿法、割线法等，针对每一种计算思想又分别给出一些改进算法。

就本章介绍的方法而言本身并不难理解，而且一般都有直观的几何意义。在工程实践中，要实现这些方法也不会需要很多代码，比较容易实现。不过，这些方法的基本思想在数学理论中往往有深刻的内涵。

本章我们主要介绍非线性方程

$$f(x)=0$$

的数值计算方法。对于上述方程，除了一些比较特殊的方程可以直接给出根的表达式，一般需要用到迭代法。所以本节需要对迭代法及其相关问题作一简单介绍。

设 $f:A\to A$ 是集合 A 到自身的一个映射，其中集合 A 可以是实数集合或复数集合。这个映射把集合 A 映入自身。同样，我们考虑 f 对自身的复合映射：

$$p\mapsto f(f(p)),\forall p\in A$$

可以把这个映射记作： $f^2:A\mapsto A$ 。这里 f^2 只是一个映射记号，而不是函数的平方。

如果把映射 $f^2:A\mapsto A$ 又可以复合以 f 而得到新的映射： $p\mapsto f(f^2(p)),\forall p\in A$ 。并记为 $f^3:A\mapsto A$。如此反复下去，便得到一个映射序列

$$f,f^2,f^3,\cdots,f^n,\cdots$$

这样便产生一个迭代序列。

对于迭代法，一般需要考虑迭代法的构造、收敛性、收敛速度与误差估计等问题。对于非线性方程 $f(x)=0$ 一个迭代法产生的序列是否收敛于方程的一个根，通常于初值有关，如果从任何可取的初值出发都能保证收敛，则称为大范围收敛。如果只是在初值选取充分接近根时才收敛，则称为局部收敛。

为了衡量迭代的收敛速度，可以用收敛阶数作为一个衡量标准。设一个迭代法收敛，$\lim\limits_{k\to\infty}x_k=p$， p 是方程 $f(x)=0$ 的一个解。令

$$e_k = x_k - p$$

如果存在实数 m 和常数 C，使得

$$\lim_{x\to\infty}\frac{|e_{k+1}|}{|e_k|} = C$$

就称迭代法为 m 阶收敛。M 的大小反映了收敛速度，如果 m=1，则是线性收敛的。对于 m>1 的情况，称为超线性收敛。

为了控制迭代法的终止问题，通常可以选择以下的一些准则。

（1）如果 $|f(x_n)| < TOL$（TOL 为误差容限）停止迭代。但是这样做有一个缺陷，有可能会出现 $f(x_n)$ 很接近 0，但是 x_n 与方程的根相差比较远。

（2）$|x_n - x_{n-1}| < TOL$，则停止迭代。缺点是可能出现 x_n 与 x_{n-1} 很接近，但是 x_n 与方程的根相差比较大。当然这样的情况是和上一种情况不同的，出现这些情况的原因和函数 $f(x)$ 的性质有关。

（3）选择相对误差作为停止迭代的标准 $\dfrac{|x_n - x_{n-1}|}{x_n} < TOL$ 。

实际应用中，应该考虑函数 $f(x)$ 的性质，选择迭代停止的标准，同时比较好的措施可以在程序设计时候考虑增加参数，控制迭代次数，当超过一定次数还没符合控制条件时给出警告信息。

5.1 Bolzano二分法

1. 实验基本原理

Bolzano 二分法又叫区间半分法是一种简单直观的方法，其基本的数学原理基于数学分析中的介值定理。这里简要阐述二分法的数学原理。

对于实函数方程

$$f(x) = 0$$

设函数 $f(x)$ 在区间[a，b]上连续，而且 $f(a)f(b) < 0$，则 $f(x)$ 在区间[a，b]上至少有一个根，这是很容易理解的。

记 $[a,b] = [a_1, b_1]$，设 p_1 为 $[a_1, b_1]$ 中点：

$$p_1 = \frac{a_1 + b_1}{2}$$

事先给定一个误差容限 TOL1（足够小），如果 $|f(p_1)| < TOL1$，则 p_1 是原方程 $f(x) = 0$ 一个很不错的近似根。

如果 $|f(p_1)| > TOL1$，那么我们并不满足于 p_1 作为近似的方程的根。二分法的思想是我们在 $[a_1, p_1]$ 与 $[p_1, b_1]$ 中更细一步的寻找根，具体在哪个区间是很容易实现的。如果 $f(p_1)f(b_1) < 0$

则表明则两点异号，则根必然在$[p_1,b_1]$这个区间里。如果$f(p_1)f(b_1)>0$则根必然在$[a_1,p_1]$这个区间里。这样我们就把原来的我们对根的寻找由$[a_1,b_1]$区间缩小了一半，如此反复，不断缩小区间，当区间缩小到我们可以接受的范围内，我们就用区间里的近似值代替真值。

上述过程是一个迭代过程，判断计算停止的条件之一是 TOL1，最好同时还要给出区间容限 TOL2，即当区间缩小到一定范围以后停止迭代。

下面给出二分法的算法：

输入：区间$[a,b]$，最大允许迭代次数 MAX，误差容限 TOL。

输出：方程的根x，该点的函数值$f(x)$，实际迭代次数$iter$。

处理：01 k=1。

02 当 k<=MAX 时，做03~07处理。

03 计算$c=\dfrac{a+b}{2}$， $f(c)$。

04 如果$f(c)=0$，则停止循环，输出结果。

05 如果$f(a)f(c)<0$，则

$b=c$

否则$a=c$

06 计算$df=|f(a)-f(b)|$（也可以计算$dx=|a-b|$作为判断标准,依需求而定）。

07 如果$df<TOL$,则停止循环。

08 输出方程的根c，该点的函数值$f(c)$，实际迭代次数$iter$。

09 结束。

2. 实验目的与要求

- 明白二分法基本思想。
- 能够给出二分法的伪代码。
- 能够编程实现二分法，处理非线性方程问题。
- 编程中，需要注意精度控制问题。

3. 实验内容与数据来源

使用二分法计算方程函数$f(x)=(x-1)^3-3x+2$在区间$[2,4]$上的根，并同时给出该点的函数值与实际运算迭代次数。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值

ITER	INTEGER	实际迭代次数

5. 程序代码

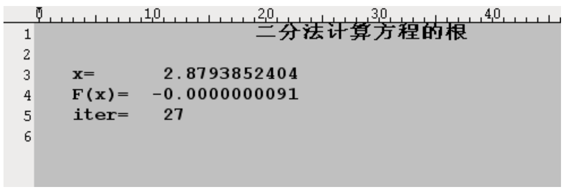
```
module bisect
!-----------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  不动点迭代法计算方程的根
!
!-----------------------------------------------------
!  Parameters  :
!      1.    MAX 最大允许迭代次数
!      2.    tol 误差容限
!      3.    a,b  为区间[a,b]
!  Contains    :
!      1.   方程函数
!      2.   方法函数
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer ::MAX=200
real*8::tol=1D-7
real*8::a=2d0,b=4d0
contains
subroutine solve(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  二分法函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.   x 方程的根
!       2.   fx 该点的函数值
!       3.   iter 实际迭代次数
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer iter,i
do i=1,MAX
```

```
    c=(a+b)/2D0
    if (func(c)==0) exit
    if (func(a)*func(c)<0d0) then
        b=c
    else
        a=c
    end if
    dfx=dabs(func(a)-func(b))
    if (dfx<tol) exit
    !--------这段用于中间输出结果，这里用于分析，实际计算中可以注释掉
    write(102,*)i,c,func(c)
    !-----------
end do
    x=c
!根
    fx=func(c)
!函数值
   iter=i
!实际迭代次数
end subroutine
function func(x)
!--------------------------------function  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  要计算的方程函数，其中 f(x)=0
!
!-----------------------------------------------------
 implicit real*8(a-z)
  func=(x-1d0)**3-3d0*x+2d0
end function func
end module bisect
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010 年月日
!-----------------------------------------------------
!  Purpose   :  二分法主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.    result.txt   计算结果文件
!      2.   IM_result.txt  中间结果
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use bisect
implicit real*8(a-z)
```

```
    integer iter
    open(unit=101,file='result.txt')
    !记录中间结果
    open(unit=102,file='Im_result.txt')
    call solve(x,fx,iter)
    write(101,501)x,fx,iter
    501 format(T20,'二分法计算方程的根',//,&
              3x,'x=    ',F15.10,/,&
              3x,'F(x)=',F15.10,/,&
              3x,'iter=',I5)
    end program main
```

6. 实验结论

程序编译连接，运行后得到结果产生两个文件，计算结果保存在文件 result.txt 文件中，结果如图 5-1 所示。



```
                  二分法计算方程的根

   x=       2.8793852404
   F(x)=   -0.0000000091
   iter=    27
```

图 5-1　二分法计算结果

中间结果保存在文件 Im_result.txt 文件中，中间结果如表 5-1 所示。

表 5-1　二分法计算的中间结果

迭代序列	x	$f(x)$
1	3.00000000000000	1.00000000000000
2	2.50000000000000	-2.12500000000000
3	2.75000000000000	-0.890625000000000
4	2.87500000000000	-3.320312500000000E-002
5	2.93750000000000	0.460693359375000
6	2.90625000000000	0.208160400390625
7	2.89062500000000	8.609390258789062E-002
8	2.88281250000000	2.610063552856445E-002
9	2.87890625000000	-3.637254238128662E-003
10	2.88085937500000	1.121016591787338E-002
11	2.87988281250000	3.781077452003956E-003
12	2.87939453125000	7.056735921651125E-005
13	2.87915039062500	-1.783679457730614E-003
14	2.87927246093750	-8.566400592826540E-004
15	2.87933349609375	-3.930573532215931E-004
16	2.87936401367188	-1.612502478849365E-004
17	2.87937927246094	-4.534275706546964E-005
18	2.87938690185547	1.261197289181837E-005

19	2.87938308715820	-1.636547413319533E-005
20	2.87938499450684	-1.876771132280908E-006

（续表）

迭代序列	x	$f(x)$
21	2.87938594818115	5.367595751870624E-006
22	2.87938547134399	1.745411028153399E-006
23	2.87938523292542	-6.568037314025332E-008
24	2.87938535213470	8.398652475705148E-007
25	2.87938529253006	3.870924176752055E-007
26	2.87938526272774	1.607060173824948E-007
27	2.87938524782658	4.751282123294231E-008

表 5-1 给出了计算机处理的详细过程，其中第一列为实际迭代序列，第二列为给出的方程的根，可以看出逐步逼近要计算的结果，第三列为根对应的函数值，可以看到函数值越来越接近零。

二分法对函数的要求较低，计算思想朴素直观，也很容易用计算机编程实现，不过二分法收敛速度不是很快。有时候，可以把这个方法计算的值作为初始值的近似。

5.2 Picard迭代法

1. 实验基本原理

对于非线性方程

$$f(x)=0$$

常常可以化成等价的方程

$$x=g(x)$$

可以选取一个初始近似值 x_0，构造迭代序列

$$x_k=g(x_{k-1}),k=1,2,\cdots$$

如此产生序列$\{x_k\}$。这种迭代方法称为不动点迭代，或 Picard 迭代。这个原理看似很容易直观理解，但是却有相当深刻的数学内涵，数学系的学生在泛函分析与微分方程理论等多门专业课程里都会遇到。如果 $g(x)$ 连续，而且 $\lim\limits_{k\to\infty} x_k=p$，则 p 是 g 的一个不动点。因此 p 为方程 $f(x)=0$ 的一个根。

下面的定理给出了迭代收敛性问题。假设 $g(x)$ 为定义在区间[a,b]上的实函数，它满足以下条件：

（1）$g(x)\in[a,b],\forall x\in[a,b]$；

（2）利普希次条件，且利普希次常数 L<1，即存在正常数 L<1，使得

$$\left|g(x)-g(y)\le L\left|x-y\right|,\right|\forall x,y\in[a,b]$$

那么对任意的初始值 $\forall x_0 \in [a,b]$，由于 Picard 迭代产生的序列都收敛于唯一的不动点 p。下面给出不动点迭代的算法：

输入：最大允许迭代次数 MAX，误差容限 TOL，迭代初值 x_0。

输出：方程的根 x，该点的函数值 $f(x)$，实际迭代次数 $iter$。

处理：01 k=1。

02 令 $x_1 = x_0$

03 当 k<=MAX 时，做 04~07 处理。

04 计算 $f(x_1)$。

05 令 $x_2 = f(x_1) + x_1$。

06 计算 $dx = |x_2 - x_1|$。

07 如果 $dx < TOL$，则停止循环。

08 令 $x_1 = x_2$。

09 输出方程的根 $c = x_2$，该点的函数值 $f(c)$，实际迭代次数 $iter$。

10 结束。

2. 实验目的与要求

- 熟悉不动点迭代的基本原理。
- 了解不动点迭代的计算流程。
- 能编程实现不动点迭代方法，计算非线性方程问题。

3. 实验内容与数据来源

计算 4 阶的勒让德多项式等于 0 在宗量等于 0.3 附近的根，其中 4 阶的勒让德多项式为

$$P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$$

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值
ITER	INTEGER	实际迭代次数

5. 程序代码

```
module picard
!----------------------------------------module coment
!  Version   :  V1.0
```

```
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  不动点迭代法计算方程的根
!
!-----------------------------------------------------
!  Parameters  :
!      1.     MAX 最大允许迭代次数
!      2.     tol 误差容许限
!      3.     迭代初值
!  Contains    :
!      1.    solve 不动点迭代方法函数
!      2.    func  要计算的方程函数
!-----------------------------------------------------
implicit real*8(a-z)
integer:: MAX=200
real*8::tol=1D-7
real*8::x0=0.3
contains
subroutine solve(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  不动点迭代计算方程的根
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.   x 方程的根
!       2.   fx 该点的函数值
!       3.   iter 实际迭代次数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer i,iter
x1=x0
do i=1,MAX
   x2=func(x1)+x1
   dx=dabs(x2-x1)
   !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
   write(102,*)i,x2,func(x2)
   !---------
   if (dx<tol) exit
   x1=x2
end do
   x=x2
```

```
    fx=func(x2)
    iter=i
end subroutine solve
function func(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    x 自变量
!      2.
!  Output parameters  :
!      1.   func 方程函数值
implicit real*8(a-z)
  func=35d0*x**4-30*x**2+3d0
  func=func/8d0
end function func
end module  picard
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   不动点迭代计算方程的根主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.   result.txt 计算结果
!      2.   IM_result.txt 计算的中间结果
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use picard
implicit real*8(a-z)
integer::iter
open(unit=101,file='result.txt')
open(unit=102,file='Im_result.txt')
call solve(x,fx,iter)
write(101,501)x,fx,iter
501 format(T20,'不动点迭代法计算方程的根',//,&
          3x,'x=   ',F15.10,/,&
          3x,'F(x)=',F15.10,/,&
          3x,'iter=',I5)
end program main
```

6．结论与分析

程序运行后产生两个数据文件，其中 result.txt 存放计算结果，如图 5-2 所示。

```
                    不动点迭代法计算方程的根

   x=        0.3399809991
   F(x)=     0.0000000829
   iter=     92
```

图 5-2　不动点迭代法计算结果

文件 lm_result.txt 记录了计算机计算的详细经过，如表 5-2 所示。

表 5-2　不动点迭代序列

迭代序列	x	$f(x)$
1	0.3729374907	-0.0619293054
2	0.3110081854	0.0532093806
3	0.3642175660	-0.0454663861
4	0.3187511799	0.0391545777
5	0.3579057576	-0.0335738366
6	0.3243319210	0.0289432142
7	0.3532751351	-0.0248680275
8	0.3284071077	0.0214474013
9	0.3498545089	-0.0184499173
10	0.3314045916	0.0159142811
⋮	⋮	⋮
80	0.3399807796	0.0000004916
81	0.3399812712	-0.0000004238
82	0.3399808473	0.0000003654
83	0.3399812128	-0.0000003151
84	0.3399808977	0.0000002716
85	0.3399811693	-0.0000002342
86	0.3399809352	0.0000002019
87	0.3399811371	-0.0000001741
88	0.3399809630	0.0000001501
89	0.3399811131	-0.0000001294
90	0.3399809837	0.0000001116
91	0.3399810952	-0.0000000962
92	0.3399809991	8.29160337E-8

同样的一个非线性方程，可以选择不同的不动点迭代形式，往往不同形式的迭代收敛速度是不同的，如果选择不合适，甚至出现不收敛的情况。一般而言，不动点迭代法，收敛速度

依然不是很快，有必要寻求其他方法。

5.3　Aitken加速与Steffensen迭代方法

1. 实验基本原理

不动点迭代法虽然简单，容易实现，但往往有时候收敛速度不够快。在这样的情况下，我们通常会考虑加速收敛。

假设一个序列$\{x_n\}: x_0, x_1, x_2 \cdots$线性收敛于 p，则有

$$\lim_{n\to\infty}\frac{x_{n+1}-p}{x_n-p}=\lambda,(\lambda\neq 0)$$

当 n 足够大时，有

$$\frac{x_{n+1}-p}{x_n-p}\approx\frac{x_{n+2}-p}{x_{n+1}-p}$$

把上式展开，我们可以得到

$$p\approx x_n-\frac{\left(x_{n+1}-x_n\right)^2}{x_{n+2}-2x_{n+1}+x_n}$$

这样我们可以定义

$$\tilde{x}_{n+1}\approx x_n-\frac{\left(x_{n+1}-x_n\right)^2}{x_{n+2}-2x_{n+1}+x_n}$$

作为迭代方法，这就是 Aitken 迭代。

如果我们采用差分符号那么记法会更简洁。可以定义一阶差分

$$\Delta x_n = x_{n+1}-x_n$$

二阶差分

$$\begin{aligned}\Delta^2 x_n &= \Delta(\Delta x_n)\\ &= x_{n+2}-2x_{n+1}+x_n\end{aligned}$$

那么 Aitken 迭代可以表示为

$$\tilde{x}_{n+1}\approx x_n-\frac{(\Delta x_n)^2}{\Delta^2 x_n}$$

Aitken 加速方法不管原序列$\{x_k\}$是如何产生的，如果把 Aitken 方法与不动点迭代法结合，则得到如下迭代格式

$$y_k = g(x_k), z_k = g(y_k)$$
$$x_{k+1} = x_k - \frac{(y_k - x_k)^2}{z_k - 2y_k + x_k}, k = 0,1,2\cdots$$

此即 Steffensen 迭代法。

Steffensen 迭代法解方程 $x = g(x)$ 可以看成是另一种不动点迭代：

$$x_{k+1} = g(x_k), k = 0,1,2\cdots$$

其中迭代函数为

$$\varphi(x) = x - \frac{[g(x) - x]^2}{g(g(x)) - 2g(x) + x}$$

由于 Steffensen 方法实际上已经包含了 Aitken 方法，更为实用，给出 Aitken 方法已经没有太大意义，这里编程就直接给出 Steffensen 方法。这里给出算法：

输入：最大允许迭代次数 MAX，误差容限 TOL，迭代初值 x_0。

输出：方程的根 x，该点的函数值 $f(x)$，实际迭代次数 $iter$。

处理：01 k=1。

02 令 $x_1 = x_0$

03 当 k<=MAX 时，做03~09处理。

04 计算 $x_2 = f(x_1) + x_1$。

05 计算 $x_3 = f(x_2) + x_2$。

06 计算 $\bar{x} = x_1 - \dfrac{(x_2 - x_1)^2}{(x_3 - 2x_2 + x_1)}$。

07 计算 $dx = |\bar{x} - x_1|$。

08 如果 $dx < TOL$，则停止循环。

09 令 $f(x)$。

10 输出方程的根 $c = \bar{x}$，该点的函数值 $f(c)$，实际迭代次数 $iter$。

11 结束。

2. 实验目的与要求

- 理解 Aitken 加速方法的基本原理。
- 能自己推导出 Aitken 加速方法。
- 理解 Steffensen 方法的原理。
- 能够编程实现 Steffensen 方法。
- 对比不动点迭代法，比较收敛速度。

3. 实验内容与数据来源

采用加速方法计算方程

$$e^{-x}-x=0$$

在 $x=0.5$ 附近的根。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值
ITER	INTEGER	实际迭代次数

5. 程序代码

```
module Steffensen
!----------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  Steffensen 加速计算方程的根
!
!-----------------------------------------------------
!  Parameters  :
!      1.     MAX 最大允许迭代次数
!      2.     tol 误差容许限
!      3.     迭代初值
!  Contains    :
!      1.    solve  Steffensen 方法函数
!      2.    func   要计算的方程函数
!      3.    picard  不动点迭代法的方法函数
!-----------------------------------------------------
implicit real*8(a-z)
integer:: MAX=200
real*8::tol=1D-7
real*8::x0=0.5d0
contains
subroutine solve(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  Steffensen 加速方法处理函数
!
!-----------------------------------------------------
!  Input  parameters  :
```

```
!       1.
!       2.
!  Output parameters  :
!       1.   x 方程的根
!       2.   fx 该点的函数值
!       3.   iter 实际迭代次数
!-----------------------------------------------------
implicit real*8(a-z)
integer i,iter
!这句为标题，计算可以注释掉
   write(102,*)'Steffensen 迭代法中间结果'
!------------------
x1=x0
do i=1,MAX
   x2=func(x1)+x1
   x3=func(x2)+x2
   x1_bar=x1-(x2-x1)**2/(x3-2*x2+x1)
   dx=dabs(x1_bar-x1)
   !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
   write(102,*)i,x1_bar,func(x2)
   !---------
   if (dx<tol) exit
   x1=x1_bar
end do
   x=x1_bar
   fx=func(x2)
   iter=i
end subroutine solve
subroutine picard(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  不动点迭代计算方程的根
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.   x 方程的根
!       2.   fx 该点的函数值
!       3.   iter 实际迭代次数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1. 此函数用于对比 Steffensen 方法，实际使用时可以删除该 Subroutine
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer i,iter
!这句为标题，计算可以注释掉
   write(104,*)'picard 迭代法中间结果'
```

```
!------------------
x1=x0
do i=1,MAX
    x2=func(x1)+x1
    dx=dabs(x2-x1)
    !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
    write(104,*)i,x2,func(x2)
    !---------
    if (dx<tol) exit
    x1=x2
end do
    x=x2
    fx=func(x2)
    iter=i
end subroutine picard
function func(x)
!---------------------------------function  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.   func 方程函数值
!-----------------------------------------------------
implicit real*8(a-z)
 func=dexp(-x)-x
end function func
end module  Steffensen
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  Steffensen 迭代法主函数
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.   result.txt 计算结果
!       2.   IM result.txt 计算的中间结果
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
use Steffensen
implicit real*8(a-z)
```

```
    integer::iter_s,iter_p
    open(unit=101,file='result_s.txt')
    open(unit=102,file='Im_result_s.txt')
    open(unit=103,file='result_p.txt')
    open(unit=104,file='Im_result_p.txt')
    call picard(x_p,fx_p,iter_p)
    write(103,501)x_p,fx_p,iter_p
    501 format(T20,'Picard不动点迭代法计算方程的根',//,&
              3x,'x=   ',F15.10,/,&
              3x,'F(x)=',F15.10,/,&
              3x,'iter=',I5)
    call solve(x_s,fx_s,iter_s)
    write(101,502)x_s,fx_s,iter_s
    502 format(T20,'Steffensen加速计算方程的根',//,&
              3x,'x=   ',F15.10,/,&
              3x,'F(x)=',F15.10,/,&
              3x,'iter=',I5)
    end program main
```

6. 实验结论

程序运行产生 4 个数据文件，其中 result_a.txt 用于存放 Steffensen 加速方法的计算结果、result_p.txt 用于存放 Picard 迭代法的计算结果、Im_reslut_a.txt 用于存放 Steffensen 方法的中间结果、Im_result_p.txt 用于存放 Picard 方法的中间结果。

Steffensen 方法结果如图 5-3 所示。

Picard 方法计算结果如图 5-4 所示。

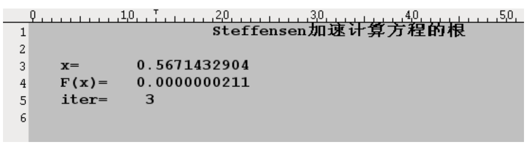


图 5-3　Steffensen 加速方法计算结果

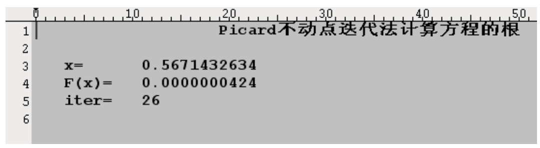


图 5-4　Picard 迭代法计算结果

Steffensen 方法计算的中间结果如表 5-3 所示。

表 5-3　Steffensen 方法计算的中间结果

迭代序列	x	$f(x)$
1	0.567623876410920	-6.129144782002838E-002
2	0.567143314105564	4.270607751841737E-004
3	0.567143290409784	2.106068552887308E-008

Picard 方法计算的中间结果如表 5-4 所示。

表 5-4　Picard 迭代的比对数据

迭代序列	x	$f(x)$
1	0.60653066	-0.06129145
2	0.54523921	0.03446388

（续表）

迭代序列	x	$f(x)$
3	0.57970309	-0.01963847
4	0.56006463	0.01110752
5	0.57117215	-0.00630920
6	0.56486295	0.00357510
7	0.56843805	-0.00202859
8	0.56640945	0.00115018
9	0.56755963	-0.00065242
10	0.56690721	0.00036998
11	0.56727720	-0.00020984
12	0.56706735	0.00011901
13	0.56718636	-0.00006750
14	0.56711886	0.00003828
15	0.56715714	-0.00002171
16	0.56713543	0.00001231
17	0.56714775	-0.00000698
18	0.56714076	0.00000396
19	0.56714472	-0.00000225
20	0.56714248	0.00000127
21	0.56714375	-0.00000072
22	0.56714303	0.00000041
23	0.56714344	-0.00000023
24	0.56714321	0.00000013
25	0.56714334	-0.00000007
26	0.56714326	4.239169E-008

可以看到，Picard 迭代进行了 26 次，而采用加速方法后，仅迭代三次就已经达到与之相当的计算精度。

有些情况下，不动点迭代法不收敛，采用 Steffensen 迭代法可能收敛，至于原先就收敛的迭代法，采用 Steffensen 迭代，则可以达到二阶收敛。

5.4　Newton-Raphson迭代法

1. 实验基本原理

Newton—Rahpson 迭代方法是解非线性方程比较著名而且也比较有效的方法之一。如果初值比较接近根，收敛速度是很快的。Newton—Rahpson 迭代法也是工程上广泛采用的方法。

Newton 迭代法有比较直观的几何意义。函数方程 $f(x)=0$ 的根是曲线 $y=f(x)$ 与 x 轴的

交点的横坐标。通过当前的点做曲线的切线（与一次导数有关），切线方程为

$$y = f(x_k) + f'(x_k)(x - x_k)$$

在上式中令 y=0 得到切线与 x 轴交点的横坐标

$$x = x_k - \frac{f(x_k)}{f'(x_k)}$$

于是我们可以构造迭代公式

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

如此反复迭代便可以比较快的得到方程的根，所以 Newton 法又叫切线法。

关于 Newton 迭代法如果函数 $f(x)$ 有二阶以上连续倒数，p 是方程 $f(x)=0$ 的单根，则当 x_0 充分接近 p 时，Newton 迭代法收敛，而且最少二阶收敛。算法如下

输入：最大允许迭代次数 MAX，误差容限 TOL，迭代初值 x_0。

输出：方程的根 x，该点的函数值 $f(x)$，实际迭代次数 $iter$。

处理：01 k=1。

02 令 $x_1 = x_0$

03 当 k<=MAX 时，做 04~07 处理。

04 计算 $x_2 = x_1 - \dfrac{f(x_1)}{f'(x_1)}$。

05 计算 $dx = |x_2 - x_1|$。

06 如果 $dx < TOL$，则停止循环。

07 令 $x_1 = x_2$。

08 输出方程的根 $c = x_2$，该点的函数值 $f(c)$，实际迭代次数 $iter$。

09 结束。

2. 实验目的与要求

- 理解牛顿迭代法的几何意义与数学内涵。
- 明白迭代函数构造原理。
- 牛顿迭代法是最重要的计算非线性方程方法之一，所以要求熟练掌握，能够编程实现。

3. 实验内容与数据来源

用 Newton-Raphson 迭代方法计算方程

$$f(x) = x^3 + 2x^2 + 10x - 20 = 0$$

在[1，2]区间内的一个根。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值
ITER	INTEGER	实际迭代次数

5. 程序代码

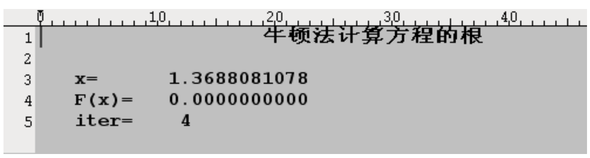
```
module newton
!----------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  牛顿迭代计算非线性方程
!
!-----------------------------------------------------
!  Parameters  :
!      1.     MAX 最大允许迭代次数
!      2.     tol 误差容许限
!      3.     迭代初值
!  Contains    :
!      1.    solve 牛顿迭代方法函数
!      2.    func  要计算的方程函数
!      3.    dfunc 导函数
!-----------------------------------------------------
implicit real*8(a-z)
integer:: MAX=200
real*8::tol=1D-7
real*8::x0=1.5d0
contains
subroutine solve(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  牛顿法计算方程的根
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.    x 方程的根
!      2.    fx 该点的函数值
!      3.    iter 实际迭代次数
!  Common parameters  :
!
!----------------------------------------------------
```

```
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer i,iter
x1=x0
do i=1,MAX
   x2=x1-func(x1)/dfunc(x1)

   dx=dabs(x2-x1)
   if (dx<tol) exit
   x1=x2

   !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
   write(102,*)i,x2,func(x2)
   !---------
end do
   x=x2
   fx=func(x2)
   iter=i-1
end subroutine solve
function func(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    x 自变量
!      2.
!  Output parameters  :
!      1.   func 方程函数值
!----------------------------------------------------
implicit real*8(a-z)
  func=x**3+2*x**2+10*x-20d0
end function func
function dfunc(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程导函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    x 自变量
!      2.
!  Output parameters  :
!      1.   dfunc 方程导函数值
!----------------------------------------------------
```

```
implicit real*8(a-z)
 dfunc=3*x**2+4*x+10d0
end function dfunc
end module  newton
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   牛顿法计算方程的根主函数
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.   result.txt 计算结果
!       2.   IM_result.txt 计算的中间结果
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
use newton
implicit real*8(a-z)
integer::iter
open(unit=101,file='result.txt')
open(unit=102,file='Im_result.txt')
call solve(x,fx,iter)
write(101,501)x,fx,iter
501 format(T20,'牛顿法计算方程的根',//,&
          3x,'x=   ',F15.10,/,&
          3x,'F(x)=',F15.10,/,&
          3x,'iter=',I5)
end program main
```

6. 实验结论

程序运行后产生两个数据文件，其中 result.txt 记录了计算结果，如图 5-5 所示。



```
                   牛顿法计算方程的根

   x=        1.3688081078
   F(x)=     0.0000000000
   iter=       4
```

图 5-5　牛顿迭代法计算结果

可以看到牛顿迭代法收敛速度还是相当快的，计算的中间数据如表 5-5 所示。

表 5-5 牛顿迭代法计算的中间数据

迭代序列	x	$f(x)$
1	1.37362637362637	0.101788683481715
2	1.36881481962396	1.415933977746420E-004
3	1.36880810783441	2.750830674358440E-010
4	1.36880810782137	0.00000000000000E+000

从程序中可以看出，牛顿法迭代一次，分别需要计算方程函数值一次和导函数值一次。

5.5 重根时的迭代改进

1. 实验基本原理

Newton 迭代法是一种实用性比较强的算法，但是在处理重根运算时候往往效果不是太好，有时候收敛速度会非常的慢。为了处理这一问题，往往我们需要对迭代法进行一些改造。

