第一章　数值分析概述

§1.1　本章目的

这一章是对数值分析的一个简单介绍，我们将在这章中讨论有限精度的计算、误差分析、稳定性(robustness)和复杂度分析等问题。

§1.2　简　介

计算机使我们解决问题的能力大大超过数值分析方法本身解决问题的能力。有些问题，比如高次非线性问题，就不可能简单地进行分析处理，还有一些问题可能工作量太大或太麻烦。

无论如何都需要付出一定的代价。其中之一是，我们让计算机处理问题，会发觉它可能是朝着正确的解题步骤在执行，也可能沿着无效的或错误的假设在运算，这时就有可能解不出题。并且，计算机的精度也不是无限的。由于速度和存贮器容量的限制，也可能导致运算结果是不精确的。程序中的算法也不可能是"完美的"。因此了解一些计算过程中的状况是有价值的。

§1.3　有限精度计算

计算机与"理想的"分析之间的主要区别在于计算的数据是有精度的。我们假定科学与工程上使用浮点进行数值计算，并且假设读者已经熟悉这种表示数的方法，即一个尾数或小数部分乘以一个标准基数的幂指数。这样我们只需存贮尾数和指数就能够表示一大批相差巨大但又有相同小数精度的数值。虽然有各种各样的标准，但是标准的 C 语言使用 double(双精度)类型进行计算(最后把计算结果存放在浮点类型变量中)，它通常要占用 64 位来存贮并且它将提供达到 10^{-20}的计算精度，相似精度的变量就可用双精度类型来存贮。

一般来说，10^{-20}的精确度已经足够了。但是必须重视精度损失。当你还是学生在实验课上进行数值分析时，你可能会遇到"有效数规则(rules for significant figures)"的问题。一般地，乘法和除法不会引起精度损失，然而加法和减法则可能丢失精度。当我们减两个非常接近的数的时候，就有可能损失精度(有效数字)。假设我们取 3 位有效数字，那么 $1.00-0.99=0.001$ 就是一个在一步简单操作中如何损失精度的(2 位数字)典型例子。浮点数计算中的一个相似危险是这种状况：

$$1.00+10^{-3}=1.00,$$

这里没有精度损失，但是假设用户没有采取保护措施这个结果将是一个祸患。

最后需要说明的一点是，想一想解一元二次方程 $ax^2+bx+c=0$ 的问题。假设 $a,b>0$ 并且我们希望解出的根是 $x>0(c<0)$。那么

$$x=[-\sqrt{(b^2-4ac)}-b]/2a$$

是希望的答案吗？假设 $b=1$ 并且 a 非常小，那么 $x=[\sqrt{(1-4ac)}-1/2a$。当 ac 的值小时，将近似地得(根据泰勒级数平方根展式)$x=-c$(由于 $c<0$，那么 $x>0$)。如果 ac 的值很小，使 $1-4ac$ 近似地表示为 1，我们就会得到 $x=0$。但要注意在某些问题中 $|c|$ 可能非常大并且仍然使 $|ac|<1$，这样在 x 事实上非常大的情况下也会得出 $x=0$ 的结果！读者将在第六章牛顿——辛普森(Newton－Raphson)方法有关电离平衡的分析讨论中遇到这个问题。这样的问题会经常出现。

对这类问题的解决方法是分子、分母同乘以一个因子 $\sqrt{(b^2-4ac)}+b$，这样就得到：

$$x=-2c/(\sqrt{(b^2-4ac)}+b),$$

这个结果对 $|ac|$ 小时也是非常满意的答案。当然，它还会有其它的问题(例如 $b<0$)。在下面章节中要讲述的牛顿——辛普森方法是解该问题的另一种方法。

这儿阐明的精度损失通常被称为“舍入(roundsff)”和“截尾(truncation)”误差。这些术语涉及到计算机是如何进行计算的，假设我们计算 1.23×4.56，则我们对精确答案(5.6088)做截尾处理得 5.60 或做舍入(在截尾之前超过了 0.005，应舍入)得 5.61。通常，机器按照后者来处理将得到更为精确的结果，但这并不是说结果很精确。术语“截尾误差(trubcation error)”通常是由于在无限数(或连续小数)中只保留有限项，或有限次数的重复一个需要无限项数才能构成其精确答案的一个有限项数的数。

上面的例子大概是由于没有完美的算法，用来获得理所当然的结果。数值分析保持着这样的手段。

§1.4 误差分析

计算机输出的结果可能与我们所期望的结果有一定程度的差别，能够估计出究竟存在多大差别对我们是很有用的。

这儿有两种估计方法：前馈(forword)和后馈(backward)方法。前者试图告诉用户他们最希望了解的事：解题方法的不完整性所给出的答案的精确度是多小，这通常是非常困难的事。而后馈误差分析方法只要求回答这样一个问题：对给出的一个计算机解法，它究竟能够精确解决什么样的问题？这是一个很容易解决的问题。假设这个问题对你希望解答的问题来说在适当的条件下是接近的(close)，那么你就很容易了。所以，后馈误差分析方法经常被采用。

例如，假设你计算的 Z 值作为多项式 $p(x)-P(z)$ 的一个精确根。这儿 $P(z)$ 是当多项式 $P(x)$ 赋值为 Z 时所得的常量值。相比之下，这个问题的前馈分析是非常困难的。

很多问题的情况可能更糟。它们会引起被解答问题的微小变化(可能是由有限精度计算造成的)导致计算结果的很大变化。高次多项式求根就是这样一个问题。这儿有一些典

型的例子(其中一个将在下面的第七章中给出),它们使演示方程的一个系数的微小变化能够剧烈地改变根的性质。另一个例子是解线性系统 $Ax=b$,这儿矩阵 A 是一个小的行列式(另一种量度是"状态数(condition number)"——见第三章线性系统)。在这两种情况下,问题即是使用很好的数字分析方法也不容易解决,因为其困难在于问题本身,而和解决方法无关。

§1.5 稳健性(Robustness)

我们在这里引入"稳健性(robustness)"的问题,即获得一个好的解答的可能性(或当无解时我们至少能够确定无解)。一种可能是通常意义下的"防弹(bulletproof)"程序,特别指那些打算作为封装的库例程的程序,一个例子检查参数是否超出它们的有效范围。稳定性远远超过这个而寻求一种对误差反应比较迟钝的方法。例如,在线性系统的求解中,"支点(pivoting)"方案是一种逃避由小系数引起潜在问题的尝试。取部分支点而不是全部通常被用作效率和稳定性之间的一种折衷办法。这种权衡的方案在我们选择数值处理方法时经常存在。

某些提示可能适合这儿的稳定性问题。可能的话,应尽量避免高次多项式。在插值时,用低次多项式代替部分高次多项式(看后面第八章的样条拟合(spline fits)),或者用有理逼近。

通常,我们试图逃避有障碍的或情况糟糕的问题。而这些经常不能逃避的问题被称为所谓的"逆转(inverse)"问题。这些问题通常出现在积分转换中或者由散射信号确定散射体的特性中。这些散射信号通常是在散射分布上的一个积分。问题的坏条件是由于积分变换是一种平滑连续的操作。因此其反操作是一种非平滑连续操作(un—smoothing),因而它对声音信号特别敏感,这种声音指数字或者指从真实世界获得的代表数据。当其逆转问题可以近似地作为一个线性系统时,即 $Ax=b$,其结果通常是带一个弱条件数的矩阵 A(参看第三章)。

解决这些问题的最好方法是不去解糟糕情况下的系统 $Ax=b$。相反地是加强一些条件以避开不可解状态,而用最小二乘法去解决一个相关系统(如果最小平方误差不合适)。可取其中一些好的标准,通常的解决方法是把期望的解答写为带有不确定系数的大量项的迭加,这些系数是用所提供的对适当的最小平方问题能较好处理的单值分解方法(SVD)得到的解(参看第五章的SVD)。SVD 方法是处理最小平方问题非常稳健(robust)的一种方法。

还有解答逆转问题的方法,比如"过滤法(filtering)"或"规律核心法(regulerzing the kernel)"。这些方法经常是等价于上面讨论的最小平方方法。

§1.6 复杂性分析

无论使用的方法是如何的稳健,如何的好,我们都必须在我们的资源范围内解决这些问题。因此,采用的方法必须足够快,而且能有效地利用存贮器。对我们来说,更关心的是

前者。从这个原因来说复杂性分析是很有意义的。

有些方法明显是失败的，例如用克莱姆法则解决线性系统，在计算行列式时比使用高斯消元法花费大得多的开销，但是高斯消元法的结果通常精确度要差一点。

无论如何，必须使用复杂性分析。例如，两个 $N \times N$ 的矩阵相乘，每次需要 N^3 次浮点运算，假设我们定义这种操作（在这样的复杂度分析中是普遍的）是一次乘法（或除法）后跟一次加法（或减法）。有一种叫斯查生（Strassen）的方法，它通过存贮一些中间运算结果来减少浮点数的运算。然而，存贮和恢复这些中间结果的方法是很少有实际意义的。

复杂性分析通常用带连续性的"冯诺曼（vonneumann）"结构机制完成。当"管道（pipelined）"或向量机或阵列处理器或更高级的并行处理器被使用时，这些分析必须重新考虑。另一方面，潘（Pan）和赖夫（Reif）所建立的解决线性系统问题的交互式方法，需要每次重复不确定的 N 次系统使用 N^3 次操作指令，这明显是不能和连续机上使用的高斯消元法相比，后者解出答案只需前者的 1/3 次操作。然而，潘——赖夫方法允许在多处理器机上并行地迭代。而高斯消元法在处理下一行之前需要前一行操作所得到的结果，它不能并行执行。因此在某些机器上，潘——赖夫方法可能比高斯消元法更快完成。

当使用方法不确定时，即当数据特性决定需要重复多少次操作时，必须进步地进行练习。对最糟糕的情况分析可能是完全悲观的，但这只能在实验室中才能查到。如果不仔细查找，在真实世界中是很难发现它的。例如 R. E. Tarjan 发展了一种"缓冲（amortized）"复杂度的概念去恰当地估计最糟状况的可能性。他使用这种分析法能够有效地解释一些自稳健的数据结构。如"展开树（splaytree）"的一些精细的复杂度分析也能够避免。

要在算法稳健性与复杂度之间给出权衡，最常用的方法是在给定的资源范围内解决问题。这样就使复杂性保持了高水平；假设我们不能够使问题在计算机上及时完成，则不论解答是多么好也是无济于事的。假如最适用的方法是可行的或我们使用一种更为经济的方法，问题就可能被决定。我们在这里讲述的方法可能对所有给定类型的问题并不都是唯一一种最好的方法。在这本书中，我们希望把程序代码做为一种库例程来使用，高的开销应花在方法的稳健性上。

第二章　科学和工程C语言

§2.1　如何使用这本书

在这本书中包含有很多程序，它们能解决许多对科学家和工程师来说感兴趣的问题。其中一些程序，比如第十三章中的统计计算函数，可以用来准确地列出统计表。第十一章和第十二章中讲述的关于解决微分方程系统的程序，则需要对指定问题的"驱动(Driver)"例程做一些修改。其它的程序将做为一种工具合并到一些随时可调用的函数库中。

这本书给出所有程序的源代码。用户用它们能准确地估计他们的猜想会出现什么样的情况。没有什么能比程序的源代码更完整、更详细地描述算法了。

数值分析传统地用FORTRAN语言来完成。由于这个原因，在个人或公共函数库中也有大量对科学家和工程师来说感兴趣的FORTRAN源代码。在这本书中也有很多把FORTRAN代码转换成C代码的简单方法。第三章中包含有为这个目的而使用的大量有用的头文件，第七章中的一些头文件能使C语言中的复杂数学计算问题变得容易。

§2.2　为什么要使用C

没有一种计算机程序语言是完美的。FORTRAN做为最老的高级语言，是专门为科学和工程应用而设计的，并且它们继续在其老用户中流行。它被更新和标准化过许多次，至少有一种以上的版本在不久的将来就会被淘汰。

我们选择C语言有许多原因。首先，C语言能容易地在各种机器上使用。从个人计算机到超级计算机。无论在哪儿，只要UNIX操作系统(或其派生物如XENIX)所到之处，C语言也紧随其后(因为UNIX操作系统大部分是用C语言写的)。在当今存在的语言中，C大概是移植性最好的语言。也许Ada语言会摘走移植性的桂冠，但是现在的Ada语言并不是普遍可用的，并且Ada编译器也很昂贵。用C语言写的程序(兼顾程序的可移植性)只需作少量改动几乎就可在任何地方运行。可以发现，经常被调动的学者或职业程序员能够(或几乎能够)放心他们程序代码的可移植性。而移植FORTRAN程序则通常是一件非常困难的事情。

与FORTRAN或Pascal语言的情况不同，目前最流行的选择是C。IBM PC的FORTRAN编译器花费了数年时间才得以完成，这主要是由于FORTRAN语法的自由性。因为没有关键词和空格可以忽略，比如下面的语句：

DO11=1 和 DO 1 I=1,27,

它们是不易被区别的。使用词法分析器产生如LEX的程序，使用象RACC等编译器

(Compiler—Compillers)是不可能的，因为它对简单的组织语言记号无能为力。当然，FORTRAN 编译器可以用可移植的 C 语言来编写，但是为了达到可执行的效果，通常仍需付出适当的努力。它们与相似的个人计算机编译器比较而言也趋于昂贵。

Pascal 语言有一系列的困难，它的输入/输出操作，包括文件句柄操作是非标准的，因此它不具有好的移植性。任何时间程序的任何部分被改变后它都必须完全重新地被编译——否则编译目标代码库不能进行连接。Modula—2 语言的开发可以补偿人们对 Pascal 的抱怨，但是它并没有变成一种流行的替代品。

使用 C 语言的另一个原因是它与操作系统之间用一种自然的方法连接，并且通过操作系统成为硬件的延伸(在 FORTRAN 中也试图接近命令行参数(Command Line arguments))，这个特征导致更大的方便和节约时间，特别是假设程序是在"一次操作(batch)"模式下运行。非数值函数诸如文件操作、输入/输出函数、读取串口和总线(例如 IEEE—488 GPIB 或 VME 总线)数据通信以及实时应用很自然地要用 C 语言来编写。工程技术人员因特殊目的使用计算机，包括个人计算机来监视实验，在实时应用中从传感器获取数据并且通常立即对这些数据进行预处理。这种处理包括重要的数据"信息(massaging)"处理。使用 C 语言能使数值和非数值程序这两种形式花费最小的代价便可接口。本书中的一些例子是基本的非数值处理函数，枚举类型的预处理器就是这种程序中的一个，这对那些读者，他们的 C 编译器不支持枚举型数据(或为过程定义的"void"标识符)然而他们又希望使用这种带有枚举型变量的 C 程序来说是有价值的，对那些需要为某些应用程序做词法分析的读者来说也是有用的。

本书的某些 C 程序能处理通常 C 编译器不支持的递归算法。

C 预处理器通常能减少很多程序代码中的冗余部分。它的应用将在下面的"处理复杂的算术计算问题或把FORTRAN 程序转换为 C 程序"中介绍。C 预处理器是大有作用的，它通过用户定义自己的算符，依赖于不同标志的设置有条件地用相同的符号代替不同的代码等方法使 C 语言具有很好的"延展性(extensible)"。FORTRAN 则不能这样。

尽管 C 是当今存在的可移植性语言中性能最好的，但是完美的可移植性是没有的。这本书中包含的绝大部分程序能在很多机器和编译器上运行。通常，DeSmet C 应用在 IBM PC 兼容机上，而 Aztec C 用在 kaypro Ⅱ CP/M 机器上。C 语言的很多程序都避开了 C 编译器的特性并且通常不需修改就几乎能在所有的环境下运行。做为一个例子，while()形式的 while 循环语句将引起无限循环，Aztec C 版本 1.05g(为 CP/M 机器设计)把这条语句当作 while(0)语句来对待，完全不执行这个循环！因此，你将看见这本书中的代码中使用这样的语句，如 while(1)或甚至 while(1==1)。做为另外一个例子，DeSmet C 需要一个 return(x)语句来结束函数，这儿 x 是一个和函数型相同的变量。你也会发现这本书所包含的某些程序中，这种形式的语句是不能执行的(unreachable)，但是它必须是符合 C 编译器的。典型地，假如 C 程序被写成一种可移植性的程序并且避开了机器的依赖性和编译器特性，则唯一需要改变的是有关标准库中的函数。从 Aztec C 转换成 DeSmet C 通常需要在 libc.h 和 math.h 头文件中移去 #include 说明。这儿也需要一个 #define 语句为 EOF 定义并且需要一个"遗漏(missing)"函数，该函数存在于 Aztec 库中而在 DeSmet 库中则没有。库函数中的大部分例程 C 程序都能调用。库函数在不同的机器上以不同方式

运行可能引起预想不到的结果。一些程序使用不同的 EOF 定义(－1 是有用的返回值,但是在某些环境中从 getahar()函数返回一个 0 值做为文件结束标志),或在不同的方式下处理文件,这些问题都不会影响这本书中的程序。

总之,不可能有理想的语言。正如其它语言一样,C 语言也有它的优缺点。例如它并不是专门为科学家和工程师进行大规模数字分析的应用而设计的,它更适合于系统程序员和关心操作系统的读者。在这些问题的应用中它也有一些缺点,这本书将尝试改进这些缺点。它将作为一种指南为那些寻找这条道路的人服务,读者可能会发现用本书的程序就能铺盖成这条路。

表 2－1 总括地比较了 C 和 FORTRAN 的优点,科学家和工程师可以参考这些优点。

表 2－1　C 和 FORTRAN 的优点

C 长处	FORTRAN 长处
系统接口	COMPLEX 数据类型(复数类型)
更好地递归支持	方便的名字表(NAMELIST)输入
预处理器	用户控制精度
可移植性	优化编译
使用命令行	

§2.3　用 C 处理数字(Number crunching with C)

这本书的重点是在数值分析方法上。对系统程序员感兴趣的 C 方面的问题,以及管理、屏幕处理方面的问题已有很多书介绍了。科学家和工程师们需要大量包含数值分析方面的应用,他们可能对尝试着翻译这样的代码为 C 语言感兴趣。直接翻译 FORTRAN 代码到 C 代码是很重要的,这本书中的程序可以做为示范帮助那些想作类似转换的人服务。

一个可能的例子是执行复杂的算术计算方面的问题,FORTRAN 有 COMPLEX(复数)数据类型,这对电气工程师来说是很感兴趣的。Ada 允许定义一个复数(Complex－data)数据类型。在 Pascal 语言中,必须使用一些笨拙的子例程来完成复数计算。在 C 语言中,可以使用预处理器来对复数进行有效的计算(他被"在线(in－line)"执行而不需要子例程调用)。本书中有关复数计算的程序可以对那些在他们自己的 C 代码中需要复数运算的人来说也是有用的。

C 语言是在贝尔实验室中为使用 PDP—11 机而研制的。这就使 C 语言具有其它语言所没有的特征。比如下面形式的语句:

```
i++;
```

它被直接地编译成一条 PDP—11 机上的指令(假设 i 是一个寄存器整型变量)。因为这个以及其它特征,C 被称为一种"只写(write－Only)"语言。读别人的源代码是很困难的

而明白将要发生的事却是很容易的。本书的程序趋于自由地加上注释，也可以在文本中进行描述。变量名能够比在FORTRAN 中做更多地说明。循环变量通常被定义为 i, j, k 等。为了保持FORTRAN 的习惯，也希望为必须被输入(to be typed)多次的索引名避开一些长名字。书中的整型变量不象通常的 FORTRAN 一样以 i 到 n 开始。

C 语言允许递归算法，但在数值分析中并不经常使用。当函数被递归地定义时，它通常是一个简单的"尾结束(tail－end)"递归，它能由已知有效的计算方法转换成相对容易的循环语句。递归对实现下面所讨论的适应性求面积程序来说比较方便。

§2.4 转换 FORTRAN 程序为 C 程序

很多数值分析方法都基于线性代数的知识，这包括解联立方程系统的矢量和矩阵代数知识。FORTRAN 和 Pasal 语言都支持调用矢量 A 中的第一个元素为 $A(1)$的惯例。在 C 中，第一个元素是 $A[0]$。假设数据有 20 个元素，我们将有下面定义的语句：

```
float A[20];
```

数组的最后一个元素将是 $A[19]$。矩阵可能有更多的问题，FORTRAN 允许矩阵(2 维数组)作为一个参数传递给子例程，维数也做为参数：

```
SUBROUTINE X(A,N);
DIMENSION A(N,N);
```

反之 C 语言则没有这种方便。我们必须为数组传递一个指针做为参数：

```
x(a,n)
float *a;
int n;
```

这是基于我们使用维数 n 的值作为下标。注意一个矢量可能这样定义：

```
float *v      或   float v[ ];
```

但是C 不允许我们传递数组维数作为参数。只有在这样一种方式下我们才可能把数组当做一个矩阵来使用，例如 $a[5][7]$。我们将在下面的例子中演示如何使用 C 预处理器来消除这种负担，预处理器也能够使所写的循环容易化。

在 C 和 FORTRAN 中，矩阵按不同的方法存贮和访问。在 FORTRAN 中，它们是按列存贮，而在 C 中则按行存贮。在 FORTRAN 中，存贮的第一个元素是 $A(1,1)$，接下来是 $A(2,1)$等等，反之在 C 中第一个元素是 $a[0][0]$而跟在其后的元素是 $a[0][1]$。这是非常重要的。在巨型机上对特殊问题都包含有大矩阵。在 C 语言中是按行进行行操作，因为行中的下一个元素也是存贮地址空间中的下一个元素。在 FORTRAN 中是按列操作。有时，为处理大矩阵而特别设计的 FORTRAN 算法按列存贮矩阵能够最小化(minimize)从第二次存贮转变来的数据。如果谁要处理大矩阵，这些都必须考虑。假设正在使用一台微型机，除非使用大矩阵否则会忘记它们。

大多数 FORTRAN 和 Pascal 程序转换成 C 程序是相当直接的。例如，在 FORTRAN 中 DO 循环变量必须是递增的，需要某些固定的结构来实现循环变量的递减。我们将示范两种转换方法，本章将停留在非常接近于 FORTRAN 的方法上面，而第四章中将阐述另

一种方法。上面提到的是用预处理器来帮助实现转换和处理复数变量与矩阵的方法。除了不同的记录变量以外，Pascal 翻译成 C 相对容易。

§2.5 Debug C 程序的 8 点提示

在 C 源代码中通常有陷阱，对无经验的 FORTRAN 程序员来说，应注意以下几点提示(很多是与函数中的参数与子例程调用有关)：

1. 数组变量从 0 开始

记住一个定义为 array[10]的数组是按从 array[0]到 array[9]来扩展的；试图访问 array[10]会发生不可预知的结果。回忆一下在 C 中的多维数组和在 FORTRAN 或 Pascal 中使用的也大不相同，而多维数组的相似性在 C 和 FORTRAN 中也大不一样。在 FORTRAN 中，两维数组的头两个元素是 $a(1,1)$和 $a(2,1)$，相反 C 中它们是 $a[0][0]$和 $a[0][1]$。

2. 函数参数不能够被调用的子程序改变

记住函数参数是按值传递，这个参数被赋值并且这个值被压栈。不要企图修改这些值或通过它们来传递结果。假设被调用函数需要返回这个变量的某些信息到父进程，则必须使用指针。

3. 在描述相同目标时可能需要用到不同的语法

传递给函数的参数在发送程序和调用程序中作不同的描述，简单的例子是整型数，描述如下：

```
int i;
```

在一个程序中它传递一个指针到子例程中去，如：

```
subr(&i);
```

然后它被描述为：

```
subr(i) int *i;
```

通过以下的赋值语句传递整数值 7 到 i：

```
*i=7;
```

假如调用的子例程有这样一个语句 $i=7$；则将产生不可预知的或是灾难性的结果。

假如一个函数名被做为一个参数来传递，则这样写：

```
subr(f)     float(*f)();
```

那么我们就知道有下面这样的用法：

```
x=(*f)(z);
```

在子例程序中 subr 将设置 x 为一个浮点值。程序调用将使用下面形式的描述：

```
float f();
```

并且用 subr(f)调用这个子例程。

假如按以下形式调用程序，将再一次产生灾难性的结果：

```
float (*f)();
```

记住当一个下标被应用时，间接的等级(level of indirection)可能被遗失。那就是，当我们按

下面形式定义时：

```
float array[10];
```

在调用程序中，array[i]将是一个浮点数。除非数组在 subr(array)中是一个指针，它将被子例程按下面的形式定义：

```
subr(array)float array[];
```

这个子例程可以把数组当做一个指针来对待(例如，传递数组到后面的子例程中去)，或者在 array[5]=1.0 中做为一个浮点数。做为一个更为复杂的例子，考虑到结构数组：

```
struct complex a[10];
```

在下面的这种调用中指针被再一次传递：

```
subr(a);
```

用调用的子例程来说明：

```
sub(a) struct complex a[];
```

在这个子例程中，变量将是一个结构数组的指针。而 a[3]将是一个结构元素，没有指针指向这个结构中的第 4 个(FOURTH)元素；为了取得指针我们需要用到 &(a[3])。(我们大写单词"forth"就是为了提醒你在 C 语言中数组的下标变量是从 0 开始而不是从 FORTRAN 中的 1。)，在关于 Muller's 方法一章中包含有这里所讨论的传递结构的例子。

4. 指针和数组定义是不可交换的

定义：

```
float array[];
```

和

```
float *array;
```

是不可交换的，即使它们两个都定义为浮点型的变量数组和指针变量。后者可能引起问题，除非它是一个函数的参数。也注意这种情况：int a[10][10]和 int *a[10]的效果是非常不同的。

5. 可能的陷阱

这儿有很多难以捉摸的陷阱。例如，你所期望的做为程序代码的结果：

```
x[i]=i++;
```

C 语言不能强制编译器对函数赋值的顺序，或者确定 x[i]定位是在 i++语句中 i 增长之前还是增长之后。

6. getc()和 getchar()返回一个整型值，而不是字符

记住 getc()和 getchar()函数，尽管想要得到一个字符输入，但都是返回一个整型值而不是字符。当文件尾遇到它们时这点很重要。

7. 子程序将忘记在两次调用之间的局部变量值

与"老(old)"的 FORTRAN 不同，C 的子程序将遗失在两次调用之间的局部变量值。在最近的 FORTRAN 标准中这种遗失也发生，尽管你能够通过 FORTRAN 的 SAVE 语句强制它记住这些局部变量。在 C 中，通过定义局部变量为静态变量或把它们作为全局变量也能达到相同的效果。

8. scanf()函数的不可靠性

scanf()函数可能是不可靠的。例如，它通常不读取双精度(double)型的值。总是“回响(echo)”这种输入的结果以确信程序接受到正确的值。做为一个“友好用户（user－friendly)”的提示，回响输入允许用户确认正确的选择或在某一点重新开始。尽管为每一个输入参数做这个工作可能很快使人疲劳。假如前面的参数可能被做为默认值而为将来的输入再使用，即记住前面输入的参数是很有用的。在下面将讨论为统计函数使用的“袖珍计算器(pocbet calculator)”中将用到这个特征。

§2.6 程序选择

本书提供的程序主要是为了应用，并且提供了示范值。试验真实(Tried－and－true)程序已经被应用在尽可能可靠的地方。

通常，所选择的算法是一种艺术尝试，且具有最大稳健性和适用性。这些算法简洁紧凑并且执行效果非常好。尽管有些人可能会认为他们所喜爱的方法比这儿列举的方法更好，其实在这儿所列举的方法和其它方法一样都是解决这些问题的最稳健最有用的方法。

第三章　矩阵的LU(下三角阵和上三角阵)分解

§3.1　本章目的

这章(主要)包含下列工具(内容):进行一般矩阵的LU分解;解齐次线性方程组;求一个矩阵的逆矩阵;计算一个矩阵的行列式的值;使含有矩阵的一个Fortran程序较容易地转化成有同等作用的C程序。

在这一章里,以上这些方法(工具)都将用于解一个简单线性系统。这些方法也适用于后面各章中出现的这样系统,如第十二章中的隐含刚性的微分解程系统的解法以及第六章的牛顿——辛普森(Newton－Raphson)解法。

作为诸如"Galerkin方法"或者"矩方法"的应用结果,线性方程组出现在很多问题中(或很多实际问题都涉及到线性系统)。下面将讨论使用这些方法可以得到什么样的线性系统。

§3.2　齐次线性方程组

最常遇到的数值问题之一是解一组齐次线性方程。这个问题可以作为另一个问题的一部分,例如,在解刚性的微分方程组中,我们将使用这儿给出的子程序。我们来解以下给出的实例:

$$\begin{aligned} 2X+3Y-Z &=5,\\ 4X+4Y-3Z &=3,\\ -2X+3Y-Z &=1. \end{aligned}$$

这个齐次线性方程组的解是$X=1,Y=2,Z=3$.这个问题若用线性代数中的$Ax=b$方法来解,那么,A是矩阵,可以表达如下:

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

而b是向量(5,3,1),向量x包含了前面方程组中表示的未知数x,y,z.

现在,我们使用C语言中的自然记号,将这些未知数表示成$x[0],x[1]$和$x[2]$。类似地,$A[0][0]$表示矩阵A中的元素2,而$A[2][0]=-2$。对于A和b而言,这些值是选定了的,因为他们给出了一个整数解向量,这个解向量能够用于检查一个机器的精度。我们也将使用这个矩阵来说明矩阵的转置,并且解含有转置矩阵$A^{\wedge}$的线性系统(矩阵的转置仍然是矩阵,而A的所有行都已变成了转置矩阵$A^{\wedge}$相应的所有列,即$A[i][j]=A^{\wedge}[j][i]$)。

§3.3 怎样提出线性系统

§3.3.1 直接产生线性系统问题

齐次线性方程组常常出现于科学和工程问题中。很常见地，一个物理系统的一个平衡或稳定状态可以通过一组线性方程来描述。这里给出一个例子：一条 AC 或者 DC 电子线路。我们感兴趣的主要变量是每个分支的电流和每个结点(分支相遇的地方)的电压。在电子线路处于平稳状态下，所有这些变量都可通过基尔霍夫(Kirchhoff)定律和欧姆定律联系起来。对于 AC 线路的平稳状态，类似的方程组将具有更复杂的系数。有多种方法可以得到一组实际可解的线性方程组，比如用结点，分支割集等公式表示。这些公式的不同表示是为了解这些公式，公式中的变量被选定为独立变量。一些公式的描述给出了具有很多变量和方程的系统，但这些公式描述的系统仍然是稀疏的，因为在这样的系统中很多变量的系数是零。一些公式不能够用于处理一般分支问题，比如在串联中，一个电池的电阻是零。图 3—1 中给出了一个有代表性的简单电路，此线路是带有电池和电阻器的一条 DC 线路，我们使用了基尔霍夫法或者分支分析法。按照基尔霍夫电流定律，$I[3]=I[1]+I[2]$。根据他的电压定律，由于电流经过电阻器，使得电压逐渐变小(下降)。经过任何回路的电压必须是零，根据欧姆定律，电路中的电压降等于电流和电阻的乘积。因此，我们得到求解的系统如下：

$$\begin{pmatrix} -1 & 1 & 1 \\ R[1] & 0 & R[3] \\ 0 & R[2] & R[3] \end{pmatrix} \begin{pmatrix} I[2] \\ I[2] \\ I[3] \end{pmatrix} = \begin{pmatrix} 0 \\ V[1] \\ V[2] \end{pmatrix}$$

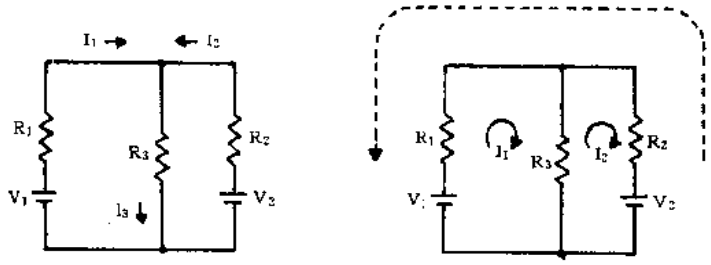
在一种叫做回路或麦克斯维(尔)(Maxwell)分析的可供选择的方法中，我们规定图形中标出的第二半部分表电流。电流与电路中的回路有关，而与电路中分支无关，并且任何分支可以携带许多回路中的电流。基尔霍夫电流定律自然满足，而我们只能写出两个方程(假定 $I[3]=0$)：

$$\begin{pmatrix} (R[1]+R[3]) & -R[3] \\ -P[3] & (P[2]+P[3]) \end{pmatrix} \begin{pmatrix} I[1] \\ I[2] \end{pmatrix} = \begin{pmatrix} V[1] \\ V[2] \end{pmatrix}$$

注意到对公式的权衡：虽然第一个矩阵是稀疏矩阵，但矩阵是 3×3 的，而第二个矩阵是稠密阵(非零)，但它是 2×2 矩阵。线性系统来源的又一个例子是建筑机械学(结构力学)，在任何点的力和力矩必须相互抵消变成零。于是就产生了这样一个问题。这个问题就是与通过建筑组件传递的力和力矩的分向量有关的每一点都涉及到一个带约束条件的方程组。如果这个问题有 N 个带约束条件的方程和 N 个未知数，那么原问题就归结为解一齐次线性方程组的问题。这样的一个结构称为静态确定的。有时(常存在于实际问题中)，结构又是"静态不确定的"，并且根据对杠杆的弹性性质的确定，还必须选出另外的带约束条件的方程。

这个问题转化为对每一根杠杆的应力解一组线性方程。例如，在有代表性的"构架"问题中，所有杠杆都连接着铰链或者槽针，以致力矩在任何点都自然为零。通常说来，这个构架的一端被滚轴支撑，以致在这一端没有水平作用力(强制力)，从而使整个结是静态确定

的。当然，在现实领域中，杠杆通常是悬臂梁(这条杠杆的方向以及端点被固定)，并且接合处没有装上铰链。因此，比起简单化的课本上提出的问题来，我们更常常不得不解齐次系统问题。



左边为分支分析　　右边为回路分支变量

图 3—1　简单电路分支分析

§3.3.2　需要用线性系统的解作为一中间步骤的问题

用另外的方式得到的线性系统是对连续问题的近似处理，如利用这种方法求积分和微分方程的解。有很多较好的数值方法，如象“加权残余方法”、“矩方法”、“加莱尔金(Galerkin)法”、“配置法”、“雷利——李茨(Rayleigh—Ritz)法”等。这些方法都有一个共同的核心思想，即展开一个或几个未知函数作为已知函数的一个线性叠加。在展开式中的未知系数是可以被解出的。当原问题本身是可以解的线性系统时，对于这些系数，上面提到的那些方法给出了一齐次线性方程组。作为两个例子，这样的线性问题频繁地出现在有趣的弹性和天线理论中，这样的方法数值上等效于“待定系数法”(这个述语一般应用于分析法)。

上述方法的区别在于怎样选定展开函数和确定展开函数中的系数，怎样由这些函数和原微分方程和(或)积分方程生成以展开函数中得到的系数为系数的方程，因为它最容易通过一个具体例子加以解释，所以我们考虑解除一个含有值域成为 OZ 的单个独立变量的函数 $F(Z)$ 的问题。我们假定 $F(Z)$ 被展开成 $F(Z)=\sum_{i=1}^{N}c(i)f(i)(z)$ 的形式。另外，我们引进函数 $W[i](z)$ 的另一种集合是有用的。这里 $W[i](z)$ 是加权函数。现假定，可解的方程的形式是 $E(F)=0$。我们通过依次乘以 N 个 $W[i]$ 形成了 N 个方程。然后，我们在 Z 的允许区间内积分。

在配置方法中，这个 $W[i]$ 被选定成形式是 $\delta(Z-Z[i])$ 狄拉克 δ 函数。因此，相应的积分导致出估算 F 在这个配置点 $Z[i]$ 的值。我们限制函数 F 的近似，而函数 F 满足在精选出的点处的方程。显然这种配置方法可能是危险的，也就是说，如果那些点选得糟糕，那么由这种配置方法将给出很差的结果。当然，此方法却是最经济的方法之一。

如果在展开函数形式 $\sum_{i=1}^{N}c[i]f[i](z)$ 中，加上 $W[i]=f[i]$ 条件，便给出了加莱尔金

(Galerkin)法。在我们所举的例子中总是希望$f[i]$满足边界条件(如Z=0和Z=1)。否则，我们将不得不把这些边界条件看作配置点或者通过一些其它的方法来解方程，$f[i]$常从一些完全集，例如，$\sin nz$和$\cos nz$，$n=0,1,\cdots\cdots,M$中选定。实际上，这种选法比人们可能想到的方法更微妙。由于依赖于F是什么算子，这样的一个"完全集"不可能估量这个算子的值域，并且方程的解可能丢失。因此，对$f[i]$和$w[i]$的选择应有一些了解，这些了解就是方程的解应该具有什么形式。对$f[i]$的明智选择能给出恰有n个展开项的极好结果。例如，在天线理论中，众所周知的是：一条正弦曲线电流分布是对薄天线的良好近似，以及正弦曲线分布常常用于矩方法计算。

当被解的系统是自伴的时，雷利——李茨(Rayleigh－Ritz)展开可以看成是Galerkin法的一种特殊情形。即使系统不是自伴的，伴函数也常常可以确定，并且这些伴函数可以用于加权函数中。

加权残差或矩方法的各种形式被总结在表3－1中。在已知展开函数目的条件下，人们可以生成任何数量的方程，例如，配置点数M可能被选定为大于数N。如果M超过N，那么有比未知数更多的方程数目，以致并非所有方程都能同时满足方程的解。对于这种情形，最适当的方法是最小二乘法。此方法是使方程的残差平方和达到最小(在选择$M<N$时，没有明显的根据，M在方程组有解时可能给出一个无穷数。)，最小二乘法将在下面讨论。当讨论了奇值分解法后，最小二乘法在计算上比一个可比较大小的线性系统的直接解答更重要。因此，在多数情况下，解方程组最好使用$M=N$。

表3－1　加权残差方法或矩方法形式

名称	加权函数
配置	δ函数
Gaterkin	相同于展开函数
Rayleigh－Ritz	伴展开函数

表3－2列出了很多在科学工程中感兴趣的方法，这些方法可能出现于解线性系统中。

表3－2　如何生成线性系统

直接地：	间接地：
平衡或平稳状态问题； 在结构力学中的静态问题； 电子线路。	连续问题的离散化； 加权残差法(MWR)和相关方法； 矩方法(又名MWR)； 待定系数方法。
配置法：	数据调整：
加莱尔金法(Galerkin)； 雷利——李茨法； 刚性普通微分方程(在第十二章中所要涉及到的方法)； 偏微分方程； 边界值问题。	例如，B－sphines(第八章中)。

最后线性方程组能将连续微分算子离散为有限差分算子，这样生成的系统几乎全部有一种稀疏以及重复的结构，采取有利于这个结构的补救措施可能是很重要(实在)的。

以上给出的解线性系统的通常方法将作为后面许多章内容的工具，刚性系统解法和B－sphine惯用法也可以用来解线性系统。

§3.4 怎样解线性系统

§3.4.1 *LU* 因子分解

什么是解一般线性系统的最好方法？有两种方法："直接的"方法和迭代的方法。迭代方法有临时添加补充值，也就是说，不考虑稀疏性。一般地，当我们使用直接方法时，原来已经是零矩阵的元素要变成非零，这会很容易地失去最初矩阵的稀疏性而具有的任何好处。

目前，对于具有当代结构的计算机来说，拥有部分元的 *LU* 分解是一合意选择。在以下讨论 *QR* 因子分解时，我们将看到，这种方法有利于并联阵列计算机的使用。

LU 因子分解(通常称为"分解")同高斯消去法有很密切关系。在高斯消去法中，我们选取问题的形式为：

$$Ax=b$$

其中，b 是右边的向量，x 是未知的解向量，而 A 是一矩阵，并且我们把 $Ax=b$ 转换成形式为：

$$Ux=b'$$

其中，U 是上三角矩阵，也就是说对角线以下的元素全为零：

$$\begin{bmatrix} u(1,1) & u(1,2) \\ 0 & u(2,2) \\ 0 & 0 \end{bmatrix}$$

这儿，类似于FORTRAN惯例，我们已使用了标准的数学记号，在上面的表示中，称 $u(1,1)$ 为第一行的第一个元素等等。例如，在C语言中 $u(1,1)$ 将被表示为 $u[0][0]$。这个变换可以通过使用"初等行运算(elementary row operations)"来实现。我们用任意希望的元素乘以这个矩阵的一行(并且乘以向量 b 中对应于这行的元素)，并且从这一行的下一行中减去行乘以希望元素后的结果，使下一行变成0。我们通常(习惯)使矩阵的第一行或第一列变成零。接着，可以用第二行(它的第一个元素是零)来消去第二行后的诸行中处于第二列的元素，等等。这样生成的系统 $Ux=b'$ 是容易解的。方程中最后一个变量 $X[n]$ 是方程 $u[n][n]x[n][n]=b'[n]$ 的解，其值是 $X[n]=b'[n]/u[n][n]$。在这个方程以上紧接着的是包括变量 $x[n-1]$ 和 $x[n]$ 的方程，现 $x[n]$ 已知，因此解这个方程也是简单的。因此，我们可以连续向上进行这样的解法。这种解法可能有一个缺点，当我们已经得到矩阵变换的第 m 行时，称这行处在对角线的元素为"枢轴元"，它的值可能小(按绝对值)或者是零。我们用这个枢轴元来消去矩阵第 m 列的余部(第 m 行后的各行位于第 m 列的所有元素)。如果 m 行的这个枢轴元是零，那么为了消除第 m 列的其它元素，用第 m 行后的各行

分别减去第 m 行乘以某一个希望的常数后的结果中，可能有有限个零。一个简单的解法是交换矩阵的这行和这行后的任何一行，同样对于 b 中的元素，也作相应的交换。这样的选择是挑选具有最大数量值元素的行进行反复交换。通过消去变换后得到的矩阵的第一行。这种方法被称之为"按行部分元"法。我们也能交换矩阵的列，当然，为了处在枢轴位置上的数是最大元素，对 X 中的变量也要实行相应的交换，以及对于这种方法我们可以使用"完全"元的概念。为了找到这个最大元素，矩阵的行和列都要被仔细检查，并且把这个元素作为枢轴元。由于改变变量的次序是一种过多的超额负担，因此，这种做法几乎从未算过；这儿作为叙述，部分元一般来说已经足够了。

我们考虑下面的模型，模型矩阵的枢轴元存在(并且，根据这个例子，枢轴元被检验)。因为 $A[I][0]=4$ 是大于第一列中任何一个元素的，所以这儿给出的矩阵 A 的"满的"LU 因子被找到：

$$A=\begin{pmatrix}4 & 4 & -3\\ -.5 & 4 & -2.5\\ .5 & 2 & 1\end{pmatrix}.$$

这个矩阵的 L 和 U 分别是：

$$U=\begin{pmatrix}4 & 4 & -3\\ 0 & 4 & -2.5\\ 0 & 0 & 1\end{pmatrix}L=\begin{pmatrix}1 & 0 & 0\\ -.5 & 1 & 0\\ .5 & .2 & 1\end{pmatrix}.$$

这个 LU 分解一点也没涉及到 b。这个分解起到了把矩阵 A 因子分解为 LU 乘积的同样作用，其中 U 是以上描述的上三角阵，L 是主对角元素全为 1 的下三角阵。这个因子分解得出的结果是正确的，也就是说，除了用于保持元交换轨迹的一个整数阵列外，不需要任何另外的存储空间。

已知系统 $LUx=b$，我们应首先解系统 $Ly=b$，然后再解 $Ux=y$。后者相同于高期消去法中的自后代换阶段。前者象后者一样，除了它继续进行从上到下的代换外。

LU 因子分解的一个极大好处是：一旦 LU 因子都已经得到，对于不同的向量 b，不需要对系统进行重新考虑便可以解因 b 变化得到的系统。LU 因子分解的计算花费近似于阶数为 $\frac{n^3}{3}$ 浮点运算(每一个浮点运算采用乘法，而乘法是借助加法来实现的。通常，乘法较之加法花的时间多，尽管在某些硬件中这条规则有可能例外)。置后代换的运算代价(指运算的快慢程度)是阶数为 $\frac{n^2}{2}$ 的浮点运算。如果有多重右边向量，其余方法的完善可能是重要的。

§3.4.2 逆矩阵与高斯——约当方法

令人感到惊奇，在很多书中常提到高斯——约当方法。高斯——约当法使用初等行运算既可消去主对角线上的元素又可消去主对角线以下的元素，对于系统 $Dx=b'$，在已知对角矩阵 D 的条件下，这个系统可能容易求解。因为这种方法需要 n^3 次运算，因此它的花费是更昂贵，并且更容易出现数值误差。这种方法的确产生了矩阵 A 的逆。然而，这种方法几乎没有用。如果有多重右边向量，使用置后代换的计算比用矩阵 A 的逆乘以 b 得到

系统的解答所需费用更便宜。如果矩阵 A 的逆已求出，那么 LU 分解同样可以象原来那样处理。这样做的方法之一是取 $b=(1,0,\cdots\cdots)$ 并且对 X 求解。X 就是这个逆矩阵的第一列。逆矩阵的第二列是根据取 $b=(0,1,0,\cdots\cdots)$ 得到，等等。这种方法比必要的方法在计算上更加昂贵，由于它没有考虑到在所有情形下 b 的特别简单的结构。这些情形就是 b 是 0 向量的所有乘法运算和相继的运算都可以省略，b 是 1 向量时的所有乘法运算也是不必要的。考虑到这些特征的计算逆矩阵的最优方法应当同高斯——约当方法一样的经济，并且在处理过程中有更少的数值计算困难，我们以上说过多次，对于某一类问题，有多种可比的计算方法，在所有情况下，没有一种计算方法明显优于另一种计算方法。可是，这并没有消除那些在所有情况下都明显的差而永不应该被使用的方法。高斯——约当法显然是这样一种方法，而用克拉姆规则来计算许多行列式的值则是另外一种方法。

以下列出的 C 源程序表明了解 $n=3$ 的一简单检验问题。现给出包含矩阵 L 和 U 的矩阵。依照置后代换，解向量也被打印出。虽然解答写在 b 向量内容的上面，但如同打印出的结果表明的那样 LU 因子并没有改变。最后，计算矩阵的逆矩阵，这个结果写在因子 LU 的上面，仔细注意到原来的矩阵 A 现包含两个矩阵 L 和 U。虽然 LU 不是一个新矩阵，但却是两个矩阵。矩阵方程 $A=LU$ 成立（应当记住：矩阵的元可能已经出现，从而导致行的互换）。的确，因子分解经常写成 $A=PLU$ 的形式，其中 P 是一个置换阵，它的元素要么是 0，要么是 1，并且每一行或每一列都有一个非零的元素。

§3.4.3 行列式

矩阵 A 的行列式是矩阵 L 和 U 的行列式的积。因为任意一个三角矩阵的行列式之值为其对角线元素的乘积。因此，在 §3.4.1 中的 L 的行列式的值是 1。我们能容易地计算行列式的值和出现在矩阵 A 中的对角线元素的乘积。然而，因为粗心大意，你可能陷入困境中：我们交换了矩阵的行，而这会改变行列式值的符号。要修改这一点是不困难的。我们只需计算不等于数 k 的元[k]的数目，便可知道在矩阵中已发生了多少次这样的交换。如果结果是奇数，那么我们便用矩阵的对角线元素之积去乘以 -1，便得到互换行后产生的矩阵的行列式的值。

这里，为了计算行列式的值，两种方案被包含进来。我们通过决定 $a\times 10^n$ 来产生行列式，其中 n 是整数，而 $1<|a|<10$。如果行列式的值很大或很小，那么这个形式是有用的。

§3.4.4 条件数

一般数值分析书讨论称为“条件数”的参数，是用于估计解答的精确程序。如果系统的情况是病态的，即假设小的枢轴元不得不使用，那么条件数会告诉我们：它是一个很悲观的估计。由于条件数没有办法考虑到向量 b 的结构，所以必须假定它自身可能是最坏的情形。为了分析电流源，一条电路分支不得不模仿为有一并联电路的电源，此电路有很大的电阻。当电路中的那些大的电阻足够大时，它们对系统的答案才有影响，但系统中的条件数主要受电路中非电阻值选择的影响。如果由 LU 因子解得到的系统意味条件数不是零，那么变量信息表明系统处理麻烦。如果我们希望，那么行列式的值也可计算，并且如果行列式的值小，那么就表明系统处理困难。一般地，对于科学家和工程师感兴趣的大多数问

题而言，零或者糟糕的系统将系统译成电码时有较大误差，而不是系统本身真正是糟糕的。由加权残数方法而引出的系统可能意味着对配置点或者展开函数或者加权函数的选择都是粗劣的。如果两个配置点太靠近或者两个配置点不能给出系统解答的独立信息，那么由于系统中的两个方程(行)将很类似，因而生成的系统的可能是极坏的。

§3.5 把FORTRAN程序转化为C程序的工具

头文件FTOC.H中有许多以前信息处理机的定义。这个头文件的构思容易将FORTRAN代码翻译或C语言代码。使用这些定义的目的是为了容易翻译含有阵列的Do loop结构和把矩阵从FORTRAN语言转换成C语言。

解线性系统的方法，包括 LU 因子分解，置后代换和求矩阵的逆，都是基于Forsythe，Moler和Malcolm给出的LINPACK程序。

对于行和列的运算，他们给出的程序使用了某些辅助程序，这些辅助程序来自于R. Hanson的Basic线性代数软件包(BLAS)。在这儿，所有这些程序都已被简化。例如，BLAS程序使用"展开"循环方法，而这儿并没有使用。展开是只有计算机有一条"管道"或者"向量"结构时才有用，并且编辑者不足以精明到很好地向量化循环套。对于某些编辑者，展开将限制最优化，这个BLAS程序是包含在LUS.C文件中。

除了LINPACK程序计算行列式不使用浮点运算以外，我们可以用类似计算行列式的第2种方法通过LINPACK程序来计算行列式。在假定其它所有条件相同的情况下，这儿介绍的C程序计算行列式的值应该是最快的。主要变化已成了解释C的向量存储的不同之处(第一个下标已经是0，而不是1)，预处理#define语句的使用简化了代码表示。$a[i][i]$单元是$a[i+coln*i]$，其中$i=0,\cdots\cdots$(rown(行数)-1)，j$=0,1,\cdots\cdots$(coln(行数)-1)。这种做法是根据函数指标来进行模拟。多种协调一致的函数DOFOR，DOBY等等，简化了循环套的代码表示。例如，FORTKAN循环产生了象下列语句的函数：

Do 27 $I=1,N$

此语句指明对于下标为1到N的某个向量(或者是一行向量或者是一列向量)。它是很容易映射成C上的具有下列形式的一个循环：

```
for (i=0;i<n;i++)
{
```

我们使用预处理#define语句以产生能形成上面那个循环的

```
DOFOR(i,n)
{
```

而其它的语句处理增量不是1的情况(通常，当语句的增量是行数(coln)时，我们用C的命令约定，对矩阵的行数进行循环)。

FORTRAN语言不允许减少Do indices(指标)。因此，当人们要后退一个阵列或者向量时，必须使用一"空的(dummy)"指标并被增量。C语言更具有一般性。子程序backsub()连续执行到空变量的使用，而子程序backt()使用一循环。指标在减小。

§3.5.1 补充细节

在这一章中，调试情况仅仅检查秩序 $n=3$ 的情形，后面的各章要使用这条存储程序和检验其它情形。对于更大的 n 的情况将在关于 B—Splines 一章中探讨。在本章中刚性的 ODE 系统的求解也使用了已知线性系统的解答存储。

检验控制器 LUD.C 解出了 $Ax=b$，结果打印出解向量(这个解向量写在方程右边的向量 b 的上面)和 LU 因子(这些因子写在矩阵 A 的上面)。这可以分两步进行：用被称为 lufact()的程序来确定这些因子，而用程序 backsub()来解包含右边向量 b 的这个系统。因为 backsub()没有 LU 因子，所以对于不同的 b 向量，不用再分解矩阵 A，我们就能得到此系统的另外解答。接着，我们使用这些矩阵因子，根据称为 backt()的程序来解。由矩阵 A 的转置 $A^{\wedge}$ 构成的系统 $A^{\wedge}x=b$。我们不得不重新构造 b，它已经在上面被写出过。如果 $A=LU$，则 $A^{\wedge}=U^{\wedge}L^{\wedge}$，其中 U 的转置 $U^{\wedge}$ 现在是下三角矩阵和 $L^{\wedge}$ 现在是上三角形矩阵。因此，程序 backt()能起的作用同程序 backsub()一样，除了 backt()。为了找到 x 的矩阵因子的适当元素，而不得不查看 A 的不同单元。其次，invm()被称作决定矩阵的逆。invm()要写在矩阵 A 的上面。最后，作为对程序 backt()的检查，我们"亲手"构造矩阵 A 的转置，并且用 lufact()和 backsub()解由 A 的转置组成的系统。

文件 LUS.c 包含了包括 BLAS 函数的反演在内的支撑程序，不管矩阵中有多少行出现，对于把矩阵打印为一可读形式而言，这个程序被包括在一个一般程序中。因为依赖于你使用的宽行打印机的列数或者显示支持，这个程序能容易地改变。宽行机或者显示支持的列显示通常是保守的，我们假定为显示 80 列和通常的 C 格式化形式。

LINPACK 程序集还有计算条件数的其它程序，我们已不打算重复产生条件数的讨论，因为对于大多数问题来说，我们并不相信条件数是很有意义的值。部分 LINPACK 程序，LINPACK 文件和其它书例如(见 Forsythe 和 Moler 的书)讨论了产生条件数解法的替代改进。这需要使用更高精度的计算和保持 b 向量和矩阵 A 的复制。一旦找到 X 的一个解，人们就应按双精度计算残差 $b-Ax$。于是，用残差作右边向量，使用以前求出的 LU 因子来计算 x 的修正值。如果 LU 因子都是精确的，那么这种方法才清楚地有用。如果矩阵 A 是"有点"病态，那么尽管不用迭代得到差的解向量 x，但 LU 因子可能是合理的精确。在这种情况下，替代法将改进解向量。如果矩阵 A 不是病态的，那么有可能没有替代点。Wilkinson 指出没有小元和小的行列式计算的矩阵能够是病态的。这样的病理情形通常是很难遇到的。在通常的 C 程序执行中，因为计算仅仅按单精度、双精度处理，因此在这样一个环境中几乎没有用替代来改进的点。

