目　　录

第 1 章　Maple V 概述

欢迎您成为 Maple V 的用户！Maple V 是一个集符号运算、数值计算、数据可视化、程序设计及文档处理等多项功能于一体的数学软件。经过国外近 20 年的考验，Maple V 以其无与伦比的符号计算能力，在当前的 30 多个数学软件特别是在数学分析型软件中独领风骚。在随后的章节中，您将会逐渐领略到 Maple V 的强大功能。

本章首先简单介绍 Maple V 的发展历程以及 Maple V 的功能特点，并给出本书的重点，接着介绍 Maple V R5 系统的安装过程和 Maple V 的用户界面。

1.1　Maple V 简介

按照软件数学处理的原始内核，可以把现有的 30 多个数学类科技应用软件大致划分为两个类别：一类是数学分析型软件，如 Mathematica、Maple 等，这类软件往往具有很强的符号计算能力，能够给出解析解和任意精度的数值解；另一类为数值计算型软件，如 MATLAB、Xmath 等，这类软件长于数值计算，能高效地进行大批数据处理。Maple 则是数学分析型软件领域的佼佼者。

本节简单介绍了 Maple 和 Maple V 的发展历程和 Maple V 的功能特点。

1.1.1　Maple V 的发展简史

Maple 是由加拿大 Waterloo 大学于 20 世纪 80 年代早期发展起来的一种具有强大符号计算和数学分析能力的软件。Waterloo Maple 公司于 1981 年发布了 Maple 的商业版本 Maple 4.0，之后又陆续发布了 Maple 4.1～4.3。Maple V 是 Maple 的更高版本，其早期的版本为 Maple V R1～R4。Waterloo Maple 公司于 1997 年 12 月推出了 Maple V R5，在用户界面、图形处理、数字处理和数学符号等方面都有了较大的改善和扩展。本书稿完成时，Maple V R5.1 已经推出。

在国外，Maple 和 Maple V 经过了近 20 年的发展，已经经受了多年的考验，在教育、科研和工业等领域得到了极为广泛的应用。近年来，国内用户正逐渐认识到 Maple V 在符号计算和数值分析方面的强大威力，因而 Maple V 也正得到越来越广泛的应用。目前，Maple 和 Maple V 已经拥有了包括科学家、工程师、数学工作者、教师和学生在内的数以万计的用户。

1.1.2　Maple V 的功能特点

Maple V 是一个功能强大的符号计算和数值分析软件，它在数据可视化和文档处理方

面具有很强的功能。Maple V R5 系统的函数库提供了 3000 多个命令和数学函数，其范围涉及数学的各个分支：基本代数、欧氏几何学、数论、有理函数、微积分、微分方程、图形学、线性代数、离散数学、群论等。下面给出了 Maple V 系统几个重要的功能特点：

1. Maple V 具有强大的符号计算能力

Maple 和 Maple V 最突出的功能为符号计算。符号计算的魅力在于：对于给定的某一问题来说，计算机给出的是没有任何误差的解析结果。举例来说，如果计算结果为 π，经符号计算后决不会显示 3.1415926…。这一点与数值计算截然不同。正是由于其无与伦比的符号计算能力，使得 Maple 在符号计算型数学软件中独领风骚。无论是 MathCAD 还是 MATLAB，在符号计算方面都得借助于 Maple 的巨大威力。

2. Maple V 具有很强的数值计算能力

和数值计算型软件如 MATLAB 相比，Maple V 的数值计算有自己的特色。Maple V 能把得到的分析解转换成任意精度的数值解。对那些没有最终分析解的问题，Maple V 能从中间分析解开始计算数值解，这将缩短误差的传递途径，从而最大限度地提高求解的精度。

3. Maple V 具有很强的数据可视化能力

Maple V 提供了包括二维和三维数据可视化、图像处理、动画制作、积分近似和微分方程求解等多方面的绘图命令。Maple V R5 提供了命令输入、菜单操作和拖曳对象到绘图区等几种方法，用户可以很方便地绘制出所需的图像或曲线。

4. Maple V 提供了一种结构化的内部编程语言

这种语言类似于 C、FORTRAN、BASIC、PASCAL 等第四代高级语言，使得用户可以快速地设计编写自己的程序。如果用户学习过其中的一种高级语言，将可以轻松地掌握 Maple V R5 的编程语言。本书限于篇幅，没有过多地讨论 Maple V 的编程知识，希望了解 Maple V 程序设计的读者可以参看其他相关资料。

5. Maple V 具有电子表格、超链接等高级文档操作的能力

Maple V R5 是一个多文档应用程序，其文档不仅可以包含命令的输入输出、纯文本等基本对象，而且可以包含图形、电子表格、可折叠区段及超链接等高级对象。这些高级对象使得用户可以方便地组织工作单中的内容，执行高级的文档操作。

6. Maple V 简单易用，具有完善的帮助系统

Maple V R5 提供了友好的集成用户界面，同时新增了虚拟键盘、上下文弹出菜单和 Standard Math 输入模式，大大方便了用户的操作。Maple V 还提供了功能完善的帮助系统，用户可以随时获得所需的帮助信息。

7. Maple V 占用很少的系统资源

Maple V 占用的硬盘空间小，内存需求量小。以 MS Windows 版的 Maple V R5 为例，完全安装也只需要 40M 左右的硬盘空间，再加上 8M 的内存就可正常工作了。这一点对于 Maple V 的推广应用来说是很有利的。

1.1.3 本书的重点

不可否认，Maple V 是一个功能强大的数学软件。Maple V 中提供的命令和函数有 3000 多个，本书不打算全部介绍 Maple V 提供的所有数学函数、命令及实现的功能，本书将重点介绍其中的各种常用数学函数、常用命令及常用功能的实现。虽然 Maple V 提供了功能同样强大的编程语言，由于篇幅所限，本书对编程的细节和技巧不作具体阐述。

本书中的例子都是在 Windows 98 操作系统上安装的 Maple V R5 中调试通过的。由于在 Maple V R5 中命令的输入字体类型默认是“Courier New”黑体，因此本书正文部分提到的 Maple 命令和函数的字体类型都采用“Courier New”黑体。

1.2 Maple V 系统的要求与安装

在开始使用 Maple V 进行工作之前，首先要安装 Maple V 系统。本节将简单介绍 Maple V R5 的系统要求和安装过程。

1.2.1 Maple V 系统的要求

Maple V 几乎适用于现有的每种操作系统，包括 Macintosh、MS Windows、MS DOS、Unix 和 VMS 等多种类型。因为 Maple V 在各种操作系统平台的基本配置几乎都是一样的，而且采用同样的命令格式，因此 Maple V 可以毫不费力地从一种操作平台移植到另外一种操作平台。本书的内容将主要以 MS Windows 操作系统上的 Maple V 为例，介绍 Maple V R5 的使用。

下面是以 MS Windows 操作系统为例 Maple V R5 的系统要求：

- IBM 或与之完全兼容的带数学协处理器的 Intel 486、奔腾及以上机型；
- 至少 8 位显示器适配卡和至少能显示 256 色的彩色显示器；
- CD-ROM 光驱；
- Windows 所支持的鼠标；
- 至少 32M 的硬盘空间；
- 至少 8M 的内存；
- Microsoft Windows 3.1、Windows NT 4.0、 Windows 95 或更高 Windows 版本。

1.2.2 Maple V 系统的安装

通常用户得到的 Maple V 安装盘中有两个安装文件，分别用来安装 Maple V 的应用程序和库文件。应用程序安装完毕后，Maple V 系统就可以正常运行了。Maple V 的共享文件库是由 Maple 命令、程序包、Maple 工作单和其他一些文件组成的。如果希望在运行 Maple V 时使用共享文件库，用户首先需要进行应用程序的安装，之后再运行库文件的安装文件。这里只介绍程序文件的安装过程，库文件的安装与之相似，用户按照安装向导的提示即可顺利完成。

