有限元技术丛书

The finite element method is a numerical method for solving problems of engineering and mathematical physics. Typical problem areas of interest in engineering and mathematical physics that are solvable by use of the finite element method include structural analysis, heat transfer, fluid flow, mass transport, and electromagnetic potential.

有限元方法编程

（第三版）

Programming the Finite Element Method, Third Edition

[美] I. M. Smith 著
D. V. Griffiths

王崧 周坚鑫 王来 裴波 等译

WILEY

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

有限元技术丛书

有限元方法编程
(第三版)

Programming the Finite Element Method
Third Edition

[美] I.M.Smith
D.V.Griffiths 著

王 崧 周坚鑫 王 来 裴 波 等译

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内容简介

许多关于有限元的教材通常只介绍有限元分析的原理以及实际工程问题的应用，而不并致力于产生实际数值结果的计算机程序的构建。本书则致力于帮助读者通过有限元技术来使用为算法而设计的“构件块”。其重点并不在于程序，而在于过程或子程序的集合。目的在于教会读者编写智能程序并使用它们。内容涉及使用有限元来解决固体及液体问题的微分方程、微分方程的算法实现与程序，以及在各个领域的应用。

本书结构合理，示例丰富，适用于致力于有限元研究与开发的学生以及工程技术人员，是一本不可多得的参考书。

I. M. Smith, D. V. Griffiths: **Programming the Finite Element Method, Third Edition**
ISBN 0-471-96543-X

图书在版编目（CIP）数据

有限元方法编程/（美）史密斯（Smith, I. M.）著；王崧等译. -3版. -北京：电子工业出版社，2003.9
（有限元技术系列）
书名原文：Programming the Finite Element Method, Third Edition
ISBN 7-5053-9115-1

Ⅰ.有… Ⅱ.①史… ②王… Ⅲ.①有限元法 ②算法语言-程序设计 Ⅳ.①O241.82 ②TP312

中国版本图书馆CIP数据核字（2003）第077648号

责任编辑：窦昊
印 刷 者：北京兴华印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路173信箱 邮编：100036
经 销：各地新华书店
开 本：787 × 1092 1/16 印张：26.75 字数：685千字
版 次：2003年9月第1版 2003年9月第1次印刷
定 价：48.00元

译 者 序

随着计算机技术的发展，有限元技术在各个工程领域正日益显示出强大的生命力。有限元技术是一种数值计算技术，它对不能用解析方法求解的问题，有着独特的应用能力。但是，掌握有限元技术又不仅仅是一个纯粹的理论问题，它要求应用人员有一定的程序设计能力，才能将它们应用于工程领域实践，否则只能“望洋兴叹”。本书就是在这样一种背景之下翻译出版的，使读者真正具有编制结构清晰可见阅读性强的有限元程序。

本书的特点之一是提供了相当多的源程序供读者参考，真正“站在前人的肩膀上”。本书提供的各个程序都利用了 Fortran 90 的强大功能，采用模块化编程技术实现，有利于读者阅读。同时，结合具体的应用实例，对本书中各个程序的应用原理、变量含义等做了简明扼要的说明。本书的另一特点是，对每一类问题都提供了不同的求解技术，如高斯直接消元法、迭代法、隐式积分法、显式积分法、共轭梯度法等。通常情况下，有限元分析对计算机的存储要求和运行速度要求都是很高的，尤其是对大型问题而言。因此，本书除了提供通常采用的单元组装技术之外，还提供了有关“逐个单元”法或“自由网格”法等求解技术，它们不需要存储大型的总刚度矩阵。

此外，本书涉及到的知识面相对较广，但主要围绕三个方面的问题展开有限元程序设计，即固体力学问题、流体力学（包括热力学）问题以及固体与流体的耦合问题（如土力学中的问题），涉及到的方程主要有静力平衡方程、传导方程和特征值方程。本书中，第 2 章和第 3 章是以后各章的基础，其中第 4 章 ~ 第 6 章是针对弹塑性问题的静力分析，应用的主要是静力平衡方程；第 7 章 ~ 第 9 章主要是针对流体问题，或者流体与固体的耦合问题，应用的主要是传导方程；第 10 章 ~ 第 11 章是弹塑性实体的动力分析，应用的主要是特征值方程。当然，它们的有限元分析过程都是类似的，这有利于读者针对不同的问题扩展现有程序。