在第四章，我们将发现确定转置系统 $A^{\wedge}x=b$ 是必要的，其目的是确定一非对称矩阵的特征值。求转置系统 $A^{\wedge}x=b$ 的解可通过程序 backt()和使用 A 的同样的 LU 因子来处理。

第四章中包含了许多有关线性系统的矩阵代数的程序，这些程序可能是有趣的。LINPACK 和其它存储程序，这些程序有利于满足矩阵性质，如象对称性，正定性，满秩性(bandness)等等。

例如，正定矩阵从不需要枢轴并且对于对称矩阵(即，对称矩阵是元素 $A(i,j)=$

i)的矩阵)的计算量能够减半。

注意到矩阵的特殊性质，如满秩性，阻碍了采取枢轴的方法，因为枢轴一般地将破坏矩阵的特殊结构。因此，为了解的有效性，依赖于矩阵的特殊结构的解在解的稳健性方面付出了代价。

表 3—3　第三章中的函数和子程序

名称	目的(作用)
FTOC.H	易使 FORTRAN 代码转换为 C 代码；
MAIN(LUD.C)	调试 LU 系统程序
ISAMAX	求一向量的最大值元素
SAXPY	包含向量的初等行运算
SDOT	求两个向量的点积
SSWAP	交换向量
SSCAL	给某一向量
SASUM	求一向量的绝对值的和
PRINIM	打印一矩阵(按行)
LUF.C	进行一方阵的 LU 因子分解
BACKSUB.C	使用 A 的 LU 因子解 $Ax=b$
BACKT.C	使用 A 的 LU 因子解 $A^{\top}x=b$
INVM.C	使用 A 的 LU 因子求 A 的逆
DET.C	作为 double 行列式返回
DETI.C	如果希望很大或很小的行列式，则使用此程序

表 3—3 中列出了本章所有的子程序和函数。子程序 backsub，backt，inv，det，deti 中需要的 LU 因子分解已经被子程序 LU 所决定。

§3.6　程序列表与测试问题输出

```
/*

header file to aid in conversion of FORTRAN code to C

PURPOSE:

performs in-line a number of useful chores including loops,
array subscripting, and finding minimum, maximum, and absolute valu
```

```
DEPENDENCIES:

none

USAGE:

invoke with preprocessor directive:
#include"ftoc.h"
or
#include <ftoc.h>
near the beginning of your program

*/

/* in-line functions for use with 2D arrays: */

/* row major order as in C indices run 0..n-1 as in C */
#define INDEX(i,j) [j+(i)*coln]

/* various loop constructors */
#define DOFOR(i,to) for (i=0;i<to;i++)
#define DFOR(i,from,to) for (i=from-1;i<to;i++)
#define DOBY(i,from,to,by) for (i=from-1;i<to;i+=by)
#define DOBYY(i,from,to ,by) for(i=form;i<to;i+=by)
#define DOBYYY(i,from,to) for (i=from;i<to;i++)
#define DOV(i,to ,by ) for(i=0;i<to;i+=by)
/* row major order as in C indices run 1..n */
/* #define INDEX1(i,j) [j-1+(i-1)*n]
*/
/* column major order,as in fortran: */
#define INDEXC(i,j) [i-1+(j-1)*rown]

/* usage:if a(20,30) is matrix,then
a(i,j) in C will be a INDEX(i,j) if n=30. */

/* to index vectors starting with 1 */
#define VECTOR(i)[i-1]

#define min(a,b)(((a)<(b))? (a):(b))
#define max(a,b)(((a)<(b))? (b):(a))
#define abs(x)(((x)>=0.)? (x):-(x))


/*
routines for linear systems processing via LU factorization

PURPOSE:
```

```
perform LU factorixation

CONTENTS:

lufact(a,coln,n,pivot,info)
   Factor an n by n matrix contained in type float array a
   which is dimensioned a[m][coln]. pivot is an integer array
   of size n which will contain pivoting information to be used
   by later routines. info is pointer to an integer which returns
   0 if all is well,otherwise the row in which problems occurred.

DEPENDENCIES:

requires header file ftoc.h and lus.c routines

*/
#include "ftoc.h"

lufact (a,coln,n,pivot,info)
float *a;
int coln,n, *pivot, *info;
{
   int i,j,k,l,kp1,nm1,last;
   float t;
    *info=0;
   nm1=n-1;
   if (nm1>=1)
   {/*nontrivial pblm*/
      DOFOR(k,nm1)
      {
        kp1=k+1
        /* partial pivoting ROW exchanges-search over column */
        /* in FORTRAN,the increment would be 1 not n in ismax call */
        pivot [k]=l=isamax((n-k),&(a INDEX(k,k)),coln)+k;
        if (a INDEX(l,K)! =0.)
        {/*nonsingular p\ivot found*/
          if(l! =k)
          {/*interchange needed*/
            t=a INDEX(l,k);
            a INDEX(l,k)=a INDEX(k,k);
            a INDEX(k,k)=t;
          }
          t=-1./a INDEX(k,k);/*scale row*/
          sscal(nm1-k,t,&(a INDEX(k+1,k)),coln);
          DOBYYY(j,kp1,n)
          {
```

```
            t=a INDEX(l,j);
            if(l! =k)
            {
                a INDEX(l,j)=a INDEX(k,j);
                a INDEX(k,j)=t;
            }
            saxpy(nm1-k,t,&(a INDEX(k+1,k)),
                coln,&(a INDEX(k+1,j)),coln);
          }
        }
          else /* pivot singular */
          {*info=k;}
        }/* main loop over k */
      }
      pivot [nm1]=nm1;
      if (a INDEX(nm1,nm1)==0.0) *info=nm1;
      return;
}

      /
#include"ftoc.h"
/*
routines for linear systems processing via LU factorization

PURPOSE:
support routines for LU factorization

CONTENTS:
isamax(n,sx,incx)
    Finds the location of the element of greatest absolute
    balue in a vector sx of length n. Each incx-th element
    is examined (hence,if sx is a 2-D matrix,may be used
    to find largest elements in each row or column,depending
    upon whether incx 1 or n.

saxpy(n,sa,sx,incx,sy,incy)
    Performs an elementary row operation sy=sy+sa sx where sx
    and sy are vectors and sa is a scalar. Used to subtract
    a scaled row sx from the sy row. incx,incy are as in isamax.
    Vectors of length n.

sdot(n,sx,incx,sy,incy)
    Takes the dot product of 2 vectors sx and sy,of length n.

sswap(n,sx,incx,sy,incy)
    Exchanges two vectors sx and sy. Used for row exchanges
    which occur during pivoting operation.
```

```
sscal(n,sa,incx)
     Scale a vector sx=sa sx where a is a scalar and sx a vector

sasum(n,sx,invx)
     Function type float which returns the sum of the absolute
     values of the elements of a vector.
(The above are all based upon BLAS routines.)
printm(a,coln,rown,col,row)
  Prints a 2—dimensional matrix a in readable form.
  The actual data of the form a(row,col) is stored
  within a matrix dimensioned a[rown][coln].

*/
#include "ftoc.h"

int isamax(n,sx,incx)
int n,incx;float *sx;
{int maxi,ix,i;
   float temp,smax;
/* returns 1 less than corresponding FORTRAN version */
   if (n<=0)return -1;
   if (n==1)return 0;
/* ix=0 */
   maxi=0;
   smax=abs(sx[0]);
   ix=incx;/* ix=ix+incx=inxc */
   DFOR(i,2,n)
   { temp=abs(sx[ix]);
      if (temp>smax)
      {smax=temp;
        maxi=i;
/* return ith element as max,NOT subscript 3[ix]ix=i*incx */
      }
      ix+=incx;
   }
   return maxi;
}

saxpy (n,sa,sx,incx,sy,incy)
int n,incx,incy;
float sa,*sx,*sy;
{/*sy=sa*sx+sy*/
  int i,iy,ix;
  if(n<=0)return;
  if(sa==0.)return;
```

```
  iy=ix=0;
  if(incx<0) ix=incx * (1-n);
  if(incy<0) iy=incy * (1-n);
  DOFOR(i,n)
  {
    sy[iy]=sy[iy]+sa * sx[ix];
    iy+=incy;
    ix+=incx;
  }
  return;
}

float sdot (n,sx,incx,sy,incy)
int n,incx,incy;
float *sx, *sy;
{float stemp;
   int i,ix,iy;
   if(n<=0)return(0.);
   ix=iy=0;
   stemp=0.0;
   if(incx<0) ix=incx * (1-n);
   if(incy<0) iy=incy * (1-n);
   DOFOR(i,n)
   {
     stemp+=sy[iy] * sx[ix];
     iy+=incy;
     ix+=incx;
   }
   return stemp;
}

sswap(n,sx,incx,sy,incy)
int n,incx,incy;
float *sx, *sy;
{
   int ix,iy,i;
   flaot t;
   if(n<=0)return;
   ix=iy=0;
   if(incx<0) ix=incx * (1-n);
   if(incy<0) iy=incy * (1-n);

   DOFOR(i,n)
   {
    t=sx[ix];
    sx[ix]= sy[iy];
    sy[iy]=t;
```

```
    lx+=incx;
    iy+=incy;
   }
   return;
}

sscal(n,sa,sx,incx)
int n,incx;float sa, *sx;
{/*scale vector*/
  int i,nincx;

  if(n<=0)return;
  nincx=incx*n;
  DOV(i,nincx,incx)
  sx[i]=sx[i]*sa;
  return;
}

float sasum(n,sx,incx)
float *sx;
int incx,n;
{/* ssum abs values*/
  int i,nincx;
  double stemp;
  stemp=0.0;
  nincx=n*incx;
  if(n<=o)return 0.0;
  DOV(I,n,nincx) stemp=stemp+abs(sx[i]);
return(stemp);
}

printm(a,coln,rown,col,row)int rown,row,col,coln;float a[ ];
{
     int i,j,btm,top,count;
     printf("\n");
     btm=top=0;
     while(btm<col)
     {
       top=min(col,(btm+8));
       printf("printing matrix columns %d to %d\n",btm.(top-1));
       DOFOR(j.row)
       {
         for(i=btm;i<top;i++)
         {
             printf("%e",a INDEX(j,l));
         }
         printf("\n");
```

```
        }
        btm+=8;
      }
      return;
}


#include"ftoc.h"

/*
routines for linear systems processing via LU factorization

PURPOSE:

solve linear system given factorization of a matrix in a

CONTENTS:

backsub(a,coln,n,pivot,b)
  Solves system ax = b. Assumes a contains LU factors,pivot
  pivoting information. The matrix stored as a[m][coln]with the
  system to be solved n x n. Answer vector returned in b.

backt(a,coln,n,pivot,b)
  As above,but will return the solution to a[T]x = b where a[T]
  is the transpose of matrix a. This is useful in inverse
  iteration for eigenvalue determination if the matrix a is
  non-symmetric,and in determining matrix condition number.

DEPENDENCIES:

requires header file ftoc.h and routines in lus.c

*/


backsub(a,coln,n,pivot,b)
int coln,n, *pivot;
float *a, *b;
{
    float t;
    int k,l,nm1;
    nm1=n-1;
    /* solve ly=b first */
    DOFOR(k,nm1)
    {
      l=pivot[k];
```

```
        t=b[l];
        if(l!  =k)
        {
            b[l]=b [k];
            b[k]= t;
        }
        saxpy(nm1-k,t,&(a INDEX(K+1,K))),coln,&(b[k+1]),1);
      }

      /* solve Ux=y */
      DOFOR(1,n)
      {
        k=nm1-l;
        b [k]=b [k]/a INDEX(k,k);
        t=-b [k];
        saxpy(k,t&(a INDEX(0,k)),coln,b,1);
      }

      return;
}

backt(a,coln,n,pivot,b)
int coln,n, *pivot;
float  *a, *b;
{/* like backsub,except solves A(T)x=b */
   float t,sdot();
   int k,l,nm1;
   nm1=n-1;

   /* solve u(T)y=b first */
   DOFOR(k,n)
   {
     t=sdot(k,&(a INDEX(0,k)),coln,b,1);
printf("k=%d t=%f\n",k,t);
     b[k]=(b[k]-t)/a INDEX(k,k);
   }

   /* solve l(T)x=y */
   if(nm1<1)return;
   for(k=nm1-1;k>=0;k--)
   {
    b[k]+=sdot(nm1-k,&(a INDEX(k+1,k)),coln,&(b[k+1]),1);
    l=pivot[k];
    if(l!  =k)
    {
        t=b[l]
        b[l]=b[k];
```

```
            b[k]=t;
        }
      }

      return;
}


#include "ftoc.h"

/*
routines for linear systems processing via LU factorization

PURPOSE:

invert a matrix given the LU factorization of that matrix

CONTENTS:

invm(a,coln,n,pivot,work)
    a contains LU factors of a matrix. a[m][coln],n is size of
    the actual matrix to be inverted. pivot has pivot information.
    work is a type float work array of size n.

DEPENDENCIES:

requires header ftoc.h and routines in LUS.C

*/

invm(a,coln,n,pivot,work)
int coln,n,*pivot;
float *a,*work;
{
    float t,ten;
    int i,j,k,l,kb,kp1,nm1;
    nm1=n-1;
/* no det calc. */
/* inverse u */
    DOFOR(k,n)
    {
      a INDEX(k,k)=t=1./a INDEX(k,k);
      t= -t;
      sscal(k,t,&(a INDEX(0,k)),coln);
      kp1=k+1;
      if (nm1>=kp1)
      {
```

```
            DOBYYY(j,kp1,n)
            {
              t=a INDEX(k,j);
              a INDEX(k,j)=0.0;
              saxpy(k+1,t,&(a INDEX(0,k)),coln,&(a INDEX(0,j)),coln);
            }
        }

      }
/* inv(u) * inv(l) */
    if (nm1>=1)
    {
        DOFOR(kb,nm1)
        {
          k=nm1--kb-1;
          kp1=k+1;
          DOBYYY(i,kp1,n)
          {
            work [i]=a INDEX(i,k);
            a INDEX(i,k)=0.0;
          }
          DOBYYY(j,kp1,n)
          {
            t=work [j];
            saxpy(n,t,&(a INDEX(0,j)),coln,&(a INDEX(0,k)),coln);
          }
          l=pivot [k];
          if(l! =k) sswap(n,&(a INDEX(0,k)),coln,&(a INDEX(0,l)),coln);
        }
    }
    return;
}


/* two routines for finding determinants

PURPOSE:return value of determinant of a matrix

CONTENTS:
det()
    Returns the determinant of a matrix factored in LU form.

defi()
    Returns the determinant in the form a * 10 * *j where j
    is an integer. it is useful if the determinant might be
    very large or small,resulting in overflows or underflows.
```

```
DEPENDENCIES:
ftoc.h header file
assumes lufact() has performed the matrix factorization
*/

#include <ftoc.h>

double dabs(x) double x;
{
  if(x>=0.)return x;
  return (-x);
}


double det (a,pivot,coln)
int pivot[ ],coln;
float *a;
{
   int i,sign;
   double d;
   sign=0;
   d=1.;
   for (i=0;i<coln;i++)
   {
       if(pivot[i]!=i)sign++;
       d*=a INDEX(i,i);
   }
  sign=sign-((sign>>1)<<1);
  if (sign) d=-d;
  return d;
}

double deti(a,pivot,coln,expon)
int pivot[ ],coln, *expon;
float *a;
{
   int i,sign;
   double d,dabs();
   sign=0;
   d=1.;
   *expon=0;
   for (i=0;i<coln;i++)
   {
       if(pivot[i]=i)sign++;
       d*=a INDEX(i,i);
       if(dabs(d)>10.)
       {
```

```
            (*expon)++;
            d*=.1;
        }
        else if(dabs(d)<.1)
        {
            (*expon)--;
            d*=10.;
        }

    }
    sign=sign-((sign>>1)<<1);
    if(sign) d=-d;
    return d;
}

/*
test driver for linear system solver;
LU factorization and related routines

PURPOSE;

tests LU factorization,solution of ordinary and transposed
systems,calculation of inverse matrix and determinant

*/

#include "floc.h"

main (argc,argv)int argc;char **argv;
{
    float a[3][3],b[3],c[3][3];
    int i,j,pivot[3],n,info,coln;float work[3];
    double det(),deti(),determ;
/* simultaneous equation test fix */
    n=coln=3;
    b[0]=5.;
    b[1]=3.;
    b[2]=1.;
    a[0][0]=2.;a[0][1]=3.;a[0][2]=-1.;
    a[1][0]=4.;a[1][1]=4.;a[1][2]=-3.;
    a[2][0]=-2.;a[2][1]=3.;a[2][2]=-1.;
/* ans 1.,2.,3.  */
    printf("lufact coln=%d,n=%d\n",coln,n);
    lufact(a,coln,n,pivot,&info);
    printf("lufact info=%d%d%d\n",info,pivof[0],pivot[1],pivot[2])
    printm(a,3,3,3,3);
/* packed LU; 4 4 -3
```

—.5 5 —2.5

```
.5.2 1.  * /
  determ =det(a ,pivot ,coln );
  printf("determinant = %f\n" ,determ);
  determ =deti(a ,pivot ,coln ,&i);
  printf("determinat = %f x 10^  %d\n" ,determ ,i);
  backsub(a ,coln ,n ,pivot ,b);
  printf("\n solution to linear system :");
  printf("ans = %f %f %f\n" ,b[0],b[1],b[2]);
/ * transposed system  * /
  b[0]=5. ;
  b[1]=3. ;
  b[2]=1. ;
  backt(a ,coln ,n ,pivot ,b);
  printf("\n solution to transposed system :");
  printf(" ans = %f %f %f\n" ,b[0],b[1],b[2]);
/ * inverse * /
  printf("\n solution to transposed system :");
  printm(a ,3,3,3,3);
/ * .25 0.  —.25
  .5 —.2 .1
1.  —.6 —.2 is inverse * /
  invm(a ,coln ,n ,pivot ,work);
  printm(a ,3,3,3,3);
/ * check transposed system solution  * /
  b[0]=5. ;
  b[1]=3. ;
  b[2]=1. ;
  a[0][0]=2. ;a[1][0]=3. ;a[2][0]=—1. ;
  a[0][1]=4. ;a[1][1]=4. ;a[2][1]=—3. ;
  a[0][2]=—2. ;a[1][2]=3. ;a[2][0]=—1. ;
  lufact(a ,coln ,n ,pivot ,&info);
  backsub(a ,coln ,n ,pivot ,b);
  printf("\n check transposed system\n ans = %f %f %f\n",
                                    b[0].b[1],b[2]);
  exit(0);
}
lufact coln =3,n =3
lufact info =0 1 2 2

printing matrix columns 0 to 2
4. 000000e+000 4. 000000e —3. 000000e+000
—5. 000000e—001 5. 000000e+000 —2. 500000e+000
5. 000000e—001 —2. 000000e—001 1. 000000e+000
determinant =20. 000000
```

```
determinant= 2.000000 ×10^ 1

solution to linear system ;ans=1.000000 2.000000 3.000000
k= 0 t=0.000000
k= 1 t=5.000000
k= 2 t=-2.750000

solution to transposed system ;ans=3.750000 -1.200000 -1.150000

LU factors as stored in a;
printing matrix columns 0 to 2
4.000000e+000 4.000000e+000 -3.000000e+000
-5.000000e-001 5.000000e+000 -2.500000e+000
5.000000e-001 -2.000000e-001 1.000000e+000

inverse of a;
printing matrix columns 0 to 2
2.500000e-001 0.000000e+000 -2.500000e-001
5.000000e-001 2.000000e+001 1.000000e-001
1.000000e+001 -6.000000e-001 -2.500000e-001

check transposed system
ans=3.750000 -1.200000 -1.150000
```

第四章　矩阵的特征值和行列式分析

§4.1　本章目的

在这一章中，我们将介绍下列工具：通过稳健的 QR 迭代确定一个矩阵的特征值和特征向量；确定几个最大或者最小的特征值，并借助幂迭代、逆迭代和雷利(Rayleigh)商法把它们结合起来；提供有关矩阵和向量的有用运算的方法集；例如，交换矩阵的行和列。

对于求出的每一个特征值，降低矩阵阶数，从而简化决定下一个特征值问题的方法将被介绍。我们将回顾确定特征值问题和讨论通常使用的多种方法。我们也将讨论特征值分析对于讨论问题的应用，比如判别分析中的应用。诸如象歇尔曼——摩利逊(Sherman—Morrison)公式和普鲁林(Prony)方法也将在本章讨论。

§4.2　特征值分析

在这一章中，对于确定特征值和特征向量，我们将讨论相当多的方法。当矩阵只有几个特征值(和它们的相伴特征向量)时，我们将特别留意此时求矩阵特征值的最适当的方法——迭代法，用这种方法使最大特征值或者最小特征值被确定。为了确定复数特征值和使特征值问题一般化，我们将讨论对称矩阵和非对称矩阵。

请注意到用特征值或者特征向量法可以处理的许多问题，也可用奇值分解(SVD)法代替特征值(特征向量)法来处理它们，这种作法尤其在涉及到相关矩阵的统计问题中是确实可靠的。对于这样的问题，矩阵是正定的(矩阵仅仅有正特征值)和由奇值(SVD)分解所确定的奇异值是矩阵的特征值的平方。因为 SVD 确定的这些特征值比用特征值分析确定的特征值更具有稳健性，所以在确定矩阵的特征值时，SVD 更可取。因为关于统计学中的数值法的很多更旧的书并没有讨论 SVD，因此当你已经习惯用一种特征值法考虑有关问题的同时，应意识到 SVD 可能不是目前最有效的方法。

§4.2.1　怎样提出特征问题的

术语“特征值”出自词语“eigenwert”，这个词可以译成“特有的”或者“特有的值”。“特有的值”是表示矩阵的不变性质(或者给出矩阵的这个问题)的那些数。

数学上，对于阶数为 n 的方阵，上述问题是确定满足 $AX=eX$ 的一个数 e 和一向量 X。对此，一般将有 n 个解。的确，如果我们把 $AX=eX$ 重新写为 $(A-Ie)X=0$，其中 I 是单位阵，那么只有当矩阵 $A-Ie$ 的行列式值为 0 时(这个条件等价于 e 的一 n 阶多项式方程)，我们才能够得到除平凡解 $X=0$ 之处的解。根据代数基本定理得知，这样的一个多项式方程将有 e 的 n 个根。

在涉及到普通微分方程的边界值问题(BVP's)中，虽然常常伴随着的是特征函数，而不是特征值，但是术语特征值却出现。这个BVP问题的特征值，其意思是完全类似的，只不过用一函数 X 乘以一线性微分算子 A 去代替了一向量 X 乘以一矩阵 A，要求函数 X 满足已知的边界条件。一般地，称之为特征值的参数 e 的对应于特征函数解的某些值，便能够求出对应于这个 e 值的非平凡特征向量 X(即 $AX=eX$ 的非平凡解)。根据这样的特征函数，我们能够展开边界值问题(BVP)的任意解，这样做允许我们放弃对应于这个BVP的微分方程和使用矩阵代数知识来处理边界值问题(BVPS)！在特征函数的展开项中，解被表示为项的向量系数。这就是为什么Schroedinger's波动方程和Heisenberg's矩阵力学两个原来都独立的量子力学形式，能够被人们证明是一致的和等价的原因。

量子力学问题的特征值是这个系统自身的共振频率，也就是说，在这个共振频率处，不需要外界施加压力，系统"自然地"能够感到自身的振动。对于一有界系统，比如一个孤立原子，这些频率给出了这个原子所处的原子能级。在这些状态下，这个原子仍然能够稳定。由于辐射场的相互影响，这种相互影响没有清楚地包含在具有最简单形式的Schroeding's方程中，我们修改这种情况以便在这些状态之间，原子发生跃迁现象，并且这个原子将从一个能级衰变到另一个能级直至达到"基态"(最低级)状态，这时，这个原子已经没有能量到达任何其它能级。这样的频率或者能级(由于能量等于普朗克(Planck)常数乘以频率，因此，在量子力学中，频率和能级是等价的。)虽然是涉及到这个原子所"特有的"。这用来说明我们正在讨论的特征值的"eigen"的使用。

用稍微不同的形式我们称特征值问题是"对角化"一个矩阵。这意味着求一个所谓的矩阵 A 的矩阵 D 的相似变换：

$$D=U^{-1}AU$$

其中，U 是一正交阵或者实西阵(即，满足 $U^{\wedge}U=U^{-1}U=1$)以及 D 是对角矩阵，即

$$D=\begin{pmatrix} d(1,1) & 0 & \cdots\cdots \\ 0 & d(2,2) & \cdots\cdots \\ 0 & 0 & d(3,3) \end{pmatrix}$$

(因为 U 是正交的，所以变换也是相合变换，也就是说，$D=U^{\wedge}AU$)

对任何对称矩阵 A，我们可以证明这个变换都是可能的。矩阵 D 的对角线元素既是 D 自身的特征值，同时也是矩阵 A 的特征值。D 的特征向量恰好是单位轴，即对应于 $d(1,0,0,\cdots\cdots)$ 的特征向量是 $(1,0,0,\cdots\cdots)$，对应于元素 $d(2,2)$ 的特征向量是 $(0,1,0,\cdots\cdots)$，等等。矩阵 U 是其列为 A 的特征向量(标准化这些特征向量以便使每一个特征向量的长度为1)的矩阵。矩阵 U 可以看作这个坐标系统到这个矩阵的主轴的一个旋转，也许，大家最熟悉的正交矩阵的形式是：

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

和对应于将 xy 坐标系统的Z轴旋转角度为 t 的一个旋转。例如，假设 θ 是90度，当向量 (x,y) 乘以这个矩阵 $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ 后，变成了 $(y,-x)$。这应该对对角化处理作出一似乎合理的旋转的几何解释。这样的一个旋转是一正交变换。一个向量 y 被变换成 Uy，一矩

阵 A 被变成 $U\hat{}AU$。为了使这种变换结果具体，我们用一固体的惯性矩作为例子。根据 $p=mv$，一个物体的质量或者惯性与它的动量 p，速度 v 有关。通过对物体施力，按照牛顿第二定律 $F=ma$，物体的质量也决定了它的加速度的大小，其中 a 是加速度向量和 F 是力向量。类似地根据 $\tau=MW$ 惯性张量 M 与这个角动量向量 τ 和角速度 W 有关，两者都是向量。因为 m 是标量，所以 p 和 v 一定是平行的(成比例的)，正象 a 和 F 一样，但是由于 M 是一张量，τ 不必平行于 W。对于三维物体而言，M 是 3 阶矩阵；为了简单起见，我们将考虑二维物体。根据向量：

$$L=\sum^{N} m[i]r[i]\times v[i]$$

由 N 个质点 $m[i]$构成的一旋转物体的角动量是已知的。在式子中，r 是一位置向量，v 是点的速度，用 x 表示向量叉积(外积)。但是 $v=w\times r$，其中 W 是角旋转向量。因此

$$L=\sum m[i]\{r[i]\times(w\times r[i])\}$$

这个式子也可以重写成

$$L=\sum m[i]\{w|r[i]|^2-r[i](r[i]*w)\}$$

用 $*$ 表示标量或者"点"积。我们再把上面这个式子，写成

$$L=MW$$

的形式。其中 M 是"并矢量"或者二阶张量，即是一矩阵。例如，对于 XY 平面(坐标)中的二维惯性张量 $M(1,1)=\int\rho^2 y\,dv$ (常常写成 M_{xx} 的形式)和 $M_{xy}=M_{yx}=-\int\rho y\,dv$，其中，$\rho$ 是物体的质量密度。根据 W 指定的任何特殊旋转的动能是标量：

$$T=W*M*W/2$$

如果我们把向量 W 表达为(a,b)，那么得到 T 的二次形式：

$$T=a^2M(1,1)+b2M(2,2)+ab\{M(1,2)+M(2,1)\}$$

这个多项式叫做 a 和 b 的二次形式。由惯性张量 M 的定义，我们知道 M 是对称的，进而作为动能决不可能是负的，矩阵 M 必定是正定阵。因此，矩阵 M 一定能够对角化，即是，与 a 和 b 相关的一组变量 A 和 B 经过一旋转后的二次方程形式中已不含 A 和 B 的叉积项 AB。因为这个二次形式是正定的，所以特征值 d 都是正的。于是，我们有：

$$T=\{A^2d(1,1)+B^2d(2,2)\}/2$$

其中，A 和 B 是 W 在这个主轴方向上的分量，即是，根据类似于旋转矩阵的类似变化，矩阵的特征向量和新的矩阵 d 同 M 旋转到新的坐标系统有关。我们能够把向量 $W=U\hat{}w=(A,B)$看成这个主轴方向上旋转 W 向量的结果。因为这个特征向量被证明是正交的所以这些轴是垂直的，由于 T 的表达式中的两个系数都是正的，因此 T 的表达式显然表明这个二次形式表示的是一椭圆。这个主轴或者特征向量是在那个椭圆的长轴和短轴方向上(对于目前讨论的这个问题，我们称为惯性椭球面)。于是，这个特征值对应于相应轴的平方根的倒数。

在别名"主分量因子分析"下，特征值出现在统计学中。它们在物体分类问题中有用。这同判别(式)分析密切相关。如果这个对象是一模型，那么这种分类是可识别的，Karhunen—Loeve 展开是用于统计模型识别方法中的主因子或者特征值方法。所有这些

题目将在这章和下一章进一步讨论。Karhunen－Loeve 展开可以用于数据压缩。一旦特征向量和相伴的特征值确定，那么任何数据向量都能展开成特征向量的线性和。这个展开式中的系数表示这个数据向量到这个特征向量的投射(数据向量和每一个特征向量的点积)。如果我们放弃对应于最小的特征值的系数，那么我们能得到这个数据向量的经济表达式，这个数据向量仍然保留我们感兴趣的主要特征。这种压缩方法常用在象处理中。

§4.2.2 一般化特征值问题

我们也将考虑一般化特征值问题，即是对于 $Ax=eBx$ 确定 e 和 x。例如，这个问题的发生在计算一振荡系统的正规方式中。在这种情形下，这个矩阵 B 对应于一动能项，A 对应于一势能项，以及这个特征值对应于振荡频率的平方。在有限元问题中，B 常常写成 M，并称 M 为质量矩阵，而 A 被写成 K，并称 K 为弹性矩阵，这类似于简单调和振荡中的通常的用词。人们能够计算矩阵 B 的逆(假定它的行列式的值不等于零)和用它去乘方程的两边，从而将一般性问题归结为一个普通问题。对我们来说，这就是有意义的，可是，因为求矩阵 B 的逆需要阶数为 n^3 的运算，以及用这个逆矩阵去乘两个矩阵(假定它们都是满秩的，见第三章)也需要类似数量的阶数运算。如果这样的问题常发生，那么 B 是稀疏的，此时归结为普通问题的做法甚至更不是所希望的。矩阵的逆一般并不是稀疏的，并且生成的特征问题也将不是稀疏的。如果可能，我们希望保持问题的稀疏性。这儿讨论的迭代法，适合处理这样的问题。最常见是问题涉及到的矩阵是对称的以及常常是一个或者两个矩阵都是正定的。这种问题时常出现在有限元方法的应用当中。它们很适合这儿讨论的迭代法。表 4－1 概括了特征值分析的应用。

表 4－1 特征值分析的通常运用

力学：	振动的正规方式； 共振频度； 来自有限元模型的一般化问题。
量子力学：	稳定状态和能量级。
统计学：	主分量分析； 判别(式)分析； 方差的多元分析(MANOVA)。
模型识别：	Karhunen－Loeve 展开。
数据浓缩：	主分量或者 Karhunen－Loeve 展开。

§4.3 特征值的基本理论

一般地，一个对称矩阵 A 将有一个特征向量，这个特征向量是它对应的唯一特征值的唯一方向。当然，由于这个问题的线性性，每一个特征向量可以乘以一个常数，且结果仍是一特征向量，对应于不同的特征值的特征向量是正交的(垂直的)，这是因为这样的特征向量的点积等于零。如果特征值是退化的，即它是重特征值，那么虽然它的特征向量可以

被正交化和扩张为一空间(Span a space),但在某些情形下将不能被正交化。这样的病态情形被称为"秩不满"。

如果矩阵不是对称的,那么以上这些结果可以一般化。对于对称矩阵,如果 e 是一特征值,那么

$X\hat{}\ A=AX=eX$,因为 e 对应的特征向量是 X。

对于非对称矩阵,我们必须考虑不同的右特征向量 X 和左特征向量 Y,对应于相同特征值满足

$Y\hat{}\ A=eY\hat{}$ 的左特征向量以及 $AX=eX$。

现在,在不同特征值的右和左特征向量是正交的,即 $Y\hat{}\ X=0$。因为这个性质,能以右和左特征向量的集合叫做"双正交"。(对于对称矩阵 A 称为共轭性质,即是 $Y\hat{}\ AX=0$,对于非对称矩阵却被称之为"双共轭性")。作为一特殊例子,我们考虑非对称矩阵:

$$A=\begin{pmatrix}1 & 1\\ 9 & 1\end{pmatrix}$$

这个矩阵有特征值是 4 和 -2,对于 $e=4$,它的右特征向量为(1,3)(实际上它是一列向量,但为方便起见,写成它的转置向量)和它的左特征向量是(1,1/3)。对于 $e=-2$,相应的右、左特征向量分别为(1,-3)和(1,$-1/3$)。对这个例子,双正交性可以很容易验证。注意到这两个右特征向量并不是正交的。一个不满秩矩阵将是具有下列形式的任何矩阵:

$$\begin{pmatrix}a & b\\ o & a\end{pmatrix}$$

这个矩阵有一个二重特征值,仅仅有一个特征向量(ZO),其中 Z 可以是任何数。这个向量既是一个右(列)特征向量,又是一个左(行)特征向量。

对称矩阵有许多方便的性质。如果矩阵是埃米尔矩阵(在埃米尔矩阵中,$a(i,j)$ 是 $a(j,i)$ 的共轭复数,埃米尔矩阵有实的特征值以及埃米尔矩阵是对称矩阵到复数或矩阵的适当一般化),那么它们的特征值一定是实数。它们的特征向量将形成一正交集,这个正交集扩张成这个空间,因此用这些特征向量,任意向量都可以展开(扩张)。

§4.3.1 迭代法

让我们首先考虑对称矩阵,然后,再考虑一般化非对称矩阵问题。

通常,我们只需要几个最大的或者最小的特征值以及可能的相伴特征向量。例如,在确定正规方式中,人们一般需要最低的或者最基本的频率和这些频率对应于最小特征值,在其它问题中,尤其在统计学(见以下内容)中,最大特征是最有意义的。为了处理这两种类型问题,有两种指定的迭代法。

我们首先考虑用矩阵的特征向量集展开一任意向量

$$Y=\sum c[i]x[i]$$

如果我们用矩阵 A 乘向量 Y,那么我们得到一向量 Y'

$$Y'=AY=\sum c[i]AX[i]=\sum c[i]e[i]x[i]$$

和 $Y''=AAY=AY'=\sum c[i]e[i]^2x[i]$,等等。

我们看到当乘以 A 时，每一项都分别乘了各自特征向量对应的特征值的更高次幂。这给了我们一种确定最大特征值和它们相伴特征向量的方法。选择一任意向量 Y 作为一始点。称具有特征向量 $X[0]$ 的最大特征值为 $e[0]$。如果我们并不是如此不幸，以致 $Y\hat{}X[0]=0$，这将在我们的展开式中出现 $c[0]=0$，那么我们能够得到 Y'，Y''，等等。通过依次乘以由矩阵 A 得到的向量，主项将是对应于最大特征值的那一项，以及很快得到连续积的比，即雷利(Rayleigh)商，$Y\hat{}AY/Y\hat{}Y$ 将是最大的特征值。

注意到我们从未用矩阵乘以矩阵(否则，付出的代价相当大)，仅仅用一向量乘以矩阵。这种方法叫做求最大特征值的"幂方法"(它的背景在斯图尔特(Stewart)、维尔金森(Wilkinson)、弗朗克林(Franklin)和帕里特的书中有充分讨论)。这种方法类似于 Krylov 迭代以及其它的方法(双迭代，梯子(treppen)迭代)，在这些方法中，始点不是 0，而是用单个向量 Y，许多向量同时使用能更好地避免麻烦。

一个明显问题是如果两个最大特征值的数量值相等，这种方法将不收敛。这个问题能够通过这个矩阵的一漂移 $A'=A-IS$ 来克服，矩阵 A' 将有特征值 $e[i]-S$。其中，$e[i]$ 是矩阵 A 的特征值。对于一起使用矩阵程序进行"密封式的"存储时，会使得幂方法不适合，但是在程序相互运行的情形下，幂方法比稳健方法计算更快，并且当麻烦出现时，用户能够再用一个不同的漂移去试。

如果我们希望得到最小特征值，那么要使用的方法是"逆迭代"。它等价于用矩阵 A 的逆矩阵的迭代，矩阵的逆矩阵的特征值是矩阵 A 的相应特征值的倒数。因此，矩阵 A 的逆矩阵的最大特征值是矩阵 A 的最小特征值的倒数。实际上，因为求矩阵 A 的逆所花的代价太大，因此我们并没有那样做。我们宁愿解系统 $AY'=Y$，接着解系统 $AY''=Y'$，等等。特别如果 A 是稀疏阵时，这样做可能更经济，假设矩阵 A 是满秩的，我们做一次 A 的 LU 因子分解要进行(见第三章)阶数为 $\frac{n^3}{3}$ 的浮点运算。其次，每个迭代仅仅需要阶数为 n^2 的运算。如果矩阵是稀疏的，那么迭代法的花费比起其它方法来便宜得多。

在逆迭代中有一种方法能够极大地改善收敛性。如果我们能使用象前面讨论的漂移特征值和把这个漂移选成雷利商，雷利商是我们对特征值是什么值的最近和最好的猜测，那么收敛会是很迅速的(立方的)。我们的读者可能并未注意到一个潜在的问题。如果 S 是接近某个特征值，那么矩阵 $A-IS$ 是病态的(见第三章)。这种情况已被证明并不是一个问题。在这样的情形下，Y'' 的解可能不精确，这可能是由于需要的特征值方向误差引起。以致所有结果(特征值)是满意的。虽然雷利(Rayleihg)商可以同求最大特征值的幂方法一起用，但它不是显著有效。明显的代价是每一次都要做一漂移，我们有一新矩阵用于迭代，因此，那个新矩阵的 LU 因子分解一定发生。如果矩阵是满秩的，那么雷利(Rayleihg)商迭代付出的代价可能从经济上来说是不大的。

用另一种方法，一旦我们得到期望限度的收敛，Y 是(近似的)希望的特征向量和雷利商给出了想要的特征值。

如果我们想求出所有的特征值，那么一般来讲，逆迭代比幂方法更可取，因为逆迭代的收敛性通常更好。

§4.3.2 降阶

一旦你已求出最大(或者最小)的特征值和它们的相伴特征向量，可能就希望继续找到下一个最大(或最小)的特征值。获得这个结果，有许多方法。你能用一向量与已求出的特征值向量正交开始你的迭代。因为舍入(roundoff)误差将使你失去正交性和主特征值将"接收"(takeover)，所以这样做是很危险的。

你能够形成矩阵 $A'=A-e[0]X\hat{}[0]X[0]$。

其中，最后一项被看成并矢量，也就是，$a(i,j)=a(i,j)-e[0]x[0][i]x[0][j]$。虽然这稍微好一些，但不考虑正交性仍将对你有吸引力。以下我们将讨论同时决定许多最大特征值的普鲁林(Prony)方法的运用。我们应用的方法是把 n 阶矩阵降低为 $n-1$ 阶矩阵，这种新的矩阵缺少所求出的特征值。因为对任何特征值，这种降阶都有效，因此降阶法可以同上面讨论的任一种迭代法一起使用。

大多数书(例如，弗朗克林(Franklin)或者维尔金森)讨论了怎样由矩阵 A 形成具有形式 $T^{-1}AT$ 的新矩阵

$$\begin{pmatrix} e[0] & \cdots \\ 0 & \\ 0 & R \\ 0 & \end{pmatrix}$$

其中，方阵 R 的阶数比 A 少 1，它与 A 除了特征值 $e[0]$ 外的所有特征值相同。形成下面的矩阵，从计算上来讲更方便。

$$\begin{pmatrix} R & 0 \\ & 0 \\ \cdots & e[0] \end{pmatrix}$$

由于这样的新矩阵 R 在存储中可找到与矩阵 A 的存储单元相同的地址。通过对弗朗克林方法的一简单修改，这样做是可以的。假定标准化特征向量 $X[0]$，以便它的第 n 个分量是 1(即是，用最后一个特征值来除以特征值中的所有元素)。如果矩阵 T 有一条主对角线和它的最后一列是这个特征向量(已被标准化的 $X[0]$)

$$T=\begin{pmatrix} 1 & & & x[0][0] \\ & 1 & & x[0][1] \\ & & 1 & x[0][2] \end{pmatrix}$$

那么，我们可能证明 T 的逆(见这章的下一节)具有下列形式：

$$T^{-1}=\begin{pmatrix} 1 & & & -x[0][0] \\ & 1 & & -x[0][1] \\ & & \cdots & \cdots \\ & & 1 & -x[0][n] \end{pmatrix}$$

经过计算，我们发现 R 中的元素仅仅与 A 中经过 $r(i,j)=a(i,j)-\sum x(i,n)a(n,j)$ 变换的那些元素有关。除了矩阵的最后一列外，$T^{-1}AT$ 的最低行也是 A 的最低行。

一旦我们得到 R 的下一个特征值和特征向量，那么对于矩阵 A 的特征向量的确定，

我们应能够使用上面的表示。比如说，我们已求出 R 的特征值 $e[1]$与特征向量 $Y[1]$，那么我们有矩阵方程：

$$\begin{pmatrix} R & 0 \\ a & e[0] \end{pmatrix}\begin{pmatrix} Y[1] \\ q \end{pmatrix}=e[0]\begin{pmatrix} Y[1] \\ q \end{pmatrix}$$

对于 q，我们可以解方程 $q=a*y[1](e[1]-e[0])$。现在，我们用向量 $y[1]q$ 乘以 T，便得到 A 的对应特征向量。这种方法可以推广到求第三个特征值，等等，并且在本书末，这种方法可以用代码来执行。

这种方法的一"额外优点"是随着每一个特征值的确定，所要用的矩阵的阶数逐渐减小。

假使特征向量的最后分量是较小或者是 0，结果将会怎样？这可以通过特征向量的最大元作为枢轴和交换矩阵 A 以及特征向量的行和列，以便特征向量中具有最大值元素是最后一个分量。这样保证了降阶过程良好的数字性质。在本章中列举的程序都实现了这些交换。

§4.3.3 逆矩阵的歇尔曼——摩利逊(Sherman—Morrison)形式

已知一个矩阵的逆矩阵，威尔弗(Wilf)给出了修改这个逆矩阵的一个很有用的公式。这个公式被称为矩阵有逆矩阵的歇尔曼——莫利逊(Sherman—Morrison)形式，它有下列形式：

$$(A+U\hat{}V)^{-1}=A^{-1}-A^{-1}U\hat{}VA^{-1}/(1+V\hat{}A^{-1}U)$$

在这个公式中，出现在右边的项 $U\hat{}V$ 必须理解为是一并矢量，不是一个标量。我们所需要矩阵 T 的逆矩阵是这个公式的特殊情形。此时，A 是单位阵 I，u 是行向量 $u=(0,0,\cdots\cdots,1)$，v 是一个除了最后一个分量为 0 外，等于特征向量进行标准化，以便它的最后分量为 1 的列向量(在变换后矩阵的(n,n)单元中的 1 由单位矩阵 I 提供)。

这个公式有很多实际应用。发展了对一个问题提高近似程序的解法，这个问题是关于激光中的一条光谱线。这个最简单的近似产生了下列形式的方程组：

$$n[j]d[j]=w[j]\qquad j=1,\cdots,N$$

其中 $n[j]$表示频率离散点的全体。对于变量 $n[j]=w[j]/d[j]$，这个系统很容易解，因为它们是分开的。现在，作为下一个改进近似，我们 $n[j]$结合在一起，这种形式的 一个新项：

$$q[j]\int n[\text{频率}]c(\text{频率})\mathrm{d}(\text{频率})$$

被添加在方程系统的左边．这个积分变成了关于 j 的和(见第九章关于求面积的内容)，因此这个系统方程的解的形式是 $n[j]+q[j]/d[j]\Sigma c[k]n[k]=w[j]/d[j]$。
由于有歇尔曼——摩利逊公式，能以解所有线性方程是不必要的。

我们既讨论了歇尔曼——摩利逊公式的推广(这要归功于伍德贝利(Woodbury))，又讨论了作为求矩阵逆的修改方法的公式的应用(见第五章 §5.1)。

§4.3.4 非对称矩阵

幂和逆方法并不要求矩阵是对称的。然而，反对称矩阵会更成问题——例如，实反对

称矩阵的特征值可能是复数。人们也能通过迭代求出剩下的特征向量，或者如果已知矩阵的某个特征值，那么相应的特征向量可以容易地得到。

§4.3.5 复特征值

如果一实反对称阵有复特征值，那么这些特征值将成共轭对出现。互为共轭的一对特征值有相等的模和迭代将对任何一个特征值都不收敛。可是，这样的非收敛性能够被检测(发现)以及能够用于确定矩阵的复特征值。这个步骤是相当简单且在弗朗克林的书中已经讨论，用幂方法，这个步骤形成了近似特征向量的三个连续迭代，例如 Y，Y'，Y''。如果 Y，Y'，Y''(即，如果使用幂方法，那么 Y，Y'，Y''中应取绝对值最大的)对应的特征值是复数，那么对于机器的精确性而言，这些特征向量迭代结果是线性相关的。在检验以上所述情况后，你将得到满足方程 $Y''+aY'+bY=0$ 的两个系数 a 和 b。对应的二次方程 $x^2+ax+b=0$ 将产生希望的特征值。

§4.3.6 普鲁林(Prony)方法和多重特征值

用于确定多重特征值的幂方法的一有趣变形是由 A.E 西格曼(Siegman)和 H.Y. 迈勒(Meller)在应用光学(Applied Optics)中给出的。在这种幂方法变形中，对于 Y，Y'，Y''等的迭代乘法不是仅仅考虑主项，而是第 K 次迭代将有下列形式：

$$Y^{[k]}=\Sigma c[i]e[i]^k X[i]。$$

对于一个对称阵，它的特征向量是正交的，对于式子中的 $c[i]$和 $e[i]$，根据取每一个后继 Y 与最后一个 Y 的积，我们能够生成一组(标量)方程(如果矩阵是非对称的，那么用类似方式，我们能够运用双正交性。)于是，我们得到一个如下的方程序列：

$$\sum c[i]=S[0]$$
$$\sum c[i]e[i]=S[1]$$
$$\sum c[i]e[i]^2=S[2]$$

根据"普鲁林(Prony)法"，我们原则上能够通过解这些方程来求得 $e[i]$。为了确定 N 个最大特征值，我们将需要 $2N$ 个方程，并且假定不包括这 N 个最大特征值后的更小特征值(从任何初始 Y 开始的迭代，人们都希望在经过几次之后，能在上面给出的和式中"丢失"这些更小的特征值)。注意到在形成的这 $2N$ 个方程中，对于 $S[0]$的那个方程没有用到。这种手段用于确定根为 $e[i]$的多项式的系数。如果这个多项式取为：

$$X^N+q[1]X^{N-1}+\cdots\cdots+q[N]=0$$

那么对于向量 q，方程组的形式为：

$$\begin{bmatrix} S[N] & S[N-1] \\ S[N+1] & S[N] \\ \cdots\cdots & \end{bmatrix}\begin{Bmatrix} q[1] \\ q[2] \\ \cdots\cdots \end{Bmatrix}=\begin{bmatrix} -S[N+1] \\ -S[N+2] \\ \cdots\cdots \end{bmatrix}$$

这个方程组可以用通常方法来解。实际上，这种方法通常用作最小二乘法。当方程数多于未知数的个数，下一章的方法在这种情形下是有用的。因为普鲁林方法是病态的，所以应小心使用。

§4.3.7 回到一般化特征值问题

幂和逆迭代法容易推广到处理迭代，这种迭代式中含一个矩阵。实际上，要补充的所有方法是要增添适当的矩阵乘法。贝斯(Bathe)和威尔逊(Wilson)讨论了迭代方法在这方面的相当广泛的应用。对于问题 $Ax=eBx$，我们能使用逆迭代，通过解系统 $AX'=BX$ 来找到最小的特征值，而通过幂方法(也被贝斯(Bathe)和威尔逊(Wilson)称为"前置迭代")可以求出最大的特征值，即是通过解系统 $BX'=AX$。在每一种情况下，已知 X 时，我们都要解出最新的迭代结果 X'。雷利(Rayleigh)商迭代和漂移可以用于普通特征值问题。

§4.3.8 判别(式)分析和一般化的特征值问题

统计学中的多重判别(式)分析可归结为一般的特征值问题，这个特征值问题具有一个正定阵(它是一相关阵)和其它对称矩阵。实际上，有一套"判别分析"方法，并且它们以许多名称出现。如：一般的 t 统计量、马哈拉诺比斯(Mahalanobis)D^2 统计量、费歇尔(Fisher)判别式以及霍特林(Hotelling)的 T^2。霍特林统计量事实上仅仅同马哈拉诺比斯统计量有关。

现在讨论判别分析。假定我们有两个样本总体。我们希望决定一条线，这两个总体被投射(即是，每个总体点按线投射和垂直于那条直线)到这条线上，且满足两个样本间的偏差要尽可能大(相对于它们的组内方差)。这个方法的实用性是相当清楚的——我们将用一维标准卷紧，即是沿着这条线在投射之后的位置，这条线将使用我们至少大体上确定在这条上的最好样本是什么，根据这条线对样本的度量来指派将来的个体。求线的方向的问题将暗示着这问题本身是一特征值问题。费歇尔在 1938 年研究过这个问题。解答的形式为 $Ax=eWx$ 的一般化特征问题，其中 W 是一相关阵"组内交叉积矩阵"，和 $A=T-W$，其中 T 是"总交叉积矩阵"。我们得到：

$$w(i,j)=\sum\sum\{x(i,k,n)-x(i,k)\}\{x(j,k,n)-x(j,k)\}$$

和

$$t(i,j)=\sum\sum\{x(i,n)-x(i)\}\{x(j,n)-x(j)\},$$

其中 $x(i,k,n)$是单个的观察和 $X(\cdots\cdots)$意味着取适当的观察组数。这个矩阵 A 能够被认为是"组间交叉积矩阵"。对于我们讨论的问题，雷利商现在具有形式：

$$X\hat{}\,AX/(X\hat{}\,WX)。$$

并且很清楚最大特征值即是雷利商，这个最大特征值在已知组内方差时，将使组间方差取得最大值。特征向量 X 给出了判别函数即系数的方向。如果基于最大特征值的一个检验在"判别性的幂"中是不充分的，那么下一个最大特征值可能被用作一后继判别因子，这样一个检验的"判别性的幂"是相关于特征值的分布，和可以用维尔克斯兰姆达(lambda)判别来表达。维尔克斯兰姆达实际上是 $|W|/|T|$，由于 W 和 T 都是正定阵，能以它们的特征值是正的以及相应的行列式的值是正的(它们是特征值的乘积)。因此，维尔克斯兰姆达是在 0 到 1 之间的一个数，具有更大值表明那个"判别性的幂"是不大的，即我们不能区分两组数据。

虽然这些书提供了极好的背景材料和例子，但在库利(Cooley)和朗尼斯(Lohnes)的书中对于确定特征值的数值方法，线性系统解法都已过时和不应该使用这些方法。注意到

对广义特征值解法的需要，一般仅对几个最大特征值和相应的特征向量的需要来讲，这个问题几乎是为迭代方法特制的。

维尔克斯(统计量)包含了推广到多重样本的信息。注意到确定判别函数的有效性是我们的主要目标。对于个体的适当分类(个体属于哪一组?)是另一个问题。我们将在下一章遇到这个问题。

在下章中讨论的奇值分解(SVD)也可以应用于判别分析。

§4.4 QR 法

虽然确实能找到特征值和特征向量的更稳健的方法，但有更多代码和可能更大的花费。EISPACK 程序包(Package)也许是最好的。通常方法涉及到把矩阵 A 因子分解为 QR，其中 R 是一上三角阵(正象在 LU 分解中的 U 矩阵一样)和 Q 是一正交阵(正象 T 一样)。这样的因子分解能够稳健地进行，不需要装枢轴(交换矩阵的行或者列)它通常在两个阶段是行得通的。首先，在对称矩阵情况下，这个矩阵被变换成一相似的上三角矩阵或者所谓的一般矩阵的黑森贝尔格(upper－Hessenberg)形式(三对角线矩阵是在主对角线)与有另外元素的矩阵。这样的变换根据豪斯赫尔德(Householder)能够实现，进行这种简化使得以后的运算经济。既然特征值问题是等价于解一个第 n 阶多项式方程和必须迭代地解 n 大于或等于 5 的那些方程，于是，我们必须使用迭代步骤。在 QR 法中，通过豪斯赫尔德(Householder)，A 被分解为形式为 QR 的因子，那么一个新的矩阵 RQ 被(间接地)形成。因为 Q 是正交的，能以 $Q\hat{}\ Q=I$ 和 $R=Q\hat{}\ A$，这个新矩阵 RQ 就是 $Q\hat{}\ AQ$，这个矩阵又被分解为因子的 QR 的形式，等等，我们能够证明这个迭代是收敛的，从而给出了特征值作为矩阵乘积(如果有多重或者复特征值，那么就有乘法，而通过当今的代码，这些问题能够处理得相当好)的对角线。实际上，用雷利商迭代时，为了加速收敛，我们引进了一漂移算子。

有许多方法用来加速这个过程。由于把双漂移当作单步执行，所以这种方法可以避免伴有一对复数共轭特征值出现，而进行相当复杂的计算常常能找到一对实特征值。进一步，当两个邻近的充分小的子对角线(Subdiagonal)元素被找到或者找到一个 0 子角线(zerosubdiagonal)元素时，矩阵就能够进行分解，从而进一步简少了需要做的工作。

§4.4.1 平衡

E. E. 奥斯默姆(Osborne)注意到特征值计算的误差通常限于机器精度乘以矩阵的范数(等于矩阵所有元素平方和的平方根)。为了减少这个范数和相伴的误差极限，他建议在进行特征值计算以前，预先限制矩阵。矩阵 A 的因子分解被矩阵 $S^{-1}AS$ 所代替，其中 S 是一对角阵，它的元素用于减少矩阵 A 的条件数。这样做的结果，A 的特征值当然没有改变。这样处理的程序由艾斯帕克(EISPACK)、维尔金森(Wilkinson)以及里恩斯奇(Reinsch)的线性代数一书给出，并且这儿仅作为 QR 程序包的一部分给出。虽然它们的使用是可选择的，但由于这两种方法的花费不多，因此它们通常值得一试。通过程序 balbak()平衡矩阵的特征向量一定可变换，并给出了原来矩阵的特征向量。