Maple V 系统应用程序的安装过程非常简单，下面是 Windows 9x 版的安装步骤：

（1）把 Maple V 系统的安装光盘或 1#安装软盘插入相应的驱动器；

（2）启动 Windows 9x，打开 Windows 资源管理器；

（3）选择 Maple V 系统安装盘的盘符，然后在该盘中找到应用程序的安装文件“setup.exe”，通常该文件所在的目录中包含了“Win32s”子目录；

（4）双击该文件，启动安装过程。系统开始检查配置信息并加载安装向导，同时出现如图 1-1 所示画面：

图 1-1 Maple V 系统的安装之一

（5）单击“Next”按钮继续安装；在接下来的软件许可协议画面（如图 1-2 所示）中单击“Yes”按钮同意该许可协议；

（6）出现如图 1-3 所示的安装路径选择对话框，单击其中的“Browse”按钮可改变缺省的安装路径；注意选择容量足够大（>32M）的硬盘以顺利完成安装；

图 1-2　Map

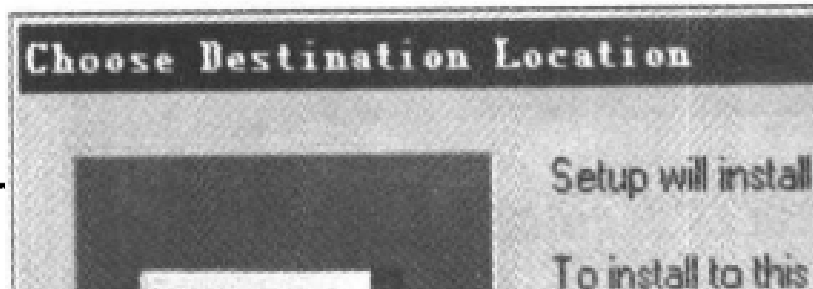
图 1-3　Map

（7）单击“Next”按钮，安装向导请用户选择在 Windows 桌面的“开始”菜单中快速启动 Maple V 系统的程序文件夹，如图 1-4 所示；

（8）单击“Next”按钮，系统开始安装 Maple V，如图 1-5 所示；

（9）之后单击“Finish”按钮，即完成 Maple V 系统的安装。

至此，Maple V 系统安装完毕。



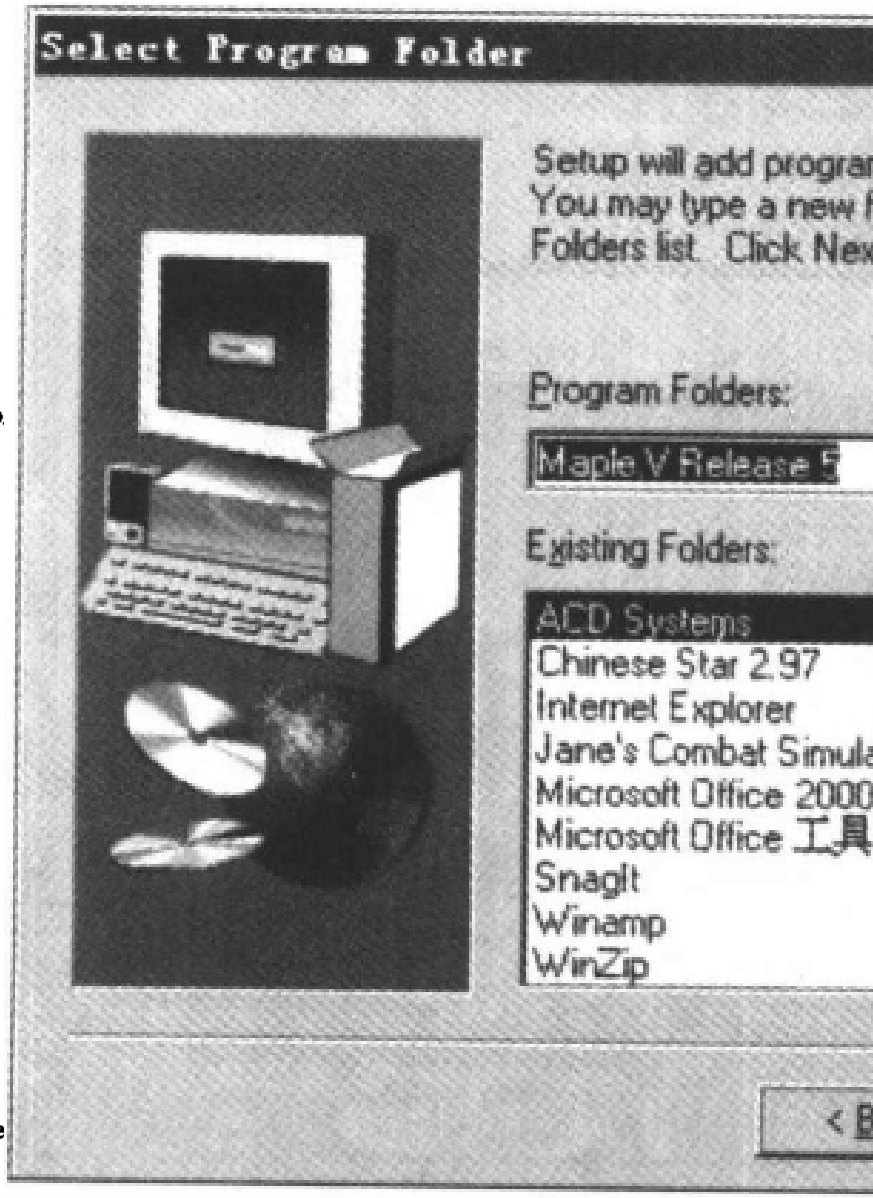
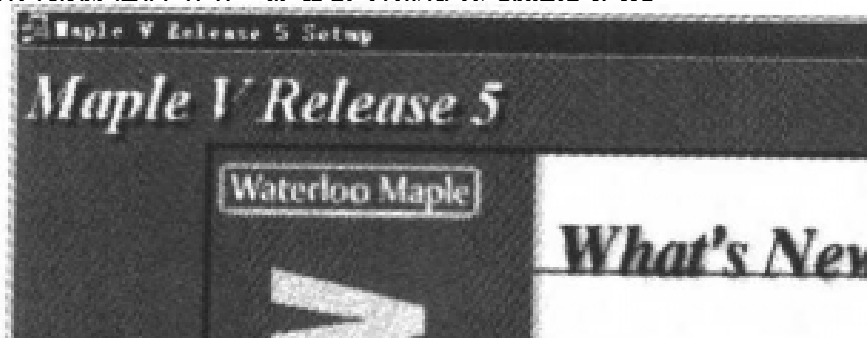


图 1-4 Map

图 1-5 Maple

1.3 Maple V 的用户界面

在操作系统上成功地安装了 Maple V 后，用户就可以启动 Maple V 进行工作了。Maple V R5 提供了命令行界面和图形用户界面(GUI)两种运行方式，本书主要讲述在 Maple V R5 的系统集成界面上进行的操作。



本节首先介绍 Maple V R5 系统的目录结构和命令行界面，接着介绍 Maple V R5 系统集成界面的组成，最后简单介绍 Maple V 系统的退出。

1.3.1　Maple V 系统的目录结构

在开始使用 Maple V 系统之前，建议用户了解一下 Maple V 系统的目录结构，这一点对于 Maple V 系统的高级用户来说是很有必要的。可以在 Maple V 系统的安装路径下查看 Maple V 系统的目录结构。表 1.1 给出了 Maple V R5 系统在 MS Windows 操作平台上安装后的目录结构。

表 1.1　　**Maple V 系统的目录结构**

子目录	包含文件的功能
\AFM	字体信息的文件
\BIN.WIN（Windows 31/32 环境） \BIN.WNT（Windows NT 或 Windows 95/98/2000 环境）	系统运行文件
\ETC	LaTeX（一种数学排版语言）的说明文件和样式文件
\LIB	Maple V 系统的库文件
\LICENSE	Maple V 的许可信息文件
\UPDATE	用来存贮 Maple V 库文件的更新信息

1.3.2　Maple V 系统的命令行界面

相对于完整的 Maple 数学引擎来说，命令行界面是一种小而有效的基于文本的运行方式。命令行界面特别适用于在速度较慢或者内存有限的计算机上求解大型的数学问题。

1. *命令行方式的启动与退出*

用户可以用两种方法启动命令行方式：

- 在 Windows 95/98/2000 或 Windows NT 的“开始”菜单的“程序”子菜单中单击“Command Line Maple”图标；
- 在 Windows 资源管理器里 Maple V 系统安装目录的 “\BIN.WNT”子目录中单击可执行文件“cmaple.exe”。

Windows 3.x 用户可以在 Windows 程序管理器中双击“Command Line Maple”图标，或者直接运行可执行文件“mapledos.exe”。如图 1-6 所示是 Maple V 系统的命令行界面。

退出命令行方式时只需在命令提示符（右尖括号）“>”后键入命令“quit”即可。如果这时没能退出，表明前一个命令的输入没有结束。用户需要首先键入“;”结束前面的命令输入，然后再键入命令“quit”。

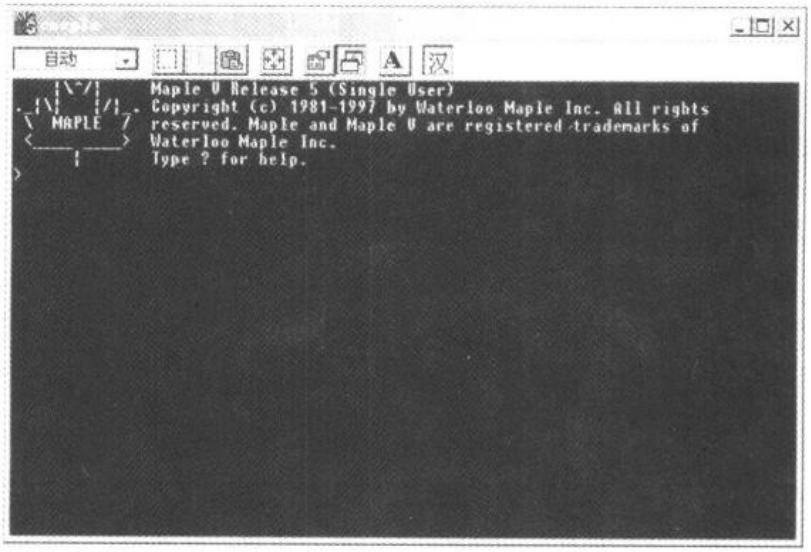


图 1-6 Maple V 系统的命令行界面

2. 命令行方式的使用

运行命令行方式几乎可以实现 Maple V 系统的所有功能。命令行界面中的所有操作都要靠键入相应的命令来完成，而在集成界面中可以使用鼠标进行各种快捷操作。命令的使用方法和在集成界面工作空间中的使用是相同的。用户可以键入“?topic”来获得主题“topic”的帮助信息，其中“topic”为所感兴趣的主题。

有关命令行方式的详细信息，可以阅读命令行方式的帮助文件“cmdline.txt”。

1.3.3 Maple V 的系统集成界面

启动 Maple V 的系统集成界面也有两种方法：

- 在 Windows 95/98/2000 或 Windows NT 的“开始”菜单的“程序”子菜单中单击“Maple V Release 5”图标；
- 直接执行对应的应用程序文件“wmaple.exe”。

Maple V 的系统集成界面由标题栏、菜单栏、工具栏、上下文关联栏、工作空间、状态栏等要素组成，如图 1-7 所示。MS Windows 用户对这种图形用户界面（GUI）是很熟悉的。这些基本要素通常是与当前的操作对象相关的；当操作对象改变时，其中的菜单栏、工具栏及上下文关联栏等都会随着发生相应的改变。下面介绍的是在通常的命令输入模式下的系统集成界面。

1. 标题栏

标题栏位于 Maple V 集成界面的顶部，用来显示该应用程序的标题。其中还包括了窗口左上角的控制菜单框和右上角的快捷按钮（最小化、最大化或还原和关闭），它们的使用方法与 Windows 程序中的习惯用法一致，这里不再赘述。

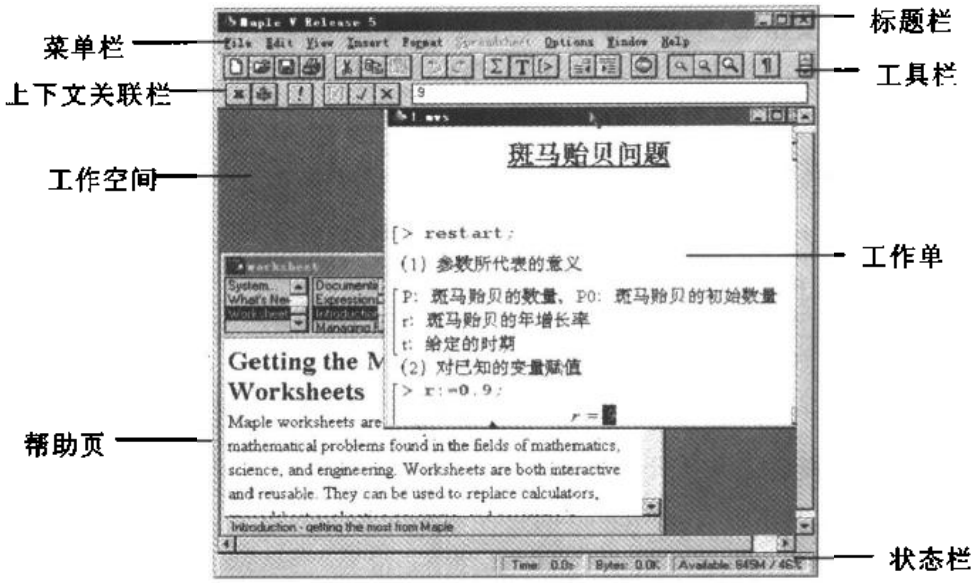


图 1-7　Maple V 的系统集成界面

2. 菜单栏

菜单栏位于标题栏的下面，包括“File”、“Edit”、“View”、“Insert”、“Format”、“Spreadsheet”、“Options”、“Window”和“Help”等 10 个主菜单。每个主菜单都包含着相应的下拉子菜单，每个子菜单的用法在以后的章节中会陆续涉及。这里将简单介绍 9 个主菜单所实现的主要功能。

“File”（文件）菜单：主要是有关 Maple V 工作单文件的创建、打开、保存和打印等操作，另外还包含对当前界面环境的保存等。

“Edit”（编辑）菜单：主要用于工作单文件的编辑，同时还包含有关 OLE 对象的操作等。

“View”（视图）菜单：设置窗口中工具栏、上下文关联栏、状态栏、调色板等要素及工作单中程序段标志的显示，还包含有关于书签和区域（Section）的操作。

“Insert”（插入）菜单：用于插入程序段、图形、电子表格、段落、区域、超链接等对象。

“Format”（格式）菜单：设置工作单中字符、文本、段落等要素的显示样式。

“Spreadsheet”（电子表格）菜单：包含与电子表格有关的一些操作。

“Options”（选项）菜单：用来选择输入输出模式，设置打印质量。

“Window”（窗口）菜单：用来选择显示窗口，设置当前窗口的显示形式；可实现关闭所有窗口或关闭所有帮助页。

“Help”（菜单）：用来选择获得帮助的方法，并可获得 Maple V 的版本信息。

3. 工具栏

菜单栏下面是可以实现常用功能的位图按钮工具栏。和所有 Windows 应用软件一样，

工具栏使得直接在某个按钮上单击即可实现相应的功能，而不必打开下拉菜单从中寻找相应的指令。如果用户打开了冒泡帮助选项（从“Help”菜单中选择“Balloon Help”选项），当鼠标位于某个快捷按钮上时，该按钮的帮助信息会以冒泡的形式显示。

Maple V 把工具栏按不同的功能分成了 9 部分，下面将分别介绍各个部分实现的功能。

 ：工作单文件的新建、打开、保存或打印。

 ：对所选内容进行剪切、复制或粘贴。

 ：取消或重复前一个操作。

 ：控制光标后的输入内容：惰性 Standard Math 表达式、文本或新的程序段。

 ：对区域的操作，使所选内容从区域中分离或包含在某个子区域中。

 ：中止命令的执行。

 ：控制显示的比例为 100%、150%或 200%。

 ：控制空格、换行等空白字符的显示。

 ：改变工作单窗口的大小使之充满当前的工作空间。

4. 上下文关联栏

上下文关联栏可以看作工具栏的一部分。根据当前操作对象的不同，上下文关联栏有不同的显示类型，从而实现不同的功能。图 1-8 中给出的是命令输入模式下的上下文关联栏和文本输入模式下的上下文关联栏。