如果采取以下的迭代方法，计算效果会很不错，但是需要计算二次导数。

$$x_k = x_{k-1} - \frac{F(x_{k-1})}{F'(x_{k-1})}，\text{其中 } F(x) = \frac{f(x)}{f'(x)}$$

展开上市式，便得到迭代公式

$$x_k = x_{k-1} - \frac{f(x_{k-1})f'(x_{k-1})}{[f'(x_{k-1})]^2 - f(x_{k-1})f''(x_{k-1})}$$

这里给出算法如下：

输入：最大允许迭代次数 MAX，误差容限 TOL，迭代初值 x_0。

输出：方程的根 x，该点的函数值 $f(x)$，实际迭代次数 $iter$。

处理：01 k=1。

02 令 $x_1 = x_0$

03 当 k<=MAX 时，做04~09处理。

04 计算 $temp1 = f(x)f'(x)$

05 计算 $temp2 = \left[f'(x)\right]^2 - f(x)f''(x)$。

06 令 $x_2 = x_1 - \dfrac{temp1}{temp2}$。

07 $dx = |x_2 - x_1|$。

08 如果 $dx < TOL$，则停止循环。

09 $x_1 = x_2$。

10 输出方程的根 $c = x_2$，该点的函数值 $f(c)$，实际迭代次数 $iter$。

11 结束。

2. 实验目的与要求

- 理解公式的构造方法。
- 会正确使用该公式，知道计算步骤。
- 能编程实现该计算方法。

3. 实验内容与数据来源

分别使用 Newton 迭代法与本实验的加速方法计算以下方程在 $x=1.5$ 附近的根。

$$f(x)=x^4-4x^2+4=0$$

比较收敛速度。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值
ITER	INTEGER	实际迭代次数

5. 程序代码

```
module multiroot
!----------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  重根时改进算法模块
!
!-----------------------------------------------------
!  Parameters  :
!      1.     MAX 最大允许迭代次数
!      2.     tol 误差容许限
!      3.     迭代初值
!  Contains    :
!      1.    solve 牛顿迭代方法函数
!      2.    func  要计算的方程函数
!      3.    dfunc 导函数
!-----------------------------------------------------
implicit real*8(a-z)
integer:: MAX=200
real*8::tol=1D-7
real*8::x0=1.5d0
contains
subroutine solve(x,fx,iter)
```

```
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  重根改进算法
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.   x 方程的根
!      2.   fx 该点的函数值
!      3.   iter 实际迭代次数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.   引用函数 func  dfunc  d2func
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer i,iter
x1=x0
!标题，实际计算时可以注释掉
write(102,501)
501 format(T20,'重根计算的中间结果')
! --------------
do i=1,MAX
   temp1=func(x1)*dfunc(x1)
   temp2=dfunc(x1)**2-func(x1)*d2func(x1)
   x2=x1-temp1/temp2
   dx=dabs(x2-x1)
   if (dx<tol) exit
   x1=x2
   !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
   write(102,*)i,x2,func(x2)
   !---------
end do
   x=x2
   fx=func(x2)
   iter=i-1
end subroutine solve
subroutine newton(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  牛顿法计算方程的根
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
```

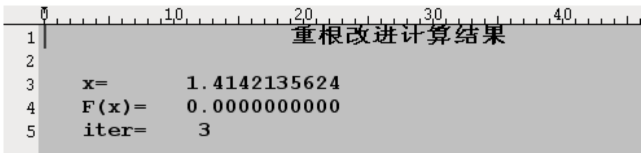
```
!       2.
!  Output parameters  :
!       1.   x 方程的根
!       2.   fx 该点的函数值
!       3.   iter 实际迭代次数
!  Common parameters  :
!
!-----------------------------------------------------
!  Post Script :
!       1.   引用函数 func  dfunc
!
!-----------------------------------------------------
implicit real*8(a-z)
integer i,iter
!标题，实际计算时可以注释掉
write(104,502)
502 format(T20,'牛顿法计算的中间结果')
! --------------
x1=x0
do i=1,MAX
    x2=x1-func(x1)/dfunc(x1)
    dx=dabs(x2-x1)
    if (dx<tol) exit
    x1=x2
    !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
    write(104,*)i,x2,func(x2)
    !---------
end do
    x=x2
    fx=func(x2)
    iter=i-1
end subroutine newton
function func(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.   func 方程函数值
!-----------------------------------------------------
implicit real*8(a-z)
   func=x**4-4*x**2+4d0
end function func
function dfunc(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
```

```
!---------------------------------------------------------
!  Purpose   :  方程导函数
!
!---------------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.   dfunc 方程导函数
!--------------------------------------------------------
implicit real*8(a-z)
  dfunc=4*x**3-8*x
end function dfunc
function d2func(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!---------------------------------------------------------
!  Purpose   :  方程二阶导函数
!
!---------------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.   d2func 方程二阶导数
!--------------------------------------------------------
implicit real*8(a-z)
d2func=12*x**2-8d0
end function d2func
end module  multiroot
program main
!------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!---------------------------------------------------------
!  Purpose   :   重根问题主函数
!
!---------------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.   result_m.txt 计算结果
!       2.   IM_result_m.txt 计算的中间结果
!       3.   result_n.txt 牛顿法计算结果
!       4.   Im_result_n.txt 牛顿法计算的中间结果
!---------------------------------------------------------
use multiroot
implicit real*8(a-z)
integer::iter m,iter n
open(unit=101,file='result_m.txt')
open(unit=102,file='Im_result_m.txt')
```

```
    open(unit=103,file='result_n.txt')
    open(unit=104,file='Im_result_n.txt')
    call solve(x_m,fx_m,iter_m)
    write(101,501)x_m,fx_m,iter_m
    501 format(T20,'重根改进计算结果',//,&
              3x,'x=   ',F15.10,/,&
              3x,'F(x)=',F15.10,/,&
              3x,'iter=',I5)
    call newton(x_n,fx_n,iter_n)
    write(103,502)x_n,fx_n,iter_n
    502 format(T20,'牛顿法计算结果',//,&
              3x,'x=   ',F15.10,/,&
              3x,'F(x)=',F15.10,/,&
              3x,'iter=',I5)
    end program main
```

6. 实验结论

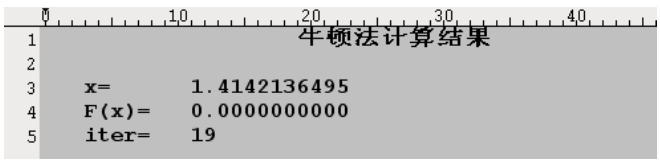
程序运行后生成4个数据文件，其中result_m.txt用于保存改进后方法的计算结果，如图5-6所示。



```
                   重根改进计算结果

   x=       1.4142135624
   F(x)=    0.0000000000
   iter=     3
```

图5-6　改进方法计算结果

文件result_n.txt保存采用原始的牛顿法计算结果，如图5-7所示。



```
                   牛顿法计算结果

   x=       1.4142136495
   F(x)=    0.0000000000
   iter=    19
```

图5-7　牛顿迭代法计算情况

文件Im_result_m.txt记录了计算机采用改进方法计算的中间结果，如表5-6所示。

表5-6　改进方法计算的中间结果

迭代序列	x	$f(x)$
1	1.41176470588235	4.789214688472043E-005
2	1.41421143847482	3.608757737083579E-011
3	1.41421356238098	0.000000000000000E+000

文件Im_result_n.txt记录了直接采用牛顿法计算的中间结果，如表5.7所示。

表 5-7 直接采用牛顿法计算的中间结果

迭代序列	x	$f(x)$
1	1.45833333333333	1.606204185956894E
2	1.43660714285714	4.075556185719975E
3	1.42549761941756	1.026783518798702E
4	1.41987792168383	2.577088417465845E
5	1.41705139127582	6.455552857964619E
6	1.41563389760442	1.615503010210517E
7	1.41492408625178	4.040782846992386E
8	1.41456891351237	1.010449306448891E
9	1.41439126025950	2.526440523453743E
10	1.41430241689773	6.316498080138899E
11	1.41425799103102	1.579174035981623E
12	1.41423577705225	3.947996596309622E
13	1.41422466980323	9.870078088169976E
14	1.41421911610984	2.467528403826691E
15	1.41421633924764	6.168932031869190E
16	1.41421495079054	1.542055372283357E
17	1.41421425663102	3.855582519918244E
18	1.41421390953562	9.645617637943360E
19	1.41421373588471	2.398081733190338E

计算结果显示，牛顿法需要迭代 19 次，而改进方法仅仅 3 次就达到与之相当的精度，不过改进方法的缺点是需要计算二阶导数。

5.6 割线法

1. 实验基本原理

在 Newton 法计算 $f(x)=0$ 中，迭代过程里下一步是用曲线 $y=f(x)$ 的切线代替曲线 $y=f(x)$，从而把切线与 x 轴的交点横坐标作为 $f(x)=0$ 的根，如此反复。要计算切线的代价是需要计算函数 $y=f(x)$ 的导数。事实上，我们可以考虑迭代前后两次的割线来代替曲线，把割线与 x 轴的交点作为根的近似，如此反复，这样也得到一种求解 $f(x)=0$ 的计算方法，而不需要计算一次导数。

割线的方程为

$$y=f(x_k)+\frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}(x-x_k)$$

该直线方程与 x 轴交点横坐标容易算出：

$$x=x_k-\frac{f(x_k)(x_k-x_{k-1})}{f(x_k)-f(x_{k-1})}$$

我们就把这个 x 作为下一次迭代的初值，那么得到迭代公式

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

这里给出算法如下：

输入：最大允许迭代次数 MAX，误差容限 TOL，迭代区间初值 $[a,b]$。

输出：方程的根 x，该点的函数值 $f(x)$，实际迭代次数 $iter$。

处理：
01 k=1。
02 令 $x_0 = a, x_1 = b$。
03 当 k<=MAX 时，做 04~07 处理。
04 计算 $x_2 = x_1 - \dfrac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_1)$。
05 $dx = |x_2 - x_1|$。
06 如果 $dx < TOL$，则停止循环。
07 $x_1 = x_2$。
08 输出方程的根 $c = x_2$，该点的函数值 $f(c)$，实际迭代次数 $iter$。
09 结束。

2. 实验目的与要求

- 从几何的角度理解割线法的几何意义。
- 比较割线法和牛顿迭代法。
- 能在数字计算机上实现割线法，解决非线性方程问题。

3. 实验内容与数据来源

采用割线法计算求方程

$$f(x) = 2x^3 - 5x - 1 = 0$$

在 $[1,2]$ 中的一个根，取 $x_0 = 2, x_1 = 1$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值
ITER	INTEGER	实际迭代次数

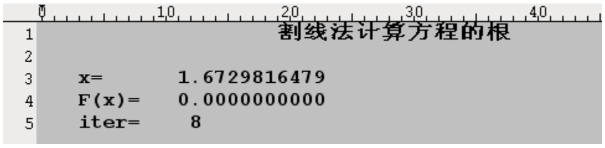
5. 程序代码

```
module secant
!---------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-3
!-----------------------------------------------------
!  Purpose   :  割线法计算非线性方程
!
!-----------------------------------------------------
!  Parameters  :
!      1.     MAX 最大允许迭代次数
!      2.     tol 误差容许限
!      3.     a,b 区间初值
!  Contains    :
!      1.    solve 牛顿迭代方法函数
!      2.    func  要计算的方程函数
!-----------------------------------------------------
implicit real*8(a-z)
integer:: MAX=200
real*8::tol=1D-7
real*8::a=2d0,b=1d0
contains
subroutine solve(x,fx,iter)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   : 割线法计算方程的根
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.   x 方程的根
!       2.   fx 该点的函数值
!       3.   iter 实际迭代次数
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer i,iter
x0=a
x1=b
do i=1,MAX
    x2=x1-(x1-x0)*func(x1)/(func(x1)-func(x0))
    dx=dabs(x2-x0)
    if (dx<tol) exit
    x0=x1
    x1=x2
```

```
    !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
    write(102,*)i,x2,func(x2)
    !---------
end do
    x=x2
    fx=func(x2)
    iter=i-1
end subroutine solve
function func(x)
!---------------------------------function  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    x 自变量
!      2.
!  Output parameters  :
!      1.   func 方程函数值
!----------------------------------------------------
implicit real*8(a-z)
  func=2*x**3-5*x-1
end function func
end module  secant
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   割线法计算方程的根主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.   result.txt 计算结果
!      2.   IM_result.txt 计算的中间结果
!-----------------------------------------------------
use secant
implicit real*8(a-z)
integer::iter
open(unit=101,file='result.txt')
open(unit=102,file='Im_result.txt')
call solve(x,fx,iter)
write(101,501)x,fx,iter
501 format(T20,'割线法计算方程的根',//,&
          3x,'x=   ',F15.10,/,&
          3x,'F(x)=',F15.10,/,&
          3x,'iter=',I5)
end program main
```

6. 实验结论

程序运行后产生两个数据文件，其中 result.txt 为输出结果，如图 5-8 所示。



```
                    割线法计算方程的根

   x=        1.6729816479
   F(x)=     0.0000000000
   iter=      8
```

图 5-8　割线法计算结果

Im_result.txt 记录了中间结果，如表 5-8 所示。

表 5-8　割线法计算的中间结果

迭代序列	x	$f(x)$
1	1.44444444444444	-2.19478737997257
2	1.98480243161094	4.71401077338144
3	1.61610539973298	-0.638649618358173
4	1.66009627557147	-0.150297503747964
5	1.67363511082663	7.710710029897427E-003
6	1.67297442344653	-8.519840962684100E-005
7	1.67298164383841	-4.736775061076060E-008
8	1.67298164785497	2.913225216616411E-013

割线法与牛顿法相比，每迭代一次只需要计算一次方程函数，而牛顿法则还需要计算导函数值，在导函数计算比较费事时，则割线法凸显其优点，不过一般而言割线法的收敛速度略慢于牛顿法。

5.7　多重迭代法

1. 实验基本原理

若给定两个迭代函数 $\varphi(x),\psi(x)$，可以构造多重迭代法

$$\begin{cases} y_k = \varphi(x_k) \\ x_{k+1} = \psi(x_k) \end{cases} \quad k = 0,1,\cdots$$

对于方程 $f(x)=0$，给定迭代函数 $\varphi(x)$，构造多重迭代函数

$$g(x) = \varphi(x) - \frac{f(\varphi(x))}{f'(x)}$$

相应迭代法写成

$$\begin{cases} y_k = \varphi_k(x) \\ x_{k+1} = y_k - \dfrac{f(y_k)}{f'(x_k)} \end{cases} \quad k = 0,1,\cdots$$

如果取

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

即取 $\varphi(x)$ 为牛顿法迭代函数，此时上述迭代为 3 阶收敛，迭代格式可以写成

$$\begin{cases} y_k = x_k - \dfrac{f(x_k)}{f'(x_k)} \\ x_{k+1} = y_k - \dfrac{f(y_k)}{f'(x_k)} \end{cases} \quad k = 0,1,\cdots$$

上式每步迭代只需计算两次函数值，与一次导函数值。

这里给出算法如下：

输入：最大允许迭代次数 $IMAX$，误差容限 TOL，迭代初值 x_0。

输出：方程的根 x，函数值 $f(x)$，实际迭代次数 iter。

处理：01　$x_1 = x_0$。

02　对 $k = 1,2,\cdots,IMAX$，做03~08处理。

03　计算函数值 $f(x_1)$，导函数值 $f'(x_1)$。

04　计算 $y = x_1 - \dfrac{f(x_1)}{f'(x_1)}$。

05　计算 $x_2 = y - \dfrac{f(y)}{f'(x_1)}$。

06　计算 $dx = |x_2 - x_1|$。

07　如果 $dx < TOL$，则退出循环。

08　$x_1 = x_2$。

09　输出 $x = x_2, f(x) = f(x_2), iter = i$。

10　结束。

2. 实验目的与要求

- 了解多重迭代的一般原理。
- 理解多重迭代收敛阶数。
- 能够编程实现原理部分给出的迭代法。

3. 实验内容与数据来源

用多重迭代法计算

$$f(x) = x^2 - 2x - 5 = 0$$

在区间[2,3]内的根，迭代初值取 $x_0 = 2$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值
ITER	INTEGER	实际迭代次数

5. 程序代码

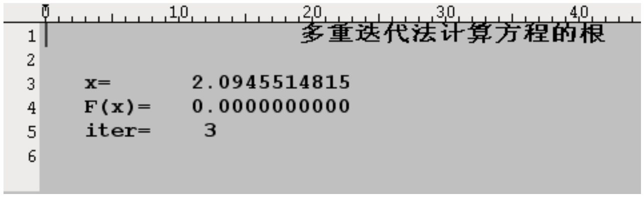
```
module mul_iter
!-----------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  多重迭代法迭代计算非线性方程
!
!-----------------------------------------------------
!  Parameters  :
!      1.     MAX 最大允许迭代次数
!      2.     tol 误差容许限
!      3.     迭代初值
!  Contains    :
!      1.    solve 迭代方法函数
!      2.    func  要计算的方程函数
!      3.    dfunc 导函数
!-----------------------------------------------------
implicit real*8(a-z)
integer:: MAX=200
real*8::tol=1D-7
real*8::x0=2d0
contains
subroutine solve(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  多重迭代法计算方程的根
!
```

```
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.   x 方程的根
!      2.   fx 该点的函数值
!      3.   iter 实际迭代次数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer i,iter
x1=x0
do i=1,MAX
    !计算函数一次
    f=func(x1)
    !计算导函数一次
    df=dfunc(x1)
    ! 二重迭代格式
    y1=x1-f/df
    x2=y1-func(y1)/df
    dx=dabs(x2-x1)
    !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
    write(102,*)i,x2,func(x2)
    !---------
    if (dx<tol) exit
    x1=x2
end do
    x=x2
    fx=func(x2)
    iter=i
end subroutine solve
function func(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    x 自变量
!      2.
!  Output parameters  :
!      1.   func 方程导函数值
!----------------------------------------------------
implicit real*8(a-z)
   func=x**3-2*x-5d0
end function func
```

```
function dfunc(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程导函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.    x 自变量
!      2.
!  Output parameters  :
!      1.   dfunc 方程函数值
!----------------------------------------------------
implicit real*8(a-z)
  dfunc=3*x**2-2d0
end function dfunc
end module  mul_iter
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   多重迭代法迭代计算方程的根主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.   result.txt 计算结果
!      2.   IM_result.txt 计算的中间结果
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use mul_iter
implicit real*8(a-z)
integer::iter
open(unit=101,file='result.txt')
open(unit=102,file='Im_result.txt')
call solve(x,fx,iter)
write(101,501)x,fx,iter
501 format(T20,'多重迭代法计算方程的根',//,&
          3x,'x=   ',F15.10,/,&
          3x,'F(x)=',F15.10,/,&
          3x,'iter=',I5)
end program main
```

6. 实验结论

程序运行后生成两个数据文件，result.txt 保存了计算结果，如图 5-9 所示。



```
多重迭代法计算方程的根

x=       2.0945514815
F(x)=    0.0000000000
iter=     3
```

图 5-9　多重迭代计算结果

文件 Im_result.txt 记录了中间数据，如表 5-9 所示。

表 5-9　多重迭代计算的中间结果

迭代序列	x	$f(x)$
1	2.09390000000000	-7.268803980999827E-003
2	2.09455148136683	-1.958843753868678E-009
3	2.09455148136683	-8.881784197001252E-016

可以看到实际上进行了 3 次迭代就达到预定的精度需求。

5.8　4阶收敛多重迭代法

1. 实验基本原理

利用牛顿迭代法还可以构造出另一个具有更高阶的多重迭代法

$$\begin{cases} y_k = x_k - \dfrac{f(x_k)}{f'(x_k)} \\ x_{k+1} = y_k - \dfrac{f(y_k)(y_k - x_k)}{2f(y_k) - f(x_k)} \end{cases} \quad k = 0,1,\cdots$$

它的迭代函数为

$$g(x) = \varphi(x) - \frac{f(\varphi(x))(\varphi(x) - x)}{2f(\varphi(x)) - f(x)}$$

其中

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

这个迭代格式与上一节介绍的迭代格式的计算量相当， 但是收敛阶为 4。

这里给出算法如下：

输入：最大允许迭代次数 $IMAX$，误差容限 TOL，迭代初值 x_0。

输出：方程的根 x，此时的函数值 $f(x)$，实际迭代次数 $iter$。

处理：01 $x_1 = x_0$。

02 对 $k=1,\cdots,IMAX$，做03~08处理。

03 计算 $f(x_1)$，即其导数值 $f'(x_1)$。

04 计算 $y = x_1 - \dfrac{f(x_1)}{f'(x_1)}$。

05 计算 $f(y)$。

06 计算 $x_2 = y - \dfrac{f(y)(y-x_1)}{2f(y)-f(x_1)}$

07 计算 $dx = |x_2 - x_1|$。

08 如果 $dx < TOL$，则退出循环。

09 输出 $x = x_2, f(x) = f(x_2), iter = i$。

10 结束。

2. 实验目的与要求

- 理解迭代格式的构造。
- 熟悉上述格式的计算流程，并知道其收敛阶数。
- 能够编程实现上述算法。

3. 实验内容与数据来源

采用 4 阶收敛的多重迭代法计算非线性方程

$$\sin(x) - \frac{x}{25} = 0$$

在 $x = 2.0$ 附近的根，初值即取 $x_0 = 2.0$，要求精度控制在10^{-7}之内。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
X	REAL*8	方程的根
FX	REAL*8	该点的函数值
ITER	INTEGER	实际迭代次数

5. 程序代码

```
module four_iter
!----------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  多重迭代法迭代计算非线性方程
!
!-----------------------------------------------------
!  Parameters  :
!      1.     MAX 最大允许迭代次数
!      2.     tol 误差容许限
!      3.     迭代初值
!  Contains    :
!      1.    solve 迭代方法函数
!      2.    func  要计算的方程函数
!      3.    dfunc 导函数
!-----------------------------------------------------
implicit real*8(a-z)
integer:: MAX=200
real*8::tol=1D-7
real*8::x0=2d0
contains
subroutine solve(x,fx,iter)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  4 阶收敛多重迭代法计算方程的根
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.   x 方程的根
!       2.   fx 该点的函数值
!       3.   iter 实际迭代次数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer i,iter
x1=x0
do i=1,MAX
    !计算函数一次
```

```
        fx=func(x1)
        !计算导函数一次
        dfx=dfunc(x1)
        y=x1-fx/dfx
        fy=func(y)
        x2=y-fy*(y-x1)/(2*fy-fx)
        dx=dabs(x2-x1)
        !该段为输出中间结果，为了用于分析，实际计算时，可以注释掉
        write(102,*)i,x2,func(x2)
        !---------
        if (dx<tol) exit
        x1=x2
    end do
        x=x2
        fx=func(x2)
        iter=i
    end subroutine solve
    function func(x)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :
    !----------------------------------------------------
    !  Purpose   :  方程函数
    !
    !----------------------------------------------------
    !  Input  parameters  :
    !      1.    x 自变量
    !      2.
    !  Output parameters  :
    !      1.   func 方程导函数值
    !---------------------------------------------------
    implicit real*8(a-z)
      func=dsin(x)-x/25d0
    end function func
    function dfunc(x)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :
    !----------------------------------------------------
    !  Purpose   :  方程导函数
    !
    !----------------------------------------------------
    !  Input  parameters  :
    !      1.    x 自变量
    !      2.
    !  Output parameters  :
    !      1.   dfunc 方程函数值
    !---------------------------------------------------
    implicit real*8(a-z)
     dfunc=dcos(x)-1/25d0
    end function dfunc
    end module  four_iter
    program main
```

```
!-------------------------------------program comment
! Version   :  V1.0
! Coded by  :  syz
! Date      :
!-----------------------------------------------------
! Purpose   :  4 阶收敛多重迭代法迭代计算方程的根主函数
!
!-----------------------------------------------------
! In put data  files :
!      1.
!      2.
! Output data files  :
!      1.   result.txt 计算结果
!      2.   IM result.txt 计算的中间结果
!-----------------------------------------------------
! Post Script :
!      1.
!      2.
!-----------------------------------------------------
use four iter
implicit real*8(a-z)
integer::iter
open(unit=101,file='result.txt')
open(unit=102,file='Im_result.txt')
call solve(x,fx,iter)
write(101,501)x,fx,iter
501 format(T20,'4 阶收敛迭代法计算方程的根',//,&
          3x,'x=   ',F12.8,/,&
          3x,'F(x)=',F12.8,/,&
          3x,'iter=',I5)
end program main
```

6. 实验结论

程序运行后产生两个数据文件，其中 result.txt 保存了计算结果，如图 5-10 所示。文件 Im_result.txt 记录了计算的中间结果，如表 5-10 所示。

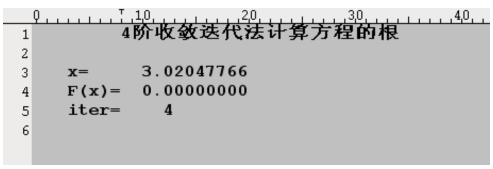


图 5-10　4 阶收敛速度的多重迭代法

表 5-10　4 阶收敛速度的多重迭代法计算的中间结果

迭代序列	x	$f(x)$
1	3.22486548427133	-0.212171242724158
2	3.02047562064798	2.107497333661557E-006
3	3.02047766146288	-5.551115123125783E-017
4	3.02047766146288	-5.551115123125783E-017

迭代的次数还与初值的选取有关系。从计算的中间结果可以看到，4 阶收敛的多重迭代法的收敛速度是非常快的。

5.9 开普勒方程的计算

1. 实验基本原理

在人造卫星轨道理论中对应的二体问题是可积系统，其中有 6 个积分常数为 $(a,e,i,\Omega,\omega,M)^T$，其中 a 表示轨道半长径，e 表示轨道偏心率，i 表示轨道倾角，Ω 是升交点赤经，ω 是近地点辐角，M 是平近点角。其中第六个积分常数，通常可以用其他的一些量替换，如过近地点时刻 t_p，真近点角度 f 与偏近点角度 E。这几个是相互等价的关系，我们这里要讲的就是平近点角 M 与偏近点角 E 的一个关系，有如下方程

$$E-e\sin E=M$$

这就是著名的 Kelper 方程，是在人造卫星运动理论或者行星运动理论中最基本的方程之一。因为如果我们已经知道卫星轨道量去计算卫星的星历时，第 6 个根数通常是使用平近点角。这时候需要把平近点角化为偏近点角，也就是需要计算上述的 Kepler 方程。具体理论这里不再阐述，而 Kepler 方程本身是我们这里感兴趣的。我们这里采用 Newton 迭代法计算 Kepler 方程，在实际工程中，一般也是这样做的，因为 Newton 迭代法收敛速度快而且精度比较高。

不动点迭代法比较简单，这里从略。牛顿法算法如下：

输入：最大允许迭代次数 MAX，误差容限 TOL，迭代区间初值，M 和 e。

输出：方程的根 E，实际迭代次数 $iter$。

处理：01 k=1。

02 令 $M_0=\dfrac{M\pi}{180}$，把输入量化为弧度，令 $E_0=M_0$。

03 当 k<=MAX 时，做 04~07 处理。

04 计算 $f=E_0-e\sin(E_0)-M_0$

05 计算 $df=1-e\cos(E_0)$。

06 计算改正值 $dE=-\dfrac{f}{df}$

07 $E_1=E_0+dE$

08 如果 $dE<TOL$，则停止循环。

09 $E_0=E_1$。

10 输出方程的根 $E=E_1$，实际迭代次数 $iter$。

11 结束。

2. 实验目的与要求

- 了解 Kepler 方程。
- 能够用牛顿迭代方法计算 Kepler 方程。
- 通过调节轨道偏心率 e 查看运算收敛情况。

3. 实验内容与数据来源

编写解 kepler 方程的方法，给定偏心率为 0.01、平近点角为 32 度时计算出偏近点角。Kepler 方程为

$$E - e\sin E = M$$

取偏心率 $e = 0.01$，分别取 $M = 10, 20, \cdots, 70, 80$，计算 E。

4. 函数调用接口说明

Newton

输出参变量	数据类型	变量说明
E	REAL*8	偏近点角
I	INTEGER	实际迭代次数
输入参变量	**数据类型**	**变量说明**
M	REAL*8	平近点角
EC	REAL*8	轨道偏心率

picard

输出参变量	数据类型	变量说明
E	REAL*8	偏近点角
I	INTEGER	实际迭代次数
输入参变量	**数据类型**	**变量说明**
M	REAL*8	平近点角
EC	REAL*8	轨道偏心率

5. 程序代码

```
module kepler
!----------------------------------------module coment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      : 2010-4-4
!-----------------------------------------------------
!  Purpose   :  计算 kepler 方程
!
!-----------------------------------------------------
```

```
!  Parameters  :
!      1. MAX 最大允许迭代次数
!      2. tol 误差容限
!----------------------------------------------------
!  Contains    :
!      1.  newton 牛顿法计算
!      2.  picard 不动点迭代法计算
!----------------------------------------------------
implicit real*8(a-z)
integer::MAX=200
real*8::eps=1d-12
contains
subroutine newton(E,M,ec,i)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-4
!----------------------------------------------------
!  Purpose   :  采用牛顿迭代法计算 kepler 方程
!
!----------------------------------------------------
!  Input  parameters  :
!       1. M 平近点角
!       2. ec 轨道偏心率
!  Output parameters  :
!       1. E 偏近点角
!       2. i 迭代次数
!
!---------------------------------------------------
!  Post Script :
!       1.计算时，需要注意角度与弧度之间的换算
!       2.本来牛顿法需要提供方程函数与导函数，这里就直接在计算中给出计算格式
!---------------------------------------------------
implicit real*8(a-z)
integer i
parameter(pi=3.141592653589793d0)
M0=M
M0=M0*pi/180
E0=M0
!最大允许迭代次
do i=1,MAX
!方程函数
  f=E0-ec*dsin(E0)-M0
!导函数
  df=1d0-ec*dcos(E0)
  dE=-f/df
  E1=E0+de
  E0=E1
   if (dE<eps) exit
end do
   E=E1*180/pi
end subroutine newton
subroutine picard(E,M,ec,i)
!---------------------------------subroutine  comment
!  Version   :  V1.0
```

```
!  Coded by  :  syz
!  Date      :  2010-4-4
!-----------------------------------------------------
!  Purpose   : 采用不动点迭代法计算 kepler 方程
!
!-----------------------------------------------------
!  Input  parameters  :
!       1. M 平近点角
!       2. ec 轨道偏心率
!  Output parameters  :
!       1. E 偏近点角
!       2. i 迭代次数
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer i
parameter(pi=3.141592653589793d0)
M0=M*pi/180
!把角度化为弧度
E0=M0
!最大允许迭代次
do i=1,MAX
   E1=M0+ec*dsin(E0)
   tol=E1-E0
   E0=E1
   if (tol<eps) exit
end do
   E=E1*180/pi
!弧度还原为角度
end subroutine picard
end module kepler
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   计算 kepler 方程
!
!-----------------------------------------------------
!  Output data files  :
!       1.  kepler.txt  计算结果
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.    非线性方程的参数可以由外部读入
!       2.
!-----------------------------------------------------
use kepler
implicit real*8(a-z)
integer j,k
open(unit=101,file='kepler.txt')
!以文件保存结果
write(101,501)
```

```
    501 format(30x,'牛顿迭代法计算结果',/,&
             10x,'M',T35,'E',T59,'iter')
    !iter 为迭代次数
    !此句为说明各量意义
    do k=1,8
     M=k*10d0
     ec=0.01D0
     call newton(E,M,ec,j)
    !调用牛顿迭代法迭代法函数
     write(101,*)M,E,j
    end do
    write(101,502)
    502 format(///,30x,'不动点迭代计算结果',/,&
              10x,'M',T35,'E',T59,'iter')
    !iter 为迭代次数
    !此句为说明各量意义
    do k=1,8
    M=k*10d0
    ec=0.01D0
    call picard(E,M,ec,j)
    !调用牛顿迭代法迭代法函数
    write(101,*)M,E,j
    end do
    end program main
```