§4.5 检验问题

§4.5.1 QR 法

产生黑森贝尔格(Hessenberg)矩阵的程序分别由 QR 迭代代码来检验。检验情况是出自于维尔金斯(Wilkinson)和里恩斯奇(Reinsch)。它们使标准化的特征向量的最大元素(实的或者复的)是 1。为了准备比较,我们后处理我们的答案,最初的检验情形只能运用于"两个根成立"的情况,既可能有实的,又可能有复的,因此我们包括了用于检验单根情况的 3×3 矩阵。由于重复特征值 2 次,这情况有一点儿病态。注意到这种退化,相伴特征向量就不是唯一确定的。

§4.5.2 幂方法

对于"幂方法"或者置前迭代(最大特征值)和雷利商迭代(使用雷利商的逆迭代,这种逆迭代为改进收敛性提供漂移),包括两种程序。

检验问题的矩阵如下:

$$a=\begin{pmatrix}5 & -4 & -7\\ -4 & 2 & -4\\ -7 & -4 & 5\end{pmatrix}$$

这个矩阵有特征值 12,6 和 −6。虽然从特征值来看,这个矩阵并不明显是病态的,但这个矩阵给我们提供了许多令人头疼的问题。首先,我们使用向量(1,1,1)作为初始猜测。因为这事实上是特征值 6 的精确特征向量,所以幂方法并不能发现 12 作为我们希望的第一个(最大的)特征值。由于改变了初始猜测(弗朗克林建议使用"异乎寻常的"猜测),我们紧接着遇到了两个特征值 6 和 −6 它们有相同的量值的麻烦。借助于一漂移的使用,这个结果是固定不变的。

实际上,用户可能会发现如果收敛性已不满足,那么提供漂移或者改变在迭代中使用的初始向量是必要的。我们并没有把这种方法写成代码,也没有提供反对称矩阵中可能出现的复数特征值。如果你感到你不能改变这个代码的片段,你应该使用虽更麻烦的,但更稳健的特征值。

两个特征值的确定之间包含了一漂移,用逆迭代方法在于试图保证达到后继的最小的特征值。否则,比如说,6 被求出,那么我们就已把这个矩阵漂移到这个新的原点,新矩阵的最小特征值是 12(它距新原点的数值是 6,而 −6 距新原点的数值是 12)。

§4.5.3 C 工具

为了确定矩阵的特征值,两个程序被包含,不是通过带有雷利商漂移的逆迭代,就是借助置前迭代。

这个程序在末尾包括了一个克莱姆——施密特(Gram—Schmidt)正交化步骤以保证特征向量的正交性。克莱姆——施密特方法涉及到从一个向量减去它在另一个向量上的

投影，结果得到一合向量，这个合向量在其它向量上的投影是 0。克莱姆——施密特方法从一已知向量开始，用这个已知向量来正交化紧接着的向量，然后再用这最初两个向量正交化下一个向量……，依次下去，由于每个特征向量与它前面的所有特征向量正交。这种方法能够产生依赖于原来的向量序列的不同答案。理论上讲，克莱姆——施密特步骤应该对特征值没有影响，由于不用此方法，特征向量也应该是正交的。这当然假定了矩阵是对称的，从而特征向量应该正交——对于非对称矩阵应该删去特征向量的正交性。克莱姆——施密特步骤对于求出一个特征向量后，马上就进行正交化可能更有效。

用来求最小的特征值的程序中，尽管矩阵是满的，但雷利商迭代仍被使用。实际上，对于阶大而满秩的矩阵而言，逆迭代可能比雷利商更有效。在这种情况下，程序 lufact() 用矩阵 A 的 LU 因子来取代 A 的信号应该在迭代内移动，并且把这个信号放在 A 的前面，当然，通过从 A 的对角线中减去雷利商后的特征值的漂移应该从程序中删除。对于满秩阵的每一迭代将需要计算阶数是 n^2 而不是 n^3 的运算时间，这种迭代显然是有意义的。

对于没有包括在前面几章的向量和矩阵代数，这章的程序列举包括了一些用于一般目的的程序。这些程序包括了求出两向量差的 vdif() 程序，交换一个矩阵的行和列的 Swaprow() 程序和 Swapcol()（当我们用特征向量改变最大量值元素的位置，从而相应地调整矩阵时，这些程序用于装枢轴）；dot() 程序形成一个点积（在第三章，我们使用有关 BLAS 程序的一流形式，这儿用的 dot 假定对每个矩阵的相邻元素都可使用）；打印一个向量的程序 pv()，一向量乘以矩阵的 mvc() 程序；通过确定一个向量的长度（范数），然后用这个向量的每个元素乘以那个范数的方法使这个向量标准化的程序 normv()；一个向量乘以一个矩阵的转置（这种方法在矩阵不是对称的某些问题中将是有用的）的程序 mvtc()；决定残差向量 $r=AX-Y$ 的程序 resid()，一向量乘以一个标量的程序 vs()，使一个向量为一个常数的程序 vset()，复制一个向量到另一个向量的 vcopy() 程序，进行形式为 $a=b+s*c$ 的初等行运算的 vv() 程序；其中 s 是一标量和 a,b,c 都是向量（在第三章，我们使用了一 BLAS 程序的简化形式），以及复制一矩阵到另一个相似维数矩阵的 mcopy() 程序。

表 4—2　第四章的程序：QR 方法，幂和逆迭代

eigenvv	求特征值和特征向量
hqr2	对黑斯贝尔格(Hessenberg)矩阵实行 QR 迭代
elmbes	转换矩阵成上(upper)黑斯贝尔格矩阵形式
eltran	变换特征向量，对于原矩阵那些的特征向量
balance	平衡矩阵
balbak	平衡变换的逆
pnormwr	打印标准化特征向量幂和逆迭代
esmall()	决定最小特征值的雷利商迭代(这种方法一般地比使用程序 eig… 更可取)
elarge()	确定最大特征值和相应特征向量
Vdiff()	求两个向量的差向量
Swaprow()	交换一个矩阵的行

Swapcol	交换一矩阵的列
mvc	象 mv 一样用向量乘以矩阵，但可推广到允许该矩阵标明矩阵的列数 n，而乘法运算仅对这些 n 起作用。
mvtc	类似于 mvc 但向量乘以的是矩阵的转置矩阵。

§4.6 程序列表与测试问题输出

```
/*
matrix eigenvalues/eigenvectors via QR decomposition

CONTENTS:

eigenvv()
   computes eigenvalues and eigenvectors of a matrix
hqr2()
   applies QR method to Hessenberg matrix
elmhes()
   converts general matrix to Hessenberg form
eltran()
   creates transform. matrix from elmhes output
balance()
  balances a matrix
balbak()
   inverse transform of balance(used on eigenvectors)
pnormwr()
   normalize and print eigenvectors. largest element=1.
printmd()
   prints type double 2D matrix

DEPENDENCIES:
NONE
*/

/* defines below are from ftoc.h except for IU,DFFOR,DFFR,DFFRR */
#define INDEX(i,j) [j+(i)*coln]
#define IU(i,j) [j+(i)*colu]
#define DOFOR(i,j) for(i=0;i<j;i++)
#define DFFOR(i,from,to) for(i=from;i<=to;i++)
#define DFFR(i,from,to) for(i=from;i<=to;i++)
#define DFFRR(i,from,to) for(i=from;i>=to;i--)
#define min(a,b)(((a)<(b))? (a):(b))
#define max(a,b)(((a)<(b))? (b):(a))
/* #define abs(x) (((x>0.)? (x):-(x)) using fabs */
```

```
#define itmax 30

printmd(a,coln,rown,col,row)int rown,row,col,coln;double a[];
{
  int i,j,btm,top,count;
  printf("\n");
  btm=top=0;
  while(btm<col)
  {
   top=min(col,(cbtm+8));
   printf("printing matrix columns %d to%d\n",btm,(top-1));
   DOFOR(j,row)
   {
    for(i=btm;i<top;i++)
    {
      printf("%e",a INDEX(j,i));
    }
    printf("\n");
   }
   btm+=8;
  }
  return;
}

double macheps;

eigenvv(coln,n,a,iwork,scale,z,wr,wi,bal,cnt)
int coln,n,iwork[],bal,cnt[];
double *a, *scale, *z,wr[],wi[];
{
int i,low,high,m,ierr;
if(bal)
   {
   balance(coln,n,a,&low,&high,scale);
   printmd(a,coln,n,n,n);
/* DOFOR(i,n)printf("scale=%e\n",scale[i]);
  printf("balance low=%d high=%d\n",low,high); */
   }
else
   {
   low=0;
   high=n-1;
   }
elmhes(coln,n,low,high,a,iwork);
printmd(a,coln,n,n,n);
printf("low=%d high=%d\n",low,high);
/* DOFOR(i,n)printf("intar=%d\n",iwork[i]); */
```

```
eltran(coln,n,low,high,a,iwork,z);
printmd(z,coln,n,n,n);
  hqr2(coln,n,low,high,a,wr,wi,z,&ierr,cnt);
  if(ierr) printf("ierr=%d\n",ierr);
  m=n;/* unless fewer returned? */
/* hqr2 uses z to transform eigenvectors which are returned in z */
if(ierr)return;
if(bal)
    {
    balbak(coln,n,low,high,scale,m,z);
    }
}

hqr2(coln,n,low,igh,h,wr,wi,vecs,ierr,cnt)
int coln,n,low,igh, *ierr,cnt[]
double *vecs, *h,wr[],wi[];
{
/* coln,n have "FORTRAN" values
low,igh have "FORTRAN-1" values */

  int i,j,k,l,m,ii,jj,kk,mm,na,nm,nn,its,mpr,enm2,en;
  int it1=10,it2=20,itlim=30,notlast;
  double p,q,r,s,t,w,x,y,z,ra,sa,vi,vr,zz,norm;
  double xr,yr,xi,yi,zr,sqrt();
  *ierr=0;
/* balance must have been called first! */
/* calculation of norm here as in EISPACK FORTRAN code */
  norm=0.;
  k=0;

  DOFOR (i,n)
  {
   if(i<low || i>igh)
   {
     wr[i]=h INDEX(i,i);
     wi[i]=0.0;
   }
   for(j=k;j<n;j++) norm+=fabs(h INDEX(i,j));
   k=i;
   cnt[i]=0;
  }
  en=igh;
  t=0.;
nextw:
  if(en>=low)
  {
    its=0;
```

```
    na=en-1;
    enm2=na-1;
nexit:
    DFFRR(1,en,low)/*algol code had one less iteration than ftn*/
    {/* as in EISPACK */
    s=fabs(h INDEX(1-1,1-1))+fabs(h INDEX(1,1));
    if(s==0.)s=norm;
      if(fabs(h INDEX(1,1-1))<=s*macheps)goto break1;
    }
    1=low;
    break1;
    x= h INDEX(en,en);
    if(1==en)
    {/*found a root*/
/*printf("single root its=%d\n",its);*/
      wr[en]=x+t;
      wi[en]=0.;
      h INDEX(en,en)=x+t;
      en=na;
      cnt[en]=its;
      goto nextw;
    }
    y= h INDEX(na,na);
    w= h INDEX(en,na)*h INDEX(na,en);
    if(t1 ==na)
    {
      if(its==itlim)
      {
          cnt[en]=3;
          *ierr=en;
          return;
      }
      if(its==it1 || its=it2)
      {
        t+=x;
        DFFR(i,low,en)h INDEX(i,i)-=x;
        s= fabs(h INDEX(en,na))+fabs(h INDEX(na,en-2));
        x= .75*s;
        y=x;
        w=-0.4375*s*s;
      }
      its++;
/*printf("its=%d\n",its);*/
      for(m=en-2;m>=1;m--)
      {
        mm=m+1;
        z= h INDEX(m,m);
```

```
        r=x-z;
        s=y-z;
        p=(r*s-w)/(h INDEX(mm,m))+h INDEX(m,mm);
        q=h INDEX(mm,mm)-z-r-s;
        r=h INDEX(m+2,mm);
        s=fabs(p)+fabs(q)+fabs(r);
        p/=s;
        q/=s;
        r/=s;
        if(m==l)break;
      if(fabs(h INDEX(m,m-1))*(fabs(q)+fabs(r))<=
        macheps*fabs(p)*
        (fabs(h INDEX(m-1,m-1))+fabs(z)+fabs(h INDEX(mm,mm))
        )break;
}

for(i=m+2;i<=en;i++)h INDEX(i,i-2)=0.;
for(i=m+3;i<=en;i++)h INDEX(i,i-3)=0.;
for(k=m;k<=na;k++)
{
    notlast=(k!=na);
    if(k!=m)
    {
       p=h INDEX(k,k-1);
       q=h INDEX(k+1,k-1);
       r=notlast?h INDEX(k+2,k-1):0.;
       x=fabs(r)+fabs(q)+fabs(p);
       if(x==0.)break;/*exit for loop*/
       p/=x;q/=x;r/=x;
    }
    s=sqrt(p*p+q*q+r*r);
    if(p<0.)s=-s;
    if(k!=m)
      h INDEX(k,k-1)=-s*x;
    else
      if(l!=m)
      h INDEX(k,k-1)=-h INDEX(k,k-1);
    p+=s;
    x=p/s;y=q/s;z=r/s;q/=p;r/=p;
    /* row modification */
    for(j=k;j<n;j++)
    {
       p=h INDEX(k,j)+q*h INDEX(k+1,j);
       if(notlast)
       {
          p+=r*h INDEX(k+2,j);
          h INDEX(k+2,j)-=p*z;
```

```
        }
      h INDEX(k+1,j) -=p * y;
      h INDEX(k,j) -=p * x;
     }
     j=(k+3)<en ? k+3 : en;
     /* column mods */
     DFFR(i,0,j)
     {
      p=x * h INDEX(i,k)+y * h INDEX(i,k+1);
      if (notlast)
      {
          p+=z * h INDEX(i,k+2);
          h INDEX(i,k+2) -=p * r;
      }
     h INDEX(i,k+1) -=p * q;
     h INDEX(i,k) -=p;
     }
     /* accumulate */
     for(i=low;i<=igh;i++)
     {
       p=x * vecs INDEX(i,k)+y * vecs INDEX(i,k+1);
       if(notlast)
       {
          p+=z * vecs INDEX(i,k+2);
          vecs INDEX(i,k+2) -=p * r;
       }
      vecs INDEX(i,k+1) -=p * q;
      vecs INDEX(i,k) -=p;
      }
    }/* end of for k loop */
    goto nextit;
   }/* 1 =na block */
/* else,l=na,tow roots found */
/* printf("double its=%d\n",its); */
     p=.5 * (y-x);
     q=p * p+w;
     z=sqrt(fabs(q));
     x+=t;
     h INDEX(en,en)=x;
     h INDEX(na,na)=y+t;
    cnt[en]=-its;cnt[na]=its;
     if (q>=0.) /* q>0 in algol */
      {/* real pair */
        z= p<0. ? p-z : p+z;
        wr[na]=x+z;
        wr[en]=wr[na]; /* EISPACK version */
        if(z ! =0.)
```

```
                {
                s=x-w/z;
                wr[en]=s;
                }
            wi[en]=0.;
            wi[na]=0.;
            x=h INDEX(en,na);
            /* r=sqrt(x*x+z*z);
            p=x/r;
            q=z/r;
            algol version */
            s= fabs(x)+fabs(z);
            p=x/s;
            q=z/s;
        r=sqrt(p*p+q*q);
            p/=r;
            q/=r;
            for(j=na;j<n;j++)
            {
              z=h INDEX(na,j);
              h INDEX(na,j)=q*z+p*h INDEX(en,j);
              h INDEX(en,j)=q*h INDEX(en,j)-p*z;
            }
            DFFR(i,0,en)
            {
              z=h INDEX(i,na);
              h INDEX(i,na)=q*z+p*h INDEX(i,en);
              h INDEX(i,en)=q*h INDEX(i,en)-p*z;
            }
            for(i=low;i<=igh;i++)
            {
              z=vecs INDEX(i,na);
              vecs INDEX(i,na)=q*z+p*vecs INDEX(i,en);
              vecs INDEX(i,en)=q*vecs INDEX(i,en)-p*z;
            }
          } /* end if real pair */
          else /*complex pair*/
          {
            wr[en]=x+p;
            wr[na]=x+p;
            wi[na]=z;
            wi[en]=-z;
          }
          en--;en--;
          goto nextw;
        }
```

```
/* all eigenvalues found */

  for(en=n-1;en>=0;en--)
  {
    p=wr[en];
    q=wi[en];
    na=en-1;
    if(q==0.)
    {
      /* real vector */
      m=en;
      h INDEX(en,en)=1.;
      for(i=na;i>=0;i--)
      {
        w=h INDEX(i,i)-p;
        r=h INDEX(i,en);
        DFFR(j,m,na)
        {
        r+=h INDEX(i,j)*h INDEX(j,en);
        }
        if(wi[i]<0.0)
        {
        z=w;
        s=r;
        }
        else
        {
        m=i;
        if(wi[i]==0.)
        {
          h INDEX(i,en)= -r/((w!=0.)? w : macheps*norm);
        }
        else
        {
          x=h INDEX(i,i+1);
          y=h INDEX(i+1,i);
          q=(wr[i]-p);
          yr=wi[i];
          q=q*q+yr*yr;
          t=(x*s-x*r)/q;
          h INDEX(i,en)=t;
          h INDEX(i+1,en)=(fabs(x)>fabs(z))?
            (-r-w*t)/x :(-s-y*t)/z;
        }/*wi!=0*/
       }/*wi>=0*/
     }/*for i loop*/
   }/*real vector,q=0*/
```

```
else
{ if(q<0.)
  {
    m=na;
    /* in algol,was -q,with xi=0.
    EISPACK commment says this alternate choice
    was to make last eigenvector component imag
    so that eigenvector matrix triangular */
    if(fabs(h INDEX(en,na))>fabs(h INDEX(na,en)))
    {
      h INDEX(na,en)=-(h INDEX(en,en)-p)/h INDEX(en,na);
      h INDEX(na,na)=q/h INDEX(en,na);
    }
    else
    {
      xr=0.;
      xi=- h INDEX(na,en);
      yr= h INDEX(na,na)-p;
      yi=q;
      if(fabs(yr)>fabs(yi))
      {
        t=yi/yr;
        yr+=t*yi;
        zr=(xr+t*xi)/yr;
        zi=(xi-t*xr)/yr;
      }
      else
      {
        t=yr/yi;
        yi+=t*yr;
        zr=(t*xr+xi)/yi;
        zi=(t*xi-xr)/yi;
      }
      h INDEX(na,na)=zr;
      h INDEX(na,en)=zi;
    }
    h INDEX(en,na)=0.;
    h INDEX(en,en)=1.;
    for(i=na-1;i>=0;i--)
    {
      w= h INDEX(i,i)-p;
      /* caveat- in algol,the assignments to ra,sa
      below were switched. This code is as
      in EISPACK */
      sa= h INDEX(i,en);
      ra=0.;
      for (j=m;j<=na;j++)
```

```
{
  ra+= h INDEX(i,j) * h INDEX(j,na);
  sa+= h INDEX(i,j) * h INDEX(j,en);
}
if(wi[i]<0.)
{
  z=w;
  r=ra;
  s=sa;
}
else
{
  m=1;
  if(wi[i]==0.)
  {
     /* cdiv complex division */
     /* more expensive but more robust
     than version in COMPLEX.H */
     xr=-ra;
     xi=-sa;
     yr=w;
     yi=q;
     if(fabs(yr)>fabs(yi))
     {
       t=yi/yr;
       yr+=t * yi;
       zr=(xr+t * xi)/yr;
       zi=(xi-t * xr)/yr;
     }
     else
     {
       t=yr/yi;
       yi+=t * yr;
       zr=(t * xr+xi)/yi;
       zi=(t * xi-xr)/yi;
     }
     h INDEX(i,na)=zr;
     h INDEX(i,en)=zi;
}
else
{
  x=h INDEX(i,i+1);
  y=h INDEX(i+1,i);
  vi=wi[i];
  t=(wr[i]-p);
  vr=t * t+vi * vi-q * q;
  vi=2.0 * q * t;
```

```
if (vr == 0. && vi == 0. )
{
   vr = macheps * norm *
      (fabs(w)+fabs(q)+fabs(x)+fabs(y)+fabs(z));
}/ * zero vr and vi * /
   / * cdiv * /
   xr =  x * r - z * ra + q * sa;
   xi = s * x - z * sa - q * ra;
   yr = vr;
   yi = vi;
   if (fabs(yr) > fabs(yi))
   {
      t = yi/yr;
      yr += t * yi;
      zr = (xr + t * xi)/yr;
      zi = (xi - t * xr)/yr;
   }
   else
   {
      t = yr/yi;
      yi += t * yr;
      zr = (t * xr + xi)/yi;
      zi = (t * xi - xr)/yi;
   }
   h INDEX(i,na) = zr;
   h INDEX(i,en) = zi;
if (fabs(x) > (fabs(z) + fabs(q)))
{
   h INDEX(i+1,na) =
      (-ra - w * h INDEX(i,na) + q *
      h INDEX(i,en))/x;
   h INDEX(i+1,en) =
      (-sa - w * h INDEX(i,en) - q *
      h INDEX(i,na))/x;
}
else
{
  / * cdiv * /
  xr = -r - y *  h INDEX(i,na);
  xi = -s - y *  h INDEX(i,en);
  yr = z;
  yi = q;
  if (fabs(yr) > fabs(yi))
  {
     t = yi/yr;
     yr += t * yi;
     zr = (xr + t * xi)/yr;
```

```
                    zi=(xi-t*xr)/yr;
                  }
                  else
                  {
                    t=yr/yi;
                    yi+=t*yr;
                    zr=(t*xr+xi)/yi;
                    zi=(t=xi-xr)/yi;
                  }
                  h INDEX(i+1,na)=zr;
                  h INDEX(i+1,en)=zi;
                }
              }/* wi{ =0*/
            }
          }/*for i*/

        }/*q<0. complex vector*/
      }/*end else*/
    }/*end for*/
  /* vector of roots*/
    DOFOR(i,n)
    {
    If(i<low || i>igh)
    {
     for(j=i;j<n;j++)
     {
       vecs INDEX(i,j)=h INDEX(i,j);
     }
    }

  }
  for(j=n-1;j>=low;j--)
  {
    m= min(j,igh);
    DFFR(i,low,igh)
      {
    z=0.;DFFR(k,low,m)z += vecs INDEX(i,k) * h INDEX(k,j);
    vecs INDEX(i,j)=z;
      }
  }/*end for j loop*/
  return;
}

elmhes(colm,n,low,igh,a,intar)
int colm,n,low,igh,intar[];
double *a;
{
```

```
    int i,j,m,la,kp1,mm1,mp1;
    double x,y;
    int coln;
    coln=n;
    la=igh-1;
    kp1=low+1;
    if(la<kp1)return;
    DFFR(m,kp1,la)
    {
      mm1=m-1;
      x=0.;
      i=m;
      DFFR(j,m,igh)
    {
      if(fabs(a INDEX(j,mm1))>fabs(x))
      {
        x= a INDEX(j,mm1);
        i=j;
      }
    }
    intar[m]=i;
  if(i! =m)
  {
    DFFOR(j,mm1,n)
    {
      y=a INDEX(i,j);
      a INDEX(i,j)=a INDEX(m,j);
      a INDEX(m,j)=y;
    }
    DFFR(j,0,igh)
    {
      y= a INDEX(j,i);
      a INDEX(j,i)= a INDEX(j,m);
      a INDEX(j,m)= y;
    }
  }
  if(x! =0.)
  {
    mp1=m+1;
    DFFR(i,mp1,igh)
    {
    y=a INDEX(i,mm1);
    if(y! =0.)
    {
     y/=x;
     a INDEX(i,mm1)=y;
     DFFOR(j,mp1,n) a INDEX(i,j) -=y* a INDEX(m,j);
```

```
      DFFR(j,0,igh) a INDEX(j,m) += y * a INDEX(j,i);
    }
   }
}
   }
}

eltran(coln,n,low,igh,a,intar,z)
int coln,low,igh,n,intar[];
double *a, *z;
{/* form matrix of accumulated transforms z */
  int colu,i,j,k,l,k1,mn,mp,mp1;
/* set z(n x n) to identity matrix; */
  colu=n;
  DOFOR(i,n)
  {
   DOFOR(j,n) z IU(i,j) =0.;
   z IU(i,j)=1.;
  }
  k1=igh-low-1;
  if(k1<1)return;
  DFFRR(i,(igh-1),(low+1))
  {
    j= intar[i];
    DFFR(k,i+1,igh) z IU(k,i)=a INDEX(k,i-1);
    if(i! =j)
    {
       DFFR(k,i,igh)
       {
        z IU(i,k) =z IU(j,k);
        z IU(j,k) =0.;
       }
       z IU(j,i)=1.;
    }
   }
}
/* not used as hqr2 does it
elmbak(colu,n,m,low,igh,a,intar,z)
int colu,n,low,igh,m,intar[];
double *z, *a;
{
  int coln,i,j,la,l,mm,mp,kp1,mp1;
  double x;
  if(m<0)return;
  la=igh-1;
  kp1=low+1;
  coln=n;
```

```
  if(la<kp1)return;
  DFFRR(mp,i-1,kp1)
  {
    mp1=mp+1;
    for (i=mp1;i<=igh;i++)
    {
      x= a IU(i,mp-1);
      if(x! =0.0)
      {
         DOFOR(j,m)
         z INDEX(i,j)+=x*z INDEX(mp,j);
      }
    }
    i=intar[mp];
    if(i! =mp)
    {
        DOFOR(j,m)
        {
         x=z INDEX(i,j);
         z INDEX(i,j)= z INDEX(mp,j);
         z INDEX(mp,j)= x;
        }
      }
  }
}
*/
  balbak(coln,n,low,igh,scale,m,z)
  int coln,n,low,igh;double scale[], *z;
  {/* m is number of eigenvectors to transform */
    int i,j,k,l;
    double s;
    if(m==0)return;
    if(igh! =low)
    {
      DFFR(i,low,igh)
      {
       s=scale[i];
       DOFOR(j,m)z INDEX(i,j) *=s;
      }
    }
    DOFOR(i,n)
    {
     i=l;
     if(i<low || i>igh)
     { /* i=0 i=low-1 i=low-1 i=0 */
       if(i<low)i=low-l-1;
       k=scale[i];
```

```
        if(k! =i)
            {
            DOFOR(j,m)
              {
               s= z INDEX(i,j);
               z INDEX(i,j)=z INDEX(k,j);
               z INDEX(k,j)=s;
              }
            }
      }
   }
   return;
}

int k,i,j,m,nn;

exch(m,d,a)
int m;
double d[], *a;
{
       double f;
       int i,coln;
       coln=nn;
       d[m]=j;
/* note an integer is being stored in scale, d type double array */
     if(j! =m)
     {
       DFFR(1,0,k)
       {
         f=a INDEX(i,j);
         a INDEX(i,j)=a INDEX(i,m);
         a INDEX(i,m)=f;
       }
DFFOR(j,1,nn)
       {
         f= a INDEX(j,i);
         a INDEX(j,i)=a INDEX(m,i);
         a INDEX(m,i)=f;
       }
     }
}

balance(coln,n,a,low,igh,scale)
double scale[], *a;
int coln,n, *low, *igh;
{/* assumed matrix a has coln columns,working on n x n portion */
   double radix=2.,b2,bi,b2i,r,c,b,s,f,g;
```

```
    int l,kt,noconv ,lim;
    l=0;
    k=n-1;
    nn=n;
    lim= max(20,n * 2);
    b=radix;
    b2=radix * radix;
    bi=1. /b;
    b2i=1. /b2;
    kt=0;
11;
   DFFRR(j,k,0)
   {
     r=0. ;
     DFFR(i,0,k)
     {
       if(i! =j)
       {
         r+=fabs(a INDEX(j,i));
       }
     }
     if(r==0. )
     {
       exch(k,scale,a);
       k--;
       kt++;
       if(kt<lim)goto 11;
       else
          {
          exit(0);
          }
     }
   }
   kt=0;
12;
   DFFR(j,1,k)
   {
     c=0. ;
     DFFR(i,1,k)
     {
       if(i! =j) c+=fabs(a INDEX(i,j));
     }
     if (c==0. )
     {
       exch(l,scale,a);
       l++;
       kt++;
```

```
        if(kt<lim)goto 12;
        else
           {
              {
              exit(0);
              }
           }
      }
      *low=l;
      *igh=k;
      DFFR(i,l,k) scale[i]=1.;
      kt=0;
iteration:
      noconv=0;
      DFFR(i,l,k)
      {
        r=0.;
        c=0.;
        DFFR(j,1,k)
        {
          if(i1 =j)
          {
            c+=fabs(a INDEX(j,i));
        r+=fabs(a INDEX(i,j));
      }
    }
        g=r/b;
        f=1.;
      s=c+r;
      if(c==0. || r==0.) continue;
      /* above if added in FORTRAN avoids infinite loops */
         13: if(c<g)
      {
            f*=b;c*=b2;goto 13;
          }
          g=r*b;
          14:if(c>=g)
      {
          f*=bi;c*=b2i;goto 14;
          }
          if(c+r)/f <.95*s)
          {
            g=1./f;
            scale[i]*=f;
            noconv=1;
            DFFOR(j,l,n) a INDEX(i,j) *=g;
            DFFOR(j,0,k) a INDEX(j,i) *=f;
```

```
        }
      }/* do i */
      kt++;
      if(noconv && kt>lim)
            {
            printf("balance iteration limit\n");
            exit(0);
            }
   if (noconv)goto iteration;
  }

pnormwr(coln,n,z,wr,wi)
double *z,wr[],wi[];
int coln,n;
{/* normalize as in Wilkinson & Reinsch with largest element=1 */
int i,j,k;
double t,xr,xi,yi,yr,zr,zi,emax,ei;
print("un normalized matrix:\n");
printmd(z,coln,n,n,n);

DOFOR(i,n)
  {
  if(wi[i]==0.)
     {
     emax=0.;
     DOFOR(j,n)
       {
       if (fabs(z INDEX(j,i))>fabs(emax))
                    emax=z INDEX(j,i);
       }
     if(emax! =0.)
       {
       emax+1./emax;
       DOFOR(j,n)
         {
         z INDEX(j,i) *=emax;
         }
       }
     }
  else /* complex eigenvector */
     {
     emax=0.;
     k=0;
     DOFOR(j,n)
       {
       xr=z INDEX(j,i) ; yr=z INDEX(j,i+1);
       xr=sqrt(xr*xr+yr*yr);
```

```
        if(xr>emax)
             {
             emax=xr;
             k=j;
             }
        }
      emax=z INDEX(k,i);
      ei=z INDEX(k,i+1);

      DOFOR(j,n)
        {
        yr=emax;
        yi=ei;
        xr=z INDEX(j,i);
        xi=z INDEX(j,i+1);
                if(fabs(yr)>fabs(yi))
                {
                  t=yi/yr;
                  yr+=t*yi;
                  zr=(xr+t*xi)/yr;
                  zi=(xi-t*xr)/yr;
                }
                else
                 {
                  t=yr/yi;
                  yi+=t*yr;
                  zr=(t*xr+xi)/yi;
                  zi=(t*xi-xr)/yi;
                 }
          z INDEX(j,i)=zr;
          z INDEX(j,i+1)=zi;
          }
      i++;
      }
    } /* DOFOR I loop */
  printf("normalized matrix of eigenvectors:\n");
  printmd(z,coln,n,n,n);

  }
  /* specify matrices by row */
  double et[4][4]={1.,2.,3.,5.,2.,4.,1.,6.,1.,2.,-1.,3.,2.,0.,1.,3.};
  double ev[4][4]={3.,1.,2.,5.,2.,1.,3.,-1.,0.,4.,1.,1.,0.,0.,2.,1.};
  double ex[3][3]={2.,4.,4.,0.,3.,1.,0.,1.,3.};
  double ey[3][3]={5.,-4.,-7.,-4.,2.,-4.,-7.,-4.,5.};

  main(argc,argv)
  int argc;
```

```
char * * argv;
{/ * test driver * /
double wr[4],wi[4],z[16],d[4];
float mach;
int i,j,iwork[4],cnt[4],bal;
elmhes(4,4,0,3,et,iwork);
printf("elmhes test\n");
printmd(et,4,4,4,4);
printf("enter macheps");scanf("%e",&mach);
macheps=mach;
printf(" macheps=%e\n",macheps);
printf("eigenv test\n");
bal=1;
eigenvv(4,4,ev,iwork,d,z,wr,wi,bal,cnt);
DOFOR(i,4)
  printf("eigenvalue real=%e imag=%e it=%d\n",wr[i],wi[i],cnt[i]);
printmd(z,4,4,4,4);
pnormwr(4,4,z,wr,wi);
printf("eigenv test 2\n");
bal=0;
eigenvv(3,3,ex,iwork,d,z,wr,wi,bal,cnt);
DOFOR(i,3)
  printf("eigenvalue real=%e imag=%e it=%d\n",wr[i],wi[i],cnt[i]);
printmd(z,3,3,3,3);
pnormwr(3,3,z,wr,wi);
printf("eign test 3\n");
bal=1;
eigenvv(3,3,ey,iwork,d,z,wr,wi,bal,cnt);
DOFOR(i,3)
  printf("eigenvalue real=%e imag=%e it=%d\n",wr[i],wi[i],cnt[i]);
printmd(z,3,3,3.3);
pnormwr(3,3,z,wr,wi);
}
```

```
elmhes test

printing matrix columns 0 to 3
1.000000e+000 8.500000e+000 5.321429e+000 3.000000e+000
2.000000e+000 1.050000e+001 6.107143e+000 1.000000e+000
5.000000e-001 -7.000000e+000 -3.000000e+000 0.000000e+000
1.000000e+000 1.071429e-001 1.607143e-001 -1.500000e+000
enter macheps macheps=1.000000e-007
eigenv test

printing matrix columns 0 to 3
3.000000e+000 5.000000e-001 1.000000e+000 2.500000e+000
```

```
4. 000000e+000 1. 000000e+000 3. 000000e+000 -1. 000000e+000
0. 000000e+000 4. 000000e+000 1. 000000e+000 1. 000000e+000
0. 000000e+000 0. 000000e+000 2. 000000e+000 1. 000000e+000

printing matrix columns 0 to 3
3. 000000e+000 5. 000000e-001 1. 000000e+000 2. 500000e+000
4. 000000e+000 1. 000000e+000 3. 000000e+000 -1. 000000e+000
0. 000000e+000 4. 000000e+000 1. 000000e+000 1. 000000e+000
0. 000000e+000 0. 000000e+000 2. 000000e+000 1. 000000e+000
low=0 high=3

printing matrix columns 0 to 3
1. 000000e+000 0. 000000e+000 0. 000000e+000 0. 000000e+000
0. 000000e+000 1. 000000e+000 0. 000000e+000 0. 000000e+000
0. 000000e+000 0. 000000e+000 1. 000000e+000 0. 000000e+000
0. 000000e+000 0. 000000e+000 0. 000000e+000 1. 000000e+000
eigenvalue real= 6. 077906e+000 imag=0. 000000e+000 it=0
eigenvalue real= -3. 195529e+000 imag=0. 000000e+000 it=0
eigenvalue real= 1. 558812e+000 imag=2. 001516e+000 it=4
eigenvalue real= 1. 558812e+000 imag=-2. 001516e+000 it=-4

printing matrix columns 0 to 3
9. 301637e-001 -1. 465495e-001 5. 554721e-001 -1. 429410e+000
6. 522062e-001 6. 428047e-001 -5. 311325e-001 -9. 691423e-002
5. 569601e-001 -6. 914051e-001 -2. 045459e-001 7. 045126e-001
2. 193660e-001 3. 295914e-001 6. 001342e-001 3. 719449e-001
un normalized matrix:

printing matrix columns 0 to 3
9. 301637e-001 -1. 465495e-001 5. 554721e-001 -1. 429410e+000
6. 522062e-001 6. 428047e-001 -5. 311325e-001 -9. 691423e-002
5. 569601e-001 -6. 914051e-001 -2. 045459e-001 7. 045126e-001
2. 193660e-001 3. 295914e-001 6. 001342e-001 3. 719449e-001
normalized matrix of eigenvectors:

printing matrix columns 0 to 3
1. 000000e+000 2. 119589e-001 1. 000000e+000 1. 581557e-017
7. 011735e-001 -9. 297077e-001 -6. 654547e-002 -3. 457149e-001
5. 987764e-001 1. 000000e+000 -4. 765180e-001 4. 207790e-002
2. 358359e-001 -4. 766979e-001 -8. 432142e-002 4. 526150e-001
eigenv test 2

printing matrix columns 0 to 2
2. 000000e+000 4. 000000e+000 4. 000000e+000
0. 000000e+000 3. 000000e+000 1. 000000e+000
0. 000000e+000 1. 000000e+000 3. 000000e+000
low=0 high=2
```

```
printing matrix columns 0 to 2
1.000000e+000 0.000000e+000 0.000000e+000
0.000000e+000 1.000000e+000 0.000000e+000
0.000000e+000 0.000000e+000 1.000000e+000
eigenvalue real=2.000000e+000 imag=0.000000e+000 it=0
eigenvalue real=4.000000e+000 imag=0.000000e+000 it=0
eigenvalue real=2.000000e+000 imag=0.000000e+000 it=0

printing matrix columns 0 to 2
1.000000e+000 2.828427e+000 0.000000e+000
0.000000e+000 7.071068e-001 -7.071068e-001
0.000000e+000 7.071068e-001 7.071068e-001
un normalized matrix:

printing matrix columns 0 to 2
1.000000e+000 2.828427e+000 0.000000e+000
0.000000e+000 7.071068e-001 -7.071068e-001
0.000000e+000 7.071068e-001 7.071068e-001
normalized matrix of eigenvectors:

printing matrix columns 0 to 2
1.000000e+000 1.000000e+000 -0.000000e+000
0.000000e+000 2.500000e-001 1.000000e+000
0.000000e+000 2.500000e-001 -1.000000e+000
eign test 3

printing matrix columns 0 to 2
5.000000e+000 -4.000000e+000 -7.000000e+000
-4.000000e+000 2.000000e+000 -4.000000e+000
-7.000000e+000 -4.000000e+000 5.000000e+000

printing matrix columns 0 to 2
5.000000e+000 -9.285714e+000 -4.000000e+000
-7.000000e+000 2.714286e+000 -4.000000e+000
5.714286e-001 -4.408163e+000 4.285714e+000
low=0 high=2

printing matrix columns 0 to 2
1.000000e+000 0.000000e+000 0.000000e+000
0.000000e+000 5.714286e-001 1.000000e+000
0.000000e+000 1.000000e+000 0.000000e+000
eigenvalue real=-6.000000e+000 imag=0.000000e+000 it=0
eigenvalue real=1.200000e+001 imag=0.000000e+000 it=0
eigenvalue real=6.000000e+000 imag=0.000000e+000 it=0

printing matrix columns 0 to 2
```

```
-6.767155e-001 6.595145e-001 -3.734378e-001
-6.767155e-001 2.452205e-014 7.468756e-001
-6.767155e-001 -6.595145e-001 -3.734378e-001
un normalized matrix:

printing matrix columns 0 to 2
-6.767155e-001 6.595145e-001 -3.734378e-001
-6.767155e-001 2.452205e-014 7.468756e-001
-6.767155e-001 -6.595145e-001 -3.734378e-001
normalized matrix of eigenvectors:

printing matrix columns 0 to 2
1.000000e+000 -1.000000e+000 -5.000000e-001
1.000000e+000 -3.718198e-014 1.000000e+000
1.000000e+000 1.000000e+000 -5.000000e-001



/*
find smallest eigenvalues of a symmetric square matrix
uses Rayleigh quotient iteration (easily changed to inverse iteration)

CONTENTS:

int esmall(a,value,vector,work,worka,pivot,workp,n,m,eigvect)
     matrix a(n,n). Find the m eigenvalues of smallest(see text)
     absolute value. If eigvect nonzero,return eigenvalues
     as well in vector[]. Eigenvalues returned in value.
     Returns count of eigenvalues found. Other arguments work space.

DEPENDENCIES:

ftoc.h header file required
vector.c routines
linear algebra routines in luf.c,lus.c,luback.c
*/

#include <ftoc.h>

#define wait 6
#define waitrs 3
#define maxit 100
#define tola 1.e-9
#define tolr .00000
#define tola2 .000000
#define tolr2 .000000
```

```
int esmall(a,value,vector,work,worka,pivot,workp,n,m,eigvect)
int n,m,pivot[],workp[],eigvect;
float a[],value[],vector[],work[],worka[];
/* find m smallest eigenvalues of symmetric matrix a(n,n)
by inverse iteration Rayleigh quotient iteration */
/* pivot,work must be at least n long */
/* the i-th eigenvalue is returned in value[i-1]
   "      vector is in the i-th ROW of vectors
note that the rows may have been swapped. This info is in pivot.
eigvect=0 if we only want eigenvalues.

*/
{
double sqrt()
double enew,eold;
double alpha,resid,deltrq,rnew,dot(),sqnor(),rq,rqold,shift,shifto;
int iflag,info,i,j,it,ke,nn,nn1,coln,index,k;
float *v1,sdot();
nn=n;
coln=n;
index=0;
rqold=1.e10;
rnew=resid=1.e6;
DOFOR(i,m*n) vector[i]=0.;
DOFOR(ke,m)
  {
  nn1=nn-1;
  shift=0.;
  /* will it still work midway between two eigenvalues? YES */
  if(ke>0)shift=value[ke-1];
      /* above is attempt to ensure the smallest m eigenvalues
        are found */
  shifto=shift;
  v1=&(vector[coln*ke]);
  if(nn==1)
      {value[ke]=a[0];
      v1[0]=1.00;
/* set single known component of eigenvector to 1. (arb.)
this will work except in the rare case its identically zero.
*/
      iflag=1;
      pivot[ke]=0;
printf("going to loner\n");
      goto loner;
      }
  vset(work,1.,n);work[0]=-.5;
  DOFOR(it,maxit)
```

```
    {
    vcopy(work,vl,nn);
    /* shift */
/* printm(a,coln,nn,nn,nn); */
    mcopy(a,worka,coln,nn,nn);
    DOFOR(index,nn) worka INDEX(index,index)-=(shift);
/* printm(worka,coln,nn,nn,nn); */
    lufact(worka,coln,nn,workp,&info);
/* printf("info=%d\n",info);pv(vl,nn); */
    if (info! =0)
            {/* we are there vl is eigenvector */
            rq=0.;/* eigenvalue=shift */
            goto convg;
            }
    backsub(worka,coln,nn,workp,vl);
/* pv(vl,nn); */
    /* Rayleigh quotient */
    rq= dot(vl,work,nn)/dot(vl,vl,nn);
/* printf(" * shift=%f,rq=%f\n",shift,rq); */
    alpha=1./sqrt(sqnor(vl,nn));
    vs(vl,alpha,nn);/* renormalize vl */
    alpha=1./sqrt(sqnor(work,nn));
    vs(work,alpha,nn);
    if(it>wait)
        {/* check for termination */
/* pv(vl,nn);pv(work,nn); */
        if(vl[0]*work[0]<0.)
           vs(work,-1.,nn);
        /* don't need work any more */
        vdif(vl,work,work,nn);
        rnew=sqnor(work,nn);
        deltrq=abs(rq-rqold);
/* printf("dif=");pv(work,nn);
printf(" rnew=%f,resid=%f\n",rnew,resid); */
        if(rnew*(alpha*alpha)<tolr||rnew<tola
           ||deltrq<tola2
           ||deltrq<tolr2*rq)
           {/* converged */
           convg:
           value[ke]=shift+rq;
/* printf(" eigenvalue=%f ke=%d\n",value[ke],ke);
printf(" eigenvector:");pv(vl,nn); */
           /* deflate */
/* printf(" before deflation");printm(a,coln,coln,nn,nn); */
           alpha=0.;/* find max element */
           DOFOR(i,nn)
             {resid=abs(vl[i]);
```

```
                if(resid>alpha)
                    {
                    j=i;
                    alpha=resid;
                    }
                }
            pivot[ke]=j;
          /*swap eigenv elements,a row and col j<->nn */
            index=nn-1;
            vcopy(v1,work,nn);
            if(j! =index)
                {
                resid=work[j];
                work[j]=work[index];
                work[index]=resid;
                swaprow(a,index,j,coln);
                swapcol(a,index,j,coln);
                alpha=1./resid;
                }
             else
                alpha=1./work[j];

            work[index]=1.;
            DOFOR(i,index) work[i] *=alpha;
            /* work[nn-1]=1., all other work<1 */
/*printf(" work vector=\n"); pv(work,nn); */
            DOFOR(i,index)
                {
printf(" deflating row %d\n",i);
                DOFOR(j,index)
                  {
                  a INDEX(i,j)-=
                  work[i] * a INDEX(index,j);
                  /* stmt ok DeSmet,error Aztec */
printf("work=%f,a=%f,i=%d j=%d index=%d\n"
          ,work[i],a INDEX(index,j),i,j,index);

                  }
                }
                nn--;
printf(" after deflation");printm(a,coln,coln,coln,coln);
                iflag=0;
loner:            if( eigvect && ke>0)
                {
                enew=value[ke];
                index=ke;
                for(j=coln-ke;j<coln;j++)
```

```
                {
                index--;
                eold=value[index];
                if(enew==eold)
                        {
                        printf(" degenerate eigenvalues\n");
                        goto giveup;
                        }
                alpha=dot(&(a INDEX(j,0)),vl,j)/(enew-eold);
printf(" eold%f enew %f j%d ke %d\n",eold,enew,j,ke);
printf(" alpha=%f\n",alpha);
                vl[j]=alpha;
                /* multiply by all previous t matrices,latest first;
                this is the permuted vector,not just vector!
                Don't unpermute until done */
pv(vl,j+1);
                /* modify alpha to account for fact eigenvectors
                are not stored with last component (biggest)=1.
                We do not use this last component of the prev
                eigvector in vv() below.
                */
                vcopy(&(vector[coln*(index)]),work,n);
                i=pivot[index];
                if(i!=0)
                    {
                    resid=work[i];
                    work[i]=work[j];
                    work[j]=resid;
                    }
                alpha/=work[j];
printf(" norm alpha by %f,new alpha%f\n",vector[coln*index+j],alpha);
                vv(vl,vl,work,alpha,j);
pv(vl,j+1);
pv(&(vector[coln*index]),j);
                normv(vl,j+1);
printf(" renorm v now\n");pv(vl,j+1);

                }

                if(iflag)goto fini;

                }/*if */
            giveup:break;
            }/* if converged */
/*           if(rnew>resid*1.1)
            {
printf(" diverging %f%f eigenvalues permuted\n",rnew,resid);
```

```
            return(ke);
            }
*/
          }/* it> wait */
        /*update iteration */
        resid=rnew;
        rqold=rq;
        vcopy(v1,work,nn);
        shifto=shift;
        if(it>waitrs)shift+=rq;/*Rayleigh quotient iteration */
        }/* do it */
   if(it>=(maxit-1)){/* iteration count exceeded,no conv */
          printf(" no converg,eigevalues permuted\n");
          return(ke);
          }
   if(nn==0)break;
   }/* loop over eignvalues */
fini:
DOFOR(i,m)printf(" pivot[%d]=%d\n",i,pivot[i]);

for(i=1;i<m;i++)
   {/* which element to pivot */
   /* permute eigenvalues */
   k=pivot[i-1];
   if(k)
       {
       for(j=i;j<m;j++)/*loop over eigenvectors to be affected */
            {
            index=j*coln+k;
            ke=j*coln+(coln-i) ;/*coln-1 was nn for ith eigenv. */
            alpha=vector[index];
            vector[index]=vector[ke];
            vector[ke]=alpha;
            }
       }
   }
/*Gram-Schmidt improvement orthog. with respect ot previous
eigenv. Cannot do sooner easily due to permutations */
if(eigvect)
{
DOFOR(i,m)
  {
  DOFOR(j,i)
    {
    alpha= dot(&(vector[i*coln]),&(vector[j*coln]),n);
    DOFOR(k,n) vector[i*coln+k]-=alpha*vector[j*coln+k];
    }
```

```
    normv(&(vector[j * coln]),n);
    }
  }
  return(m);
  }


/*
Driver for small eigenvalue solver

DEPENDENCIES;
  ftoc.h header file
  pv() in vector.c
  esmall(),etc. in esmall.c
*/

#include <ftoc.h>

main(argc,argv) int argc; char * *argv;
{
static float a[9]={5.,-4.,-7.,-4.,2.,-4.,-7.,-4.,5.};
/* ex 2.24 of Bathe & Wilson,eigenvalues 6,-6,12 */
float value[3],vector[9],work[9],aa[9],worka[9];
int i,n,m,iter,pivot[3],workp[3];
/* must preserve a,as deflation modifies it */
DOFOR(i,9)aa[i]=a[i];
m=esmall(aa,value,vector,work,worka,pivot,workp,3,3,1);
printf(" %d eigenvalues found\n",m);
DOFOR(i,3)
  {
  printf(" eigenvalue=%f\n",value[i]);
  }
DOFOR(i,3)
  {
  pv(&(vector[3*i]),3);
  }
exit(0);
}


deflating row 0
work=0.999806,a=-4.000000,i=0 j=0 index=2
work=0.999806,a=-4.000000,i=0 j=1 index=2
deflating row 1
work=0.999809,a=-4.000000,i=1 j=0 index=2
work=0.999809,a=-4.000000,i=1 j=1 index=2
```

```
after deflation
printing matrix columns 0 to 2
8.999224e+000 -3.000777e+000 -4.000000e+000
-3.000765e+000 8.999235e+000 -4.000000e+000
-4.000000e+000 -4.000000e+000 2.000000e+000
deflating row 0
work=0.999996,a=-3.000777,i=0 j=0 index=1
after deflation
printing matrix columns 0 to 2
1.200000e+001 -3.000765e+000 -4.000000e+000
-3.000777e+000 8.999224e+000 -4.000000e+000
-4.000000e+000 -4.000000e+000 2.000000e+000
eold -6.000000 enew 5.998459 j 2 ke 1
alpha=-0.471465
printing vector from 0 to 2
7.071082e-001 7.071054e-001 -4.714651e-001
norm alpha by 0.577314,new alpha-0.816497
printing vector from 0 to 2
2.357347e-001 2,357305e-001 -4.714651e-001
printing vector from 0 to 1
5.773123e-001 5.774244e-001
renorm v now
printing vector from 0 to 2
4.082520e-001 4.082447e-001 -8.164966e-001
going to loner
eold 5.998459 enew 12.000000 j 1 ke 2
alpha=-0.500001
printing vector from 0 to 1
1.000000e+000 -5.000010e-001
norm alpha by 0.408245,new alpha-1.224758
printing vector from 0 to 1
4.999900e-001 -5.000010e-001
printing vector from 0 to 0
4.082520e-001
renorm v now
printing vector from 0 to 1
7.070990e-001 -7.071146e-001
eold -6.000000 enew 12.000000 j 2 ke 2
alpha=0.000003
printing vector from 0 to 2
7.070990e-001 -7.071146e-001 3.457069e-006
norm alpha by 0.577314,new alpha0.000006
printing vector from 0 to 2
7.071025e-001 -7.071111e-001 3.457069e-006
printing vector from 0 to 1
5.773123e-001 5.774244e-001
renorm v now
```

```
printing vector from 0 to 2
7.071025e-001 -7.071111e-001 3.457069e-006
pivot[0]=1
pivot[1]=0
pivot[2]=0
3 eigenvalues found
eigenvalue=-6.000000
eigenvalue=5.998459
eigenvalue=12.000000
printing vector from 0 to 2
5.773123e-001 5.774244e-001 5.773140e-001
printing vector from 0 to 2
4.083044e-001 -8.164442e-001 4.082971e-001
printing vector from 0 to 2
7.071054e-001 4.927770e-006 -7.071082e-001
```

```
/*
find largest eigenvalues of a symmetric square matrix
uses vector iteration

CONTENTS:

int elarge(a,value,vector,work,worka,pivot,workp,n,m,eigvect)
    matrix a(n,n). find the m eigenvalues of smallest(see text)
    absolute value. If eigvect nonzero,return eigenvalues
    as well in vector[]. Eigenvalues returned in value.
    Function itself returns count of eigenvalues found.
    Other arguments work space.

DEPENDENCIES:

ftoc.h header file required
vector.c routines
*/

#include <ftoc.h>
/* #include"libc.h"
#include"math.h"
*/
#define INDEX(i,j) [j+(i)*coln]
#define INDEX(i,j) [j+(i)*k]
#define DOFOR(i,to) for(i=0;i<to;i++)
#define min(a,b) ((a)<(b)? (a):(b))
#define max(a,b) ((a)>(b)? (a):(b))
```

```
#define abs(a) ((a)>0? (a):-(a))

#define wait 6
#define waitrs 3
#define maxit 100
#define tola 1.e-9
#define tolr .00000
#define tola2 .000000
#define tolr2 .000000

int elarge(a,value,vector,work,worka,pivot,workp,n,m,eigvect)
int n,m,pivot[],workp[],eigvect;
float a[],value[],vector[],work[],worka[];
/* find m largest eigenvalues of symmetric matrix a(n,n)
by iteration */
/* pivot,work must be at least n long */
/* the ith eigenvalue is returned in value[i-1]
   "        vector is in the i-th ROW of vectors
note that the rows may have been swapped. This info is in pivot.
eigvect=0 if we only want eigenvalues

*/
{
double sqrt();
double enew,eold;
double alpha,resid,deltrq,rnew,dot(),sqnor(),rq,
       rqold,shift,shifto;
int iflag,info,i,j,it,ke,nn,nn1,coln,index,k;
float *vl,sdot();
nn=n;
coln=n;
index=0;
rqold=1.e10;
rnew=resid=1.e6;
DOFOR(i,m*n) vector[i]=0.;
DOFOR(ke,m)
  {
  nn1=nn-1;
  shift=0.;
  /* will it still work midway between two eigenvalues? YES */
  if(ke>0)shift=value[ke-1];
       /* above is attempt to insure the largest m eigenvalues
          are found */
  shifto=shift;
  vl=&(vector[coln*ke]);
  if(nn==1)
       {value[ke]=a[0];
```

```
        v1[0]=1.00;
/* set single known component of eigenvector to 1. (arb.)
this will work except in the rare case it's identically zero-
*/
        iflag=1;
        pivot[ke]=0;
printf(" going to loner\n");
        goto loner;
        }
      vset(work,1.,n);work[0]=-1.;
      DOFOR(it,maxit)
        {
        vcopy(work,v1,nn);
        /* shift */
/* printm(a,coln,nn,nn,nn); */
        mcopy(a,worka,coln,nn,nn);
        DOFOR(index,nn) worka INDEX(index,index)-=(shift);
/* printm(worka,coln,nn,nn,nn); */
        mvc(worka,work,v1,nn,nn,coln);
        /* Rayleigh quotient */
        rq= dot(v1,work,nn)/dot(work,work,nn);
/* printf(" shift=%f,rq=%f\n",shift,rq); */
        alpha=1./sqrt(sqnor(v1,nn));
        vs(v1,alpha,nn);/* renormalize v1 */
        alpha=1./sqrt(sqnor(work,nn));
        vs(work,alpha,nn);
        if(it>wait)
            {/* check for termination */
/* pv(v1,nn);pv(work,nn); */
            if(v1[0]*work[0]<0.)
              vs(work,-1.,nn);
            /* don't need work any more */
              vdiff(v1,work,work,nn);
              rnew=sqnor(work,nn);
              deltrq=abs(rq-rqold);
/* printf(" dif=");pv(work,nn);
printf(" rnew=%f,resid=%f\n",rnew,resid); */
            if(rnew*(alpha*alpha)<tolr||rnew<tola
              || deltrq<tola2
              || deltrq<tolr2*rq)
              {/* converged */
              convg:
              value[ke]=shift+rq;
/* printf(" eigenvalue=%f ke=%d\n",value[ke],ke);
printf(" eigenvector:");pv(v1,nn); */
              /* deflate */
/* printf(" before deflation");printm(a,coln,coln,nn,nn); */
```

```
                alpha=0. ;/* find max element */
                DOFOR(i,nn)
                  {resid=abs(v1[i]);
                  if(resid>alpha)
                      {
                      j=i;
                      alpha=resid;
                      }
                  }
              pivot[ke]=j;
            /* swap eigenv elements,a row and col j<->nn */
              index=nn-1;
              vcopy(v1,work,nn);
              if(j!  =index)
                  {
                  resid=work[j];
                  work[j]=work[index];
                  work[index]=resid;
                  swaprow(a,index,j,coln);
                  swapcol(a,index,j,coln);
                  alpha=1./resid;
                  }
              else
                  alpha=1./work[j];

              work[index]=1.;
              DOFOR(i,index) work[i]*=alpha;
              /*  work[nn-1]=1.,all other work<1 */
/*printf("work vector=\n"); pv(work,nn); */
              DOFOR(i,index)
                {
printf(" deflating row %d\n",i);
                DOFOR(j,index)
                  {
                  a INDEX(i,j)-=
                  work[i]*  a INDEX(index,j);
printf("work=%f,a=%f,i=%d,J=%d index=%d\n"
                ,work[i],a INDEX(index,j),i,j,index);
                  }
                }
                nn--;
printf(" after deflation");printm(a,coln,coln,coln);
                iflag=0;
loner:            if(eigvect && ke>0)
                {
                enew=value[ke];
                index=ke;
```

```
                for(j==coln-ke;j<coln;j++)
                {
                index--;
                eold=value[index];

                if(enew==eold)
                        {
                        printf("degenerate eigenvalues\n");
                        goto giveup;
                        }
                alpha=dot(&(a INDEX(j,0)),vl,j)/(enew-eold);
printf(" eold %f enew %f j %d ke %d\n",eold,enew,j,ke);
printf(" alpha=%f\n",alpha);
                vl[j]=alpha;
                /* multiply by all previous t matrices,latest first
                this is the permuted vector,not just vector!
                don't unpermute until done */
pv(vl,j+1);
                /* modify alpha to account for fact eigenvectors
                are not stored with last component (biggest)=1.
                we do not use this last component of the previous
                eigvector in vv() below.
                */
                vcopy(&(vector[coln*(index)]),work,n);
                i=pivot[index];
                if(i!=0)
                    {
                    resid=work[i];
                    work[i]=work[j];
                    work[j]=resid;
                    }
                alpha/=work[j];
printf(" norm alpha by %f,new alpha%f\n",vector[coln*index+j],alpha);
                 vv(vl,vl,work,alpha,j);
pv(vl,j+1);
pv(&(vector[coln*index]),j);
                 normv(vl,j+1);
printf("renorm v now\n");pv(vl,j+1);
                }
                if(iflag)goto fini;

                }/*if */
            giveup: break;
            }/* if converged */
/*           if(rnew>resid*1.1)
            {
printf(" diverging %f%f eigenvalues permuted\n",rnew,resid);
```

```
            return(ke);
            }
*/
          }/* it> wait */
        /* update iteration */
        resid=rnew;
        rqold=rq;
        vcopy(vl,work,nn);
        shiftо=shift;
        }/* do it */
  if(it>=(maxit-1)){/* iteration count exceeded,no conv */
          printf(" no converg,eigevalues permuted\n");
          return(ke);
          }
  if(nn==0)break;
  }/* loop over eignvalues */
fini;
DOFOR(i,m)printf(" pivot[%d]=%d\n",i,pivot[i]);

for(i=1;i<m;i++)
  {/* which element to pivot */
  /* permute eigenvalues */
  k=pivot[i-1];
  if(k)
       {
       for(j=i;j<m;j++)/* loop over eigenvectors to be affected */
          {
          index=j*coln+k;
          ke=j*coln+(coln-i);/* colu- i was nn for ith eigenv. */
          alpha=vector[index];
          vector[index]=vector[ke];
          vector[ke]=alpha;
          }
       }
  }
/* Gram-Schmidt improvement orthog. with respect to
previous eigenv. cannot do sooner easily due to permutations */
if(eigvect)
{
DOFOR(i,m)
  {
  DOFOR(j,i)
    {
    alpha=dot(&(vector[i*coln]),&(vector[j*coln]),n);
    DOFOR(k,n)vector[i*coln+k]-=alpha*vector[j*coln+k];
    }
  normv(&(vector[i*coln]),n);
```

```
  }
}
return(m);
}



/*
Driver for large eigenvalue solver

DEPENDENCIES:

  ftoc.h
  pv() in vector.c
  elarge() in elarge.c
*/

#include<ftoc.h>

/* #include "libc.h"
#include"math.h"
*/

main(argc,argv) int argc;char **argv;
{
static float a[9]={5.,-4.,-7.,-4.,2.,-4.,-7.,-4.,5.};
/* ex 2.24 of Bathe & Wilson,eigenvalues 6,-6,12 */
float value[3],vector[9],work[9],aa[9],worka[9];
int i,n,m,iter,pivot[3],workp[3];
/* must preserve a,as deflation modifies it */
DOFOR(1,9)aa[i]=a[i];
m=elarge(aa,value,vector,work,worka,pivot,workp,3,3,1);
printf("%d eigenvalues found\n",m);
DOFOR(1,3)
  {
  printf(" eigenvalue=%f\n",value[i]);
  }
DOFOR(1,3)
  {
  pv(&(vector[3*i]),3);
  }
exit(0);
}


deflating row 0
work=-1.000000,a=-7.000000,i=0 j=0 index=2
```

```
work=-1.000000,a=-4.000000,i=0 j=1 index=2
deflating row 1
work=0.000015,a=-7.000000,i=1 j=0 index=2
work=0.000015,a=-4.000000,i=1 j=1 index=2
after deflation
printing matrix columns 0 to 2
-2.000000e+000 -8.000000e+000 -7.000000e+000
-3.999893e+000 2.000061e+000 -4.000000e+000
-7.000000e+000 -4.000000e+000 5.000000e+000
deflating row 0
work=0.500007,a=-8.000000,i=0 j=0 index=1
after deflation
printing matrix columns 0 to 2
6.000120e+000 -3.999893e+000 -4.000000e+000
-8.000000e+000 -2.000000e+000 -7.000000e+000
-4.000000e+000 -7.000000e+000 5.000000e+000
eold 12.000000 enew -6.000089 j 2 ke 1
printing vector from 0 to 2
8.944246e-001 4.472189e-001 3.726776e-001
norm alpha by 0.707107,new alpha0.527046
printing vector from 0 to 2
5.217469e-001 4.472245e-001 3.726776e-001
printing vector from 0 to 1
-7.071068e-001 1.076857e-005
renorm v now
printing vector from 0 to 2
6.674170e-001 5.720882e-001 4.767281e-001
going to loner
eold -6.000089 enew 6.000120 j 1 ke 2
alpha=-0.666655
printing vector from 0 to 1
1.000000e+000 -6.666551e-001
norm alpha by 0.572088,new alpha-1.165301
printing vector from 0 to 1
2.222581e-001 -6.666551e-001
printing vector form 0 to 0
6.674170e-001
renorm v now
printing vector from 0 to 1
3.162787e-001 -9.48663e-001
eold 12.000000 enew 6.000120 j 2 ke 2
alpha=-0.895943
printing vector from 0 to 2
3.162787e-001 -9.486663e-001 -8.959429e-001
norm alpha by 0.707107,new alpha-1.267054
printing vector from 0 to 2
1.212222e+000 -9.486800e-001 -8.959429e-001
```

```
printing vector form 0 to 1
-7.071068e-001 1.076857e-005
renorm v now
printing vector from 0 to 2
6.806166e-001 -5.326480e-001 -5.030380e-001
pivot[0]=2
pivot[1]=0
pivot[2]=0
3 eigenvalues found
eigenvalue=12.000000
eigenvalue=-6.000089
eigenvalue=6.000120
printing vector from 0 to 2
-7.071068e-001 1.076857e-005 7.071068e-001
printing vector from 0 to 2
5.773489e-001 5.773618e-001 5.773401e-001
printing vector from 0 to 2
4.082502e-001 -8.164884e-001 4.082627e-001
```

```
/* vector processing routines
often simplified versions of BLAS routines for vectors
with contiguous storage
```