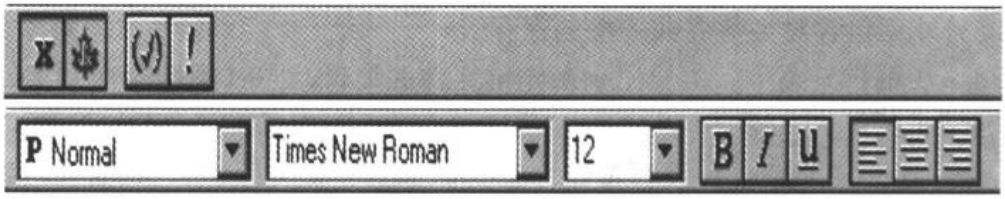


图 1-8 上下文关联栏的两种特例

5. 工作空间

工作空间是 Maple V 工作单的容器。Maple V 工作单是由 Maple V 系统的图形用户界面所创建的文档。工作单具有类似笔记本的功能，是用户进行命令输入及结果输出和进行相关描述的场所。在 Maple V R5 的工作单里，用户可以执行各种科学计算，可以描述问题的求解过程等。在工作空间里，用户可以同时打开多个工作单，各个工作单之间可以相互切换。帮助页是 Maple V 显示在线帮助的页面，属于一种特殊的工作单。关于工作单的基本操作，将在 2.3 节中介绍；关于 Maple V 的帮助系统，将在 2.5 节中介绍。

6. 状态栏

位于 Maple V 运行窗口底部的状态栏显示了 Maple V 运行过程中的诊断信息和其他

系统信息。如果在工具栏的快捷按钮上按下鼠标左键，在状态栏的前半部分将显示出该快捷按钮的作用；同样，如果鼠标位于菜单的直接选项上，该选项的作用也显示在状态栏的前半部分，这也是 Maple V 所提供的帮助的一部分。

通常，工具栏、上下文关联栏和状态栏都处于显示状态。用户可以通过“View”菜单中的三个选项“Tool Bar”、“Context Bar”和“Status Line”来控制工具栏、上下文关联栏和状态栏的显示与隐藏。如图 1-9 所示，其中的三个选项都处于选中状态，因此系统集成界面中能够看到相应的工具栏、上下文关联栏和状态栏。

图 1-9　“View”菜单的下拉菜单

默认情况下，在“File”菜单的下拉菜单中，“Auto Save Settings”选项处于选中状态，如图 1-10 所示。因此，用户对系统集成界面的环境重新设置后，Ma
改后的设置。当用户再次启动 Maple V 后，系统集成界面的设置与
同。用户也可以取消“Auto Save Settings”选项，当要保存所作的修
单的下拉菜单中选择“Save Settings”选项即可。

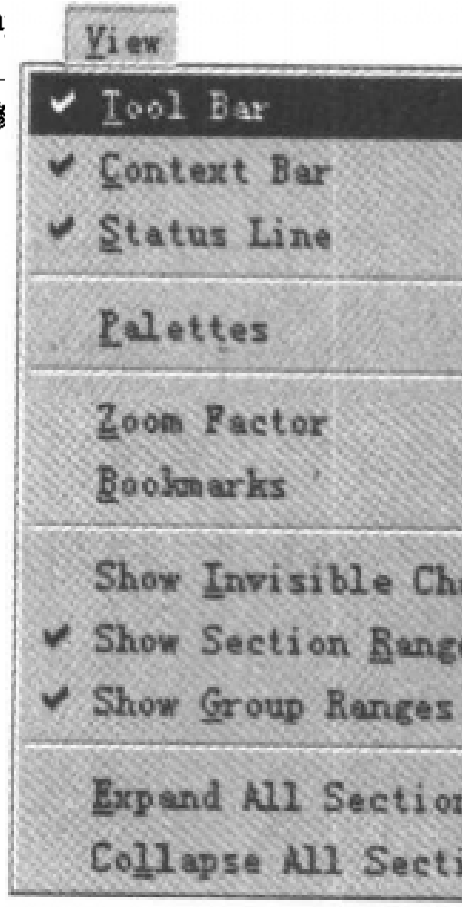


图 1-10　“File”菜单的下拉菜单

另外，与菜单栏对应，Maple V 也提供了上下文菜单栏。当用户在工作单中工作时，单击鼠标右键，Maple V 会根据当前的工作状态及光标所在的对象弹出相应的上下文菜单。用户可以在上下文菜单中选择不同的子菜单完成相应的不同操作。Maple V 的上下文菜单也是系统集成界面的组成部分，使得用户不必从菜单栏中选择即可实现相应的菜单操作。

1.3.4 Maple V 系统集成界面的退出

和其他 Windows 应用软件一样，Maple V 系统集成界面的退出也比较简单。这里给出四种方法：

- 使用快捷键：“Alt” + “F4” 或 “Alt” + “F” + “X”；
- 单击 Maple V 系统集成界面标题栏右上角的快捷按钮 ；
- 从 “File” 菜单的下拉菜单中选择 “Exit” 子菜单；
- 单击 Maple V 系统集成界面标题栏左上角的控制图标，从下拉菜单中选择 “关闭” 子菜单。

第 2 章　Maple V 的基础知识

在上一章中，读者对 Maple V 及其系统集成界面有了一个大略的认识。在开始使用 Maple V 进行科学计算之前，有必要了解 Maple V 在输入输出方面的一些基本知识。工作单是 Maple V 用户进行科学计算的场所，其中包含了丰富的文档对象，使用户可以执行各种较为复杂的文档操作。程序段作为工作单中的基本计算单元，在执行科学计算的过程中充当了非常重要的角色。完善的在线帮助系统为用户快速掌握 Maple V 提供了方便。

本章首先介绍 Maple V 命令的输入输出，接着介绍 Maple V 的工作单和程序段，最后介绍 Maple V 的在线帮助系统。

2.1　Maple V 命令的输入

用户输入命令，Maple V 执行相应的操作，这是使用 Maple V 进行科学计算时用户与 Maple V 实现交互的基本方式。这里的命令包括任意合法的语句，包括 Maple V 自身命令的调用以及由其他合法表达式构成的语句等。

本节介绍与命令输入有关的一些内容，包括基本的语法规则、输入模式的选择、输入样式的改变以及 Maple V 的注释。在 2.1.3 节中介绍了 Maple V R5 最新提供的一种辅助输入工具——模拟键盘。

2.1.1　Maple V 的基本语法规则

当 Maple V 启动运行后，Maple V 自动打开一个具有临时名字“Untitled（1）”的空白工作单，如图 2-1 所示。工作单的窗口由标题栏和用户区组成，标题栏包含有左上角的控制菜单框和右上角的控制按钮，并在控制菜单框后面显示该工作单的文件名。

图 2-1　Maple V 自动打开的空白工作单

在新打开的空白工作单中，光标位于用户区的首行，光标的左边是 Maple V 的命令提示符“>”，命令提示符的左边是方括号“[”，如图 2-1 所示。在 Maple V 中，用方括号把命令的输入和输出标识为一个整体，称为一个程序段，以命令提示符开头的行也称为命令输入行。在工作单的右侧出现滑动条，当工作单窗口较小时下侧也会出现滑动条，调整滑动条中的滑动块可以改变当前窗口中的显示内容。

下面介绍了 Maple V 命令的一些基本语法规则和基本的续行操作。

1. 基本语法规则

Maple V 的科学计算功能主要是以命令输入的方式来实现的。Maple V 的命令有自己的使用规则和语法。在使用 Maple V 进行科学计算之前，首先要了解 Maple V 命令使用的基本规则。下面给出了利用 Maple V 进行科学计算时的一些基本语法规则：

- Maple V 的命令在提示符“>”的右边键入，每行命令要以分号“;”或冒号“:”结尾。
- 在命令输入完毕后按下“Enter”键，Maple V 将立即执行该行命令。
- 如果命令以分号结尾，Maple V 将在下一行给出相应的输出结果，并把光标移到下一个程序段的开始行；如果命令以冒号结尾，Maple V 执行命令但不显示输出结果，光标直接移到下一个程序段的开始。
- Maple V 中乘号为星号“*”，两项相乘时乘号不能省略。
- 对变量赋值时用赋值运算符“:=”，而不是通常的等号“=”。
- 除号为斜杠符号“/”，$\frac{a}{b+c}$ 的输入格式为：**a/(b+c)**。
- 乘方运算符为：“^”或“**”，负指数必须包含在圆括号中。
- 函数的参数必须用圆括号界定，数组或矩阵的下标用方括号界定。
- 变量不需要预先定义，严格区分字母的大小写。
- 在运算符和操作数之间可以插入空格或者其他空白字符，但在运算符和标识符内部不能插入空格或其他空白字符。
- 三个环境变量“%”、“%%”和“%%%”，分别代表当前工作空间最近三次的非空输出结果。

下面给出了 Maple V 运算的几个例子，内容涉及字符串、数的运算、方程的求解和图像的绘制，可使读者初步认识 Maple V 的工作方式。在这些例子中，每行命令都以分号结尾，因此 Maple V 在输入的下一行即给出相应的输出，并把光标移到下一个程序段的开始。

```
>"I am a string";
```

"I am a string"

```
>(3+4)*12;
```

```
>%/2^2;
```

$$21$$

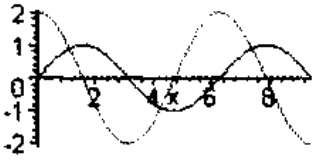
```
>evalf(pi,50);
```

3.14159265358979323846264338327950288419716939937 51

```
>root1:=solve(x^3-2*x+1=0);
```

$$root1 = 1, \frac{1}{2}\sqrt{5} - \frac{1}{2}, -\frac{1}{2} - \frac{1}{2}\sqrt{5}$$

```
>plot({sin(x),2*cos(x)},x=0..3*Pi);
```



2. 基本的续行操作

在 Maple V 命令的输入过程中，当命令的长度超过当前工作单的宽度时，Maple V 会自动换行，下一行的开始不再有命令提示符，表明该行是上一行的继续。另外，用户可以使用在特定的位置强制换行，把命令分为多行输入。

通称，用户习惯于在需要换行的位置直接按下“Enter”键。这时，Maple V 会在当前行下边插入新的命令行，光标位于该行命令提示符的右边，同时在程序段的最后给出下面的警告信息：“Warning, premature end of input”。用户可以忽略该警告信息，在下一行中继续输入命令的余下部分。命令输入完毕并以“:”或“;”结束时，按下“Enter”键，警告信息会自动消失。看下面的例子：

```
>a:=1+x
>
warning,premature end of input
```

```
>a:=1+x
>+x^2;
```

$$a := 1 + x + x^2$$

用户还可以使用组合键“Shift”+“Enter”键来实现换行。与直接按下“Enter”键不同的是，Maple V 在当前行下边插入新命令行时左边没有命令提示符，光标位于该行的开头，用户可以继续输入命令的余下部分。

```
>a:=1+x
```

在下面的例子中，把赋值语句“a:=1+x+x^2”分成了两行输入：

```
> a:=1+x
  +x^2;
```

$$a := 1 + x + x^2$$

2.1.2 Maple V 的命令输入模式

命令的输入模式指的是命令中数学表达式、特殊符号等的显示格式。Maple V 提供了 Maple Notation 和 Standard Math 两种命令输入模式。下面将简单介绍这两种输入模式：

- Maple Notation：输入是一维的，即命令中的元素都位于同一行中。命令必须以分号“;”或冒号“:”结尾，然后按“Enter”键结束。
- Standard Math：输入是二维的，即命令中的元素可以不在同一行中，分式、微分、积分、矩阵运算等都可以按照数学中的标准符号进行输入。命令不必以分号“;”或冒号“:”结尾，通常要按两次“Enter”键结束。