6. 实验结论

程序运行后，生成数据文件 kepler.txt。计算结果如图 5-11 所示。

牛顿迭代法计算结果

M	E	iter
10.0000000000000	10.1004824827757	2
20.0000000000000	20.1978208433850	2
30.0000000000000	30.2889778649518	2
40.0000000000000	40.3711254112930	2
50.0000000000000	50.4417374963576	2
60.0000000000000	60.4986705337862	2
70.0000000000000	70.5402279408279	2
80.0000000000000	80.5652072806913	2

不动点迭代计算结果

M	E	iter
10.0000000000000	10.1004824827756	6
20.0000000000000	20.1978208433848	6
30.0000000000000	30.2889778649516	6
40.0000000000000	40.3711254112929	6
50.0000000000000	50.4417374963576	6
60.0000000000000	60.4986705337862	6
70.0000000000000	70.5402279408279	6
80.0000000000000	80.5652072806912	5

图 5-11　kepler 方程解算结果

可以看到牛顿迭代法明显要比不动点迭代法快。

作者先前曾编写过计算方法书籍，在答疑中经常有初学者对以下的问题不知道如何处理。即一般书上总是介绍形如

$$x^2+3x=5$$
$$\vdots$$
$$\cos x+e^x-7x=0$$

这些类型的方程，因为这些方程的系数是已知的，所以便于利用书上的方法。而有时候读者要编写的方程形如

$$ax^3+bx=c$$
$$\vdots$$
$$\cos x+ae^x-by=c$$

这样的方程，其中系数 a,b,c 是由外部输入的。很多读者对此不知道如何处理，其实方法很即在编写函数时把系数作为参数，由外部传入。本实验即属于这样的情况，M,e 两个变量都是由外部输入的。实际上略微有些编程经验的人，这不是一个真正的问题，不过很多初学者经常提到，这里做简要的说明。

本章小结

非线性方程的数值方法因为内容并不是很多，而且方法相对其他内容也比较简单一些，所以在很多数值分析教材中往往被安排在导言之后的第一章或者教材的最后一章。

本书之所以把非线性方程放在在线性方程组之后，主要是基于以下的考虑。如果一开始就讲非线性方程，那么后续的一章非线性方程组的计算方法则不太好衔接上，因为在非线形方程组一章需要用到非线性方程计算方法与线性方程组的方法。而本书在线性方程组与非线性方程的解法具备之后紧接着介绍非线性方程组。如此安排，使读者有比较好的基础知识准备，而且衔接比较自然、紧凑。

就数值方法本身而言，目前的方法对于非线形方程的计算总体效果还是不错的，基本上能满足一般的工程需求，而且在程序设计时一般比较容易实现。当然针对一些特殊的问题，可能有时候还需要构造一些特别的方法。

第 6 章

◀非线性方程组的数值方法▶

上一章介绍了非线性方程的计算方法，在实际应用中，往往更多情况是非线性方程组。可以认为非线性方程是非线性方程组的一种简单特例，非线性方程的一些方法可以推广到非线性方程组的处理方法中。不过非线性方程组实际上要比非线性方程复杂许多，很多理论依然没有完善。本章介绍了多种重要的非线性方程组的计算方法，这些方法针对适合定方程。对于超定的非线性方程组，需要本章与最小二乘法一章的知识准备，我们留在应用范例一章的非线性参数估计中介绍。阅读本章需要的准备知识是线性方程组的计算方法与非线性方程的计算方法。

6.1 牛顿迭代法

1. 实验基本原理

牛顿迭代法是非线性方程组中最基础也是最重要的方法。虽然在本章后面会介绍各种“更先进”的拟牛顿系列算法，但是实际上工程使用中一般用的最多的还是牛顿迭代法。牛顿迭代法编程简单，收敛速度快。实际上，上一章介绍的非线性方程的牛顿迭代法是本节介绍的方程组牛顿迭代法的特例。任意的非线性方程租可以化为如下形式

$$\begin{cases} f_1(x_1,x_2,\cdots,x_n)=0 \\ f_2(x_1,x_2,\cdots,x_n)=0 \\ \qquad\vdots \\ f_n(x_1,x_2,\cdots,x_n)=0 \end{cases}$$

欲求解 $x_1,x_2,\cdots,x_n$，在 $\mathbf{x}^{(k)}$ 处按照多元函数的泰勒展开，并取线性项得到

$$\begin{pmatrix} f_1^{(k)}(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)}) \\ f_2^{(k)}(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)}) \\ \vdots \\ f_n^{(k)}(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)}) \end{pmatrix} + \mathbf{f}'(\mathbf{x}^{(k)}) \begin{pmatrix} x_1^{(k+1)} - x_1^{(k)} \\ x_2^{(k+1)} - x_2^{(k)} \\ \vdots \\ x_n^{(k+1)} - x_n^{(k)} \end{pmatrix} = \mathbf{0}$$

其中

$$\mathbf{f}'(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{pmatrix}$$

这样便得到迭代公式：

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix} = \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} - \left[\mathbf{f}'(\mathbf{x}^{(k)})\right]^{-1} \begin{pmatrix} f_1^{(k)}(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)}) \\ f_2^{(k)}(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)}) \\ \vdots \\ f_n^{(k)}(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)}) \end{pmatrix}$$

以上就是牛顿迭代法的计算公式分量形式。如果以向量形式表示则更为简洁。可以看到，在使用牛顿法时每一步迭代需要计算两个函数，一是方程函数，另一个是偏导数矩阵函。一般做程序设计时，会分别把这两个数学意义上的函数用程序函数编写出来，为每次调用提供准备。偏导数矩阵一般是需要手工推导的。

2. 实验目的与要求

- 了解多应变量、多自变量泰勒线性展开的矩阵表示方法。
- 能够自行推导牛顿迭代格式。
- 能够编程实现牛顿迭代法。

3. 实验内容与数据来源

采用牛顿迭代法计算非线性方程组

$$\begin{cases} 6x^3 + xy - 3y^2 - 4 = 0 \\ x^2 - 18xy^2 + 16y^3 + 1 = 0 \end{cases}$$

迭代初值取 $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$，要求计算精度达到 10^{-8}。

5. 程序代码

```
module m_gauss
!---------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-----------------------------------------------------
!  Description : 高斯列主元消去法模块
!               此模块主要为牛顿法中解线性化方程所调用
!-----------------------------------------------------
!  Contains    :
!      1.   lineq 解线性方程组
!      2.
!-----------------------------------------------------
contains
subroutine lineq(A,b,x,N)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!                Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   b(N)右向量
!       3.   N 方程维数
!  Output parameters  :
!       1.  x  方程的根
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵[Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!###########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
   elmax=dabs(Ab(k,k))
   id_max=k
   !这段为查找主元素
   !这段程序的主要目的的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
    do i=k+1,n
     if (dabs(Ab(i,k))>elmax) then
```

```
        elmax=Ab(i,k)
        id_max=i
      end if
    end do
!至此，已经完成查找最大元素，查找完成以后与第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id_max,:)
    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!#########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine lineq
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module m_gauss
```

```
module  m newton
!---------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   要计算的方程相关模块
!
!-----------------------------------------------------
!  Contains    :
!      1.    函数文件
!      2.    偏导数文件
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
contains
subroutine  solve()
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算非线性方程组
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.    report.txt 计算结果迭代序列
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.    需要用到线性方程组的解法，这里用选主元消去法
!       2.    需要准备函数文件与偏导数文件
!-----------------------------------------------------
use m_gauss
!IVF 中，在 USE 模块时，需要模块在被调用之前
!当然在调用之前 include 包含模块的 fortran 文件也是可以的
implicit real*8(a-z)
integer::I,itmax=50
integer::N=2
real*8::x(2),f(2),dx(2)
real*8::df(2,2)
!itmax 最大允许迭代次数
!N 方程组维数
!df 偏导数矩阵
open(unit=11,file='result.txt')
write(11,101)
101 format(/,T6,'牛顿法计算非线性方程组迭代序列',/)
x=(/2d0,2d0/)
tol=1d-8
do i=1,itmax
  call  func(f,x)
```

```
    call  jac(df,x)
    call  lineq(df,-f,dx,N)
    x=x+dx
    write(11,102)i,x
  102 format(I5,2F16.10)
  !判断计算精度，当满足误差容限时退出循环。
     dx2=dsqrt(dx(1)**2+dx(2)**2)
      if (dx2<tol) exit
  !------
  end do
  end subroutine solve
  subroutine func(f,x)
  !---------------------------------subroutine  comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      :
  !-----------------------------------------------------
  !  Purpose   :  方程函数
  !
  !-----------------------------------------------------
  !  Input  parameters  :
  !       1.    x 自变量
  !       2.
  !  Output parameters  :
  !       1.    f 方程函数
  !       2.
  !  Common parameters  :
  !
  !----------------------------------------------------
  implicit real*8(a-z)
  real*8::x(2),f(2)
  f(1)=6*x(1)**3+x(1)*x(2)-3*x(2)**3-4
  f(2)=x(1)**2-18*x(1)*x(2)**2+16*x(2)**3+1
  end subroutine func
  subroutine  jac(df,x)
  !---------------------------------subroutine  comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      :
  !-----------------------------------------------------
  !  Purpose   :  偏导数矩阵
  !
  !-----------------------------------------------------
  !  Input  parameters  :
  !       1.
  !       2.
  !  Output parameters  :
  !       1.
  !       2.
  !  Common parameters  :
  !
  !----------------------------------------------------
  !  Post Script :
  !       1.
  !       2.
  !----------------------------------------------------
```

```
    implicit real*8(a-z)
    real*8::x(2),df(2,2)
    df(1,1)=18*x(1)**2+x(2)
    df(2,1)=2*x(1)-18*x(2)**2
    df(1,2)=x(1)-9*x(2)**2
    df(2,2)=-36*x(1)*x(2)+48*x(2)**2
    end subroutine jac
    end module m_newton
    program main
    !-------------------------------------program comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :  2010.08.04
    !----------------------------------------------------
    !  Purpose     :  牛顿法解非线性方程组的主函数
    !
    !  Post Script :
    !      1.
    !      2.
    !
    !----------------------------------------------------
    !  In put data  files :
    !      1.
    !      2.
    !  Output data files  :
    !      1.
    !      2.
    !
    !----------------------------------------------------
    use m_newton
    !调用牛顿法中的函数
    call solve()
    end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，打开该文件可以看到计算结果如图 6-1 所示。

牛顿法计算非线性方程组迭代序列

1	1.3725806452	1.3403225806
2	1.0783868120	1.0538012326
3	1.0053496890	1.0026926187
4	1.0000336787	1.0000224377
5	1.0000000011	1.0000000006
6	1.0000000000	1.0000000000

图 6-1　牛顿迭代法计算非线性方程组

可以看到计算牛顿迭代法收敛速度还是比较快的。在 solve 函数中，我们直接把计算结果用文件保存起来。

6.2　简化牛顿法

1. 实验基本原理

牛顿迭代法在每次迭代过程中都需要计算雅可比矩阵的逆（实际计算时，我们是直接计算线性方程组的），为了减少牛顿法的计算量，有时候可以使用如下迭代格式

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} - \left[\mathbf{F}'\left(\mathbf{x}^{0}\right)\right]^{-1}\mathbf{F}\left(\mathbf{x}^{k}\right) \quad (k=0,1,\ldots)$$

这种方法在除了在开始计算$\left[\mathbf{F}'\left(\mathbf{x}^{0}\right)\right]^{-1}$，以后每步迭代只计算一个函数值，大大减少了计算量，但这个算法缺点是只有线性收敛，收敛速度较慢。

2. 实验目的与要求

- 复习牛顿迭代法原理。
- 知道简化牛顿法的优缺点。
- 能够编程实现简化牛顿法计算非线性方程组。

3. 实验内容与数据来源

采用简化牛顿迭代法计算

$$\begin{cases} 6x^3 + xy - 3y^2 - 4 = 0 \\ x^2 - 18xy^2 + 16y^3 + 1 = 0 \end{cases}$$

迭代初值取$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$，要求计算精度达到$10^{-8}$。

5. 程序代码

```
module inv mat
!--------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :  2010-4-8
!-----------------------------------------------------
!  Description : 计算逆矩阵
!
!-----------------------------------------------------
!  Contains    :
!      1.   inv 计算逆矩阵
!-----------------------------------------------------
```

```
contains
subroutine inv(A,invA,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算逆矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i
real*8::A(n,n),invA(n,n),E(n,n)
E=0
!设置 E 为单位矩阵
do i=1,n
   E(i,i)=1
end do
call mateq(A,E,invA,N,N)
end subroutine inv
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程
!                AX=B
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   B(N,M)右矩阵
!      3.   N 方程维数,
!      4.   M---B 的列数
!  Output parameters  :
!      1.  X  方程的根（N,M）维
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N,M,i
```

```
real*8::A(N,N),B(N,M),X(N,M)
real*8::btemp(N),xtemp(N)
do i=1,M
   btemp=B(:,i)
   call elgauss(A,btemp,xtemp,N)
   X(:,i)=xtemp
end do
end subroutine mateq
subroutine elgauss(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   b(N)右向量
!       3.   N 方程维数
!  Output parameters  :
!       1.  x  方程的根
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!###########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
   elmax=dabs(Ab(k,k))
   id_max=k
   !这段为查找主元素
   !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
    do i=k+1,n
     if (dabs(Ab(i,k))>elmax) then
        elmax=Ab(i,k)
        id_max=i
     end if
   end do
!至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
   vtemp1=Ab(k,:)
   vtemp2=Ab(id_max,:)
   Ab(k,:)=vtemp2
   Ab(id_max,:)=vtemp1
```

```
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!###################################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为上三角矩阵
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine elgauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module inv_mat
module  simple newton
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   要计算的方程相关模块
!
!-----------------------------------------------------
!  Contains     :
!      1.    函数文件
!      2.    偏导数文件
!      3.    solve 函数
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
```

```
!----------------------------------------------------
contains
subroutine  solve()
!------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!----------------------------------------------------
!  Purpose   :  简化牛顿法计算非线性方程组
!
!----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.    report.txt 计算结果迭代序列
!      2.
!----------------------------------------------------
!  Post Script :
!      1.    需要用到线性方程组的解法，这里用选主元消去法
!      2.    需要准备函数文件与偏导数文件
!----------------------------------------------------
use inv mat
implicit real*8(a-z)
integer::I,itmax=100
integer::N=2
real*8::x(2),f(2),dx(2)
real*8::df(2,2),invdf(2,2)
!itmax 最大允许迭代次数
!N 方程组维数
!df 偏导数矩阵
!invdf 偏导数矩阵的逆矩阵
open(unit=11,file='result.txt')
write(11,101)
101 format(/,T6,'简化牛顿法计算非线性方程组迭代序列',/)
x=(/2d0,2d0/)
tol=1d-8
!计算初值的偏导数矩阵
call jac(df,x)
!计算偏导数的逆矩阵
!仅进行一次矩阵求逆，不进入循环
call  inv(df,invdf,N)
do i=1,itmax
  call  func(f,x)
   dx=-matmul(invdf,f)
  x=x+dx
  write(11,102)i,x
102 format(I5,2F16.10)
!判断计算精度，当满足误差容限时退出循环。
    dx2=dsqrt(dx(1)**2+dx(2)**2)
     if  (dx2<tol)  exit
!------
end do
end subroutine solve
subroutine func(f,x)
!---------------------------------subroutine  comment
```

```
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.    f 方程函数
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
real*8::x(2),f(2)
f(1)=6*x(1)**3+x(1)*x(2)-3*x(2)**3-4
f(2)=x(1)**2-18*x(1)*x(2)**2+16*x(2)**3+1
end subroutine func
subroutine  jac(df,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  偏导数矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
real*8::x(2),df(2,2)
df(1,1)=18*x(1)**2+x(2)
df(2,1)=2*x(1)-18*x(2)**2
df(1,2)=x(1)-9*x(2)**2
df(2,2)=-36*x(1)*x(2)+48*x(2)**2
end subroutine jac
end module simple newton
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.08.04
```

```
!-------------------------------------------------------
!  Purpose     :    简化牛顿法主函数
!
!  Post Script :
!       1.
!       2.
!
!-------------------------------------------------------
use simple newton
!调用方法函数
call solve
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，如表 6-1 所示。

表 6-1 简化牛顿法迭代序列

迭代序列	x^i	y^i
1	1.3725806452	1.3403225806
2	1.2167942743	1.1816005356
3	1.1358334158	1.1023919764
4	1.0881712805	1.0582208605
5	1.0584331991	1.0325429682
6	1.0392634917	1.0174327469
7	1.0266524170	1.0085907081
8	1.0182404829	1.0035215879
9	1.0125717957	1.0007242760
10	1.0087202625	0.9992806777
⋮	⋮	⋮
46	1.0000001458	0.9999996203
47	1.0000001099	0.9999997130
48	1.0000000828	0.9999997830
49	1.0000000625	0.9999998360
50	1.0000000471	0.9999998761
51	1.0000000355	0.9999999063
52	1.0000000268	0.9999999292
53	1.0000000202	0.9999999465
54	1.0000000152	0.9999999596
55	1.0000000115	0.9999999695
56	1.0000000087	0.9999999769

从表 6-1 中可以看到对于相同的初值和精度要求，简化牛顿法进行了 56 次迭代才满足，虽然计算中省去了多次计算逆矩阵，不过代价是付出了较多的迭代次数。

6.3 拟牛顿之Broyden方法

1. 实验基本原理

牛顿迭代法每迭代一次，计算当前一步的 Jacobi 矩形阵的逆矩阵，计算量还是比较大，而简化牛顿法不失为一种方法，但有时候计算效果不能令我们满意。为了不每次迭代都计算逆矩阵，我们设法构造 H_k 逼近 $\mathbf{f}'(\mathbf{x}_k)$ 的逆矩阵。这样迭代公式为

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k\mathbf{f}(\mathbf{x}_k)$$

选取不同的 $\mathbf{H}_k$ 就得到各种类型的拟牛顿方法。这里主要介绍 Broyden 方法，Broyden 方法的基本迭代格式为：

$$\begin{cases}\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k\mathbf{f}(\mathbf{x}_k) \\ \mathbf{H}_{k+1} = \mathbf{H}_k + \dfrac{(\Delta\mathbf{x}_k - \mathbf{H}_k\mathbf{y}_k)(\Delta\mathbf{x}_k)^T\mathbf{H}_k}{(\Delta\mathbf{x}_k)^T\mathbf{H}_k\mathbf{y}_k}\end{cases}$$

其中

$$\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \mathbf{y}_k = \mathbf{f}(\mathbf{x}_{k+1}) - \mathbf{f}(\mathbf{x}_k)$$

拟牛顿法是 20 世纪 60 年代以来发展起来的算法，相对而言是比较新的一种方法，它克服了牛顿法需要求导数和求逆的缺点，是目前实际使用较为有效的一种方法。

2. 实验目的与要求

- 了解拟牛顿法的大致思想。
- 能够理解 Broyden 迭代格式中各变量的含义。
- 会使用作者编写的函数进行非线性方程计算。
- 最好能自行编写 Broyden 方法。

3. 实验内容与数据来源

采用拟牛顿之 Broyden 方法计算非线性方程组

$$\begin{cases}3x - \cos(yz) - \dfrac{1}{2} = 0 \\ x^2 - 81(y + 0.2) + \sin z + 1.06 = 0 \\ \exp(-xy) + 20z + \dfrac{10\pi - 3}{3} = 0\end{cases}$$

初值取 $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.1 \\ -0.1 \end{pmatrix}$，要求精度达到 10^{-8}。

4. 函数调用接口说明

输入参变量	数据类型	变量说明
X0	REAL*8(N)	初值
N	INTEGER	方程的维数

5. 程序代码

```
module inv_mat
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-----------------------------------------------------
!  Description : 计算逆矩阵
!
!-----------------------------------------------------
!  Contains    :
!      1.   inv 计算逆矩阵
!-----------------------------------------------------
contains
subroutine inv(A,invA,N)
!---------------------------------subroutine  comment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date       :
!-----------------------------------------------------
!  Purpose    :  计算逆矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i
```

```
real*8::A(n,n),invA(n,n),E(n,n)
E=0
!设置 E 为单位矩阵
do i=1,n
  E(i,i)=1
end do
call mateq(A,E,invA,N,N)
end subroutine inv
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程
!               AX=B
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   B(N,M)右矩阵
!      3.   N 方程维数,
!      4.   M---B 的列数
!  Output parameters  :
!      1.  X  方程的根（N,M）维
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N,M,i
real*8::A(N,N),B(N,M),X(N,M)
real*8::btemp(N),xtemp(N)
do i=1,M
   btemp=B(:,i)
   call elgauss(A,btemp,xtemp,N)
   X(:,i)=xtemp
end do
end subroutine mateq
subroutine elgauss(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
```

```
implicit real*8(a-z)
integer::i,k,N
integer::id max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!###########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
       if (dabs(Ab(i,k))>elmax) then
          elmax=Ab(i,k)
          id max=i
       end if
    end do
!至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id max,:)
    Ab(k,:)=vtemp2
    Ab(id max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!##########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!           | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!           | 0  0  *  *  # |
!           | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine elgauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!                Ax=b
```

```
!-----------------------------------------------------
! Input parameters :
!      1.  A(N,N)系数矩阵
!      2.  b(N)右向量
!      3.  N方程维数
! Output parameters :
!      1. x 方程的根
!      2.
! Common parameters :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module inv mat
module broyden1
!----------------------------------------module coment
! Version    : V1.0
! Coded by   : syz
! Date       :
!-----------------------------------------------------
! Description :  要计算的方程相关模块
!
!-----------------------------------------------------
! Contains   :
!     1.   函数文件
!     2.   偏导数文件
!-----------------------------------------------------
! Post Script :
!     1.
!     2.
!-----------------------------------------------------
use inv mat
contains
subroutine solve(x0,N)
!--------------------------------subroutine comment
! Version  : V1.0
! Coded by : syz
! Date     :
!-----------------------------------------------------
! Purpose  :  Broyden 第一公式方法函数
!
!-----------------------------------------------------
! Input parameters :
!      1.
!      2.
! Output parameters :
```

```
!       1.
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::i,n,itmax=50
real*8::x1(n),x0(n),y(n),f0(n),f1(n),dx(n)
real*8::v1(n),v2(n),v3(n)
real*8::H0(n,n),H1(n,n),df(n,n)
!itmax 最大允许迭代次数
tol=1d-8
!误差容限
call jac(df,x0)
!注意：设置 H0 初值为 偏导数矩阵的逆矩阵
!  也可以直接设置 H0 为单位矩阵，但是那样收敛较慢
!  设置 H0 初值为偏导数逆矩阵，但是在迭代过程中则不计算逆矩阵
call inv(df,H0,N)
write(11,101)
write(12,102)
do i=1,itmax
    !计算函数值
    call func(f0,x0)
    !更新方程的根
    x1=x0-matmul(H0,f0)
    dx=x1-x0
    call func(f1,x1)
    y=f1-f0
    v1=matmul(H0,y)
     v2=dx-v1
    t1=vdot(v2,dx,N)
    t2=vdot(dx,v1,N)
    H1=H0+t1/t2*H0
    !把计算中的 H 矩阵写入文件
    write(12,103)H0
    x0=x1
    H0=H1
    !把计算迭代序列写入文件
    write(11,104)i,x0
    !判断计算精度，当满足误差容限时退出循环。
    dx2=dsqrt(dx(1)**2+dx(2)**2)
     if (dx2<tol) exit
!------
end do
101 format(/,T5,'Broyden 秩方法计算序列',/)
102 format(/,T5,'Broyden H 矩阵序列为：',/)
103 format(<N>(<N>F16.10/),/)
104 format(I8,3F16.10)
end subroutine solve
function vdot(a,b,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
```

```
!  Date      :
!-----------------------------------------------------
!  Purpose   : 计算向量 a,b 内积
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.   N 向量维数
!       2.
!----------------------------------------------------
!  Post Script :
!       1.   vdot=a1(1)*(1)+a(2)*b(2)+...
!       2.
!----------------------------------------------------
integer::i,N
real*8::a(n),b(n),vdot
vdot=0
do i=1,N
  vdot=vdot+a(i)*b(i)
end do
end function vdot
subroutine func(f,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.    f 方程函数
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
real*8::x(3),f(3),pi=3.141592653589793
f(1)=3*x(1)-dcos(x(2)*x(3))-1d0/2
f(2)=x(1)**2-81*(x(2)+0.1d0)**2+dsin(x(3))+1.06d0
f(3)=dexp(-x(1)*x(2))+20*x(3)+(10*pi-3d0)/3
end subroutine func
subroutine  jac(df,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  偏导数矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
```

```
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
real*8::x(3),df(3,3)
df(1,1)=3d0
df(1,2)=x(3)*dsin(x(2)*x(3))
df(1,3)=x(2)*dsin(x(2)*x(3))
df(2,1)=2*x(1)
df(2,2)=-162*(x(1)+0.1)
df(2,3)=dcos(x(3))
df(3,1)=-x(2)*dexp(-x(1)*x(2))
df(3,2)=-x(1)*dexp(-x(1)*x(2))
df(3,3)=20d0
end subroutine jac
end module broyden1
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  Broyden 方法计算非线性方程组主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.   result.txt 给出了计算结果
!      2.   Hmatrix.txt 给出了计算中的 H 矩阵值
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use  inv mat
use  broyden1
implicit real*8(a-z)
integer::N=3
real*8::x0(3)
open(unit=11,file='result.txt')
open(unit=12,file='Hmatrix.txt')
x0=(/0.1d0,0.1d0,-0.1d0/)
call solve(x0,n)
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，打开 result.txt 文件可以看到结果图 6-2 所示。

```

                      Broyden 秩1方法计算序列

        1    0.4998696729    0.0194668487    -0.5215204865
        2    0.4999863755    0.0087378390    -0.5231745890
        3    0.5000064375    0.0008476767    -0.5235707275
        4    0.4999943455    0.0000360183    -0.5235879349
        5    0.5000056351   -0.0000002421    -0.5236079247
        6    0.4999968377    0.0000002190    -0.5235937074
        7    0.5000000071    0.0000000108    -0.5235987840
        8    0.5000000000    0.0000000052    -0.5235987900
```

图 6-2　迭代序列计算结果

为了方便读者以后编程进行调试，这里给出了计算的中间结果，在文件 Hmatrix.txt 中给出了计算过程中迭代矩阵的变化，结果如下所示。

$$\mathbf{H}_1=\begin{pmatrix} 0.3333331840 & 0.0021086068 & 0.0016605204 \\ 0.0000102385 & -0.0308688253 & -0.0001527577 \\ 0.0000161570 & 0.0015358359 & 0.0500076828 \end{pmatrix}$$

$$\mathbf{H}_2=\begin{pmatrix} 0.3348909937 & 0.0021184613 & 0.0016682808 \\ 0.0000102864 & -0.0310130886 & -0.0001534716 \\ 0.0000162325 & 0.0015430135 & 0.0502413902 \end{pmatrix}$$

$$\mathbf{H}_3=\begin{pmatrix} 0.5772941813 & 0.0036518610 & 0.0028758277 \\ 0.0000177319 & -0.0534612036 & -0.0002645585 \\ 0.0000279821 & 0.0026598886 & 0.0866074716 \end{pmatrix}$$

$$\mathbf{H}_4=\begin{pmatrix} 0.6365913495 & 0.0040269644 & 0.0031712203 \\ 0.0000195533 & -0.0589525079 & -0.0002917328 \\ 0.0000308563 & 0.0029331009 & 0.0955034175 \end{pmatrix}$$

$$\mathbf{H}_5=\begin{pmatrix} 0.6652119891 & 0.0042080135 & 0.0033137958 \\ 0.0000204324 & -0.0616029656 & -0.0003048489 \\ 0.0000322435 & 0.0030649708 & 0.0997971751 \end{pmatrix}$$

$$\mathbf{H}_6=\begin{pmatrix} 0.5206610321 & 0.0032936097 & 0.0025937060 \\ 0.0000159924 & -0.0482166049 & -0.0002386050 \\ 0.0000252370 & 0.0023989508 & 0.0781111901 \end{pmatrix}$$

6.4　Broyden第二公式计算非线性方程组

1. 实验基本原理

Broyden 第二方法和 Broyden 第一方法一样，推导起来比较麻烦一些，这里就不再介绍推导过程，直接给出计算公式。有兴趣的读者可以参考《现代应用数学手册-计算与数值分析卷》（清华大学出版社 2005 年出版）。

$$\begin{cases} \mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k \mathbf{f}(\mathbf{x}_k) \\ \mathbf{H}_{k+1} = \mathbf{H}_k + \left(\Delta\mathbf{x}_k - \mathbf{H}_k\mathbf{y}_k\right)\dfrac{\left(\Delta\mathbf{x}_k - \mathbf{H}_k\mathbf{y}_k\right)^T}{\left(\Delta\mathbf{x}_k - \mathbf{H}_k\mathbf{y}_k\right)^T \mathbf{y}_k} \end{cases}$$

其中 $\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \mathbf{y}_k = \mathbf{f}(\mathbf{x}_{k+1}) - \mathbf{f}(\mathbf{x}_k)$。

2. 实验目的与要求

- 了解 Broyden 第二公式各变量的含义。
- 会使用作者编写的函数进行非线性方程组的计算。
- 最好能自行编写 Broyden 第二公式方法函数。

3. 实验内容与数据来源

采用 Broyden 第二公式计算非线性方程组

$$\begin{cases} 3x - \cos(yz) - \dfrac{1}{2} = 0 \\ x^2 - 81(y+0.1)^2 + \sin z + 1.06 = 0 \\ e^{-xy} + 20z + \dfrac{10\pi - 3}{3} = 0 \end{cases}$$

初值取 $\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \\ -0.1 \end{bmatrix}$，要求精度达到 10^{-8}。

4. 函数调用接口说明

输入参变量	数据类型	变量说明
X0	REAL*8(N)	初值
N	INTEGER	方程的维数

TOL	REAL*8	误差容限

5. 程序代码

```
module inv_mat
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-----------------------------------------------------
!  Description : 计算逆矩阵
!
!-----------------------------------------------------
!  Contains    :
!      1.   inv 计算逆矩阵
!-----------------------------------------------------
contains
subroutine inv(A,invA,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算逆矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i
real*8::A(n,n),invA(n,n),E(n,n)
E=0
!设置 E 为单位矩阵
do i=1,n
   E(i,i)=1
end do
call mateq(A,E,invA,N,N)
end subroutine inv
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
```

```
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程
!               AX=B
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   B(N,M)右矩阵
!      3.   N方程维数,
!      4.   M---B的列数
!  Output parameters  :
!      1.  X  方程的根(N,M)维
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N,M,i
real*8::A(N,N),B(N,M),X(N,M)
real*8::btemp(N),xtemp(N)
do i=1,M
   btemp=B(:,i)
   call elgauss(A,btemp,xtemp,N)
   X(:,i)=xtemp
end do
end subroutine mateq
subroutine elgauss(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
```

```
Ab(:,N+1)=b
!################################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id_max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
      if (dabs(Ab(i,k))>elmax) then
         elmax=Ab(i,k)
         id_max=i
      end if
    end do
!至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id_max,:)
    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!###############################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!              | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!            | 0  0  *  *  # |
!            | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine elgauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   b(N)右向量
!       3.   N 方程维数
!  Output parameters  :
!       1.  x  方程的根
```

```
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
   x(i)=b(i)
  do j=i+1,N
   x(i)=x(i)-a(i,j)*x(j)
  end do
   x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module inv_mat
module  broyden2
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   要计算的方程相关模块
!
!-----------------------------------------------------
!  Contains    :
!      1.    函数文件
!      2.    偏导数文件
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use inv_mat
contains
subroutine solve(x0,N,tol)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  Broyden 第二公式方法函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
```

