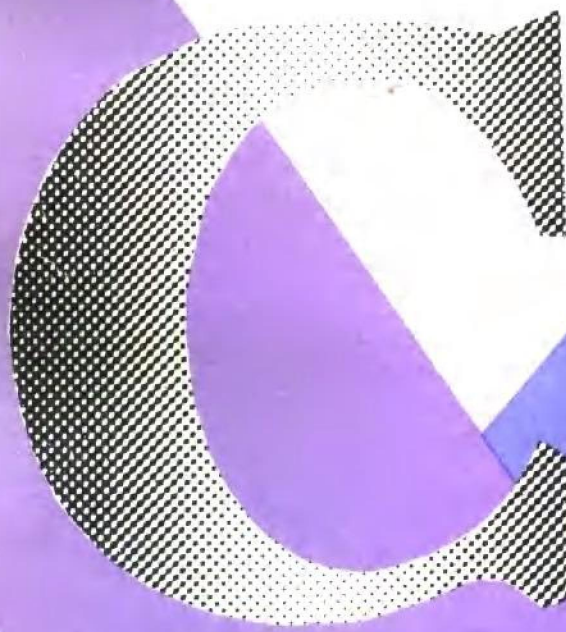
科学工程C
源程序范例解

邹贻明　郭新明　张一立等编



四川大学出版社

科学工程C源程序范例解

邹贻明　郭新明　张一立　等编

四　川　大　学　出　版　社
1994年·成都

(川)新登字 014 号

责任编辑:石大明
封面设计:冯先洁
技术设计:石大明

科学工程C源程序范例解

邹贻明　郭新明　张一立　等编

四川大学出版社出版发行　　(成都市望江路 29 号)
四川省新华书店经销　　郫县犀浦印刷厂印刷
787×1092mm　16 开本　　15 印张　　325 千字
1994 年 9 月第一版　　1994 年 9 月第一次印刷
印数:1—5000 册
ISBN 7—5614—1093—X/TP・17　　定价:12.00 元

前　言

本书的目的是提供一个测试算法的集合，这些算法立刻可以应用到科学家(自然科学和社会科学)、工程师和程序员所遇到的现实问题中去。所有码都已经过测试而且提供了测试情况。

不要错误地认为本书提供的只是程序片段，而用户在构造他们的大程序时必须是应用这些程序块的专家。实际这里给出的程序可以为大量的码服务，很多都是完整的程序，用户可以方便使用。

用本书有许多方法。一种是提出程序体，先运行测试情况然后应用于手边的问题。介绍的许多方法在处理问题时可能作很少修改或不作修改。我们鼓励用户阅读讨论并理解这些方法，这会避免不必要的事情发生。另一种方法是作为C码例子集。本书介绍的数值方法是数值分析教科书很有用的补充，而非数值算法应该是程序员感兴趣的。统计学家、社会科学家或学习这些课程的学生将会对统计方法感兴趣。同样，工程师和自然科学家、研究工作者和学生都能在本书中找到有用的程序。

本书的程序是用C语言写的。该语言是计算机世界的通用语言，从微机到超级计算机，PC到crays都有C，它是结构化的高级语言。用C写的算法有很好的可移植性，容易迅速地转变为其它高级语言。

本书的算法按方法集中而不是按应用。这样选择的目的是对不同训练者提供有用的方法，同时也讨论了各种可能的应用。

前两章为初学数值分析或C语言的读者提供服务。特别给出了FORTRAN转变为C的接口。因为许多科学码和工程码是用FORTRAN写的，所以接口应该是感兴趣的。

第三、四、五章包含了线性代数的例行子程序。它们包括解联立方程系统、逆矩阵、特征值分析和实行带奇异值分解的多变回归(最小二乘法拟合)。第五章讨论的奇异值分解(SVD)在统计学中有许多应用，同时讨论了C头文件的使用使FORTRAN码转变为C语言码更容易。

第六、七两章介绍了从线性方程转向非线性方程。牛顿——辛普森(Raphson)方法首先用来解具有实根的方程系统。按詹金斯——特劳布(Jenkins－Traub)对多项式根的迭代法提出的求复数根的穆勒(Muller)方法使复数算术能够实现。

第八至十二章对微分方程数值解提供了工具。首先讨论了扦值，特别是B－样条函数的使用。在自适应求面积(正交)程序中，应用扦值于积分。一个非常重要的积分是傅里叶变换。用于初值问题和"稳定问题"的方法在这一章提出，并介绍了龙格——库塔——费尔堡(Runge－Kutta－Fehlberg)方法。有些书常常省略了稳定系统求解程序，而这对处理如化学反应系统是非常重要的。

第十三章对图形进行了描述，同时给出了简单容易的绘图程序。

对各种可能的情形，几乎没有一个算法是十全十美的。因此对不同的情况将讨论更合适的方法。由于计算机科学在快速发展，这些方法也会不断地改进。

我们希望读者能从失败中解救出来，努力地应用这里提出的例行子程序。也希望学生和其他的人有机会学习这里提出的方法。这些方法是程序和数值分析领域中许多人长期努力的精华。本书的读者一定会感觉到当他们使用这里提出的方法时自己是站在许多巨人的肩上；可以肯定他们将使自己的工作做得更快、更好、更容易。

这里提出的方法已作了各种测试以保证其正确性，留给用户最后的回答是肯定他们的结果正确。

参加本书编写的同志有：张一立、黄武、黄桂钦、张茂孝。

目　录

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：

DO1I=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