本书作为一本教材，主要针对学习有限元编程技术的读者。另外，对相关领域内从事有关有限元程序设计及应用的技术人员也有指导作用。

在本书翻译过程中，译者一般都对照了源程序，力求翻译准确、文字简单明了，但由于涉及到的知识相当多，难免有不足之处，敬请广大读者批评指正！参加本书翻译的人员主要有：王崧、周坚鑫（航空物探遥感中心）、王来（天津大学建工学院博士）、裴波、刘丽娟、唐国兵、唐伯鉴，全书统一由王崧校核。最后，感谢天津大学建工学院王铁成教授对本次翻译的悉心指导！

第三版前言

本书在前两版的基础上做了大量的修改和补充，其中最主要的修改有：用 Fortran 90 重写了所有的子程序库和示例程序；在各章同时给出了“自由网格法”或“逐个单元法”，作为对传统的“网格剖分法”或“整体组合法”的补充。此外，举例分析时，采用的是解析法而非数值积分法。

后面的大多数章节还附有例题，这使得本书更适合作为一本教材。

第 1 章用了大量篇幅来解释现在编程时所考虑的因素，以及新的编程标准 Fortran 90 的主要特性，这一新的编程标准在有限元编程中非常有用。

在第 3 章中，我们将介绍迭代法，特别是“前处理的共轭梯度法”，这些方法将用来求解分析静力平衡问题时所产生的线性代数方程系统。读者将会看到，这些方法使得在有内存限制的小型计算机上采用“逐个单元法”求解超大型问题变得更为容易。另外，在求解特征值问题和瞬态问题上也会有类似的应用，同时读者还会看到 Fortran 90 在求解这些问题方面强大的生命力。

在第 5 章中，本书将第一次引入与矢量化并行程序有关的问题，并提供单元积分的解析求解法，作为对传统的数值积分法的一种替代选择。

本书的一个重要特点是，突出了它在有限元应用方面的通用性，但在第 6 章还是专门针对岩土力学方面的应用讨论了有关的塑性问题。因此，本书第一次新增了与堤坝建设和土方开挖等施工过程有关的内容。另外，在第 9 章也新增加了与岩土力学有关的非线性耦合问题。

尽管本书未包括与前处理（网格生成）和后处理（结果显示）有关的程序，但有关例子会说明后处理的重要性。事实上，包含动画演示这样的功能是极其重要的，尤其是对于大型问题，更是如此。为便于前后处理，本书第一次增加了有通用目的的程序。这些程序使得数据可以由用户选择的网格生成器来提供，而不仅仅通过简单的“几何结构”子程序来提供，这些子程序更多地用做解释和讲解目的。

本书中所有的软件（程序库和程序）都可以通过 FTP 方式或 Web 方式获得，其地址分别为 ftp://golden.eng.man.ac.uk/pub/fe 和 http://eng.man.ac.uk/geotech/software。

目　录

第 1 章　预备知识:计算机基础

1.0　引言

现在关于有限元方面的书籍已经有很多,但大多数都是讲述有限元分析方法的原理及其在解决实际工程问题方面的广泛应用,鲜有讲授计算机有限元编程方面的书。即使有,这些书要么假定读者已能使用那些已编制好的程序(也可能是很复杂的"程序包"),要么假定读者已具备编制计算机程序方面的能力。但事实上,对于那些在有限元领域缺乏丰富经验的人而言,在理解有限元分析的原理和具体如何编程实现方面,仍有相当的距离。

本书即为解决上述问题而编写的。它旨在通过一套专门针对有限元计算而设计的"模块构建"策略,帮助读者编制好自己的计算机程序,以解决特定的工程问题。本书随后给出的一些程序实质上并不是某一个程序或者某一组程序,而是一个有关过程或子程序的集合(函数库),它能够执行类似于标准函数(SIN,SQRT,ABS 等)的某些功能——这里所说的标准函数是指在所有的科学计算机语言中,其永久函数库都能提供的函数。由于有限元方程往往呈现某种矩阵结构形式,所以大多数构建模块程序都与矩阵运算有关。

这些构建模块可以以不同的方式进行组合,生成解决各种工程、科学问题的测试程序。这种编程方式的意义在于:一个这样的测试程序就可以作为一个开发平台,有兴趣的用户可以在该平台上开发新的应用程序。

本书的目的是教会读者如何编写巧妙的程序并使用之。不过,在编写程序时,程序的效率并不是最为重要的。此外,本书中所有的构建模块程序(超过 100 个)和测试程序(超过 50 个)已经在多种计算机上测试通过。