```
! Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::i,n,itmax=50
real*8::x1(n),x0(n),y(n),f0(n),f1(n),dx(n)
real*8::v1(n),v2(n),v3(n)
real*8::H0(n,n),H1(n,n),df(n,n),dH(N,N)
!itmax 最大允许迭代次数
!tol 误差容限
call jac(df,x0)
!注意：设置 H0 初值为 偏导数矩阵的逆矩阵
!  也可以直接设置 H0 为单位矩阵，但是那样收敛较慢
!  设置 H0 初值为偏导数逆矩阵，但是在迭代过程中则不计算逆矩阵
call inv(df,H0,N)
write(11,101)
write(12,102)
do i=1,itmax
    !计算函数值
    call func(f0,x0)
    !更新方程的根
    x1=x0-matmul(H0,f0)
    dx=x1-x0
    call func(f1,x1)
    y=f1-f0
    v1=dx-matmul(H0,y)
    t1=vdot(v1,y,N)
    dH=vvmat(v1,v1,N)
    dH=dH/t1
    H1=H0+dH
    !把计算中的 H 矩阵写入文件
    write(12,103)H0
    x0=x1
    H0=H1
    !把计算迭代序列写入文件
    write(11,104)i,x0
    !判断计算精度，当满足误差容限时退出循环。
    dx2=dsqrt(dx(1)**2+dx(2)**2)
     if (dx2<tol) exit
!------
end do
101 format(/,T5,'Broyden 第二公式计算序列',/)
102 format(/,T5,'Broyden 第二公式 H 矩阵序列为：',/)
103 format(<N>(<N>F16.10/),/)
104 format(I8,3F16.10)
end subroutine solve
function vdot(a,b,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   : 计算向量 a,b 内积
!
```

```
!----------------------------------------------------
!  Input  parameters  :
!      1.   N 向量维数
!      2.
!---------------------------------------------------
!  Post Script :
!      1.   vdot=a1(1)*(1)+a(2)*b(2)+...
!      2.
!---------------------------------------------------
integer::i,N
real*8::a(n),b(n),vdot
vdot=0
do i=1,N
  vdot=vdot+a(i)*b(i)
end do
end function vdot
function vvmat(a,b,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!----------------------------------------------------
!  Purpose   :  N 维列向量 a 乘以 N 维横向量 b
!               结果为  n*n 维矩阵
!---------------------------------------------------
!  Post Script :
!      1.
!      2.
!---------------------------------------------------
integer::n,i,j
real*8::a(n),b(n),vvmat(n,n)
do i=1,n
  do j=1,n
    vvmat(i,j)=a(i)*b(j)
  end do
end do
end function vvmat
subroutine func(f,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!----------------------------------------------------
!  Purpose   :  方程函数
!
!----------------------------------------------------
!  Input  parameters  :
!      1.    x 自变量
!      2.
!  Output parameters  :
!      1.    f 方程函数
!      2.
!  Common parameters  :
!
!---------------------------------------------------
```

```
implicit real*8(a-z)
real*8::x(3),f(3),pi=3.141592653589793
f(1)=3*x(1)-dcos(x(2)*x(3))-1d0/2
f(2)=x(1)**2-81*(x(2)+0.1d0)**2+dsin(x(3))+1.06d0
f(3)=dexp(-x(1)*x(2))+20*x(3)+(10*pi-3d0)/3
end subroutine func
subroutine  jac(df,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  偏导数矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
real*8::x(3),df(3,3)
df(1,1)=3d0
df(1,2)=x(3)*dsin(x(2)*x(3))
df(1,3)=x(2)*dsin(x(2)*x(3))
df(2,1)=2*x(1)
df(2,2)=-162*(x(1)+0.1)
df(2,3)=dcos(x(3))
df(3,1)=-x(2)*dexp(-x(1)*x(2))
df(3,2)=-x(1)*dexp(-x(1)*x(2))
df(3,3)=20d0
end subroutine jac
end module broyden2
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   Broyden 第二公式计算非线性方程组主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.  result.txt 给出了计算结果
```

```
!      2.   Hmatrix.txt 给出了计算中的 H 矩阵值
!---------------------------------------------------
!  Post Script :
!      1.
!      2.
!---------------------------------------------------
use  inv_mat
use  broyden2
implicit real*8(a-z)
integer::N=3
real*8::x0(3)
open(unit=11,file='result.txt')
open(unit=12,file='Hmatrix.txt')
x0=(/0.1d0,0.1d0,-0.1d0/)
!1d-8  表示允许的误差容限
call solve(x0,n,1d-8)
end program main
```

6. 实验结论

计算迭代序列保存在文件 result.txt 文件中，打开该文件如图 6-3 所示。

```
            Broyden 第二公式计算序列

      1    0.4998696729    0.0194668487    -0.5215204865
      2    0.5000501900    0.0028710742    -0.5240790742
      3    0.5000035538    0.0002160914    -0.5236051277
      4    0.4999999829    0.0000044806    -0.5235982600
      5    0.4999999991    0.0000000323    -0.5235987798
      6    0.5000000000   -0.0000000005    -0.5235987901
      7    0.5000000000   -0.0000000008    -0.5235987902
```

图 6-3　Broyden 第二公式迭代序列

计算中的迭代矩阵保存在文件 Hmatrix.txt 中，结果如下所示。

$$
\mathbf{H}_1 = \begin{pmatrix} 0.3333331840 & 0.0021086068 & 0.0016605204 \\ 0.0000102385 & -0.0308688253 & -0.0001527577 \\ 0.0000161570 & 0.0015358359 & 0.0500076828 \end{pmatrix}
$$

$$
\mathbf{H}_2 = \begin{pmatrix} 0.3333311218 & 0.0022981893 & 0.0016897486 \\ 0.0001998210 & -0.0482980293 & -0.0028398360 \\ 0.0000453851 & -0.0011512424 & 0.0495934131 \end{pmatrix}
$$

$$
\mathbf{H}_3 = \begin{pmatrix} 0.3333287819 & 0.0021649756 & 0.0017135288 \\ 0.0000666073 & -0.0558818462 & -0.0014860331 \\ 0.0000691654 & 0.0002025605 & 0.0493517430 \end{pmatrix}
$$

$$\mathbf{H}_4=\begin{pmatrix} 0.3333275048 & 0.0020892961 & 0.0017159849 \\ -0.0000090722 & -0.0603666532 & -0.0013404830 \\ 0.0000716215 & 0.0003481106 & 0.0493470193 \end{pmatrix}$$

$$\mathbf{H}_5=\begin{pmatrix} 0.3333274882 & 0.0020938623 & 0.0017165185 \\ -0.0000045060 & -0.0616254186 & -0.0014875624 \\ 0.0000721550 & 0.0002010312 & 0.0493298339 \end{pmatrix}$$

$$\mathbf{H}_6=\begin{pmatrix} 0.3333271167 & 0.0021070900 & 0.0017206718 \\ 0.0000087217 & -0.0620963485 & -0.0016354293 \\ 0.0000763084 & 0.0000531643 & 0.0492834053 \end{pmatrix}$$

$$\mathbf{H}_7=\begin{pmatrix} 0.3333263905 & 0.0021288222 & 0.0017283355 \\ 0.0000304539 & -0.0627467538 & -0.0018647887 \\ 0.0000839721 & -0.0001761951 & 0.0492025238 \end{pmatrix}$$

6.5 DFP方法

1. 实验基本原理

DFP 方法全称是 Davidon-Fletcher-Powell 方法，也是拟牛顿法的一种。Broyden 方法及其第二方法是秩 1 的拟牛顿方法，而本节要介绍的 DFP 方法与下一个实验的 BFS 方法是是秩 2 的校正方法。

这里给出迭代格式

$$\begin{cases} \mathbf{x}_{k+1}=\mathbf{x}_k-\mathbf{H}_k\mathbf{f}(\mathbf{x}_k) \\ \mathbf{H}_{k+1}=\mathbf{H}_k+\dfrac{(\Delta\mathbf{x}_k)(\Delta\mathbf{x}_k)^T}{(\Delta\mathbf{x}_k)^T\mathbf{y}_k}-\dfrac{\mathbf{H}_k\mathbf{y}_k\mathbf{y}_k{}^T\mathbf{H}_k}{\mathbf{y}_k\mathbf{H}_k\mathbf{y}_k} \end{cases}$$

其中

$$\Delta\mathbf{x}_k=\mathbf{x}_{k+1}-\mathbf{x}_k,\mathbf{y}_k=\mathbf{f}(\mathbf{x}_{k+1})-\mathbf{f}(\mathbf{x}_k)$$

其推导原理这里略去，有兴趣的读者可以阅读《数值计算方法（下）》（南京大学编，科学出版社出版，林成森）或者《数值计算方法》（冯康等著，国防工业出版社 1978 年出版）

2. 实验目的与要求

- 了解 DFP 方法大致原理。
- 会使用作者编写的 DFP 方法计算非线性方程组。
- 最好能自行编写出 DFP 方法函数。

3. 实验内容与数据来源

计算非线性方程组

$$\begin{cases} 3x-\cos(yz)-\dfrac{1}{2}=0 \\ x^2-81(y+0.1)^2+\sin z+1.06=0 \\ e^{-xy}+20z+\dfrac{10\pi-3}{3}=0 \end{cases}$$

4. 函数调用接口说明

输入参变量	数据类型	变量说明
X0	REAL*8(N)	初值
N	INTEGER	方程的维数
TOL	REAL*8	误差容限

5. 程序代码

```
  module inv_mat
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-----------------------------------------------------
!  Description : 计算逆矩阵
!
!-----------------------------------------------------
!  Contains    :
!      1.   inv 计算逆矩阵
!-----------------------------------------------------
contains
subroutine inv(A,invA,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
```

```
!--------------------------------------------------------
!  Purpose   :  计算逆矩阵
!
!--------------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!------------------------------------------------------
!  Post Script :
!      1.
!      2.
!------------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i
real*8::A(n,n),invA(n,n),E(n,n)
E=0
!设置 E 为单位矩阵
do i=1,n
   E(i,i)=1
end do
call mateq(A,E,invA,N,N)
end subroutine inv
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!--------------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程
!               AX=B
!--------------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   B(N,M)右矩阵
!      3.   N 方程维数,
!      4.   M---B 的列数
!  Output parameters  :
!      1.  X  方程的根（N,M）维
!      2.
!  Common parameters  :
!
!------------------------------------------------------
implicit real*8(a-z)
integer::N,M,i
real*8::A(N,N),B(N,M),X(N,M)
real*8::btemp(N),xtemp(N)
do i=1,M
    btemp=B(:,i)
    call elgauss(A,btemp,xtemp,N)
```

```
    X(:,i)=xtemp
end do
end subroutine mateq
subroutine elgauss(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   b(N)右向量
!       3.   N方程维数
!  Output parameters  :
!       1.  x  方程的根
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!###########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id_max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
      if (dabs(Ab(i,k))>elmax) then
         elmax=Ab(i,k)
         id_max=i
      end if
    end do
!至此，已经完成查找最大元素，查找完成以后与 第k行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id_max,:)

    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!##########################################################
```

```
     do i=k+1,N
       temp=Ab(i,k)/Ab(k,k)
       Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
     end do
  end do
  !------------------------------
  ! 经过上一步，Ab 已经化为如下形式的矩阵
  !             | *  *  *  *  # |
  !     [A b]= | 0  *  *  *  # |
  !            | 0  0  *  *  # |
  !            | 0  0  0  *  # |
  !
  Aup(:,:)=Ab(1:N,1:N)
  bup(:)=Ab(:,N+1)
  !调用用上三角方程组的回带方法
  call uptri(Aup,bup,x,n)
  end subroutine elgauss
  subroutine uptri(A,b,x,N)
  !---------------------------------subroutine  comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      :  2010-4-8
  !-----------------------------------------------------
  !  Purpose   :  上三角方程组的回带方法
  !               Ax=b
  !-----------------------------------------------------
  !  Input  parameters  :
  !       1.   A(N,N)系数矩阵
  !       2.   b(N)右向量
  !       3.   N 方程维数
  !  Output parameters  :
  !       1.  x  方程的根
  !       2.
  !  Common parameters  :
  !
  !----------------------------------------------------
  implicit real*8(a-z)
  integer::i,j,N
  real*8::A(N,N),b(N),x(N)
  x(N)=b(N)/A(N,N)
  !回带部分
  do i=n-1,1,-1
      x(i)=b(i)
     do j=i+1,N
      x(i)=x(i)-a(i,j)*x(j)
     end do
      x(i)=x(i)/A(i,i)
  end do
  end subroutine uptri
  end module inv_mat
  module  DFP
  !----------------------------------------module coment
  !  Version      :  V1.0
  !  Coded by     :  syz
  !  Date         :
```

```
!--------------------------------------------------------
!  Description :   要计算的方程相关模块
!
!--------------------------------------------------------
!  Contains    :
!      1.    函数文件
!      2.    偏导数文件
!--------------------------------------------------------
!  Post Script :
!      1.
!      2.
!--------------------------------------------------------
use inv_mat
contains
subroutine solve(x0,N,tol)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!--------------------------------------------------------
!  Purpose   :   DFP 方法函数
!
!--------------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.
!       2.
!  Common parameters  :
!
!-------------------------------------------------------
!  Post Script :
!       1.
!       2.
!-------------------------------------------------------
implicit real*8(a-z)
integer::i,n,itmax=50
real*8::x1(n),x0(n),y(n),f0(n),f1(n),dx(n)
real*8::v1(n),v2(n)
real*8::H0(n,n),H1(n,n),df(n,n)
real*8::m1(n,n),m2(n,n)
!itmax 最大允许迭代次数
!tol 误差容限
call jac(df,x0)
!注意：设置 H0 初值为 偏导数矩阵的逆矩阵
!  也可以直接设置 H0 为单位矩阵，但是那样收敛较慢
!  设置 H0 初值为偏导数逆矩阵，但是在迭代过程中则不计算逆矩阵
call inv(df,H0,N)
write(11,101)
write(12,102)
do i=1,itmax
    !计算函数值
    call func(f0,x0)
    !更新方程的根
```

```
        x1=x0-matmul(H0,f0)
        dx=x1-x0
        call func(f1,x1)
        y=f1-f0
        m1=vvmat(dx,dx,n)
        t1=vdot(dx,y,N)
        !第一项矩阵
        m1=m1/t1
        !第二项分子结果为矩阵
        m2=vvmat(y,y,n)
        m2=matmul(H0,M2)
        M2=matmul(m2,H0)
        !第二项分母
        v1=matmul(H0,y)
        t2=vdot(y,v1,N)
        m2=m2/t2
        H1=H0+m1-m2
        !把计算中的 H 矩阵写入文件
        write(12,103)i,H0
        x0=x1
        H0=H1
        !把计算迭代序列写入文件
        write(11,104)i,x0
        !判断计算精度，当满足误差容限时退出循环。
        dx2=dsqrt(dx(1)**2+dx(2)**2)
         if (dx2<tol) exit
    !------
    end do
    101 format(/,T18,'DFP 方法计算序列',/)
    102 format(/,T5,'DFP 方法 H 矩阵序列为: ',/)
    103 format(T5,'iter=',I4,/,<N>(<N>F16.10/),/)
    104 format(I4,3F16.10)
    end subroutine solve
    function vdot(a,b,N)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :
    !-----------------------------------------------------
    !  Purpose   : 计算向量 a,b 内积
    !
    !-----------------------------------------------------
    !  Input  parameters  :
    !      1.   N 向量维数
    !      2.
    !----------------------------------------------------
    !  Post Script :
    !      1.   vdot=a1(1)*(1)+a(2)*b(2)+...
    !      2.
    !----------------------------------------------------
    integer::i,N
    real*8::a(n),b(n),vdot
    vdot=0
    do i=1,N
      vdot=vdot+a(i)*b(i)
```

```
end do
end function vdot
function vvmat(a,b,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  N 维列向量 a 乘以 N 维横向量 b
!              结果为  n*n 维矩阵
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
integer::n,i,j
real*8::a(n),b(n),vvmat(n,n)
do i=1,n
  do j=1,n
    vvmat(i,j)=a(i)*b(j)
  end do
end do
end function vvmat
subroutine func(f,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.    f 方程函数
!       2.
!  Common parameters  :
!
!-----------------------------------------------------
implicit real*8(a-z)
real*8::x(3),f(3),pi=3.141592653589793
f(1)=3*x(1)-dcos(x(2)*x(3))-1d0/2
f(2)=x(1)**2-81*(x(2)+0.1d0)**2+dsin(x(3))+1.06d0
f(3)=dexp(-x(1)*x(2))+20*x(3)+(10*pi-3d0)/3
end subroutine func
subroutine  jac(df,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  偏导数矩阵
!
```

```
!-------------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.
!       2.
!  Common parameters  :
!
!------------------------------------------------------
!  Post Script :
!       1.
!       2.
!------------------------------------------------------
implicit real*8(a-z)
real*8::x(3),df(3,3)
df(1,1)=3d0
df(1,2)=x(3)*dsin(x(2)*x(3))
df(1,3)=x(2)*dsin(x(2)*x(3))
df(2,1)=2*x(1)
df(2,2)=-162*(x(1)+0.1)
df(2,3)=dcos(x(3))
df(3,1)=-x(2)*dexp(-x(1)*x(2))
df(3,2)=-x(1)*dexp(-x(1)*x(2))
df(3,3)=20d0
end subroutine jac
end module DFP
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-------------------------------------------------------
!  Purpose   :   DFP 方法计算非线性方程组主函数
!
!-------------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.   result.txt 给出了计算结果
!       2.   Hmatrix.txt 给出了计算中的 H 矩阵值
!-------------------------------------------------------
!  Post Script :
!       1.
!       2.
!-------------------------------------------------------
use  inv_mat
use  DFP
implicit real*8(a-z)
integer::N=3
real*8::x0(3)
open(unit=11,file='result.txt')
open(unit=12,file='Hmatrix.txt')
x0=(/0.1d0,0.1d0,-0.1d0/)
```

```
!1d-8  表示允许的误差容限
call solve(x0,n,1d-8)
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，打开该文件结果如图 6-4 所示。

DFP方法计算序列

1	0.4998696729	0.0194668487	-0.5215204865
2	0.4991991549	0.0089641311	-0.5223350556
3	0.5000200726	0.0007381429	-0.5235828977
4	0.4999985735	0.0000296401	-0.5235972866
5	0.5000000243	0.0000000412	-0.5235987956
6	0.4999999998	0.0000000002	-0.5235987901
7	0.5000000000	-0.0000000008	-0.5235987902

图 6-4　DFP 方法迭代序列

其中 **H** 矩阵序列保存于文件 Hmatrix.txt 中，打开该文件结果如下所示。

$$\mathbf{H}_1=\begin{pmatrix} 0.3333331840 & 0.0021086068 & 0.0016605204 \\ 0.0000102385 & -0.0308688253 & -0.0001527577 \\ 0.0000161570 & 0.0015358359 & 0.0500076828 \end{pmatrix}$$

$$\mathbf{H}_2=\begin{pmatrix} 0.3344372773 & 0.0008166496 & 0.0003436749 \\ -0.0023109383 & -0.0302418744 & 0.0023169055 \\ -0.0003706664 & 0.0027619617 & 0.0505796669 \end{pmatrix}$$

$$\mathbf{H}_3=\begin{pmatrix} 0.3347465856 & 0.0000923753 & 0.0001264187 \\ 0.0000498164 & -0.0545071297 & -0.0013364636 \\ -0.0006152287 & -0.0013443512 & 0.0502532185 \end{pmatrix}$$

$$\mathbf{H}_4=\begin{pmatrix} 0.3348236231 & 0.0000064424 & 0.0000442658 \\ -0.0001065610 & -0.0591926213 & -0.0014172857 \\ -0.0007112128 & -0.0014040040 & 0.0503470828 \end{pmatrix}$$

$$\mathbf{H}_5=\begin{pmatrix} 0.3348299588 & 0.0000103334 & 0.0000419415 \\ 0.0000146919 & -0.0616651107 & -0.0015433676 \\ -0.0007118937 & -0.0015476048 & 0.0503427453 \end{pmatrix}$$

$$\mathbf{H}_6=\begin{pmatrix} 0.3348338005 & -0.0000122464 & 0.0000400061 \\ -0.0000364824 & -0.0617495100 & -0.0015318121 \\ -0.0007158479 & -0.0015249547 & 0.0503447155 \end{pmatrix}$$

$$\mathbf{H}_7=\begin{pmatrix} 0.3359260250 & 0.0003438095 & -0.0002362068 \\ 0.0005157805 & -0.0632140367 & -0.0017056005 \\ -0.0010835733 & -0.0018023000 & 0.0504344420 \end{pmatrix}$$

6.6 BFS方法

1. 实验基本原理

BFS 是 Broyden-Flether-Shanno 三位作者提出来的方法，也是拟牛顿法的方法范畴。前面已经说明拟牛顿法是实用价值和理论价值都比较高的算法,前面介绍的三个算法都是比较优秀的。很多数值计算表明 BFS 方法比 DFP 方法有更好的数值稳定性，是拟牛顿法里比较成功的算法。

这里给出迭代格式

$$\begin{cases} \mathbf{x}_{k+1}=\mathbf{x}_k-\mathbf{H}_k\mathbf{f}(\mathbf{x}_k) \\ \mathbf{H}_{k+1}=\mathbf{H}_k+\dfrac{\mu_k\Delta\mathbf{x}_k(\Delta\mathbf{x}_k)^T-\mathbf{H}_k\mathbf{y}_k(\Delta\mathbf{x}_k)^T-(\Delta\mathbf{x}_k)\mathbf{y}_k{}^T\mathbf{H}_k}{(\Delta\mathbf{x}_k)^T\mathbf{y}_k} \\ \mu_k=1+\dfrac{\mathbf{y}_k{}^T\mathbf{H}_k\mathbf{y}_k}{(\Delta\mathbf{x}_k)^T\mathbf{y}_k} \end{cases}$$

其中

$$\Delta\mathbf{x}_k=\mathbf{x}_{k+1}-\mathbf{x}_k,\mathbf{y}_k=\mathbf{f}(\mathbf{x}_{k+1})-\mathbf{f}(\mathbf{x}_k)$$

至于迭代过程的推导，这里不再给出，有兴趣的读者可以参考相关专著。

2. 实验目的与要求

- 了解 BFS 算法的大致流程。
- 明白迭代格式中各变量的意义。
- 会使用作者编写的 BFS 方法计算非线性方程组。

3. 实验内容与数据来源

采用 BFS 方法计算上一节中的非线性方程组。

4. 函数调用接口说明

输入参变量	数据类型	变量说明
X0	REAL*8(N)	初值
N	INTEGER	方程的维数
TOL	REAL*8	误差容限

5. 程序代码

```
module inv mat
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010-4-8
!-----------------------------------------------------
!  Description : 计算逆矩阵
!
!-----------------------------------------------------
!  Contains    :
!      1.   inv 计算逆矩阵
!-----------------------------------------------------
contains
subroutine inv(A,invA,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算逆矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
```

```
integer::n
integer::i
real*8::A(n,n),invA(n,n),E(n,n)
E=0
!设置 E 为单位矩阵
do i=1,n
  E(i,i)=1
end do
call mateq(A,E,invA,N,N)
end subroutine inv
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程
!               AX=B
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   B(N,M)右矩阵
!      3.   N 方程维数,
!      4.   M---B 的列数
!  Output parameters  :
!      1.  X  方程的根（N,M）维
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N,M,i
real*8::A(N,N),B(N,M),X(N,M)
real*8::btemp(N),xtemp(N)
do i=1,M
   btemp=B(:,i)
   call elgauss(A,btemp,xtemp,N)
   X(:,i)=xtemp
end do
end subroutine mateq
subroutine elgauss(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
```

```
!       2.
!  Common parameters  :
!
!---------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!##########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id_max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
      if (dabs(Ab(i,k))>elmax) then
         elmax=Ab(i,k)
         id_max=i
      end if
    end do
 !至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id_max,:)
    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!#########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!             | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!            | 0  0  *  *  # |
!            | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine elgauss
subroutine uptri(A,b,x,N)
!----------------------------------subroutine  comment
```

```
    ! Version   : V1.0
    ! Coded by  : syz
    ! Date      : 2010-4-8
    !-----------------------------------------------------
    ! Purpose   : 上三角方程组的回带方法
    !             Ax=b
    !-----------------------------------------------------
    ! Input parameters :
    !     1.  A(N,N)系数矩阵
    !     2.  b(N)右向量
    !     3.  N方程维数
    ! Output parameters :
    !     1. x  方程的根
    !     2.
    ! Common parameters :
    !
    !----------------------------------------------------
    implicit real*8(a-z)
    integer::i,j,N
    real*8::A(N,N),b(N),x(N)
    x(N)=b(N)/A(N,N)
    !回带部分
    do i=n-1,1,-1
       x(i)=b(i)
      do j=i+1,N
       x(i)=x(i)-a(i,j)*x(j)
      end do
       x(i)=x(i)/A(i,i)
    end do
    end subroutine uptri
    end module inv_mat
    module  m_bfs
    !----------------------------------------module coment
    ! Version    :  V1.0
    ! Coded by   :  syz
    ! Date       :  2010.05.11
    !-----------------------------------------------------
    ! Description :   要计算的方程相关模块
    !
    !-----------------------------------------------------
    ! Contains    :
    !     1.    函数文件
    !     2.    偏导数文件
    !-----------------------------------------------------
    ! Post Script :
    !     1.
    !     2.
    !-----------------------------------------------------
    use inv_mat
    contains
    subroutine solve(x0,N,tol)
    !---------------------------------subroutine  comment
    ! Version   : V1.0
    ! Coded by  : syz
    ! Date      : 2010.05.11
```

```
!----------------------------------------------------
!  Purpose   :  BFS 方法函数
!
!----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!---------------------------------------------------
!  Post Script :
!      1.
!      2.
!---------------------------------------------------
implicit real*8(a-z)
integer::i,n,itmax=50
real*8::x1(n),x0(n),y(n),f0(n),f1(n),dx(n)
real*8::v1(n),v2(n),v3(n)
real*8::H0(n,n),H1(n,n),df(n,n)
real*8::m1(n,n),m2(n,n),m3(n,n)
!itmax 最大允许迭代次数
!tol 误差容限
call jac(df,x0)
!注意：设置 H0 初值为 偏导数矩阵的逆矩阵
!  也可以直接设置 H0 为单位矩阵，但是那样收敛较慢
!  设置 H0 初值为偏导数逆矩阵，但是在迭代过程中则不计算逆矩阵
call inv(df,H0,N)
write(11,101)
write(12,102)
do i=1,itmax
    !计算函数值
    call func(f0,x0)
    !更新方程的根
    x1=x0-matmul(H0,f0)
    dx=x1-x0
    call func(f1,x1)
    y=f1-f0
    !这段程序求得 miu
    v1=matmul(H0,y)
    t1=vdot(y,v1,n)
    t2=vdot(dx,y,N)
    u=1d0+t1/t2
    !分子第一项
    m1=vvmat(dx,dx,n)*u
    !分子第二项
    m2=vvmat(dx,y,n)
    m2=matmul(m2,H0)
    !分子第三项
    v3=matmul(H0,y)
    m3=vvmat(v3,dx,n)
    !分母
    t3=vdot(dx,y,n)
```

```
    H1=(m1-m2-m3)/t3
    H1=H0+H1
    !把计算中的H矩阵写入文件
    write(12,103)i,H0
    x0=x1
    H0=H1
    !把计算迭代序列写入文件
    write(11,104)i,x0
    !判断计算精度，当满足误差容限时退出循环。
    dx2=dsqrt(dx(1)**2+dx(2)**2)
     if (dx2<tol) exit
!------
end do
101 format(/,T5,'BFS方法计算序列',/)
102 format(/,T5,'BFS方法H矩阵序列为：',/)
103 format(T5,'iter=',I4,/,<N>(<N>F16.10/),/)
104 format(I8,3F16.10)
end subroutine solve
function vdot(a,b,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   : 计算向量a,b内积
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.   N 向量维数
!       2.
!----------------------------------------------------
!  Post Script :
!       1.   vdot=a1(1)*(1)+a(2)*b(2)+...
!       2.
!----------------------------------------------------
integer::i,N
real*8::a(n),b(n),vdot
vdot=0
do i=1,N
  vdot=vdot+a(i)*b(i)
end do
end function vdot
function vvmat(a,b,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  N维列向量a 乘以 N维横向量b
!              结果为  n*n维矩阵
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
```

```
integer::n,i,j
real*8::a(n),b(n),vvmat(n,n)
do i=1,n
  do j=1,n
    vvmat(i,j)=a(i)*b(j)
  end do
end do
end function vvmat
subroutine func(f,x)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!----------------------------------------------------
!  Purpose   :  方程函数
!
!----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.    f 方程函数
!       2.
!  Common parameters  :
!
!---------------------------------------------------
implicit real*8(a-z)
real*8::x(3),f(3),pi=3.141592653589793
f(1)=3*x(1)-dcos(x(2)*x(3))-1d0/2
f(2)=x(1)**2-81*(x(2)+0.1d0)**2+dsin(x(3))+1.06d0
f(3)=dexp(-x(1)*x(2))+20*x(3)+(10*pi-3d0)/3
end subroutine func
subroutine  jac(df,x)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!----------------------------------------------------
!  Purpose   :  偏导数矩阵
!
!----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.
!       2.
!  Common parameters  :
!
!---------------------------------------------------
!  Post Script :
!       1.
!       2.
!---------------------------------------------------
implicit real*8(a-z)
```

```
real*8::x(3),df(3,3)
df(1,1)=3d0
df(1,2)=x(3)*dsin(x(2)*x(3))
df(1,3)=x(2)*dsin(x(2)*x(3))
df(2,1)=2*x(1)
df(2,2)=-162*(x(1)+0.1)
df(2,3)=dcos(x(3))
df(3,1)=-x(2)*dexp(-x(1)*x(2))
df(3,2)=-x(1)*dexp(-x(1)*x(2))
df(3,3)=20d0
end subroutine jac
end module m_bfs
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.11
!----------------------------------------------------
!  Purpose   :  BFS 方法计算非线性方程组主函数
!
!----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.   result.txt 给出了计算结果
!       2.   Hmatrix.txt 给出了计算中的 H 矩阵值
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
use  inv_mat
use  m_bfs
implicit real*8(a-z)
integer::N=3
real*8::x0(3)
open(unit=11,file='result.txt')
open(unit=12,file='Hmatrix.txt')
x0=(/0.1d0,0.1d0,-0.1d0/)
!1d-8  表示允许的误差容限
call solve(x0,n,1d-8)
end program main
```

6. 实验结论

计算结果保存于文件 result.txt 中，打开该文件如图 6-5 所示。

```
          BFS方法计算序列

  1    0.4998696729    0.0194668487   -0.5215204865
  2    0.4991975995    0.0089854025   -0.5223304185
  3    0.5000163055    0.0007400750   -0.5235811416
  4    0.4999991828    0.0000296287   -0.5235975655
  5    0.5000000118    0.0000000598   -0.5235987907
  6    0.4999999999   -0.0000000002   -0.5235987902
  7    0.5000000000   -0.0000000008   -0.5235987902
```