表 2.1 对两种输入模式作了简单对比。

表 2.1　　Maple V 的两种命令输入模式

输入模式	命令提示符	输入举例
Maple Notation	[>	(a+b)/c
Standard Math	[> ?	$\frac{a+b}{c}$

要改变当前的命令输入模式，可用下面的三种方法：

- 单击上下文关联栏中的快捷按钮 。
- 在当前输入行中单击鼠标右键，从弹出的菜单中选择或取消“Standard Math”选项，如图 2-2 所示，此时鼠标所在的行采用的是 Standard Math 输入模式。

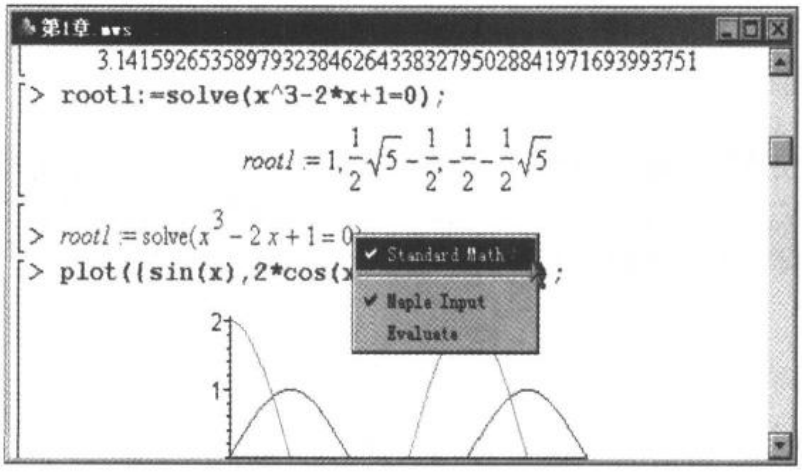


图 2-2　从弹出菜单改变输入模式

- 从“Options”菜单的子菜单“Input Display”弹出的两个选项中选择所要采用的输入模式。

前两种方法只是把当前行的输入模式转化为另一种，不影响其他行的输入模式；最后一种方法则不会立即生效，当鼠标转入新的一行时生效。

“Maple Notation”是 Maple V 中最基本的命令输入模式，Maple V 首次启动运行后，默认的命令输入模式为“Maple Notation”。本书主要采用 Maple Notation 输入模式介绍命令的使用和输入规则，在 2.1.1 节中所给出的几个基本的语法规则基本上都是对这种命令输入模式而言的。

在 Maple Notation 输入模式下，命令的输入都具有文本的形式，因此用户可以对这些命令进行常见的编辑操作，如复制、剪切、删除、粘贴等。这些操作和在其他应用软件中的操作类似，并采用了 Windows 默认的快捷键，这里不再介绍。

2.1.3　Maple V R5 的模拟键盘

模拟键盘是 Maple V R5 新增加的功能，模拟键盘使得用户可以像使用 MS Word 中的公式编辑器一样输入希腊字符、常用的数学表达式及各种结构的矩阵。Maple 提供了符号键盘、表达式键盘和矩阵键盘三种模拟键盘，下面简单介绍这三种模拟键盘的作用：

- 符号键盘：如图 2-3 所示。其中给出了全部 24 个大小写的希腊字符和四个常量 e、I、π、∞。值得注意的是，这 24 个希腊字符和后三个常量对应的表示符都是 Maple V 的保护名，用户不能对这些标识符执行其他赋值操作。

图 2-3　Maple V R5 的符号键盘

- 表达式键盘：如图 2-4 所示。其中包含了加、减、乘、除、乘方、开方、求和、连乘积、积分、极限、微分、方程、赋值、下标、阶乘、求绝对值和 6 个常用函数等在内的 24 个常用数学运算符号。

图 2-4　Maple V R5 的表达式键盘

- 矩阵键盘：如图 2-5 所示。其中包含了 16 种矩阵结构的数学描述符号。

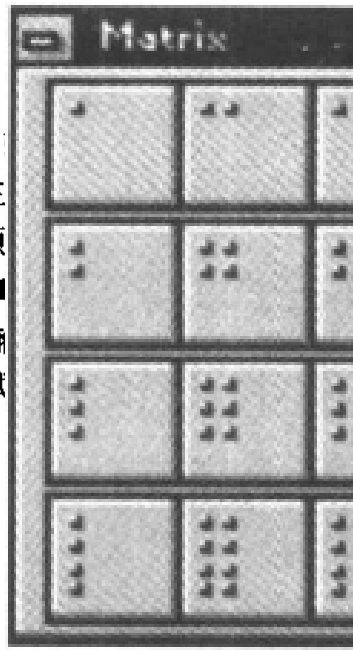


图 2-5 Maple V R5 的矩阵键盘

这三种模拟键盘在默认情况下是隐藏的，用户可以从“View”菜单的“
菜单中分别选择“Symbol Palette”、“Expression Palette”、“Matrix Palette”三
示对应的符号键盘、表达式键盘或矩阵键盘，如图 2-6 所示，其中的三个选项
状态。如果要隐藏模拟键盘，只需单击该模拟键盘标题栏左上角的控制图标 ■

模拟键盘在两种输入模式下都可以使用，但更多是用在“Standard Math”
使用时，用户只需在模拟键盘上单击相应的按钮即可。对应于不同的输入模式
在命令输入行的显示也不同。

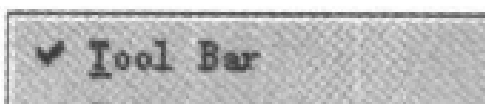


图 2-6 模拟键盘的显示控制

2.1.4 Maple V 的命令输入样式

命令输入样式指的是命令输入的字体类型、颜色、大小及其他属性。在默认情况下，Maple V 的命令及注释以红色显示，字体类型为“Courier New”黑体。

用户可以根据自己的喜好改变命令输入样式。下面给出了改变输入样式的步骤：

（1）从“Format”菜单的下拉菜单中选择“Style”子菜单；

（2）在出现的“Style Management”对话框（如图 2-7 所示）中的“Style”列表中包含了许多样式名，从中选择“Maple Input”项；

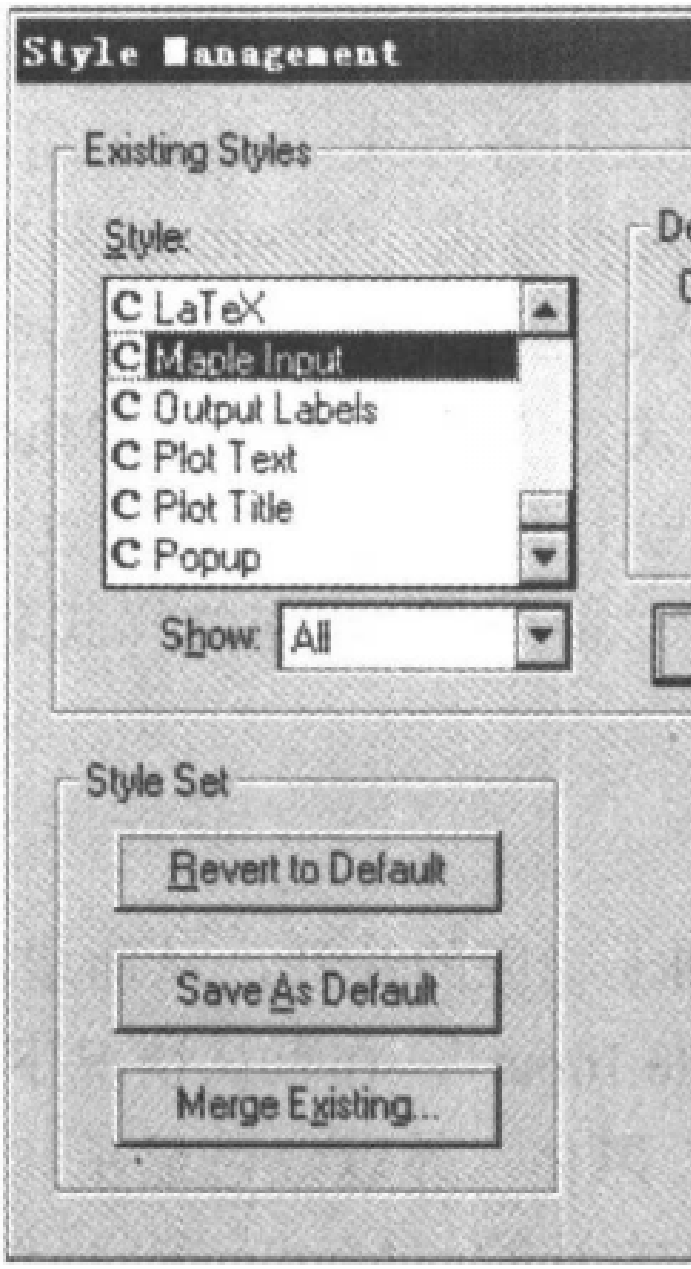


图 2-7　"Style Managemer

（3）在右侧的"Description"域中可以看到当前"
+ bold + colored"；如果要改变这种设置，单击"Mo

（4）在出现的"Character Style"对话框（如图 2
的字体类型、大小、属性、颜色等性质，在右下角的
例。设置完后单击"OK"按钮确定；

图 2-8　"Character Style"对话框

（5）如果想把修改后的设置存为 Maple V 的默认设置，在"Style Management"对话

框的“Style Set”域中单击“Save As Default”按钮即可。

在默认情况下，Maple V 命令的输入采用左对齐方式，当命令超过当前工作单窗口的宽度时自动换行。用户可以改变当前行输入的对齐方式，“Format”菜单中的“Left Justify”、“Center”、“Right Justify”，分别对应着左对齐、居中、右对齐三种对齐方式。

工作单中的每种对象都有默认的显示样式，对应着“Style Management”对话框的“Style”域中各自的样式名。用户可以按照与上面类似的步骤来改变这些对象的样式，只需在“Style Management”对话框的“Style”域中选择相应的样式名即可。

2.1.5 Maple V 的注释

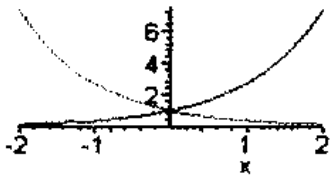
用户可以为输入的命令或者进行的操作加上相应的注释，从而增加每一步工作的可读性。Maple V 把注释和命令输入通称为“Maple Input”，在默认情况下，Maple V 注释和命令输入是一样的，字体类型为“Courier New”黑体，并以红色显示。

在 Maple V 中，用“#”标志注释部分的开始。Maple V 将忽略命令输入过程中的“#”号及同一行中“#”号后面的内容。值得注意的是，在命令输入过程中如果在注释的中间执行了强制换行操作，新行中的注释也必须以“#”开始。下面给出了两个包含注释的命令。

```
> s:="I am a string";#定义了一个字符串
```

$$s := \text{"I am a string"}$$

```
> plot({exp(x),exp(-x)},x=-2..2);
  # 同时绘制指数函数和负指数函数的图像
```



在注释或者其他文本中也可以输入中文，但中文可能暂时无法正常显示；在中文输入完毕后再多输入一个空格字符，然后再按“Backspace”键删除该空格，即可以看到正常显示的中文。当然，用户也可以采用其他方法如刷新屏幕等使中文显示正常。

2.2 Maple V 命令的输出

Maple V 合法命令产生的结果称为 Maple V 输出。Maple V 输出的对象包括字符串、标识符、表达式、方程、不等式、图形及各种数学符号。从广义来讲，由各种不合法的输

入所产生的错误和警告信息也属于输出的范畴。

本节介绍了与 Maple V 输出有关的内容，包括 Maple V 的输出模式、输出对象以及 Maple V 的错误和警告等。

2.2.1　命令的输出模式

Maple V 提供了 4 种输出模式供用户选择："Maple Notation"、"Character Notation"、"Typeset Notation"、"Editable Math"。其中，"Editable Math" 是 Maple V 的默认输出模式。用户可以从 "Options" 菜单的子菜单 "Output Display" 中选择要使用的输出模式，如图 2-9 所示。