CONTENTS:

```
dot(a,b,n)
    returns double value of dot product of two vectors a,b
    of n elements
pv(v,n)
    prints vector of n elements
mv(a,x,y,m,n)
    y=ax where a is m x n matrix
normv(v,n)
    normalize a vector(scale its length to one)
sqnor(v,n)
    square fo the length of vector v of n elements
mvt(a,x,y,m,n)
    like mv except y=a^ x a^ =transpose of a
resid(r,a,x,n)
    r=ax-x,a matrix all other vectors
vs(v,s,n)
    scale vector by multiplying each element by s
vset(v,s,n)
```

```
    set vector v to scalar value s for each element
vcopy(x,y,n)
    y=x,vectors
vv(a,b,c,s,n)
    a=b+c*s,s scalar,a,b,c,vectors

DEPENDENCIES:
NONE
*/

#include "ftoc.h"
double dot(a,b,n) int n; float a[],b[];
{
int i;
double sum;
if(n<=0)return(0.);
sum=0.;
DOFOR(i,n)sum+=a[i]*b[i];
return(sum);
}

pv(v,n) int n;float v[];
{
int btm,top,i;
btm=0;
top=0;
while (btm<n)
    {
    top=min(btm+6,n);
    printf(" printing vector from %d to %d\n",btm,(top-1));
    for(i=btm;i<top;i++)printf("%e",v[i]);
    printf("\n");
    btm+=6;
    }
  return;
}

mv(a,x,y,m,coln) int coln,m;
float x[],y[],a[];
/* y=ax a(m,n) m rows n columns [row of length n] */
{
int i,j,k;
float sum;
DOFOR(i,m)
    {
    sum=0.;
    DOFOR(j,coln) sum+= x[j]*a INDEX(i,j);
```

```
        y[i]=sum;
        }
    return;
    }

    normv(v,n) int n; float v[];
    {
    double x,sqrt(),sqnor();
    x=sqnor(v,n);
    if(x!  =0.)x=1./sqrt(x);
    vs(v,x,n);
    return;
    }

    double sqnor(x,n) float x[];int n;
    {
    int i;
    double ans;
    ans=0.;
    if(n<=0)return(ans);
    DOFOR(i,n)ans+=x[i]*x[i];
    return(ans);
    }

    mvt(a,x,y,m,coln) int coln,m;
    float a[],x[],y[];
    {
    /* y=a^ x a(m,n) a m rows n columns [n elements/row]
    a^  n rows m columns */
    float sum;
    int i,j;
    DOFOR(i,coln)
        {
        sum=0.;
        DOFOR(j,m)sum+=x[j]*a INDEX(j,i);
        y[i]=sum;
        }
    return;
    }

    resid(a,x,y,r,n) int n;
    float a[],x[],y[],r[];
    {
    int i;
    mv(a,x,r,n,n);
    DOFOR(i,n)r[i]-=y[i];
    return;
```

```
}

vs(v,s,n) int n;float v[],s;
{
int i;
DOFOR(i,n)v[i] *=s;
return;
}

vset(x,s,n) int n;float s,x[];
{
int i;
DOFOR(i,n)x[i]=s;
return;
}

vcopy(x,y,n) int n;float x[],y[];
{
int i;
DOFOR(i,n)y[i]=x[i];
return;
}

vv(a,b,c,s,n) int n;float a[],b[],c[],s;
{
int i;
DOFOR(i,n) a[i]=b[i]+s*c[i];
return;
}

/* matrix & vector processing routines

CONTENTS:

vdif(a,b,dif,n)
     vector difference of two vectors a and b
swaprow(a,from,to,coln)
     swaq rows from and to of matrix a
swapcol(a,from,to,coln)
     swap columns coln= # of columns

The following two routines are similar to mv()and mvt()
in VECTOR.C Here,we process the first n columns of a
matrix of coln columns. Previously,it was assumed n=coln. Note
that there is one additional argument,the last.

mvc
```

```
        multiply vector by matrix
mvct
        multipy vector by transpose of matrix
mcopy(a,aa,coln,nrow,ncol)
        Copy the upper left-hand portion of one 2D matrix,a,to
        matrix aa.nrow rows and ncol columns are copied. It is
        assumed that both a and aa are dimensioned to have coln columns.

DEPENDENCIES:
NONE
*/

#include <ftoc.h>

vdif(a,b,dif,n) int n;float a[],b[],dif[];
{
int i;
DOFOR(i,n) dif[i]=a[i]-b[i];
return;
}

swaprow(a,from,to,coln) int from,to coln;float a[];
{
int i;
float x;
DOFOR(i,coln)
        {
        x=a INDEX(from,i);
        a INDEX(from,i)=a INDEX(to,i);
        a INDEX(to,i)=x;
        }
return;
}

swapcol(a,from,to,coln) int from,to ,coln;float a[];
{/*matrix assumed to be square=>number of rows=coln*/
int i;
float x;
DOFOR(i,coln)
        {
        x= a INDEX(i,from);
        a INDEX(i,from)=a INDEX(i,to);
        a INDEX(i,to)=x;
        }
return;
}
```

```
mvc(a,x,y,n,m,coln) int n,coln,m;
float x[],y[],a[];
/* y=ax a(m,n) */
{
int i,j,k;
float sum;
DOFOR(i,m)
    {
    sum=0.;
    DOFOR(j,n) sum+=x[j]*a INDEX(i,j);
    y[i]=sum;
    }
return;
}

mvtc(a,x,y,n,m,coln) int coln,n,m;
float a[],x[],y[];
{
/* y=a(T)x a(n,m) */
float sum;
int i,j;
DOFOR(i,m)
    {
    sum=0.;
    DOFOR(j,n)sum+=x[j]*a INDEX(j,i);
    y[i]=sum;
    }
return;
}

mcopy(a,aa,coln,nrow,ncol) float a[],aa[];
int coln,nrow,ncol;
/* copy uppeer left part of one matrix to another.
assumed matrices are float,and have same number of columns */
{
int i,j;
DOFOR(i,nrow)
    {
    DOFOR(j,ncol) aa INDEX(i,j)=a INDEX(i,j);
    }
return;
}
```

第五章　奇值分解：稳健(robust)最小二乘方估计、因子分析

§5.1　本章目的

本章将向矩阵的"奇值分解(SVD)"和利用 SVD 施行最小二乘方强拟合提供很有用的理论工具。[x1]

我们将在介绍 SVD 之后阐述此理论，在本章中集中讨论因子分解的情况下，将继续讨论上一章(关于特征值)有关 SVD 的统计应用。

§5.2　奇值分解

由于奇值分解应用的多面性，这里首次使用所谓"数值分析的瑞士军刀"是一种尝试。但这对 SVD 来说是不公平的。SVD 是处理许多问题的一种选择。"瑞士军刀"虽然方便，但早有解决该问题的最优工具。在本章，我们将讨论此方法并将其应用于数据的最小二乘方拟合以及因子分解(主分置，R—模式与 Q—模式)。SVD 应用在诸如相关问题、推广的最小二乘方多元方差与协方差分析(MANOVA)、以及约束最小二乘方等方面。由于受篇幅限制没能对其各种应用作充分的讨论。

§5.2.1　SVD

对任一个维数为 $m\times n$ 的矩阵 A，$A\hat{}\,A$ 是一方阵($n\times n$)，且是正定的。它的特征值(见第四章)一定是非负的。所以它们可以记成 S^2。这些数 S 称为"该矩阵 A 的奇值"。我们将看到无论何处出现 $A\hat{}\,A$。SVD 可能有其用武之地。我们已经讨论了一类涉及将矩阵 A 因子分解成具有有用性质的一些其它矩阵的乘积；第二章中的 LU 分解和第四章中的 QR 分解。在 SVD 中，我们分解矩阵 A 的因子，它通常未必是方阵，可以是 $m\times n$ 维。如：

$$USV\hat{}$$

这里 S 是一个维数为 $m\times n$ 的对角阵，而 U 与 V 是分别为 $m\times m$ 和 $n\times n$ 维的正交阵。S 的对角线元素是 A 的奇值，可以表示成 $A\hat{}\,A$，用 $A\hat{}=VSU\hat{}$ 及 $U\hat{}\,U=I$ 来证明。这些奇值要求为非负。如果有些为零，这些必须接着(有着比它们更高的指标)非零(正的)奇值。在算法中过去有一种倾向按下降值产生奇值。但这不一定可实现。因为我们将给出一个例子说明它不是这样的。一般来说，如果矩阵 A 是病态的(见第二章)则奇值很可能不按那样的顺序。

我们将给出一个程序，它允许去求满 U 阵，通常，只有 U 的主部(leading part)是有兴趣的。绝大多数可用的程序只提供 U 的这个部份作为输出。

在元素超过 $n\times n$ 子矩阵的 U 矩阵中，以及与零奇值相联系的元素中，SVD 具有相当

值得重视的非唯一性。因此，如果你的工具不适合于这里对那些分量给出的结果。不要过于担心。但是，如果你的奇值或有意义的结果（诸如最小二乘方检验法的那些结果）不适合，就该查找问题出在什么地方。

§5.2.2 数值方法

如果你读了前一章中的 QR 方法的简要介绍。SVD 方法没有什么东西应使你惊奇。你将回忆起在一普通矩阵 A 的 AR 因子分解中。首先把这个矩阵典型地化成 upper Hessenberg 形式（或者，如果是对称的，化成主对角线形），这可用一相似变换来完成，因它保持其特征值不变。典型地可用 Householder 变换来实现，它把在主对角线下方的一切元素都变成零，除了紧靠此主对角的下方的元素。在 EISPACK 程序中，ELMHES 与 TREDN（$N=1,2$ 或 3 取决于所希望的选项）完成这个变换。然后用 QR 或 QL 迭代（后者只是在用一较低阶的三角矩阵代替一上三角矩阵上有所不同，特别注意这两种迭代法的关系）。由于该矩阵右边的另一正交矩阵的灵活性，可以把该矩阵对角线上方化成零，留下一双对角线矩阵。在第四章已讨论过，没有可靠办法把一个矩阵在有穷的时间条件下化为对角矩阵。因为这等价于求 n 次方程的根。所以，用迭代法把留下的 S 矩阵对角线化。如果得到了充分的精确度，迭代就终止。通过首先化为双对角线型，这个方法的经济性就有很大的改进。其价值是（粗略估计）为 $2mn^2+4n^3$，对这里给出的工具而言。如果 $m>n$，还有某种程序更经济的方法存在（见下面）。

§5.3 正交相似变换

有两种基本类型的相似变换用于把该矩阵所希望的位置化为零。它们是反射和旋转变换。

反射法为 Householder 所建立，此法利用乘以一矩阵 $P=I-2W\hat{}W$，其中（如第四章所讨论的 Sherman—Morrison 公式中那样）第二项必须考虑一个并向量，其纯量积 $W\hat{}W=1$，如果乘向量 V 以此矩阵，结果得到的向量其分量按 W 向量方向的则反号。而垂直于 W 向量的分量则不变。仿佛向量 V 被反射在一面镜子里，W 是其单位正则向量一样，因此用了“反射”这个术语。我们可以逐次乘一矩阵以一些 P 矩阵把一个矩阵的列的适当部份化为零。于是这相似变换的正交矩阵就是这些 P 矩阵的乘积。选择适当的 W 向量来完成这个变换留给了 Wikinson 或 Stewart（参看前一章），但一般来说，要将这最下方的 $n-r$ 个元素化成零（n 阶矩阵的前 $r+1$ 列），记 $P=I-U\hat{}U/q$，则 U 的前 r 个元素是零（$u[i]=0, i=0,1,\cdots,r-1$），$u[r]=a[r-1][r]+S$，$u[i]=a[r-1][i]\ (i=r+1,n-1)$，

$$S=A\{\sum_{i=r}^{n}(a[r-1][i]^2)\}^{1/2},\qquad q=Su[r]$$

其中，A 选成 $+1$ 或 -1，为的是使 S 与 $a[r-1][i]$ 同号（因而可避免消去以及可能伴随的数值上的困难）。我们已经用了 C 附标惯例。所以一切附标以零为基准，$r+1st$ 列的列指数是 r，等等。

一个更可供选择的化元素为零的方法是旋转。在第四章我们遇见过基本的旋转矩阵，

它的一般形式是：

$$\begin{bmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & \cos t & & & & \sin t & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & -\sin t & & & & \cos t & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{bmatrix}$$

这种类型的正交变换与 Givens 的名字相联系.SVD 算法的后一半，即是，在双对角线化之后，利用这些变换化 S 矩阵为对角形。在化下对角元素为零里，引入附加的非零元素，但是这些必须化为零，这是由迭代法造成的。人们从这个矩阵的左上角开始，并沿对角线"追赶"这些元素再做几次迭代之后，通常此矩阵就被化为精确的对角形。

§5.4 最小二乘方问题

如果我们有 n 个未知数 n 个方程，一般我们可以求得唯一解。但是，如果试图用拟合经验(实验)数据决定 n 个变量，我们常常希望采取大大多于最少 n 次的观测。由于随机误差等，不可能期望得出一个解能满足 m 个观测点。于是我们要求有一"最好的拟合"，一个最方便的最好拟合是最小二乘方拟合。它把这些剩余量的平方和变为极小，即是说，这些被拟合的数据点与拟合的差为极小。

这就是"多重回归"问题。我们将看到标准的方差分析(ANOVA)问题和与此问题相似程度大大超过在 ANOVA 中用过的颇为不同的术语和方法。

§5.4.1 SVD 在最小二乘方问题中的应用

在最小二乘方问题中，我们要求把 $|Ax-b|$ 的值极小化。其中 b 是 m 个观测值的向量，x 是 n 个参数的待定的向量，A 是 $m\times n$ 阶的系数矩阵。这里考虑的只是线性的非约束的最小二乘方问题，既是说，不允许在此回归方程中强加上约束。例如，加上 $x[1]<0$ 或 $x[1]+x[2]=5$。我们还假定作为观测的模型是线性的，即是说，这些观测 $b[i]$ 被假定为要被模化为一些项的和，各项与参数 $x[i]$ 成比例。我们可以把 $x[i]$ 看作是这些项数。例如，我们假定把一份血浆的电阻模化为两项的和：一项由于与前载体与中性粒子的碰撞，另一项由于与带电荷粒子的碰撞。我们可以在不同的温度和密度条件下用模型作多次测量：

$$R(\text{temp},\text{density})=X[0]\text{Rneut}(\text{temp},\text{density})+x[1]\text{Rionic}(\text{temp},\text{density})$$

其中 Rneut 与 Rionic 的模型根据所研究的有关的温度及密度范围的物理知识，应该与温度和密度有关。即使这些模型十全十美，仍然要假定其全电阻可以表为这两项的和。

假定我们想把 $|Ax-b|$ 极小化。用 SVD，这就变成极小化 $|USV\hat{}\,x-b|$ 问题。定义 $z=V\hat{}\,x$ 并遍乘以 $U\hat{}$，定义 $d=U\hat{}\,b$。现在我们需要极小化 $|Sz-d|$。遍乘 $U\hat{}$ 是许可的，

因为它是一个正交矩阵，取$|U|=1$，于是用$U^{\wedge}U=1$。因为S是对角的，这个问题对Z求解现在是平凡的：$Z[i]=d[i]/S[i][i]$因为$S[i][i]$不为零。如果$S[i][i]$是零，$Z[i]$的值对此问题不重要，可以指定它为任何值。典型的是，在这种情况下置$Z[i]$为零，给出在此过程中极小模(向量长度)Z向量。一旦Z已知，$x=Vz$就是这(非约束的)线性最小二乘方问题的解。

注意，可以简单地将A作因子分解为QR而不用SVD。置$c=Q^{\wedge}b$，解$Rx=c$系统(视R为上三角形)，按第二章backsub()程序的后半部份，这很容易办到。这样，$|QRx-b|$就极小化了。为什么用SVD去多找麻烦，即在这种情况下为什么把R因子分解为SVD？因为SVD更多地告诉我们其解的特征。为零的(或非常小的)那些奇值告诉我们此问题的秩，如果这个问题其最小二乘方解不唯一，假如其最小二乘方拟合是唯一的(具有n个非零奇值的"满秩"问题。奇值的相对值告诉我们在决定此拟合时(见以下的因子分解讨论)向量Z的分量的相对重要性。如果这些奇值不是单调减小的(在第二个测验题中将遇到)，对可能的病态阵将作处理。

一矩阵A的最小二乘问题的解也可写成$x=(A^{\wedge}A)^{-1}A^{\wedge}b$。右端矩阵的乘积叫做矩阵$A$的"广义逆"，或"Moore—Penrose"。有时记为A^{+}。以下将证明，如果完全可能的话，应该避免真正去作这个乘法和求逆。SVD就提供了替代办法：

$$A^{+}=Vs^{+}U^{\wedge}$$

其中S^{+}定义为一对角阵其元素是S的元素的倒数，即这些奇值的倒数，如果奇值是零，则S^{+}的对应元素也为零。

注意，QR因子分解可以像过去的LU因子分解一样用来解线性系统。如果$QRy=b$，得出$d=Q^{\wedge}b$，于是解$Ry=d$。这比LU因子分解作更多的运算，但数值上稳定，不用绕轴转动。因此，对平行处理结构它使人感兴趣，转动的策略对此可能在处理器之间导致通讯的可怕的失误。

§5.5 因子分析

这一节将讨论主分量因子分析，并触及R—模式与Q—模式因子分析。其它类型的因子分析——"旋转"诸如Varimax，Equimax，Quartimax，Quartimin，Promax，Covarmin等等，本书将不作讨论，旋转方法说起来是带有主观性的，这可以从大量的可供选择的旋转去选取。给定任一旋转，通过一个正交变换就可以从这个旋转得到另一旋转，甚至由这样的分析得出的因子数目也是可变的。

在较早的一些书中，主向量方法是作为特征值问题来讨论。的确它可以这样来处理，然而SVD是处理这个问题更稳健的方法。

相关系数矩阵实际上具有形式$X^{\wedge}X$(见下面)，这样的矩阵乘积一定是对称、正定的。这样的矩阵乘积的条件数是矩阵X的条件数的平方。从而如果X的条件数大，那$X^{\wedge}X$的就更大，这很容易导致数值上的困难。

作为一个专为数字通用计算机的例子，考虑矩阵A为：

$$\begin{pmatrix} 1 & 1 & 1 \\ e & 0 & 1 \\ 0 & e & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

其中$|e|<1$，其乘积 $A\hat{}A$ 为：

$$\begin{pmatrix} 1+e^2 & 1 & 1+e \\ 1 & 1+e^2 & 1+e \\ 1+e & 1+e & 4 \end{pmatrix}$$

如果$|e|$小于机器的精度，则矩阵 A 的前两列和第三行是奇的。如果 e^2 小于机器的精确度则乘积矩阵将是奇的(要么前两行，要么前两列将是恒等的)。如果$|e|$小，它的平方就更小。于是，通过表 $A\hat{}A$ 的方法，变一个几乎奇矩阵为一奇阵是容易的。这就是为什么我们不应这么作的原因，要是有一个替代的办法的话。(且不说它要求阶为 n 次方的浮点运算)。甚至更病态的例子也可由对称矩阵构造出来。矩阵 A：

$$\begin{pmatrix} 1 & L \\ 0 & 1 \end{pmatrix}$$

有二重退化特征值。此外，它只有一个以方向(1,0)为方向的特征向量。此矩阵的谱半径或条件数，即特征值的值域是 1。矩阵：

$$A\hat{}A = \begin{pmatrix} 1 & L \\ L & 1+L^2 \end{pmatrix}$$

有特征值 $2+L^2$ 与 $1/(2+L^2)$。如果 L 大时，导致阶为 L^4 的谱半径或条件数。当这是一种特别的病态情形时，在一切可能条件下人们不应试图构造矩阵 $A\hat{}A$。

双共轭梯度算法试图把共轭梯度方法应用于对称阵 A，在避免构造 $A\hat{}A$ 的处理过程时代之以构造对称阵：

$$\begin{pmatrix} 0 & A \\ A & 0 \end{pmatrix}$$

然而，这种努力并不总令人满意。

假定相关矩阵 R 是已知的，其对角线上的元素都是 1，由于任一变元都很好地与自己相关。其它的元素表示一个变元与另一个变元相关性。此矩阵显然是对称的，因为变元 i

和 j 的相关性必须与变元 j 和 i 的相关性一致。在主分量分析中，我们试图决定在这些变元的线性组合中它们是最小相关的，即正交的(无相关性)。这是特征值问题。因为特征向量(如果它们对应于不同的特征值)都是正交的。但可以用 SVD 更稳健的功能来处理，因为用不着构成 $A\hat{}A$ 型的相关性。

取代处理可能的病态相关矩阵 R，我们去处理此数据矩阵。这是一个 $m\times n$ 阶的矩阵 X。其中，n 是作为最小二乘问题中的变元的个数而 m 是观测次数。出于数值上的考虑希望构造"零平均"(Zero－mean)数据矩阵，它形如 $Y=X-Ux\hat{}$，这里 $x\hat{}$ 是变元的平均值，U 是向量$(1,1,\cdots,1)$，而向量的乘积经证明是一并向量(dyad)。此外，对每个变元 i 给出了其标准离差 $s[i]$，我们构造一对角矩阵 D 就象用奇值构造 S^{+} 那样。则 $Z=YD$ 就是"标准化"了的数据矩阵，而矩阵 $Y\hat{}Y/(n-1)$ 就是协方差矩阵，而 $Z\hat{}Z/(n-1)$ 就是相关矩阵，不去构造此相关矩阵，而在零平均矩阵上施行 SVD。于是，此矩阵 V 含有正交奇向量。关于第 i 列的方差是 $s[i]^2/(n-1)$，因为 $s[0]$ 应是最大的奇值，V 的第一列最通常是主分量——但如前面所注明的，这并非始终是真的，尤其是如果此矩阵是病态的。因此一定要检查哪个奇值是最大并用相关的列作为第一主分量。其余的分量，如果要求的话，应按下降的奇值顺序。根据矩阵 US，变换了的数据矩阵是已知的。它的列应该有尽可能大的方差，第一列的方差最大而随后的方差不增。

表 5－1　由数据矩阵构造的矩阵

零——平均数据矩阵	Y	$X-U\hat{}X$
标准数据矩阵	Z	YD
协方差矩阵	C	$Y\hat{}Y/(n-1)$
相关矩阵	R	$Z\hat{}Z/(n-1)$

表中：n＝变元个数；$U=(1,1,1,\cdots)$；x＝观测数据矩阵的变元的平均值向量(每行的平均值)；D＝每个变元的标准离差的对角矩阵。

表 5－1 所列的各矩阵已经在应用。这完成了主分量分析。对 R－模式与 Q－模式分析，构成虚构的相关矩阵(或数据矩阵)。在 R－模式中，置自方差等于 1，而在 Q－模式中，变元与观测值交换。在这些方法中，每个主分量分析应用于这些矩阵。典型的是，只有几个特征值有兴趣，因此，人们有用特征值分析而不用 SVD 的倾向。

§5.6　程序和测试情况

此程序基于 Forsythe，Malcolm 和 Moler 的 FORTRAN 代码，它们又是 EISPACK 2 包里的程序的一个改进。这软件包是基于 Wilkinson 与 Reinsch 里的 ALGOL 程序的。为计算满 U 矩阵设计了代码段(Section)。由 Forsythe，Malcolm 与 Moler 引入的改变算法为防止下溢在 Desmet 环境下特别重要。此外，他们引入过的对大机器依赖性的改变保留在这里。我们已把 SVD 程序写成一个子程序时，剖分成两个程序，因为它对 Desmet 编译器来说是太大了。

我们用两个试验问题。在第一个里，决定 Forsythe，Malcolm 和 Moler 给 出的矩阵的 SVD。这个矩阵是退化的，即有一个零奇值。我们作两次分解，一次只要求 U 矩阵的 $m \times n$ 部份。再一次求满 $m \times m$ U 矩阵。请注意通常的 U 矩阵不是唯一的。的确，由于这种情况下的零奇值，只有对应于零奇值的矩阵的 3×2 部份是唯一的。我们的结果与 Forsythe，Malcolm 和 Moler 对此满矩阵的不一致，虽然这些结果对 U 的 3×2 子矩阵以及对满 U 和 S 矩阵适合。按 SVD 的 FORTRAN 方法，我们已经写成在带协处理器 8087 的 IBM PC 机上运行的程序，对此 U 矩阵我们仍然得到其它的值。

在同样的程序列表中，程序 main()得到的是最小二乘方的解。这个问题涉及一组数据的二次多项式的最小二乘方问题(见对此数据的主程序)。

§5.7 程序列表与测试问题输出

```
/*perform robust least-squares regession using SVD
note- uses double-precision storage- DO NOT mix with routines
from vector.c or matrix.c, as these assume vectors to be of
type float.

CONTENTS:
lstsq() perform lst-sq. solution Az=y
     A dimensioned for coln columns
     neq equations (rows)
     col columns in use (variables)
     other variables working storage

type double routines similar to those in vector.c and matrix.c:
pvd()
setrow()
vdiffd()
mvd()
mvtd()

DEPENDENCIES:
svd() in svd.c required
*/
/*
#include "libc.h"
#include "math.h"
*/
#define INDEX(i,j) [j+(i)*coln]
#define DOFOR(j,to) for(i=0;i<to;i++)
#define min(a,b) ((a)<(b)? (a):(b))

lstsq(a,y,coln,neq,col,u,s,v,z,resid,work)
```

```
int neq ,col ,coln ;
double a[],y[];
double u[],v[],work[],s[],z[],resid[];
{
int matu ,matv,ierr ,i;
matu=1;
matv=1;

svd(coln ,neq ,col ,a ,s ,matu ,u ,matv ,v,&ierr ,work);
printf(" ierr=%d\n singular values=\n",ierr);
pvd(s,col);
printmd(u,coln ,neq ,col ,neq);
printmd(v,coln ,neq ,col ,col);

/* work d=U[t] * y */
mvtd(u,y,work ,col ,neq ,coln);

DOFOR(i,col)
    {
    if( s[i]>0. ) work[i]/=s[i];
    }
mvd(v,work ,z,col ,col ,coln);
printf(" answer is:\n");
pvd(z,col);
printf(" residual errors for each eqn:\n");
mvd(a ,z,work ,col ,neq ,coln);
vdifd(work ,y,work ,neq);
pvd(work ,neq);return;
}

setrow(a ,coln ,row ,x) int row,coln; double x,a[];
{
a INDEX(row,0)=1.;
a INDEX(row,1)=x;
a INDEX(row,2)=x * x;
return;
}

main(argc ,argv) int argc; char * *argv;
{
/* test pblm 1st sq. */
static double y[10]={1.1,1.3,2.0,2.1,2.7,2.8,3.4,3.6,4.1,4.0};
int i,j,k,coln ,rown ,ierr;
double x,a[30],z[3],s[10],resid[10],work[30],u[30],v[30];

static double aa[5][3]={1. ,6. ,11. ,2. ,7. ,
  12. ,3. ,8. ,13. ,4. ,9. ,14. ,5. ,10. ,15. };
```

```
svd(3,5,3,aa,s,1,u,1,v,&ierr,resid);
printf(" ierr=%d\n" ,ierr);
DOFOR(ierr,3)printf(" sing value=%f\n",s[ierr]);
printf("u matrix:");
printmd(u,3,4,3,5);
printf("v matrix:");
printmd(v,3,3,3,3);
svd(3,5,3,aa,s,-1,u,1,v,&ierr,resid);
printf(" ierr=%d\n" ,ierr);
DOFOR(ierr,3)printf(" sing value=%f:\n",s[ierr]);
rintf("u matrix:");
printmd(u,5,5,5,5);
printf("v matrix:");
printme(v,3,3,3,3);

coln=3;
rown=10;
DOFOR(i,5)
  {
  x=(i-2);
  DOFOR(j,2)
    {
    k=j+2*i;
    setrow(a,coln,k,x);
    }
  }
printf(" coef.matrix is :\n");
printmd(a,coln,rown,coln,rown);
printf(" rhs is\n");
pvd(y,rown);
lstsq(a,y,coln,rown,coln,u,s,v,z,resid,work);
exit(0);
}
pvd(v,n) int n;double v[];
{
int btm,top,i;
btm=0;
top=0;
while (btm<n)
  {
  top=min(btm+6,n);
  prinif(" printing vector from %d to %d\n", btm,(top-1));
  for(i=btm;i<top;i++)printf(" %e",v[i]);
  printf("\n");
  btm+=6;
  }
return;
```

```
}

mvd(a,x,y,n,m,coln) int n,coln,m;
double x[],y[],a[];
/* y=ax a(m,n) */
{
int i,j;
double sum;
DOFOR(i,m)
  {
  sum=0.;
  DOFOR(j,n) sum+= x[j]*a INDEX(i,j);
  y[i]=sum;
  }
return;
}
mvtd(a,x,y,n,m,coln) int n,coln,m;
double x[],y[],a[];
/* y=a[t]x a(m,n) */
{
int i,j;
double sum;
DOFOR(i,n)
  {
  sum=0.;
  DOFOR(j,m) sum+= x[j]*a INDEX(j,i);
  y[i]=sum;
  }
return;
}

vdifd(x,y,z,n) int n;double x[],y[],z[];
/*z=x-y*/
{
int i;
DOFOR(i,n)z[i]=x[i]-y[i];
return;
}
/*
singular Value Decomposition (SVD) of a matrix

CONTENTS:
svd()   calculates SVD
diag()  second half of SVD calculation
dsign() type double function. equivalent to FORTRAN sign
  transfer function
printmd() print double precision matrix  (as in qrm package)
```

```
DEPENDENCIES;
  NONE
*/
/*
#include "libc.h"
#include "math.h"
*/
/* defines below are from ftoc.h except for IU,DFFOR */
#define INDEX(i,j) [j+(i)*coln]
#define IU(i,j) [j+(i)*colu]
#define DOFOR(i,j) for(i=0;i<j;i++)
#define DFFOR(i,from,to) for(i=from;i<to;i++)
#define min(a,b) (((a)<(b))? (a): (b))
#define max(a,b) (((a)<(b))? (b):(a))
#define abs(x) ( ((x)>0.)? (x):-(x))

#define itmax 30

printmd(a,coln,rown,col,row) int rown,row,col,coln;double a[];
{
int i,j,btm,top,count,ncol=4;/* 4 col max */
printf("\n");
btm=top=0;
while(btm<col)
  {
  top=min(col,(btm+ncol));
  printf(" printing matrix columns %d to %d\n",btm,(top-1));
  DOFOR(j,row)
    {
    for(i=btm;i<top;i++)
      {
      printf(" %e",a INDEX(j,i));
      }
    printf("\n");
    }
  btm+=ncol;
  }
return;
}
svd(coln,m,n,a,w,matu,u,matv,v,ierr,rv1)
int coln,m,n,matu,matv,*ierr;
double a[],w[],u[],v[],rv1[];
/*
  all matrices a,u,v have coln>=n columns

  a(m,n) v(n,n) u(m,n or m)
```

```
typically,m>n.

matu = 0 do not compute u
  1 compute u(m,n)
  -1 compute u(m,m)
NB: U MUST BE DIMENSIONED (M,N) IF MATU=1 AND(M,M) IF MATU
matv = 0 do not compute v
  1 compute v
ierr=error return integer
rv1 working storage (see text)
*/
{
int i,j,k,mn,il,I,I1,its,m1,n1,k1,top,colu;
double sqrt(),dsign(),dum,c,f,g,h,s,x,y,z,eqs,scale,anorm;
*ierr=0;
colu=n;
if (matu<0)colu=m;
/*printmd(a,coln,m,n,m);*/

m1=m-1;
n1=n-1;
DOFOR(i,m)
  {
  DOFOR(j,n) u IU(i,j)=a INDEX(i,j);
  }
/* reduction to bidiagonal form via Householder xfms */
g=0.;
scale=0.;
anorm=0.;
DOFOR(I,n)
  {
  I=i+1;
  rv1[i]=g*scale;
  g=0.;
  s=0.;
  scale=0.;
  if(i<=m1)/* in fortran, <= but c I runs 0-m1 */
    {
    DFFOR(k,i,m)scale+= abs(u IU(k,i));
    if(scale! =0.)
      {
      DFFOR(k,i,m)
        {
        dum=u IU(k,i)/scale;
        s+=dum*dum;
        u IU(k,i)=dum;
```

```
        }
      f= u IU(i,l);
      g= -dsign(sqrt(s),f);
      h=1./(f*g-s);/* 1/h of others- * faster than /*/
      u IU(i,l)=f-g;
      if (l! =nl)
        {
        DFFOR(j,l,n)
        {
        s=0.;
        DFFOR(k,i,m)s+=
        u IU(k,l)*u IU(k,j);
        f=s*h;
        DFFOR(k,l,m)
          u IU(k,j)+=f*u IU(k,l);
        }
       }/*i! =n*/
     DFFOR(k,i,m)u IU(k,l)*=scale;
     }/*scale! =0*/
   }/*i<=m*/
w[i]=scale*g;
g=0.;
s=0.;
scale=0.;
if(i<=m1 && i! =n1)
  {
  DFFOR(k,l,n)scale+= abs(u IU(i,k));
  if(scale! =0.)
    {
    DFFOR(k,l,n)
      {
      dum= u IU(i,k)/scale;
      s+=dum*dum;
      u IU(i,k)=dum;
      }
    f=u IU(i,l);
    g=-dsign(sqrt(s),f);
    h=1./(f*g-s);
    u IU(i,l)=f-g;
    DFFOR(k,l,n)rv1[k]=u IU(i,k)*h;
    if(j! =m1)
      {
      DFFOR(j,l,m)
        {
        s=0.;
        DFFOR(k,l,n)
          s+=u IU(j,k)*u IU(i,k);
```

```
        DFFOR(k,1,n)u IU(j,K)+=s*rv1[k];
        }
      }
    DFFOR(k,1,n)u IU(1,k)*=scale;
    }
  }
y=abs(w[1]=abs(rv1[1]);
anorm=max(anorm,y);
  }
/*accum.rhs*/
/*printf(" acc rhs\n");*/
if(matv)
  {
  for(i=n-1;i>=0;i--)
  {
  if(i!=n1)
    {
    if (g!=0.)
      {/*here FMM do division by g,u indiv*/
      /*h=1./(g*u IU(1,1);*/)
      DFFOR(j,1,n)v INDEX(j,i)
        =(u IU(i,j)/u IU(i,1))/g;
      DFFOR(j,1,n)
        {s=0.;
        DFFOR(k,1,n)s+=
          u IU(i,k)*v INDEX(k,j);
        DEFOR(k,1,n)v INDEX(k,j)+=
          s* v INDEX(k,i);
        }
      }
    DFFOR(j,1,n)
      {
      v INDEX(i,j)=0.;
      v INDEX(j,i)=0.;
      }
      }
    v INDEX(i,i)=1.;
    g=rv1[i];
    l=i;
    }
  }
/*accum.lhs*/
/*printf(" accum lhs\n");*/
if(matu)
  {
  if(matu==-1)
    {
```

```
    DFFOR(i,n,m)
      {
      DFFOR(j,n,m) u IU(i,j)=0.;
      u IU(1,i)=1.;
      }
    }
  mn=n;
  if(m<n) mn=m;
  top=n;
  if(matu==-1)top=m;
  for(l=mn-1;l>=0;l--)
    {
    l=l+1;
    g=w[i];
    if (l! =n1)
      {
      DFFOR(j,l,top)u IU(l,j)=0.;
      }
    if(g! =0.)
      {
      /*h=1./(g* u IU(i,i);*/
      if(l! =(mn-1))
      {
      DFFOR(j,l,top)
        {
        s=0.;
      DFFOR(k,l,m)
        s+= u IU(k,i) * u IU(k,j);
      f=(s/u IU(l,i))/g;
      DFFOR(k,i,m)u IU(k,j)+=f*u IU(k,i);
      }/* DFFOR j */
    }/* l! =mn-1 */
      DFFOR(j,i,m)u IU(j,i)/=g;

      }
    else/*g=0*/
      {
      DFFOR(j,i,m)u IU(j,i)=0.;
      }
    u IU(i,l)+=1;
    }
  }
diagon(u,v,rvl,w,anorm,coln,colu,n,m,ierr);
return;
}
diagon(u,v,rvl,w,anorm,coln,colu,n,m,ierr);
double anorm,u[],v[],rvl[],w[];
```

```
int coln,colu,n,m, *ierr;
{
int i,j,k,l,ll,nml,kl,il,matu,matv,its;
double c,s,h,f,g,x,z,y;
matu=1;matv=1;
/*
printf(" diagonalizing,anorm=%f\n",anorm);
pvd(w,n);pvd(rvl,n);
printmd(u,coln,m,n,m);
printmd(v,coln,n,n,n);
 */
/* diagonalize binary form */
for(k=n--1;k>=0;1--)
  {
  kl=k-1;
  its=0;
testsplit:
  for(l=k;1>=0;l--)
    {
    ll=l-1;
  /* if( abs(rv1[1])<=eps)goto conv;
    if(abs(w[ll]<=eps)break;
   */
    if ( abs(rv1[l])+anorm==anorm)goto conv;
    if(abs(w[ll])+anorm==anorm) break;
    }
  c=0.;
  s=1.;
  DFFOR(1,i,k+1) /* want i to range from 1 to k inclusive */
    {
    f=s*rv1[i];
    rv1[i]*=c;
/* if(abs(f)>eps)
*/
  if(abs(f)+anorm!=anorm)
      {
      g=w[i];
      h=sqrt(f*f+g*g);
      w[i]=h;
      c=g/h;
      s=-f/h;
      if(matu)
        {
        DOFOR(j,m)
          {
          y=u IU(j,ll);
          z=u IU(j,i);
```

```
            u IU(j,i1)=y*c+z*s;
            u IU(j,i)=-y*s+z*c;
            }
          }/*matu*/
        }/* if */
      }/*DFFOR i,l,k*/
      conv;;
      z=w[k];
    If(l!  =k)
      {
      if (its>=itmax)
        {
        *ierr=k;
        return;
        }
      its++;
      x=w[l];
      y=w[k1];
      g=rv1[k1];
      h=rv1[k];
      f=((y-z)*(y+z)+(g-h)*(g+h))/(2.*h*y);
      g=sqrt(f*f+1.);
      f=((x-z)*(x+z)+h*(y/(f+dsign(g,f))-h))/x;
      /* do another QR xfm */
      c=1.;
      s=1.;
      DFFOR(il,l,k1+1)/* il ranges l to k1 inclusive */
        {
        i=il+1;
        g=rv1[i];
        y=w[i];
        h=s*g;
        g=c*g;
        z=sqrt(f*f+h*h);
        rv1[il]=z;
        c=f/z;
        s=h/z;
        f=x*c+g*s;
        g=-x*c+g*c;
        h=y*s;
        y=y*c;
        if(matv)
          {
          DOFOR(j,n)
            {
            x= v INDEX(j,il);
            z= v INDEX(j,i);
```

```
            v INDEX(j,i1)=
              x*c+z*s;
            v INDEX(j,i)=
              -x*s+z*c;
            }
          }
        z=sqrt(h*h+f*f);
        w[i1]=z;
        if(z!  =0.)
          {
          c=f/z;
          s=h/z;
          }
        f=c*g+s*y;
        x=-s*g+c*y;
        if(matu)
          {
          DOFOR(j,m)
            {
            y= u IU(j,i1);
            z= u IU(j,i);
            u IU(j,i1)=
              y*c+z*s;
            u IU(j,i)=
              -y*s+z*c;
            }
          }
      }
      rv1[1]=0;
      rv1[k]=f;
      w[k]=x;
      goto testsplit;
      }/*11  =k*/
/*convergence*/
if(z<0.)
  {/*  make w[k] non-neg*/
  w[k]=-z;
    if(matv)
    {
    DOFOR(j,n) v INDEX(j,k)=-v INDEX(j,k);
    }
    }
  }
return;
}

double dsign(to,from) double to,from;
```

```
{
double x;
x=0.;
if(from>0.)x=1.;
if(from<0.)x=-1;
return( x*abs(to));
}
ierr=0
sing value=35.127223
sing value=2.465397
sing value=0.000000
u matrix:
printing matrix columns 0 to 2
-3.545570E-1 -6.886866E-1 6.055520E-1
-3.986963E-1 -3.755545E-1 -5.059518E-1
-4.428356E-1 -6.242241E-2 -3.441072E-1
-4.869750E-1 2.507097E-1 -2.161381E-1
-5.311143E-1 5.638418E-1 4.606451E-1
v matrix:
printing matrix columns 0 to 2
-2.016649E-1 8.903171E-1 -4.082482E-1
-5.168305E-1 2.573316E-1 8.164965E-1
-8.319960E-1 -3.756538E-1 -4.082482E-1
ierr=0
sing value=35.127223;
sing value=2.465397;
sing value=0.000000;
u matrix:
printing matrix columns 0 to 3
-3.545570E-1 -6.886866E-1 6.055520E-1 -5.757403E-1
-3.986963E-1 -3.755545E-1 -5.059518E-1 3.668383E-1
-4.428356E-1 -6.242241E-2 -3.441072E-1 3.270464E-1
-4.869750E-1 2.507097E-1 -2.161381E-1 5.483533E-1
-5.311143E-1 5.638418E-1 4.606451E-1 -6.664978E-1
printing matrix columns 4 to 4
-5.286540E-1
7.463180E-1
2.175492E-1
-5.594363E-1
1.242231E-1
v matrix:
printing matrix columns 0 to 2
-2.016649E-1 8.903171E-1 -4.082482E-1
-5.168305E-1 2.573316E-1 8.164965E-1
-8.319960E-1 -3.756538E-1 -4.082482E-1
coef. matrix is:
printing matrix columns 0 to 2
```

```
1.000000E0 -2.000000E0 4.000000E0
1.000000E0 -2.000000E0 4.000000E0
1.000000E0 -1.000000E0 1.000000E0
1.000000E0 -1.000000E0 1.000000E0
1.000000E0 0.000000E0 0.000000E0
1.000000E0 0.000000E0 0.000000E0
1.000000E0 1.000000E0 1.000000E0
1.000000E0 1.000000E0 1.000000E0
1.000000E0 2.000000E0 4.000000E0
1.000000E0 2.000000E0 4.000000E0
rhs is
printing vector from 0 to 3
1.100000E0 1.300000E0 2.000000E0 2.100000E0
printing vector from 4 to 7
2.700000E0 2.800000E0 3.400000E0 3.600000E0
printing vector from 8 to 9
4.100000E0 4.000000E0
ierr=0
singular values=
printing vector from 0 to 2
8.615557E0 1.942207E0 4.472136E0

printing matrix columns 0 to 2
-4.777907E-1 1.207191E-1 4.472136E-1
-4.777907E-1 1.207191E-1 4.472136E-1
-1.453291E-1 -3.385169E-1 2.236068E-1
-1.453291E-1 -3.385169E-1 2.236068E-1
-3.450857E-2 -4.915956E-1 1.495917E-16
-3.450857E-2 -4.915956E-1 1.495917E-16
-1.453291E-1 -3.385169E-1 -2.236068E-1
-1.453291E-1 -3.385169E-1 -2.236068E-1
-4.777907E-1 1.207191E-1 -4.472136E-1
-4.777907E-1 1.207191E-1 -4.472136E-1

printing matrix columns 0 to 2
-2.973106E-1 -9.547808E-1 8.657743E-17
4.814340E-17 -1.056692E-16 -1.000000E0
-9.547808E-1 2.973106E-1 -9.321233E-17
answer is:
printing vector from 0 to 2
2.788571E0 7.150000E-1 -3.928571E-2
residual errors for each eqn:
printing vector from 0 to 3
1.014285E-1 -9.857142E-2 3.428571E-2 -6.571428E-2
printing vector from 4 to 7
8.857142E-2 -1.142857E-2 6.428571E-2 -1.357142E-1
printing vector from 8 to 9
-3.857142E-2 6.142857E-2
```

第六章　牛顿——辛普森(Newton—Raphson)及其有关方法

§6.1　本章目的

在本章里将求得C工具，用改进的牛顿——辛普森(Newton—Raphson)(NR)迭代解非线性方程系统。此项改进的意图是增强这个方法的功能，即在初始猜测不好的情况下求它的解。还用此工具处理多种电离平衡，这是一个模拟化学平衡的问题。

这里建立的牛顿——辛普森迭代法在第十三章里还会遇见，那里它是作为许多已知函数的逆的程序的基础。

§6.2　非线性系统求解的方法

在本章及下章，我们考虑非线性方程的解。这里讨论最常用的解此类问题的牛顿——辛普森方法。所有解非线性问题的方法(除特殊情形，如次数低于5次的多项式方程)都要求从初始猜测开始作迭代，一般，对任意的初始猜测，不能保证迭代收敛到一个解。的确，最原始的牛顿——辛普森方法处理这类问题有时还较困难。然而，作一处或两处改进以后，这方法得以大大增强，我们将讨论这样的决窍。