图 6-5　BFS 方法迭代序列

为了方便读者自行编程比对，这里给出了计算中的 **H** 矩阵序列，该矩阵序列保存于文件 Hmatrix.txt 中，结果如下所示。

$$\mathbf{H}_1=\begin{pmatrix} 0.3333331840 & 0.0021086068 & 0.0016605204 \\ 0.0000102385 & -0.0308688253 & -0.0001527577 \\ 0.0000161570 & 0.0015358359 & 0.0500076828 \end{pmatrix}$$

$$\mathbf{H}_2=\begin{pmatrix} 0.3344394379 & 0.0007871034 & 0.0003372340 \\ -0.0023155255 & -0.0301791423 & 0.0023305808 \\ -0.0003714067 & 0.0027720853 & 0.0505818738 \end{pmatrix}$$

$$\mathbf{H}_3=\begin{pmatrix} 0.3346476413 & 0.0000937951 & 0.0001686036 \\ 0.0000652752 & -0.0545019170 & -0.0013432949 \\ -0.0003311433 & -0.0013486321 & 0.0501247966 \end{pmatrix}$$

$$\mathbf{H}_4=\begin{pmatrix} 0.3347139976 & 0.0000162494 & 0.0000957237 \\ -0.0000597815 & -0.0591918888 & -0.0014391899 \\ -0.0004126429 & -0.0014302985 & 0.0502057763 \end{pmatrix}$$

$$\mathbf{H}_5=\begin{pmatrix} 0.3347168978 & 0.0000081268 & 0.0000944481 \\ 0.0000093008 & -0.0616554221 & -0.0015412782 \\ -0.0004124026 & -0.0015435411 & 0.0502021149 \end{pmatrix}$$

$$\mathbf{H}_6=\begin{pmatrix} 0.3347180653 & -0.0000052051 & 0.0000937172 \\ -0.0000155068 & -0.0617803354 & -0.0015400880 \\ -0.0004137463 & -0.0015374526 & 0.0502026310 \end{pmatrix}$$

$$\mathbf{H}_7=\begin{pmatrix} 0.3347700935 & 0.0000495876 & 0.0000867298 \\ 0.0000715923 & -0.0623905179 & -0.0015670351 \\ -0.0004253221 & -0.0015860360 & 0.0502033949 \end{pmatrix}$$

6.7 拓展收敛域之数值延拓法

1. 实验基本原理

两个拓扑流形 $\mathbf{M}$ 与 $\mathbf{N}$ 间有两个连续映射 $\mathbf{f}$ 与 $\mathbf{g}$

$$\mathbf{f},\mathbf{g}:\mathbf{M}\to\mathbf{N}$$

如果存在一簇连续映射

$$\begin{aligned}\mathbf{H}:\mathbf{M}\times[0,1]\to\mathbf{N}\\ \mathbf{x},t\to\mathbf{H}(\mathbf{x},t)\end{aligned}$$

在这簇映射中，如果 $\mathbf{H}(\mathbf{x},t)$ 中参数 t 由 0 连续的变化大 1 时，有

$$\mathbf{H}(\mathbf{x},\mathbf{t})=\mathbf{f}(\mathbf{x}),\mathbf{H}(\mathbf{x},\mathbf{t})=\mathbf{g}(\mathbf{x})$$

使映射 $\mathbf{f}$ 连续的变为 $\mathbf{g}$，则称此连续映射 $\mathbf{f}$ 与 $\mathbf{g}$ 同伦。

对于适定的非线性方程组

$$\mathbf{F}(\mathbf{x})=\mathbf{0}$$

可以构造映射 $\mathbf{H}(\mathbf{x},t):D\times[0,1]\subset R^{n+1}\to R^n$，使之满足条件

$$\mathbf{H}(\mathbf{x},0)=\mathbf{F}_0(\mathbf{x}),\mathbf{H}(\mathbf{x},1)=\mathbf{F}(\mathbf{x}),\forall\mathbf{x}\in D$$

如果方程

$$\mathbf{H}(\mathbf{x},t)=\mathbf{0},t\in[0,1]$$

有解 $\mathbf{x}=\mathbf{x}(t),\mathbf{x}:[0,1]\to R^n$ 连续的依赖于 t，当 $t=1$ 时，即为方程 $\mathbf{F}(\mathbf{x})=\mathbf{0}$ 的解，此即构造了同伦映射，把原方程的求解问题转化为求解同伦方程

$$\mathbf{H}(\mathbf{x},t)=\mathbf{0},t\in[0,1]$$

的解。

关于同伦 $\mathbf{H}(\mathbf{x},t)=\mathbf{0},t\in[0,1]$，解的曲线 $\mathbf{x}=\mathbf{x}(t)$ 的存在性可以针对特殊情形给出如下的定理。

设映射 $\mathbf{F}:D\subset R^n\to R^n$ 在 D 上连续可导，假定存在一个开球 $S=S\left(\mathbf{x}^0,r\right)\subset D$，使对 $\forall\mathbf{x}\in S$，$\left\|\mathbf{F}'(\mathbf{x})^{-1}\right\|\le\beta$ 成立，其中 $r\ge\beta\left\|\mathbf{F}\left(\mathbf{x}^0\right)\right\|$，则方程

$$\mathbf{F}(\mathbf{x})=(1-t)\mathbf{F}(\mathbf{x}^0),t\in[0,1],\mathbf{x}\in S$$

存在唯一的解 $\mathbf{x}:[0,1]\to S\subset R^n$，且 $\mathbf{x}(t)$ 连续可导并满足常微分方程初值问题:

$$\begin{cases}\mathbf{x}'(t)=-\left[\mathbf{F}'(\mathbf{x}(t))\right]^{-1}\mathbf{F}(\mathbf{x}^0),\forall t\in[0,1]\\ \mathbf{x}(0)=\mathbf{x}^0\end{cases}$$

上述定理表明同伦方程

$$\mathbf{H}(\mathbf{x},t)=\mathbf{F}(\mathbf{x})+(t-1)\mathbf{F}(\mathbf{x}^0)=\mathbf{0}$$

存在唯一的解.除了上面给出的同伦之外，还可以构造其他一些形式的同伦方程，这里不多作介绍。

数值延拓法是计算同伦方程的一种有效算法，目前相关文献中一般也是处理适定方程组.这里对这一方法作了推广，用以处理大范围收敛的非线性回归问题.作上述推广是基于以下的考虑。如果测量数据没有误差，则任意选择与方程维数相同个数的测量数据组成方程组，然后采用数值延拓法或者参数微分法计算同伦方程的解，一般这样的结果有一定的精度保证，所以也不失为一种可行的方法。但是，如果在选择的数据中有些数据的测量误差非常大，则由参数微分法计算的解会很不理想，甚至结果不可靠。在这个过程中没有充分利用大量测量数据的统计信息。

设 $\mathbf{H}(\mathbf{x},t):D\times[0,1]\subset R^{n+1}\to R^n$ 是满足 $\mathbf{H}(\mathbf{x},0)=\mathbf{F}_0(\mathbf{x})$，$\mathbf{H}(\mathbf{x},1)=\mathbf{F}(\mathbf{x}),\forall\mathbf{x}\in D$ 的已知同伦，可以将区间 $[0,1]$ 划分为 N 等分，记 $t_i=\dfrac{i}{N}(i=1,\cdots,N-1)$。用某种迭代法（如牛顿法，拟牛顿法等）求方程 $\mathbf{H}(\mathbf{x},t_i)=\mathbf{0},(i=1,2,\cdots,N)$ 的解 $\mathbf{x}^i$ 时，如果 t_i-t_{i-1} 充分小，则可以期望 $\mathbf{x}^{i-1}$ 是 $\mathbf{x}^i$ 的一个足够好的近似，从而使迭代法收敛。在计算第 i 个方程组时，是为超定非线性方程组，可以采用高斯-牛顿法进行计算，由此得到数值延拓法序列

$$\begin{cases}\mathbf{x}^{i,j+1}=\mathbf{x}^{i,j}-\left\{\left[\mathbf{H}_\mathbf{x}(\mathbf{x}^{i,j},t_i)\right]^T\mathbf{H}_\mathbf{x}(\mathbf{x}^{i,j},t_i)\right\}^{-1}\left[\mathbf{H}_\mathbf{x}(\mathbf{x}^{i,j},t_i)\right]^T\mathbf{H}(\mathbf{x}^{i,j},t_i)\\ \mathbf{x}^{1,0}=\mathbf{x}^0,\mathbf{x}^{i+1,0}=\mathbf{x}^{i,m_i}\\ \mathbf{x}^{k+1}=\mathbf{x}^k-\left\{\left[\mathbf{H}_\mathbf{x}(\mathbf{x}^k,1)\right]^T\mathbf{H}_\mathbf{x}(\mathbf{x}^k,1)\right\}^{-1}\left[\mathbf{H}_\mathbf{x}(\mathbf{x}^k,1)\right]^T\mathbf{H}(\mathbf{x}^k,1)\\ \mathbf{x}^N=\mathbf{x}^{N,0}\\ j=0,1,\cdots,m_j-1;i=1,2,\cdots,N-1;k=N,N+1,\cdots\end{cases}$$

上述计算方法是一种大范围收敛方法，第一式利用延拓法求出 $\mathbf{x}(1)$ 一个足够好的近似，使之进入高斯-牛顿法的收敛域，从而保证后续计算收敛。在实际计算中，选择一定的同伦映射后可以对上面的计算方法作进一步简化.如果同伦映射取

$$\mathbf{H}(\mathbf{x},t)=\mathbf{F}(\mathbf{x})+(t-1)\mathbf{F}(\mathbf{x}^0)=\mathbf{0}$$

$m_j\equiv 1$,则大范围的高斯-牛顿序列简化为

$$\begin{cases}\mathbf{x}^{k+1}=\mathbf{x}^k-\left[\mathbf{F}_\mathbf{x}\left(\mathbf{x}^k\right)^T\mathbf{F}_\mathbf{x}\left(\mathbf{x}^k\right)\right]^{-1}\mathbf{F}_\mathbf{x}\left(\mathbf{x}^k\right)^T\left[\mathbf{F}\left(\mathbf{x}^k\right)+\left(\dfrac{k}{N}-1\right)\mathbf{F}\left(\mathbf{x}^0\right)\right]\\(k=0,1,\cdots,N-1)\\\mathbf{x}^{k+1}=\mathbf{x}^k-\left[\mathbf{F}_\mathbf{x}\left(\mathbf{x}^k\right)^T\mathbf{F}_\mathbf{x}\left(\mathbf{x}^k\right)\right]^{-1}\mathbf{F}_\mathbf{x}\left(\mathbf{x}^k\right)^T\mathbf{F}\left(\mathbf{x}^k\right)\\(k=N,N+1,\cdots)\end{cases}$$

一般而言，通常用上述公式的第一式获得迭代初值，所以 N 不必取很大。回到适定方程情况则迭代格式即简化为

$$\begin{cases}\mathbf{x}^{k+1}=\mathbf{x}^k-\mathbf{F}'\left(\mathbf{x}^k\right)^{-1}\left[\mathbf{F}\left(\mathbf{x}^k\right)-\left(1-\dfrac{k}{N}\right)\mathbf{F}\left(\mathbf{x}^0\right)\right]\\k=0,1,\cdots,N-1\\\mathbf{x}^{k+1}=\mathbf{x}^k-\mathbf{F}'\left(\mathbf{x}^k\right)^{-1}\mathbf{F}\left(\mathbf{x}^k\right)\\k=N,N+1,\cdots\end{cases}$$

2. 实验目的与要求

- 大致了解同伦的数学含义。
- 了解数值延拓法的基本思想。
- 会使用作者编写的程序进行相关问题处理。

3. 实验内容与数据来源

计算非线性方程组

$$\mathbf{F}(\mathbf{X})=\begin{pmatrix}x_1^2-x_2+1\\x_1-\cos\left(\dfrac{\pi}{2}x_2\right)\end{pmatrix}=\mathbf{0}$$

迭代初值取 $\mathbf{X}_0=\begin{pmatrix}1\\0\end{pmatrix}$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明

XT	REAL*8(N)	终止点向量
输入参变量	**数据类型**	**变量说明**
X0	REAL*8(N)	起始点向量
N	INTEGER	方程组维数
STEP	INTEGER	划分序列

5. 程序代码

```
module inv_mat
!----------------------------------------module coment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date       :  2010-4-8
!-----------------------------------------------------
!  Description : 计算逆矩阵
!
!-----------------------------------------------------
!  Contains    :
!      1.   inv 计算逆矩阵
!-----------------------------------------------------
contains
subroutine inv(A,invA,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  计算逆矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i
real*8::A(n,n),invA(n,n),E(n,n)
E=0
!设置 E 为单位矩阵
do i=1,n
   E(i,i)=1
end do
```

```
call mateq(A,E,invA,N,N)
end subroutine inv
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程
!               AX=B
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   B(N,M)右矩阵
!      3.   N方程维数，
!      4.   M---B的列数
!  Output parameters  :
!      1.  X  方程的根（N,M）维
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::N,M,i
real*8::A(N,N),B(N,M),X(N,M)
real*8::btemp(N),xtemp(N)
do i=1,M
   btemp=B(:,i)
   call elgauss(A,btemp,xtemp,N)
   X(:,i)=xtemp
end do
end subroutine mateq
subroutine elgauss(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
```

```
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!###########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
      if (dabs(Ab(i,k))>elmax) then
         elmax=Ab(i,k)
         id max=i
      end if
    end do
!至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id max,:)
    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!##########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!           | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!           | 0  0  *  *  # |
!           | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine elgauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
```

```
!       1.   A(N,N)系数矩阵
!       2.   b(N)右向量
!       3.   N 方程维数
!  Output parameters  :
!       1.  x  方程的根
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module inv mat
module  homotopy
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   要计算的方程相关模块
!
!-----------------------------------------------------
!  Contains    :
!      1.    函数文件
!      2.    偏导数文件
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use inv_mat
contains
subroutine solve(x0,N,step,xt)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :   homotopy 方法函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
```

```
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::k,n,itmax=50
integer::step  !划分序列
real*8::x1(n),x0(n),y(n),f0(n),f1(n),dx(n)
real*8::x2(n),xt(n),vect1(n)
!xt 计算终值
real*8::H0(n,n),H1(n,n),df(n,n)
real*8::m1(n,n),m2(n,n)
!itmax 最大允许迭代次数
!计算初始函数值
call func(f0,x0)
x1=x0
write(11,101)
do k=1,step-1
   call func(f1,x1)
   call jac(df,x1)
   call inv(df,H1,n)
   vect1=f1-(1d0-k*1d0/step*1d0*f0)
   x2=x1-matmul(H1,vect1)
   !把计算迭代序列写入文件
   write(11,102)k,x2
    x1=x2
end do
101 format(/,T12,'数值延拓法获取初值序列',/)
102 format(I4,3F16.10)
 xt=x2
end subroutine solve
function vdot(a,b,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   : 计算向量 a,b 内积
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.   N 向量维数
!      2.
!----------------------------------------------------
!  Post Script :
!      1.   vdot=a1(1)*(1)+a(2)*b(2)+...
!      2.
!----------------------------------------------------
integer::i,N
real*8::a(n),b(n),vdot
```

```
vdot=0
do i=1,N
  vdot=vdot+a(i)*b(i)
end do
end function vdot
function vvmat(a,b,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  N维列向量a 乘以 N维横向量b
!              结果为  n*n维矩阵
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
integer::n,i,j
real*8::a(n),b(n),vvmat(n,n)
do i=1,n
  do j=1,n
    vvmat(i,j)=a(i)*b(j)
  end do
end do
end function vvmat
subroutine func(f,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.    f 方程函数
!       2.
!  Common parameters  :
!
!-----------------------------------------------------
implicit real*8(a-z)
real*8::x(2),f(2),pi=3.141592653589793
f(1)=x(1)**2-x(2)+1d0
f(2)=x(1)-dcos(pi/2d0*x(2))
end subroutine func
subroutine  jac(df,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
```

```
!  Purpose   :  偏导数矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
real*8::x(2),df(2,2),pi=3.141592653589793
df(1,1)=2d0*x(1)
df(1,2)=-1d0
df(2,1)=1d0
df(2,2)=dsin(pi/2d0*x(2))*pi/2d0
end subroutine jac
end module homotopy
subroutine newton(x0,N,itmax)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  采用牛顿法获得最终计算结果
!
!----------------------------------------------------
!  Post Script :
!      1.   x0 为初值，N 为阶数，tol 为误差容限
!      2.   计算结果保存在文件中
!----------------------------------------------------
use  inv_mat
use  homotopy
implicit real*8(a-z)
integer::i,n,itmax
real*8::x2(N),x1(N),x0(N),dx(N),f(N)
real*8::df(N,N),H(n,n)
x1=x0
write(12,101)
do i=1,itmax
   call func(f,x1)
   call jac(df,x1)
   call inv(df,H,N)
   dx=-matmul(H,f)
   x2=x1+dx
   write(12,102)i,x2
!更新 x 值
   x1=x2
end  do
```

```
101 format(/,T5,'由数值延拓法提供初值后牛顿法计算序列',/)
102 format(I4,2F16.9)
end subroutine newton
program main
!------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      : 2010.05.11
!----------------------------------------------------
!  Purpose   :   数值延拓法计算非线性方程组主函数
!
!----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.   result.txt 给出了计算结果
!      2.   Hmatrix.txt 给出了计算中的 H 矩阵值
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
use  inv_mat
use  homotopy
implicit real*8(a-z)
integer::N=2,step=8
!N 为方程组的维数，step 为微分方程的步数
real*8::x0(2),x1(2),xt(2)
open(unit=11,file='result.txt')
open(unit=12,file='newton.txt')
x0=(/1d0,1d0/)
!直接调用牛顿法，则计算失败不收敛,可以保留此语句
!而注释下面两行语句，进行验证
!--------
!call newton(x0,n,1d-8)
!--------
!step 为微分方程步数
! 通过数值延拓法计算的中间结果放在文件 result.txt 中
! 最后终值放在参数 xt 中回代出来，后面作为牛顿法初值
call solve(x0,n,step,xt)
!把数值延拓法计算结果作为牛顿法初值，最大允许迭代次
!计算结果放在文件 newton.txt 中
call newton(xt,n,5)
end program main
```

6. 实验结论

如果直接采用牛顿迭代法则计算失败，采用数值延拓法计算结果保存在文件 result.txt 中，计算结果如图 6-6 所示。

数值延拓法获取初值序列

1	0.9224091755	0.9698183510
2	0.8375237808	0.9442405531
3	0.7458600681	0.9229050050
4	0.6457200566	0.9069263696
5	0.5343515027	0.8981285736
6	0.4071196983	0.8995585167
7	0.2551067455	0.9169715138

图 6-6　数值延拓法计算获取的初值

在数值延拓法的基础上采用牛顿迭代法 5 次就已经得到较高的精度，计算结果保存在文件 newton.txt 中，内容如图 6-7 所示。

由数值延拓法提供初值后牛顿法计算序列

1	0.056889867	0.963946526
2	0.004358435	0.997259445
3	0.000029419	0.999981260
4	-0.000000042	0.999999999
5	-0.000000044	1.000000000

图 6-7　在延拓法基础上牛顿法获得最终计算结果

6.8　拓展收敛域之参数微分法

1. 实验基本原理

上一节简要介绍了拓扑学中的同伦方法的基本思想，除了采用数值延拓方法之外，还可以采用参数微分法解算非线性方程组。求解同伦方程的可以转化为等价的微分方程组的初值问题.假设 $\mathbf{x}(t)$ 与 $\mathbf{H}$ 都是 Frechet 可微的，同伦方程两边对 t 求导，有

$$\mathbf{H}_{\mathbf{x}}\left(\mathbf{x}(t),t\right)\frac{d\mathbf{x}}{dt}+\mathbf{H}_t\left(\mathbf{x}(t),t\right)=0$$

如果 $\mathbf{H}_{\mathbf{x}}\left(\mathbf{x}(t),t\right)$ 非奇异，则求同伦方程的解等价于微分方程的初值问题

$$\begin{cases}\dfrac{d\mathbf{x}}{dt}=-\left[\mathbf{H}_{\mathbf{x}}\left(\mathbf{x}(t),t\right)\right]^{-1}\mathbf{H}_t\left(\mathbf{x}(t),t\right)\\ \mathbf{x}(0)=\mathbf{x}^0\end{cases}$$

如此，便可以利用常微分方程初值问题的各种解法求 $t_N=1$ 时的数值解 $\mathbf{x}^N$ 作为 $\mathbf{x}=\mathbf{x}(1)$ 的近似解。实际使用时，一般可以针对具体的同伦方程构造一定的微分方程计算格式进行初值问题计算。

比如构造微分方程的欧拉中点求积公式，则有

$$
\begin{cases}
\mathbf{x}^1 = \mathbf{x}^0 - \dfrac{1}{N}\left[\mathbf{F}'\left(\mathbf{x}^0\right)\right]^{-1}\mathbf{F}\left(\mathbf{x}^0\right) \\
\mathbf{x}^{k+\frac{1}{2}} = \mathbf{x}^k + \dfrac{1}{2}\left(\mathbf{x}^k - \mathbf{x}^{k-1}\right), k = 1,\cdots, N-1 \\
\mathbf{x}^{k+1} = \mathbf{x}^k - \dfrac{1}{N}\left[\mathbf{F}'\left(\mathbf{x}^{k+\frac{1}{2}}\right)\right]^{-1}\mathbf{F}\left(\mathbf{x}^0\right)
\end{cases}
$$

实际上即便是采用简单的欧拉格式（非中点格式），在某种意义上参数微分法即相当于牛顿迭代法。可以期望用更精确的数值方法和变步长的数值积分法得到更好的结果，因这涉及到更多的微分流形与拓扑学知识，这里就不多做讨论。

2. 实验目的与要求

- 大致了解参数微分法的基本思想。
- 理解参数微分法公式中各量的意义。
- 会使用作者编写的程序进行相关问题处理。

3. 实验内容与数据来源

计算非线性方程组

$$
\mathbf{F}(\mathbf{X}) = \begin{pmatrix} x_1^2 - x_2 + 1 \\ x_1 - \cos\left(\dfrac{\pi}{2}x_2\right) \end{pmatrix} = \mathbf{0}
$$

迭代初值取 $\mathbf{X}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
XT	REAL*8(N)	积分终点向量
输入参变量	**数据类型**	**变量说明**
X0	REAL*8(N)	积分起点向量
N	INTEGER	方程组维数
STEP	INTEGER	微分方程积分步数

5. 程序代码

```
module inv_mat
!-----------------------------------------module coment
! Version     :  V1.0
! Coded by    :  syz
! Date        :  2010-4-8
!-----------------------------------------------------
! Description : 计算逆矩阵
!
!-----------------------------------------------------
! Contains    :
!     1.   inv 计算逆矩阵
!-----------------------------------------------------
contains
subroutine inv(A,invA,N)
!---------------------------------subroutine  comment
! Version   :  V1.0
! Coded by  :  syz
! Date      :
!-----------------------------------------------------
! Purpose   :  计算逆矩阵
!
!-----------------------------------------------------
! Input  parameters  :
!      1.
!      2.
! Output parameters  :
!      1.
!      2.
! Common parameters  :
!
!----------------------------------------------------
! Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i
real*8::A(n,n),invA(n,n),E(n,n)
E=0
!设置 E 为单位矩阵
do i=1,n
   E(i,i)=1
end do
call mateq(A,E,invA,N,N)
end subroutine inv
subroutine mateq(A,B,X,N,M)
!---------------------------------subroutine  comment
! Version   :  V1.0
! Coded by  :  syz
! Date      :  2010-4-8
```

```
!-------------------------------------------------------
!  Purpose   :  高斯列主元消去法计算矩阵方程
!                AX=B
!-------------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   B(N,M)右矩阵
!       3.   N 方程维数,
!       4.   M---B 的列数
!  Output parameters  :
!       1.  X  方程的根 (N,M) 维
!       2.
!  Common parameters  :
!
!------------------------------------------------------
implicit real*8(a-z)
integer::N,M,i
real*8::A(N,N),B(N,M),X(N,M)
real*8::btemp(N),xtemp(N)
do i=1,M
   btemp=B(:,i)
   call elgauss(A,btemp,xtemp,N)
   X(:,i)=xtemp
end do
end subroutine mateq
subroutine elgauss(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-------------------------------------------------------
!  Purpose   :  高斯列主元消去法
!                Ax=b
!-------------------------------------------------------
!  Input  parameters  :
!       1.   A(N,N)系数矩阵
!       2.   b(N)右向量
!       3.   N 方程维数
!  Output parameters  :
!       1.  x  方程的根
!       2.
!  Common parameters  :
!
!------------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵 [Ab]
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!###########################################################
```

```
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id_max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
       if (dabs(Ab(i,k))>elmax) then
          elmax=Ab(i,k)
          id_max=i
       end if
    end do
!至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id_max,:)
    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!##########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!           | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!           | 0  0  *  *  # |
!           | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine elgauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!                Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
```

```
!
!-----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module inv_mat
module  homotopy
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :
!------------------------------------------------------
!  Description :   要计算的方程相关模块
!
!------------------------------------------------------
!  Contains    :
!      1.    函数文件
!      2.    偏导数文件
!------------------------------------------------------
!  Post Script :
!      1.
!      2.
!------------------------------------------------------
use inv_mat
contains
subroutine solve(x0,N,step,xt)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!------------------------------------------------------
!  Purpose   :   homotopy 方法函数
!
!------------------------------------------------------
!  Input  parameters  :
!       1.
!       2.
!  Output parameters  :
!       1.
!       2.
!  Common parameters  :
!
!-----------------------------------------------------
!  Post Script :
!       1.
```

```
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::k,n,itmax=50
integer::step  !微分方程步数
real*8::x1(n),x0(n),y(n),f0(n),f1(n),dx(n),xbar(n)
real*8::x2(n),xt(n)
!xt 计算终值
real*8::H0(n,n),H1(n,n),df(n,n)
real*8::m1(n,n),m2(n,n)
!itmax 最大允许迭代次数
!tol 误差容限
call func(f0,x0)
call jac(df,x0)
call inv(df,H0,N)
x1=x0-1d0/step*matmul(H0,f0)
write(11,101)
do k=1,step-1
   xbar=x1+1d0/2*(x1-x0)
   call jac(df,xbar)
   call inv(df,H1,n)
   x2=x1-matmul(H1,f0)/step
   !把计算迭代序列写入文件
   write(11,102)k,x2
   x0=x1
    x1=x2
end do
101 format(/,T12,'参数微分法获取初值序列',/)
102 format(I4,3F16.10)
 xt=x2
end subroutine solve
function vdot(a,b,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date     :
!-----------------------------------------------------
!  Purpose   : 计算向量 a,b 内积
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.   N 向量维数
!      2.
!----------------------------------------------------
!  Post Script :
!      1.   vdot=a1(1)*(1)+a(2)*b(2)+...
!      2.
!----------------------------------------------------
integer::i,N
real*8::a(n),b(n),vdot
vdot=0
do i=1,N
  vdot=vdot+a(i)*b(i)
end do
end function vdot
```

```
function vvmat(a,b,n)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  N 维列向量 a 乘以 N 维横向量 b
!              结果为  n*n 维矩阵
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
integer::n,i,j
real*8::a(n),b(n),vvmat(n,n)
do i=1,n
  do j=1,n
    vvmat(i,j)=a(i)*b(j)
  end do
end do
end function vvmat
subroutine func(f,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方程函数
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.    x 自变量
!       2.
!  Output parameters  :
!       1.    f 方程函数
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
real*8::x(2),f(2),pi=3.141592653589793
f(1)=x(1)**2-x(2)+1d0
f(2)=x(1)-dcos(pi/2d0*x(2))
end subroutine func
subroutine  jac(df,x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  偏导数矩阵
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.
```

```
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!---------------------------------------------------
!  Post Script :
!      1.
!      2.
!---------------------------------------------------
implicit real*8(a-z)
real*8::x(2),df(2,2),pi=3.141592653589793
df(1,1)=2d0*x(1)
df(1,2)=-1d0
df(2,1)=1d0
df(2,2)=dsin(pi/2d0*x(2))*pi/2d0
end subroutine jac
end module homotopy
subroutine newton(x0,N,tol)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!----------------------------------------------------
!  Purpose   :  采用牛顿法获得最终计算结果
!
!---------------------------------------------------
!  Post Script :
!      1.   x0 为初值，N 为阶数，tol 为误差容限
!      2.   计算结果保存在文件中
!---------------------------------------------------
use  inv_mat
use  homotopy
implicit real*8(a-z)
integer::i,n,itmax=50,k
!itmax 允许最大迭代次数
real*8::x2(N),x1(N),x0(N),dx(N),f(N)
real*8::df(N,N),H(n,n)
x1=x0
write(12,101)
do i=1,itmax
   call func(f,x1)
   call jac(df,x1)
   call inv(df,H,N)
   dx=-matmul(H,f)
   x2=x1+dx
  write(12,102)i,x2
!更新 x 值
   x1=x2
   !判断迭代停止标准
   len_dx=0
   do k=1,n
     len_dx=len_dx+dx(i)**2
   end do
```

```fortran
    len_dx=dsqrt(len_dx)
    if (len_dx<tol)  exit
  end  do
  101 format(/,T5,'由参数微分提供初值后牛顿法计算序列',/)
  102 format(I4,2F16.9)
  end subroutine newton
  program main
  !-------------------------------------program comment
  !  Version   :  V1.0
  !  Coded by  :  syz
  !  Date      : 2010.05.11
  !-----------------------------------------------------
  !  Purpose   :   参数微分方法计算非线性方程组主函数
  !
  !-----------------------------------------------------
  !  In put data  files :
  !      1.
  !      2.
  !  Output data files  :
  !      1.   result.txt 给出了计算结果
  !      2.   Hmatrix.txt 给出了计算中的 H 矩阵值
  !-----------------------------------------------------
  !  Post Script :
  !      1.
  !      2.
  !-----------------------------------------------------
  use  inv_mat
  use  homotopy
  implicit real*8(a-z)
  integer::N=2,step=8
  !N 为方程组的维数，step 为微分方程的步数
  real*8::x0(2),x1(2),xt(2)
  open(unit=11,file='result.txt')
  open(unit=12,file='newton.txt')
  x0=(/1d0,0d0/)
  !直接调用牛顿法，则计算失败不收敛,可以保留此语句
  !而注释下面两行语句，进行验证
  !--------
  !call newton(x0,n,1d-8)
  !--------
  !step 为微分方程步数
  ! 通过参数微分法计算的中间结果放在文件 result.txt 中
  ! 最后终值放在参数 xt 中回代出来，后面作为牛顿法初值
  call solve(x0,n,step,xt)
  !把参数微分法计算结果作为牛顿法初值，计算精度要求达到的-8 次方
  !计算结果放在文件 newton.txt 中
  call newton(xt,n,1d-8)
  end program main
```

6. 实验结论

由参数微分法提供的初值保存在文件 result.txt 中，结果图 6-8 所示。

在获得初值后，利用参数微分法获得的初值进行牛顿迭代，计算结果保存在 newton.txt

文件中。结果如图 6-9 所示。

参数微分法获取初值序列

1	0.9205311130	0.3410622260
2	0.8336428488	0.4380004385
3	0.7336618444	0.5299907158
4	0.6188253608	0.6229698900
5	0.4841321122	0.7217343928
6	0.3189711407	0.8340610006
7	0.0955542544	0.9784336725

图 6-8　参数微分法获得初值序列

由参数微分提供初值后牛顿法计算序列

1	0.011035999	0.992978458
2	0.000185281	0.999882296
3	0.000000010	0.999999966

图 6-9　牛顿法再次微分改正序列

可以看到牛顿法再迭代三次以后已经达到较高的精度。

本章小结

本章介绍了常见的非线性方程组的数值方法。如本章引言所述，非线性方程组的一些理论依然没有完善。尤其是对于大范围收敛性问题，一直是困扰数学家的一个难题，目前还没有一般性的方法彻底解决这个问题，甚至有人预言在即便是在将来，采用传统方法这依然是一个相当困难的问题的。本章最后两小节是数学家采取的一些扩大收敛域的方法，但依然没有完全解决这个问题。幸而，近年来一些新兴的智能算法可以在一定程度上弥补以上的缺陷，因而受到部分计算数学家的关注和期望。

第 7 章

插值法

插值法属函数逼近范畴，是数值分析中一个基础性的内容。数值微分、数值积分、微分方程数值解法等理论都需要具备插值法基础。给定一组基函数，函数值用基函数的线性组合表示，在 Hilbert 空间下给定各基函数的坐标后，如果函数值完全等于 Hilbert 空间下的向量值，则称为插值问题。使用较多也相对简单的基函数为多项式函数，当然根据需要也会选择其他的一些基函数如样条函数、连分式等。