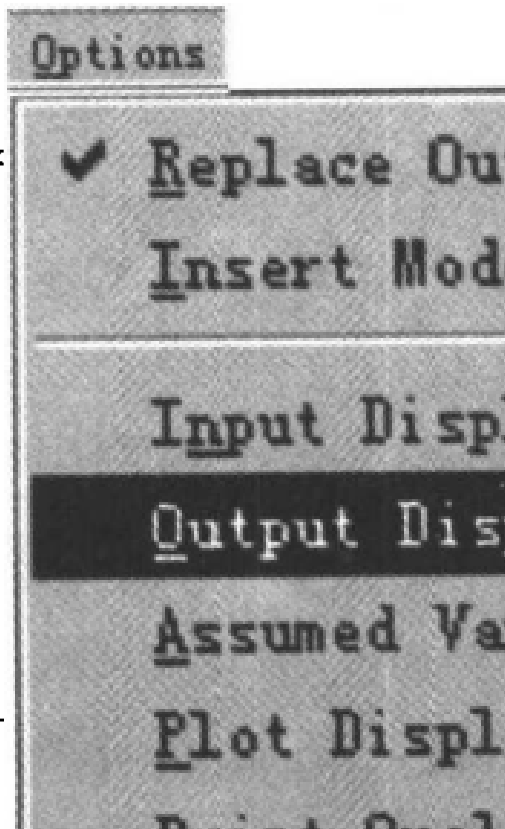
图 2-9　输出模式的选择

下面以 Maple Notation 输入模式下的一个积分恒等式为例，依次给出 Character Notation、Maple Notation、Typeset Notation、Editable Math 四种输出模式下不同的输出形式，读者可以从中发现四种输出模式之间的差别。

```
> Int(x^2*sin(x)^2,x)=int(x^2*sin(x)^2,x);
    /
    |
    |  x^2 sin(x)^2 dx=x^2 (-1/2 cos(x) sin(x)+1/2 x
    |
   /

        -1/2×cos(x)^2 +1/4 cos(x) sin(x)+1/4×


        -1/3 x^3
```



```
> Int(x^2*sin(x)^2,x)=int(x^2*sin(x)^2,x);
Int(x^2*sin(x)^2,x)=x^2*(-1/2*cos(x)*sin(x)+1/2*x)-
1/2*x*cos(x)^2+1/4*cos(x)*sin(x)+1/4*x-1/3*x^3
```

```
> Int(x^2*sin(x)^2,x)=int(x^2*sin(x)^2,x);
```

$$\int x^2\sin(x)^2\,dx = x^2\left(-\frac{1}{2}\cos(x)\sin(x)+\frac{1}{2}x\right)-\frac{1}{2}x\cos(x)^2+\frac{1}{4}\cos(x)\sin(x)+\frac{1}{4}x-\frac{1}{3}x^3$$

```
> Int(x^2*sin(x)^2,x)=int(x^2*sin(x)^2,x);
```

$$\int x^2\sin(x)^2\,dx = x^2\left(-\frac{1}{2}\cos(x)\sin(x)+\frac{1}{2}x\right)-\frac{1}{2}x\cos(x)^2+\frac{1}{4}\cos(x)\sin(x)+\frac{1}{4}x-\frac{1}{3}x^3$$

通过上面的例子，读者可以看到最后两种输出模式的输出结果形式完全一样，二者之间的差别在于：在“Typeset Notation”输出模式下，用户只能对输出的对象进行整体的选择、复制、粘贴等编辑操作；而在“Editable Math”输出模式下，用户可以对输出的对象进行分解，可以选择其中的各个部分并执行复制、粘贴等编辑操作；

因为“Editable Math”是一种更受用户欢迎的输出模式，所以本书后面的例子都是在“Editable Math”输出模式下完成的。

2.2.2 Maple V 的输出对象

Maple V 的输出对象大致可以分为两类，第一类包括标识符、字符串、表达式、方程、不等式、各种数学符号等非图形类的输出，这里通称为符号类输出对象，在 Maple V 中称为“Maple Output”；第二类主要包括 Maple V 绘制的各种图形，这里称之为图形类输出对象，在 Maple V 中称为“Maple Plot”。关于 Maple V 的图形类输出对象，将在绘图一章中重点介绍。本小节主要介绍符号类输出对象的显示样式和有关操作。

1. 符号类输出对象的显示样式

默认情况下，符号类输出对象通常以蓝色显示，其中字符的字体为“Courier New”正常体。用户可以按照与 2.1.4 节中改变命令输入样式类似的步骤，重新设置 Maple V 的输出样式。

默认情况下，Maple V 的输出都位于输出行的中间，即对齐方式为居中。当工作单窗口的大小改变时，Maple V 将自动调整输出的位置。当不能在一行输出时，Maple V 会在合适的位置自动换行。和 Maple V 的命令输入一样，用户也可以改变当前光标所选输出对象的对齐方式，使其左对齐、居中或者右对齐等。

2. 符号类输出对象的基本编辑

在 Editable Math 输出模式下用户可以选取符号类输出对象的局部或全部，进行基本的编辑操作或者其他操作。基本的编辑操作包括对输出对象的选择、删除、复制、剪切和粘贴等。

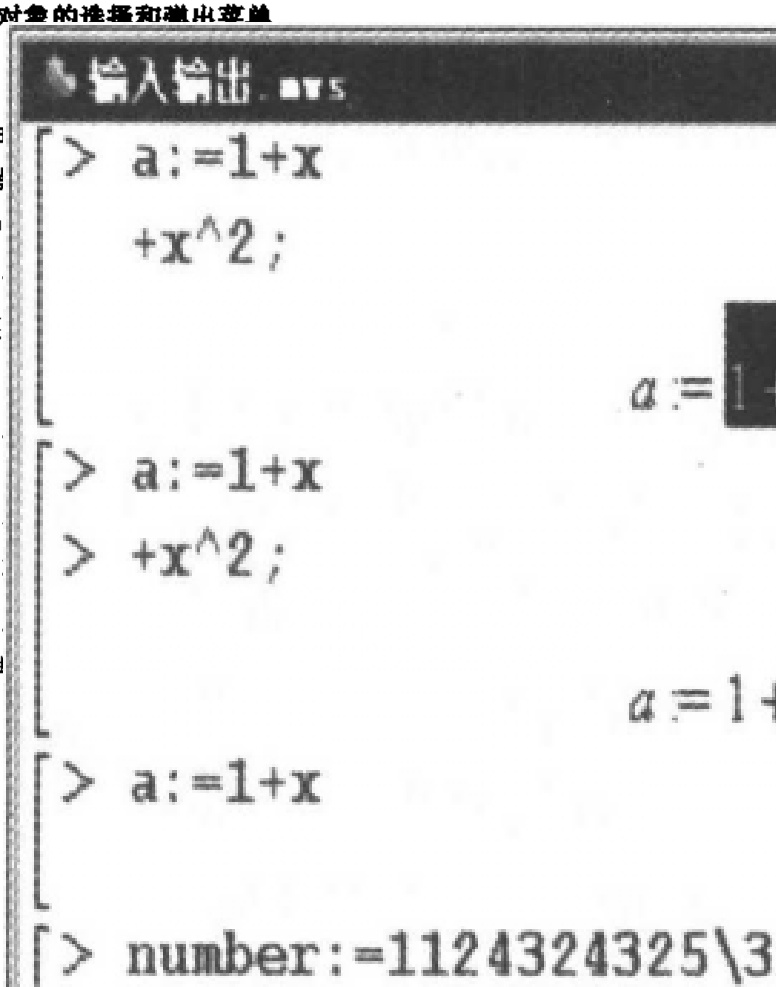
进行各种编辑前，首先应选定要编辑的对象。对于符号类对象，用户可以选择对象的部分，只要用鼠标在输出对象上滑动，鼠标经过的部分即被选中，选中的对象或选中的部分处于反显状态。在图 2-10 中，选定了多项式 $1+x+x^2$。

图 2-10　符号类输出对象的选择和弹出菜单

对选中的对象，用户可以进行删除、复制、
默认的快捷键来完成，如“Delete”或者“Ba
键执行复制操作、“Ctrl”+“X”键执行剪切操
者使用 Maple V 工具栏中的快捷按钮或相应的

值得注意的是，符号类输出对象只能被整
类对象局部的删除操作。如果选中的是符号类
切操作这时与复制的效果相同。

对符号类输出对象进行复制或者剪切操作
他类型的文档中。符号类输出对象粘贴后的显
于粘贴时光标所处的位置。如果光标位于命令
局部转化为相应的简单命令，用户执行该命令
标位于非执行的文本段内，符号类输出对象的
定的多项式 $1+x+x^2$ 复制后，分别粘贴到工作
不同的显示情况。



[>1+x+x^2

[$1+x+x^2$

Maple V 中的符号类输出对象经复制后，可以粘贴到其他文档处理软件的文档中，当然这里的文档应该能支持图像、公式等复杂对象，例如 Microsoft Word，而 Windows 自带的记事本、写字板则不在此列。但直接粘贴不能得到和 Maple V 中相同的显示格式，需要使用特殊的粘贴方法；以 Microsoft Word 为例，需要从“编辑（Edit）”菜单中选择“选择性粘贴（Special Paste）”子菜单，粘贴后的内容在 Microsoft Word 中以外来图片的形式显示。同时，Microsoft Word 不能接受 Maple V 符号输出类对象局部的粘贴；当用户试图粘贴某一符号类对象的局部时，Microsoft Word 默认选中的是整个输出对象。在本书的写作过程中，用到的很多数学公式和符号都是采用这种方法实现的。

3. 符号类输出对象的其他操作

通常，在选定的对象上单击鼠标右键，会弹出与当前对象相关的上下文菜单，其中包含了对当前对象所能进行的操作选项。通过选择弹出菜单中不同的选项，用户可以对所选的对象进行其他各种可能的操作，而不必手动输入有时相当复杂的命令。

在图 2-10 中，显示的是选中多项式 $1+x+x^2$ 后的弹出菜单，其中包含了复制、微分、积分、近似和绘图等多个选项；当选定类似的代数表达式时，弹出的菜单中通常包含类似的选项。下面给出了选择弹出菜单中“plots”子菜单的“2D-Plot”选项后的输出结果，这是一个绘图选项；Maple V 在当前程序段下边插入了一个新的程序段，其中包含了命令“**smartplot(1+x+x^2);**”和相应的图形输出，如图 2-11 所示。**smartplot** 命令是 Maple V 中的一个绘图工具，读者可以用**?smartplot** 获得相关的帮助信息。

图 2-11　利用弹出菜单中的绘图命令

当光标选中符号类输出对象的部分或全部时，按下“Enter”键，在当前程序段的下边将插入一个新的程序段；该程序段中包含一行与所选的符号类输出对象整体对应的命令和相应的输出。当然，这时的输出和原来的一样。

2.2.3　错误和警告

错误和警告是 Maple V 中一种特殊的输出对象，通常是由不合法的命令或者输入所引发的。Maple V 通过错误信息或警告信息告诉用户当前的输入中所存在的错误，这些信息通常是浅显易懂的，即使是 Maple V 的新用户也可以通过这些信息发现输入中的错误。这也是 Maple V 用户界面友好的一个表现。

Maple V 把警告信息也归入“Maple Output”中，因此默认情况下以蓝色显示，字体类型为“Courier New”正常体；警告信息通常具有“Warning, …”的形式。Maple V 把错误信息称为“error”，默认情况下以洋红色显示，字体类型为“Courier New”正常体。

值得注意的是，用户不能对错误信息或警告信息执行删除、剪切等编辑操作。在用户改正了命令输入中的错误后，错误或警告会自动消失。

2.3　Maple V 的工作单

工作单是 Maple V 用户输入各种命令及完成其他工作的场所，其中记录了用户所进行的全部工作。工作单是一个可编辑的而且可再执行的 Maple V 文档，其中提供了丰富的文档对象，使得用户不仅可以在工作单中完成科学计算，而且可以很方便地描述问题求解过程。

本节首先介绍工作单的文件操作，接着简单介绍工作单中的文档对象，最后介绍与文本段有关的一些内容。

2.3.1　工作单的文件操作

工作单作为 Maple V 的基本文档，同样是以文件的形式来保存的。Maple V 启动运行后为用户打开一个新的未命名（Untitled）的空白工作单，如图 2-1 所示。用户也可以再次创建新的工作单；或者打开以前保存的工作单，继续以前的工作；还可以导入其他用户的 Maple 文本文件，在别人工作的基础上继续工作。这些操作都是以与工作单对应的文件为对象的，本小节分别介绍这些基本的操作。