在第十三章里，我们应用牛顿——辛普森方法求一已知函数的逆，即一个方程的系统，$y-f(x)=0$，其中y已知，求解x，这被用来决定其误差函数的逆。设有了通用程序invr()，它用来给出Student´的t分布函数和其它的分布函数。

§6.3　牛顿——辛普森方法

我们首先讨论求满足方程$f(x)=0$简单的一个变量的问题，假定$f(x)$是实的，我们从一个初始"猜测"开始作为通向适合的x。比如说$x[0]$，如果$f(x)$是一个简单的线性函数，即$f(x)=ax+b$，我们能解合适的根为$x=-b/a$，让我们局部地(在对x的猜测的邻域)逼近此函数f。此函数f将通过点$(x[0],f(x[0])$的直线逼近，并在此点直线与函数相切，即有斜率$f'(x[0])$，我们希望给$x[0]$一个调整量，即得到$x[1]=x[0]+dx$，使得在$x[1]$处切线与x轴相交。因为函数f在这里并不一定等于零，除非它是一个线性函数，否则我们不得不迭代以求得希望的精度。因此，我们更一般地记成$x[m+1]=x[m]+dx$。据此求得$dx=-f(x[n])/f'(x[n])$，于是如能估计f及其离差，就能施行基本的牛顿——辛普森迭代。通常离差的估计式为：

$$f'(x[n])=\{f(x[n]+ds)-f(x[n]-ds)\}/2ds$$

(用非中心差是更省事，但精确度对某些程序要小些，为的是此函数的一个估值是

$f(x[n])$，于是可以再次使用这个值）。此法十分类似于对此问题用“反余弦方法”以及“假位置规则”或试位法，这样的迭代一直进行直到：

1）$f(x[n])$的剩余值在零的所希望的容许范围内；

2）x 的改变量 dx，要么是如此之小以致其根求到希望的精度，要么是这个运算的精度极限；

3）出了毛病，dx 振动得厉害或表现出其它反常性质。例如，这可能意味 x 代表一些物理变量，比方说是密度或温度，它应该是正的但它却沿着切线被外推向负值。以下将讨论在这些情形下做些什么？

在很多有重要意义的情况下，并不产生这样的问题。牛顿——辛普森迭代法能平方收敛到要求的根，这就是说如果 x 中的误差小，在每次迭代中这个误差就按平方减小，0.1 的误差变成 0.01，接着就是 0.0001，然后 0.00000001，等等。例如，假如我们要求一个数的平方根，解方程 $x^2-A=0$ 得 A 的平方根。于是我们把 $f(x)$ 放在这个方程的左端，然后有 $f'=2x$，从而 $x[n+1]=(x[n]+A/x[n])/2$ 就是求平方根的牛顿——辛普森迭代，这个方法运行很好时常在库平方根程序中用到。

计算机用计算近似商作浮点长除法，然后用牛顿——辛普森迭代去提炼答案的精度。假定我们求 A 的倒数，$x=1/A$，此迭代将是 $x[n+1]=x[n](2-Ax[n])$，作为例子，取 $A=3$，$x[0]=.5$ 于是 $x[1]=.25$，$x[2]=.3125$，$x[3]=.332\cdots$，等等。

§6.3.1 多维形式

上述牛顿——辛普森迭代很现成地就推广到处理多变量问题。单变量问题可以称之为“一维搜索”，因为我们沿 x 轴作一次又一次地迭代。从而术语“多维搜索”是在多变量情况下，对多个变量作多个方向的搜索。贯穿整个讨论，把 f 作为函数值 f 的向量（对 n 个变量的 n 个值），dx 是对 x 的修正向量。运算除以 f' 是代之以一导数矩阵的逆乘以此 f 向量，称之为 Jacobian 矩阵，$J[i][j]=df[i]/dx[j]$（这里 $x[j]$ 指的是第 j 个变量而不是第 j 次迭代），即我们解线方程系统 $Jdx=-f$。按第三章已讨论的，比起作出矩阵 J，取其逆，再作此向量的矩阵乘法要省事得多。

§6.4 陷阱及补救措施

因为牛顿——辛普森迭代如果初始猜测离要求的零点太远就会出错，即使函数不是明显反常的。明显的反常函数在初始猜测与零点之间有一极小值或者说一个谷。如果引此谷在 x 轴上方，我们就会掉进去永远不能接近 x 轴。

更易出错的情形是在拐点，即关于函数使 $f''(x)=0$ 的情形。如考虑 $f(x)=\arctg(x)$ 它有一个零点即在 $x=0$ 为拐点，如果初始猜测充分接近零，不会有问题，很快就求出根 $x=0$。然而，如果初始猜测太大，图 6－1 说明会发生什么情况。从标号为 1 的点出发，沿着点 $x[2]$的切线，但因点 1 与点 2 之间有一拐点，又函数 f 在点 2 的斜率相对地小，我们得到点 3。对于函数 $\arctg(x)$，$x[n+1]=x[n]-\arctg(x[n])(1+x^2)$，我们可以证明，如果初始猜测长度大于 1.4，就会产生发散。另一个出现的问题是多重零点，例如 $f(x)=x^2$，这里

$f(x)$的切线当我们接近要求的根时趋于水平。多亏有限精确的算法，我们容易发现由于小的除数$f'(x)$引起的大dx。

对于介值极小这一反常情形，我们毫无办法。但在多重零点或拐点，有一相对简单应急的处置法。注意到修正量dx的符号是对的但其量太大。这引起从n步到$n+1$步$|f(x)|$的误差增加而不是减小，应急的办法是存储值$f(x[n])$，如果$f(x[n+1])$的长度更大，则拒绝此迭代而进入一个循环，其中dx逐次减半直到得到更小的$f(x[n]+dx)$为止。这个新的x值将作为$x[n+1]$被接受。如果从$f(x[n])$起，数值不能分辨，我们将终止此程序，因为牛顿——辛普森方法已经竭尽所能了。

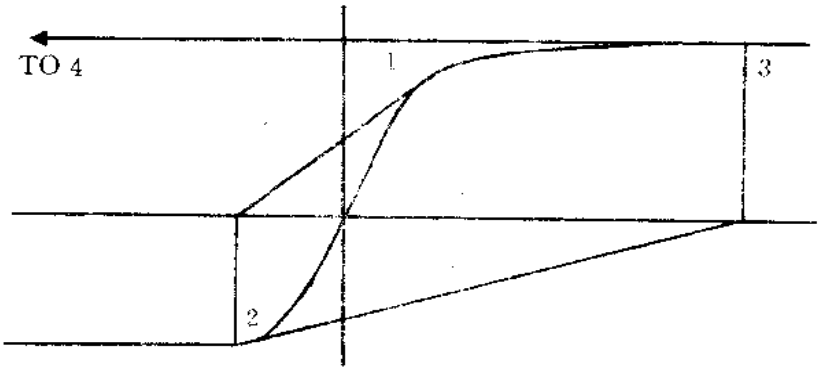


图 6—1　靠近拐点失效

当然此应急方法不完善。它没有改进dx的符号或dx太小这种情况。如果在猜测与要求的根之间有一个谷或者说局部极小，这两者之一就会产生。但它可使任何"合理的"函数作为NR方法处理。

该应急方法很快推广到多维问题。应该找到适当的测量误差的方法。显然向量f的长度$|f(x)|$是一个，或者也许向量的最大分量也是。在不用任何特别知识的情况下，将向量dx的每个分量减半是最保险的。

有人建议对NR迭代的一维形式，如果发现必须按这个方式分割步长dx，则在紧后的时间步长受到限制。例如，在每个时间步长里其长度不得超过原来的两倍。

§6.5　有关的方法

上述较简单的搜索方法较NR方法更稳健，但在接近根时收敛到此根要慢一些。由而一种结合这两者最好特点的方法便产生了，这种方法称为"变量度量"法。它们运用在从一些较简单的、在误差测度或者说度量大时更强的方法"进化"到快速收敛的NR方法上。当误差呈现相对地小而且使用NR迭代大概又安全又快速的时候。此外在估算Jacobian矩阵的计算时依赖于函数的估计，还依赖于计算这些导数是否有一解析表达式，或者这些导数必须数值计算。这导致所谓的"现代(updating)"方法，精确的Jacobian在每次迭代中不再计算，只近似地被更新，并且只循环地再次计算。结果，有了许多处理所讨论的一个或两

个问题兼有的“拟牛牛顿(quasi－Newton)”方法。

显然，函数 $F(x)=|f(x)|$ 的极小化问题与求 $f(x)$ 根是相同的，因为这里 $F(x)$ 将取得其最小可能值——零。于是为极小化一个函数 $F(x)$ 的更一般的问题提供了一求根方法(root－finding method)。请注意在求零点的问题里使我们易出现的“谷”在此新方法里已成功地实现局部极小。

§6.6 程序和测试情况

决定汽化的砂子的离子化平衡，测试问题是典型的物理问题。这样的等离子区有硅原子与氧原子按 2：1 的比例与通常的电子海洋处于平衡。整个的边界电子与自由电子必须守衡，并且由于此物质可以假定一直是初始不带电荷，它应没有净电荷。因此，其电子数应与离子带的电荷相等。此外，对氧和硅，原子核的数目应守恒。

除了三个守恒方程之外(电子、氧、和硅)，我们有介于这几种之间的离子化平衡 Saha 方程，这些可以认为是应用所详述过的平衡定律于此情形的结果。例如，O[0]表每立方厘米原子中没有离子化的氧的密度，O[1]则表单个地离子化了的氧的密度，等等。由于一个电子与单个离子化的氧离子的再组合，O[0]的形成率与 O[1]ne2 成比例，其中 ne 是电子数的密度。这里假定再组合是遵循三体碰撞力学的，在导出平衡公式中，必须假定碰撞的过程。如果在再组合中的动量与能量守恒，需要三体都参加碰撞过程。在局部热动力平衡条件下我们不处理放射再组合。此等离子区必须是光学厚度，并且这样的放射没有净(net)效应，因为它将形成离子化原子的放射“浴”。此中性构成率，或者说从 O[1]到 O[0]的损失必须与从 O[0]到 O[1]的构成度平衡。由此逆过程进行碰撞离子化，其中，一个电子与一个中性氧原子碰撞产生两个电子和一个 O[1]离子，这样的过程与 O[0]ne 成比例的速率发生。所以在形如：

$$O[0]ne/O[1]=A$$

的速率之间必须有一平衡，其中 A 不依赖于密度 O[i]与 ne 的数字，但依赖于所涉及的碰撞过程的物理学。从而，A 依赖于温度，因为在等离子区，温度控制着微粒运动的快慢，从而控制微粒发生碰撞有多频繁；以及必须转移多少能量去离子化原子。按上式，A 应该有如下的一般形式 $B\exp(-I/KT)T^{3/2}u\{O[1]\}/u\{O[0]\}$，其中 $u\{\}$ 是所涉及几种东西的统计权重(典型地，对原子状态，它是 $(2J+1)$，其中 J 是状态的自旋)。统计权重的因子数 2 已被吸收入常数 B 中，I 是离子化位势，K 是波尔兹曼常数，T 是温度，温度 T 与 I 以电子伏特量度，$B=6.e21$(大约)。有的书中给出类似的形式和指数[2]，他们是用这几种东西的分压而不是它们的密度，他们给出额外的 T 的因子在此表达式中(以及改变 B)。

该程序假定氧原子、硅原子与分离的电子海洋在相互作用(它们不这样)，并且只出现单个的离子化的原子的情况下，产生一初始逼近，非常适合于低温状态，于是 $ne=O[1]=xO[total]$ 而 $O[0]=(1-x)O[total]$，x 是分数的离子化(fational ionazation)，这允许用这些守恒方程得 Saha 方程的闭型解。这里必须回避一个有趣的数值陷阱。在解电子密度的二次方程中，我们有一般的表达式：

$$(\sqrt{(1+2a)}-1)/a。$$

其性质是：对于小的 a，其平方根近似于 $1+a$，而且此表示式应估值到 1，即完全离子化，(对大的 a，此式趋于零)。然而，在有限精度的算术计算中，对充分小的 a，此平方根接近 1，对大的 a 接近零。此函数对于从小的 a 变到大的 a，单调地从 1 变到零，而在还算小的 a 处有一极大值，在大和小的 a 处为零。此陷阱说明了在两个同号的数相减或相差不多的数量相减时精确度的损失，以及使用"精确"公式不谨慎易产生的错误。

用数值微分法得到 Jacobian 矩阵。建议读者用数值方法得出 Jacobian 矩阵，至少在初步调试程序时。其理由是，这较能保证 Jacobian 矩阵与你要解的方程系是相容的。一旦你这样调试代码，通过求出部份或者全部 Jacobian 矩阵的解析表达式，你可以节省计算机费用。

主程序是测试有问题的驱动器，它接受以 grams/c.c 为单位的密度作为输出。然后它解一类温度方程系统，迭代到希望的收敛判定，这些判定是输入(样品程序运行时要求用的适当值)。对有兴趣的温度体制，只有每个原子的非一离子化状态以及单个的或双个的离子化状态才加以考虑。Saha 方程的特别形式，如在函数 Sahc() 中执行的，把所有的密度数目小的放在分子上，而把大的放在分母以减小数值上的困难。这种形式是在用别的，不那么适合的的表达式在实验后得出。函数 dr() 施行构成 Jacobian 矩阵所要求的数值微分。它返回到偏导数 $df[i]/dx[j]$ 差。$dx[j]$ 是由各常数项的和加一个与 $x[j]$ 成比例的项构成，为的是按 x 提供一个有意义的区间，无论 $x[j]$ 是零或者 $x[j]$ 非常之大，子程序 Saha() 就做这件事，即产生初始猜测(当我们处理温度时我们用它作为先前较低温度解的猜测，除了第一种情形而外)。此程序的大部份作 NR 迭代，包括在剩余误差向量的长度增加时作拟合。

在上述情况中不曾写过总程序，比方说，nr()，它要调用 Sahac()。要写的话就会阻止用户修改这样的程序的"内脏"。例如，在这里讨论的离子化平衡问题，防止总体变成负的或大于全体可以得到的品种总体是有用的。一般说来，要修改这个程序用于别的问题不困难。表 6—1 列出了本章工具中用到的程序。

表 6—1　第六章的函数和子程序

Sahac	双函数型(Type double function)，返回 Saha 方程的右端
eqn	双函数型：返回每个求解方程的剩余(如果方程是严格满足，则返回零)
dr	双函数型：施行数值微分以构成 Jacobian 矩阵
Saha	解非线性离子化平衡方程系统的 Terforms 迭代。

从我们的样本情况可以看出我们的初始逼近不是太好的，在起点温度处，低温的温度近似就差。的确，对 NR 方法用于拐点时作的修改不起作用，而且剩余开始增加。但一旦我们接近于解，收敛就非常快。随后的较高温情形随着初解(较低温时的解)给出的猜测而快速收敛。

§6.7 程序列表与测试问题输出

```
/* Newton Raphson iteration solution of system of nonlinear
equations. Modified for greater robustness
near points of inflection.

TEST PROBLEM:
solve the Saha equations for the ionization equilibrium of
SIQ2

DEPENDENCIES:
uses linear system solver from chapter 3
uses pv() function in vector.c
*/

#define abs(x) ((x)>=0? (x):-(x))

/* in-line functions for use with 2D arrays: */

/* row major order as in C indices run 0..n-1 as in C */
#define INDEX(i,j) [j+(i)*coln]

/*various loop constructors */
#define DOFOR(i,to) for(i=0;i<to;i++)
#define DFOR(i,from,to) for(i=from-1;i<to;i++)
#define DOBY(i,from,to,by) for(i=from-1;i<to;i+=by)
#define DOBYY(i,from,to,by) for(i=from;i<to;i+=by)
#define DOBYYY(i,from,to) for(i=from;i<to;i++)
#define DOV(i,to,by) for(i=0;i<to;i+=by)
/* row major order as in C indices run 1..n */
/* #define INDEX1(i,j) [i-1+(j-1)*rown] */
/*
/* column major order, as in FORTRAN: */
#define INDEXC(i,j) [i-1+(j-1)*rown]
/* usage: if a(20,30)is matrix,then
a(i,j) in C will be a INDEX(i,j) if n=30 */

#define min(a,b) ((a)<(b)? (a):(b))
#define max(a,b) ((a)>(b)? (a):(b))

float pop[10],fns,fno,fscale,dmult,dadd,if,conv,conv2,zroot;

#define ne 8
```

```
#define neq 9

main(argc,argv) int argc;char * *argv;
{
float rho,t,fne;
int i,j;
printf(" enter density (g/cc) ");
scanf(" %f",&rho);
printf(" rho=%f\n enter conv,conv2,rcon",rho);
scanf("%e %e %e",&conv,&conv2,&rcon);
printf("echo %e %e %e\n",conv,conv2,rcon);
pop[8]=0.;
fno=rho/(1.6e-24*(16.+.5*28.));
fns=.5*fno;
for(i=1;i<9;i++)
{t=1;
fne=saha(fns,fno,t);
printf(" t=%f rho=%f/n",t,rho);
pv(pop,neq);
}
exit(0);
}

/* u[]i[] 1=1 silicon 2=oxygen are the statistical weights
of the ionic ground states
  pi are the ionization potentials in eV
  */
float u[12][2]={1.,5.,2.,4.,1.,1.,2.,3.,1.,1.,4.,
2.,5.,1.,4.,2.,1.,0.,3.,0.,1.,0.,3.,0.};
float pi[12][2]={8.149,13.614,16.34,35.146,33.46,54.934,
45.13,77.394,166.7,113.873,205.11,138.080,
246.41,739.114,303.87,871.12,351.83,0.,401.3,0.,476.0,0.,523.2,0.};

double sahac(iatom,istate,t)
float t;
int iatom,istate;
{
double sa; double exp();

sa= 6.e21*u[istate+1][iatom]/u[istate][iatom]
   *exp(-pi[istate][iatom]/t);
return (fscale/sa);
}
double eqn(i,f) float f[]; int i;
{
double x;
/* the equations to be solved-Saha unless noted */
```

```
switch (i)
  {
  case 0: x=f[ne]*f[1]-f[0]/sahac(0,0,tf);
    break;
  case 1: x=f[ne]*f[2]-f[1]/sahac(0,1,tf);
    break;
  case 2: x=f[ne]*f[3]-f[2]/sahac(0,2,tf);
    break;
/* conservation of silicon nuclei: */
  case 3: x=f[0]+f[1]+f[2]+f[3]-fns/fscale;
    break;
  case 4: x=f[ne]*f[5]-f[4]/sahac(1,0,tf);
    break;
  case 5: x=f[ne]*f[6]-f[5]/sahac(1,1,tf);
    break;
  case 6: x=f[ne]*f[7]-f[6]/sahac(1,2,tf);
    break
/* conservation of oxygen nuclei: */
  case 7: x=f[4]+f[5]+f[6]+f[7]-fno/fscale;
    break;
/* charge neutrality: */
  case 8: x=f[1]+2.*f[2]+3.*f[3]
    +f[5]+2.*f[6]+3.*f[7]-f[ne];
    break;

  }
return (x);
}

/* numerical differentiation for forming jacobian matrix: */
double dr(ieq,j,f) int ieq,j;
float f[];
{
double hold,dx,dn;
hold=eqn(ieq,f);
dx=f[j]*dmult+dadd;
f[j]+=dx;
dn=eqn(ieq,f);
f[j]-=dx;
return( (dn-hold)/dx);
}

#define itmax 20
#define ihmax 10
#define fignore 1.e-5

float b[9],a[9][9];
```

```
saha(fns,fno,t) float fns,fno,t;
{
int iter,coln,info,pivot[9],i,j,ic,ih;
double sqrt(),eqn(),dr();
float resid,newresid,fm,arg,abarg,relarg,rworst,aworst,fnsi,fnoi;
float popmin,popmax;
tf=t;
fscale=1.e20;
fnsi=fns/fscale;
fnoi=fno/fscale;
popmin=0.;
popmax=10.*(fnsi+fnoi);

printf("f=%e %e scaled %e %e\n",fno,fns,fnoi,fnsi);
dmult=.001;
dadd=.01*fnoi;
/* initial guess */
if (pop[neq-1]==0.)
  {
  arg=sahac(0,0,t);
  if (arg>1.e-4)
    fm=(sqrt(1.+2.*arg)-1.)/arg;
  else
    fm=1.;

  pop[0]=fns*(1.-fm);
  pop[1]=fns*fm;
  pop[ne]=pop[1];
  pop[3]=.0001*pop[1];
  pop[2]=0.;
  arg=sahac(1,0,t);
  if (arg>1.e-4)
    fm=(sqrt(1.+2.*arg)-1.)/arg;
  else
    fm=1.;

  pop[4]=fno*(1.-fm);
  pop[5]=fno*fm;
  pop[neq-1]+=pop[5];/*electron number density*/
  pop[6]=.0001*pop[5];
  pop[[7]=0.;
  };
DEFOR(i,neq)printf(" guess pop=%e\n",pop[i]);

coln=neq;/* number of equations */
```

```
/*scaling*/
DOFOR(i,neq) pop[i]=pop[i]/fscale;
/*Newton—Raphson iteration*/
DOFOR(iter,itmax)
{

/*rhs residual vector—
also calculate sq. error */
resid=0.;
DOFOR(i,neq)
  {
  arg=—eqn(i,pop);
  resid+=arg*arg;
  b[i]=arg;
  DOFOR(j,neq)
    {/* JACOBIAN*/
    a[i][j]=dr(i,j,pop);
    };
  };
printf(" square residual sum=%e\n",resid);
if (resid<rcon)
    {
    ic=—1;
    goto converged;
    }
lufact(a,coln,neq,pivot,&info);

DOFOR(i,j)printf("resid=%e\n",b[i]);
if( info! =0)
  {
  printf(" bad jacobian,index=%d\n",info);
  goto salvage;
  }
backsub(a,coln,neq,pivot,b);
ic=—1;
rworst=0.;
aworst=0.;
DOFOR(i,neq)
  {
  arg=abs(b[i]);
  abarg=abs(pop[i]);
  relarg= (arg/(abarg+fignore));
  if(relarg<conv || arg<conv2)continue;
  ic=i;
  rworst= max(rworst,relarg);
  aworst= max(aworst,arg);
  };
```

```
converged;
if(ic<0)
  {/* converged */
  DOFOR(i,neq)pop[i] *=fscale;
  return;
  };

printf(" no converge i%d,f[i]=%e,b[i]=%e\n rworst=%e,aworst=%e\n"
,ic,pop[ic],b[ic],rworst,aworst);
/* no good&/
salvage:
DOFOR(j,neq)printf(" f=%e,df=%e\n",pop[j],b[j]);
  /* tentatively update solution */
  DOFOR(i,neq)
    {
    pop[i]+=b[i];
    pop[i]=max(pop[i],popmin);
    pop[i]=min(pop[i],popmax);
    }
DOFOR(ih,ihmax)
  {
  /* have things gotten better? */
  newresid=0.;
  DOFOR(i,neq)
    {
    arg=-eqn(i,pop);
    newresid+=arg*arg;
    };
  if(newresid<resid)break;/* exit ih loop */
  /* things got worse */
  printf(" new residual=%e worse\n",newresid);
  /* cancel half of the previous update */
  /* this will ensure convergence near inflection points
  -it assumes we are going in the right direction with b[]
  corrections,merely too far */
  /* example- if pop were initially 0,b 1,then
       first update(above) pop=1
       after loop below(once) b=.5 pop=1-.5=.5
            twice b=.25 pop=.25
            thrice b=.125 pop=.125
            etc.
  */
  DOFOR(i,neq)
    {
    b[i] *=.5;
    pop[i]-=b[i];
    }
```

```
  }/* ih loop */;
/* continue NR iteration */
}/* iter loop */
  printf(" no convergence NR\n");
exit(0);
}
nr
enter density (g/cc) .01
rho=0.010000
enter conv,conv2,rcon .01 .001 1.e-10
echo 9.999999E-3 1.000000E-3 1.000000E-10
f=2.083333E20 1.041666E20 scaled 2.083333E0 1.041666E0
guess pop=8.010783E19
guess pop=2.405883E19
guess pop=0.000000E0
guess pop=2.405883E15
guess pop=2.060879E20
guess pop=2.245425E18
guess pop=2.245425E14
guess pop=0.000000E0
guess pop=2.630425E19
square residual sum=1.293816E-3
resid=-3.550125E-2
resid=5.781278E--7
resid=-6.328299E-6
resid=-2.404790E-5
resid=-5.785418E-3
resid=-5.907848E-7
resid=5.612334E--28
resid=-2.190226E-6
resid=-7.667548E-5
no converge i 8,f[i]=2.630425E-1,b[i]=-7.457716E-2
rworst=6.955301E-1,aworst=7.457716E-2
f=8.010783E-1,df=5.896924E-2
f=2.405883E-1,df=-5.897768E-2
f=0.000000E0,df=1.663697E-6
f=2.405808E-5,df=-1.723843E-5
f=2.060879E0,df=1.562403E-2
f=2.245425E-2,df=-1.562456E-2
f=2.245966E-6,df=-1.609195E-6
f=0.000000E0,df=1.966248E-9
f=2.630425E-1,df=-7.457716E-2
square residual sum=2.070351E-5
resid=-4.398379E-3
resid=1.228859E-7
resid=-1.285291E-6
resid=-5.398462E-8
```

```
resid=-1.165236E-3
resid=-1.199371E-7
resid=-3.506935E-10
resid=-6.143359E-8
resid=1.901140E-8
no converge i 8,t[i]=1.884654E-1,b[i]=-1.400441E-2
rworst=8.294959E-1,aworst=1.400441E-2
f=8.600475E-1,df=8.317695E-3
f=1.816106E-1,df=-8.312116E-3
f=1.663541E-6,df=6.691557E-7
f=6.819774E-6,df=-6.312907E-6
f=2.076503E0,df=5.674012E-3
f=6.829688E-3,df=-5.673494E-3
f=6.363882E-7,df=-5.890997E-7
f=1.860784E-9,df=-1.806582E-9
f=1.884654E-1,df=-1.400441E-2
square residual sum=1.986340E-8
resid=-1.164064E-4
resid=9.340538E-9
resid=-8.830053E-8
resid=1.838912E-9
resid=-7.945388E-5
resid=-8.115858E-9
resid=1.162455E-29
resid=-9.927820E-8
resid=1.140078E-8
t=1.000000 rho=0.010000
printing vector from 0 to 3
8.683653E19 1.732985E19 2.333423E14 5.061333E13
printing vector from 4 to 7
2.082177E20 1.156194E17 4.651961E12 0.000000E0
printing vector from 8 to 8
1.744610E19
f=2.083333E20 1.041666E20 scaled 2.083333E0 1.041666E0
guess pop=8.683653E19
guess pop=1.732985E19
guess pop=2.333423E14
guess pop=5.061333E13
guess pop=2.082177E20
guess pop=1.156194E17
guess pop=4.651961E12
guess pop=0.000000E0
guess pop=1.744610E19
square residual sum=3.044321E0
resid=1.741305E0
resid=1.470990E-3
resid=-8.828534E-8
```

```
resid=-5.776573E-8
resid=1.103374E-1
resid=-7.711034E-9
resid=9.866029E-18
resid=-9.927820E-8
resid=1.140078E-8
no converge i 8,f[i]=1.744610E-1,b[i]=1.213494E0
rworst=3.264007E3,aworst=1.1213494E0
f=8.683653E-1,df=-6.945132E-1
f=1.732985E-1,df=6.542592E-1
f=2.333423E-6,df=4.025638E-2
f=5.061333E-7,df=-2.582452E-6
f=2.082177E0,df=-4.787276E-1
f=1.156194E-3,df=4.787288E-1
f=4.651960E-8,df=5.930091E-7
f=0.000000E0,df=-6.109744E-9
f=1.744610E-1,df=1.213494E0
squart residual sum=9.702127E-1
resid=-7.939402E-1
resid=-4.885090E-2
resid=2.619960E-7
resid=-2.016382E-6
resid=-5.809347E-1
resid=-7.204198E-7
resid=1.357565E-16
resid=-1.990666E-6
resid=-6.048591E-6
no converge i 8,f[i]=1.387955E0,b[i]=-4.242075E-1
rworst=7.980300E-1,aworst=4.242075E-1
f=1.738521E-1,df=1.387472E-1
f=8.275578E-1,df=-1.151531E-1
f=4.025872E-2,df=-2.359625E-2
f=0.000000E0,df=6.960353E-8
f=1.603449E0,df=2.618671E-1
f=4.798850E-1,df=-2.618682E-1
f=6.401098E-7,df=-3.894713E-7
f=0.000000E0,df=7.641646E-9
f=1.387955E0,df=-4.242075E-1
square residual sum=1.482673E-2
resid=-4.884917E-2
resid=-1.000972E-2
resid=4.208303E-8
resid=1.462170E-7
resid=-1.110868E-1
resid=-1.657640E-7
resid=-7.173309E-9
resid=-5.661243E-7
```

resid=9.003889E−7
no converge i 8,f[i]=9.637480E−1,b[i]=−9.433630E−2
rworst=5.187979E−1,aworst=9.433630E−2
f=3.125993E−1,df=−3.335788E−3
f=7.124046E−1,df=1.198560E−2
f=1.666246E−2,df=−8.649642E−3
f=6.884901E−8,df=−8.143664E−9
f=1.865316E0,df=8.902086E−2
f=2.180168E−1,df=−8.902122E−2
f=2.512058E−7,df=−1.797519E−7
f=7.443137E−9,df=−5.402530E−9
f=9.637480E−1,df=−9.433630E−2
square residual sum=7.246934E−5
resid=1.130781E−3
resid=−8.159789E−4
resid=−1.241073E−9
resid=−2.887902E−8
resid=−8.397907E−3
resid=−1.631016E−8
resid=−1.617787E−9
resid=−7.516875E−9
resid=−1.226033E−7
no converge i 5,f[i]=1.289956E−1,b[i]=−8.115259E−3
rworst=6.290624E−2,aworst=8.115295E−3
f=3.092635E−1,df=−1.895107E−3
f=7.243902E−1,df=2.741719E−3
f=8.012823E−3,df=−8.466367E−4
f=6.140588E−8,df=−6.454815E−9
f=1.954337E0,df=8.115268E−3
f=1.289956E−1,df=−8.115259E−3
f=7.070980E−8,df=−2.145347E−8
f=1.860784E−9,df=−1.634508E−9
f=8.694117E−1,df=−7.066746E−3
square residual sum=3.700210E−9
resid=1.939694E−5
resid=−5.983755E−6
resid=−1.502965E−9
resid=1.814718E−8
resid=−5.734247E−5
resid=6.037447E−10
resid=1.026067E−17
resid=−5.678423E−9
resid=6.452549E−8
t=2.000000 rho=0.010000
printing vector from 0 to 3
3.073685E19 7.271319E19 7.166188E17 5.582353E12
printing vector from 4 to 7

```
1.962452E20 1.208803E19 4.838039E12 0.000000E0
printing vector from 8 to 8
8.623450E19
f=2.083333E20 1.041666E20 scaled 2.083333E0 1.041666E0
guess pop=3.073685E19
quess pop=7.271319E19
quess pop=7.166188E17
quess pop=5.582353E12
```

第七章　复数算术、穆勒(Muller)和詹金斯—特劳布(Jenkins—Traub)方法

§7.1　本章目的

这一章将开发下列工具：在C语言中实行复数算术，包括通常的复函数，诸如极坐标到直角坐标的转换，取幂，等等；求任意函数的零点(可能为复数)；对詹金斯——特劳布算法这类问题用专门算法的多项式求其零点。

这里提出的复数算术包括在下面的等离子体(血浆、淋巴液、原生质、…)扩散函数和快速傅里叶变换的计算中用到。

§7.2　非线性方程的零点、多项式的根

这一章讨论有一个或多个根的单一非线性方程的解。用到的穆勒法在许多年来证实了是十分健全和有用的方法。它可以用来确定复函数或实函数的复数根。

这一章也引入了在C语言中用复数工作的说明和功能包。而这很难在FORTRAN中容易地作到，在FORTRAN里复数运算是语言的一个自然部分。然而，审慎地使用#define语句也相当容易地完成这一功能。注意，作为使用这些技术的结果，处理复数运算如除法的码是嵌在C语言码中的，避免在子例行子程序顶调用。如果C编译优化工作得好，结果码应该和FORTRAN中对应码同样有效。

如第六章所指出的，为了确定多项式的根必须用到迭代法，但是迭代法可能不收敛。珀里斯(Press)等人(数值方法)要求拉盖尔(Laguerre)方法"从开始点就保证向根收敛"。然而这一要求是不准确的，正如达尔奎斯特(Dahlguist)等人(数值方法)提出的："对具复数根的代数方程来说，对初值估计的任一选择拉盖尔方法的收敛性不再为真"。詹金斯和特劳布的术语"整体收敛根"只能对多项式求得。这一章是对多项式提出的，当函数不是多项式时用穆勒法。

确定多项式的根可能是一个病态问题。威尔金森(Wilkinson)(代数特征值问题参看第三章)给出了一个多次引用的多项式例子，该多项式的根给出了非常病态条件的问题。那个多项式就是$(x-1)(x-2)\cdots(x-20)$，其根是显然的。x^{20}项的系数为1，项x^{19}的系数为210，其余的系数按大小增加到常数项，其值为20!或者粗略地为2.43×10^{18}。现在想象从第二项即x^{19}的系数减去2×10^{-23}。当然，一般来说，计算机不会存储足够有效的数字来表示如此精确的210。威尔金森发现这个新多项式的准确根现在包含十个带有效虚数部分的复数根。其余的根，1,2,3,4无更多影响，20变成20.8469…，10和11变成$10.095\pm0.6435i$和它的共扼复数等等。显然，不管怎么好的数值方法对高次多项式都可能会有麻

烦。

§7.3 C语言中的复数算术

这一章我们引入了一个复数算术包，该包大量用到C语言的结构化数据类型和预处理。我们在第十章的FFT中还会再次遇到这个包。这与新的方案ANST标准是兼容的，该标准提供了下面的定义：

```
Struct complex
        {double x,y;
        }
```

和函数cabs。

复数表示为双浮点点数对组成的结构。由给定的这个定义，复数就容易构造成数组。为了以后使用，用预处理的#define特性去产生许多有用的函数。注意，编译程序在码上操作之前用预处理程序可以扩展这些函数，所以这些函数是"在线"编码而无须在子例行子程序顶上调用。又注意到为了简洁，复数乘法CMULT和除法CDIV被其它预定义函数所代替。函数定义了包括四种算术运算CADD()、CSUB()、CMULT()、CDIV()，全部带有三个自变量z,x,y，第一个(即z)是运算结果。绝对值函数cabs()的定义使之可以相似于"一般"abs()函数一样使用，该函数是对实数据类型、整数据类型用三重算子组合来实现的。CONJG()完成复数求共扼(不用单变量版本CONJ()，在Aztec C中也指出了避免预处理程序限制定义的函数个数的问题)。CTREAL()是用实数乘复数，CMPLX()是从给定的实部和虚部形成一个复数。正如FORTRAN函数的命名一样，存在着函数的变化对另一个复变量赋与一个复数值。

例行子程序库也找到了在复变量上的标准操作。表末为cexp()，对复数Z返还值为e^z。csqrt()返还复数的平方根(当然对任一数有两个平方根，第二根可以从第一根的负数得到)，clog()返还对数的主要截部。clog()要求的自变量由argmnt()提供，自变量是复数极坐标表示中的幅角部分。例行子程序argmnt()返还0到2π间的一个角。例行子程序详细注释了如何改变0到2π为$-\pi$到π之间的结果。函数polarxy()将复数的极坐标转变为直角坐标表示(x为实数部分，y为虚数部分)。

§7.4 穆勒方法

前一章曾以直线在猜想的解$x(n)$处n次迭代去拟合函数$f(x)$。该直线是在点$x(n)$处函数的切线，且外推至它的零点截断从而找到$x(n+1)$，下一次迭代就近似于$f(x)$的零点。这一过程包括了求导数$f'(x)$。此数可以由估计对小步长ds的$f(x+ds)$并微分得到数值近似值。事实上是实行割线的改型，在$x(n)$附近近似于$f(x)$的直线是曲线在该点的割线。

穆勒方法是抛弃直线拟合而代以用二次式拟合函数，这当然增加了复杂性，要求选取平方根去找出拟合二次式与x轴的交点。三个拟合点之间的区间不需要很小，只要能确

定切线即可。很自然地可以找到复数根，虽然用牛顿法处理复数根有点麻烦。二次式的两个根由于数值上的原因最好选取这样的一个，相加的两项有相同的符号，这样删去时也不失精确性。第五章讨论豪斯霍尔德法(Householder)反射时已证明了是正确的。与两点决定一条直线一样，二次式需要三点拟合。我们常常用迭代中的最后三点。程序按用户给出的输入"自动地"生成一个初始猜想的点集(参看下面)。

穆勒方法并不限于确定多项式的根，虽然在我们的测试情况中只用于该目的。与牛顿——辛普森(Newton－Raphson)迭代一样，多维问题不宜采用。

如果多项式的根希望多于一个，与第四章讨论过的紧缩过程相似要用到"紧缩"过程，所以程序不再求以前确定了的根。这很容易作到。如果对 $i=0,\cdots,k$ 已经知道了根 $r[i]$，就搜索函数

$$F(x)=f(x)/\{(x-r[0])(x-r[1])\cdots(x-r[k])\}$$

的根，不再求这些根 $r[i]$了。

§7.5 詹金斯——特劳布方法

詹金斯(M. A. Jenkins)和特劳布(J. F. Traub)开发了求多项式零点的高级三阶段迭代法。与穆勒方法一样，求一次根后用"紧缩"移去。第一次迭代与典型的贝努里迭代相似，第二阶段是同样的过程，但移动原点。事实上，贝努里方法是将根的问题转变成以该多项式为特征多项式的矩阵特征值问题，然后对矩阵采用幂法。这也可以想象成解释多项式为差分方程，例如：

$$x^3+ax^2+bx^1+c=0$$

可作为差分方程

$$y[k+3]=-ay[k+2]-by[k+1]-cy[k]$$

的特征方程，如果假设每一步在指数中用 X 乘 y 值。与在幂法中一样，从一个初始猜想向量如 $y[0]\cdots y[n-1]=0, y[n]=1$开始迭代直到收敛为止。

§7.6 程序和测试情况

§7.6.1 穆勒方法

我们的程序以孔特(Conte)和德·波尔的 FORTRAN 实现为基础。这些程序都作了运行，证实了穆勒方法的实用。这里的内容是解实系数单个三次式的复数根。

表 7－1 第七章的函数和子例行子程序

argmnt	求复数的幅角(按弧度)。
clog	复数对数的主值。
csqrt	复数的平方根(只返还一个根——另一个平方根为其负)。
cexp	复数指数。

polarxy	复数从极坐标转变为直角坐标表示。
muller	用穆勒方法求复函数的根。
setfr	用 muller 提供的紧缩函数的子例行子程序。
jt	詹金斯—特劳布方法求多项式的根。
errev	jt()多项式估价的误差界。
nexth	求下一个移位多项式。
cauchy	零点的模之下界。
calct	计算变量移位。
fxshft	固定移位(方法的第二阶段)。
noshft	没有移位(方法的第一阶段)。
vrshft	变量移位(方法的第三阶段)。
polyev	估计多项式。
cmod	复数的模。
cdiv	准确的复数除法,避免溢出。
scale	确定系数的尺度因子。

改变种类是使程序应用时更适合,这并不是教学用的例子。初始猜想是作为自变量如 g 传入,另两个猜想为 $g-0.5$和 $g+0.5$。在猜想点间的区间让用户控制,这里取 0.5,同时作为判断误差容限。

读者将注意到整变量 fnreal 允许求解程序尝试着只去求实根。如果允许这个选择,程序就用于所有的根。不难构造值的 fnreal 向量,第 i 个应用到第 i 个根。我们感到会使用户受到太多妨碍,因为用户一般并不要用参数去强求实根,所以对这样的问题也不要设置所有零点的向量。

§7.6.2 詹金斯——特劳布方法

我们的程序是以 CACM 算法 FORTRAN 码为基础的。

测试表明当作为计算时用到机器精度,有时编译程序产生了报告收敛失败的码,然而实际上已找到了正确的根。因此可能的话,用户可以更快地得到精度较差的根,程序允许对机器精度估价用一个乘数。在 IBM—PC 上测试,要求对 Turbo C 和 Quick C 两者都用乘数(约为)7,Aztec C 显然迁到好的乘数 1。是否要求乘数是独立于用 8087 协处理器(用 Turbo C 测试时)。

§7.7 程序列表与测试问题输出

```
Program Listing and Test Problem Output
/*
Muller's root solver for multiple complex roots of
arbitrary functions.

CONTENS:
```

```
double argmt(y,x)
  Returns the argument of the complex number with
  imaginary part y and real part x. Answer
  returned is angle between 0 and 2pi.

colg(x,ans) struct complex *x, *ans;
  returns *ans=complex logarithm of *x
csqrt(x,ans) struct complex *x, *ans;
  returns *ans = complex square root of *x

polarxy(r,angle,x,y)
  converts from rectangular to polar form

printc(x) struct complex *x;
  prints a complex number x

muller
  applies muller's method to find complex roots
  see listing for details

setfr
  called by muller to evaluated the deflated function

cexp(x,ans) struct complex *x, *ans;
   *ans = complex exponential of *x;

main
driver for test problem

fn
  test function whose roots are to be found.

DEPENDENCIES:
  complex.h header file
*/
#include "complex.h"

struct complex    ci,cl,c0,o,o2,ir;

int iterp;/* global to return count used */

/*
#define min(a,b) (((a)<(b))? (a): (b))
*/
#define max(a,b) (((a)>(b))? (a): (b))
#define abs(x) ((x)? (x):-(x))
```

```
double argmt(y,x)
double x,y;
{/* returns answer between 0 and twopi,as needed in
complex principal argument */
/* caveat— Aztec C returns 0,not + or —halfpi if x=0. ,y nonzero */
double ans,ratio,twopi=6.283185307,
  pi=3.14159265358979,halfpi=1.570796327;
double atan(),undef=0.0;/* change if desired */

if (x==0.){if(y>0.) return(halfpi);
    if(y<0.) return(pi+halfpi);
    return(undef);
    };
/* if —pi to pi: return (halfpi*sign(y)); */
ratio=(y/x);ans=atan(ratio);
/*atan returns answer between - halfpi and halfpi */
/* now move to correct quadrant between -pi and pi */
  if(ratio>0.){/*ratio,ans>0*/
    if (x>0.) return(ans);/* quadrant Ⅰ */
    /* else x<0. ,y<0. quadrant Ⅲ */
    return (pi+ans);
    };
/* else ratio,ans<0. */
if(x>0.)return(twopi+ans);/*quadrant Ⅳ */
/*else x<0. ,y>0. ,quadrant Ⅱ */
return(pi+ans);

/* if answer bwtn -pi and pi desired: if ans<=pi accept ans unchanged
  else ans -twopi
  this will affect quadrant Ⅲ,Ⅳ only.change:
    Ⅲ:pi+ans to ans—pi
    Ⅳ:twopi+ans to ans
*/
}

/*————————————complex logarithm function ————————————*/

clog(x,ans) struct complex *x,*ans;
{
double r,argmt(),sqrt(),log(),angle;
r= sqrt(CNORM(*x));
angle=argmt(x->y,x->x);
ans->x=log(r);
ans->y=angle;
return;
}
```

```
/* ----------comolex square root---------- */

csqrt(z,ans) struct complex *z, *ans;
{
double x,y,r,sqrt(),argmt(),angle;
r= sqrt(sqrt(CNORM(*z)));
angle=.5*argmt(z->y,z->x);
polarxy(r,angle,&x,&y);
ans->x=x;
ans->y=y;
return;
}
/* ------convert from polar to rectangular coordinates---- */

polarxy(r,angle,x,y) double r,angle, *x, *y;
{double sin(),cos();
 *x=r*cos(angle);
 *y=r*sin(angle);
return;
}

main (argc,argv) int argc;char **argv;
{
double ep1,ep2;
struct complex rts[12],delta;
int i,maxit=20,kn,n,fnreal,fn();
kn=0;n=3;fnreal=0;
ep1=1.e-8;
ep2=1.e-8;
delta.x=.5;delta.y=0.;
for(i=0;i<n;i++){
     rts[i].x=0.;
     rts[i].y=0.;
     };
printf(" in main");printc(&(rts[0]));printc(rts);printf("\n");
muller(kn,n,rts,maxit,ep1,ep2,fn,fnreal,&delta);
for(i=0;i<n;i++){printf(" root=");
     printc(&(rts[i]));
     printf("\n");
     };

exit(0);
}

/* ----------test function to find zeros (roots) of---------- */
```

```
fn(x,ans) struct complex *x, *ans;
{
/* x^3-x-1. */
struct complex cdum,cdu1,cdu2;
/*printf(" in fn ");printc(x); */
CLET(cdu1, *x);
CMULT(cdu2,cdu1,cdu1);
CMULT(cdum,cdu1,cdu2);
CSUB(cdum,cdum,cdu1);cdum.x=cdum.x-1.;
CLET(*ans,cdum);
/*printc(ans);printf("\n"); */
return;
}

printc(x) struct complex *x;
{
printf("%f %f",x->x,x->y);
return;
}

/*--------------------muller's method
  based on FORTRAN program of Conte and DeBoor---------- */

muller(kn,n,rts,maxit,ep1,ep2,fn,fnreal,delt)
int fnreal,n,kn,maxit;
double ep1,ep2;int (*fn)();
struct complex *delt,rts[];
{
double sqrt(),eps1,eps2,real;
int i,j,kount,ibeg,iend,zerodiv,setfr();
struct complex rt,h,delfpr,frtdef,lambda,delta,
cdum,cdu1,delf,dfprlm,num,den,g,sqr,frt,frtprv, *root;

struct complex *kptr,kludge;
/* tolerences */
eps1= max(ep1,1.e-12);
eps2= max(ep2,1.e-20);
/* skip any previously found roots */
ibeg=kn;
iend=kn+n;
CASSN(delta,delt);

/*printf(" muller");printc(delt);printc(&delta);
printf("\n");
for(i=0;i<n;i++)
  {printf(" roots=");
  root= &(rts[i]);printc(root);printf("rootptr %u",root);
```

```
    printf(" \n");
    };
  */
 for(i=0;i<iend;i++)
    {
    kount=3;
    /* initial guesses:guess,guess+ and -delta */
    start: CLET(h,delta);
    root=&(rts[i]);
    /* printf(" rootptr %u",root);printc(root); */
    CADD(rt, *root,h);
    /* printc(&rt);printf("\n"); */
    setfr(&delfpr,fn,&rt,i,rts,eps2,&zerodiv,&delta);
    if(zerodiv)goto start;
    CSUB(rt,rt,h);
    CSUB(rt,rt,h);
    setfr(&frtprv,fn,&rt,i,rts,eps2,&zerodiv,&delta);
    if(zerodiv)goto start;
    CADD(rt,rt,h);
    setfr(&frtdef,fn,&rt,i,rts,eps2,&zerodiv,&delta);
    if(zerodiv)goto start;
    lambda.x=-.5;lambda.y=0.;
    while(1)
    {
    /* next estimate for root */
    CSUB(delf,frtdef,frtprv);
    CMULT(dfprlm,delfpr,lambda);
    CLET(cdum,lambda);cdum.x=cdum.x+1.;
    CTREAL(cdum,cdum,(-2.));
    CMULT(num,frtdef,cdum);
    CTREAL(cdum,lambda,2.);
    cdum.x=cdum.x+1.;
    CMULT(cdul,cdum,delf);
    CMULT(cdum,lambda,dfprlm);
    CSUB(g,cdul,cdum);
    CSUB(cdum,delf,dfprlm);
    CTREAL(cdum,cdum,2.);
    CMULT(cdul,,cdum,num);
    CMULT(cdum,cdul,lambda);
    CMULT(sqr,g,g);
    CADD(sqr,sqr,cdum);
    if( fnreal && sqr.x<0.) {sqr.x=0.;sqr.y=0.;};
    csqrt(&sqr,&sqr);
    CADD(den,dqr,g);
    real= g.x*sqr.x+g.y*sqr.y;
    if (real<0.)
      {CSUB(den.g,sqr);};
```

```
  if(cabs(den)==0.) {den.x=1.;den.y=0.;};
  CDIV(lambda,num,den);
  CLET(frtprv,frtdef);
  CLET(delfpr,delf);
  CMULT(cdum,h,lambda);
  CLET(h,cdum);
  CADD(rt,rt,h);
  if(kount>maxit) goto fini;
  while(1)
    {
    kount++;
    setfr(&frtdef,fn,&rt,i,rts,eps2,&zerodiv,&delta);
    if(zerodiv)goto start;
    if( cabs(h)<eps1 * cabs(rt)) || cabs(frtdef)<eps2) goto fini;
    /* unconverged to desired tolerance */
    if( cabs(frtdef)<10. * cabs(frtprv)) break;/* compute next estimate */
    /* diverged cut step size */
    printf(" cutting stepsize\n");
    CTREAL(h,h,.5);
    CTREAL(lambda,lambda,.5);
    CSUB(rt,rt,h);
    };/* infinite while loop—unconverged */
    };/* infinite while loop— next extimate for root */
  fini:;root=&(rts[i]);
  /* printf(" befor fini ");printc(&(rts[i]));printc(&(rts[i+1]));
  printf("\n");printf(" %u\n",root); */
  rts[i].x=rt.x; rts[i].y=rt.y;
}/* loop over desired roots */
return;
}

setfr(frtdef,fn,rt,i,rts,eps2,zerodiv,delta) int *zerodiv,i;double eps2;
struct complex *rt, *frtdef,rts[], *delta;int (*fn)();
{int j;
double sqrt();
struct complex *root,den,cdum;
*zerodiv=0;
/* printf(" setfr");printc(rt);printf("\n"); */
(*fn)(rt,frtdef);
/* printf("in setfr");printc(rt);printc(frtdef);printf("\n"); */
if(i>0)
  {/* deflate for found roots */
  for (j=0;j<i;j++)
    {root=&(rts[j]);
    CSUB(den, *rt, *root);
    /* printf(" deflating for root");
    printc( root);printf("\n"); */
```

```
    if(cabs(den)<eps2){CTREAL(cdum,*delta,.1);
        root=&(rts[i]);
        CADD(cdum,cdum,*root);
        CLET(*root,cdum);
        *zerodiv=1;
printf(" zero denom. in setfr\n");
        return;
        }
    CDIV(cdum,*frtdef,den);
    CLET(*frtdef,cdum);
    }
  };/*i>0*/
/*printf(" return from setfr\n");*/
return;
}

/*----------complex exponential function-----------*/

cexp(x,ans) struct complex *x,*ans;
{
double cos(),sin(),y,z;
struct complex c1,c2;
y = exp (x->x);
c2.x= cos (x->y);
c2.y= sin (x->y);
CTREAL(c1,c2,y);
/*printf(" cexp");printc(&c1);printc(x);printf("\n");*/
CLET(*ans,c1);
return;
}


in main 0.000000 0.000000 0.000000 0.000000
root=-0.662359 0.562280
root=1.324718 0.000000
root=-0.662359 -0.562280

/*
Jenkins-Traub polynomial root finder

CONTENTS:see text

DEPENDENCIES:
  complex.h header file
*/

#include "complex.h"
```

```
struct complex  ci,c1,c0,o,o2,ir;

int iterp;/* global to return count used */

#define abs(x) ((x>0.)? (x):-(x))
#define min(a,b) (((a)<(b))? (a): (b))
#define max(a,b) (((a)>=(b))? (a): (b))

double baker;

main (argc,argv) int argc;char * *argv;
{
FILE *fopen(), *fileid;
int degree,i,fail;
double x,y,cmod();
float fudge;
struct complex a,b,z,coef[4],ans[3];
/* test problem */
CMPLX(a,3.,4.);x= cmod(&a);
printf(" test of cmod=%e\n",x);
CMPLX(a,1.,0.);CMPLX(b,0.,1.);
cdivid(&a,&b,&z);
printf(" enter fudge factor for eta");
scanf("%f",&fudge);
baker=fudge;
printf(" eta scaled by %e\n",baker);
printf(" test cdivid=");printc(&z);printf("n\");
degree=3;
CMPLX(coef[0],1.,0.);
CMPLX(coef[1],0.,0.);
CMPLX(coef[2],-1.,0.);
CMPLX(coef[3],-1.,0);
jt(coef,degree,ans,&fail);
if(fail)
  {
  printf(" failed\n");
  }
for(i=0;i<3;i++)
  {
  printf(" root=");
  printc(&(ans[i]));
    printf("\n");
  }
exit(0);
}
printc(z) struct complex *z;
{
```

```
printf("%f %f",z->x,z->y);
}

#define MAXDEG 50
struct complex p[MAXDEG],h[MAXDEG],qp[MAXDEG]
       ,qh[MAXDEG],sh[MAXDEG];
struct complex s,t,pv;
double are,mre,eta,infin,smalno,base,tempa[MAXDEG],tempb[MAXDEG];
double xx,yy,cosr,sinr;
int nn;

jt(op,degree,zero,fail)
int *fail,degree;
struct complex op[],zero[];
{
double bnd,xxx,scale(),cmod(),sqrt(),cauchy();
int i,idnn2,cnt11,cnt12,nm1,conv;
struct complex z;
mcon(&eta,&infin,&smalno,&base);
are=eta;/* factor of 2 my addition */
mre=2.*sqrt(2.)*eta;
xx=.70710678;
yy=-xx;
cosr=-0.069756474;
sinr=.99756405;
*fail=0;
nn=degree+1;

CMPLX(t,-99.,-99.);/* is t initially undefined in fxshft? */
if (op[0].x==0. && op[0].y==0.)
   {
   *fail=1;
   return;
   }
for(; (op[nn-1].x==0. && op[nn-1].y==0.)&&nn>=0 ;nn--)
   {
   printf(" zero constant term found\n");
   idnn2=degree-nn+1;
   CMPLX(zero[idnn2],0.,0.);
   }
for(i=0;i<nn;i++)
   {
   CLET(p[i],op[i]);
   tempa[i]=cmod(&(p[i]));
   }
bnd=scale(nn,tempa,&eta,&infin,&smalno,&base);
if(bnd!=1.)
```

```
  {
  for(i=0;i<nn;i++)
    {
    CTREAL( (p[i]),(p[i]),bnd);
    }
  }
findzero:
if(nn==1)return;
if(nn<=2)
  {
  cdivid(&p[1],&p[0],&zero[degree-1]);
CTREAL(zero[degree-1],zero[degree-1],-1.);
  return;
  }
  for(i=0;i<nn;i++)
  {
  tempa[i]=cmod(&(p[i]));
  }
bnd=cauchy(nn,tempa,tempb);
for(cntl1=0;cntl1<=1;cntl1++)
  {
  noshft(5);
  for(cntl2=1;cntl2<=9;cntl2++)
    {
    xxx=cosr * xx-sinr * yy;
    yy=sinr * xx+cosr * yy;
    xx=xxx;
    s.x=bnd * xx;
    s.y=bnd * yy;
    fxshft(10 * cntl2,&z,&conv);
    if(conv)
      {
      idnn2=degree-nn+1;
      CLET(zero[idnn2],z);
      nn--;
      for(i=0;i<nn;i++)
        {CLET(p[i],qp[i]);
        }
      goto findzero;
      }
    }
  }
 *fail=1;
return;
double errev(nn,q,ms,mp,are,mre)
int nn;
struct complex *q;
```

```
double ms,mp,are,mre;
{
double ans,e,cmod();
int i;
e=cmod(&(q[0])) * mre/(are+mre);
for(i=0;i<nn;i++)
  {
  e=e * ms+cmod(&(q[i]));
  }
ans=e * (are+mre)-mp * mre;
return(ans);
}

nexth(boolean)
int * boolean;
{
double t1,t2;
int n,nm1,j;
n=nn-1;
/* nm1=n-1; */
if( * boolean)
  {
  for(j=1;j<n;j++)
    {
    CLET(h[j],qh[j-1]);
    }
  CMPLX(h[0],0.,0.);
  return;
  }
/* else */
for(j=1;j<n;j++)
  {
CMULT(h[j],t,gh[j-1])
CADD(h[j],h[j],qp[j])
  }
CLET(h[0],qp[0]);
return;
}

double cauchy(nn,pt,q)
double pt[],q[];int nn;
{
int n,i,nm;
double x,dx,df,log(),exp(),f,xm;
pt[nn-1] * =-1;
n=nn-1;
nm=n-1;
```

```
x=exp(log(-pt[n]))-log(pt[0])/((double)nn);
if(pt[nm1]!=0.)
  {xm=-pt[n]/pt[nm1];
  x=min(xm,x);
  }
repeat:
xm=x*.1;
f=pt[0];
for(i=1;i<nn;i++)
  f=f*xm+pt[i];
if(f>0.)
  {
  x=xm;
  goto repeat;
  }
dx=x;
while(abs(dx/x)>.005)
  {
  q[0]=pt[0];
  for (i=1;i<nn;i++)
    q[i]=q[i-1]*x+pt[i];
  f=q[n];
  df=q[0];
  for (i=1;i<n;i++)
    df=df*x+q[i];
  dx=f/df;
    x-=dx;
    }
return(x);
}

calct(boolean)
int *boolean;
{
int n,nm1,i;
double cmod();
struct complex hv;
n=nn-1; nm1=n-1;
polyev(n,&s,h,qh,&hv);
*boolean= (cmod(&hv)<= are*10.*cmod(&(h[nm1])));
if(*boolean)
  {
  CMPLX(t,0.,0.);
  return;
  }
cdivid(&pv,&hv,&t);
CTREAL(t,t,-1.);
```

```
return;
}
vrshft(13,z,conv)
int *conv,13;
struct complex *z;
{
double cmod(),mp,ms,omp,relstp,r1,r2,sqrt(),errev(),tp;
int boolean,b,j,i;
b=0;
*conv=0;
CASSN(s,z);
for(i=0;i<13;i++)
  {
  polyev(nn,&s,p,qp,&pv);
  mp=cmod(&pv);
  ms=cmod(&s);
  if(mp<=(20. * errev(nn,qp,ms,mp,are,mre)))
    {
    *conv=1;
    CSET(z,s);
    return;
    }
  if(i!=0)
    {
    if( !b && !(mp<omp) && (relstp<.05))
        {
        tp=relstp;
        b=1;
        if(relstp<eta)tp=eta;
        r1=sqrt(tp);
        r2=s.x*(1.+r1)-s.y*r1;
        s.y=s.x*r1+s.y*(1.+r1);
        s.x=r2;
        polyev(nn,&s,p,qp,&pv);
        for(j=0;j<5;j++)
            {
            calct(&boolean);
            nexth(&boolean);
            }
        omp=infin;
        goto skp;
        }
    if(mp*.1>omp)return;
    }
  omp=mp;
skp:
  calct(&boolean);
```

```
  nexth(&boolean);
  calct(&boolean);
  if(!boolean)
    {
    relstp=cmod(&t)/cmod(&s);
    CADD(s,s,t);
    }
  }
return;
}

fxshft(12,z,conv)
int 12, *conv;
struct complex *z;
{
int i,j,n,test,pasd,boolean;
double cmod();
struct complex svs,ot,lou;
n=nn-1;
polyev(nn,&s,p,qp,&pv);
test=1;
pasd=0;
calct(&boolean);
for(j=0;j<12;j++)
  {
  CLET(ot,t);
  nexth(&boolean);
  calct(&boolean);
  CADD((*z),s,t);
  if( (!boolean &&test && (j!=(12-1)) )
      {
      CSUB(lou,t,ot);
      If( cmod(&lou)>=.5*cmod(z) )
          {pasd=0;}
      else if (!pasd) {pasd=1;}
      else
        {
        for(i=0;i<n;i++)
            {
            CLET(sh[i],h[i]);
            }
        CLET(svs,s);
        vrshft(10,z,conv);
        if(*conv)
          {
          return;
          }
```

```
            test=0;
            for(i=0;i<n;i++)
                {
                CLET(h[i],sh[i]);
                }
            CLET(s,svs);
            polyev(nn,&s,p,qp,&pv);
            calct(&boolean);
            }
          }
        else
          {
          }
      }
    vrshft(10,z,conv);
    return;
    }

    noshft(l1)
    int l1;
    {
    int n,nml,i,j,jj;
    double xni,cmod(),t1,t2;
    n=nn-1;
    nml=n-1;
    for(i=0;i<n;i++)
      {
      xni=((double)(n-i)/((double) n);
      CTREAL(h[i],p[i],xni);
      }
    for(jj=0;jj<l1;jj++)
      {
      if(cmod(&(h[nml]))> eta * 10. * cmod(&(p[nml])))
        {
        cdivid(&(p[n]),&(h[nml]),&t);
        CTREAL(t,t,-1.);
        for(i=1;i<=nml;i++)
          {
          j=nn-i-1;
        /*   t1=h[j-1].x;
          t2=h[j-1].y;
          h[j].x=t.x * t1-t.y * t2;
          h[j].y=t.x * t2+t.y * t1; */
          CMULT(h[j],t,h[j-1]);
          CADD(h[j],h[j],p[j]);
          }
        CLET(h[0],p[0]);
```

```
    }
  else
    {
    for(i=1;i<=nm1;i++)
      {
      j=nn-i-1;
      CLET(h[j],h[j-1]);
      }
    CMPLX(h[0],0.,0.);
    }
  }
return;
}

polyev(nn,s,p,q,pv)
int nn;
struct complex *s,p[],q[],*pv;
{/* nested polynomial evaluation */
int i;
double t;struct complex temp;
CLET(q[0],p[0]);
CLET((*pv),q[0]);
for(i=1;i<nn;i++)
  {
/*  t=pv->x *s->x -pv->y*s->y +p[i].x;
  pv->y=pv->x*s->y+pv->y*s->x+p[i].y;
  pv->x=t;
*/
  CMULT(temp,(*pv),(*s));
  CADD((*pv),temp,(p[i]));
  CASSN(q[i],pv);
  }
return;
}

mcon(eta,infin,smalno,base)
double *eta,*infin,*smalno,*base;
{
/* will compute eta */
printf(" entered mcon");
for(*eta=1.;(1.+(*eta))>1.;(*eta)*=.5)
 *eta *=baker;/* my modification */
printf("eta=%e\n",*eta);
/*
for(*smalno=1.e-30;*smalno*=.5)
 */
 *smalno=(1.e-30);
```

```
    printf(" smalno=%e\n", *smalno);
/* set by hand as some compilers can't handle overflows/underflows */
 *infin=(1.e30);
 *base=2.;

printf(" leaving mcon\n");
return;
}

double cmod(x) struct complex *x;
{/* robust but expensive version of cabs() */
double ans,sqrt(),rpart,ipart,aux;
rpart= x->x; ipart= x->y;
rpart= abs(rpart);
ipart= abs(ipart);
if(rpart>ipart)
  {
  anx= ipart/rpart;
  ans=rpart*sqrt(1.+aux*aux);
  return(ans);
  }
else if (rpart<ipart)
  {
  aux=rpart/ipart;
  ans=ipart*sqrt(1.+aux*aux);
  return(ans);
  }
/*else*/
return (rpart*sqrt(2.));
}

cdivid(a,b,c) struct complex *a,*b,*c;
{/* robust c=a/b */
double r,d,dummy,infin;
if(b->x==0. && b->y==0.)
  {
  mcon(&dummy,&infin,&dummy,&dummy);
  c->x=infin;
  c->y=infin;
  return;
  }
if (abs(b->x)>=abs(b->y))
  {
  r=b->y/b->x;
  d=1./(b->x+r*b->y);
  c->x=(a->y*r+a->x)*d;
  c->y=(-a->x*r+a->x)*d;
```

```
  }
else
  {
  r=b->x/b->y;
  d=1./(b->y + r*b->x);
  c->x=(a->x *r + a->y)*d;
  c->y=(a->y *r - a->x)*d;
  }
return;
}

double scale(nn,pt,eta,infin,smalno,base)
double pt[], *eta, *infin, *smalno, *base;
int nn;
{
int i;
double scal,maxx,minn,hi,lo,x,sc,sqrt(),log(),pow();
hi=sqrt(*infin);
lo= *smalno/(*eta);
printf(" hi %e low %e nn %d\n",hi,lo,nn);
maxx=0.;
minn= *infin;
for(i=0;i<nn;i++)
  {
  x=pt[i];
  maxx= max(x,maxx);
  if(x!=0.)minn=min(x,minn);
  }
scal=1.;
if(minn>=lo && maxx<=hi)
  {
  return(scal);
  }
  x=lo/minn;
  if(x>1.)
  {
  sc=x;
  if( *infin/sc >maxx) sc=1.;
  }
else
  {
  sc=1./(sqrt(maxx)*sqrt(minn));
  }
printf(" before log,pow %e\n",sc);
i= (int) (log(sc)/log(*base)+.5);
scal= pow(*base,((double) i));
return(scal);
```

```
}


test of cmod=5. 000000e+000
enter fudge factor for eta 10.
eta scaled by 1. 000000e+001
test cdivid=0. 000000 -1. 000000
entered mcon eta=5. 421011e-019
smalno=1. 000000e-030
leaving mcon
hi 1. 000000e+015 low 1. 844674e-012 nn 4
root=1. 324718 -0. 000000
root=-0. 662359 0. 562280
root=-0. 662359 -0. 562280
```