7.1 拉格朗日插值

1. 实验基本原理

拉格朗日插值方法是比较基础的方法，方法本身比较容易实现，而且效果还不错，也容易理解。

通过平面上不同两点可以确定一条直线经过这两点，这就是拉格朗日线性插值问题，对于不在同一条直线的三个点得到的插值多项式为抛物线。

这里给出一般的插值公式，拉格朗日插值的基多项式为：

$$l_i(x)=\prod_{\substack{j=0\\ j\neq i}}^{n}\frac{x-x_i}{x_i-x_j},i=0,1,2,\cdots,n$$

对于没有接触过插值方法的读者，一个比较好的快速熟悉拉格朗日插值方法是把上面的多项式展开一下，看看如何计算基函数。

有了基函数以后就可以直接构造插值多项式，插值多项式为：

$$p_n=\sum_{i=0}^{n}f(x_i)l_i(x)$$

虽然拉格朗日插值法较为基础，但因算法实现简单，且在一定条件下精度能得到保证，故而很多实际应用问题就采用该方法实现插值。

2. 实验目的与要求

- 理解拉格朗日插值多项式方法。
- 会构造插值基函数。
- 能够编程实现拉格朗日多项式插值法。
- 能用插值多项式解决一些应用问题。

3. 实验内容与数据来源

已经知道 $\cos 30^\circ=\frac{\sqrt{3}}{2}$，$\cos 45^\circ=\frac{\sqrt{2}}{2}$，$\cos 60^\circ=\frac{1}{2}$，cos，使用拉格朗日插值法计算 $\cos 47^\circ$，$\cos 53^\circ$，c的数值。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(M)	输出插值点的结果
输入参变量	**数据类型**	**变量说明**
X	REAL*8(N)	节点自变量
Y	REAL*8(N)	节点函数值
N	INTEGER	节点个数
M	INTEGER	要插值点的个数
T	REAL*8(N)	插值节点

5. 程序代码

```
module lagrange
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   拉格朗日插值法模块
!
!-----------------------------------------------------
!  Contains    :
!      1.  solve 方法函数
!      2.   lag  单点插值
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
```

```
contains
subroutine solve(n,x,y,m,t,ty)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  拉格朗日插值函数，可以直接针对向量插值
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.   n  节点个数
!       2.   x 节点自变量值
!       3.   y 节点因变量值
!       4.   m  要插值点的个数
!       5.   t 要插值点的值 为向量
!  Output parameters  :
!       1.
!       2.   ty 插值点的结果 为向量
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,m
integer:: i
real*8::x(n),y(n),t(m),ty(m)
do i=1,m
  call la(n,x,y,t(i),ty(i))
end do
end subroutine solve
subroutine  la(n,x,y,t,ty)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.11
!-----------------------------------------------------
!  Purpose   :  拉格朗日单点插值，为 solve 所调用
!
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,i,k
!输入序列 x,y  为 N 维向量
!要插值的点 t
!t 对应的插值结果 ty
real*8::x(n),y(n),a(n)
!a(i)为各项
do i=1,n
```

```
    a(i)=y(i)
    do k=1,n
       if (k==i) cycle
       a(i)=a(i)*(t-x(k))/(x(i)-x(k))
    end do
end do
ty=0
do i=1,n
  ty=ty+a(i)
end do
end subroutine la
end module lagrange
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   : 拉格朗日插值法主函数
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
use lagrange
implicit real*8(a-z)
integer::N=4
real*8::x0(4),x(4),y(4),t0(3),t(3),ty(3),tyreal(3)
real*8,parameter::pi=3.141592653589793d0
open(unit=11,file='result.txt')
!插值节点及其函数值，化为弧度
x0=(/30,45,60,90/)
x=x0*pi/180
y=(/dsqrt(3d0)/2,dsqrt(2d0)/2,1d0/2,0d0/)
!待计算的点，化为弧度
t0=(/47,53,79/)
t=t0*pi/180
call solve(4,x,y,3,t,ty)
!调用系统逐元函数
tyreal=dcos(t)
write(11,101)
!输出标题
write(11,102)t0
!输出角度
write(11,103)ty
!输出插值结果
write(11,104)tyreal
```

```
    !输出实际函数值
    101 format(/,T22,'拉格朗日插值法',/)
    102 format(T3,'角度：',T16,3F12.5)
    103 format(T3,'插值结果：',T16,3F12.6)
    104 format(T3,'实际结果:',T16,3F12.6)
    end program main
```

6. 实验结论

计算结果保存在文件 result.txt 文件中，如图 7-1 所示。

```

                    拉格朗日插值法

  角度:            47.00000    53.00000    79.00000
  插值结果:         0.681956    0.601712    0.191458
  实际结果:         0.681998    0.601815    0.190809

```

图 7-1　拉格朗日插值结果

可以看出在数据比较稀疏而且较少的情况下，插值结果与实际结果已经符合的很好了。

7.2　牛顿插值法

1. 实验基本原理

拉格朗日插值多项式是一种很优秀的方法，其理论在很多方面都有应用。然而，如果增加一个插值基点，原先计算的插值多项式 $p_n(x)$ 对 $p_{n+1}(x)$ 没有用，这样必然增加计算工作量，尤其在没有计算机的情况下这个问题更加突出。我们期望增加插值基点时原先的计算结果对后面的计算仍然有用，本实验的牛顿插值方法具备这样的特点。在介绍牛顿插值之前先要了解一下差商的基本概念。

函数 $f(x)$ 的差商定义为：

$$
\begin{aligned}
&f[x_k]=f(x_k)\\
&f[x_{k-1},x_k]=\frac{f[x_k]-f[x_{k-1}]}{x_{k-1}-x_k}\\
&f[x_{k-2},x_{k-1},x_k]=\frac{f[x_{k-1},x_k]-f[x_{k-2},x_{k-1}]}{x_{k-2}-x_k}\\
&\qquad\vdots\\
&f[x_{k-j},x_{k-j+1},\cdots,x_k]=\frac{f[x_{k-j+1},\cdots,x_k]-f[x_{k-j},\cdots,x_{k-1}]}{x_k-x_{k-j}}
\end{aligned}
$$

有了上面的准备，便可以给出牛顿插值多项式为：

$$N_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2] \\ (x - x_0)(x - x_1) + \cdots + f[x_0, x_1, \cdots, x_n] \\ (x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

如果记 $w_k = (x - x_0)(x - x_1)\cdots(x - x_{k-1}), k = 1, 2, \cdots, n$，则牛顿插值可以表达为：

$$N_n(x) = f[x_0] + f[x_0, x_1]w_1 + f[x_0, x_1, x_2]w_2 \\ + \cdots + f[x_0, x_1, \cdots, x_n]w_n$$

有了上面的牛顿插值公式，就可以对给定的数据处理了，实验中的程序设计正是围绕这个公式展开的。

2. 实验目的与要求

- 熟悉差商、均差的基本概念。
- 比较牛顿插值方法与拉格朗日插值方法。
- 能够正确构造牛顿插值公式。
- 能够把牛顿插值方法用计算机程序实现。
- 要求程序中可以同时实现对多数据的一次处理。而不是只对单个数据处理。

3. 实验内容与数据来源

有离散数据

k	x_k	$f(x_k)$
1	1	0
2	2	-5
3	3	-6
4	4	3

用牛顿插值法计算 $f(2), f(3)$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(M)	要计算的插值结果
输入参变量	**数据类型**	**变量说明**
N	INTEGER	节点个数
M	INTEGER	要计算的节点个数
X	REAL*8(N)	节点向量
Y	REAL*8(N)	节点函数值
T	REAL*8(M)	要计算的节点向量

5. 程序代码

```
module  newton
!-----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   牛顿插值方法模块
!
!-----------------------------------------------------
!  Post Script :
!      1.   提供两个函数，一个是单点插值，一个是 solve 函数
!           注意两者的 N 是不一样的。
!           在单点插值中 N 是节点的个数减
!           为方便使用，在 solve 中 N 就直接指节点个数
!-----------------------------------------------------
contains
subroutine  solve(n,x,y,m,t,ty)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  方法主函数
!               可直接对多点插值
!-----------------------------------------------------
!  Input  parameters  :
!       1.  n  节点个数
!       2.  x 节点自变量值 N 维向量
!       3.  y 节点因变量值  N 维向量
!       4.  m  要计算点的个数
!       5.  t  要计算的点  M 向量
!  Output parameters  :
!       1.  ty  计算结果，为 M 维向量
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.   注意这里的 N  就直接指节点的个数
!       2.   而子函数 new 中 N 是指节点个数减 1
!----------------------------------------------------
implicit real*8(a-z)
integer::n,m
integer::i,j,k
real*8::x(n),y(n),t(m),ty(m)
do i=1,m
call new(n-1,x,y,t(i),ty(i))
end do
end subroutine solve
subroutine new(n,x,y,t,ty)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
```

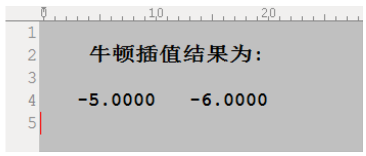
```
!------------------------------------------------------
!  Purpose   :  单点插值程序
!              此函数目的为 solve 所调用
!------------------------------------------------------
!  Input  parameters  :
!       1.
!       2.   t 标量
!  Output parameters  :
!       1.
!       2.    ty 标量
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.    N 是节点个数减 1，比如共个节点 则 N=3
!       2.
!       3.   详细算法可以参看《数值计算方法》
!            南京大学林成森编 科学出版社
!----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i,j,k
real*8::x(0:n),y(0:n)
real*8::b(n+1)
real*8::Q(0:n,0:n)
do i=0,n
Q(i,0)=y(i)
end do
do i=1,n
  do j=1,i
   Q(i,j)=(Q(i,j-1)-Q(i-1,j-1))/(x(i)-x(i-j))
  end do
end do
b(n+1)=Q(n,n)
do k=n,1,-1
  b(k)=Q(k-1,k-1)+b(k+1)*(t-x(k-1))
end do
ty=b(1)
end subroutine new
end module newton
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!------------------------------------------------------
!  Purpose   :  牛顿插值法主函数
!               solve 函数可以直接对向量插值
!------------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.
!       2.
!------------------------------------------------------
!  Post Script :
!       1.
```

```
!       2.
!-----------------------------------------------------
use newton
implicit real*8(a-z)
integer::N=4,M=2
real*8::x(4),y(4),t(2),ty(2)
open(unit=11,file='result.txt')
!插值节点极其函数值
x=(/1d0,2d0,3d0,4d0/)
y=(/0d0,-5d0,-6d0,3d0/)
!要计算的点
t=(/2d0,3d0/)
call solve(4,x,y,2,t,ty)
write(11,101)
write(11,102)ty
101 format(/,T5,'牛顿插值结果为:')
102 format(/,2F10.4)
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，打开该文件可以看到结果如图 7-2 所示。

与上一节一样，这里给出的插值方法可以输入向量，计算结果亦为这些向量的插值结果。



牛顿插值结果为:

-5.0000   -6.0000

图 7-2　牛顿插值法计算结果

7.3　Hermite插值

1. 实验基本原理

前面几个实验中介绍的插值公式都要求插值多项式在插值点出的取值和函数值相等。有些实际问题，不仅要求插值多项式在插值点与函数值相同，而且还要求导数相同，这类问题就是 Hermit 插值问题。

如果 $f(x)$ 在区间[a,b]上连续可以导，$x_0,x_1,\cdots x_n \in [a,b]$ 是互异的，那么存在唯一的多项式 $H_{2n+1}(x)$ 满足多项式在这些点上的值与函数 $f(x)$ 的值相等、多项式在这些点的一阶导数值与函数的一阶导数值相等。

这个多项式可以表示为

$$H_{2n+1}(x)=\sum_{i=0}^{n} f(x_i)[1-2(x-x_i)l_i'(x_i)]l_i^2(x)$$
$$+\sum_{i=0}^{n} f'(x_i)(x-x_i)l_i^2(x)$$

其中

$$l_i(x)=\prod_{\substack{j=0\\j\neq i}}^{n}\frac{(x-x_i)}{(x_i-x_j)},i=0,1,\cdots,n$$

$$l_i'(x_i)=\sum_{\substack{j=0\\j\neq i}}^{n}\frac{1}{x_i-x_j},i=0,1,\cdots,n$$

这就是 Hermite 插值的基本公式。

2. 实验目的与要求

- 了解 Hermite 插值的适用条件。
- 理解插值公式的计算流程。
- 会使用作者编写的函数解决 Hermite 插值问题。
- 最好能自行编程实现 Hermite 插值方法。

3. 实验内容与数据来源

已知函数 $f(x)$ 在以下各点处的函数值与导数值如下：

k	x_k	$f(x_k)$	$f'(x_k)$
1	1.3	0.6200860	-0.5220232
2	1.6	0.4554022	-0.5698959
3	1.9	0.2818186	-0.5811571

用 Hermite 插值求 $f(1.5), f(1.7)$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(M)	输出的插值结果
输入参变量	**数据类型**	**变量说明**
N	INTEGER	节点个数
X	REAL*8(N)	节点向量
Y	REAL*8(N)	节点函数值
DY	REAL*8(N)	节点导数值
T	REAL*8(M)	要计算的节点
M	INTEGER	要计算的节点个数

5. 程序代码

```
module Herm
!-----------------------------------------module coment
```

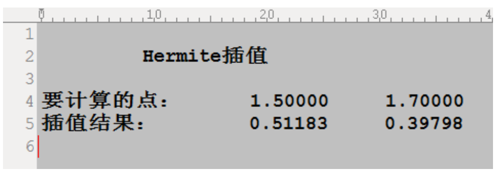
```
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :
!-----------------------------------------------------
!  Description :    Hermite 插值模块
!                 提供两个函数
!                 solve 函数为方法函数
!                 single 函数为单点插值函数
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
contains
subroutine solve(n,x,y,dy,m,t,ty)
!---------------------------------subroutine  comment
!  Version    :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose    :   Hermite 插值方法函数
!               可直接对向量插值
!-----------------------------------------------------
!  Input  parameters  :
!       1.   n 节点个数
!       2.   x  节点向量 N 维
!       3.   y  节点函数值  N 维
!       4.   dy  节点导数值 N 维
!       5.   m  需要插值向量维数
!       6    t  需要插值的点  M 维
!  Output parameters  :
!       1.   ty  插值结果 M 维 -----与 t 对应
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.      需要注意的是这里的输入参数 N  即表示节点个数
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,m
integer::i,j,k
real*8::x(n),y(n),dy(n),t(m),ty(m)
do i=1,m
call single(n-1,x,y,dy,t(i),ty(i))
end do
end subroutine solve
subroutine  single(n,x,y,dy,t,ty)
!---------------------------------subroutine  comment
!  Version    :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose    :  单点 Hermite 插值
```

```
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.   需要注意：输入参数 N 表示节点数减
!      2.      如个节点，则 N=5
!           t,ty 为标量
!----------------------------------------------------
implicit real*8(a-z)
integer::N
integer::i,j
real*8::x(0:n),y(0:n),dy(0:n)
real*8::l(0:n),dl(0:n)
!---------这段程序求得 l
do i=0,n
   l(i)=1
   do j=0,n
     if (j==i) cycle
    l(i)=l(i)*(t-x(j))/(x(i)-x(j))
   end do
end do
!-------这段程序求得 dl
do i=0,n
  dl(i)=0
  do j=0,n
    if (j==i) cycle
    dl(i)=dl(i)+1d0/(x(i)-x(j))
  end do
end do
ty=0
do i=0,n
!& 表示续行
  ty=ty+y(i)*(1d0-2d0*(t-x(i))*dl(i))*l(i)**2  &
     +dy(i)*(t-x(i))*l(i)**2
end do
end subroutine single
end module Herm
program main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.11
!-----------------------------------------------------
!  Purpose   :  Hermite 插值主函数
!              可直接对向量插值
!
!-----------------------------------------------------
```

```
!  Post Script :
!       1.   计算结果保存在文件 result.txt 中
!       2.
!-----------------------------------------------------
use Herm
implicit real*8(a-z)
integer::N=3
real*8::x(3),y(3),dy(3),t(2),ty(2)
open(unit=11,file='result.txt')
x=(/1.3d0,1.6d0,1.9d0/)
y=(/0.620086d0,0.4554022d0,0.2818186d0/)
dy=(/-0.5220232d0,-0.5698959d0,-0.5811571d0/)
t=(/1.5d0,1.7d0/)
call solve(3,x,y,dy,2,t,ty)
write(11,101)
write(11,102)t
write(11,103)ty
101 format(/,T10,'Hermite 插值',/)
102 format('要计算的点：',T14,2F12.5)
103 format('插值结果：',T14,2F12.5)
end
```

6. 实验结论

程序运行后，计算结果保存在文件 result.txt 文件中，打开该文件如图 7-3 所示。



```

         Hermite插值

要计算的点：      1.50000     1.70000
插值结果：        0.51183     0.39798
```

图 7-3　Hermite 插值计算结果

Hermite 插值也是一种常见的插值方法，但是需要提供插值基点的导数值。

7.4　三次样条插值之固支条件

1. 实验基本原理

在具有收敛性与稳定性的插值函数中，最重要也最常用的是样条函数插值。样条函数给出的光滑插值曲线或曲面在飞机、汽车、船舶、精密机械等方面有相当广泛的应用。在函数逼近、数值积分、微分方程数值方法等计算数学领域中，样条函数也是最重要的工具之一。

设 $[a,b]$ 上一个划分 $\Delta: a=x_0<x_1<\ldots<x_n=b$，如果函数 $s(x)$ 满足条件

（1） $s(x)\in C^{m-1}[a,b]$；

（2） $s(x)$ 在每个子区间 $[x_{i-1},x_i], i=1,2,\ldots,n$ 上是 m 次代数多项式；

则称 $s(x)$ 为关于节点划分 Δ 的 m 次样条多项式。

实际应用中，较为常用的是三次样条插值，三次样条插值不同的边界条件构成了不同的插值问题，这一节介绍固支条件下插值方法。固支条件也称为第一类边界条件即

$$s'(x_0)=f'(x_0),s'(x_n)=f'(x_n)$$

求解三次样条插值有多种方法，这里介绍三弯矩方法。

对于划分 Δ，记子区间 $[x_{i-1},x_i]$ 的长度

$$h_i=x_{i+1}-x_i,i=0,1,2,\dots,n-1$$

三次样条插值可以构造多项式

$$s(x)=M_i\frac{(x_{i+1}-x)^3}{6h_i}+M_{i+1}\frac{(x-x_i)^3}{6h_i}$$
$$+\left[f(x_i)-\frac{M_ih_i^2}{6}\right]\frac{x_{i+1}-x}{h_i}+\left[f(x_{i+1})-\frac{M_{i+1}h_i^2}{6}\right]\frac{x-x_i}{h_i}$$

$$x\in[x_i,x_{i+1}],i=0,1,\dots,n-1$$

进而可以通过以下步骤求解插值点的函数值 $y_i=f(x_i),i=0,1,2,\dots,n$ 。

计算参数

$$\begin{cases}h_i=x_{i+1}-x_i,\\ f[x_i,x_{i+1}]=\dfrac{y_{i+1}-y_i}{h_i},i=0,1,,n-1\end{cases}$$

$$\begin{cases}\mu_i=\dfrac{h_{i-1}}{h_{i-1}+h_i},\lambda_i=1-\mu_i\\ f[x_{i-1},x_i,x_{i+1}]=\dfrac{f[x_{i-1},x_i]-f[x_i,x_{i+1}]}{x_{i-1}-x_{i+1}},i=1,2,\dots,n-1\\ d_i=6f[x_{i-1},x_i,x_{i+1}]\end{cases}$$

设置固支边界条件相关参数

$$\begin{cases}d_0=\dfrac{6}{h_0}\left(f[x_0,x_1]-f'(x_0)\right)\\ d_n=\dfrac{6}{h_{n-1}}\left(f'(x_n)-f[x_{n-1},x_n]\right)\end{cases}$$

求解线性方程组

$$
\begin{pmatrix}
2 & 1 & & & \\
\mu_1 & 2 & \lambda_1 & & \\
 & \ddots & \ddots & \ddots & \\
 & & \mu_{n-1} & 2 & \lambda_{n-1} \\
 & & & 1 & 2
\end{pmatrix}
\begin{pmatrix}
M_0 \\ M_1 \\ \vdots \\ M_{n-1} \\ M_n
\end{pmatrix}
=
\begin{pmatrix}
d_0 \\ d_1 \\ \vdots \\ d_{n-1} \\ d_n
\end{pmatrix}
$$

把求解步骤 3 中的方程得到$\{M_i\}_{i=0}^{n}$，带入三次样条多项式。即得到函数逼近式，如果对于要计算的点，先搜索其在整个区间中的位置，然后用逼近式给出该点的插值结果。

实际上，我们在程序设计时，也是对要插值点的个各分量分别搜索，然后给出插值结果的向量。

2. 实验目的与要求

- 了解样条函数的重要性。
- 大致了解样条插值的原理。
- 了解固支条件下三次样条插值方法。
- 会用作者编写的程序进行相关问题处理。

3. 实验内容与数据来源

已知离散点列如下

i	x_i	$f(x_i)$
0	-2	-0.9093
1	-1.5	-0.9975
2	-1	-0.8415
3	-0.5	-0.4794
4	0	0
5	0.5	0.4794
6	1	0.8415
7	1.5	0.9975
8	2	0.9093

已知 $f'(-2)=-0.4161$，$f'(2)=-0$。试用三次样条方法对以上数据插值，并求 $\mathbf{x}=(0.4\quad -0.6\quad 1.7\quad 0.8\quad 1.8)^T$ 处的值。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(1:NT)	要计算的函数值

（续表）

输入参变量	数据类型	变量说明
N	INTEGER	插值节点个数减 1
X	REAL*8(0:N)	节点自变量
Y	REAL*8(0:N)	节点因变量
NT	INTEGER	要计算的向量的维数
T	REAL*8(1:NT)	要计算的节点向量
DA	REAL*8	起点处导数
DB	REAL*8	终点处导数

5. 程序代码

```
module  spline
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010.05.12
!-----------------------------------------------------
!  Description :   三次样条插值之固支条件模块
!
!-----------------------------------------------------
!  Contains    :
!      1.     solve 函数即方法函数
!      2.
!      3.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.     可以直接对向量插值
!-----------------------------------------------------
contains
subroutine solve(n,x,y,da,db,nt,t,ty)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.12
!-----------------------------------------------------
!  Purpose   :   三次样条之固支条件
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.   n-----插值节点个数减，如有九个节点则 N=8
!       2.   x ---节点自变量 为（: N）维向量
!       3.   y----节点因变量 （: N）维向量
!       4.   nt 要计算向量的维数
!       5.   t 要计算的向量 (1:nt) 维向量
!       6.   da  ------起点处导数 f'(x(0))
!       7.   db -------终点处导数 f'(x(0))
!  Output parameters  :
!       1.   ty ---要计算的向量，（: nt）维
!
!  Common parameters  :
```

```
!
!-----------------------------------------------------
!  Post Script :
!          算法可以参考施妙根，顾丽珍编写的科学计算
!          对于要插值的向量分量，可以不必按大小排列
!-----------------------------------------------------
implicit real*8(a-z)
integer::n,nt
!n 为插值节点数减，即如果有个节点，则 n=8
!nt 要插值点的个数，及向量 t,ty 的维数
integer::i,j,k
real*8::x(0:n),y(0:n)
real*8::t(nt),ty(nt)
real*8::h(0:n-1)
real*8::f1(0:n-1),f2(1:n-1)
real*8::u(1:n-1),namda(1:n-1),d(0:n)
real*8::M(0:n)
real*8::A(0:n,0:n)
do i=0,n-1
  h(i)=x(i+1)-x(i)
  f1(i)=(y(i+1)-y(i))/h(i)
end do
!对固支边界条件而言  设置 d(0) 与 d(n)
d(0)=6d0/h(0)*(f1(0)-da)
d(n)=6d0/h(n-1)*(db-f1(n-1))
!求得 u, namda, d
do i=1,n-1
u(i)=h(i-1)/(h(i-1)+h(i))
 namda(i)=1-u(i)
f2(i)=(f1(i-1)-f1(i))/(x(i-1)-x(i+1))
  d(i)=6d0*f2(i)
end do
!设置 A 矩阵值
A=0
do i=1,n-1
a(i,i)=2d0
end do
do i=2,n-1
a(i,i-1)=u(i)
end do
do i=1,n-2
a(i,i+1)=namda(i)
end do
!-------相比于自然条件，这里需要设置 A 矩阵首末行元素，其他相同
!设置 A 矩阵的首行元素
a(0,0)=2d0
a(0,1)=1d0
!设置 A 矩阵元素的末行元素
a(n,n-1)=1d0
a(n,n)=2d0
! 设置右向量值
d(1)=d(1)-u(1)*M(0)
d(n-1)=d(n-1)-namda(n-1)*M(n)
call gauss(a,d,M,N+1)
!--------以上以及求得系数
```

```
!已经完成插值多项式的建立
!----------------------------------------------
! 以下开始计算具体值
do k=1,nt
!------------
!  对要插值向量每个分量而言，先找到其在数据中的位置
do i=1,n-1
 if (t(k)<x(i+1)) exit
end do
  ty(k)=M(i)*(x(i+1)-t(k))**3/6d0/h(i)+ M(i+1)*(t(k)-x(i))**3/6d0/h(i)+ &
     (y(i)-M(i)*h(i)**2/6d0)*(x(i+1)-t(k))/h(i)+ &
     (y(i+1)-M(i+1)*h(i)**2/6d0)*(t(k)-x(i))/h(i)
end do
!------------------------------
!为了方便读者比对结果，这里输出中间值，实际应用时可以去掉输出部分
!!-------------------------
!write(11,101)
!write(11,102)((i,h(i),m(i)),i=0,n-1)   !输出 h
!
!
!write(11,103)
!write(11,104)((i,u(i),namda(i),d(i)),i=1,n-1)
!
!101 format(/,T5,'        h           m')
!102 format(I3,2F16.8)
!
!103 format(/,T5,'       u           namda          d')
!104 format(I3,3F16.8)
!!--------------------------------------------------------------
end subroutine solve
subroutine gauss(A,b,x,N)
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!----------------------------------------------------
!  Purpose   :  高斯列主元消去法
!               Ax=b
!----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N 方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!---------------------------------------------------
implicit real*8(a-z)
integer::i,k,N
integer::id_max  !主元素标号
real*8::A(N,N),b(N),x(N)
real*8::Aup(N,N),bup(N)
!Ab 为增广矩阵 [Ab]
```

```
real*8::Ab(N,N+1)
real*8::vtemp1(N+1),vtemp2(N+1)
Ab(1:N,1:N)=A
Ab(:,N+1)=b
!###########################################################
!  这段是列主元消去法的核心部分
do k=1,N-1
    elmax=dabs(Ab(k,k))
    id_max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
      if (dabs(Ab(i,k))>elmax) then
         elmax=Ab(i,k)
         id_max=i
      end if
    end do
!至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id_max,:)
    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!##########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!------------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!           | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!           | 0  0  *  *  # |
!           | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine gauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
```

```
!       3.   N 方程维数
!  Output parameters  :
!       1.  x  方程的根
!       2.
!  Common parameters  :
!
!----------------------------------------------------
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module spline
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  固定支边界条件下的三次样条插值
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.
!       2.   reslut.txt 文件保存了计算结果
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
use spline
implicit real*8(a-z)
integer::i,j
real*8::x(0:8),y(0:8),t(5),ty(5),simty(5)
open(unit=11,file='result.txt')
!插值基点
x=(/-2d0 ,-1.5d0 ,-1d0  ,  -0.5d0,  0d0  ,  0.5d0  ,  1d0  ,  1.5d0,2.0d0/)
y=(/ -0.9093d0,-0.9975d0,-0.8415d0, -0.4794d0,0d0,0.4794d0,0.8415d0,
0.9975d0,0.9093d0/)
!第一类边界条件，即一次导数边界条件
da= -0.4161d0
db= -0.4161d0
!要进行计算的点 ------不需要按照大小排列
t=(/0.4d0,-0.6d0,1.7d0,0.8d0,1.8d0/)
```

```
    !调用方法函数
    call solve(8,x,y,da,db,5,t,ty)
    !仿真函数值
    simty=dsin(t)
    write(11,101)
    write(11,102)
    write(11,103)((i,t(i),ty(i),simty(i)),i=1,5)
    101  format(/,T9,'三次样条插值之固支条件',/)
    102  format('  序列    插值点       插值结果       真值')
    103  format(I3,3F16.8)
    end program main
```

6. 实验结论

计算结果保存在文件 result.txt 文件中，内容如图 7-4 所示。

三次样条插值之固支条件

序列	插值点	插值结果	真值
1	0.40000000	0.38958497	0.38941834
2	-0.60000000	-0.56423331	-0.56464247
3	1.70000000	0.99375613	0.99166481
4	0.80000000	0.71643249	0.71735609
5	1.80000000	0.97519209	0.97384763

图 7-4　三次样条插值之固支条件插值结果

实际上原题数据来源于仿真函数 $y=\sin x$ 在区间 $[-2,2]$ 上的值，通过与真值比较可以发现计算精度还是不错的。

7.5　三次样条插值之自然边界条件

1. 实验基本原理

三次样条的第二类边界条件为

$$s''(x_0)=f''(x_0),s''(x_n)=f''(x_n)$$

尤其是如果

$$f''(x_0)=f''(x_n)=0$$

称为自然边界条件，可见自然边界条件是第二类边界条件的特殊情况。不过在程序设计时，我们依然给的是一般意义下的第二类边界条件，如果是自然边界条件情况值需要将 $f''(x_0)$ 与 $f''(x_n)$ 分别设置为 0，这样程序更有一般性。

求解第二类边界条件插值问题的算法与固支条件下较为类似，其中第一步是相同的，其步骤如下：

01 计算参数

$$\begin{cases} h_i = x_{i+1} - x_i, \\ f\left[x_i, x_{i+1}\right] = \dfrac{y_{i+1} - y_i}{h_i}, i = 0,1,,n-1 \end{cases}$$

$$\begin{cases} \mu_i = \dfrac{h_{i-1}}{h_{i-1} + h_i}, \lambda_i = 1 - \mu_i \\ f\left[x_{i-1}, x_i, x_{i+1}\right] = \dfrac{f\left[x_{i-1}, x_i\right] - f\left[x_i, x_{i+1}\right]}{x_{i-1} - x_{i+1}}, i = 1,2,\ldots,n-1 \\ d_i = 6f\left[x_{i-1}, x_i, x_{i+1}\right] \end{cases}$$

02 设置第二类边界条件相关参数

$$\begin{cases} M_0 = f''(x_0) \\ M_n = f''(x_n) \end{cases}$$

03 采用第 1 章中介绍的追赶法求解三对角线性方程组

$$\begin{pmatrix} 2 & \lambda_1 & & & \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-2} & 2 & \lambda_{n-2} \\ & & & \mu_{n-1} & 2 \end{pmatrix} \begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-2} \\ M_{n-1} \end{pmatrix} = \begin{pmatrix} d_1 - \mu_1 M_0 \\ d_2 \\ \vdots \\ d_{n-2} \\ d_{n-1} - \lambda_{n-1} M_n \end{pmatrix}$$

04 把求解**03**中的方程得到$\{M_i\}_{i=0}^n$，带入三次样条多项式。即得到函数逼近式，如果对于要计算的点，先搜索其在整个区间中的位置，然后用逼近式给出该点的插值结果。

可以看到第二类边界条件解算的方程组的维数与固支条件下是不同的，其中M_0与M_n事先就给定了。

2. 实验目的与要求

- 知道三次样条插值的第二类边界条件。
- 能看懂算法的大致流程。
- 会使用作者编写的函数进行相关数据处理。

3. 实验内容与数据来源

已知以下离散数据

i	x_i	$f(x_i)$
0	1	3.0
1	2	3.7
2	3	3.9
3	6	4.2
4	7	5.7
5	8	6.6
6	10	7.1
7	13	6.7
8	17	4.5

要求在自然边界条件采用三次样条插值方法计算向量

$$\mathbf{x}=\begin{pmatrix}3.5 & 1.5 & 5.5 & 6.5 & 7.5 & 9.0 & 11.5 & 15.0\end{pmatrix}^T$$

处的值。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(1:NT)	要计算的函数值
N	INTEGER	插值节点个数减 1
X	REAL*8(0:N)	节点自变量
Y	REAL*8(0:N)	节点因变量
NT	INTEGER	要计算的向量的维数
T	REAL*8(1:NT)	要计算的节点向量
D2FA	REAL*8	起点二阶导数
D2FB	REAL*8	终点二阶处导数

5. 程序代码