本书所选用的编程语言是 FORTRAN 的最新标准——Fortran 90。本章后面,将对 Fortran 90 中与有限元法编程密切相关的一些特点做一全面的阐述。现在仅需说明的是,相对于 FORTRAN 语言以前的标准 FORTRAN 77(本书前两版中的程序均用 FORTRAN 77 编写)而言,Fortran 90 有了彻底的改进;同时,毫无争辩地,Fortran 仍然是编写大型工程和科学计算程序最适合的语言。

1.1　计算机硬件

原则上,任何能够编译和运行 Fortran 程序的计算机均可以执行本书中所给出的有限元分析。但实际上,从用于普通计算和教学目的的个人计算机到用于大型计算(尤其是非线性的三维分析)的超级计算机,计算机硬件的差别非常之大。在此向读者推荐一个重要的编程策略,即同一个软件能够运行于所有类型的机器之上。至于矢量计算机和并行计算机的特性将在 1.3 节单独讲述。

用户对计算机的选择取决于性价比。比如说,一般情况下,一台价值 1500 英镑(约合 2500 美元)的个人计算机,比一台价值 15 000 英镑(约合 25 000 美元)的工作站计算速度慢 10 倍。所以,在一台工作站上花 5 分钟就可完成的任务在个人计算机上可能需要一个小时。很显然,选择哪种计算机更好,完全取决于个人情况。但我们仍然建议不要选用相对于工作任务而言太差的计算机;也就是说,我们建议用户不要在计算机能力的极限状态下进行运算处理。否

则,在任何设计周期中,都会因为所需的处理时间过长而使得这样的计算毫无价值。

1.2 存储管理

在本书所有的程序中,都假定有足够的内存用以存储数据和执行程序。但是,在有限元计算中需要处理的数组很可能非常大,比如说 100 000 × 1000,这样计算机就需要 1×10^8 个字的内存来存储这些信息。尽管有这样的计算机存在,但也相当少,目前更典型的计算机内存容量仍然是在 1×10^7 个字这样的数量级上。

避免这个问题的一种办法是让程序员编写一个能"溢出内存"的子程序,以便能处理内存中大量的数组,以及在外存和内存之间传输适当的数据。

另外一种办法是将存储管理的控制权从程序员手中交给系统硬件和软件。在这种情况下,程序员看到的仅仅是一个容量很大的单一内存,数据信息在控制程序或执行程序的控制下,在二级存储器和主存之间来回传输。这就是"虚拟"内存的概念,它是 1961 年在 ICL ATLAS 论坛上首次出现的,到现在这一技术已经非常普遍了。

很显然,这种计算机系统必须能够将变量的虚拟地址转换成在内存中的真实地址。这一转换通常包含了一个复杂的位模式匹配过程,称为"分页"。将虚拟存储器划分成大小固定或可变的段或页,通过页表引用,监控程序就能够根据用户存取数据的方式"学习",以便能按照某种预测的方式管理存储空间。但是,存储管理是不可能完全脱离程序员的控制的,而且总是假定程序员能按照某种合理的逻辑来进行操作,譬如按顺序访问数组元素(就像编译器和编程语言按行或列那样组织)。如果程序员以随机方式访问一个容量为 108 个字的虚拟内存,则调页请求会使程序的执行效率非常之低(参见 Willé,1995)。

在不久的将来,"大型"的有限元分析——比如说含有 100 000 个以上的未知量——可能要用下一节即将介绍的矢量计算机或并行计算机进行处理。当采用这种计算机时,如果程序员在程序中中断计算过程去执行内存与外存之间的数据传输,或者处理过程中自动出现换页的情况,则计算过程仍会耗费相当多的时间。因此,在本书的第 3 章将介绍一些特殊的策略,使得大型的有限元分析无需内外存数据交换就可以处理。