第八章　B－样条扦值

§8.1　本章目的

在这一章里，我们提供一些工具：使象B－样条函数一类特别有用的样条曲线与数据相拟合。

§8.2　样条扦值

本书的其它章，特别是第四章和第七章，提到了与高次多项式相联系的各种问题。在匹配所希望的函数扦值时它们在那些点可能广泛摆动，解决办法是对扦值函数的象域的受限范围用低次多项式进行扦值。为了在这些范围的边界上使产生的曲线光滑，不仅要匹配函数值，而且应更小心地匹配斜率。这个一般过程被称为"样条"扦值。三次样条是十分通用的，正如四个系数的三次多项式，它们可以用来匹配函数值和每一端点的斜率，给出光滑的拟合。福瑟亚(Forsythe)、马尔可姆(Malcolm)和莫勒(Moler)对三次样条作了很好的讨论，同时提供了实现三次样条扦值的一对FORTRAN 例行子程序。

这里将提出称为"B－样条"的一般方法，对样条的应用能够提供更为有力和灵活的形式体系。当然，这是以程序复杂度和随之而来的计算花销为代价的。

§8.3　B－样条

B－样条或"基本"样条最初是肖恩堡(I. J. Schoenberg)开发的，但首先变成通用则是由于德·波尔(C. deBoor)为B－样条的数值应用开发了一个FORTRAN 例行子程序包。

样条函数隐含的思想是逐点定义的多项式近似拟合一个函数。例如，用在不同的不相重叠的区域上的三次多项式拟合一个三次样条函数。术语"样条函数"是指多项式的系数选择得适合于在边界上是连续的，那些边界是两个区域间的边界，不仅拟合的多项式之值相同，而且一个或多个导数也与斜率(一阶导数)等等相匹配。三次样条函数值和一阶导数(斜率)，对很多目标应该是联结得充分光滑的结果。

B－样条隐含的思想是在"基本"样条$B(x)$中扩展函数，其中$B(x)$在要拟合的区域上许多地方为0。

$B(x)$是样条函数，不是简单的多项式。即是说，在不同的区域上是不同的多项式。考虑最简单而有用的B－样条函数即三次样条。在$x[i]$和$x[i+3]$之间的区域中$B(x)$非0；对$x<x[i]$或$x>x[i+3]$，$B(x)=0$。且$B(x[i])=B(x[i+3])=0$，对三次B－样条函数来说函数$B(x)$可写成：

$$B(x)=A(x-x[i])_+^3+B(x-x[i+1])_+^3+C(x-x[i+2])_+^3+D(x-x[i+3])_+^3$$

其中$(x-y)_+=\begin{cases}x-y, & \text{若 } x-y>0\\ 0, & \text{否则}\end{cases}$

于是，在最左区间的端点处，只有与 A 成比例的项出现。而在最右区间端处，所有的项都出现。在最右区间的端点处，$B(x)$的导数的其它条件为 $B'(x[i+3])=B''(x[i+3])=0$，则 B 样条函数是唯一地符合于 B 的倍数的一个正规化常数。最后的 B—样条函数是非负的钟铃形状的函数(参看图 8—1a)。实际中一般使用的是三次 B—样条函数，但也可以定义高次(奇次)B—样条函数。

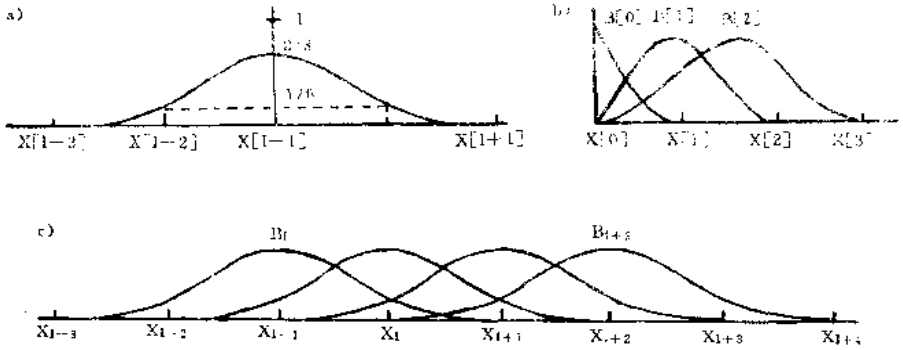


图 8—1　三次 B—样条函数

a)在一内点处的 B—样条　b)边界附近　c)三次 B—样条函数有影响的象域举例

除定义域端点外，在定义域边界上要对B—样条函数定义一些虚构点。典型地是德·波尔在左端选择了 $x[-3]=x[-2]=x[-1]=x[0]$，$x[0]$是定义域的端点，同样地可在右端作(参看图 8—1b)。在已定义的拟合点 $x[i]$象征性地称为样条的"结"或"断点"。通常，在定义域里结是均匀分布的，即 $x[i+1]-x[i]=\mathrm{d}x$ 是常数，虽然这不是必要的。

用样条函数拟合一个函数过程包括满足用户提出的条件的样条函数系数的确定。典型地是这些条件在指定的点集上确定的函数及其导数应该是匹配的。为了做到这些，应从重叠的 B—样条函数建立起一个近似于函数的样条函数。第一个三次 B—样条函数可以复盖从 $x[0]$到 $x[3]$的区域；下一个定义在 $x[1]$到 $x[4]$上，如此下去(参看图 8—1c)。回忆上面定义的每个三次 B—样条函数，它有一个自由度就是它的尺度因子。完成拟合就是要确定这个尺度因子。因为在象域端点 B—样条函数取值为零，故在每一内点上将有三个非零 B—样条函数出现在和的值里。需要求解的线性系统必须拟合近似于函数值集的 B—样条函数，应在定义域内部是三对角系统。因为在边上有多个结，故每一端点处多少是更复杂些，这样的系统仍然是带形的。所以与第三章一样，整个系统求解器没有复杂度，一般是能够求解的。

B—样条函数的特性是影响的局部性。拟合函数值只对在区间上B—样条函数非零的系数有影响。因此，三次样条函数只有四个系数受影响。可以有意识地制造"错误"并改变拟合点之一或改变一个边界条件来试验得出这一点，这对出现在区间的其它端点处系数或拟合无影响。

§8.4 程序和测试情况

例子包含了在区间 $0<x<1$ 内具有均匀的 $dx=0.1$ 的对函数 sin(pix)拟合的三次样条函数。B－样条函数的的灵活性允许我们在区间端点处强加上函数导数为零的条件。当然，这对三次样条函数一般是不能这样作的。但在此是可能的，因为结 $x=0$ 和 $x=1$ 是多重结，在这里有多于一个 B－样条函数出现在函数及其导数中，当然这是以增加系数和增加复杂度为代价的。对这些系数来说需要求解的线性系统在边界附近不再是简单的三对角形了。在 ACM 的集成算法中，德·波尔创作了算法 546，块解法。它是一个 FORTRAN 例行子程序的包，用这种样条函数拟合时适合于迁到的形式为矩阵。

德·波尔的 B－样条函数包是十分一般的。数组 $t[\ ]$ 按升序包含结的位置。如果结是一个多重断点，则在 $t[\ ]$ 中出现多次，数组 mult[ ]保持每一结点的多重数。如上面所指出的，端点典型地是大于 1 的多重数结点。也可能内点是有较大重数的结。为了使不连续一阶导数能够成功，对于三次样条函数对应的点必须是三重结。

假设用 d 次多项式的样条函数，在这个样条函数中系数个数为 $k=d+1$，例如 $d=3$ 的三次样条函数时为 4。参数 k 在包里是一个基本参数称之为样条函数的"阶"。令 n 为断点或结的个数包含端点在内，nb 为 B－样条函数系数的个数。变量 n，k 和 nb 是相关联的，但关系依赖于所用的结的重数。在数组 $t[\ ]$ 中，结是按次序存储的，是几重结就存储几次。这个数组必须为 $(nb+k)$ 长。

C 语言包的第一个函数 interv()是 BSPLINE 包要求的名字，并不是基于 BSPLINE 包中任意相似例行子程序。其目的是在数组 $t[\ ]$ 中定出给定点所在区间的位置。如果点在拟合象域中，变量 mflag 的返还值为零；若 x 大于最大结值，mflag 返还值为 $+1$；若 $x<x[0]$，mflag 返还值为 -1。在任意一种情形下，变量 ileft 的返还值都是靠近最右边点，用到将搜索区间重复取一半的二元搜索程序。照这样，要实行的测试次数为数组 $t[\ ]$ 中点数的对数阶，而不是数组点数的阶。这样就可以考虑存放很大的数组。

例行子程序的其余部分存放 main()是基于 BSPLINE 包。子例行子程序 bsplvb()计算 x 所指定的点处不为零的 B－样条函数的值。估价 B－样条函数用递归关系。子例行子程序 bsplvd()是用 bsplvb()计算导数，附加给函数值，应肯定德·波尔包约定自变量 nderiv 对 bsplvd()为 1 加上所希望的最高阶导数值。这些例行子程序的每一个都假定点 x 所在的区间通过左边整自变量所指定。子例行子程序 bvalue()不作这个假定而是用子例行子程序 interv()去确定其值。bvalue()的自变量 jderiv 是所希望的最高阶导数，当所希望的最高阶导数不止一个时，bvalue()的自变量为 nderiv。

主程序包含更多，它普遍应用于更多的 B－样条函数问题。首先设置测试函数，在对应的 $xi[\ ]$ 横坐标处的函数值作为数组 $yi[\ ]$。向量 mult[ ]包含每个结 $xi[\ ]$ 的重数，我们在这里对所有内结重数为 1，对所有端点重数为 k。主程序中下面的嵌套循环用 $xi[\ ]$ 和 mult[ ]设置 $t[\ ]$ 数组。边界条件和内点条件用方程来强加，这些方程应为拟合系数所满足，并为第三章(一般)系统求解程序所解得的线性系统所满足。如果拟合一个大的点集，用"带"求解程序存储时空是一个很好的想法。因为有匹配函数值的 n 个数据点，导数为零

的两个另外的条件，必须确定的B－样条函数系数为 $nb=n+k-2=n+2$ 个。如果作了只与函数值匹配的"有用"样条拟合，则只需要更少的系数即 n 去匹配 n 个数据点。这样就更容易在每一端点的一阶导数上使用条件。

虽然这个例子可同样应用于已隔开的结和三次样条函数，包是普遍的但也可用提供的子例行子程序实现其它的选择。注意混合样条函数的阶，即在象域的一端用三次样条函数而在另一端用四次不容易做。例如，对三次样条函数，结处三阶导数最大(或不连续)。如用哈斯汀(Hasting)程序得到拟合测定了误差曲线位置，即拟合与所希望函数间的差，不论加上新的结还是移动已有的结，误差都是最大的。汉森推荐了从少数结开始逐渐加上其它结(这些结处误差曲线在最大的误差值之间跨过零)的另一种程序。

德·波尔包有从样条函数近似变为多项式近似等等的多种例行子程序。样条函数在搭配方法时用来求解微分方程和两点边值问题。

在计算机图形应用中，B－样条函数按多种方法来使用。一般使用参数表示，其中B－样条函数是参数如 t 或 t 的二不同函数给出的点的位置 x 和 y 的多项式。当 t 在描述的限制内变化时画出该曲线。显然可以推广到曲面。非均匀有理B－样条函数NURBS，是对曲线或曲面拟合的B－样条函数。是拟合曲线或曲面的齐次坐标 (x,y,z,w)，并使用投影得到空间坐标 (w,y,z)。

表 8－1　第八章的函数

interv()	围绕指定点 x 之结的确定。
bsplvb()	估计基本B－样条函数。
bsplud()	计算给定B－样条函数拟合的函数和其导数的值。
bvalue()	求给定的B－样条函数拟合在一点的函数值

对于自变量的详细情况，参看下面程序所列。

§8.5　程序列表与测试问题输出

```
/* B—spline package
      (based on De Boor's FORTRAN package)

    test problem:fit sin (pi * x) for 0<=x<=1
    with constraints that the function has zero derivatives at the endpoints
    the test problem uses the linear system solution routines of Chapter 3
    to solve for the coefficients of the fit;
    the fundamental B—spline routines do not use the linear system solver
    themselves

    CONTENTS:
    interv
      determines which knots bound the position x
```

```
bsplvb
  evaluates the B-spline basis functions
bsplvd
  evaluates function and its derivatives from B-spline fit

bvalue
  returns j-th derivative of function from B-spline fit

main
  test driver

printm
  another version of the matrix printer, determines
  number of values/row from allowed column count.

DEPENDENCIES:
linear systems solver from Chapter 3
 */

#define DOFOR(i,to) for(i=0;i<to;i++)
#define jmax 20
#define INDEC(i,j,col) [j+(i)*col]
#define min(a,b) ((a)<(b)? (a):(b))
#define max(a,b) ((a)>(b)? (a):(b))

/* determines which interval (between knots) the location x is

  uses binary search

 */

interv(t,nbk,x,left,mflag) int nbk, *left, *mflag;
float x,t[];
{
int i,n,top,btm;
ffloat below,above;
n=nbk-1;/* index of last value in t[] */
if(x<t[0])
  {*mflag=-1; *left=0;}
else
if(x>=t[n])
  {*mflag=1; *left=n;}
else
  {
  *mflag=0;
  /* search */
```

```
    /* binary search */
    btm=0;top=n;
    while(1){
      i=(top+btm)>>1;
      below=t[i];above=t[i+1];
  /* printf(" top,btm,i %d %d %d\n",top,btm,i); */
    if( below<=x)
      {
      if(x<above)
        {
         *left=i;
        return;
        }
        /* x>=above */
        btm=i;
        }
      else{      /* x<below */
        top=i;
        }
      }
    }
  return;
  }

  /* evaluates B-spline basis functions */

  bsplvb(t,jhigh,index,x,left,biatx)
  int jhigh,index,left;
  float t[],x,biatx[];
  {static int j=0; int jp1,i;
  static float deltal[jmax],deltar[jmax],saved,term;
  /* index=0 start from scratch else j+1 to jout generated */
  /* printf(" bsplvb jhigh %d index %d left %d j %d\n",jhigh,index,left,j); */
  if(! index)
    {
    j=0;
    biatx[0]=1.;
    if(jhigh<2)return;
    }
  for( ;j<jhigh-1;j++)
    {
    jp1=j+1;
    deltar[j]=t[left+j+1]-x;
    deltal[j]=x-t[left-j];
  /* printf(" x %f t %f %f %f %f\n",x,t[left+j+1],t[left-j],deltal[j],deltar[j]);
  */
    saved=0.;
```

```
  DOFOR(i,j+1)
    {
    term=biatx[i]/(deltar[i]+deltal[j-i]);
    biatx[i]=saved+term*deltar[i];
/*printf(" term=%e saved=%e ans=%e\n",term,saved,biatx[i]);*/
    saved=deltal[j-i]*term;
/*printf(" j=%d,i=%d in b\n",j,i);*/
    }
  biatx[jp1]=saved;
/*printf(" b[%d]=%e\n",jp1,saved);*/
  }
/*printf(" leaving b\n");*/
return;
}


/* calculates values and derivatives of a function
represented by a B-spline
uses bsplvb()
*/

bsplvd(t,k,x,left,a,biatx,nderiv) int k,left,nderiv;
float t[],x,a[],biatx[];
{
double fk,factor,sum;
int i,il,j,jlow,jpmid,kp1mm,ldummy,m,kp1,mhigh,ideriv;
kp1=k+1;
/*printf(" bsplvd k %d,left %d,nderiv %d\n",k,left,nderiv);*/
mhigh= min(nderiv,k);mhigh=max(mhigh,1);
bsplvb(t,kp1-mhigh,0,x,left,a);
/*de Boor uses column ordering of FORTRAN,sends biatx not a*/
if(mhigh==1)
  {
  DOFOR(j,k) biatx INDEC(j,0,k)=a[j];
/*   DOFOR(i,k)printf(" a[%d]=%e\n",i,a[i]);*/
  return;
  }
ideriv=mhigh;
for(m=2;m<=mhigh;m++)
{
jpmid=0;
  for(j=ideriv;j<=k;j++)
    {
    biatx INDEC(j-1,ideriv-1,k)=a[jpmid];
    jpmid++;
    }
  ideriv--;
```

```
    bsplvb(t,kp1-ideriv,1,x,left,a);
    }
  DOFOR(j,k) biatx INDEC(j,0,k)=a[j];
  /* DOFOR(i,k)printf(" a[%d]=%e\n",i,a[i]; */

  jlow=0;
  DOFOR(i,k)
    {
    for(j=jlow;j<k;j++)a INDEC(j,i,k)=0;
    jlow=i;
    a INDEC(i,i,k)=1.;
    }

  for(m=2;m<=mhigh;m++)
    {
    kp1mm=kp1-m;
    fk=kp1mm;
    il=left;
    i=k-1;
    DOFOR(idummy,kp1mm)
      {
      factor=fk/(t[il+kp1mm]-t[il]);

      DOFOR(j,i+1)
        {
        a INDEC(i,j,k)=
         factor*(a INDEC(i,j,k)-a INDEC(i-1,j,k));
        }
      il--;i--;
      }
    DOFOR(i,k)
      {
      sum=0.;
      jlow= max(i,m-1);
      for(j=jlow;j<k;j++)
        sum+=a INDEC(j,i,k) * biatx INDEC(j,m-1,k);
      biatx INDEC(i,m-1,k)=sum;
  /* printf(" biatx[%d][%d]=%f\n",i,(m-1),sum); */
      }
    }
  /* printm(a,4,4,4,4);
  printf(" leaving d\n"); */
  return;
  }

  /* ----------test driver---------- */
```

```
main (argc,argv) int argc; char * * argv;
{
/* don amos `example 2a— fit spline and check it out */
float xi[11],z,y,x,yi[11],t[17],cm[169],r[13],a[13],vnikx[16];
float work[16],bvalue();/*dimensioned kxk */
/* CAVEAT some differences between De Boor's book (1978) and
  previous articles; for example,bsplvd has an extra
  argument,the work array,in the later book.
*/
int pivot[11];
double sin();
int mult[11],interv(),ilft,i,nm1,m,imk,nbk,n,nb,kp1,nm;
int j,it,km1,k,jshift,nderiv,mflag,info;
k=4;n=11;/* cubic spline order k=4, 11 points */

/*test problem */
DOFOR(i,n)
  {
  xi[i]= x=i* .1;
  yi[i]=y=sin(x*3.141592653589);
  }

/* the rest of the routine is fairly general */
nm1=n-1;
mult[0]=k;
for(i=1;i<nm1;i++) mult[i]=1;
mult[nm1]=k;
j=0;
DOFOR(i,n)
  {
  DOFOR(m, (mult[i]))
    {
    t[j++]= xi[i];
    }
  }
nb=j-k;
DOFOR(i,nb)
  {
  DOFOR(j,nb) cm INDEC(i,j,nb)=0.;
  }
nbk=nb+k;
km1=k-1;
nderiv=3;/*second deriv b.c. */
ilft=k-1;
printf(" n=%d nb=%d nbk=%d\n",n,nb,nbk);
DOFOR(j,nbk)printf(" t=%f\n",t[j]);
DOFOR(i,n)printf(" mult=%d\n",mult[i]);
```

```
bsplvd(t,k,xi[0],ilft,work,vnikx,nderiv);
DOFOR(j,k) cm INDEC(0,j,nb)=vnikx INDEC(j,2,k);
r[0]=0.;
/* rest of equations */
nderiv=1;
DOFOR(i,nm1)
  {
  x=xi[i];
  interv(t,nbk,x,&ilft,&mflag);
/*printf(" ilft %d mflag=%d x %e\n",ilft,mflag,x); */
  jshift=ilft-km1+1;
  if(jshift+k >nb )jshift=nb-k+1;
  bsplvd(t,k,x,ilft,work,vnikx,1);
  DOFOR(j,k)
    {
    it=jshift-1+j;
    cm INDEC(i+1,it,nb)= vnikx INDEC(j,0,k);
/*if (! j)printf(" first it=%d\n",it); */
    r[i+1]=yi[i];
    }
  }
/*left limits of last two eqns */
nderiv=3;/*second derivative b.c. */
jshift=nb-km1;
bsplvd(t,k,xi[nm1],ilft,work,vnikx,nderiv);
DOFOR(j,k)
  {
  it=jshift+j-1;
/*if(! j)printf(" first it=%d\n",it); */
  cm INDEC(nb-2,it,nb)= vnikx INDEC(j,0,k);
  cm INDEC(nb-1,it,nb)= vnikx INDEC(j,2,k);
  }
r[nb-2]=yi[nm1];
r[nb-1]=0.;/*second deriv. =0 */

/* lufact backsub solve system cm x= r */
printm(cm,nb,nb,nb,nb);
lufact(cm,nb,nb,pivot,&info);
if(info1 =0)printf(" info=%d\n",info);
DOFOR(i,nb)printf("data r=%e\n",r[i]);
backsub(cm,nb,nb,pivot,r);
DOFOR(i,nb)printf("solution=%e\n",r[i]);
DOFOR(i,20)
  {
  x=i*.05;
  y=bvalue(t,r,nb,k,x,0);
```

```
    z=bvalue(t,r,nb,k,x,1)/3.14159265358979;/* deriv of sin=cos */
    printf(" x=%f sin=%f exact=%f cos=%f\n",x,y,sin(x*3.14159265),z);
    }

}
/* bvalue returns the jberiv-th derivative of a function
represented by a bspline, at any point x within the
range of the fitted function.

uses interv()
*/


float bvalue(t,bcoef,n,k,x,jderiv) int n,k,jderiv;
float t[],bcoef[],x;
{
int i,ilo,nmi,imk,j,jc,jcmin,jcmax,jj,kmj,km1,mflag,nm1;
float fkmj,aj[jmax],dl[jmax],dr[max];
if(jderiv>=k)return (0.);
interv(t,n+k,x,&i,&mflag);
if(mflag)return(0.);
km1=k-1;
if(km1<=0)return(bcoef[0]);
jcmin=0;
imk=i-k;
/*printf(" i %d,k %d,imk %d\n",i,k,imk); */
if(imk<-1)
  {
  jcmin=-1-imk;
  DOFOR(j,j+1)dl[j]=x-t[i-j];
  for(j=1;j<km1;j++)
    {
    aj[km1-j-1]=0.;
    dl[j]=dl[i];
/*printf(" j%d i%d aj[%d]\n",j,i,(km1-j-1)); */
    }
  }
else
  {
  DOFOR(j,km1) dl[j]=x-t[i-j];
  }
jcmax=k;
nm1=n-1;
if(nmi<1)
  {
  jcmax=k+nmi;
  DOFOR(j,jcmax)dr[j]=t[i+j+1]-x;
```

```
    for(j=jcmax-1;j<km1;j++)
      {
      aj[j+1]=0.;
      dr[j]=dr[jcmax];
      }
    }
  else
    {
    DOFOR(j,km1) dr[j]=t[j+j+1]-x;
    }
  /*printf(" jcmin %d max %d\n",jcmin,jcmax);
  DOFOR(j,km1)printf(" dr=%f dt=%f\n",dr[j],dt[j]);*/
  for(j=jcmin;j<=jcmax;j++) aj[j]=bcoef[j+imk+1];
  if(jderiv!=0)
    {
    DOFOR(j,jderiv)
      {
      kmj=km1-j;
      ilo=kmj-1;
      fkmj=kmj;
      DOFOR(jj,kmj)
        {
  /*if(ilo<0)printf(" warn1 ilo=%d\n",ilo);*/
        aj[jj]=fkmj*((aj[jj+1]-aj[jj])/(dt[ilo]+dr[jj]));
        ilo--;
        }
      }
    }
  if(jderiv!=km1)
    {
    for(j=jderiv+1;j<=km1;j++)
      {
      kmj=km1-j;
      ilo=kmj;

      DOFOR(jj,kmj+1)
        {
  /*if(ilo<0)printf(" warn2 ilo=%d\n",ilo);*/
        aj[jj]=(aj[jj+1]*dt[ilo]+aj[jj]*dr[jj])
        /(dt[ilo]+dr[jj]);
        ilo--;
        }
      }
  return(aj[0]);
    }
  return(0.);/* for De Smet compiler */
  }
```

```
int colpp=48;

printm(a,coln,rown,col,row) int rown,row,col,coln;float a[];
{
  int i,j,ne; btm,top,count;
  putchar('\n');
  btm=top=0;ne=colpp/11;
  while(btm<col)
  {
    top=min(col,(btm+ne));
    printf(" printing matrix columns %d to %d\n",btm,(top-1));
    DOFOR(j,row)
    {
      for(i=btm;i<top;i++)
        printf(" %10.2e", a INDEC(j,i,coln));
      putchar('\n');
    }
    btm+=ne;
  }
  return;
}

n=11 nb=13 nbk=17
t=0.000000
t=0.000000
t=0.000000
t=0.000000
t=0.100000
t=0.200000
t=0.300000
t=0.400000
t=0.500000
t=0.600000
t=0.700000
t=0.800000
t=0.900000
t=1.000000
t=1.000000
t=1.000000
t=1.000000
mult=4
mult=1
mult=1
mult=1
mult=1
mult=1
```

```
mult=1
mult=1
mult=1
mult=1
mult=4

printing matrix columns 0 to 3
6.00e+002 -9.00e+002 3.00e+002 0.00e+000
1.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 2.50e-001 5.83e-001 1.67e-001
0.00e+000 0.00e+000 1.67e-001 6.67e-001
0.00e+000 0.00e+000 0.00e+000 1.67e-001
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
printing matrix columns 4 to 7
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
1.67e-001 0.00e-000 0.00e+000 0.00e+000
6.67e-001 1.67e-001 0.00e+000 0.00e+000
1.67e-001 6.67e-001 1.67e-001 0.00e+000
0.00e+000 1.67e-001 6.67e-001 1.67e-001
0.00e+000 0.00e+000 1.67e-001 6.67e-001
0.00e+000 0.00e+000 0.00e+000 1.67e-001
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e-000
printing matrix columns 8 to 11
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
0.00e+000 0.00e+000 0.00e+000 0.00e+000
1.67e-001 0.00e+000 0.00e+000 0.00e+000
6.67e-001 1.67e-001 0.00e+000 0.00e+000
1.67e-001 6.67e-001 1.67e-001 0.00e+000
0.00e-000 1.67e-001 5.83e-001 2.50e-001
0.00e+000 0.00e+000 0.00e+000 0.00e+000
```

```
 0.00e+000 0.00e+000 3.00e+002 -9.00e+002
printing matrix columns 12 to 12
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 0.00e+000
 1.00e+000
 6.00e+002
data r=0.000000e+000
data r=0.000000e+000
data r=3.090170e-001
data r=5.877852e-001
data r=8.090170e-001
data r=9.510565e-001
data r=1.000000e+000
data r=9.510565e-001
data r=8.090170e-001
data r=5.877852e-001
data r=3.090171e-001
data r=7.932658e-013
data r=0.000000e+000
solution=5.513430e-009
solution=1.047140e-001
solution=3.141421e-001
solution=5.975338e-001
solution=8.224345e-001
solution=9.668300e-001
solution=1.016585e+000
solution=9.668296e-001
solution=8.224345e-001
solution=5.975337e-001
solution=3.141420e-001
solution=1.047140e-001
solution=7.932658e-013
x=0.000000 sin=0.000000 exact=0.000000 cos=0.999945
x=0.050000 sin=0.156430 exact=0.156434 cos=0.987710
x=0.100000 sin=0.309017 exact=0.309017 cos=0.951005
x=0.150000 sin=0.453979 exact=0.453991 cos=0.891026
x=0.200000 sin=0.587785 exact=0.597785 cos=0.808972
x=0.250000 sin=0.707088 exact=0.707107 cos=0.707122
```

x＝0. 300000 sin＝0. 809017 exact＝0. 809017 cos＝0. 587753
x＝0. 350000 sin＝0. 890983 exact＝0. 891007 cos＝0. 454001
x＝0. 400000 sin＝0. 951057 exact＝0. 951057 cos＝0. 309000
x＝0. 450000 sin＝0. 987663 exact＝0. 987688 cos＝0. 156438
x＝0. 500000 sin＝1. 000000 exact＝1. 000000 cos＝－0. 000000
x＝0. 550000 sin＝0. 987663 exact＝0. 987688 cos＝－0. 156439
x＝0. 600000 sin＝0. 951056 exact＝0. 951056 cos＝－0. 309000
x＝0. 650000 sin＝0. 890983 exact＝0. 891007 cos＝－0. 454000
x＝0. 700000 sin＝0. 809017 exact＝0. 809017 cos＝－0. 587753
x＝0. 750000 sin＝0. 707088 exact＝0. 707107 cos＝－0. 707122
x＝0. 800000 sin＝0. 587785 exact＝0. 587785 cos＝－0. 808973
x＝0. 850000 sin＝0. 453979 exact＝0. 453990 cos＝－0. 891026
x＝0. 900000 sin＝0. 309017 exact＝0. 309017 cos＝－0. 951004
x＝0. 950000 sin＝0. 156430 exact＝0. 156435 cos＝－0. 987710

第九章　自适应求面积

§9.1　本章目的

这一章给出找到自适应积分函数(即由误差估计控制估值函数)的工具。

§9.2　数值积分

这一章和下一章主要讨论"求面积(或正交)"的问题，即对已知函数积分。这一章将提出自适应方法，此方法能够估计自身的误差，如果误差超过一个特定的容限，还能够对解的估计进行改进。

设函数为 $f(x)$，在积分范围内，x 的任意值为已知；如果不是这样，例如，假设是用表的数据或其它测量数据，下面的讨论和第六章中的扦值方法，可以用来提供这个函数 $f(x)$ 的值。

偶尔也需要二重积分或多重积分，这可以从定义一个函数 $F(x)$ 作为 $f(x,y)$ 的积分，并计算 F 在 x 上的积分递归来完成。在 FORTRAN 语言中必须小心地试验着去进行，因为许多的 FORTRAN 实现都不支持递归，可能发现无穷菊花链程序。在 C 语言里就没有这个问题。事实上，自适应求面积程序将呈现出其本身的递归性。

§9.2.1　求面积

更经常地，数值积分是作为辅助工具去解决其它问题。在前面讨论过的有限元法和矩法中，在一个单元上得到函数展开式或权函数的解析积分通常是不可能的，必须用数值方法。这方面的一个特殊例子就是天线理论，扩展函数的表达式，例如可以由衍射的几何理论或某些有关的近似方法来提供。甚至当表达式是解析的，但却不简单的时候，求解系统中也必须用数值积分来确定系数。在这些情况下，被积分的函数常常是很复杂的——例如，电流靠近完全可导平面的边界，而该平面有衍射入射波，有奇异点，该奇异点是弱可积的。如果没有真实的说明很可能使人们头痛。力学里也有相似的问题发生，在有限元模型中典型地要为各单元设置质量矩阵 M 和刚度矩阵 K。

其次把积分作为问题的一个积分部分，传输问题就是一个例子。例如辐射，中子等等。辐射密度 I 是位置和方向的标量函数，在辐射中频率也是这样。或者中子的能量等函数 I 在一点的矩是在方向上的积分，用一个固定方向，加上方向余弦的幂作为权。平均密度 E 与该方向上 I 的积分成比例，即零阶矩(在 E 的定义里不同的作者用不同的因子)。通常向量 F 由关于坐标系三个轴的方向余弦和 I 的一阶矩组成。二阶矩和高阶矩也类似。二阶矩和力学中惯性张量相似。例如辐射场与物质的交互作用是依赖于 E 的函数。F 不是

边界条件就是在某一点感兴趣的主要结果，或者是兼有这两方面。处理这些问题时，可以不对 I 求解，而是去积分找到各阶矩——或者在求解过程中至少找到隐藏的积分公式。特别地，高斯求面积公式形成了传输理论的 P—N 方法的基础。

社会科学很少使用积分。重要的统计函数不是已经制好了表，就是已经有现成的积分公式（例如 β 函数或误差函数）。并对这些积分作了近似法。但是当你的工作在难以计算时，进行数值积分可能是必要的。

§9.2.2 牛顿——库茨(Cotes)公式和辛普森规则

以下我们从 $x[min]$到 $x[max]$对函数 $f(x)$积分，并将估计在点 $x[i]$上 f 的值，在 $x[i]$上 f 取值为 $f(i)$。

在空体积里求面积的公式完全由在 x 的范围上实施函数 f 的准备（扦值方法）来导出。准备函数的积分可以解析确定，通常是由某类多项式提供函数数据。这是因为一旦已经知道多项式系统，多项式就很容易积分。但是，那些多项式如果次数很高，在扦值点之间就容易广泛摆动。所以在数值分析中，扦值多项式常常是二次或三次的，四次或五次用得很少，高次的几乎永远不使用。用低次多项式提供较小范围，可以得到比较好的结果，比如第八章已讨论过的样条扦值和下面应用的方法。

“牛顿——库茨公式(规则)”是应用于积分的一种方法。若函数 $f(x)$在数据点 $x[i]$要求估值，并已提供了可积分的扦值函数，则等分 x 并通过这些点扦值得到最低次的多项式。当在连接的 $x[i]$之间的区间上积分，如果在某一时刻对各范围处理两个点，就用通过这两点的直线扦值（这些不重叠的每一区域都被分为积分的完全区域，称之为“象素”。对于线性扦值，象素可以从 $x[i]$扩展到 $x[i+1]$；对二次扦值则通过三点 $x[i]$，$x[i+1]$，$x[i+2]$，等等进行函数扦值）。这就产生了一个梯形，它的面积（积分）为：

$$(x[i+1]-x[i])*(f[i]+f[i+1])/2.$$

除非这个函数在整个被积范围上是线性函数，以 $x[1]=x[min]$，$x[2]=x[max]$应用上面的公式实际上是不够的。有两种可能的方法得到积分更好的近似：一是用更高次的扦值多项式，因为两点确定一条直线，要求三点确定二次式，于是就必须用更多的点估价函数。二是再分积分范围，对分后的每一部分再使用上面的规则。这两种方法被称为“合成”。扦值直线的线段一般都偏离函数，值比函数值小。如果这样做，在整个被积范围上将公式求和得到的梯形，规则中每一内点 i 对积分作了两次运算，一次是它在区间的右端，一次是它在区间的左端。进一步假设

$$h=x[i+1]-x[i]$$

为一个常数，则积分为：

$$h/2\{f(x[min])+2f(x[min+h])+2f(x[min+2h])+\cdots-2f(x[max-h])+f(x[max])\}$$

辛普森规则是用两个区间作象素扦值一个二次式。对这个象素，公式结果（下面给出推导）为：

$$h/3\{f(x)+4f(x+h)+f(x+2h)\}$$

对所有象素求和，产生合成公式

$$h/3\{f(x[min])+2f(x[min]+h)+4f(x[min]+2h)+2f(x[min]+3h)+\cdots+f(x[max])\}$$

在内点上系数为 2,4,2,4,…的更迭模式。注意:辛普森规则要求使用偶数个区间,而函数估值为奇数个。又 h 为一个区间的长或整个象素长的一半。

下面的牛顿——库茨规则对有三个区间的象素称为"辛普森 3/8 规则",对三次式是精确的。下面的博德(Bode)规则对有四个区间的象素及函数五个估值的五次扦值多项式是精确的。包括了象素端点值的这些公式被称为"封闭型"。也给出了开放型牛顿——库茨公式。例如,有两个区间一个内点的象素有"中间规则"

$$2hf(x[min+h]);$$

或有三个区间因此有两个内点的规则

$$3h/2(f[min+h]+f[min+2h]);$$

或四个区间的规则

$$4h/3(2f[min+h]-f[min+2h]+2f[min+3h]).$$

表 9—1 归纳了最低次牛顿——库茨公式的性质。注意辛普森规则特别的精确性,用辛普森 3/8 规则对同样的精确度则会偏离函数估值,偏少一些。虽然用于抛物线扦值(即二次或二次曲线),实际上这个公式对三次式也是精确的。因此,辛普森规则是一种较好的选择,它就是自适应积分模式的基础。

表 9—1 牛顿——库茨公式

规则名	n 次多项式的精确度	误差阶 h^*
梯形	1	3
辛普森	3	5
辛普森 3/8	3	5
博德	5	7

此处 h 为步长。

误差项与被积函数(在积分范围内某一点)的第 $n-1$ 次导数成比例。

§9.2.3 辛普森规则的推导

令 $y=A^2x+Bx+C$ 为要积分函数的准备,则

$$f[0]=C$$
$$f[1]=Ah^2+Bh+f[0]$$
$$f[2]=4Ah^2+2Bh+f[0]$$
$$x[i]=ih,i=0,1,2.$$

于是解出 A 和 B:

$$A=(f[0]+f[2]-2f[1])/2h^2$$
$$B=(2f[1]-3f[0]/2-f[2]/2)/h.$$

从 0 到 $2h$ 之下的面积为 $8Ah^3/3+2Bh^2+2Ch$,即 $(h/3)\{f[0]+2f[1]+f[2]\}$。

现在考虑三次的 $y=Dx^3$。在从 0 到 $2h$ 这个区间上,y 下的面积为 $4Dh^4$。还可得 $f[0]=0,f[1]=Dh^3,f[2]=8Dh^3$。辛普森规则给出面积为 $12Dh^4/3=4Dh^4$ 显然是精确的,因为积分是一个线性过程。当对任意二次和简单三次多项式辛普森规则精确时,对它们的

[illegible]般三次[illegible]式也是精确的。

设计[illegible]应辛普森求面积的例行子程序就是利用了这个优点，重复应用辛普森规则对有的函数积分也[illegible]定的作用。

§9.2.4 厄尔米特(Hermite)扦值及其它方法

为了处理具有已知[illegible]质的被积函数，还有各种不同的方法。例如，如果已知函数的导数或者可以很容易地而且很经济地(不用数值微分)估值，就可以用所谓"厄尔米特[illegible]扦值法"、"修正[illegible]则"：

[illegible]$min]+f[min+h]+h^2/12(f'[min]-f'[min+h]$、

其中 $f'[x]$[illegible]的导数估值，就是一个例子。

克伦肖([illegible])—柯里(Curit)方法作为"半自适应"类的方法是应该提到的，它[illegible]合[illegible]连续被[illegible]数。如果猜想函数是病态的[illegible]应该使用这个方法。然而，如果这个[illegible]合适时，对[illegible]精确度的函数要求估值一般应更加经济些。用切比雪夫多项式进行扦值的方法依靠快[illegible]傅里叶变换技术(参看第十章)使计算最优化。

罗门堡(Romberg)外推是另一种"半自适应"方法，它用到积分模式的误差估计，结合对不同步长估值积分，对零步长求极限估值积分。对非病态积分这个方法也是适[illegible]的。

§9.3 自适应求面积

在自适[illegible]!(自适应正交)中，允许用程序对结果误差进行估计，并与用[illegible]定[illegible]误差判断进行[illegible]如果满足，则接受此结果。如果不满足，程序就尝试产生更精确的回答。第十章和第[illegible]章里，IVP 求解程序用到相似的自适应技术，其中的时间步[illegible]间[illegible]解的两种不同计算里的差别已经作了调整，与自适应求面积(正交)完全相似。用两种方法估计积分，如果有足够[illegible]致性就接受该结果。典型地是更加细分区间，例如一[illegible]细分，使用在整个区间[illegible]样的公式[illegible]理子区间，再把结果加起来得到的第二个结果与[illegible]单个区间[illegible]分的结果作[illegible]较。[illegible]面积(正交)方法应用到这里是精确的，[illegible]到递归，[illegible]子区间的精确度作为[illegible]来检验。各区间用的公式为辛普森规则，如果不满足误差判断，就三等分该区域，再对[illegible]个子区域积分并求和。注意：递归的使用[illegible]应该集中效果的区域上例行子程[illegible]的效果。在 C 语言中使用递归例行子程序是[illegible]的，可以用数据结构"栈"来模拟递归。FORTRAN 中对递归法必须作变动，至今[illegible]FORTRAN 版本也不支持递归。是否[illegible]依赖于怎样更好地编译优化递归调用，[illegible]怎样更好地处理非递归码。福瑟西(Forsythe)、马尔可姆(Malcolm)和穆勒(Moler)[illegible]自适应求面积(正交)的非递归 FORTRAN 程序 QUANC8。这个程序基于 8[illegible]牛顿-柯茨公式而不是辛普森规则。

首先注意，若用辛普森规则估计区域的[illegible]就必须估计区域边界点的函数值。如果[illegible]决定细分区域，再计算细分区域上的面积，[illegible]以再次使用区域边界点已经确定的值。事[illegible]上，积分范围边界点的函数值已作为参数传送给积分例行子程序，所以不需要[illegible]计算它们。还应注意到递归深度是受到限制的，这样使困难问题处理时能阻止例行子程序[illegible]

地找寻"末尾"。参数 7 是误差容限(小数),在积分估值中它是差(绝对值)与绝对面积(即函数绝对值的积分)之比。原来出版的算法书本中如果误差均匀分布在三个子区间上,麦肯曼对变量步误差估计乘上 1/3 加以修正。然而在自适应求面积的任意积分以及算法的证明中很少发生这一情况。麦肯曼还推荐了用 1.7(即$\sqrt{3}$)减一步长,即是用 0.58 乘步长来修正改变。这个值必须看作"经验规则"值。特别难处理的积分大概是这一个比较小的因子,即是要更加逐步地归约步长。限制水平也可以跟着改变(原来发表的是假定的 7 而不是 20)。

§9.4 程序和测试性能

对 $f(x)=1/\mathrm{sqrt}(\mathrm{abs}(x))$在−9 到 1000 的区间中步长为 0.1 的积分,麦肯曼发表了一个完全直接的测试。

我们也用一个简单的二次式测试了希望从辛普森规则得到的精确结果。

注意,已经用了 19 个函数估值来产生这个结果。自适应求面积一般应该对"不好的"积分(函数带奇异点或函数快速改变)留作专用,或者检验里面经济的方法所得到的结果。

应该在很大程序上有保留地采用步长误差估计,这种误差估计基于近似结果(辛普森),所以它本身也是近似的。在求面积(正交)的情况下非偶然地删去积分中的一些点,即如果 $f(x)$在符号上快速地摆动,其中的小数误差可能会比所希望的大。(这里可以考到费龙(Filon)方法:用 sinx 或 cosx 乘被积函数。费龙方法简短的讨论,以及使用时的说明可以在数学函数手册中找到)。表 9—2 列出了这一章的子例行子程序

表 9—2 第九章的程序

adsimp	自适应辛普森求面积(Exx)