```
module  spline
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010.05.12
!-----------------------------------------------------
!  Description :   三次样条插值之第二类边界条件模块
!
!-----------------------------------------------------
!  Contains    :
!      1.     solve 函数即方法函数
!      2.
!      3.     chase 为用追赶法计算线性方程，因为插值中的
!             方程为三对角方程，用追赶法计算效率较高。
!-----------------------------------------------------
!  Post Script :
!      1.
```

```
!     2.    可以直接对向量插值
!-----------------------------------------------------
contains
subroutine solve(n,x,y,d2fa,d2fb,nt,t,ty)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.12
!-----------------------------------------------------
!  Purpose   :   三次样条之第二类边界条件
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.   n-----插值节点个数减，如有九个节点则 N=8
!      2.   x ---节点自变量 为（: N）维向量
!      3.   y----节点因变量 （: N）维向量
!      4.   nt 要计算向量的维数
!      5.   t 要计算的向量 (1:nt) 维向量
!      6.
!      7.    d2fa,d2fb  起点于终点处的二阶导数条件，
!             如果是自然边界条件，则二者为皆为
!  Output parameters  :
!      1.   ty ---要计算的向量，（: nt）维
!
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!
!      1.    自然边界条件是第一类边界条件的特殊情况
!            本程序可以直接处理第一类边界条件
!      2.
!            对于要插值的向量分量，可以不必按大小排列
!----------------------------------------------------
implicit real*8(a-z)
integer::n,nt
!n 为插值节点数减，即如果有个节点，则 n=8
!nt 要插值点的个数，及向量 t,ty 的维数
integer::i,j,k
real*8::x(0:n),y(0:n)
real*8::t(nt),ty(nt)
real*8::h(0:n-1)
real*8::f1(0:n-1),f2(1:n-1)
real*8::u(1:n-1),namda(1:n-1),d(1:n-1)
real*8::M(0:n),v(1:n-1)
real*8::A(1:n-1,1:n-1)
M(0)=d2fa
M(n)=d2fb
do i=0,n-1
  h(i)=x(i+1)-x(i)
  f1(i)=(y(i+1)-y(i))/h(i)
end do
!求得 u, namda, d
do i=1,n-1
 u(i)=h(i-1)/(h(i-1)+h(i))
```

```
 namda(i)=1-u(i)
f2(i)=(f1(i-1)-f1(i))/(x(i-1)-x(i+1))
  d(i)=6d0*f2(i)
end do
!设置 A 矩阵值
A=0
do i=1,n-1
a(i,i)=2d0
end do
do i=2,n-1
a(i,i-1)=u(i)
end do
do i=1,n-2
a(i,i+1)=namda(i)
end do
! 设置右向量值
d(1)=d(1)-u(1)*M(0)
d(n-1)=d(n-1)-namda(n-1)*M(n)
call chase(a,d,v,N-1)
do i=1,n-1
   M(i)=v(i)
end do
!--------以上以及求得系数
!已经完成插值多项式的建立
!------------------------------------------------
! 以下开始计算具体值
do k=1,nt
!------------
!  对要插值向量每个分量而言，先找到其在数据中的位置
do i=1,n-1
 if (t(k)<x(i+1)) exit
end do
  ty(k)=M(i)*(x(i+1)-t(k))**3/6d0/h(i)+ M(i+1)*(t(k)-x(i))**3/6d0/h(i)+ &
     (y(i)-M(i)*h(i)**2/6d0)*(x(i+1)-t(k))/h(i)+ &
     (y(i+1)-M(i+1)*h(i)**2/6d0)*(t(k)-x(i))/h(i)
end do
!-------------------------------
!为了方便读者比对结果，这里输出中间值，实际应用时可以去掉输出部分
!-------------------------
write(11,101)
write(11,102)((i,h(i),m(i)),i=0,n-1)   !输出 h
write(11,103)
write(11,104)((i,u(i),namda(i),d(i)),i=1,n-1)
101 format(/,T5,'         h           m')
102 format(I3,2F16.8)
103 format(/,T5,'        u            namda           d')
104 format(I3,3F16.8)
!----------------------------------------------------------------
end subroutine solve
 subroutine chase(A,f,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!-----------------------------------------------------
```

```
! Purpose   : 追赶法计算三对角方程组
!            Ax=f
!-------------------------------------------------------
! Input parameters  :
!       1.  A 系数矩阵
!       2.  f 右向量
! Output parameters  :
!       1.  x 方程的解
!       2.  N 维数
! Common parameters  :
!
!------------------------------------------------------
! Post Script :
!       1.   注意：该方法仅适用于三对角方程组
!       2.
!-----------------------------------------------------
implicit real*8(a-z)
 integer::N
 real*8::A(N,N),f(N),x(N)
 real*8::L(2:N),u(N),d(1:N-1)
real*8::c(1:N-1),b(N),e(2:N)
integer::i
real*8::y(N)
!----------------把 A 矩阵复制给向量 e,b,c
do i=1,N
   b(i)=a(i,i)
end do
do i=1,N-1
  c(i)=a(i,i+1)
end do
do i=2,N
 e(i)=a(i,i-1)
end do
!-------------------------
do i=1,N-1
 d(i)=c(i)
end do
u(1)=b(1)
do i=2,N
 L(i)=e(i)/u(i-1)
 u(i)=b(i)-L(i)*c(i-1)
end do
!------开始回带,求得 y
y(1)=f(1)
do i=2,N
 y(i)=f(i)-L(i)*y(i-1)
end do
!-----开始回带，求得 x
x(n)=y(n)/u(n)
do i=n-1,1,-1
x(i)=(y(i)-c(i)*x(i+1))/u(i)
end do
end subroutine chase
end module spline
program main
```

```
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  三次样条插值之自然边界条件
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
!       1.
!       2.
!-----------------------------------------------------
!  Post Script :
!       1.
!       2.
!-----------------------------------------------------
use spline
implicit  real*8(a-z)
integer::i
real*8::x(0:8),y(0:8)
real*8::t(8),ty(8)
open(unit=11,file='result.txt')
!插值节点，九个节点
x=(/1d0,2d0,5d0,6d0,7d0,8d0,10d0,13d0,17d0/)
!插值因变量
y=(/3d0,3.7d0,3.9d0,4.2d0,5.7d0,6.6d0,7.1d0,6.7d0,4.5d0/)
!要计算的点，可以不必按照大小排列
t=(/3.5d0,1.5d0,5.5d0,6.5d0,7.5d0,9.0d0,11.5d0,15d0/)
write(11,101)
!
!调用函数
call solve(8,x,y,0d0,0d0,8,t,ty)
! 作者提供的函数具有一般性，可以直接处理第一类边界条件，
!而如果是自然条件，则边界处的二阶导数为
write(11,102)
write(11,103)((i,t(i),ty(i)),i=1,8)
101 format(/,T5,'三次样条插值之自然边界条件')
102 format(/,T5,'       t             ty')
103 format(I3,2F16.8)
end program main
```

6. 实验结论

计算结果保存在 result.txt 文件中。打开该文件可以看到，插值结果。为了便于读者编程比对自己的程序，这里给出了计算中间数据。其中 h_i 、 m_i 见表 7-1。u_i 、 λ_i 、 d_i 见表 7-2。插值结果见表 7-3。

表 7-1　h_i 、 m_i 序列

i	h_i	m_i
0	1.00000000	0.00000000
1	3.00000000	-0.51422654
2	1.00000000	0.10460410
3	1.00000000	2.10584684
4	1.00000000	-1.32799146
5	2.00000000	-0.39388098
6	3.00000000	-0.10436132
7	4.00000000	-0.15620829

表 7-2　u_i 、 λ_i 、 d_i 序列

i	u_i	λ_i	d_i
1	0.25000000	0.75000000	-0.95000000
2	0.75000000	0.25000000	0.35000000
3	0.50000000	0.50000000	3.60000000
4	0.50000000	0.50000000	-1.80000000
5	0.33333333	0.66666667	-1.30000000
6	0.40000000	0.60000000	-0.46000000
7	0.42857143	0.57142857	-0.35714286

表 7-3　插值结果

i	x	$s(x)$
1	3.50000000	4.03041262
2	1.50000000	3.36712867
3	5.50000000	3.91184682
4	6.50000000	4.90138404
5	7.50000000	6.25761703
6	9.00000000	6.97456058
7	11.50000000	7.04657040
8	15.00000000	5.75620829

7.6　三次样条之周期边界条件

1. 实验基本原理

三次样条的周期边界条件为

$$\begin{cases} s'(x_0+0)=s'(x_n-0) \\ s''(x_0+0)=f''(x_n-0) \end{cases}$$

其中函数值的周期条件 $s(x_0+0)=s(x_n-0)$，由于已知 $f(x_0)=f(x_n)$ 确定。周期边界条

件亦称为第三类边界条件。三种类型的边界条件都有它们的实际背景及相应的物理意义，针对不同的情况。

求解周期边界条件插值问题的算法与前两节介绍的算法也较为类似，其中第一步是相同的，其步骤如下：

01 计算参数

$$\begin{cases} h_i = x_{i+1} - x_i, \\ f\left[x_i, x_{i+1}\right] = \dfrac{y_{i+1} - y_i}{h_i}, i = 0,1,,n-1 \end{cases}$$

$$\begin{cases} \mu_i = \dfrac{h_{i-1}}{h_{i-1} + h_i}, \lambda_i = 1 - \mu_i \\ f\left[x_{i-1}, x_i, x_{i+1}\right] = \dfrac{f\left[x_{i-1}, x_i\right] - f\left[x_i, x_{i+1}\right]}{x_{i-1} - x_{i+1}}, i = 1,2,\ldots,n-1 \\ d_i = 6f\left[x_{i-1}, x_i, x_{i+1}\right] \end{cases}$$

02 设置周期边界条件相关参数

$$\begin{cases} \mu_n = \dfrac{h_{n-1}}{h_0 + h_{n-1}} \\ \lambda_n = 1 - \mu_n \\ \tilde{d}_n = \dfrac{6}{h_0 + h_{n-1}}\left(f\left[x_0, x_1\right] - f\left[x_{n-1}, x_n\right]\right) \end{cases}$$

03 计算线性方程组（注意该方程不是三对角方程）。

$$\begin{pmatrix} 2 & \lambda_1 & & & \mu_1 \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-2} & 2 & \lambda_{n-2} \\ \lambda_n & & & \mu_{n-1} & 2 \end{pmatrix} \begin{pmatrix} M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ \tilde{d}_n \end{pmatrix}$$

04 把求解03中的方程得到$\{M_i\}_{i=0}^n$，带入三次样条多项式。即得到函数逼近式，如果对于要计算的点，先搜索其在整个区间中的位置，然后用逼近式给出该点的插值结果。

可以看到各类边界条件下直接求解方程的维数是不同的，但是多项式的形式是相同的，最后都将给出$\{M_i\}_{i=0}^n$，从而为计算插值结果提供前提条件。

2. 实验目的与要求

- 了解三次样条插值的周期边界条件。
- 能看懂周期边界条件下插值方法。
- 会使用作者编写的函数实现对相关问题的数据处理。

3. 实验内容与数据来源

已知离散数据点列如下

i	x_i	$f(x_i)$
0	0	0.6000
1	0.3000	0.8687
2	0.6000	1.0598
3	0.9000	1.1563
4	1.2000	1.1495
5	1.5000	1.0399
6	1.8000	0.8375
7	2.1000	0.5603
8	2.4000	0.2330
9	2.7000	-0.1151
10	3.0000	-0.4529
11	3.3000	-0.7502
12	3.6000	-0.9806
13	3.9000	-1.1233
14	4.2000	-1.1657
15	4.5000	-1.1040
16	4.8000	-0.9437
17	5.1000	-0.6990
18	5.4000	-0.3919
19	5.7000	-0.0499
20	6.0000	0.2967
21	6.2832	0.6000

已知数据为周期函数，要求对以上数据采用三次样条插值，计算

$$\mathbf{x} = \begin{pmatrix} 0.4 & 2.8 & 5.9 & 6.1 & 4.7 \end{pmatrix}^T$$

处的函数值。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(1:NT)	要计算的函数值

（续表）

输入参变量	数据类型	变量说明
N	INTEGER	插值节点个数减 1
X	REAL*8(0:N)	节点自变量
Y	REAL*8(0:N)	节点因变量
NT	INTEGER	要计算的向量的维数
T	REAL*8(1:NT)	要计算的节点向量

5. 程序代码

```
module  spline
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :  2010.05.12
!-----------------------------------------------------
!  Description :   三次样条插值之周期条件模块
!
!-----------------------------------------------------
!  Contains    :
!      1.     solve 函数即方法函数
!      2.
!      3.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.     可以直接对向量插值
!-----------------------------------------------------
contains
subroutine solve(n,x,y,nt,t,ty)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.12
!-----------------------------------------------------
!  Purpose   :   三次样条之周期条件
!
!-----------------------------------------------------
!  Input  parameters  :
!       1.   n-----插值节点个数减，如有九个节点则 N=8
!       2.   x ---节点自变量 为（: N）维向量
!       3.   y----节点因变量 （: N）维向量
!       4.   nt 要计算向量的维数
!       5.   t 要计算的向量 (1:nt) 维向量
!  Output parameters  :
!       1.   ty ---要计算的向量，（: nt）维
!
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!
```

```
!           对于要插值的向量分量，可以不必按大小排列
!----------------------------------------------------
implicit real*8(a-z)
integer::n,nt
!n 为插值节点数减，即如果有个节点，则 n=8
!nt 要插值点的个数，及向量 t,ty 的维数
integer::i,j,k
real*8::x(0:n),y(0:n)
real*8::t(nt),ty(nt)
real*8::h(0:n-1)
real*8::f1(0:n-1),f2(1:n-1)
real*8::u(1:n),namda(1:n),d(1:n)
real*8::M(1:n)
real*8::A(1:n,1:n)
do i=0,n-1
  h(i)=x(i+1)-x(i)
  f1(i)=(y(i+1)-y(i))/h(i)
end do
!求得 u, namda, d
do i=1,n-1
 u(i)=h(i-1)/(h(i-1)+h(i))
 namda(i)=1-u(i)
f2(i)=(f1(i-1)-f1(i))/(x(i-1)-x(i+1))
  d(i)=6d0*f2(i)
end do
!对于周期条件而言，系数设置
u(n)=h(n-1)/(h(0)+h(n-1))
namda(n)=1-u(n)
!右向量
d(n)=6d0/(h(0)+h(n-1))*(f1(0)-f1(n-1))
!设置 A 矩阵值
A=0
do i=1,n
a(i,i)=2d0
end do
do i=2,n
a(i,i-1)=u(i)
end do
do i=1,n-1
a(i,i+1)=namda(i)
end do
!设置 A 矩阵的首行元素
a(1,n)=u(1)
!设置 A 矩阵元素的末行元素
a(n,1)=namda(n)
!注意三种条件下，用 Gauss 法解线性方程时 N 大小设置
call gauss(a,d,M,N)
!--------以上以及求得系数
!已经完成插值多项式的建立
!------------------------------------------------
! 以下开始计算具体值
do k=1,nt
!------------
!  对要插值向量每个分量而言，先找到其在数据中的位置
do i=1,n-1
```

```
  if (t(k)<x(i+1)) exit
 end do
   ty(k)=M(i)*(x(i+1)-t(k))**3/6d0/h(i)+ M(i+1)*(t(k)-x(i))**3/6d0/h(i)+ &
      (y(i)-M(i)*h(i)**2/6d0)*(x(i+1)-t(k))/h(i)+ &
      (y(i+1)-M(i+1)*h(i)**2/6d0)*(t(k)-x(i))/h(i)
 end do
 !-------------------------------
 !为了方便读者比对结果，这里输出中间值，需要这些数值时可以去掉注释
 !!-------------------------
 !write(11,101)
 !write(11,102)((i,h(i),m(i)),i=0,n-1)   !输出 h
 !
 !
 !write(11,103)
 !write(11,104)((i,u(i),namda(i),d(i)),i=1,n-1)
 !
 !101 format(/,T5,'        h           m')
 !102 format(I3,2F16.8)
 !
 !103 format(/,T5,'       u           namda          d')
 !104 format(I3,3F16.8)
 !!-----------------------------------------------------------------
 end subroutine solve
 subroutine gauss(A,b,x,N)
 !---------------------------------subroutine  comment
 !  Version   :  V1.0
 !  Coded by  :  syz
 !  Date      :  2010-4-8
 !-----------------------------------------------------
 !  Purpose   :  高斯列主元消去法
 !               Ax=b
 !-----------------------------------------------------
 !  Input  parameters  :
 !      1.   A(N,N)系数矩阵
 !      2.   b(N)右向量
 !      3.   N 方程维数
 !  Output parameters  :
 !      1.  x  方程的根
 !      2.
 !  Common parameters  :
 !
 !----------------------------------------------------
 implicit real*8(a-z)
 integer::i,k,N
 integer::id_max  !主元素标号
 real*8::A(N,N),b(N),x(N)
 real*8::Aup(N,N),bup(N)
 !Ab 为增广矩阵 [Ab]
 real*8::Ab(N,N+1)
 real*8::vtemp1(N+1),vtemp2(N+1)
 Ab(1:N,1:N)=A
 Ab(:,N+1)=b
 !##########################################################
 !  这段是列主元消去法的核心部分
 do k=1,N-1
```

```
    elmax=dabs(Ab(k,k))
    id max=k
    !这段为查找主元素
    !这段程序的主要目的不是为了赋值最大元素给 elmax，而是为了找出最大元素对应的标号
     do i=k+1,n
      if (dabs(Ab(i,k))>elmax) then
        elmax=Ab(i,k)
        id max=i
      end if
    end do
!至此，已经完成查找最大元素，查找完成以后与 第 k 行交换
 !交换两行元素，其他不变
    vtemp1=Ab(k,:)
    vtemp2=Ab(id max,:)
    Ab(k,:)=vtemp2
    Ab(id_max,:)=vtemp1
!
!以上一大段是为交换两行元素，交换完成以后即按照消元法进行
!##########################################################
   do i=k+1,N
     temp=Ab(i,k)/Ab(k,k)
     Ab(i,:)=Ab(i,:)-temp*Ab(k,:)
   end do
end do
!-----------------------------
! 经过上一步，Ab 已经化为如下形式的矩阵
!           | *  *  *  *  # |
!     [A b]= | 0  *  *  *  # |
!           | 0  0  *  *  # |
!           | 0  0  0  *  # |
!
Aup(:,:)=Ab(1:N,1:N)
bup(:)=Ab(:,N+1)
!调用用上三角方程组的回带方法
call uptri(Aup,bup,x,n)
end subroutine gauss
subroutine uptri(A,b,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-8
!-----------------------------------------------------
!  Purpose   :  上三角方程组的回带方法
!               Ax=b
!-----------------------------------------------------
!  Input  parameters  :
!      1.   A(N,N)系数矩阵
!      2.   b(N)右向量
!      3.   N方程维数
!  Output parameters  :
!      1.  x  方程的根
!      2.
!  Common parameters  :
!
!----------------------------------------------------
```

```
implicit real*8(a-z)
integer::i,j,N
real*8::A(N,N),b(N),x(N)
x(N)=b(N)/A(N,N)
!回带部分
do i=n-1,1,-1
    x(i)=b(i)
   do j=i+1,N
    x(i)=x(i)-a(i,j)*x(j)
   end do
    x(i)=x(i)/A(i,i)
end do
end subroutine uptri
end module spline
program main
!------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  周期边界条件下的三次样条插值
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.
!      2.   reslut.txt 文件保存了计算结果
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use spline
implicit real*8(a-z)
integer::i,j
real*8::x(0:21),y(0:21),t(5),ty(5),simty(5)
open(unit=11,file='result.txt')
x=(/  0d0,&
    0.3000d0,&
    0.6000d0,&
    0.9000d0,&
    1.2000d0,&
    1.5000d0,&
    1.8000d0,&
    2.1000d0,&
    2.4000d0,&
    2.7000d0,&
    3.0000d0,&
    3.3000d0,&
    3.6000d0,&
    3.9000d0,&
    4.2000d0,&
    4.5000d0,&
```

```
    4.8000d0,&
    5.1000d0,&
    5.4000d0,&
    5.7000d0,&
    6.0000d0,&
    6.2832d0 /)
y=(/0.6000d0,&
    0.8687d0,&
    1.0598d0,&
    1.1563d0,&
    1.1495d0,&
    1.0399d0,&
    0.8375d0,&
    0.5603d0,&
    0.2330d0,&
   -0.1151d0,&
   -0.4529d0,&
   -0.7502d0,&
   -0.9806d0,&
   -1.1233d0,&
   -1.1657d0,&
   -1.1040d0,&
   -0.9437d0,&
   -0.6990d0,&
   -0.3919d0,&
   -0.0499d0,&
    0.2967d0,&
    0.6000d0  /)
!要进行计算的点 ------不需要按照大小排列
t=(/0.4d0,2.8d0,5.9d0,6.1d0,4.7d0/)
!调用方法函数
call solve(21,x,y,5,t,ty)
!仿真函数值
simty=dsin(t)+dcos(t)*0.6d0
write(11,101)
write(11,102)
write(11,103)((i,t(i),ty(i),simty(i)),i=1,5)
101  format(/,T9,'三次样条插值之周期条件',/)
102  format('  序列    插值点       插值结果        真值')
103  format(I3,3F16.8)
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，打开该文件可以看到内容如图 7-5 所示。

```
                三次样条插值之周期条件

 序列      插值点           插值结果           真值
 1       0.40000000       0.94200589       0.94205494
 2       2.80000000      -0.23037804      -0.23034525
 3       5.90000000       0.18260098       0.18261039
 4       6.10000000       0.40780541       0.40779856
 5       4.70000000      -1.00736694      -1.00735646
```

图 7-5 三次样条之周期条件

实际上，原题的数据来源于函数 $f(x)=\sin x+0.6\cos x$，该函数在 $0\sim 2\pi$ 为周期函数。在产生数据时，并不要求等间隔，比如最后两个节点之间即非等间隔。通过插值函数与函数真值比较可以看到计算还是有相当不错的精度。

7.7 反插值

1. 实验基本原理

设函数 f 的离散数据为

$$(x_i, y_i), y_i = f(x_i), i = 0,1,\ldots,n$$

插值的目的是在给定 $x_0, x_1, \cdots, x_n$ 之间给定自变量的值之后，去求函数 f 的近似值，不同的构造方法，得到不同插值方法。与之相反，反插值的目的在 $y_0, y_1, \cdots, y_n$ 之间给定函数值之后，求自变量 x 的近似值，其基本途径的实质仍然是插值方法。

这里以牛顿插值法为例说明反插值方法。设函数在点 $x_0, x_1, \cdots, x_n$ 上的值为 $y_0, y_1, \cdots, y_n$。若 f 的反函数 f^{-1} 存在，则 f^{-1} 在点 $y_0, y_1, \cdots, y_n$ 上的自变量值为 $x_0, x_1, \cdots, x_n$。如此，反函数 f^{-1} 以 $y_0, y_1, \cdots, y_n$ 为节点的牛顿插值多项式为

$$\begin{aligned} p(y) = & f^{-1}(y_0) + f^{-1}[y_0, y_1](y - y_0) + \\ & f^{-1}[y_0, y_1, y_2](y - y_0)(y - y_1) + \cdots + \\ & f^{-1}[y_0, y_1, \cdots, y_n](y - y_0)(y - y_1)\cdots(y - y_{n-1}) \end{aligned}$$

2. 实验目的与要求

- 了解反插值问题的应用背景。
- 能够构造出反插值多项式。
- 能编程处理反插值问题。

3. 实验内容与数据来源

设 $f(x)=x^3-3x^2-x+9$，已知 f 的下列函数值

x	$f(x)$
1.7200	0.988889766004701
1.7400	0.985719178835553
1.7600	0.982154317137618
1.7800	0.978196606808045
1.8000	0.973847630878195
1.8200	0.969109128880456

现求

方程 $f(x)=0$ 在区间 $[-1.7,-1.3]$ 的根。

求使 $f(x)=-1.6$ 的 x 值。

5. 程序代码

```
module  newton
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   反插值问题
!
!-----------------------------------------------------
!  Post Script :
!      1.   提供两个函数，一个是单点插值，一个是 solve 函数
!           注意两者的 N 是不一样的。
!           在单点插值中 N 是节点的个数减
!           为方便使用，在 solve 中 N 就直接指节点个数
!-----------------------------------------------------
contains
subroutine  solve(n,x,y,m,t,ty)
!---------------------------------subroutine  comment
!  Version    :  V1.0
!  Coded by   :  syz
!  Date       :
!-----------------------------------------------------
!  Purpose    :  方法主函数
!                可直接对多点插值
!-----------------------------------------------------
!  Input  parameters  :
!       1.  n   节点个数
!       2.  x  节点自变量值 N 维向量
!       3.  y  节点因变量值  N 维向量
!       4.  m   要计算点的个数
!       5.  t   要计算的点   M 向量
!  Output parameters  :
!       1.  ty  计算结果，为 M 维向量
```

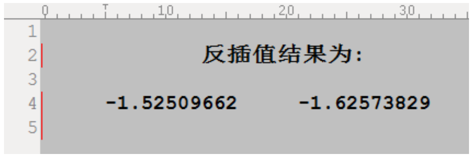
```
!      2.
!  Common parameters  :
!
!-----------------------------------------------------
!  Post Script :
!      1.   注意这里的 N  就直接指节点的个数
!      2.   而子函数 new 中 N 是指节点个数减 1
!-----------------------------------------------------
implicit real*8(a-z)
integer::n,m
integer::i,j,k
real*8::x(n),y(n),t(m),ty(m)
do i=1,m
call new(n-1,x,y,t(i),ty(i))
end do
end subroutine solve
subroutine new(n,x,y,t,ty)
!----------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!------------------------------------------------------
!  Purpose   :  单点插值程序
!               此函数目的为 solve 所调用
!------------------------------------------------------
!  Input  parameters  :
!      1.
!      2.   t 标量
!  Output parameters  :
!      1.
!      2.    ty 标量
!  Common parameters  :
!
!-----------------------------------------------------
!  Post Script :
!      1.    N 是节点个数减 1，比如共个节点 则 N=3
!      2.
!-----------------------------------------------------
implicit real*8(a-z)
integer::n
integer::i,j,k
real*8::x(0:n),y(0:n)
real*8::b(n+1)
real*8::Q(0:n,0:n)
do i=0,n
Q(i,0)=y(i)
end do
do i=1,n
  do j=1,i
   Q(i,j)=(Q(i,j-1)-Q(i-1,j-1))/(x(i)-x(i-j))
  end do
end do
b(n+1)=Q(n,n)
do k=n,1,-1
  b(k)=Q(k-1,k-1)+b(k+1)*(t-x(k-1))
end do
ty=b(1)
end subroutine new
```

```
end module newton
program main
!-------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.12
!-----------------------------------------------------
!  Purpose   :  反插值问题主函数
!              solve 函数可以直接对向量插值
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.
!      2.
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
call drive
end program main
subroutine  drive
!--------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.12
!-----------------------------------------------------
!  Purpose   :  反插值问题驱动函数
!
!-----------------------------------------------------
!  Input  parameters  :
!      1.
!      2.
!  Output parameters  :
!      1.
!      2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
use newton
implicit real*8(a-z)
integer::N=5,M=2
real*8::x(5),y(5),t(2),ty(2)
open(unit=11,file='result1.txt')
!插值节点极其函数值
x=(/-1.3d0,-1.4d0,-1.5d0,-1.6d0,-1.7d0/)
y=(/3.033d0,1.776d0,0.375d0,-1.176d0,-2.883d0/)
!要计算的点
t=(/0d0,-1.6d0/)
call solve(n,y,x,m,t,ty)
write(11,101)
write(11,102)ty
```

```
101 format(/,T5,'反插值结果为:')
102 format(/,2F10.4)
end subroutine drive
```

6. 实验结论

计算结果保存在文件 result.txt 文件中，如图 7-6 所示。



反插值结果为:

-1.52509662   -1.62573829

图 7-6　反插值结果

当把-1.52509662 带入原方程函数时候得到 8.524562075606923e-005，-1.62573829 把带入方程函数得到-1.600203596439146。从计算结果可以看到反插值精度是很不错的。

7.8　第一类标准B样条

1. 实验基本原理

近几十年，函数逼近在理论研究与实际应用中均取得重大进展，其中十分活跃的分支即样条函数。

前面几节导出三次样条插值函数在每个子区间有一表达式，这在应用和理论分析中都很不方便。接下来几小节专门讨论样条函数系统，使得其他所有样条函数都可以由它的线性组合得到。这些样条构成某些样条空间的基，称为 B 样条。一旦给定节点，B 样条很容易通过递推关系产生，而且算法也较为简单。B 样条以其优美的理论和数值计算中的典型性著称。在许多实践中都有了较为广泛的应用，如弹道逼近等领域。有兴趣的读者可以参看王正明等著《弹道跟踪数据的校准与评估》及刘利生等著《卫星导航测量差分自校准融合技术》。

记

$$u_+^m=\begin{cases}u^m,(u\ge 0)\\0,(u<0)\end{cases},(m=1,2,\ldots)$$

称为 m 次半截单项式，并规定

$$u_+^0=\begin{cases}1,(u>0)\\\dfrac{1}{2},(u=0)\\0,(u<0)\end{cases}$$

设 Ω_k 是 $(-\infty,+\infty)$ 上的函数，

$$\Omega_k = \frac{\sum_{j=0}^{k+1}(-1)^j \binom{k+1}{j}\left(x+\frac{k+1}{2}-j\right)_+^k}{k!}$$

称之为 k 次基样条或 B 样条。

如

$$\Omega_3 = \begin{cases} 0,\left(|x| \ge 2\right) \\ \frac{1}{2}|x|^3 - x^2 + \frac{2}{3},\left(|x| \le 1\right) \\ -\frac{1}{6}|x|^3 + x^2 - 2|x| + \frac{4}{3},\left(1 < |x| < 2\right) \end{cases}$$

设划分是均匀的，即

$$\begin{cases} x_i = a + ih \\ h = \frac{b-a}{n} \end{cases},i = 0,1,\dots,n$$

三次 B 样条中共有 $n+3$ 个 B 样条，考虑以它们的线性组合作为插值函数

$$S(x) = \sum_{j=-1}^{n+1} c_j \Omega_3\left(\frac{x-x_0}{h} - j\right)$$

B 样条插值与前面介绍的三次样条插值有相似的边界条件处理情况，这一节先介绍第一类边界条件问题。

第一类边界条件即

$$\begin{cases} S'(x_0) = y_0' \\ S'(x_n) = y_n' \end{cases}$$

根据插值条件及边界条件有

$$\begin{cases} S'(x_0) = \frac{1}{h}\sum_{j=-1}^{n+1} c_j \Omega_3'(-j) = y_0' \\ S(x_i) = \sum_{j=-1}^{n+1} c_j \Omega_3'(i-j) = y_i \qquad ,i = 0,1,\dots,n \\ S'(x_0) = \frac{1}{h}\sum_{j=-1}^{n+1} c_j \Omega_3'(n-j) = y_n' \end{cases}$$