1.3 矢量计算机与并行计算机

早期的数字计算机是以"串行"方式运行的,通俗地讲,如果有 1000 个操作要执行,仅当第一个操作完成第二个操作才会开始,依此类推。不过,当操作是在数组上进行时,我们完全可以想像,如果对两个数组元素进行操作的结果不会影响对另外两个数组元素的操作,那么这样的计算完全可以同时执行。计算机中实现这种功能的硬件称为"管道",一般而言,所有的现代计算机都或多或少使用了这种硬件技术。由可实现管道化操作的专用硬件组成的计算机称为矢量计算机。由于"管道"的长度毕竟有限,所以,为了使操作能同时进行,就必须使相关的操作数在正确的时间内放到管道中;此外,还必须遵守一个原则,即同时进行的两个操作不能互相依赖。这两点要求(众多要求的一部分)意味着:为使计算机的矢量处理能力得到最大限度地发挥,在编写程序时必须采取一些必要的措施。另外,还有一个很有意思的"副作用",即为矢量计算机编制好的程序在其他任何计算机上都会运行得更好,这是因为计算所需的数据信息能在恰当的时间出现在恰当的位置上(比如在一个特殊的缓存当中)。正是因为矢量计算机

的这些优点,就连现代所谓的"标量计算机"也逐渐开始含有具有矢量功能的硬件。在本书第5章的开头部分将举例说明完全"矢量化"的程序。

但是,真正的矢量计算机是非常昂贵的。另一个可替代的办法是采用"并行"计算——能对部分数组信息进行并发处理。属于这一概念(其中存在许多变体)的计算机有几种,它们在物理硬件上各有千秋(比如说从比较昂贵的到相对便宜的)。在这种计算机中,程序和数据可以在不同的处理器中进行处理,当然这要求处理器之间能够互相通信。

目前,有两种可预知的通信组织方式(就像前面讲述的内存管理那样)。一种是由程序员控制通信进程——使用一个称之为"消息传递"的程序设计机制;另一种是由程序自动执行而无需程序员的控制。第二种策略毫无疑问是很吸引人的,并因此导致了"高性能 Fortran"(简称为 HPF)的出现(参见 Koelbel et al.,1995),而且 HPF 被设计成 Fortran 90 的一个扩展。将非 HPF 编译器视为注释的"目录"插入到 Fortran 90 程序中,并将数据及对这些数据进行操作的协议一起映射到并行处理器上,那么计算就可以并行执行了。这一策略的诱人之处在于程序的可移植性,也就是说,程序可以很容易地从一台计算机上搬到另一台计算机上执行。当然,我们也期望有厂商能生产出更好地利用这种硬件特性的编译器来。值得高兴的是,在本书编写时第一个 HPF 编译器刚刚发布。

另外,在程序员控制下的消息传递策略可以在诸如 MPI("消息传递接口")这样的环境中实现。这一策略同样也试图保持程序在各种计算机之间的可移植性,但这同时会带来很多额外的编程量,并且明显不符合非并行程序(构成本书主体的)简短紧凑的形式。

本书第 5 章给出了一个带 MPI 插件的有限元程序。在此,我们希望 HPF 最终能够成功,但同时也希望生产并行计算机的厂商能提供他们自己专门的消息传递环境。

1.4　软件

因为计算机硬件(指令格式、矢量计算能力等)的不同,存储管理方式也有所不同。所以,为了最有效地利用这些不同的设备,运行于其上的程序的结构当然应该因机器的不同而不同。但是,为了便于程序的移植和程序员的培训,所有计算机上的工程计算程序通常都是用"高级"语言编写的,这种高级语言独立于计算机的硬件。

这些"高级"语言由一个称为"编译器"的程序翻译成机器指令。

FORTRAN 是到目前为止用于工程与科学计算最为广泛的语言,本节将重点介绍 Fortran 的最新标准(Fortran 90)用于有限元计算中的一些主要特点。

图 1.1 所示是一个典型的 Fortran 90 程序(Smith,1995),用于处理有关的民意测验,在这里主要用它来向 FORTRAN 77 或其他类似语言的读者说明 Fortran 90 的基本结构。

从这个例子可以看到,程序可以用自由格式书写,也就是说,程序员可以在页面或屏幕上随意安排语句的位置。其他值得注意的特点还包括:

- 大小写字符可以任意组合。
- 多条语句可以置于一行,仅需以";"相隔。
- 长语句行可以在行尾使用"&"进行续行,并同时在续行的开头也加上"&"。
- 注释以"!"开头,在编译时为编译器忽略。
- 长名字(包括下划线在内最多 31 个字符)允许用有表征意义的标识符。
- "IMPLICIT NONE"语句强制要求所有变量名和常量名都必须显式声明。这在调试程序