§9.5 程序列表与测试问题输出

```
/*
adaptive integration routine
uses Simpson's rule adaptively
(from "C Tools for Scientists and Engineers" by L. Baker)

CONTENTS:

adsimp
  adaptively computes integral of function

simp
  recursively applies Simpson's rule
```

```
main
  test driver

funct
  function to be integrated

DEPENDENCIES;
NONE
*/

/* #include "libc.h"
#include "math.h"
*/
#define abs(x) ((x)? (x):-(x))
#define max(a,b) ((a)>(b)? (a): (b))
int level,maxlev,levmax,feval;

double absimp(a,b,eps,f) double a,b,eps,(*f)();
{
double simp(),aa,epss,absarea,ans,fa,fb,fm,range,est;
levmax=1;
feval=3;
level=1;
maxlev=20;
aa=a;epss=eps;
absarea=1.;
est=1.;
range=b-a;
printf(" adsimp\n");
fa=(*f)(a);
fb=(*f)(b);
fm=(*f)(.5*(a+b)) *4.;
/* printf(" intergrand fa,fb,fm %e %e %e\n",fa,fb,fm)*/;
ans=simp(aa,range,fa,fm,fb,absarea,est,epss,f);
return(ans);
}

double simp(a,da,fa,fm,fb,area,est,eps,f)
double a,da,fa,fm,fb,area,est,eps,(*f)();
{
double absarea;
double funct();
double dx,x1,x2,est1,est2,est3,f1,f2,f3,f4,sum,epss,norm=.588;
absarea=area;
/*printf(" simp %e %e %e %e %e \n %e %e %e %d\n"
,a,da,fa,fm,fb,absarea,est,eps,level);*/
```

```
dx=.333333333*da;
epss=eps*norm;
x1=a+dx;
x2=x1+dx;
f1=4.*(*f)(a+.5*dx);
f2=(*f)(x1);
f3=(*f)(x2);
f4=4.*(*f)(a+2.5*dx);
/*
f1=4.*funct(a+.5*dx);
f2=funct(x1);
f3=funct(x2);
f4=4.*funct(a+2.5*dx);
*/
feval+=4;
est1=(fa+f1+f2)*dx*.166666666666;
est2=(f2+fm+f3)*dx*.166666666666;
est3=(f3+f4+fb)*dx*.166666666666;
absarea=area-abs(est)+abs(est1)+abs(est2)+abs(est3);
sum=est1+est2+est3;
level++;
levmax=max(level,levmax);
if ((abs(est-sum)>epss*absarea)||(est==1.)) && level<maxlev)
  sum=simp(a,dx,fa,f1,f2,absarea,est1,epss,f)
    +simp(x1,dx,f2,fm,f3,absarea,est2,epss,f)
    +simp(x2,dx,f3,f4,fb,absarea,est3,epss,f);

level--;
return(sum);
}

main(argc,argv) int argc;char **argv;
{
double funct();
double a,adsimp(),b,eps,ans;
a=0.;
b=1;
eps=.001;
ans=adsimp(a,b,eps,funct);
printf("fini\n");
printf("area=%e max depth=%d\n function evaluations=%d\n"
,ans,levmax,feval);
exit(0);
}

double funct(x) double x;
{
```

```
return(x * x);
}

absimp
fini
area=3. 333333E-1 max depth=3
function evaluations=19
```

第十章　傅里叶变换

§10.1　本章目的

这一章将给出数组的快速傅里叶变换(FFT)的计算工具。讨论该方法的理论以及如何将它推广到其它一些有意义的变换。

§10.2　防止误解的说明

如果你还没有用过 FFT，可能对这里给出的结论感到意外。在通常的数值傅里叶分析特别是 FFT 中存在着许多困难。例如 1986 年 4 月 21 日出版的物理评论通信第 56 卷 16 期 1757 页的勘误中指出的："从傅里叶变换算法得到的系数的扦值中的一个失误，……"，那个失误* 的作者都是该领域中富有经验而且受尊重的工作者。这一失误夸大了纯理论与计算之间的分歧，原始文章提出一个模型，对这个分歧说明其原因。考虑到该计算方法中的不确定性，伴随成倍的增长率，纯理论与计算结果却处于很好的一致。我们将试图指明运用 FFT 中主要的"困难"，并提出避免它们发生的办法。下面涉及的一切变量都假设是取复数值的。

§10.3　傅里叶分析及其应用

可以把傅里叶分析看作函数 $f(t)$表示为指数函数之和的简单展开，即：

$$f(t)=\sum a[n]\exp(i\omega nt) \qquad (1)$$

关于 $a[n]$的系数，可以简单地将右端各项表示为 $\sin(\omega nt)$和 $\cos(\omega nt)$的和。我们并不把 $f(t)$展开为某些任意的指数函数，而是展开为指数为纯虚数且为 $i\omega t$ 整数倍的指数函数。注意，这里的 n 在求和中通常必须包括既取正值又取负值。由于展开式的形式，函数 $f(t)$隐含着是 t 的周期函数的假定，$f(t+T)=f(t)$，其中 $T=2\pi/\omega$。

因为通常 f 是时间的函数，故视 t 为独立变量。基本角频率 ω 的整倍数 $n\omega$ 为高次谐波，通过不同频率的性能来分析电路性能时常常用到它，即设定的信号具有形如 $\exp(i\omega t)$ 的时间相倚性时用到它。标准电路成分的性能，如用电容仪、示波器对该信号作简单的描述。对这样的指数(正弦)的信号，电路方程转变为简单的代数方程而不是该信号的微分方程。通常电路可以有效地特征化，用作频率的有理函数来研究，称之为"转移函数"。

讨论"电磁谱"是自然的。"谱"这个词是涉及频域特性的，即作为 n 的函数，$a[n]$的分

---

* 物理评论通信第 48 卷 677 页，1982

面。当研究无限小的 ω 时，其极限使求和转变为积分，就得到了傅里叶积分，离散的 $a[n]$ 就变成连续函数 $F(\omega)$。离散变量 n 对应于频率 $n\omega$，在 $F(\omega)$ 中，我们用 ω 表示连续的自变量而不是一个特殊的基频 w，称之为"$f(t)$的傅里叶变换"。而 $f(t)$可以称之为在时间域上做工作，反过来，通过研究 $F(\omega)$的性质来分析同一系统时，则称之为在"频域"上做工作。

在光学里，简单的聚焦作用在焦平面上产生象目标的FT，这正是FT在光学里有意义的部分原因，通常用于激光共振器的分析。近来光学计算方式转移，改变了这一关系，主要是利用透镜"计算"傅里叶变换，而不用FT去计算光学系统的性质。在电学系统中假设为单频(光学术语称为单色)"驱动"或迫使系统允许最大的简化。在光学中FT的使用允许光学变换函数的定义和模转移函数的定义与电学中的转移函数相类似。

即使对基本上非周期的系统，例如随机系统，考虑其"频域"性质常常是有用的。或许部分在于测量设备的特性。人们常常听说"白"噪声，即一个不依赖于频率的随机信号，如 $a[n]$的大小表现为全都相同。或"有色"噪声，即"红"噪声在低频上更强而"兰"噪声在高频上更强。

在许多信号处理的应用中FT都是有用的。例如两个函数卷积的FT等于这两个函数FT之积。于是，可以由函数各自的变换相乘再求积的逆变换得到二函数的卷积。两个函数的相关函数或一个函数的自相关函数也可以得出类似的结论。在各式各样的信号分析过程中，例如预测编码与谱估计，卷积与相关都是十分重要的。

已知 $f(t)$时，求出 $a[n]$是我们的目的。特别是 $f(t)$的已知值是按周期方式测度时。具体地说，取定时间步长 $\mathrm{d}t$ 和整数值 $m=0,1,\cdots\cdots,M$，给出 $f(m,\mathrm{d}t)$(再次指出已假设 $f(t)$是周期函数，$f(0)=F(M\mathrm{d}t)$，当我们考虑到傅里叶分析时，这一点是不容忽视的)。这一章下面部分着重讨论针对这种等时段 $f(t)$测量值的特殊情形，从技术上看，这也是最为重要的情形。反过来说，在这种情况下，对 $a[n]$在频域上基频的整倍数给定为 ω 的等长区间的傅里叶系统数的研究也具有极重要的意义。我们将假设时域中有 N 个数据点，即只有 $f(n\cdot\mathrm{d}t)$在 $n=0,1,\cdots\cdots,N-1$ 的值为已知，则有可能解出 N 个系数 $a[n]$，得出$F(\omega)$的 N 个值，其中 $\omega=\frac{m}{T}$，$m=0,1,\cdots\cdots,N-1$，$(T=2\pi/(N-1))$。

当需要的频率值跨越很宽的范围时，这不是很有用的，FFT技术不可能有意义。老式的"慢"FT可能更适用，其中各点都分隔开了，即频率按对数取值。

对 $\omega=mW$ 的函数 $F(\omega)$，$a[n]$可以由下面方法得出：将(1)的各项乘以 $\exp(imkW\ell)$，其中 k 为整数，对 k 求和，利用指数函数的性质给出关系式：

$$a[n]=\sum f(m\ \mathrm{d}t)\exp(-inmW\ell)$$

对每一个 $a[n]$，我们都有对 N 个项求和的计算。甚至还假设所有涉及到的指数函数都已事先算出来并列成了表，我们将完成 N 次浮点乘法与加法运算。或者说，要完成全部运算需要进行 $2N^2$ 次浮点乘法。如果只有1000个数据点，就需要百万次浮点乘法。按照 N 的平方度量，这或许应该抑制这种浪费。

§10.4 FFT

所谓"快速傅里叶变换"(FFT)就是计算系数 $a[n]$，或等价地计算 $F(\omega)$的一种有效方

法。如果涉及 $F(\omega)$ 的计算，则只是在固定的离散点处的值。这个想法于 1965 年形成，当库勒(J. W. Cooley)和图凯(J. W. Tukey)宣布它时，正是人们已经认为在实践中使计算成为不应有的浪费之时。从那时起，人们发现这一基本思路在许多文章里，包括高斯(C. F. Gauss)的工作，被具体化了。

基本思想是源于这样的事实：$\exp(a+b)=\exp(a)\exp(b)$ 和 $\exp(2\pi i)=1$。同时运用上述事实时，就发现计算 $a[n]$ 时 N^2 项中许多项是相同的。如果我们能灵活地计算，使这些项只计算一次，就能大大地节约时间。

定义 $q=\exp(iWt)$，在弗兰德和其他一些人的著作里全面讨论的一种方法是将(1)中的项对 $f(m\ dt)$ 按 q 的奇次幂和偶次幂分开。偶次幂的和是 $a[0],a[2],\cdots\cdots$ 的傅里叶变换，奇次幂的和是 $a[1],a[3],\cdots\cdots$ 乘以 q^m 的傅里叶变换。若 $N=2k$，则每一个和含 k 个项。然而只需计算这许多和中的一半，因为对任意一个 m 与对 $m'=m+k$ 的和是相同的从 $q^k=1$ 可以得出。而这来自假设有 N 个点在傅里叶展开式中具有相同个数的正频率与负频率，因此可以砍掉一半的浮点运算。若 N 为 2 的方幂，可以递归地分解，每次用一个因子去砍掉一半的工作。花费时间与 $N\log N$ 成正比，当 N 增加时，$N\log N$ 的增加极为缓慢，这就使含上千数据点的计算十分合算。

前面一段描述了"基 2"FFT。还有许多可能的变形，它们无需点数 N 为 2 的方幂。

另一种方法是将 FFT 作为矩阵的计算。考虑 $N=4$ 的 FFT，由定义它是：

$$\begin{bmatrix}a[0]\\a[1]\\a[2]\\a[3]\end{bmatrix}=\begin{bmatrix}q^0&q^0&q^0&q^0\\q^0&q^1&q^2&q^3\\q^0&q^2&q^4&q^6\\q^0&q^3&q^6&q^9\end{bmatrix}\begin{bmatrix}f(0)\\f(dt)\\f(2dt)\\f(3dt)\end{bmatrix}$$

应用关系式 $q^4=1$，则有

$$\begin{bmatrix}a[0]\\a[1]\\a[2]\\a[3]\end{bmatrix}=\begin{bmatrix}1&1&1&1\\1&q&-1&-q\\1&-1&1&-1\\1&-q&-1&q\end{bmatrix}\begin{bmatrix}f(0)\\f(dt)\\f(2dt)\\f(3dt)\end{bmatrix}$$

交换上面第二、三行，则可将矩阵分解为两个矩阵的乘积

$$\begin{bmatrix}a[0]\\a[2]\\a[1]\\a[3]\end{bmatrix}=\begin{bmatrix}1&1&0&0\\1&-1&0&0\\0&0&1&q\\0&0&1&-q\end{bmatrix}\begin{bmatrix}1&0&1&1\\0&1&0&1\\1&0&-1&0\\0&1&0&-1\end{bmatrix}\begin{bmatrix}f(0)\\f(dt)\\f(2dt)\\f(3dt)\end{bmatrix}$$

最初的矩阵需要 16 个复数相乘，而新矩阵中只要求含复变数 q 的两个(尽管需要加

上与−1相乘)。这是以重新排列向量a所得到的，由此得到向量的变址所希望的重新排列可以由a的址数二进制码的逆序来实现。因此例子中0与3保持不变，而2(二进制中为10)与1(二进制中为01)互换位置。这是因为一个"完全搅混"是由分离和数中的奇偶项来执行的，而每个一半又再次被搅混。

§10.4.1 潜藏着的危险：FFT给出什么回答

下面的测试程序产生出衰减指数函数$f(t)=\exp(-t)$的FFT。设置的$f(0)=0.5$不是1，原因是$f(t)$应该是一个周期函数。在$f(0)$的值应该和在$f(T)$的值相同，因为后者本质上为零，在$f(0)$的值应为0和1的平均数0.5。这个最好的点并不具有决定性的意义。

现在考虑回答。$f(t)$的傅里叶变换应该是$1/(1+j\omega)$，当ω增加时$f(t)$单调减少。但是FFT结果中的实数部分减少到中点然后又增加。负频率成分也在数组中，而且它们出现的次序可能令人惊奇。重要的最小频率出现在数组的两端，并向中心增加。

如果有一个频域函数希望转换回时域，必须"折迭"数组的中间部分，并对$n<N/2$嵌入虚数部分的符号，否则就得不到解答中的实函数(参看§10.4.4的实函数和表因函数)。得到FFT与傅里叶变换一致的数值，还必须用已经转换的时间周期来尺度化(即乘上该时间周期)。相似地，逆变换必须用各点间增加的频率来尺度化。

§10.4.2 潜藏着的危险：替换(时域)

事实上替换涉及到不同的傅里叶分量。当周期性地抽样时，可能出现其它频率，故可以从其它分量实际的上升来给出频率分量的上升。

如果用FFT分析光学系统，例如光圈，用上面提到的函数周期性，分量已经转换了。实际上FFT是槽缝的无限周期数组(事实上就是衍射光栅)，效果例示在图10—1中。如果槽缝每个"象"的衍射模式有一个与一个重叠现象发生，其高频成分(表示为衍射模式的"尾")与相邻模式重叠，结果进入个别槽缝的模式边界会产生乱真的影响。

纠正这个问题就须设置适当的使$f(t)$为零的"保护"带，在保护带的边缘变换实际的函数$f(t)$，图象变动到重叠可忽略不计为止。当然，需要多少保护单元得视问题而定，但常常是有确定意义的数目，如"实"数据值的一半。FFT应用于非周期函数的卷积或相关(包括自相关)函数这类目的时也必须用相似的保护。

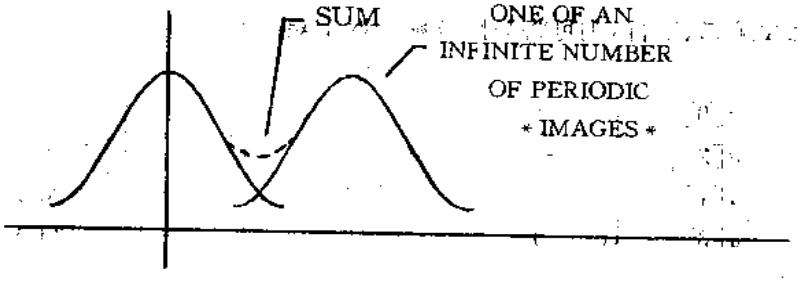


图10—1 替换如何发生

§10.4.3 潜藏着的危险：替换(频域)

替换时可能在计算中引入了误差，以致使处理变得困难。如设具有非零分量$\sin(5\omega t)$的函数在进行傅里叶变换，但只在$4\omega t$的周期处用抽样点，因为$\sin(5\omega t)=\sin(4\omega t)\cos(\omega t)+\cos(4\omega t)\sin(\omega t)$，而在这些抽样点处$\sin(4\omega t)=0$，$\cos(4\omega t)=1$，所以$\sin(5\omega t)=\sin(\omega t)$，即为形如$A\sin(5\omega t)$或$A\sin(\omega t)$的函数。由于产生同样的影响，所以这些函数是不可辨别的，仅就在4ω频率处抽样来说是这样的。于是在5ω处没能再解出频率，将产生在ω处的表现信号。

事实上这是在"时域"替换下讨论过的同样的替换机制，图10—1中的"图象"也适用于频域。现在周期为$T=\pi/\omega$，其中ω为FFT的频率分辨率。在表达式中少了一个因子2，原因是信号的最大频率分量的频率要求抽样两次，使能得到相和幅角的信息(即从余弦推算出正弦)，这被称之为"奈奎斯特(Nyguist)频率"。

频域替换问题有两种可能的处理方法。第一种同时域替换一样，移动图象直到重叠部分可被忽略时为止。第二种是对于频域替换还要求在奈奎斯特频率N上或者在超过奈奎斯特频率的频率上进行抽样，这样可以防止浪费；或者假如数据已经取出并作为既成事实提出了分析，在这种情况下可以使用反替换滤波。移去危险的高频分量这种想法当然很简单，那些分量对实用地分析没有足够的分辨率，但是可能导致出错。当已经取了数据(即在系统中嵌入了低通滤波)或去掉麻烦的频率在数据上可作光滑处理，如果有足够的分辨率也还是可以作的。如果在测量本身过程中发生替换，以后纠正则是困难的。

§10.4.4 实函数与表因函数

当已知被变换的函数的特征时，在不同的条件下都可能得到方便。例如，被变换的函数是实函数，就可以用"快速哈特勒(Hartley)变换"(FHT)去掉一半计算量。然而有的作者使用的正规化约定在这里已经与在FFT中所用的不相同，所以用尺度变换更容易比较结果。

若$f(t)$是实函数，其变换$F(\omega)$必须满足$F(-\omega)=F^*(\omega)$，这里$*$表示共轭复数。这就导致了抽样问题，对$F=R-iX$，得出$R(\omega)=R(-\omega)$和$X(\omega)=-X(-\omega)$。

表因函数。即对于$f(t)=0, t>0$，在信号处理中常常受到关注。"表因"一词表示$f(t)$事先已假设为实值的。在这样的情况下，F的实部与虚部就由希尔伯特变换来相互联系。物理学家可能承认希尔伯特变换，作为克莱姆——克罗利希(Kramers—Kronig)关系。但是通过希尔伯特变换，函数R和X之间的关系就极其复杂，增加了计算量反而不能减少给定R和X的观测值。人们可以检测一致性，然而一定要假设$f(t)$是表因函数。

常常看到$|F|$所确定的频谱丢弃了相信息，即丢弃了(对任意的ω)R和X间的关系。这样就不可能唯一地确定产生$|F|$的那个函数f，即使f是表因函数也是这样，然而有标准形式来构造出具有"博德极小相"的函数f。如果F是一个对电子网络的变换函数，则可证明网络的时域响应必须由极小相f给出。例如网络是梯形网络。

§10.5 程序和测试情况

程序用到了第七章的复计算包来计算 FFT，也计算了逆 FFT 以肯定计算的精度。用 C 语言中的移位运算来产生 $a[*]$ 的逆位变址是有利的。此外，指数函数之幂的计算（程序中的 ω，文本中的 q）放在辅助例行子程序中，因此不需要每次去计算。而逆变换所需要的幂向量仅仅是直接变换所用到的那些向量的复共扼。于是对于给定的 N，指数函数的赋值集合对一切 FFT 变换——直接的或求逆的都足够了。$\log N$ 的计算用整数算术来完成，而不是要求输入变量。注意如果进行直接的 FFT 或者逆 FFT，函数 fft( )并不"知道"。进行逆 FFT 时，应该用 $wi[\ ]$ 数组而不是 $w[\ ]$，并且对结果除以 N，这是通常的正规化约定。有时采用对直接变换与求逆变换两者的结果都除以 N 的平方根以求对称。

表 10-1 第十章的函数与子程序

Log2	计算一个整数的以 2 为底的对数(设结果取整数)
fftinit	FFT 初始化
bitr	对 FFT 实行逆位
fft	计算快速傅里叶变换
fhtinit	FHT 初始化
fht	计算快速哈特勒变换

§10.6 程序列表与测试问题输出

```
/* fast Fourier transform (FFT)

CONTENTS:

log2
  integer log to base 2 of an integer power of two
bitr
  do bit reversal for FFT array addressing
fftinit
  initialize arrays for given size (n) fft
fft
  perform FFT
printc
  print complex number
main
  test driver

DEPENDENCIES:
```

```
complex.h header file required
*/

#include "complex.h"
#define twopi 6.283185307

int iterp;/* global to return count used */

#define max(a,b) (((a)>(b))? (a): (b))
#define abs(x) ((x)? (x):-(x))
#define DOFOR(i,to) for(i=0;i<to;i++)

main(argc,argv) int argc;char **argv;
{int i,ii,nh,n=16;
double invn,exp(),dt=.25,omega,realpt,imagpt;
struct complex w[32],wi[32], data[32];
nh=n>>1;
DOFOR(i,n)
  {
  CMPLX(data[i], exp(-dt*(i)),0.);
  };
/* caveat */ data[0].x=.5;/* not 1. see text */
printf(" before transform:\n");
DOFOR(i,n){printc(&(data[i])) ;printf("\n");}
invn=1./n;
fftinit(w,wi,n);
/* (printf(" w factors:\n");
DOFOR(i,n){printc(&(w[i])) ;printc(&(wi[i])) ;printf("\n");}\*/
printf(" transformed:\n");
fft(data,w,n);
DOFOR(i,n){printc(&(data[i])) ;printf("\n");}
printf(" scaled by T=n*dt and compared to analytic answer:\n");
DOFOR(i,n)
  {
  ii= (i-nh)<0 ? i:i-n ;
  omega= twopi*ii/(n*dt);
  realpt= 1./(1.+omega*omega);
  imagpt= -realpt*omega;
  printf(" %f %f %f %f\n",dt*data[i].x,dt*data[i].y,realpt,imagpt);
  }
/* inverse FFT */
fft(data,wi,n);
printf(" transformed back and scaled by d(frequency)=1/n:\n");
DOFOR(i,n)
  {
  CTREAL(data[i],data[i],(invn));
  printc(&(data[i]));printf("\n");
```

```
    }
exit(0);
}

int bitr(k,logn) int k,logn;
{
int ans,j,i;
ans=0;
j=k;
DOFOR(i,logn)
    {
    ans=(ans<<1)+(j&1);
    j=j>>1;
    }
return(ans);
}

int log2(n) int n;
{
int i;
i=-1;/* will return -1 if n<=0 */
while(1)
    {
    if(n==0)break;
    n=n>>1;
    i++;
    }
return(i);
}

printc(x) struct complex *x;
{
printf("%f %f",x->x,x->y);
return;
}

fftinit(w,wi,n) int n; struct complex w[],wi[];
{
int i;
double realpt,imagpt,cos(),sin();
double factr,angle;
factr=twopi/n;
DOFOR(i,n)
    {angle=i*factr;
    realpt=cos(angle);imagpt=sin(angle);
    CMPLX(w[i],realpt,imagpt);
    CMPLX(wi[i],realpt,(imagpt));
```

```
  CMPLX(wi[i],realpt,(-imagpt));
  }
return;
}

fft(x,w,n) int n;struct complex w[],x[];
{
int n1,logn,i,j,k,l,i1,logl,exponent,p;
struct complex s,t,a,b;
logn=log2(n);
n1=n>>1;
j=logn-1;
/* transform */
k=0;
DOFOR(logl,logn)
  {
  do{
  DOFOR(i,n1)
    {
    p=bitr((k>>j),logn);
    l=k+n1;
      CONJG(s,w[p]);
      CMULT(t,s,x[l]);
      CSUB(x[l],x[k],t);
      CADD(x[k],t,x[k]);
      k++;
    };/* dofor 1 */
  k+=n1;
  }while (k<n);
k=0;
j--;
n1=n1>>1;
}

/*reorder */
for(i=1;i<n;i++)
  {
  k=bitr(i,logn);
  if (i>k)
    {
    /* exchange i,k elements */
    CLET(s,x[i]);
    CLET(x[i],x[k]);
    CLET(x[k],s);
    }
  };
return;
```

```
}
before transform:
0.500000 0.000000
0.778801 0.000000
0.606531 0.000000
0.472367 0.000000
0.367879 0.000000
0.286505 0.000000
0.223130 0.000000
0.173774 0.000000
0.135335 0.000000
0.105399 0.000000
0.082085 0.000000
0.063928 0.000000
0.049787 0.000000
0.038774 0.000000
0.030197 0.000000
0.023518 0.000000
transformed:
3.938010 0.000000
1.143903 -1.746777
0.373174 -1.070216
0.181974 -0.699026
0.111059 -0.475893
0.078526 -0.320685
0.062163 -0.199640
0.054256 -0.096066
0.051880 0.000000
0.054256 0.096066
0.062163 0.199640
0.078526 0.320685
0.111059 0.475893
0.181974 0.699026
0.373174 1.070216
1.143903 1.746777
scaled by T=n*dt and compared to analytic answer:
0.984503 0.000000 1.000000 -0.000000
0.285976 -0.436694 0.288400 -0.453018
0.093294 -0.267554 0.092000 -0.289025
0.045493 -0.174757 0.043091 -0.203062
0.027765 -0.118973 0.024705 -0.155223
0.019631 -0.080171 0.015953 -0.125293
0.015541 -0.049910 0.011133 -0.104922
0.013564 -0.024017 0.008203 -0.090200
0.012970 0.000000 0.006293 0.079077
0.013564 0.024017 0.008203 0.090200
0.015541 0.049910 0.011133 0.104922
```

0.019631 0.080171 0.015953 0.125293
0.027765 0.118973 0.024705 0.155223
0.045493 0.174757 0.043091 0.203062
0.093294 0.267554 0.092000 0.289025
0.285976 0.436694 0.288400 0.453018
transformed back and scaled by d(frequency)=1/n:
0.500000 -0.000000
0.778801 -0.000000
0.606531 -0.000000
0.472367 0.000000
0.367879 0.000000
0.286505 0.000000
0.223130 0.000000
0.173774 -0.000000
0.135335 0.000000
0.105399 0.000000
0.082085 0.000000
0.063928 -0.000000
0.049787 -0.000000
0.038774 -0.000000
0.030197 -0.000000
0.023518 -0.000000
/*
discrete Fast Hartley Transform
based upon BASIC code in 2nd revised edition of R. N. Bracewell's
"The fourier Transform and Its Applications" (McGrav-Hill, 1986)
various modifications have been made to reduce storage required
(transform is done in place, unlike Bracewell) and reduce
computation (e.g., cos and sin only computed for n/4 +1 values,)
not n as in Bracewell).

CONTENTS:

log2
  returns integer logarithm to base 2 of integer (power of two assumed)

fhtinit
  intializes arrays of sines and cosines needed for FHT

fht
  do FHT. fhtinit need be called oncefor multiple xfms of
  arrays if size unchanged.

main
  test driver.

DEPENDENCIES:

```
NONE
*/
#define min(a,b) (((a)<(b))? (a): (b))
#define max(a,b) (((a)>(b))? (a): (b))
#define abs(x) ((x)? (x):-(x))
#define DOFOR(i,to) for(i=0;i<to;i++)
#define INDEX(i,j) [j]
/* above def is same as for 1-d array on index j only */

main(argc,argv) int argc;char * *argv;
{int i,n=16,l,log2();
double invn,exp();
int m[256];
double dt=.25,f[256],r[256], w[256],wi[256],x[256];
n=8;/* bracewell testcase */
DOFOR(i,n){
  f[i]= i+1;
  };
DOFOR(i,n){printf(" input %e",f[i];printf("\n");}
fhtinit(w,wi,m,n,256);
fht(f,r,x,w,wi,m,n);
printf(" warning- FFT results unscaled by T-for ramp test case\n");
printf(" Fourier Tranform Hartley Transform\n(Real Pt.)  (Imaginary)\n");
DOFOR(i,n){printf(" %e %e %e",r[i],x[i],f[i]);printf("\n");}
n=16;
DOFOR(i,n){
  f[i]= exp(-dt * (i));
  };
f[0]=.5;
DOFOR(i,n){printf(" input %e\n",f[i] );}
fhtinit(w,wi,m,n,256);
fht(f,r,w,wi,m,n);
invn=16. * dt;/* scaling for FT */
printf(" decaying exponential; FT scaled\n");
printf(" fourier Tranform Hartley Transform\n(Real Pt.)  (Imaginary)\n");
DOFOR(i,n){ r[i] * =invn;x[i] * =invn;
      printf(" %e %e %e\n",r[i],x[i],f[i]);
      }
exit(0);
}

int log2(n) int n;
{
int i;
i=-1;/* will return -1 if n<=0 */
while(1)
  {
```

```
  if(n==0)break;
  n=n>>1;
  i++;
  }
return(i);
}

#define twopi 6.283185307

int cols,logn,n2,n4,logl;

fhtinit(c,s,m,n,columns) int n,m[ ],columns;
  double c[ ],s[ ];
{
int i,log2(),l;
double cos(),sin(),con,angl;
cols=columns;
con=twopi/n;
m[0]=1; logn=log2(n);logl=logn-1;
n2=n>>1;n4=n2>>1;
DOFOR(i,logn)
  {
    n[i+1]=(m[i]<<1;
  }
;
angl=0.;
DOFOR(i,n4+1)
  {c[i]=cos(angl);s[i]=sin(angl);
  angl+=con;
    }
return;
}

fht(f,r,x,c,s,m,n) int n,m[ ];
double x[ ],r[ ],c[ ],s[ ],f[ ];
{
double temp,temp2,temp3,temp4,temp1,invn;
int t,u,i,j,k,l,ll,d,e,ss,so,s2,exponent,p;
invn=1./n;
/*reorder,i.e.,permute */
if(logn>1)
{
j=-1;i=-1;
do  {
  do{
  i++;
  k=logn;
```

```
  do  {
        k--;
        j-=m[k];
    }while(j>=-1);
  j+=m[k+1];
  }while(i<=j);
  temp=f[i+1];
    f[i+1]= f[j+1];
    f[j+1]=temp;
  }while(i<n-3);
};
/* transform */
/* 2 element DHT unrolled inner loop */
for(i=0;i<=n-2;i+=2)
  {
  temp= f[i];
  temp2=f[i+1];
  f[i]=temp+temp2;
  f [i+1]= temp-temp2;
    }
i=1;
if(logn! =1)
{
i=2;
/* 4 element DHT unrolled inner loop */
for(i=0;i<=n-4;i+=4)
  {
  temp= f[i];
  temp2=f [i+2];
  temp3=f [i+3];
  temp1=f [i+1];
  f [i]=temp+temp2;
  f [i+1]= temp1+temp3;
  f [i+2]=temp-temp2;
  f [i+3]=temp1-temp3;
    }
/* remaining loop */
}
if(logn>2)
{
ss=4;
u=log1;
for(i=2;i<logn;i++)
  {
    s2=ss<<1;
    u--;
    so=m[u-1];
```

```
      for(k=0;k<n;k+=s2)
        {
        i=k;
        d=i+ss;
        temp= f[i];
        temp2=f [d];

        f [i]=temp+temp2;
        f [d]=temp-temp2;
        i1=d-1;
        for (j=so;j<=n4;j+=so)
          {
          i++;
          d=i+ss;
          e=i1+ss;
          temp= f[d]*c[j]+f[e]*s[j];
        temp2=f[d]*s[j]-f[e]*c[j];
        temp3=f[i];
        temp1=f[ i1];
        f[i]=temp3+temp;
        f[d]=temp3-temp;
          f[i1]=temp1+temp2;
        f[e]=temp1-temp2;
              i1--;
            }
          e=i1+ss;
        }
      ss=s2;
    }
  }
  f[0]*=invn;
  /*dft*/
  r[0]=f[0]/*+f INDEX(1,0)*/;
  x[0]=0.;
  for(i=1;i<=n2;i++)
    {
      trmp2=f[n-i]*invn;
      temp= f[i]*invn;
      r[i]= (temp2+temp)*.5;
      x[i]= (temp2-temp)*.5;
      f[n-i]=temp2;
      f[i]= temp;
    }
  /* reflect for other components of Fourier Transform */
  for(i=n2+1;i<n;i++)
    {
    j=n-i;
```

```
    r[i]=n[j];
    x[i]=-x[j];
    }
  return;
  }
  input 1.000000E0
  input 2.000000E0
  input 3.000000E0
  input 4.000000E0
  input 5.000000E0
  input 6.000000E0
  input 7.000000E0
  input 8.000000E0
  warning— FFT results unscaled by T for ramp test case
  Fourier Tranform Hartley Transform
  (Real Pt.)  (Imaginary)
  4.500000E0 0.000000E0 4.500000E0
  -5.000000E-1 1.207106E0 -1.707106E0
  -5.000000E-1 5.000000E-1 -1.000000E0
  -5.000000E-1 2.071067E-1 -7.071067E-1
  -5.000000E-1 0.000000E0 -5.000000E-1
  -5.000000E-1 -2.071067E-1 -2.928932E-1
  -5.000000E-1 -5.000000E-1 -2.244904E-11
  -5.000000E-1 -1.207106E0 7.071067E-1
  input 5.000000E-1
  input 7.788007E-1
  input 6.065306E-1
  input 4.723665E-1
  input 3.678794E-1
  input 2.865048E-1
  input 2.231301E-1
  input 1.737739E-1
  input 1.353352E-1
  input 1.053992E-1
  input 8.208499E-2
  input 6.392786E-2
  input 4.978706E-2
  input 3.877420E-2
  input 3.019738E-2
  input 2.351774E-2
  decaying exponential: FT scaled
  Fourier Tranform Hartley Transform
  (Real Pt.)  (Imaginary)
  9.845025E-1 0.000000E0 2.461256E-1
  2.859758E-1 -4.366942E-1 1.806675E-1
  9.329362E-2 -2.675540E-1 9.021190E-2
  4.549346E-2 -1.747565E-1 5.506250E-2
```

2.776464E−2 −1.189732E−1 3.668446E−2
1.963139E−2 −8.017116E−2 2.495063E−2
1.554076E−2 −4.990996E−2 1.626368E−2
1.296997E−2 0.000000E0 3.242492E−3
1.356402E−2 2.401650E−2 −2.613119E−3
1.554076E−2 4.990996E−2 −8.592298E−3
1.963139E−2 8.017116E−2 −1.513494E−2
2.776464E−2 1.189732E−1 −2.280214E−2
4.549346E−2 1.747565E−1 −3.231577E−2
9.329362E−2 2.675540E−1 −4.356509E−2
2.859758E−1 4.366942E−1 −3.767960E−2

第十一章　微分方程系统

§11.1　本章目的

这一章将给出下列C语言工具：作为初值问题(IVP)部分的常微分方程(ODE)积分系统的龙格——库塔——费尔堡(Runge－Kutta－Fehlberg)(RKF)方法的实现；RKF方法通用第五版的实现。

本章将讨论这些方法的理论基础和稳定性概念。下一章将看到稳定系统的另一个适合的积分程序。

§11.2　微分方程的数值处理

对工程师或自然科学家来说，解方程系统和微分方程系统几乎是最频繁的数值工作。严格地说，微分方程集合本身不能组成要求解的问题还必须对系统增加各种附加说明，通常称之为"边界条件"。

这一章和下章将在"初值问题"或IVP范围内讨论常微分方程系统的求解方法(这些方程具有单个独立变量)。下一章详细处理所谓"稳定"系统。

§11.3　初值问题是怎样发生的

"初值问题"术语中隐含着时间是独立变量的想法。在一个初值问题里，已知在最早的时刻，例如$t=0$时系统的状态，希望确定系统以后的发展。

通过引入新的因变量表示其它因变量的导数，常微分方程的任一系统可转变成一阶微分方程系统。例如调和振子方程(表示导数)

$$x''+x=0,$$

可写成系统

$$y=x',$$
$$y'+x=0.$$

假设把所有方程写成这种形式，那么就应该有由N个方程，N个因变量和N个初值条件组成的问题。如果我们不这样做，则会有太多或太少的事情，其结果是非唯一解或者完全无解。这一规则也有例外(若这些方程是非线性或具有奇异点)，一般规则总是需要方程个数，变量个数和初值条件个数相同。

IVP常常会发生，牛顿定律$F=ma$就是一个例子。如果假设给了力，限制运动为一维质点，则实际上是一个二阶微分方程。在三维中，三个分量都有相似的方程，单个质点已六

阶系统给出。含许多质点的系统只需对每一质点的加速度和速度写成系统：

$$a = F/m = V',$$

$$x' = V.$$

其中 x 为质点位置，V 是它的速度(如果可能的话可看作向量，或对 y 和 z 及对应的速度写出相似的方程)。这个系统也是轨道力学和天文动力学要解决的问题，电学装置(速调管，相对论性的二极管，等等)的质点码模拟中也用到。

IVP 还有其它的应用，如电路求解器，电压穿过导体可由一阶微分方程 $V = Li'$ 给出。电容器上电荷由 $q = CV$ 给出或用 $i = q'$，$i = CV'$ 的微分形式给出。这些关系是用克希霍夫定律得到描述电路的 ODE。解 ODE 系统时都把在 $t = 0$ 时作为 IVP，此时"开关"最初是关闭的，此时的电路指的是电源。

系统模拟不必限制物理系统如质点系统。用 ODE 模拟时，常常称此过程为"连续"模拟，相反为"离散事件"模拟。一般离散事件模拟应用于排队和清单目录这类模型，其中的事件就如象"到达的顾客"、"次序的调度"、"船队到达"等等。连续模型如流动过程的模拟，例如容器的泄漏或充满、伺服(随动)机构的动作，等等。

连续模拟也用于物理科学之外。例如，凯恩斯(Keynes)说的"边缘倾向"不是别的而是导数即变化率。经济模型中如"国家产品总变化率与投资率成比例"看作一个模型假设是常见的。如果引入关系"任何时候国家总的生产等于投资、消费及政府消耗的总和"，那么按此关系给出消费及政府消耗，就可以使经济模型转化为一个 ODE 系统。于是经济学家们不趋向于使用典型的数学术语如导数，他们常常讨论 ODE。人口模型是典型的 ODE 系统。

独立变量不一定需要时间。如在杠杆理论中独立变量可以是沿杆的纵坐标，因变量为在该点处的垂直位移。费曼(Feynman)曾对这一理论作了简洁的介绍，导出了近似方程：

$$Z'' = F/YI.$$

其中 Z 是挠度，F 是力，YI 是杠杆的"稳定性"。(I 是截面的惯量矩，Y 是杨(young)模量)。

若 ODE 和 IVP 包含因变量的线性式，则系统容易表示为形如 $d = Ax + F$ 的矩阵代数。其中 d 为导数向量，x 为因变量向量，A 是系数矩阵，F 为系统外力的表示项。初始条件可写成 $x[t=0] = x[0]$，$x[0]$是一个给定向量，这样写出系统就类似于前一节讨论 IVP 的关系。若令 $d = 0$，就有 $Ax + F = 0$，即第三章讨论过的代数系统。这表示了平衡问题中如果状态已经确定，在那些状态中系统保持不变，如果进一步设 $F = 0$，一般只有平凡解 $x = 0$，除非矩阵 A 是降秩的。若 A 的系数为参数的函数，可以确定参数的值，对于该值可能齐次问题有非平凡解——这就是特征值问题。若时间是独立变量，可以找到共振频率，在共振频率上，系统(在没有损耗时)将永远作"阻尼振荡"。杠杆理论的情况，一般对杆的曲度参数，这一特征值给出了一个判断值，即一点在无(附加)力时产生(任意大的)位移。

§ 11.4 IVP 的数值处理

在考虑了为什么要解 IVP 后，现在转向如何求解。有许多方法可供选择。本章将考虑

方法的一族，龙格—库塔—费尔堡公式，并集中在族的第四阶成员上。求解 IVP 的任意方法的选择基本方案是每一时间步函数估计值的个数与时间步长中允许完成了数目在所期望容限内平衡。要求一个方法能自动监控它完成，并按照时间步去掉或增加。RKF 族提供了近似的局部误差(定义如下)估计和按要求调整时间步的能力。这个方法的性质采用到问题中去是很有用的，同时在前面第九章讨论自适应求面积技术时已经遇到过。

§ 11.5 稳定性

有一个潜在的障碍物即所谓"稳定性"问题。术语"稳定"似乎来自稳定弹跳的概念，它对应于调和振子方程 $x''+Kx=0$(对此问题虎克定律为 $F=-(K)^{1/2}x$)的弹跳常数。事实上，大 K 对应于大的频率，即 x 的改变是短时间尺度的。稳定问题之所以成为问题不仅因为它们包含着短时间尺度，而且它们还包含着不可比的时间尺度。系统中一个方程可能有一变量在很短的时间尺度上改变，而其它变量发展得很慢。要得到解答最后的平衡状态，要求对这些较长时间尺度得到系统的积分，但这样做会受到快速反应的时间尺度的限制，必然将引起浪费。图 11—1 列出了困难的本质。考虑简单的一阶 IVP，$y'=-Ay$ $(A>0)$，$y(t=0)=1$，它的解为 $y=1-\exp(-At)$，这没有什么困难就可求解该方程。但是对于独立变量 t，用太大的步就会有麻烦的结果。如用简单的积分模式象尤拉方法，即写出

$$y(t+\mathrm{d}t)=y(t)-A\mathrm{d}ty(t)=Y(t)(1-A\mathrm{d}t),$$

就需要用 $A\mathrm{d}t$ 很小的值去得出精确解。若 $A\mathrm{d}t>1$，必须代替歧异的振荡。若 y 表示的是化学反应的集中或某些其它内在正量，这个过程就会变得极端麻烦化。若只有一个方程求解，则 $\mathrm{d}t$ 容易限制为 $1/A$。如果要精确描绘 y 随时间的变化，就应该保持 $\mathrm{d}t$ 的这个阶。若是一个方程的系统，要找出 y 的详细变化，必须先解此系统，而且 $\mathrm{d}t$ 必须生存在 $1/A$ 阶上。当研究许多化学反应的行为时，一定不要用快的反应时间步。对于在较长时间尺度上求解系统都能得到正确的结果时，在短时间尺度上反应物的详细变化就能可靠地近似。

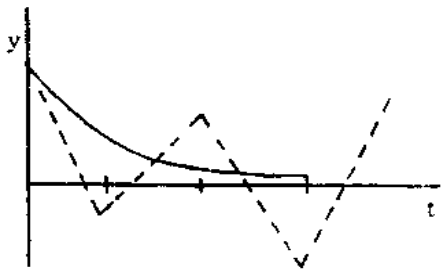


图 11—1 歧异数值振荡

围绕这个困难有许多解决方法，我们将在下一章里讨论。这里给出的方法在中等稳定问题上常常是十分好的，但在真正稳定问题上小时间步是很不经济的。如果发生了这种情况，对问题就试用下一章的例行子程序。稳定问题可以用许多方法产生。通常，化学反应系统有广泛全异的速率就会产生稳定问题，下面讨论的轨道动力学也可能有稳定问题发生。

§11.6 龙格——库塔方法

这一章只讨论处理IVP的简单"单一步"方法，下一章将讨论多步。测试程序即校正程序和多值方法，作为稳定系统要求理解方法的先决条件。单一步的好处之一是它们是"自开始的"，即在 $t=0$ 输出集不修改就可以使用来确定 $t=\mathrm{d}t$ 时的解。典型地，多步方法要求在许多时刻如 $t-\mathrm{d}t$ 的值，第一步时它们不存在。通常它们用简单的方法产生，列如在很小的时间步上用尤拉法(参看下面)。单步方法的另一特性是与多步方法相比不浪费时间步 $\mathrm{d}t$ 的变化。对同样的精度似乎已经得出了多步方法更有效的结论，对非稳定问题和中等精度的讨论表明：RKF 方法恰好是经济的而且是最方便的。

以下假设独立变量为时间，谈到的是时间步。对下列形式的微分方程：

$$y'=f(y,t)$$

(对一个微分方程系统，y 和 y' 可看作 N 维向量，这时 f 是函数向量)对计算开发合适的离散近似必须作出两个决策：

1)如何使导数 y' 模型化。

2)如何使 y 的"有效"平均值模型化。

因为一般会迁到非线性问题，其中函数 f 是因变量 y 的函数。这些方程的右端不能说明，且在确定 y' 时的误差将通过 f 后面的估值误差反馈给将来要确定的 y'，这会导至局部误差和整体误差的不同。局部误差是从 $y[t]$ 到 $y[t+\mathrm{d}t]$ 工作时产生的，假定已确切地知道 $y[t]$。一般它是截断误差组合成的，即由于 $\mathrm{d}t$ 有限性的近似误差与由于计算机算术有限精度的舍入误差组合而成。一般假设后者比前者小，因为局部误差不仅会累计，一般还会使 y 不精确。在 $y[t+\mathrm{d}t]$ 的计算值与 $y[t+\mathrm{d}t]$ 准确值之差这是整体误差，比另外希望的更坏。

最简单的方法是尤拉法，它可以认为是一阶龙格——库塔方法。对赋予的时间步 $\mathrm{d}t$，y' 取作 $(y[t+\mathrm{d}t]-y[t])/\mathrm{d}t$，$f=f(y[t],t)$，其中 t 和 $y[t]$ 已知。很容易求解未知量

$$y[t+\mathrm{d}t]=y[t]+\mathrm{d}t f(y[t],t).$$

此法的基本问题是得出了可接受的精度，一般 $\mathrm{d}t$ 必须很小，从函数 f 的许多估值中得到。作出的近似对 $\mathrm{d}t$ 仅仅是一阶的，即对 $\mathrm{d}t$ 是二阶的可达到希望的局部误差。对 $\mathrm{d}t$ 为一阶的这些局部误差累计为整体截断误差。以下将尝试在近似公式中得到更高阶的精度，通过用较大的时间步会经济地改进结果，改进的结果与依赖于 f 的估值有关。每一时间步将作多次 f 的估值，但是同一结束时间达到相似精度要求较少时间步是问题所在。

龙格——库塔方法的一般处理是看作形如：

$$y[t+\mathrm{d}t]=y[t]+\mathrm{d}t\{a\ f(y[t],t)+b\ f(y[t]+c*\mathrm{d}t f(y[t],t)+\mathrm{d}d t)+\cdots\cdots\}$$

的公式。它可以重写成

$$y1=f(y[t],t)\mathrm{d}t,$$

$$y2=f(y[t]+cy1,t+\mathrm{d}t),$$

$$y[t+\mathrm{d}t]=y[t]+\mathrm{d}t\{ay1+by2+\cdots\}.$$

在这个最简公式中必须作两个函数 $y1,y2$ 的估值，要选择四个常数 a,b,c,d，每一时

间步对每一因变量作函数估计。若将 $y[t+dt]$ 关于 $y[t]$ 展开成泰勒级数，与 dt 之幂匹配就得到三个限制：$a+b=1, bc=bd=1/2$。根据它们的选择可以对其它参数一直作选择，首先是爱德华·费尔堡对这些常数的最优确定付出了很多的努力，其次他还提出了 $y[t+dt]$ 计算的两种估计如何更经济些，这些对(局部)误差的估计是可以比较的。例如通常的教科书都选择

$$a=b=1/2, \qquad c=d=1.$$

这样总截断误差为 $(y1-y2)dt/2$ 阶，对小 dt，差 $y1-y2$ 希望为 dt 阶，故为二阶精度。费尔堡推导出的类似公式(用三个函数估值)有 $(y3-y1)/512$ 阶误差，于是大约作 256 次较好(此时虽然对附加函数估计有花费)，多数情况这是有利的交换。

费尔堡对这种情况给出了公式两次估计 $y[t+dt]$：

$$y1=f(y[t],t),$$
$$y2=f(y[t]+y1/2,t+dt/2),$$
$$y3=f(y[t]+y1/256+y2\ 255/256),$$
$$y[t+dt]=y[t]+dt\{y1/512+y2\ 255/256+y3/512\},$$

又
$$y[t+dt]=y[t]+dt\{y1/512+y2\ 255/256\}.$$

这些公式方便之处如下：

1) $y[t+dt]$ 的第二个公式用第一个再用函数估值，故 f 没有"浪费的"估计。

2) 第二个公式中的系数 1/256 和 255/256 相加为 1，与 $y3$ 估计的系数一样，这是费尔堡巧妙的设计，用此结果下一步不需要再估计新的 $y1=f(t=[t'+dt], t'=t+dt)$ 只需从前一时间步取 $y3$ 的结果。因此，对大量的时间步，只存放每一时间步的函数估值！费尔堡用同样的技巧于他所谓的二阶和三阶公式。

实际上并不是对 $y[t+dt]$ 的两种公式估计，而是仅仅用到后者(重新应用上面已作描述)估值，对 $y[t+dt]$ 估计两种"说法"的差。

费尔堡的四阶公式我们将予以实现，不是从前一步再用存放的函数估计这种装置。一般高阶公式出现被认为是阶(每一时间步函数估值个数)和允许的时间步大小间最好的折衷。函数 f 不论是极其简单还是复杂，大概应分别使用高或低阶方法。多数稳定问题大概用高阶公式更好，虽然也可用下一章真正稳定问题的多值方法。

四阶公式为：

$$y1=f(y[t],t),$$
$$y2=f(y[t]-dt\,y1/4,t+dt/4),$$
$$y[t+dt]-y[t]=dt\{\cdots\cdots\}$$

其中 $\{\cdots\cdots\}$ 为：

$16/135\ y1+6656/12845\ 3\ y3+28561/56430\ y4-9/50\ y5+2/255\ y6$ 或 $25/216\ y1+1408/2565\ y3+2197/4104\ y4-1/5\ y5$，就给出形为

$dt\{-1/360\ y1+128/4275\ y3+2197/5240\ y4--y5/50-2/55\ y6\}$ 的局部误差项。

$y[t+dt]$ 的第一个公式有第五阶的阶局部误差，后者有第四阶局部误差，给出第五阶精度(局部误差)方法，费尔堡称之为 RK4(5)，但它涉及到 RKF45，是代替总体精度的四阶方法。对库塔和萨拉费恩(Sarafyan)的结果的误差项进行比较是有利的。库塔方法为

$$y1=f(y[t],t),$$
$$y2=f(y[t]+y1\,dt/2,t+dt/2),$$
$$y3=f(y[t]+y2\,dt/2,t+dt/2),$$
$$y4=f(y[t]+dt\,y3,t+dt),$$
$$y[t+dt]-y[t]=dt/6(y1+2y2+2y3+y4).$$

它有 1 和 2 阶量级中较大者的误差项。

如何控制步长？假定已经得到了准确的解，"最简单地"与用户指定的容限比较误差，可用时间步再试而结束。若必须删去许多时间步，就断定系统是稳定的。

§11.6.1 高阶和低阶公式

费尔堡给出了 RK3(4)(也称为 RKF34)方法：

$$y1=f(y[t],t),$$
$$y2=f(y[t]+2/7\,dt\,y1,t+2/7\,dt),$$
$$y3=f(y[t]+dt(77/900\,y1+343/900\,y2),t+7/15\,dt),$$
$$y4=f(y[t]+dt(805/1444\,y1-77175/54872\,y2+97125/54872\,y3),t+35/38\,dt),$$
$$y5=f(y[t]+dt(79/490\,y1+2175/3626\,y3+2166/9065\,y4,t+dt),$$
$$y[t+dt]-y[t]=dt(\cdots\cdots).$$

其中(……)为：

$229/1470\,y1+1125/1813\,y3+13718/81585\,y4+y5/18$ 或 $79/490\,y1+2.75/3626\,y3+2166/9065\,y4$.

注意，再用 f 最后的计算值对下一步作为 $y1$ 的第一个值。同样，RK2(3)或 RK23 方法为：

$$y1=f(y[t],t),$$
$$y2=f(y[t]+dt/4y1,t+dt/4),$$
$$y3=f(y[t]+dt(-189/800\,y1+729/800\,y2),t+27/40\,dt),$$
$$y4=f(y[t]+dt(214/891\,y1+y2/33+650/891\,y3),t+dt),$$
$$y[t+dt]\quad y[t]=dt(\cdots\cdots).$$

其中(……)为：

$533/2106\,y1+800/1053\,y3-y4/78$ 或 $214/891\,y1+y2/33+650/891\,y3$.

可以在 1970 年计算杂志第六卷 61 页费尔堡的论文中找到高阶方法，后来维纳(J. H. Verner)发展了高阶方法。1986 年汤玛斯(B. Thomas)提出了他对五阶和七阶的表形式方法。汤玛斯的文章里有一些印刷错误。RK4(5)方法中的系数应为 28561/56430，错印成 285161/56430。图 3 的第 8 步列出对 $x=0$ 应完成运算，事实是对 $x\neq 0$ 应完成计算。表 7 和表 8 中的负号应省去。

§11.7 程序和测试情况

RKF45 程序有很长的历史。桑迪亚(Sandia)实验室的莎平(L. Shampine)和瓦茨(H.