1. 创建新的工作单

和使用 Ms Word 时新建新的 Word 文档一样，新建工作单有下面三种方法：

- 从“File”主菜单的下拉菜单中选择“New”子菜单。
- 使用快捷键“Ctrl”+“N”或“Alt”+“F”+“N”。
- 单击工具栏中的快捷图标  。

2. 打开原有的工作单

用户可以打开以前保存过的工作单继续某个问题的解决，修改其内容或者改变其外

观。Maple V 工作单的文件类型为“Maple Worksheet”，文件名的后缀为“mws”，用户可以按下面的步骤来打开原有的工作单：

（1）从“File”菜单中选择“Open”子菜单，或单击工具栏中的快捷图标  ，或使用快捷键“Ctrl”+“O”或“Alt”+“F”+“O”；

（2）在出现的“打开”对话框（如图 2-12 所示）中选择文件所在的文件夹，确认“列出文件类型”下拉域中选择了“Maple Worksheet”文件类型，从文件列表中选择要打开的文件，或直接键入要打开的文件名；

（3）指定了正确的文件名后，单击“确定”按钮，或直接在文件列表中双击所选择的文件名，即可打开以前保存的工作单。

进行上面的操作时，要确认欲打开的工作单文件是存在的。

图 2-12 “打开”对话框

3. 导入 Maple 文本文件

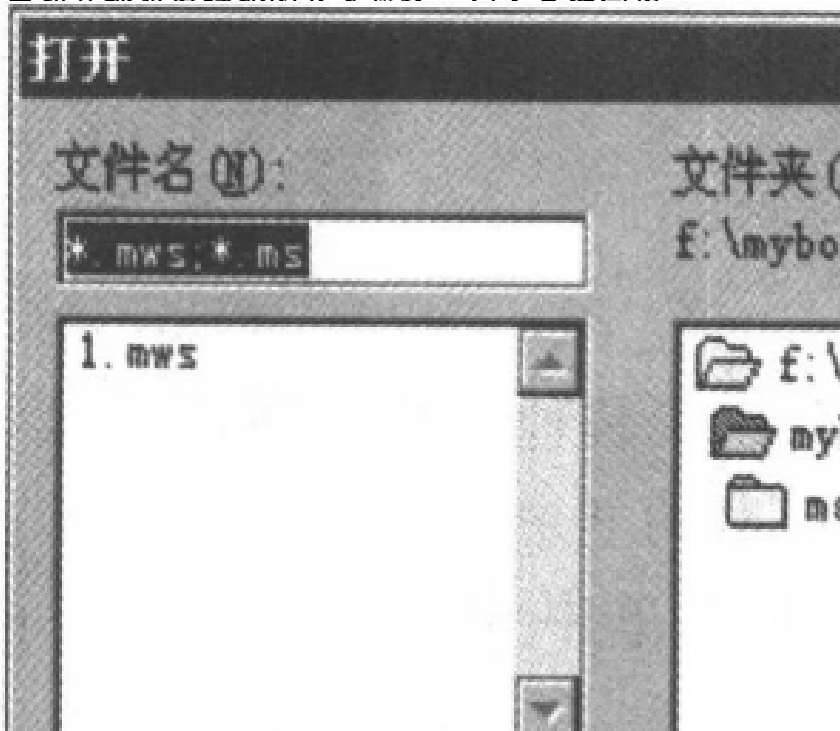
在 Maple V 中，除了可以打开已有的工作单文件外，还可以导入 Maple 文本文件。Maple 文本文件是对工作单中的 Maple 输入、输出和其他对象结构的文本描述。导入其他同事发送的 Maple 文本文件，可以实现知识和学

导入 Maple 文本文件的操作步骤为：

（1）从“File”菜单中选择“Open”
使用快捷键“Ctrl”+“O”或“Alt”+“F”

（2）在出现的“打开”对话框（如图
“列出文件类型”下拉域中选择了“Maple
文件，或直接键入要打开的文件名，单击“

（3）在随后出现的“Text Format Choice
如果导入的是纯文本文件，则选择“Text”，



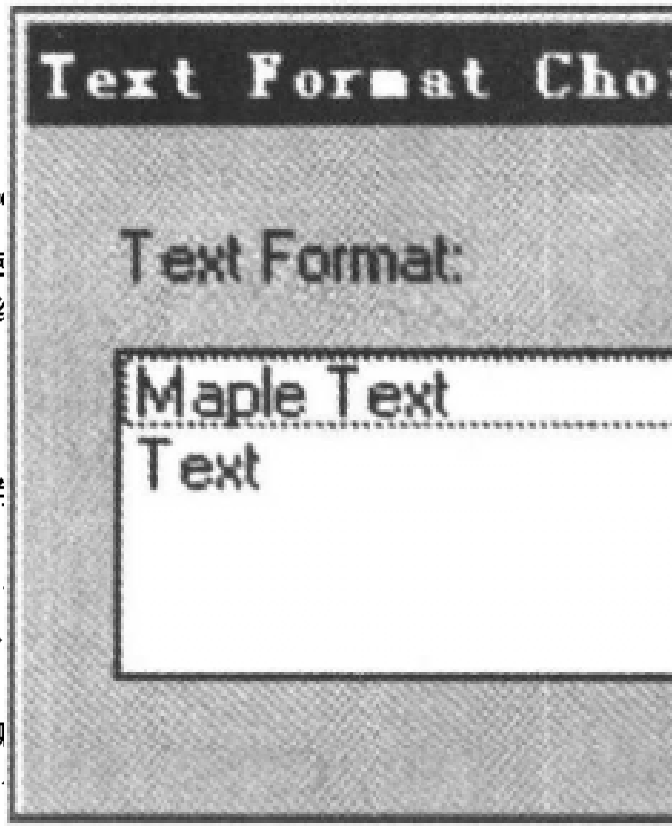


图 2-13　“Text Format Choic

如果导入的是 Maple 文本文件，则将打开一个包
单，用户可以将其另存为工作单文件；如果导入的是纯
含纯文本。

4. 工作单的保存

在用 Maple V 进行工作的过程中，需要及时保存
三种方法：

- 从“File”主菜单的下拉菜单中选择“Save”
- 使用快捷键“Ctrl”+“S”或“Alt”+“F”+
- 单击工具栏中的快捷图标 。

如果是第一次保存目前的工作单，用户需要在出现
图 2-14 所示）中选择保存工作单文件的文件夹并输入

图 2-14　“另存为”对话框

如果用户想把当前的工作单存为另外一个文件作为备份，则可用以下方法：

（1）从“File”主菜单的下拉菜单中选择“Save As”子菜单，或使用快捷键“Alt”+“F”+“A”；

（2）在出现的“另存为（Save As）”对话框中选择保存工作单文件的文件夹并输入合适的文件名。

5. 工作单的关闭

关闭当前不再使用的工作单，可以释放该工作单所占用的系统资源。工作单的关闭也有三种方法：

- 从当前工作单的控制菜单框的下拉菜单中选择“关闭”。
- 单击当前工作单标题栏右上角的关闭按钮 ▩。
- 使用快捷键“Ctrl”+“F4”或“Alt”+“F”+“C”。

如果要关闭当前打开的所有帮助页，可从“Window”菜单中选择“Close All Help”，其快捷键为：“Alt”+“F”+“E”；如果要关闭当前打开的所有工作单，可从“Window”菜单中选择“Close All”，其快捷键为：“Alt”+“F”+“L”。当退出 Maple V 时，所有的工作单也自动关闭。如果要关闭新修改过的工作单，Maple V 会提示用户保存所作的修改。

除了上面的三种方法外，Maple V 还提供了 3 个命令（quit、done、stop）用于退出当前的工作单。在工作单中，用户可以随时键入这三个命令。但需要提醒读者的是，这三个命令直接关闭退出的工作单，不给出保存最新修改的警告信息。这三个命令是 Maple V 为命令行界面提供的，不提倡用户使用这种方法退出工作单。

6. 多个工作单之间的切换

在 Maple V 中，用户可以同时打开多个工作单，多个工作单之间可以自由切换。最基本的切换方法是利用系统集成界面的“Window”菜单，如图 2-15 所示。

图 2-15 “Window”菜单的下拉菜单

在“Window”菜单下拉菜单中的下半部分，按各个工作单打开的先后顺序列出了当前打开的所有工作单包括帮助页的名字。如果当前打开的工作单超过 9 个，则可以在“其

他窗口”选项弹出的“选择窗口”对话框（如图 2-16 所示）中选择第 9 个工作单以后的工作单，在选择窗口对话框中列出了当前打开的所有工作单的名字。从列表中选择所需的工作单，该工作单即成为当前工作单，并处于活动状态。

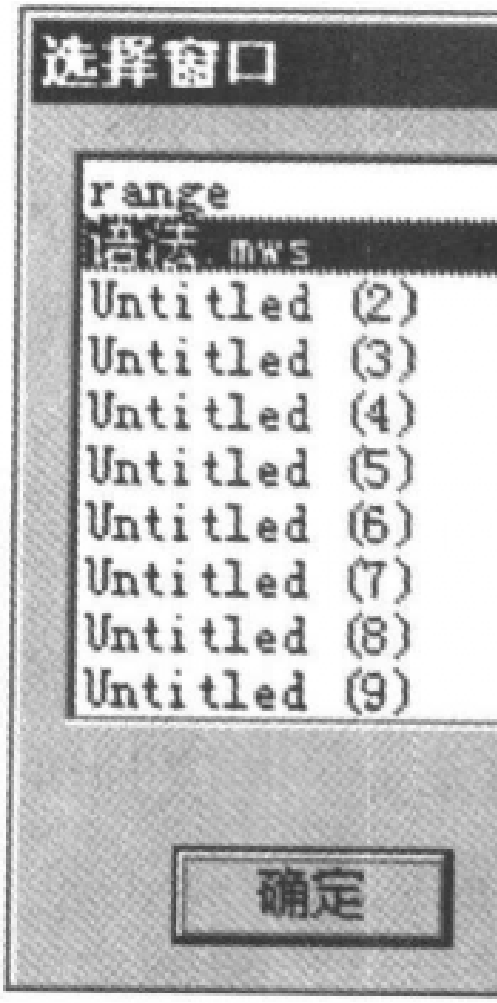


图 2-16　“选择窗口”对话框

对于当前暂时不用的工作单，可以单击该工作单标题栏右上角
于最小化状态。处于最小化状态的工作单，按顺序排列在工作空间
示；每个工作单都只显示出标题栏，标题栏中给出了该工作单的名
户可以用下面两种方法切换到所需的工作单：

- 双击任一个工作单的标题栏。
- 单击某一个工作单的标题栏，从弹出的菜单中，选择“恢

这两种方法都可以使所选择的工作单成为活动的当前工作单。

图 2-17　工作单最小化后的工作空间

2.3.2 工作单中的文档对象

工作单中的内容是很丰富的，包括命令的输入输出、文本、图形、公式等基本的文档对象，而且包括程序段、文本段、电子表格、可折叠区段和超链接等高级文档对象。其中的高级文档对象使得用户可以很方便地组织工作单中的内容，执行高级的文档操作。

前面两节介绍了 Maple V 命令的输入和输出，文本和公式的输入在 2.3.3 节中介绍，图形则在绘图一章中介绍。本小节将简单介绍文本段、程序段、电子表格、可折叠区段和超链接的功能和特点。

1. 程序段

程序段的主要任务是把一条或多条命令输入、注释性文字和相应的输出结果合并为一个可再执行的单元，它是工作单中的基本计算单元。广义地讲，工作单中由一个大方括号“[”从左边界定在一起的内容称为一个程序段，其中除了命令的输入输出和注释性文字外，还可以包含描述性的文本段、图形、电子表格、超链接等。在使用 Maple V 进行科学计算的过程中，程序段作为基本的计算单元，是用户操作的基本对象。