时有极大的帮助。

- 变量声明使用双冒号“::”。
- 程序中没有带标号的语句。

```
PROGRAM GALLUP_POLL
! TO CONDUCT A GALLUP POLL SURVEY
IMPLICIT NONE
INTEGER : :  SAMPLE, I, COUNT, THIS_TIME, LAST_TIME, TOT_REP, TOT_ &
             MAV, TOT_DEM, TOT_OTHER, REP_TO_MAV, DEM_TO_MAV, &
             CHANGED_MIND
READ*, SAMPLE
COUNT=0;  TOT_REP=0; TOT_MAV=0; TOT_DEM=0; TOT_OTHER=0; REP_ &
       TO_MAV=0
DEM_TO_MAV=0; CHANGED_MIND=0
OPEN (10, FILE='GALLUP.DAT')
DO I = 1, SAMPLE
  COUNT=COUNT+1
  READ (10, '(I3,I2)', ADVANCE='NO') THIS_TIME, LAST_TIME
  VOTES: SELECT CASE (THIS_TIME)
    CASE (1); TOT_REP=TOT_REP+1
    CASE (3); TOT_MAV=TOT_MAV+1
     IF (LAST_TIME/=3) THEN
       CHANGED_MIND=CHANGED_MIND+1
       IF (LAST_TIME==1)REP_TO_MAV=REP_TO_MAV+1
       IF (LAST_TIME==2)DEM_TO_MAV=DEM_TO_MAV+1
     END IF
    CASE (2); TOT_DEM=TOT_DEM+1
    CASE DEFAULT; TOT_OTHER=TOT_OTHER+1
END SELECT VOTES
END DO
  PRINT*,  'PERCENT REPUBLICAN IS', REAL (TOT_REP)/REAL (COUNT)*100.0
  PRINT*,  'PERCENT MAVERICK IS', REAL (TOT_MAV)/REAL(COUNT)*100.0
  PRINT*,  'PERCENT DEMOCRAT IS', REAL (TOT_MAV)/REAL(COUNT)*100.0
  PRINT*,  'PERCENT OTHERS IS', REAL (TOT_OTHER)/REAL(COUNT)*100.0
  PRINT*,  'PERCENT CHANGING REP TO MAV IS', REAL (REP_TO_MAV)/&
            REAL (CHANGED_MIND)*100.0
  PRINT*,  'PERCENT CHANGING DEM TO MAV IS', REAL (DEM_TO_MAV)/&
           REAL(CHANGED_MIND)*100.0
END PROGRAM GALLUP_POLL
```

图 1.1 一个典型的 Fortran 90 程序

1.4.1 算术精度

有限元分析是一门计算性很强的技术(见第 6 章和第 10 章的例子),它能达到的最大数值精度由 64 位的机器字长决定。Fortran 90 中有一些内部函数可用于决定、改变处理器的精度。例如,下面的语句:

```
IWP=SELECTED_REAL_KIND (P =15)
```

将返回一个整型变量 IWP ,使处理器上的“KIND”变量取得 15 位的十进制精度。如果处理器达不到这个精度,IWP 将返回一个负值。

在为 IWP 赋予必要的值之后,Fortran 90 将采用如下形式声明 REAL 型变量:

```
REAL (KIND=IWP): :A,B,C
```

并使它们获得如下的赋值形式:

```
A=1.0_IWP; B=2.0_IWP; C=3.0_IWP
```

为简明起见,本书给出的所有程序都假定 KIND 的默认值已足够,因此不再需要附加符_ IWP。不过,必须提醒读者注意的是,本书所有的计算结果都是在 64 位的处理器上获得的,如果要在字长更短的处理器上计算,这些结果将会有一定的误差。事实上,对一些要求严格的计算,在低于 64 位的处理器上进行计算可能会导致严重错误的结果,也可能导致毫无结果的运算失败。

1.4.2　条件语句

Fortran 90 有两种基本形式的条件语句。第一种是大多数高级语言中都有的 IF...THEN...ELSE 结构条件语句。这种结构采取如下形式:

```
NAME_OF_CLAUSE:IF (逻辑表达式) THEN
                                    .
                                    .
                                    .
                                    第一部分语句
                                    .
                                    .
                                    .
                              ELSE
                                    .
                                    .
                                    .
                                    第二部分语句
                                    .
                                    .
                                    .
        END IF NAME_OF_CLAUSE
```

比如:

```
    CHANGE_SIGN: IF(A/=B) THEN
                                    A=-A
                              ELSE
                                    B=-B
        END IF CHANGE_SIGN
```