从而归结为线性方程组

$$\mathbf{Ac} = \mathbf{f}$$

其中

$$
\mathbf{A}=\begin{pmatrix}
4 & 2 & & & & \\
1 & 4 & 1 & & & \\
 & 1 & 4 & 1 & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & 1 & 4 & 1 \\
 & & & & 2 & 4
\end{pmatrix}
$$

$$
\mathbf{f}=\begin{pmatrix}
6y_0+2hy_0' \\
6y_1 \\
6y_2 \\
\vdots \\
6y_{n-1} \\
6y_n-2hy_n'
\end{pmatrix},\mathbf{c}=\begin{pmatrix}
c_0 \\
c_1 \\
\vdots \\
c_{n-1} \\
c_n
\end{pmatrix}
$$

还有关系式

$$
\begin{cases}
c_{-1}=c_1-2hy_0' \\
c_{n+1}=c_{n-1}+2hy_n'
\end{cases}
$$

以上方程为三对角方程，可以采用第一章介绍的追赶方法计算。此时，已经完整的给出样条基在 Hilbert 空间下的投影坐标，有此可以方便的计算出插值结果。在程序设计时，我们直接可以输入要插值的向量，计算输出插值结果向量。

2. 实验目的与要求

- 大致了解 B 样条背景。
- 大致了解第一类边界条件下 B 样条插值方法。
- 会使用作者编写的程序进行相关数据处理。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(1:M)	要计算的函数值
输入参变量	**数据类型**	**变量说明**
N	INTEGER	插值节点个数减 1
X	REAL*8(0:N)	节点自变量
Y	REAL*8(0:N)	节点因变量
M	INTEGER	要计算的向量的维数
T	REAL*8(1:M)	要计算的节点向量
DA	REAL*8	起点处导数
DB	REAL*8	终点处导数

3. 实验内容与数据来源

已知离散点列如下

i	x_i	$f(x_i)$
1	1.0000	0.841470984807897
2	1.2000	0.932039085967226
3	1.4000	0.985449729988460
4	1.6000	0.999573603041505
5	1.8000	0.973847630878195
6	2.0000	0.909297426825682
7	2.2000	0.808496403819590
8	2.4000	0.675463180551151
9	2.6000	0.515501371821464
10	2.8000	0.334988150155905
11	3.0000	0.141120008059867

且

$f'(1) = 0.540302305868140$

$f'(3) = -0.989992496600445$

今给定 $\mathbf{x} = (2.1 \quad 1.7 \quad 1.9 \quad 2.5)^T$，要求用标准 B 样条插值求 $\mathbf{f}(\mathbf{x})$。

5. 程序代码

```
module spline
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date       :
!-----------------------------------------------------
!  Description :   第一类标准 B 样条模块
!
!-----------------------------------------------------
!  Parameters  :
!      1.
!      2.
!-----------------------------------------------------
!  Contains    :
!      1.   方法函数
!      2.   3 阶样条基
!      3.   追赶法解三对角方程
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
contains
subroutine solve(n,x,y,da,db,m,t,ty)
!----------------------------------subroutine  comment
```

```
! Version   : V1.0
! Coded by  : syz
! Date      :
!-----------------------------------------------------
! Purpose   : 第一类标准 B 样条方法函数
!              可以直接对多点插值
!-----------------------------------------------------
! Input parameters :
!      1.n 节点个数减,比如九个节点 N=8
!      2. x 节点向量  (: n) 维
!      3. y 节点处函数值(0:n)维
!      4. da  起点处导数值
!      5. db  终点处导数值
!      6.  m  要计算点的个数
!      7.  t  要计算的点  (: m) 维向量
! Output parameters  :
!      1.  ty  计算结果 (: m) 维向量
!      2.
! Common parameters  :
!
!----------------------------------------------------
! Post Script :
!      1.
!      2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,m,i,j,k
real*8::x(0:n),y(0:n)
real*8::t(m),ty(m)
real*8::A(0:n,0:n),c0(0:n),f(0:n),c(-1:n+1)
!设置 A 矩阵-------------------
!设初值
A=0
do i=0,n
   A(i,i)=4d0
end do
!主对角以下元素
do i=1,n-1
   A(i,i-1)=1d0
end do
A(n,n-1)=2d0
!主对角以上元素
do i=1,n-1
A(i,i+1)=1d0
end do
A(0,1)=2d0
!------------------------
!设置 f 向量
h=x(1)-x(0)  ! 步长间隔
do i=1,n-1
f(i)=6d0*y(i)
end do
f(0)=6d0*y(0)+2d0*h*da
f(n)=6d0*y(n)-2d0*h*db
!注意调用追赶法时，方程维数的值
```

```
    call chase(A,f,c0,N+1)
    do i=0,n
      c(i)=c0(i)
    end do
    c(-1)=c0(1)-2d0*h*da
    c(n+1)=c0(n-1)+2d0*h*db
    !至此已经给出完整的系数 c
    !以下开始计算值
    do k=1,m
    ty(k)=0
    do j=-1,n+1
       ty(k)=ty(k)+c(j)*omiga3((t(k)-x(0))/h-j)
    end do
    end do
    end subroutine solve
    function omiga3(x)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :
    !-----------------------------------------------------
    !  Purpose   :  三阶 B 样条基
    !
    !----------------------------------------------------
    !  Post Script :
    !       1.
    !       2.
    !----------------------------------------------------
    implicit real*8(a-z)
    if (dabs(x)>=2d0) then
     omiga3=0
    else if (dabs(x)<=1d0) then
     omiga3=0.5d0*(dabs(x))**3-x**2+2d0/3d0
    else if (dabs(x)>1d0.and.dabs(x)<2d0) then
     omiga3=-1d0/6d0*(dabs(x))**3+x**2-2d0*dabs(x)+4d0/3d0
    end if
    end function omiga3
     subroutine chase(A,f,x,N)
    !---------------------------------subroutine  comment
    !  Version   :  V1.0
    !  Coded by  :  syz
    !  Date      :  2010-4-9
    !-----------------------------------------------------
    !  Purpose   :  追赶法计算三对角方程组
    !             Ax=f
    !-----------------------------------------------------
    !  Input  parameters  :
    !       1.  A 系数矩阵
    !       2.  f 右向量
    !  Output parameters  :
    !       1.  x 方程的解
    !       2.  N 维数
    !  Common parameters  :
    !
    !----------------------------------------------------
```

```
!  Post Script :
!       1.   注意：该方法仅适用于三对角方程组
!       2.
!-----------------------------------------------------
implicit real*8(a-z)
 integer::N
 real*8::A(N,N),f(N),x(N)
 real*8::L(2:N),u(N),d(1:N-1)
real*8::c(1:N-1),b(N),e(2:N)
 integer::i
 real*8::y(N)
!---------------把A矩阵复制给向量e,b,c
do i=1,N
   b(i)=a(i,i)
end do
do i=1,N-1
   c(i)=a(i,i+1)
end do
do i=2,N
  e(i)=a(i,i-1)
end do
!------------------------
do i=1,N-1
 d(i)=c(i)
end do
u(1)=b(1)
do i=2,N
  L(i)=e(i)/u(i-1)
  u(i)=b(i)-L(i)*c(i-1)
end do
!------开始回带,求得y
y(1)=f(1)
do i=2,N
  y(i)=f(i)-L(i)*y(i-1)
end do
!-----开始回带，求得x
x(n)=y(n)/u(n)
do i=n-1,1,-1
x(i)=(y(i)-c(i)*x(i+1))/u(i)
end do
end subroutine chase
end module spline
program  main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.13
!-----------------------------------------------------
!  Purpose   :  第一类标准B样条主函数
!
!-----------------------------------------------------
!  In put data  files :
!       1.
!       2.
!  Output data files  :
```

```
!      1.
!      2.   result.txt 保存计算结果
!----------------------------------------------------
!  Post Script :
!      1.
!      2.
!----------------------------------------------------
use spline
implicit real*8(a-z)
real*8::x(0:10),y(0:10),t(4),ty(4),rty(4)
 open(unit=11,file='result.txt')
!节点
x=(/    1.0000d0,&
   1.2000d0,&
   1.4000d0,&
   1.6000d0,&
   1.8000d0,&
   2.0000d0,&
   2.2000d0,&
   2.4000d0,&
   2.6000d0,&
   2.8000d0,&
   3.0000d0/)
! 节点因变量
y=(/   0.841470984807897d0,&
  0.932039085967226d0,&
  0.985449729988460d0,&
  0.999573603041505d0,&
  0.973847630878195d0,&
  0.909297426825682d0,&
  0.808496403819590d0,&
  0.675463180551151d0,&
  0.515501371821464d0,&
  0.334988150155905d0,&
  0.141120008059867d0/)
!边界导数值  这里实为 dcos(1d0)、dcos(3d0)
  da=0.540302305868140d0
  db=-0.989992496600445d0
! 要计算的点,不必按照大小排列
  t=(/2.1d0,1.7d0,1.9d0,2.5d0/)
! 调用方法函数
  call solve(10,x,y,da,db,4,t,ty)
 !真值
 rty=dsin(t)
 write(11,101)
write(11,102)((t(i),ty(i),rty(i)),i=1,4)
101 format(/,T15,'第一类标准 B 样条',//,&
           '    节点      插值结果    真值',/)
102 format(3F12.6)
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，打开该文件如图 7-7 所示。

第一类标准B样条

节点	插值结果	真值
2.100000	0.863206	0.863209
1.700000	0.991661	0.991665
1.900000	0.946296	0.946300
2.500000	0.598470	0.598472

图 7-7　第一类标准 B 样条插值结果

实际上实验中的数据来源于仿真函数 $f(x)=\sin x$，可以看到采用第一类标准 B 样条插值结果还是很不错的。

7.9　第二类标准B样条

1. 实验基本原理

第二类边界条件即

$$\begin{cases} S''(x_0)=y_0'' \\ S''(x_n)=y_n'' \end{cases}$$

当

$$S''(x_0)=S''(x_n)=0$$

时称为自然边界条件，可以看出自然边界条件是第二类边界条件的特殊情况。

线性方程组

$$\mathbf{Ac}=\mathbf{f}$$

其中

$$\mathbf{A}=\begin{pmatrix} 4 & 2 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 4 & 1 \\ & & & & 2 & 4 \end{pmatrix}$$

$$\mathbf{f}=\begin{pmatrix} 6y_1-y_0+\dfrac{h^2}{6}y_0'' \\ 6y_2 \\ 6y_3 \\ \vdots \\ 6y_{n-2} \\ 6y_{n-1}-y_n+\dfrac{h^2}{6}y_n'' \end{pmatrix},\mathbf{c}=\begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-2} \\ c_{n-1} \end{pmatrix}$$

还有关系式

$$\begin{cases} c_0=y_0-\dfrac{h^2}{6}y_0'' \\ c_{-1}=2c_0-c_1+h^2y_0'' \\ c_n=y_n-\dfrac{h^2}{6}y_n'' \\ c_{n+1}=2c_n-c_{n-1}+h^2y_n'' \end{cases}$$

至此，也已经给出基函数在 Hilbert 空间下的投影分量。计算具体的插值结果只需要对基做展开即可。

2. 实验目的与要求

- 了解第二类边界条件下 B 样条插值算法的基本流程。
- 会使用作者编写的函数进行相关数据处理。

3. 实验内容与数据来源

已知离散点列如下

i	x_i	$f(x_i)$
1	1.0000	0.841470984807897
2	1.2000	0.932039085967226
3	1.4000	0.985449729988460
4	1.6000	0.999573603041505
5	1.8000	0.973847630878195
6	2.0000	0.909297426825682
7	2.2000	0.808496403819590
8	2.4000	0.675463180551151
9	2.6000	0.515501371821464
10	2.8000	0.334988150155905
11	3.0000	0.141120008059867

且

$f''(1)=0.540302305868140$

$f''(3)=-0.989992496600445$

今给定 $\mathbf{x}=(2.1\quad 1.7\quad 1.9\quad 2.5)^T$，要求用标准 B 样条插值求 $\mathbf{f}(\mathbf{x})$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(1:M)	要计算的函数值
输入参变量	**数据类型**	**变量说明**
N	INTEGER	插值节点个数减 1
X	REAL*8(0:N)	节点自变量
Y	REAL*8(0:N)	节点因变量
M	INTEGER	要计算的向量的维数
T	REAL*8(1:M)	要计算的节点向量
D2FA	REAL*8	起点二阶导数
D2FB	REAL*8	终点二阶处导数

5. 程序代码

```
module spline
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   第二类标准 B 样条模块
!
!-----------------------------------------------------
!  Parameters   :
!      1.
!      2.
!-----------------------------------------------------
!  Contains     :
!      1.    方法函数
!      2.    3 阶样条基
!      3.    追赶法解三对角方程
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
contains
subroutine solve(n,x,y,d2a,d2b,m,t,ty)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.13
```

```
!----------------------------------------------------
!  Purpose   :  第二类标准 B 样条方法函数
!               可以直接对多点插值
!----------------------------------------------------
!  Input  parameters  :
!      1.n 节点个数减,比如九个节点 N=8
!      2. x 节点向量  (: n) 维
!      3. y 节点处函数值(0:n)维
!      4. d2a  起点处二阶导数值
!      5. d2b  终点处二阶导数值
!      6.  m  要计算点的个数
!      7.  t  要计算的点  (: m) 维向量
!  Output parameters  :
!      1.  ty  计算结果 (: m) 维向量
!      2.
!  Common parameters  :
!
!---------------------------------------------------
!  Post Script :
!      1.
!      2.
!---------------------------------------------------
implicit real*8(a-z)
integer::n,m,i,j,k
real*8::x(0:n),y(0:n)
real*8::t(m),ty(m)
real*8::A(1:n-1,1:n-1),c0(1:n-1),f(1:n-1),c(-1:n+1)
!设置 A 矩阵-------注意与第一类 B 样条的下标不同
!设初值
A=0
do i=1,n-1
   A(i,i)=4d0
end do
!主对角以下元素
do i=2,n-1
   A(i,i-1)=1d0
end do
!主对角以上元素
do i=1,n-2
A(i,i+1)=1d0
end do
!-------------------------
!设置 f 向量
h=x(1)-x(0)  ! 步长间隔
do i=2,n-2
f(i)=6d0*y(i)
end do
f(1)=6d0*y(1)-y(0)+h**2/6d0*d2a
f(n-1)=6d0*y(n-1)-y(n)+h**2/6d0*d2b
!注意调用追赶法时，方程维数的值,这与第一类 B 样条不同
!由参数意义决定
call chase(A,f,c0,N-1)
do i=1,n-1
  c(i)=c0(i)
end do
```

```
c(0)=y(0)-h**2/6d0*d2a
c(-1)=2d0*c(0)-c(1)+h**2*d2a
c(n)=y(n)-h**2/6d0*d2b
c(n+1)=2d0*c(n)-c(n-1)+h**2*d2b
!至此已经给出完整的系数 c
!以下开始计算值
do k=1,m
ty(k)=0
do j=-1,n+1
   ty(k)=ty(k)+c(j)*omiga3((t(k)-x(0))/h-j)
end do
end do
end subroutine solve
function omiga3(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  三阶 B 样条基
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
if (dabs(x)>=2d0) then
 omiga3=0
else if (dabs(x)<=1d0) then
 omiga3=0.5d0*(dabs(x))**3-x**2+2d0/3d0
else if (dabs(x)>1d0.and.dabs(x)<2d0) then
 omiga3=-1d0/6d0*(dabs(x))**3+x**2-2d0*dabs(x)+4d0/3d0
end if
end function omiga3
 subroutine chase(A,f,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!-----------------------------------------------------
!  Purpose   :  追赶法计算三对角方程组
!              Ax=f
!-----------------------------------------------------
!  Input  parameters  :
!       1.  A 系数矩阵
!       2.  f 右向量
!  Output parameters  :
!       1.  x 方程的解
!       2.  N 维数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.   注意：该方法仅适用于三对角方程组
```

```
!      2.
!---------------------------------------------------
implicit real*8(a-z)
 integer::N
 real*8::A(N,N),f(N),x(N)
 real*8::L(2:N),u(N),d(1:N-1)
 real*8::c(1:N-1),b(N),e(2:N)
 integer::i
real*8::y(N)
!---------------把 A 矩阵复制给向量 e,b,c
do i=1,N
   b(i)=a(i,i)
end do
do i=1,N-1
  c(i)=a(i,i+1)
end do
do i=2,N
  e(i)=a(i,i-1)
end do
!------------------------
do i=1,N-1
 d(i)=c(i)
end do
u(1)=b(1)
do i=2,N
  L(i)=e(i)/u(i-1)
  u(i)=b(i)-L(i)*c(i-1)
end do
!------开始回带,求得 y
y(1)=f(1)
do i=2,N
  y(i)=f(i)-L(i)*y(i-1)
end do
!-----开始回带，求得 x
x(n)=y(n)/u(n)
do i=n-1,1,-1
x(i)=(y(i)-c(i)*x(i+1))/u(i)
end do
end subroutine chase
end module spline
program  main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.13
!-----------------------------------------------------
!  Purpose   :  第二类标准 B 样条主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.
!      2.   result.txt 保存计算结果
```

```
!-------------------------------------------------------
!  Post Script :
!      1.
!      2.
!-------------------------------------------------------
use spline
implicit real*8(a-z)
real*8::x(0:10),y(0:10),t(4),ty(4),rty(4)
 open(unit=11,file='result.txt')
!节点
x=(/    1.0000d0,&
    1.2000d0,&
    1.4000d0,&
    1.6000d0,&
    1.8000d0,&
    2.0000d0,&
    2.2000d0,&
    2.4000d0,&
    2.6000d0,&
    2.8000d0,&
    3.0000d0/)
! 节点因变量
y=(/   0.841470984807897d0,&
   0.932039085967226d0,&
   0.985449729988460d0,&
   0.999573603041505d0,&
   0.973847630878195d0,&
   0.909297426825682d0,&
   0.808496403819590d0,&
   0.675463180551151d0,&
   0.515501371821464d0,&
   0.334988150155905d0,&
   0.141120008059867d0/)
!第二类边界条件  二阶导数值
   d2a=0.540302305868140d0
   d2b=-0.989992496600445d0
! 要计算的点,不必按照大小排列
   t=(/2.1d0,1.7d0,1.9d0,2.5d0/)
! 调用方法函数
   call solve(10,x,y,d2a,d2b,4,t,ty)
  !真值
  rty=dsin(t)
  write(11,101)
write(11,102)((t(i),ty(i),rty(i)),i=1,4)
101 format(/,T15,'第二类标准B样条',//,&
              '    节点      插值结果    真值',/)
102 format(3F12.6)
end program main
```

6. 实验结论

计算结果保存在文件 result.txt 中，内容如图 7-8 所示。

第二类标准B样条

节点	插值结果	真值
2.100000	0.863217	0.863209
1.700000	0.991710	0.991665
1.900000	0.946281	0.946300
2.500000	0.598581	0.598472

图 7-8　第二类标准 B 样条

实际上本实验的数据插值节点与上一实验相同，对于边界条件这里但是二阶导数未按照仿真函数取值，可以看到在区间内插值依然精度很好，可见样条插值方法的优良性。

7.10　第三类标准B样条

1. 实验基本原理

第三类边界条件即周期边界条件

$$S(x_0)=S(x_n)=y_0$$

$$S^{(k)}(x_0+0)=S^{(k)}(x_0-0),(k=1,2)$$

由插值条件及第三类边界条件可以推出

$$\begin{cases} c_{n+1}=c_1 \\ c_0=c_n \\ c_{-1}=c_{n-1} \end{cases}$$

及线性方程

$$\mathbf{Ac}=\mathbf{f}$$

其中

$$\mathbf{A}=\begin{pmatrix} 4 & 2 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 4 & 1 \\ & & & & 2 & 4 \end{pmatrix}$$

$$\mathbf{f}=\begin{pmatrix}6y_1\\6y_2\\6y_3\\\vdots\\6y_{n-1}\\6y_0\end{pmatrix},\mathbf{c}=\begin{pmatrix}c_1\\c_2\\\vdots\\c_{n-1}\\c_n\end{pmatrix}$$

可以由线性方程先解出 $\mathbf{c}$，继而由

$$\begin{cases}c_{n+1}=c_1\\c_0=c_n\\c_{-1}=c_{n-1}\end{cases}$$

可以给出 c_{-1},c_0,c_{n+1}。

在解线性方程组时，因是三对角阵，故可采用追赶法计算。至此，已经给出第三类边界条件下 B 样条插值基函数在 Hilbert 空间下的坐标，若要计算插值结果只需要对基函数展开。

2. 实验目的与要求

- 了解第三类边界条件下 B 样条插值的适用范围。
- 大致了周期条件下 B 样条插值的计算流程。
- 会使用作者提供的函数进行相关数据处理。

3. 实验内容与数据来源

已知周期条件下的离散点列如下：

i	x_i	$f(x_i)$
1	0	1
2	2	1.144122806
3	4	1.260073511
4	6	1.344997024
5	8	1.396802247
6	10	1.414213562
7	12	1.396802247
8	14	1.344997024
9	16	1.260073511
10	18	1.144122806
11	20	1
12	22	0.831253876
13	24	0.642039522

（续表）

i	x_i	$f(x_i)$
14	26	0.437016024
15	28	0.221231742
16	30	1.11E-16
17	32	-0.221231742
18	34	-0.437016024
19	36	-0.642039522
20	38	-0.831253876
21	40	-1
22	42	-1.144122806
23	44	-1.260073511
24	46	-1.344997024
25	48	-1.396802247
26	50	-1.414213562
27	52	-1.396802247
28	54	-1.344997024
29	56	-1.260073511
30	58	-1.144122806
31	60	-1
32	62	-0.831253876
33	64	-0.642039522
34	66	-0.437016024
35	68	-0.221231742
36	70	-3.33E-16
37	72	0.221231742
38	74	0.437016024
39	76	0.642039522
40	78	0.831253876
41	80	1

今给定向量 $\mathbf{x}=(35.2\quad 27.6\quad 19\quad 65.7\quad 55.7)^T$，采用标准 B 样条插值计算 $\mathbf{f}(\mathbf{x})$。

4. 函数调用接口说明

输出参变量	数据类型	变量说明
TY	REAL*8(1:M)	要计算的函数值
输入参变量	**数据类型**	**变量说明**
N	INTEGER	插值节点个数减 1
X	REAL*8(0:N)	节点自变量
Y	REAL*8(0:N)	节点因变量
M	INTEGER	要计算的向量的维数
T	REAL*8(1:M)	要计算的节点向量

5. 程序代码

```
module spline
!----------------------------------------module coment
!  Version     :  V1.0
!  Coded by    :  syz
!  Date        :
!-----------------------------------------------------
!  Description :   第三类标准 B 样条模块
!                 处理周期函数插值
!-----------------------------------------------------
!  Parameters  :
!      1.
!      2.
!-----------------------------------------------------
!  Contains    :
!      1.   方法函数
!      2.   3 阶样条基
!      3.   追赶法解三对角方程
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
contains
subroutine solve(n,x,y,m,t,ty)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.13
!-----------------------------------------------------
!  Purpose   :  第三类标准 B 样条方法函数
!              可以直接对多点插值
!              处理周期函数插值
!-----------------------------------------------------
!  Input  parameters  :
!       1.n 节点个数减,比如九个节点 N=8
!       2. x 节点向量  (: n) 维
!       3. y 节点处函数值(0:n)维
!       6.  m  要计算点的个数
!       7.  t  要计算的点  (: m) 维向量
!  Output parameters  :
!       1.  ty  计算结果 (: m) 维向量
!       2.
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
integer::n,m,i,j,k
real*8::x(0:n),y(0:n)
```

```
real*8::t(m),ty(m)
real*8::A(1:n,1:n),c0(1:n),f(1:n),c(-1:n+1)
!设置 A 矩阵-------注意与 A 矩阵下标
!设初值
A=0
do i=1,n
   A(i,i)=4d0
end do
!主对角以下元素
do i=2,n
   A(i,i-1)=1d0
end do
!主对角以上元素
do i=1,n-1
A(i,i+1)=1d0
end do
!-------------------------
!设置 f 向量
do i=1,n-1
f(i)=6d0*y(i)
end do
f(n)=6d0*y(0)
!注意调用追赶法时，方程维数的值
!由参数意义决定
call chase(A,f,c0,N)
do i=1,n
  c(i)=c0(i)
end do
c(n+1)=c(1)
c(0)=c(n)
c(-1)=c(n-1)
!至此已经给出完整的系数 c
h=x(1)-x(0)
!以下开始计算值
do k=1,m
ty(k)=0
do j=-1,n+1
   ty(k)=ty(k)+c(j)*omiga3((t(k)-x(0))/h-j)
end do
end do
end subroutine solve
function omiga3(x)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :
!-----------------------------------------------------
!  Purpose   :  三阶 B 样条基
!
!----------------------------------------------------
!  Post Script :
!       1.
!       2.
!----------------------------------------------------
implicit real*8(a-z)
if (dabs(x)>=2d0) then
```

```
 omiga3=0
else if (dabs(x)<=1d0) then
 omiga3=0.5d0*(dabs(x))**3-x**2+2d0/3d0
else if (dabs(x)>1d0.and.dabs(x)<2d0) then
 omiga3=-1d0/6d0*(dabs(x))**3+x**2-2d0*dabs(x)+4d0/3d0
end if
end function omiga3
 subroutine chase(A,f,x,N)
!---------------------------------subroutine  comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010-4-9
!-----------------------------------------------------
!  Purpose   :  追赶法计算三对角方程组
!              Ax=f
!-----------------------------------------------------
!  Input  parameters  :
!      1.  A 系数矩阵
!      2. f 右向量
!  Output parameters  :
!      1.  x 方程的解
!      2.  N 维数
!  Common parameters  :
!
!----------------------------------------------------
!  Post Script :
!      1.   注意：该方法仅适用于三对角方程组
!      2.
!---------------------------------------------------
implicit real*8(a-z)
 integer::N
 real*8::A(N,N),f(N),x(N)
 real*8::L(2:N),u(N),d(1:N-1)
real*8::c(1:N-1),b(N),e(2:N)
 integer::i
 real*8::y(N)
!---------------把A矩阵复制给向量e,b,c
do i=1,N
   b(i)=a(i,i)
end do
do i=1,N-1
  c(i)=a(i,i+1)
end do
do i=2,N
  e(i)=a(i,i-1)
end do
!-------------------------
do i=1,N-1
 d(i)=c(i)
end do
u(1)=b(1)
do i=2,N
  L(i)=e(i)/u(i-1)
  u(i)=b(i)-L(i)*c(i-1)
end do
!------开始回带,求得y
```

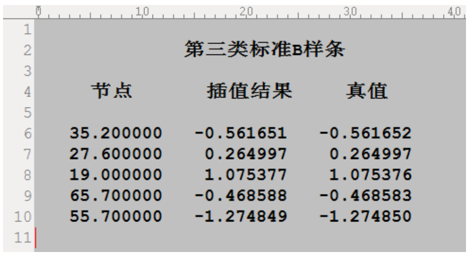
```
y(1)=f(1)
do i=2,N
  y(i)=f(i)-L(i)*y(i-1)
end do
!-----开始回带，求得 x
x(n)=y(n)/u(n)
do i=n-1,1,-1
x(i)=(y(i)-c(i)*x(i+1))/u(i)
end do
end subroutine chase
end module spline
program  main
!--------------------------------------program comment
!  Version   :  V1.0
!  Coded by  :  syz
!  Date      :  2010.05.13
!-----------------------------------------------------
!  Purpose   :  第三类标准 B 样条主函数
!
!-----------------------------------------------------
!  In put data  files :
!      1.
!      2.
!  Output data files  :
!      1.
!      2.   result.txt 保存计算结果
!-----------------------------------------------------
!  Post Script :
!      1.
!      2.
!-----------------------------------------------------
use spline
implicit real*8(a-z)
real*8::x(0:40),y(0:40),t(5),ty(5),rty(5)
real*8,parameter::pi=3.141592653589793d0
 open(unit=11,file='result.txt')
!节点
x=(/ 0d0,  &
     2d0,  &
     4d0,  &
     6d0,  &
     8d0,  &
    10d0,  &
    12d0,  &
    14d0,  &
    16d0,  &
    18d0,  &
    20d0,  &
    22d0,  &
    24d0,  &
    26d0,  &
    28d0,  &
    30d0,  &
    32d0,  &
    34d0,  &
    36d0,  &
```

```
        38d0,  &
        40d0,  &
        42d0,  &
        44d0,  &
        46d0,  &
        48d0,  &
        50d0,  &
        52d0,  &
        54d0,  &
        56d0,  &
        58d0,  &
        60d0,  &
        62d0,  &
        64d0,  &
        66d0,  &
        68d0,  &
        70d0,  &
        72d0,  &
        74d0,  &
        76d0,  &
        78d0,  &
        80d0  /)
    ! 节点因变量
    y=(/   1.000000000000000d0 ,&
       1.144122805635369d0 ,&
       1.260073510670101d0 ,&
       1.344997023927915d0 ,&
       1.396802246667421d0 ,&
       1.414213562373095d0 ,&
       1.396802246667421d0 ,&
       1.344997023927915d0 ,&
       1.260073510670101d0 ,&
       1.144122805635369d0 ,&
       1.000000000000000d0 ,&
       0.831253875554907d0 ,&
       0.642039521920206d0 ,&
       0.437016024448821d0 ,&
       0.221231742082474d0 ,&
       0.000000000000000d0 ,&
      -0.221231742082474d0 ,&
      -0.437016024448821d0 ,&
      -0.642039521920206d0 ,&
      -0.831253875554907d0 ,&
      -1.000000000000000d0 ,&
      -1.144122805635369d0 ,&
      -1.260073510670101d0 ,&
      -1.344997023927915d0 ,&
      -1.396802246667420d0 ,&
      -1.414213562373095d0 ,&
      -1.396802246667421d0 ,&
      -1.344997023927915d0 ,&
      -1.260073510670101d0 ,&
      -1.144122805635369d0 ,&
      -1.000000000000000d0 ,&
      -0.831253875554907d0 ,&
      -0.642039521920206d0 ,&
```

```
     -0.437016024448821d0 ,&
     -0.221231742082475d0 ,&
     -0.000000000000000d0 ,&
      0.221231742082474d0 ,&
      0.437016024448821d0 ,&
      0.642039521920206d0 ,&
      0.831253875554907d0 ,&
      1.000000000000000d0/)
   ! 要计算的点,不必按照大小排列
      t=(/35.2d0,27.6d0,19d0,65.7d0,55.7d0/)
   ! 调用方法函数
      call solve(40,x,y,5,t,ty)
     !真值
     rty=dsin(t*pi/40)+dcos(t*pi/40)
     write(11,101)
   write(11,102)((t(i),ty(i),rty(i)),i=1,5)
   101 format(/,T15,'第三类标准 B 样条',//,&
                 '     节点       插值结果    真值',/)
   102 format(3F12.6)
   end program main
```

6. 实验结论

实验中的数据实际上是用函数 $f(x)=\sin\left(\frac{\pi}{40}x\right)+\cos\left(\frac{\pi}{40}x\right)$ 仿真得到的。计算结果保存在 result.txt 文件中，内容如图 7-9 所示。



第三类标准B样条

节点	插值结果	真值
35.200000	-0.561651	-0.561652
27.600000	0.264997	0.264997
19.000000	1.075377	1.075376
65.700000	-0.468588	-0.468583
55.700000	-1.274849	-1.274850

图 7-9　第三类标准 B 样条

可以看到计算结果还是很不错的。至此已经介绍完三类标准 B 样条，在使用这三类 B 样条时需要注意各自的适用条件。

本章小结

本章介绍常见的插值方法，其中拉格朗日插值、牛顿插值、Hermite 插值是基础。本章重点强调了样条插值方法，分别介绍了三类条件下的三次样条插值及三类条件下的 B 样条插值方法。B 样条插值插值方法因其实用强，理论优美而应该给于一定的重视。除了本章介绍的插值方法之外，还有其他一些插值方法如有理插值等未作介绍，有兴趣的读者可以参阅相关资料或专著。