2. 文本段

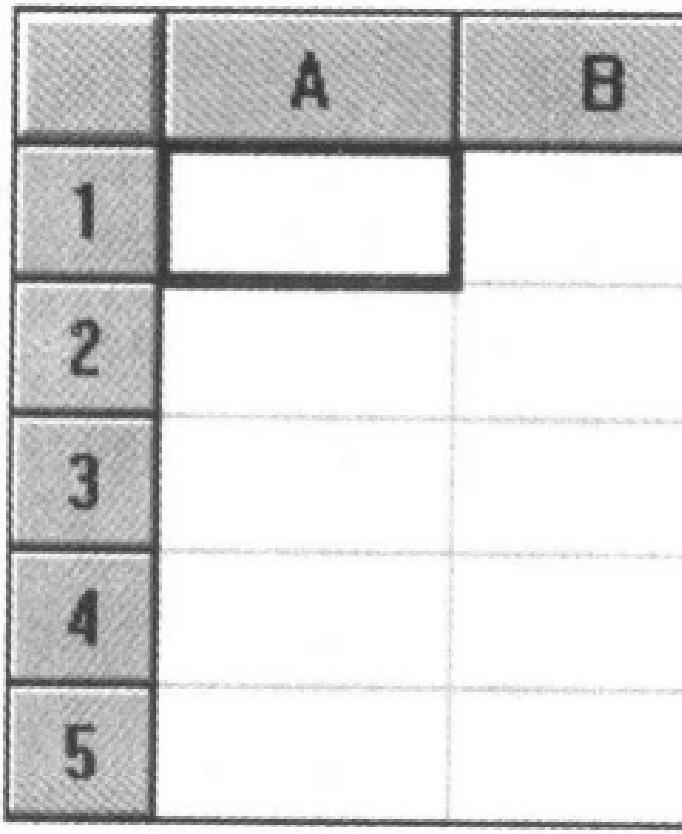
工作单中除了命令的输入输出和注释性文字外的内容通称为文本段，包括不可执行的纯文本、公式、图形、超链接、电子表格等。通称文本段是不可执行的描述性文字，可以独立存在，也可以用大方括号界定作为程序段的一部分或者全部。

3. 电子表格

Maple V 提供了电子表格的功能，使得用户可以方便地将工作单中的内容组织为表格的形式，方便地进行数据的比较。用户可以在任意需要的位置插入电子表格，只需从“Insert”菜单中选择“Spreadsheet”选项，Maple V 在当前光标的下边插入一个标准的空白电子表格，即处于电子表格输入状态；上下文关联栏中将出现用于电子表格操作的快捷按钮，如图 2-18 所示；该电子表格包含在一个空白程序段中，其中没有任何其他内容，用户可以执行其他相关操作，在该程序段中添加其他内容。

图 2-18 电子表格编辑工具栏

下面给出了一个包含空白电子表格的程序段。用户可以很方便地在电子表格的各个域中输入数据，执行成批的计算，或者改变电子表格的样式等。



4. 可折叠区段

可折叠区段用于把工作单中相关的内容包含在一
述，问题的分析求解过程以及问题的结果和讨论等。
广泛地用在 Maple V 的在线帮助页中，用户在使用
折叠区段的功用。

可折叠区段是由一个顶部带控制按钮的大方括号
按钮来控制可折叠区段的折叠与展开。当可折叠区
的形式，当处于展开状态时，控制按钮变为 ▣ 的形
了两个可折叠区段，其中第一个可折叠区段处于折叠
态。

用户既可以插入空的可折叠区段，然后在其中添
接等对象，也可以把分散的对象封装在一个可折叠区
合并为一个可折叠区段，或者把一个可折叠区段分解
以嵌套，如图 2-19 所示的第二个区段中即包含了一个处于展开状态的子区段。

图 2-19　包含可折叠区段的工作单

5. 超链接

Maple V 在工作单中提供了超链接功能，用户可以很方便地在工作单内部从一个专题跳到相关的另一个专题，或者从当前的工作单转到另一个包含相关内容的工作单。这种功能在求解大型问题或者编写大型问题的科学报告时非常有用。超链接在 Maple V 自身的在线帮助系统中得到了广泛的应用，读者在阅读 Maple V 的在线帮助时可以感受到超链接的优越性。

默认情况下，工作单中的超链接文本以暗绿色显示，并带下划线，字体类型为“Times New Roman”正常体。用户也可以改变超链接的样式设置。

2.3.3 工作单中的文本段

本小节介绍与文本段有关的内容，包括 Maple V 的文本输入模式、文本段中的公式输入和纯文本的样式。

1. 文本输入模式

文本输入模式指输入非执行性文本时所处的环境，区别于命令输入时的环境，命令输入时的环境也称为命令输入模式。通常，Maple V 总是处于命令输入模式下，用户可以用三种方法进入文本输入模式：

- 在命令行，把光标移到命令提示符的左边，即可开始键入文本，如图 2-20 所示。此时，上下文关联栏中即出现文本编辑工具栏，说明当前处于文本输入模式下。

图 2-20 Maple V 中的文本输入模式

用这种方法进入文本编辑模式后，当键入第一个字符时，命令提示符自动消失。当完成文本行的输入并直接按下“Enter”键时，Maple V 在同一程序段内插入以命令提示符开始的命令行，返回命令输入模式。

- 单击工具栏中的快捷按钮  或从“Insert”菜单的下拉菜单中选择“Text Input”选项，对应的快捷键为“Ctrl + T”。

这两种操作可立即进入文本输入模式，当前光标之后输入的内容都是纯文本内容。虽然用这两种方法用户甚至可以在命令的中间插入非执行性的文本内容，但是这并不是一种好的做法。这时要返回到命令输入模式，可以单击工具栏中的快捷按钮  ，Maple V 在当前程序段下边插入一个新的空程序段。

- 当光标处于闪烁状态时，从“Insert”菜单的下拉菜单中选择“Paragraph”子菜单，对应于“Before”和“After” 两个选项，用户可以在光标前或光标后插入新的文本行。插入的文本行仍位于当前的程序段内。在光标前插入文本行的快捷键为“Shift + Ctrl + K”，在光标后插入文本行的快捷键为“Shift + Ctrl + J”。

另外，在工作单中插入可折叠区段、超链接等文本对象后同时进入了文本输入模式。任何时候，光标位于文本段包括空的文本行中间时，当前即处于文本输入模式。在文本输入模式下，用户可以对文本进行再编辑，包括插入、删除、复制、剪切、粘贴等基本操作。

2. 文本段中的公式输入

文本段中的公式指的是经过排版的数学表达式。在 Maple V 的文本段中输入公式是很方便的。最简单的办法是，在命令行中按照 Maple V 的语法输入公式对应的表达式并执行，然后把输出的结果复制并粘贴到文本段中相应的位置。

在文本输入模式下，也可以从“Insert”菜单的下拉菜单中选择“Maple Input”选项，这时的输入状态类似于 Standard Math 命令输入模式，用户可以直接输入公式并进行编辑。

3. 纯文本的样式

在默认情况下，进入文本输入模式后输入的文本都属于正文部分，其样式名称为“P Normal”，以黑色显示，字体类型为“Times New Roman”，字符大小为 12。用户可以使用文本编辑工具栏或“Format”菜单中的选项来改变文本的字体、大小、颜色、对齐方式及其他属性等。

同时用户可以从文本编辑工具栏最左边的文本样式域中选择要输入文本的样式，如图 2-21 所示。在文本样式域的可选框中列出了 Maple V 中所有的文本样式名，包括正文（Normal）、作者（Author）、各类标题（Heading）、超链接（Hyperlink）等等，用户也可以定义自己的样式，还可以选择已输入的文本，再应用相应的样式及其他属性。

图 2-21　文本编辑工具栏

2.4 Maple V 的程序段

程序段是工作单的基本计算单元，在 Maple V 的使用过程中充当着很重要的角色。程序段可以再次执行，决定了工作单具有可再次执行的特征。用户可以在工作单中对程序段进行整体操作。本节首先介绍程序段的再执行操作，接着介绍程序段的编辑操作。

2.4.1 程序段的再执行

程序段可以再次执行，体现了工作单的可执行性。当然只有包含了可执行命令的程序段才可以再次执行。用户可以同时选择多个程序段甚至是整个工作单中的全部程序段，重新执行其中的命令。

1. 单个程序段的再执行

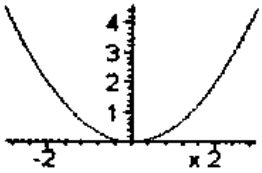
程序段的再执行操作非常简单，把光标停在希望再次执行的程序段的任意命令行中，按下“Enter”键即可。此时，程序段中的所有命令按先后顺序再次执行，对应的输出结果依次显示在程序段的最后几行，同时光标移到下一个包含可执行命令的程序段中。如果再次执行的一个程序段下边没有其他的程序段，则 Maple V 将自动插入一个新的空程序段。

下面给出了三个程序段，前两个程序段内各自包含了一条可执行命令和相应的输出，首行为文本行，最后一个程序段只包含一行文本。

```
这里做一个积分运算。
> int(x,x);
```

$$\frac{1}{2}x^2$$

```
这里绘制积分函数的图像。
> plot(%,x=-3..3);
  这里加入一行文本，输出显示在下边。
```




```
[这里是当前最后一行。
```

按上面的操作方法，读者可以多次执行两个程序段，当然每次的结果都是一样的。注

意当光标位于文本行时按下“Enter”键，有时并不能使程序段再次执行。

2. 多个程序段的再执行

上面介绍的方法只适用于单个程序段的再次执行。用户可以一次选择多个程序段，然后从“Edit”菜单中选择“Execute”子菜单的“Selection”选项，所选中的多个程序段中的命令将再次执行。如果选择的是某个程序段中的一部分命令，执行上面的操作后该程序段中的所有命令都将再次执行，这和单个程序段的再执行操作是等价的。

如果选择的是整个工作单中的所有程序段，则执行上面的操作可以使其中的所有可执行命令再次执行。用户也可以从“Edit”菜单中选择“Execute”子菜单的“Worksheet”选项，则当前工作单中的所有程序段都将重新执行。这种操作在处理大型问题时非常有用，用户可能无法在一次操作中完成所有的分析计算，工作单的再次执行可以使用户再次回到上次操作结束时 Maple V 工作空间的状态，即工作单中用到的所有变量都恢复到上次的设置。

2.4.2 程序段的基本编辑

除了工作过程中 Maple V 自动插入的程序段外，用户还可以在需要的位置插入新的程序段。同时，可以对程序段整体进行基本编辑，或者把多个程序段合并为一个程序段，或者把一个程序段拆分为多个程序段等。

1. 插入新的程序段

用户可以在任意需要的位置插入新的程序段，输入新的命令或者添加其他的内容，这一点体现了工作单的可编辑特征。在工作单中插入新的程序段，可以采用下面的两种方法：

- 把光标停在要插入新程序段的程序段内的任意一行，也可以选中程序段中的输入输出对象，然后单击工具栏中的快捷按钮  。Maple V 将在光标所在程序段的下边插入一个新的空程序段。
- 当光标呈闪烁状态时，从“Insert”菜单中选择“Execution Group”子菜单，对应于“Before Courser”或“After Courser”两个选项，用户可以选择在当前程序段之前或之后插入新的程序段，对应的快捷键分别为“Ctrl + K”和“Ctrl + J”。

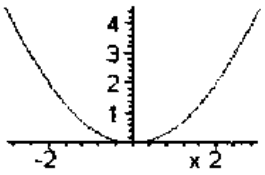
接 2.4.1 节中的例子，在原来的两个程序段之间插入一个新的程序段，读者可以采用上面的任一种方法。

```
这里做一个积分运算。
> int(x,x);
```

$$\frac{1}{2}x^2$$

```
>
```

这里绘制积分函数的图像。
```
>plot(%,x=-3..3);
```
这里加入一行文本，输出显示在下边。



插入新的程序段之后，光标停在新程序段的首行命令提示符的右边，等待用户的输入。

2. 程序段整体的基本编辑

用户不仅可以分别对程序段中的输入、输出或其他内容进行编辑，而且可以对程序段的整体进行编辑，包括程序段的选定、删除、复制、剪切和粘贴等基本编辑操作。

上面的几种基本操作既可以用菜单操作来完成，也可以用快捷键或者快捷按钮来完成。其基本方法和其他常用的文档处理软件类似，也采用 Windows 默认的各种快捷键设置，这里不再详细介绍。需要指出的是，这种整体的编辑操作不能用于和其他文档处理软件实现交互。