上面这个例子中,条件语句的名称 NAME _ OF _ CLAUSE 和 CHANGE _ SIGN 是可选的,可以省略。

第二种条件语句在图 1.1 所示的程序中出现过,它包含了"SELECT CASE"结构。如果是在非常简单的条件下进行选择,比如,判断条件是一个已赋值的 INTEGER(整型)、LOGICAL(逻辑型)或者 CHARACTER(字符型)变量,就可以采用下面这种结构形式:

```
SELECT_CASE_NAME: SELECT CASE (变量或者表达式)
                        CASE (SELECTOR)
                        第一部分语句
                        CASE (SELECTOR)
                        第二部分语句
                              .
                              .
                              .
                        CASE DEFAULT
                        默认语句
    END SELECT_CASE_NAME
```

这种结构现替代了 FORTRAN 77 中很糟糕的"go to"结构。

1.4.3　循环语句

Fortran 90 中有两种循环结构。在第一种结构中,循环体重复执行固定的次数,比如:

```
FIXED_ITERATIONS: DO I = 1,N
                                    循环体
END DO FIXED_ITERATIONS
```

在第二种结构中,循环的退出与继续依赖于某种条件的结果,例如:

```
EXIT_TYPE: DO
                    循环体
                     IF (条件语句) EXIT
                    循环体
END DO EXIT_TYPE
```

或者

```
CYCLE_TYPE: DO
                    循环体
                     IF (条件语句) CYCLE
                    循环体
END DO CYCLE_TYPE
```

上述第一个循环变体在条件成立后跳出循环,将程序的控制权交给 END DO 后的第一个语句;第二个循环变体在条件成立后返回循环体的顶部,并跳过 CYCLE 和 END DO 之间所有的语句。

在上述两个例子中,如条件语句的情况一样,循环语句的名称也是可选的。在本书所有的程序中,对有明显意义的循环和条件语句都取有名称,仅比较简单的才予以省略。

1.4.4 数组

1.4.4.1 动态数组

早期 FORTRAN 语言的最大缺陷也许是它在大规模数组计算中的不足,而大规模的数组计算在有限元分析中是不可避免的,因此,Fortran 90 修正了这一缺陷,它允许声明"动态"数组,也就是说,数组的大小不必在程序编译时就指定,而可以在某些数据已读入程序后或已求出中间结果后再进行内存大小的"分配"。现举一个简单的例子来说明:

```
PROGRAM DYNAMIC
  ! 举例说明数组的动态分配
   IMPLICIT NONE
  ! 声明二维数组A, 空间大小可变
   REAL, ALLOCATABLE :: A (:,:)
  ! 读入数组A的上下界
   READ*,M,N
  ! 给数组A分配实际的存储空间
   ALLOCATE (A(M,N))
   READ*, A
   PRINT*, 2.*SQRT (A) + 3.
   DEALLOCATE (A)! 释放数组A的存储空间
END PROGRAM DYNAMIC
```

这个简单的程序同时也显示了新标准的其他一些有用特性,其中最重要的一个特点是允许把整个数组作为一个操作数进行运算,所以,通过一个语句即可完成整个数组数据的读入,或计算数组中所有元素的平方根。这些特性所带来的效率提升程度会因为实际编译器的不同而有所不同。

1.4.4.2　数组"广播"

在上面的例子中,一个称为"广播"的特性允许对整个数组与标量(如前面例子中的 2 或 3)进行操作。在此,我们就说这些标量"广播"到了数组的所有元素上,因此前面的例子最终打印出来了数组中所有元素开方后再乘以 2.0 并加上 3.0 之后的结果。

1.4.4.3　数组构造器

Fortran 90 中除了按非常方式对数组元素赋值外,还允许对一维数组(即矢量)以如下形式进行赋值:

$$V = (/1.0, 2.0, 3.0, 4.0, 5.0/)$$

这等同于:

$$V(1) = 1.0;\ V(2) = 2.0;\ V(3) = 3.0;\ V(4) = 4.0;\ V(5) = 5.0$$

另外,数组构造器本身也可以有数组,如:

$$W = (/V,V/)$$

这会因 W 中不同的 10 个数而产生各种各样的结果。

1.4.4.4　矢量下标

整型矢量可用于定义数组的下标,这对有限元法(及其他数组方法)中的"汇集"(gather)与"散布"(scatter)操作是非常有用的。图 1.2 显示了由 8 节点四边形单元构成的网格的一部分,根据图中所示,网格的总节点数至少为 107 个。当"局部"计算必须涉及到个体单元时,比如求单元的应变或流量时,必须有一个局部索引矢量,以保存每个单元的节点编号,即:

单元 65:82,76,71,72,73,77,84,83

单元 73:93,87,82,83,84,88,95,94

等等。该索引或"方向"矢量可称为 STEERING _ VECTOR。当全局(GLOBAL)矢量要向局部(LOCAL)矢量中"汇集"时,可按下式表示:

```
LOCAL = GLOBAL (STEERING_VECTOR)
```

当 LOCAL 矢量要向 GLOBAL 矢量"散布"时,则是:

```
GLOBAL (STEERING_VECTOR) = LOCAL
```

在本例中,LOCAL 和 STEERING _ VECTOR 矢量是 8 个字长,而 GLOBAL 矢量则可能是数千甚至数百万个字长。

1.4.4.5　子数组

如果给数组的一个或多个下标指定一定的范围,我们就可以用它们来引用数组之中的某些部分(或称为"子数组")。如果某一下标未指明范围,则意味着引用该下标所在的整个范围。所以,如果 A 和 B 为二维数组,则 A(:,1:3)引用的是数组 A 中前 3 列所有的项,而 B(11:13,:)引用的则是 B 中第 11 行到 13 行所有的项。如果子数组与原数组形式一致,即有正确的行数和列数,则可以像对"整个"数组那样对其进行操作。

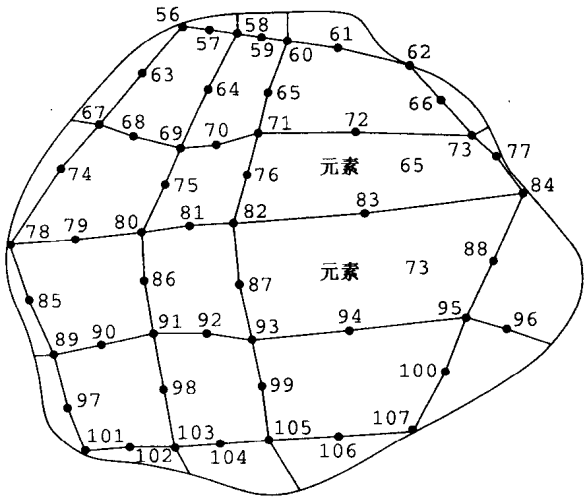

图 1.2 一个带节点号和单元号的有限元网格的一部分

1.4.4.6 数组的整体操作

Fortran 90 中有一个值得强调的与数组计算有关的特性，即允许对整个数组进行操作，这一特点使得在 FORTRAN 77 中很重要的几个子程序(本书前两版中都在使用)失去了存在的必要性。举个例子来说，如果矩阵 A、B 和 C 结构形式一致，S 为一个标量，则下面的运算都是合法的：

```
A=B+C    !不必编写一个包含 DO 循环的矩阵相加的程序
A=B-C    !不必编写一个矩阵相减的程序
A=S*B    !不必编写一个矩阵与标量相乘的程序
A=B/S    !不必编写一个矩阵与标量相除的程序
A=0.     !不必编写一个初始化矩阵的程序
```

不过，对相容的数组 A、B 和 C 而言，尽管 A = B * C 有意义，但它的结果是逐个单元法中数组 B 与 C 的单元乘积，读者不要把它与下一节将要讲述的矩阵相乘相混淆。

1.4.4.7 进行数组运算的内部函数

为了实现整个数组的算术运算，Fortran 90 提供有一些内部函数，在有限元计算中非常有用。这些内部函数通常分为两类，一类用于数组计算，另一类用于数组检查。用于数组计算的函数有：

```
FUNCTION MATMUL (A,B)          !返回A与B的乘积矩阵
FUNCTION DOT_PRODUCT (V1,V2)   !返回V1与V2的点积
FUNCTION TRANSPOSE (A)         !返回A的转置矩阵
```

以上三个内部函数在本书的程序中经常要用到，从而使用户不必像在 FORTRAN 77 中那样编写自己的子程序。

用于数组检查的函数有：

```
FUNCTION MAXVAL (A)    ! 返回数组A中的最大值元素(不是绝对值)
FUNCTION MINVAL (A)    ! 返回数组A中的最小值元素(不是绝对值)
```