A. Watts)开发和实现了 FORTRAN 程序，那是试验和真实的实现。我们是模块化作的，RK4(5)模式说明的码，我们从例行子程序的主体中减去了一些东西。用于其它阶的模式时全局整数参数的阶和全局系数数组必须作改变。当改变时间步估计作到的改变有多大要用到参数的阶，这是以 RKF 模式阶的理论误差性质为基础的。此外，低阶模式中一步最后函数的估计存放作为下一步第一个估值，输入变量 iflag 可以用来"记住"该值是否存放或是否再计算(它应送到子例行子程序 feval 的位置)。函数的最后值在工作数组里，对 i=0,……,(neq−1)编址来数组 ffINDEC(5,i,neq)有 neq 个方程。

rkf 的自变量表中的参数为：

neq	系统中要求解方程的个数。
y	因变量(一个向量)。
t	浮点变量指针；输入：时间步开始的时刻；输出：rkf 到达的时刻。
tout	浮点变量指针；输入：时间步结束时希望的时刻。
relerr,abserr	因变量中误差容限；码尝试保持 y[i]局部误差(按绝对值大小)小于 relerr * y[i]+abserr。
flag	整数指针；输入：问题开始时设为＋1；迫使码尝试从 t 到 tout 在单步中积分时设为－1； 输出： 2 正常步， －2 接续的单步， 3 误差容限太小，relerr 增加， 4 需要超过 3000 个导数估值；积分不能成功地完成， 5 因为 abserr＝0 而 relerr 太小积分不能完成；不改变 abaserr 就用一步方法， 6 因为最小步长不能作到足够的精度积分不能完成；增加误差容限， 7 rkf 用于问题可能无效——用一步模式， 8 无效输入参数——纠正它们。
work,iwork	程序用到的工作数组；对 rkf，iwork 至少必须为 5 维，work 至少必须为 3＋(阶＋2) * neq。
f	形如 $f(t,x,x\mathrm{dot})$的函数指针，它对系统方程的右端返还一个大小为 neq 的向量；即 $x[]$的时间导数。

iflag 中的某些误差码要求加以讨论。若 iflag＝4 或 7，对问题来说 rkf 大概不适合，实际误差容限可能设置得太低。若 iflag 为 5 或 6 问题就严重了，必须改变误差容限或者改用其它方法。若这些之一发生特别是 iflag＝6 对求解程序来说可能问题太稳定，若子例行子程序 rkf 在 iflag＝3 或 4 时返还有值再调用它且不改变 iflag 会使程序继续运行。否则，若 iflag 为 5、6 或 7 在 rkf 再调用之前 iflag 必须重置为 2 或者 rkf 中止。调用例行子程序时显然对误差条件必须检验 iflag。

在 rkf 中"最小可能步长"按用的算术精度设置数目，在时间步开始时乘上时间的绝

对值。显然多少这是一个"有意义的规则",因为可以只移动时间原点就能够改变这个最小步。若 iflag＝6 返还发生了虽然不是问题稳定就是得到的解有奇异性,仔细观察结果认为当前状态是正确的。相似地,iflag 为 4 返还时是基于经验的"有意义的规则"。如果你相信你有更好的东西改变或忽略这些判别法,但不能不同于新期望的。抽样测试问题使用户在用户指定弹性常数的调和振子与适度地稳定问题之间进行选择,适度地稳定问题是吉尔(Gear)、福瑟西、马尔柯姆和莫勒等人讨论过的。这个问题有如右端的因变量的两个线性组合。解是衰减(负)指数之和,$x[1]=4\exp(-t)-3\exp(-1000t)$,$x[2]=-2\exp(-t)+3\exp(-1000t)$。在稳定问题中的结果对解的两个分量用 1000 的因子除时间尺度。在我们的测试中,对这个问题 RKF45 作得特别地好。下一章将用到同样的测试问题来测试我们的稳定方法,为了比较数值结果主程序将打印出准确的解答。

对稳定问题只有吉尔提出了一个适当的求解程序。福瑟西等人对轨道力学问题作了测试。这种多体问题,每一质点与其它质点有交互引力作用,可以很容易变成稳定问题。质点中两个作准确的逼近发生了什么就变成了一个棘手的有界二元系统且绕它们的共同质心接着发生旋转。这个系统的两个成员的轨道周期作为一个整体可以比系统发展的时间尺度短很多。处理这样的稳定系统其方法之一是识别此二元系统,用系统中的对象质量之和作为一个对象的质量以之作简单的代替。从多体系统的观点看是模拟,这些结合对象的引力场几乎恰好等于二元系统的引力场(如果圆周运动很快,它自己的场作平均与单个点的场紧密近似)。当它分解时可以得到系统最后紧密逼近的结果。

除非用户特别要求程序去作(问题一开始就设置 iflag＝－1,对其它步允许 iflag＝－2),时间步 $dt=tout-t$ 时码不是简单地作一个单步,一般若它感到需要这样作就将试更小的步。所以不与有另外的码的 rkf() 比较结果,而是简单地说明 tout－t 这个差并与之比较,唯一最好的比较是使用计算机时间和达到相似精度的结果间进行比较。当这样做时会看到在稳定问题上下一章的求解程序 stiff() 比 rkf() 更好。

作为另一个测试问题,提出简单调和振子发现这个选择的例子很好。

在 Kaypro I (Z80 处理器)上使用 AztecC 和在 IBM PC 上使用 AztecC86 成功地处理了稳定问题,无论是否使用 8087 数学协处理芯片。对各种情形 u26 和 eps 的值都不同,反映了不同浮点的精度。表 11－1 列出了本章所包含的程序,不要求其它章的例行子程序或标头——这些码是"独立的"。

表 11－1　第十一章的函数与子程序

rkf	龙格——库塔——费尔堡积分程序的体
error	RKF 方法计算的误差项
feval	RKF 方法计算的项
sign	FORTRAN 的相似符号转换
isign	对整数的 FORTRAN 的相似符号转换

§11.8 程序列表与测试问题输出

```
/*
Runge—Kutta—Fehlberg integrator for
  system of Ordinary Differential Equations
  (Initial Value Problems)non—stiff to mildly stiff
  general order;RKF45 i.e. RK4(5) parameters supplied
(from "C Tools for Scientists and Engineers" by L. Baker)

CONTENTS:

rkf()
  Runge=Kutta—Fehlberg integrator. General order.
  (see text for qualifications of this statement)

feval()
  evaluates steps
error()
  evaluates error term

sign() similar to FORTRAN routine for sign transfer
isign() ditto

DEPENDENCIES:
ftoc.h header file
*/
#include "ftoc.h"
/* #include "math.h" */
#define labs(x) (((x)>=0)? (x):-(x))
/* C ordering but with k */
#define INDEC(i,j,k)[j+(1) * (k)]
#————global variables to remember eps,u26,etc between calls of rkf */
float u26,eps,crit,criti;
double order,iorder;

/* The constant order is the only scheme dependent parameter
in RKF;other parameters are globals used by error and feval
the functions to evalute the error terms and the intermediate
derivatives. order should be set to m=n+1,e.g. 5 for RKF45 */
```

```
int norder=5;

/* subroutine rkf for general Runge—Kutta—Fehlberg integrations */

rkf(neq,y,t,tout,relerr,abserr,iflag,work,iwork,f)
int iwork[],neq,*iflag;int (*f)();
float *t,*tout,y[],abserr,relerr,work[];
/* NB that relerr and abserr are float variables,t and tout pointers to float variables.
 */
/* iwork must be dimensioned at least 5,work at least 3+7*neq for rkf45 */
/* generally work must be 3+neq*(2+norder)norder=5 for rkf45 */
{
int base,*nfe,*kop,*init,*kflag,*jflag,hfaild,output,k,maxnfe=3000;
int i,mflag,gflag,coln;
float *h,*yp,*ss,s,*ff,*savre,*savae,a,dt,ee
,eeoet,esttol,et,hmin,remin=1.e-12,rer,scale,tol,epspl,ypk;
/* FORTRAN code had two unused variables;ae and toln */
float sign(),error();
double pow();
/*assign to work—this will enable different ODE's to be solved
simultaneously by passing in different work,iwork vectors */

coln=neq;
/* set pointers to work areas */
nfe=iwork;
base=1;
kop=&(iwork[base++]);
init=&(iwork[base++]);
kflag=&(iwork[base++]);
jflag=&(iwork[base++]);
base=neq*(1+norder);
yp=work;
ff=work;
ss=&(work[base]);
base+=neq;
h=&(work[base++]);
savre=&(work[base++]);
savae=&(work[base]);
/* valid input? */
mflag=iabs(*iflag);

if(neq<1 || relerr<0. || abserr<0.
```

```
 ‖ mflag==0‖mflag>8‖((*t==*tout) && (*kflag!=3)))
        {*iflag=8;
        return;}
 gflag=0;
 if (mflag==1)
   {/*first call set precision limit*/
   eps=1.;
   while(1)
     {
     eps=eps*.5;
     epsp1=eps+1.;
     if(epsp1<=1.)break;
     }
   u26=26.*eps;
   /*printf(" u26,eps %e %e\n",u26,eps);8087;eps=5.96e-8*/
 /* now set the limits to stepsize change—these depend upon
 the parameter order,and are larger for larger orders */
   order=(double)norder;
   iorder=1./(order);
   crit=pow(9.,order);
   criti=pow(.9/order),order);/* make sure 5 here is order */
   /* crit=59059., criti=1.889568e-4 for rkf45 */
   }
 else
   {

     if(*iflag==3‖(mflag==2 &&(*init==0‖*kflag==3)))
       {gflag=1;goto nest;}
     if(*iflag==4)‖(*kflag==4&&mflag==2))
       {*nfe=0;if(mflag!=2)gflag=1;goto next;};
     if((*kflag==5 && abserr==0.)‖
     (*kflag==6 && relerr<*savre && abserr<savae))
           {/* unrecoverable serious pblm */
           printf("iflag=5,6,7,8 nto properly handled by user\n");
         *iflag=9;return;
         }
   };
 next:
 if(gflag){*iflag=*jflag;if(*kflag==3)mflag=iabs(*iflag);}

 *jflag=*iflag;
 *kflag=0;
```

```
 *savre=relerr;
 *savae=abserr;
rer=2. *eps+remin;
if(relerr<rer)
    {
    relerr=rer;
     *iflag= *kflag=3;
    return;
    };
gflag=0;
dt= *tout- *t;
if(mflag==1)
   {/* initialize */
   *init=0;
   *kop=0;
   a= *t;
   gflag=1;
   ( *f)(a,y,yp);
   *nfe=1;
   if( *t== *tout){ *iflag=2;return;}
   };/* mflag==1 */
if( *init==0||gflag)
     {
     *init=1;
     *h=abs(dt);
     /*toln=0.;in FORTRAN version but not used */
     DOFOR(k,neq)
     {
     tol=relerr *abs(y[k])+abserr;
     if(tol>0.)
         {
         /*toln=tol;in FORTRAN but not used */
         ypk=abs(yp[k]);
         if(ypk * pow((double) *h,order)>tol) *h=pow((tol/ypk),lorder);
         }/*if tol */
     } /*do */
  if(tol<=0.) *h=0.;
  ypk=max(abs(dt),abs( *t));
  *h=max( *h,u26 *ypk);
  *jflag=isign(2, *iflag);
  }/* if int ==0 */
```

```
*h=sign(*h,dt);
/* too many output requests? */
if(bas(*h)>=2.*abs(dt)) (*kop)++;
if(*kop==100){*kop=0;*iflag=7;return;}
if(abs(dt)<=u26* abs(*t))
    {
    DOFOR(k,neq)y[k]+=dt*yp[k];
    a=*tout;
    (*f)(a,y,yp);
    (*nfe)++;
    *t=*tout;
    *iflag=2;
    return;
    }
output=0;
scale=2./relerr;
/*ae=scale*abserr;not used-in FORTRAN*/
while(1)/* infinite loop over steps */
{
hfaild=0;
hmin=u26*abs(*t);
dt=*tout-*t;
if(abs(dt)<2*abs(*h))
    {
    if(abs(dt)<=abs(*h)){output=1;*h=dt;}
    else *h=.5*dt;
    }

if(*nfe>maxnfe)
    {
    *iflag=*kflag=4;return;
    };
step:
feval(f,neq,y,*t,*h,ff,ss);
DOFOR(i,neq)ff INDEX(1,i)=ss[i];
(*nfe)+=norder;
/* error ok? */
eeoet=0.;
DOFOR(k,neq)
  {
  et=abs(y[k])+abs(ff INDEX(1,k));
  if(et<=0.){*iflag=5;return;}
```

```
    ee=abs(error(k,ff,coln));
    eeoet=max(eeoet,ee/et);
    };
esttol=abs(*h)*eeoet*scale;
if(esttol>1.)
    {/*step failed*/
    hfaild=1;
    output=0;
    s=.1;
    if(esttol<crit)s=.9/pow((double)esttol,iorder);
    *h=s**h;
    if(abs(*h)>=hmin)goto step;
    *iflag=*kflag=6;
    return;
    };
/*successful step*/
(*t)+=*h;
DOFOR(k,neq)y[k]=ff INDEX(1,k);
a=*t;
(*f)(a,y,yp);
(*nfe)++;
s=5.;
/* power is .2=1/5 for 4th-5th order scheme
crit=9**(1/power)criti=(.9/5)**(1/power)*/
if(esttol>criti) s=.9/pow((double)esttol,iorder);
if(hfaild)s=min(1.,s);
*h=sign(max(hmin,s*abs(*h)),*h);
if(output){*t=*tout;*iflag=2;return;};
if(*iflag<=0){*iflag=-2;return;}
};/* end while loop--loop back for next step */
}

/* coefficients for RKF45 4-5th order scheme */

double e[6]={0.0027777777,0.0,-0.0299415,-0.0291998,0.02
                                    ,0.036363636};
double a[6]={0.,.25,.375,.9230769,1.,.5};
double c[6]={.1157407,.0,.5489278,.5353313,-.2,0.};
double b[6][5]={0.,0.,0.,0.,0.,.25,0.,0.,0.,0.,
.09375,.28125,0.,0.,0.,.8793809,-3.2771961,3.3208921,0.,0.,
2.0324074,-8.,7.1734892,-.2058966,0.,
-.29629629629,2.,-1.3816764,.4529727,-.275};
```

```
/* note that the zeros in b are merely placeholders to simplify indexing
   —not used in calculations */

float error(k,ff,coln)
int k,coln;float *ff;{
int l;
double sum;
sum=0.;
DOFOR(j,(norder+1))sum+=e[j]* ff INDEX(j,k);
return(sum);
}
/* compute terms in RKF steps */

feval(f,neq,y,t,h,ff,s)
int neq;float y[],t,h,ff[],s[];
int (*f)();
{
int i,j,m;
double x;
DFOR(j,2,(norder+1))
  {
  DOFOR(i,neq)
    {x=0.;
    DOFOR(m,j)
      x+=b[j][m]* ff INDEC(m,i,neq);
    s[i]=x*h+y[i];
    }
  x=t+a[j]*h;
  (*f)(x,s,&(ff INDEC(j,0,neq)));
  }
DOFOR(i,neq)
  {
  x=0.;/* note that c[5]=0. */
  DOFOR(j,norder)x+=c[j]* ff INDEC(j,i,neq);
  s[i]=h*x+y[i];
  }
return;
}

isign(to,from)int to,from;
{
int sign;
```

```
if (from>=0)sign=1;
if (from<0)sign=-1;
return(abs(to) * sign);
}

float sign (to,from)float to,from;
{
float sin;
if (from>=0)sin=1.;
if (from<0)sin=-1.;
return(abs(to) * sin);
}
/* test driver for

   RKF45 Runga-Kutta-Fehlberg integrator for
   system of Ordinary Differential Equations
   (Initial Value Problems)
   non-stiff to mildly stiff
(from "C Tools for Scientists and Engineers" by L. Baker)

DEPENDENCIES:
ftoc.h header file required
/*
#include "libc.h"
#include "math.h"
*/
#include "ftoc.h"

int problem;float ksq;

main (argc,argv)int argc;char * *argv;
{
double sin(),cos(),exp(),sqrt();
float a,b,time,x[8],tout,t,u,v;
int f();float abserr,relerr,work[2000],dt;
int i,j,iwork[25],ibug,neq,iter;
x[0]=x[1]=1.;t=0.;
ibug=1;neq=2;
for (i=0;i<1000;i++)
abserr=.001;relerr=.01;
printf("enter dt");scanf("%f",&dt);
printf("enter 0 for harmonic osc 1 for stiff\n");
```

```
scanf("%d",&problem);
if(!problem)
    {printf("enter square of spring constant\n");
    scanf("%f",&ksq);
    x[1]=0.;
    a=sqrt((double)ksq);
    }
printf("Time x1(Calculated) x1(Exact) x2(Calculated) x2(Exact)\n");
for(i=0;i<100;i++)
  {
  tout=t+dt;neq=2;
  rkf(neq,x,&t,&tout,relerr,abserr,&ibug,work,lwork,f);
  if(ibug>2){printf("ibug=%d\n",ibug);exit(0);};
  time=t;
  t=tout;/*bumpt t*/
  if ((i% 10)==0 ||(i<5))
    {
  if(problem)
    {
    a=exp(-time);b=exp(-1000.*time);
      u=4.*a-3.*b;
      v=-2*a+3.*b;
      }
    else
      {
      b=time*a;
      u=cos((double)b);
      v=-a*sin((double)b);
        }
    printf("%f %f %f %f\n",time,x[0],u,x[1],v);
    };
  };
exit(0);
}

/*-------right-hand sides of ODE'S----------------------*/

f(t,x,xdot)float t,x[],xdot[];
{float u,v;
u=x[0];
v=x[1];
if(problem==0)
```

```
    {
    xdot[0]=v;
    xdot[1]=-ksq*u;
    }
 else
    {
    xdot[0]=998.*u+1998.*v;
    xdot[1]=-999.*u-1999.*v;
    }
 return;
 }


 enter dt .05
 enter 0 for harmonic osc 1 for stiff
0
 enter square of spring constant
1.
 Time x1(Calculated) x1(Exact) x2(Calculated) X2(Exact)
 0.050000 0.998750 0.998750 -0.049979 -0.049979
 0.100000 0.995004 0.995004 -0.099833 -0.099833
 0.150000 0.988771 0.988771 -0.149438 -0.149438
 0.200000 0.980067 0.980067 -0.198669 -0.198669
 0.250000 0.968912 0.968912 -0.247404 -0.247404
 0.550000 0.852525 0.852524 -0.522687 -0.522687
 1.050000 0.497571 0.497571 -0.867423 -0.867423
 1.550000 0.020795 0.020795 -0.999784 -0.999784
 2.049999 -0.461071 -0.461072 -0.887363 -0.887363
 2.549999 -0.830052 -0.830053 -0.557685 -0.557685
 3.049998 -0.995808 -0.995808 -0.091467 -0.091466
 3.549998 -0.917755 -0.917755 0.397145 0.397146
 4.049997 -0.615005 -0.615004 0.788523 0.788524
 4.549999 -0.161678 -0.161677 0.986843 0.986844
 enter dt .01
 enter 0 for harmonic osc 1 for stiff
1
 Time x1(Calculated) x1(Exact) x2(Calulated)X2(Exact)
 0.010000 3.959537 3.960063 -1.979437 -1.979963
 0.020000 3.920065 3.920795 -1.959667 -1.960397
 0.030000 3.885781 3.881782 -1.944893 -1.940891
 0.040000 3.843771 3.843158 -1.922191 -1.921579
 0.050000 3.805195 3.804918 -1.902736 -1.902459
```

```
 0.110000 3.574208 3.583337 -1.782539 -1.791668
 0.210000 3.249930 3.242337 -1.628761 -1.621168
 0.310000 2.952858 2.933788 -1.485963 -1.466894
 0.410000 2.657840 2.654601 -1.330539 -1.327301
 0.510000 2.401938 2.401983 -1.200946 -1.200991
 0.610000 2.164270 2.173404 -1.077568 -1.086702
 0.710000 1.966569 1.966578 -0.983280 -0.983289
 0.810000 1.779166 1.779433 -0.889449 -0.889717
 0.909999 1.607824 1.610098 -0.802776 -0.805049
 enter dt .02
 enter 0 for harmonic osc 1 for stiff
1
 Time x1(Calculated) x1(Exact) x2(Calculated) X2(Exact)
 0.020000 3.921461 3.920795 -1.961064 -1.960397
 0.040000 3.857060 3.843158 -1.935482 -1.921579
 0.060000 3.766207 3.767058 -1.882678 -1.883529
 0.080000 3.692584 3.692465 -1.846350 -1.846233
 0.100000 3.619435 3.619350 -1.809760 -1.809675
 0.220000 3.205029 3.210075 -1.599991 -1.605083
 0.420000 2.628188 2.628187 -1.314094 -1.314094
 0.620000 2.151779 2.151778 -1.075889 -1.075889
 0.820000 1.761728 1.761727 -0.880864 -1.880864
 1.020000 1.442381 1.442380 -0.721191 -0.721190
 1.219999 1.180922 1.180921 -0.590461 -0.590461
 1.419999 0.966857 0.966857 -0.483429 -0.483428
 1.619999 0.791596 0.791596 -0.395798 -0.395798
 1.819999 0.648104 0.648104 -0.324052 -0.324052
```

第十二章　微分方程的刚性组(Stiff Systems)

§12.1　本章目的

在这一章你将找到以下工具：有效地求刚性的普通微分方程系统(ODEs)的积分。这将使包含这样的系统的初始值问题的解法成为可能。

我们将讨论解刚性微分方程组的隐含方法的一般思想，这些方法对于处理解微分方程组具有重要意义。

§12.2　什么是 ODEs 的刚性组

你在阅读这个子标题时，请你阅读一下第十一章中关于刚性的那一节。在这一章，我们将考虑怎样解含有这样系统的初始值问题(IVPS)。如果你对在这章中任何地方出现的术语感到陌生，那么请重新阅读前面一章，这将对你有所帮助。

§12.3　对于刚性组富有启发性的方法

在前一章，我们讨论了引力相互作用的点的质量的多体问题。怎样才可能把这样一个系统变成既刚性，又简单固定的系统，这样做的基础是：

1)认识这个系统的什么部分是"刚性"，也就是说，按一短的时间比例，系统的这部分相对于系统的所有其它各部分成比例的变化。

2)专门探讨刚性系统，为了从总体上决定刚性系统部分在整个系统中的作用，因而取通过那个短的时间刻度的平均数。这样做，忽略了系统的时间相依性。

考虑刚性系统的其它普通类型，如一组化学反应。一个化学反应的反应速度比另外的化学反应更快。在已知的某个时刻，化学反应的种类的浓度导数根据反应达到平衡能很快确定，反应平衡是由于有"更慢的"种类参加的原故。因此，通过令这个导数为零和解这个系统的余部，通常用一个代数方程去代替刚性方程是可能的。把这种思想转 换成一种算法的过程中，存在许多困难。首先，程序必须要决定哪些方程应该被看成是刚性的和给出这样的特殊处理。其次，我们必须要解代数方程以及 ODEs 的余下部分，而 ODEs 的余部可能不易解。事实上，对于解非线性方程组的一些方法也可用来处理象 ODEs 的方程组，即非线性方程 $f=0$ 可以看作 $de/dt=f$ 来解，其中 e 是一虚设的误差项，并且我们试图对状态 $de/dt=e=0$ 的初始猜测规定的初始值问题求积分。而这种方法的作用强烈地依赖于 ODEs 的性质以及将刚性项转换成代数方程后是否容易解。

这样的启发性方法可能不具有一般用途，并且我们将不再进一步考虑它们。

§12.4 多值方法

C. W. Gear 使用术语"多值法"来表示系统中每个相关变量使用了不止一个值的方法。通常，这些方法使用定义于前面时间步的值。在这种情形下，这些方法被称之为多步方法。一个例子是 Adams－Bashbord 方法：

$$y[t+dt]=y[t]+dt/24\{55f[t]-59f[t-dt]+37f[t-2dt]-2dt]-9f[t-3dt]\}.$$

以下通过假定 dt 的固定值让我们来简化上面式子中的表示法，t 表示第 0 步，$t-dt$ 表示第－1 步，等等。通过 f 由－3 到 0 的四个值，这个公式靠一适当的三次多项式给出积分上下限分别是 1 和 0 和外推三次多项式的积分。虽然，这个方法给出了同 dt 的五次方成比例的截断误差，但是它在每一时间的步长值只需用一函数来赋值，并把这个方法同 RKF45 相比，而 RKF45 需要六个函数赋值。可是，由于误差项比例于 y 的第五阶导数，因此它可能很大。实际上，Adams－Bashford 法靠利用获得的 $y[1]$值在一额外函数赋值的花费方面已被改进，这个 $y[1]$值并不是"最终的"值，只不过作为 $y[1]$的"预报值"或估计值。接着，这个 $y[1]$值根据 f 从－2 到 1 的四个数值得到的一适当三次方程的结果被修正。我们便可用 y_1 的预报值来估计 f。那么修正后的 y 为：

$$y[1]=y[0]+dt/24\{9f[1]+19f[0]-5f[-1]+f[-2]\}.$$

如果我们在这一点终止，我们就得到了众所周知的"艾德姆——莫尔顿(Adams－Moulton)法"修正公式。如果 $y[1]$的预报值和修正值间的差大得不能接受，人们能用它来代替迭代法，这样的预报和修正方法是很经济的，对于大多数问题它比单步法更可取。在龙格——库塔(Runge－Kutta)法与费黑尔贝格(Fehlberg)和弗纳(Verner)的简化误差的形式以前，以上提到多值法在开始时有困难。这在第十一章中已经提到过，例如我们计算时，在 $t=0$，并没有相应的 $y[-1]$，等等。

我们所考虑的方法到目前已变得清起来，即，我们已经知道计算 f 的值能需要的全部 y 值。在 Adams－Moulton 方法中，$y[1]$的值被预报者提供。对于刚性系统，解这个系统希望使用一隐含的方法。也就是说，用一个能把 y 在 1 的导数写成一置后差公式的形式，这个置后差公式包含了 y 在 1 和以前多步的值。在 Adams－Moulton 法的修正步骤，是解这个自相容的方程，而不是使用预测值(注意上面提到的迭代是这种方法的一粗糙形式)。因为 f 一般是 y 的非线性函数，所以这将涉及到解一非线性代数方程。可是，我们应该对 y 在某一点的那个值有一好的猜测，比如，对 $y[1]$的预测值。对于求这个非线性方程的解，不用简单迭代，而是用一更迅速收敛的方法，比如在第六章讨论的牛顿——辛普森迭代，这样的方法能够被指定为具有 A－稳定性的性质(参考第十一章)。这种方法意味着如果眼下只占据带有衰减指数项(不增长)的解，那么离散化的方程的解将类似地衰减。Gear 已经创立了这样的方法，并且他的 DIFSUB(用 FORTRAN 语言编写)程序可以在本书中找到。作为多值法，这些方法不是自起动的。通常这些方法的实现(执行)是通过使用一简单的自起动方案来模拟一自动代码。可进行多值法的时间步长(进程)为一充分大的数以前，最初几乎没有信号(具有一很小的 dt)。

Dahlquist 已经证明(参考第七章)用一种能获得解的稳定性的方法，我们为此不得不

以解的精确性为代价。在这个意义下，对于差分公式的项数来说能够得到的具有最高阶截短误差的方法将不是稳定的。因此，牛顿——科茨(Newton－Cotes)类型公式的隐含方法的作用效果可能比某些类似于高斯求面积的求解方法更好，特别对于刚性问题，更是如此。

§12.5 隐含方法和稳定性

为了阐明稳定性和隐含方法之间的关系，我们考虑具有初始条件 $y[t=0]=1$ 的普通微分方程(ODE)$y'=-ky$ 的初始值(IVP)问题。它的精确解为 $y=\exp(-t)$。首先，我们考虑最简单的方法，这就是欧拉方法：

$$\frac{y[t+dt]-y[t]}{dt}=-ky[t].$$

这个方程可以解，结果得到 $y[t+dt]=y[t]\{1-kdt\}=y[t]\{1-a\}$，其中 $a=kdt$。注意对于 $a>1$ 时，关于 y 的解及时从一个时间步长摆动到另一时间步长。对 $a>2$ 时，以上微分方程的所有解将是不稳定的。这个解是"超稳定性"的。这个解摆动于有越来越大的振幅的显示器符号中。虽然看起来好象这种方法能识别这个标记的宽度并且试图使这个解反回到零，但这个方法可能弄巧成拙而具有越来越坏的作用。现在考虑最简单的隐含方法：

$$\frac{y[t+dt]-y[t]}{dt}=-\frac{k\{y[t]+y[t+dt]\}}{2}.$$

如果引进 $b=kdt/2$，那么我们就得到 $y[t+dt]=y[t]\{(1-b)/(1+b)\}$。我们现在得到具有 $b>1$ 记号的一个振荡解，这个解对应于 $a>2$，因此我们借助于两个时间步长的一个因子，已经推延了这个问题。然而，这个解永远不会增加振幅。当然，如果 kdt 很大，且这个解具恰当的振幅，那么这解将不会衰减。用一隐含方法，虽然不能保证解的精确性，但是它会极其稳定。

§12.6 程序和测试情况

我们引进的解刚性系统的程序是基于 R. K. Brayton、F. G. Gustavson 和 G. D. Hachtel 算法，"一种解微分代数系统的新的有效方法是使用隐含置后微分公式法"。* 这些作者都说他们的方法优于 Gear 法，如果时间幅度 dt 变化迅速。

Brayton 等人的论文没有包括一代码列举。这儿的程序是基于由 W. R. Zimmerman 博士给出的 FROTRAN 程序来实现。在他的程序实现中，这个方法具有固定的阶，即第二阶。在 Brayton 等人和 Gear 著作中，不仅这个时间幅度而且相应的指令都是自动改变的。在代码的复杂性和超越方面，对增加时间步长的可能性之间有一权衡，因此这儿的程序是更有效。

代码输入参数如下：

* Proc. IEEE, Vol. 60, p. 88, 1972.

t	大小为 4 的向量，它的最后一个元素 $t[3]$ 应是当前时间。
x	是一相关变量值的矩阵，它的第一个指标——变量数。
dektmn	关于输入：指针指向希望的时间步长 dt。关于输出：指针返回用过的 dt。
delt	指针指向可允许的时间步长 dt 的最小值。
delst	指针指向要使用的 dt 后，开始积分(应该比希望的通常步长 dt 要小)。
deltman	指针指向允许的时间步长 dt 的最大值。
eman	指针指向用于时间步长控制的误差标准；如果 $dx[i]/(\text{abs}(x[i]+\text{deltax}))$(对每一个 $i=0,\cdots\cdots,\text{neq}-1$ 求值)的最大绝对值小于被估计的最大值(eman)那么代码将接受一步。
deltax	指针指向用于上面收敛检验的变量，如果 $x[i]$ 可能是 0，那么它可用于防止除法中除数为 0 的情形出现。
deltbx	类似于防止除数为 0 的不合逻辑的除法出现的 deltax，它也使指针指向要用的变量；至于 deltbx，它是关于 $x[i]$ 的数值微分，$x[i]$ 可根据牛顿——辛普森迭代，使用雅可比(Jacobian)矩阵来确定(见第六章)。
acc	对每一次牛顿——辛普森迭代，指向能限制一个向量的最大值的小数变化的向量。deltax 是每一次用来防止用 0 作除数的假除法出现；虽然在代码中使用的值是保守的，但这个值的增加或者减小可能实用。
pivot	在牛顿——辛普森解法中(这个解法使用了第二章中的线性系统解法)必须至少取 neq 个数用作工作空间的整数阵列。
work	工作空间阵列，它的大小必须至少取 $max(3,\text{neq})^3+3\text{neq}$。

来自于 Zimmerman 的 FORTRAN 程序实现，有一些较小的变化。类似于在 rkf()中使用过的标准，一整体的误差标准可转化为更熟悉的局部误差限制。另外，对于误差标准的 delt 而言，步长接受标准和这个变化有更大的效率而应稍稍改变。最后，一些较少的细致改进被引入代码中。

由于 Gear、Forsythe 和 press 等人的讨论，检验问题与第十一章的 stiff 问题相同。注意到机械地比较 stiff2()和 rkf()的结果将是不合理的，虽然后者可能被使用。

作为它的时间步长查询——t，可能是具有更小步长的"子循环"。真正的比较应是对解达到的精确程度(不仅仅是相似的误差标准)与计算机花费的时间的比较。

因为这种方法是多步和不能自起动，所以对于起初的很少的几步包含了一种简单的欧拉步，这最初几步欧拉步，用作它们的时间值 deltst。这个值应该不大，它应在指定的最小时间步长值附近或者可能更小。这个严格解确实在具有衰减解的问题中，显得好象有"自调整"性质。因为即使由起初的欧拉步长显得不精确，但之后的时间步长值恢复到具有某种精确性。可是，这不能作一般的保证。另外的自起动方法，比如 RKF 反演可能用于代替上面多步长方法。

表 12—1 列出了解程序(The solver packaye)的子程序。函数 pown()仅仅用于提供一个数的最小的积分指数。

表 12—1 第十二章中的程序

stiff2	求普通微分方程(ODE)初始值(IVP)系统的积分
pown	为 0 和 2 间的 n 提供 x^n

§12.7 程序列表与测试问题输出

```
/*
/*
stiff IVP—ODE solver
based on method Brayton,Gustavson,Hachtel as implemented by Zimmerman second—order method
only
(from "C Tools for Scientists and Engineers" by L. Baker)

DEPENDENCIES:
ftoc.h header file required
uses the linear system solver from Chapter 3
*/
#include "ftoc.h"
#define INDEC(i,j,col)[j+(i)*col]
main (argc,argv)int argc;char **argv;
{
double exp();
float a,b,time,x[8],t[4],delt,deltmn,deltmx,u,v,emax;
int f();
static float deltax=1.e-6,deltbx=.01,acc=.001;
float deltst,work[100];
int i,j,pivot[3],ibug,coln=4,iter;
x INDEX(0,3)=x INDEX(1,3)=1.;time=0.;iter=1;
ibug=0;
printf(" enter ibug>0 for debug info from stiff2\n");
scanf("%d",&ibug);
printf("ibug=%d\n",ibug);
t[0]=t[1]=t[2]=t[3]=0.0;
printf("enter delt,deltmn,deltmx,deltst,emax\n");
scanf("%f %f %f %f %f",&delt,&deltmn,&deltmx,&deltst,&emax);
printf("delt %f min %f max %f st %femax %f\n",delt,deltmn,deltmx,deltst,emax);
printf("Time x1(Catculated) x1(Exact) x2(Calculated) x2(Exact) dt\n");
for (i=0;i<200;i++)
{
```

```
t[3]=time;
/* printf(" calling stiff2 %f %f %f %f\n",delt,deltmn,deltmx,emax); */
stiff2(1,x,&iter,2,&delt,&deltmn,&deltst,&deltmx,&emax,&ibug,f,deltax,deltbx,acc,pivot,
work);
time=t[3];
if (ibug>0)printf("WARNING—ibug=%d\n",ibug);
if (ibug==5)ibug=0;/* reset */
if ((i % 10)==0 || (i<5)){
  a=exp((double)-time);b=exp((double)-1000. * time);
  u=4. * a-3. * b;
  v=-2. * a+3. * b;
printf(" %f %f %f %f %f %f\n",
time,u,x INDEX(0,3),v,x INDEX(1,3),delt);
  };
};
exit(0);
}

f(x,t,xdot,neq)int neq;float t,x[],xdot[];
{float u,v;int coln=4;
u=x INDEX(0,3);
v=x INDEX(1,3);
xdot[0]=998. * u+1998. * v;
xdot[1]=-999. * u-1999. * v;
return;
}

stiff2(t,x,iter,neq,delt,deltmn,deltst,deltmax,emax,ibug,f,deltax,deltbx,acc,pivot,work)
/* pivot must be dimensioned at least neq,work max(3,neq) * * 2+3neq;
see text for details of other input parameters
note that t is a vector of length 4 with t[3] the current time
and x is a matrix of the dependent variables;x[i][3] is the
i-th dependent variable at the current time,both on entry and exit
*/
int pivot[],neq,*ibug,*iter;int (*f) ();
float t[],x[],*delt,*deltmn,*deltst,*deltmax,*emax,
                                    deltax,deltbx,acc,work[];
{
int base,nlist=1000,norder=2,ierr1=0,newton=20,
                    iredmx=10,ierr=0,isflag=0;
int coln,nn,ired,i,j,k,npow,iflag,info;
/* will modify so that a,xdot,xdot1,xp,pivot point to sections of a single work array passed in */
folat *a,alpha[3],sum,gamma[3],*xdot,*xdot1,*xp,pown(),delx,erm;
folat w1,detold,detnew,w2,w3,w4,ww5,ddx;
float dum;float xnorm;int lufact(),backsub();
coln=3;a=work;base=max(3,neq);base=base * base;
xdot=&(work[base]);base+=neq;
```

```
xdot1=&(work[base]);base+=neq;
xp=&(work[base]);
/*shift arrays*/
if (*delt<=0.){
        *delt=*deltmn;
       printf("bad delt input to stiff2,reset to %f\n",*delt");
            };
if(*emax<=0.)
   {
   printf("bad emax=%f\n",*emax);
   return;
   }
for (i=0;i<neq;i++)
    {
    x INDEC(i,0,4)=xINDEC(i,1,4);
    x INDEC(i,1,4)=x INDEC(i,2,4);
    x INDEC(i,2,4)=x INDEC(i,3,4);
    };
t[0]=t[1];t[1]=t[2];t[2]=t[3];

if( (*iter-norder)<=1)
    {/* start by Euler steps at (small) delst */
    *delt=*deltst;
    t[3]=t[2]+*delt;

    (*f)(x,t[3],xdot,neq);
    for (i=0;i<neq;i++)x INDEC(i,3,4)=x INDEC(i,2,4)+*delt*xdot[i];
    (*iter)++;
/*detold just for diagnostics printed by stiff2*/
  detold=*delt;
  return;
  }

if((*iter-norder)>=2)
    {
redo:for(ired=0;ired<iredmx;ired++)
     {
     if((*ibug)>0){printf("ired %d delt %f ibug %d\n",ired,*delt,*ibug);};
     t[3]=t[2]+*delt;
     /* predictor */
     nn=norder+1;
     coln=3;
     for (K=0;k<nn;K++)
         {
         gamma[k]=0.;
         if(!k)gamma[k]=1.;
         dum=((t[3]-t[2-k])/(*delt));
```

```
/* pulled out of loop—dum is independent of i */
          for (i=0;i<nn;i++)
               {
               a INDEX(i,k)=pown(dum,i);
               };
          }/* for k */;
          lufact(a,coln,nn,pivot,&info);
          if(info) {printf("bad matrix 1in stiff2\n");
                    *ibug=1;return;};

            backsub(a,coln,nn,pivot,gamma);
          for (k=0;k<neq;k++)
              {
              sum=0.;
              for (i=0;i<nn;i++)
                  {
                  sum+=gamma[i]*xINDEC(k,2-i,4);
                  }
              xp[k]=sum;
              x INDEC(k,3,4)=xp[k];
              } /* for k */
          /* prepare for iteration */
          nn=norder+1;
          for (k=0;k<nn;k++)
              {
              alpha[k]=0.;
              if(k==1)alpha[k]=1.;
dum=((t[3]-t[3-k]) /(*delt));
/*similarly moved out of loop */
   for (i=0;i<nn;i++)
               {
            a INDEX(i,k)=pown(dum,i);
            };
   }/*for k */;

            lufact(a,coln,nn,pivot,&info);
            if(info){printf(" bad matrix 2in stiff2\n");
              *ibug=2;return;};

            backsub(a,coln,nn,pivot,alpha);
            coln=neq;

   /* Newton=Raphson iteration */
   for (j=0;j<newton;j++)
      {/*zerom(a,neq);not needed */
      (*f)(x,t[3],xdif,neq);
```

```
        for(k=0,k<neq;k++)
        {
        w4=x INDEC(k,3,4);
        ddx=deltbx * w4;
        x INDEC(k,3,4)=w4-ddx;
        (*f)(x,t[3],xdot1,neq);
        x INDEC(k,3,4)=w4;
        ww5=0.;
        if (ddx!=0.)ww5= *delt/(ddx * alpha[0]);
        for(i=0;i<neq;i++)
           {
           w2=0.0
           if(i==k)w2=1.;
           w3=ww5 * (xdost[i]-xdot1[i]);
               w4=w2+w3;
          /* a=1+Jacobian */
          /* no need to zero a,or add to itself */
          a INDEX(i,k)=w4;
          }
        }/* end k */
    /* rhs vector calc */
    for (k=0;k<neq;k++)
        {
        sum=0.;
        for(i=1;i<nn;i++)
        sum+=alpha[i] * x INDEC(k,3-i,4);
        xdot[k]=-x INDEC(k,3,4)
        -(sum+ * delt * xdot[k])/alpha[0];
        }/*end k */
    lufact(a,coln,neq,pivot,&info);
if(info){printf("bad matrix 3 in stiff2\n");
    *ibug=4;return;};

        backsub(a,coln,neq,pivot,xdot);
    /*advance state vector */
    delx=0.;
    xnorm=0.;
    ddx=0.;
    iflag=0;
for (i=0;i<neq;i++)
       {
       ddx=abs(xdot[i]);
       xnorm=abs(x INDEC(i,3,4))+deltax;
       w1=ddx/xnorm;
       if(w1>acc)iflag=1;
       x INDEC(i,3,4)+=xdot[i];
       if(w1>delx)delx=w1;
```

```
      }/*i*/
     if((*ibug)>0)
     {printf(" j%d,delx %f acc %f ibug %d\n",j,delx,acc,*ibug);};
     if(iflag!=1)goto convg;
     /* else continue iteration */
     }/* for j Newton-Raphson */
  /* no convergence in NR- cut delt and repeat or exit */
  if( *delt<= *deltmn) {isflag=1;
       if((*ibug)>0)printf("dt<dtmin at",t[3]);
                       }
   *delt= *delt*.25;
  if(isflag>=1)goto goodbye;
   *delt= max(*delt,*deltmn);
};/* end iteration ired */
  /* warn about outerloop nonconvergence */
  printf(" outer loop stiff2 nonconvrg\n");
convg:
      erm=0.;
      for (i=0;i<neq;i++)
          {
          ddx=abs(xp[i]-x INDEC(i,3,4));
          xnorm=abs(x INDEC(i,3,4))+deltax;
          w1=ddx/(xnorm*(t[3]-t[2-norder]));
          if(w1>erm)erm=w1;
          }
/* erm is now (x[predictor]-x[corrector])/(t-t)/x-norm;
compared to Henrici local trunc.error,need to mult by dt and xnorm-this is a relative local trunc
error if we mult by dt */
  erm=erm*(*delt);
    if(erm>=.25* *emax && erm<= *emax*4.)goto goobaye;
    if(erm < *emax*.25) {*delt=2.* *delt;/* was 4. */
          *delt=min(*delt,*deltmax);
          /* alternatively,set iflag-rather
              than exit */
              goto goodbye;};
          if (erm> *emax*4.)
              {if(*delt<= *deltmn)goto goodbye;
               *delt= *delt*.5;
              }
           *delt=max(*delt,*deltmn);
           *delt=min(*delt,*deltmax);
    if((*ibug)>0)printf("redoing in stiff2 error=%f %f\n",erm,*emax);
       goto redo;
goodbye:
      w1= *deltmax;
      if(erm>0.)w1= *delt* *emax/erm;
      detnew=min(w1,(2.* *delt));
```

```
        *delt=detnew;
        *delt=max(*delt,*deltmn);
        *delt=min(*delt,*deltmax);
(*iter)++; }/*end of if iter-norder>=2(90 in FTN)*/
detold=*delt;
return;
}

float pown(x,n)int n;double x;
{float ans,power;
if (!n)return (1.);
if(n==1)return(x);
power=x*x;
/* as only need up to n=2,will not do binary decomp of n*/
if(n==2)return(power);
if(n==3)retunr(power*x);
printf(" pown n=%d>3\n",n);
return(1.);/* should never get here*/
}
zerom(a,n,m)int n,m;double a[];
{int i,k;
k=n*m;
for(i=0;i<k;i++) a[i]=0.0;
return;
}
       stfode
enter ibug>0 for debug info from stiff2
0
ibug=0
enter delt,deltmn,deltmx,deltst,emax
.01 .001 .1 .001 .1
delt 0.010000 min 0.001000 max 0.100000 st 0.001000emax 0.100000
Time x1(Calculated) x1(Exact) x2(Calculated) x2(Exact) dt
0.001000 2.892364 3.996000 -0.894363 -1.998000 0.001000
0.002000 3.586002 3.992004 -1.589998 -1.996002 0.001000
0.003000 3.838657 3.988012 -1.844648 -1.994006 0.001000
0.004000 3.929085 3.984025 -1.937069 -1.992013 0.004000
0.008000 3.967121 3.968120 -1.983057 -1.984060 0.016000
0.488000 2.455411 2.452420 -1.227706 -1.226210 0.100000
1.488000 0.903295 0.898926 -0.451648 -0.449463 0.100000
2.448000 0.332304 0.329496 -0.166152 -0.164748 0.100000
3.487999 0.122248 0.120775 -0.061124 -0.060388 0.100000
4.487998 0.044937 0.044269 -0.022486 -0.022135 0.100000
5.487997 0.016544 0.016227 -0.008272 -0.008113 0.100000
6.487996 0.006086 0.005948 -0.003043 -0.002974 0.100000
7.487995 0.002239 0.002180 -0.001120 -0.001090 0.100000
8.487996 0.000824 0.000799 -0.000412 -0.000400 0.100000
```

9. 488000 0. 000303 0. 000293 −0. 000152 −0. 000146 0. 100000
10. 488004 0. 000111 0. 000107 −0. 000056 −0. 000054 0. 100000
11. 488008 0. 000041 0. 000039 −0. 000021 −0. 000020 0. 100000
12. 488011 0. 000015 0. 000014 −0. 000008 −0. 000007 0. 100000
13. 488015 0. 000006 0. 000005 −0. 000003 −0. 000003 0. 100000
14. 488019 0. 000002 0. 000002 −0. 000001 −0. 000001 0. 100000
15. 488023 0. 000001 0. 000000 −0. 000000 −0. 000000 0. 100000
16. 488026 0. 000000 0. 000000 −0. 000000 −0. 000000 0. 100000
17. 488029 0. 000000 0. 000000 −0. 000000 −0. 000000 0. 100000
18. 488033 0. 000000 0. 000000 −0. 000000 −0. 000000 0. 100000

C:\FOR

第十三章　描绘离散图和函数

"计算的目的在于获得对计算的深刻认识，而不在于数字本身，"——(R. 哈姆明(Hamming))。

"所有计算机都有长于生成许多数的能力，而短于对所产生的数的深刻认识。"——(J. 贝克(Baker))。

§13.1　本章目的

在这一章里，我们将介绍出现在本书中的既最短，又最有用的程序之一。由于提供说明有一定的困难，我们借助于"一图顶千字"的格言，为计算机用户提供所要处理的问题的说明。这个说明用一个图来理解较之一张包含许多数的表格让人感到更容易。

§13.2　迅速和涂改绘图(Quick and Dirty Plots)

这个程序是特意安排的，但它并不复杂。我们希望此程序能尽可能移植，在大多数终端屏幕或者宽行打印机上呈现可阅读的图。我们希望用这个程序既能描绘连续函数、又能描绘离散图中的所有点。我们想要清楚地显示出数据的间隔，而不是提供一些似乎合理，但有可能出错的一些内推程序。对于原始数据的分析是必须的。对于出版质量图，你将不得不把这个程序的输出作为草图或者使用更复杂的程序得到其它图形(见以下的讨论)。

§13.3　程序和测试情况

检验驱动器描绘一条 $y=x^2$ 的简单抛物线，函数 splot()是绘图程序。这个方程中的两个变量都是浮点阵列 X[]和 Y[]，这两个阵列表示了要描绘的点(或者函数)以及同阵列提供的点数。在 X[]和 Y[]中，前者是横轴或者横坐标，后者是竖轴或者纵坐标。为了适合大多数终端屏幕和宽行打印机，程序限制每横行的实际长度为打印 78 个字符的长度(否则，屏幕在每行的第八十个字符会自动"重叠"，我们有时将得到"扭曲的"图形。)。同理，图的高度限制为 20 行。首先，在检测处理中的常量函数之后，用绘图程序 splot()来绘制这个图形。然后，逐行打印这个图形。这样，可能出现在任何已知行中的阵列 Y[]中的点被查询，也就是说，在阵列 Y[]中的最上面的 Y 变量和最后一个 Y 变量之间的所有值被查询。在阵列 Y[]中或者充分靠近最小值的值被描绘在横坐标上("on"the abscissa)。

在图的竖限制下，辅助程序 setl()可用于制表。子程序 plotl()输出一单独行。子程序 unclob()确保空(NULL)字符被打印成空白；在不同的终端屏幕，这些程序可以做不同的事情(在正常运算下，空字符不应该出现在要绘制的行中)。

我们可以把绘制的点按纵坐标分类，以致不需要每一次查询整个阵列。这个逻辑的缺陷是在处理时，我们将查询阵列 20 次或者关于纵坐标 20N 对检验，分类将进行某个因子乘以 NlogN 次检验。正如我们所希望的那样，N 通常比 20 大得多，并且由于分类的花费并不便宜，因此我们认为在这种情况下简明即是效率。

绘图不是很完美的。因为只有 40 个数据点被描绘出，尽管横坐标中有更多的点。所以对于小的 X 值，绘图有明显的缺陷。

正如上面指明的那样，辅助程序 set1()是起骨干作用的程序。为了改进这个程序，人们想到了许多修改方法和可供选择的方案。它们中的一些可能是：

1)选用更多和更复杂的表格，循环记号标志能够放置在坐标轴上，图和轴自身能够给出标题。

2)绘制不止一个函数的选择，通过用一被绘制的事物(函数或者点的集合)的计数去代替 n，这个选择是很容易做到的。沿着具有这个点计数的一个阵列和具有被给出的字符的类似阵列，这将需要一个包含在 splot()内的作图表示和能表明函数记数的循环程序。

3)对于绘制轨道、地图等等，指定横轴和竖轴的刻度是有用的(特殊情况下，横坐标刻度和纵坐标刻度相同)。如果一条轨道是圆，那么它看起来应该是圆形的，而不是椭圆形的。

一般这种方法是不可移植的，用身边的屏幕作的某个实验或者绘图将得到说明这种方法的纵横比的最好近似值。用所有可能的方法，在这个屏幕和相应的打印机上，我们看到绘图的结果将是不同的。

4)可光滑的和内推的连续线能够画出，能够用一指定字符或者一张更复杂的表格循环地标出。为此，我们可处理出版质量的水平问题。正象上面提到的那样，在原始数据分析中，有理由避免如此复杂的方法，并且这样的一个程序不适合于作离散图。使用把点分解成纵坐标的几个区域的 splot()程序的方法已不再有意义，而这种方法希望有同样多的这样的组作为绘图解法的中间手段。对于这样高级的绘图解法，它对于在点之间作曲线是更有效，画点之间的最容易的方法是直线。从一对 X，Y 到屏幕上的像数点的相应的逻辑地址空间的映射能够处理，正象在 splot()程序中那样，也就是在 splot()允许的范围内确定包含在 dx 和 dy 中的比例因子。更复杂的内推法是产生光滑线的一种通用的替换，这种内推法是基于 splines，拟合成的 spline 曲线不得不被分解为要绘制的线段。关于 B－splines 的运用，见第八章。而这些方法使用简单的三次 splines 可能更经济，而简单三次 splines 要求函数值和函数在每个区间端点处的一阶导数匹配(参见第八章的讨论)。

表 13－1　第十三章的函数和子程序

splot()	产生 Y 与 X 阵列的简单绘图
plotl()	在光栅输出设备上绘制一条线
setl()	使绘图中的线变得清晰
setl()	在本级字符间作线
unclob()	保证空白线

§13.4 程序列表与测试问题输出

```
/* simple line printer plotting package
(from "C Tools for Scientists and Engineers" by L. Baker)

CONTENTS:
splot
  plots point set
setl
  initializes contents of a row of the plot
plotl
  plots a row
unclob
  ensures no premature NULLs in row to be plotted

DEPENDENCIES:
NONE
*/
#define max(a,b) (((a)>(b))? (a):(b))
#define min(a,b) (((a)<(b))? (a):(b))

#define NLINES 20

main(argc,argv)int argc; char * *argv;
{
float y[100],x[100],z;;int i,j;
for (i=0;i<40;i++)
    {
    x[i]=i;y[i]=i*i;
    }
splot(x,y,40);
exit(0);
}

setl(line,n,chr)
char chr, *line;int n;
{
int i;
for (i=0;i<n;i++)line[i]=chr;
return;
}

plotl(line,n) char line[];int n;
{
```

```
line[n]='\000';
printf("%s\n",line);
return;
}

splot(x,y,n)float x[],y[];int n;
{
folat xamx,ymax,xmin,ymin,dx,dy,ytop,ybtm,ybar,dxi;
int k,i,j;
char line[81],label[17];
xmin=xmax=x[0];
ymin=ymax=y[0];
for (i=1<n;i++)
    {
    xmin=min(xmin,x[i]);
    xmax=max(xmax,x[i]);
    ymin=min(ymin,y[i]);
    ymax=max(ymax,y[i]);
    }

dy=(ymax-ymin)/NLINES;
dx=(xmax-xmin);
dxi=50./dx;
if(dx==0 || dy==0)
   {printf(" plot range=%f domain=%f\n",dx,dy);
     return;
   }

setl(label,19,' ');
label[18]='\000'; /* make sure label is NULL terminated */

ytop=ymax;
ybtm=ytop-dy;
for (i=0;i<NLINES;i++)
    {if(i<(NLINES-1))setl(line,80,' ');
    else setl(line,80,'-');
    line[19]='|';
    for(j=0;j<n;j++)
      {
      if(y[j]>ybam && y[j]<=ytop)
      {k=(x[j]-xmin)*dxi+19;
      line[k]='*';

      }/* if */
    };/* for */
if(i==0){sprintf(label,"%10.2f",ymax);
   strncpy(line,label,10);
```

```
    unclob(line,12);
    }
 else if (i==(NLINES-1))
    {sprintf(label,"%10.2f",ymin);
    strncpy(line,label,10);
    unclob(line,12);
    };
    plotl(line,78);
    ytop-=dy;
    ybtm-=dy;
    }/* end of loop over lines */
setl(line,80,' ');
/* x values print */
sprintf(label,"%10.2f",xmin);strncpy(&(line[15]),label,10);
sprintf(label,"%10.2f",xmax);strncpy(&(line[65]),label,10);
plotl(line,78);
return;
}

unclob(line,kt)int kt;char line[];
{int i;
for (i=0;i<=kt;i++)
    {
    if(line[i]=='\000')line[i]=' ';
    }
}
/* COMPLEX.H header file
 * use for complex arethmetic in C
 * see MULLER.C for support functions such as
   csqrt(),cdiv(),cexp(),argm(),polarxy()
 (from"C Tools for Scientists and Engineers" by L.Baker)

DEPENDENCIDS:
NONE
*/

struct complex { double x;
    double y; };
double d,sqrt();  /* dummy */

/* for below,X,Y are complex structures,and one is returned */

#define CMULTR(X,Y)((X).x*(Y).x-(X).y*(Y).y)
#define CMULTI(X,Y) ((X).y*(Y).x+(X).x*(Y).y)
#define CDRN(X,Y  ((X).x*(Y).x+(Y).y*(X).y)
#define CDIN(X,Y) ((X).y*(Y).x-(Y).x*(X).y)
#define CNORM(X) ((X).x*(X).x+(X).y*(X).y)
```

```
/*
#define CNRM(x) (X->x*X->x+X->y*X->y)
*/
#define CDIV(z,nu,de) {d=CNORM(de);z.x=CDRN(nu,de)/d;z.y=CDIN(nu,de)/d;}
#define CONJG(z,X) {(z).x=(X).x;(z).y=-(X).y;}
/* #define CONJ(X) {(X).y=-(X).y}
*/
#define CMULT(z,X,Y) {(z).x=CMULTR((X),(Y));(z).y=CMULTI((X),(Y));}
#define CADD(z,X,Y) {(z).x=(X).x+(Y).x;(z).y=(X).y+(Y).y;}
#define CSUB(z,X,Y) {(z).x=(X).x-(Y).x;(z).y=(X).y-(Y).y;}
#define CLET(to,from) {(to).x=(from).x;(to).y=(from).y;}
#define cabs(X) sqrt((X).y*(X).y+(X).x*(X).x)
#define CMPLX(X,real,imag) {(X).x=(real);(X).y=(imag);}
#define CASSN(to,from) {to.x=from->x;to.y=from->y;}
#define CTREAL(z,X,real) {(z).x=(X).x*(real);(z).y=(X).y*(real);}
#define CSET(to,from) {to->x=(from).x;to->y=(from).y;}
```