实际上，用户可以删除程序段左边的大方括号。单击大方括号的任意位置，大方括号就处于选中状态，这时按下“Delete”键，大方括号就消失了。删除大方括号后，原程序段中的各行都相互独立开来。下面给出的例子中，删除了程序段的大方括号：

这里作一个积分运算。

```
>int(x,x);
```

$$\frac{1}{2}x^2$$

这种操作对于文本段可能是需要的，但对于命令行来说通常是不必要的，因此并不提倡这种操作。

3. 程序段的合并

很多时候需要把一些相关的命令放在同一个程序段内执行，除了在同一程序段内一次同时输入多条命令外，用户也可以把几个独立的程序段合并为一个程序段。这种操作通常只能实现相邻程序段的合并；如果想把不相邻的程序段合并在一起，还是要通过复制、剪切、粘贴等操作把两个程序段按顺序排列在一起。下面给出了合并相邻的两个程序段的步骤：

（1）把光标放在第一个程序段中；

（2）从“Edit”菜单中的“Split or Join”子菜单中选择“Join Execution Groups”，对应的快捷键为功能键“F4”。

程序段合并后，在没有重新执行新的程序段之前，其中的输入输出和其他内容仍按原

来的顺序排列。下面给出了 2.4.1 节中给出的三个程序段合并后的结果：

这里做一个积分运算。

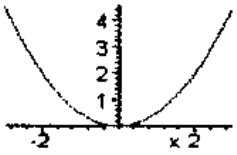
```
>int(x,x);
```

$$\frac{1}{2}x^2$$

这里绘制积分函数的图像。

```
>plot(%,x=-3..3);
```

这里加入一行文本，输出显示在下边。



这里是当前最后一行。

用户可以重新执行合并后的程序段，这时的输出都依次排列在程序段的最后几行，而所有的命令输入及其他内容都按原顺序排列在前几行。前面的程序段再次执行后，呈现下面的状态：

这里做一个积分运算。

```
>int(x,x);
```

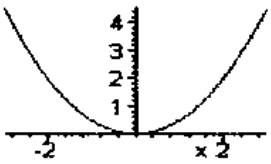
这里绘制积分函数的图像。

```
>plot(%,x=-3..3);
```

这里加入一行文本，输出显示在下边。

这里是当前最后一行。

$$\frac{1}{2}x^2$$



4. 程序段的拆分

同样，用户可以把一个包含多行输入输出或其他内容的程序段拆分为几个相邻的程序段。下面给出了拆分程序段的操作步骤：

（1）把光标放在程序段中合适的位置，使光标处于闪烁状态；

（2）从“Edit”菜单中的“Split or Join”的子菜单中选择“Split Execution Group”，对应的快捷键为功能键“F3”。

如果光标所在的行是命令行，程序段在当前行与下一行之间分离；如果光标所在的行

是输出行，则程序段在当前行和上一行之间分离；如果光标所在的行是文本行，则程序段在光标所在的位置分离，即可以把当前文本行分离在两个程序段内。如果拆分后的程序段中只包含输出结果，Maple V 将自动在首行插入一个空的命令行。

程序段拆分以后，用户可以再次执行其中包含可执行命令的程序段。而只包含输出的程序段再次执行后只留下一个空命令行，原来的输出结果消失。如果拆分后的程序段中只包含文本段和输出结果，当光标位于文本段内按下“Enter”键，其中的输出结果也自动消失。

2.5 Maple V 的帮助系统

Maple V R5 提供了一套完善的帮助系统，用户可以很方便地获得所需的帮助信息。用户可以用以下几种方法获得帮助：帮助浏览器、主题搜索、全文搜索、历史搜索和冒泡帮助系统等。其中帮助浏览器是 Maple V 系统所有在线帮助的一个索引，主题搜索和全文搜索的结果中都包含了帮助浏览器，这种显示帮助的方式也是 Maple V 系统的一个特色。

本节将分别介绍 Maple V 的冒泡帮助系统、帮助浏览器、主题搜索系统、全文搜索系统和历史搜索系统。

2.5.1 Maple V 的冒泡帮助系统

Maple V 提供了冒泡帮助系统，读者可能在其他的应用软件中见过类似的帮助系统。从“Help”菜单中选择“Balloon Help”选项，即可打开冒泡帮助系统。当“Balloon Help”选项前有被选标志时表明冒泡帮助系统已经打开。

当冒泡帮助系统打开后，鼠标位于工具栏的快捷按钮或者菜单弹出的直接选项上时，该快捷按钮或该选项的帮助信息就会以冒泡的形式显示出来，如图 2-22 所示。对于 Maple V 的初学者来说，打开冒泡帮助系统有助于更快地掌握快捷按钮和各种菜单的使用。

图 2-22 Maple V 的冒泡帮助

如果用户已基本掌握了快捷按钮和各种菜单的使用方法，可以再次从“Help”菜单中选择“Balloon Help”选项关闭冒泡帮助系统。这时，“Balloon Help”选项没有选中标志。

2.5.2　Maple V 的帮助浏览器

Maple V 的帮助浏览器如图 2-23 所示，出现在 Maple V 在线帮助页的顶部，通称包含 5 个专栏。在左边第一个专栏里，列出了 Maple V 帮助系统所有的根主题。单击其中的某一主题，如果该主题以省略号“...”结尾，接着的专栏将列出该主题所包含的子主题；这些子主题还可以包含下一级子主题，显示在该子主题右边的专栏里，直到第五个专栏。在每个专栏里可以移动专栏右边的滑动块查找所感兴趣的主题，在所选择的主题上单击，如果该主题没有子主题，则在专栏下面的浏览器窗口中显示该主题的帮助信息。

图 2-23 中显示的是“Managing Files and Worksheets（文件和工作单的管理）”的帮助信息，其对应的根主题是“Worksheet Interface...”，一级子主题是“Managing Files and Worksheets...”，二级子主题是“Overview”。

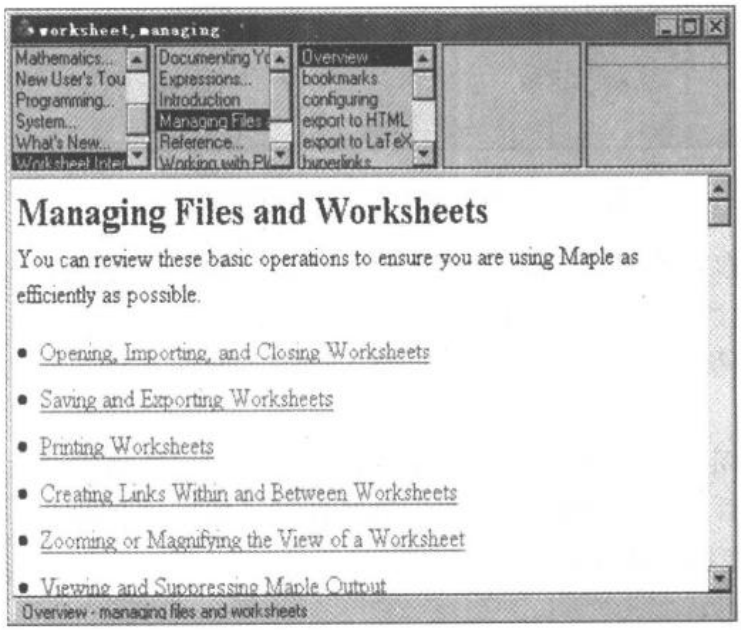


图 2-23　Maple V 系统的帮助浏览器

2.5.3　Maple V 的主题搜索系统

Maple V 提供了主题搜索系统，用户可以很方便地获得有关某一主题的帮助信息。主题搜索系统提供了一种模糊查找功能，用户可以不必知道所关心主题的整个单词，Maple V 能够根据用户提供的前几个字母自动查找与这几个字母匹配的主题。用户可以从中选择自己所关心的主题。主题搜索可以通过主题搜索对话框和帮助命令两种方式进行，下面分别介绍这两种方式。

1. 利用主题搜索对话框

具体的操作步骤为：

（1）从“Help”菜单里选择“Topic Search”选项，打开主题搜索对话框，如图 2-24 所示；

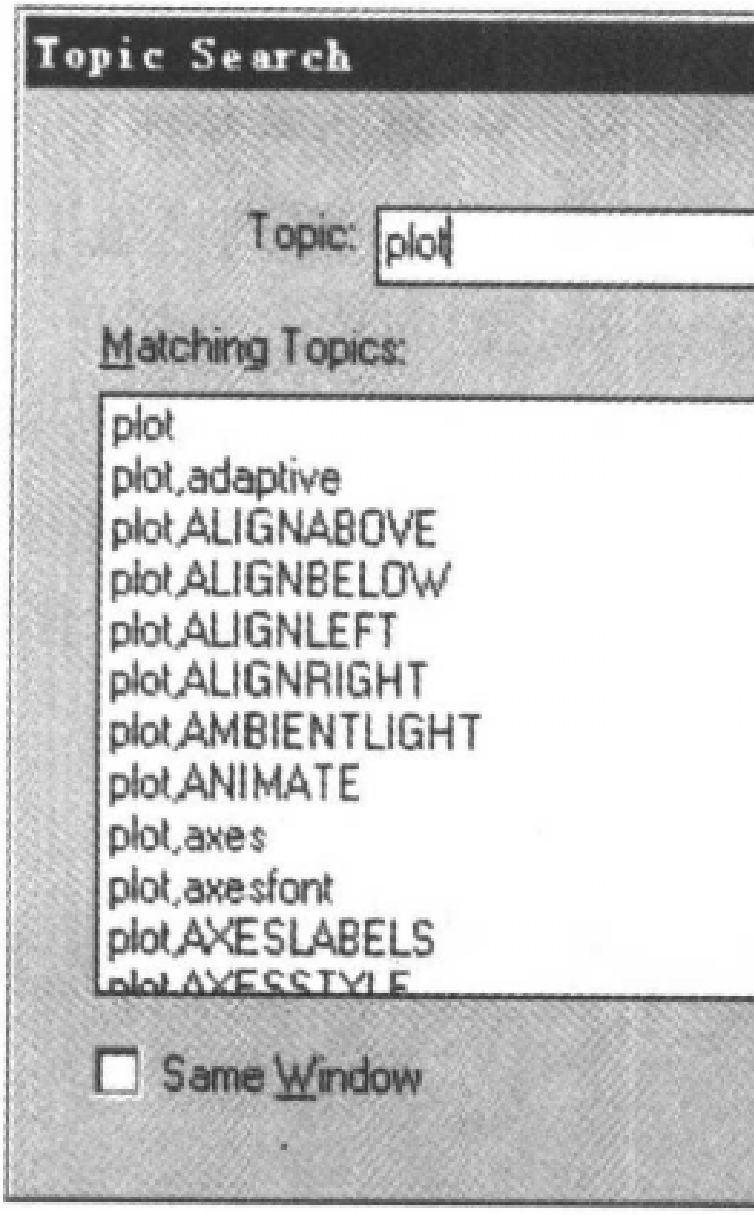


图 2-24 Maple V 系统的

（2）在主题搜索对话框的“Topic”栏里输入感
Topics”栏里将列出所有以这些字母开头的帮助主

（3）从得到的帮助主题列表中选择希望了解
的帮助信息；

（4）当不能确定是否找到了正确的主题并且
的主题上单击，然后单击“Apply”按钮；这时打
仍然留在屏幕上方便以后的使用。

2. 利用帮助命令

上面的操作也可以用帮助命令来完成。用户只
或者 **help('topic')**，其中 **topic** 代表所关心的主题
键即可。如果 Maple V 帮助系统中存在与所输入
命令将直接打开该主题对应的帮助页；如果存在多
母相同，Maple V 打开一个标题为“Help Choice”
示了所有相关的主题，用户可以从中选择关心的主
系统中不存在该主题的帮助信息，Maple V 将以对

例如，在命令提示符“>”后输入命令**?plot**，
如图 2-25 所示，其中给出了命令 **plot** 的帮助信息

为“Help Choice”的帮助页，如图 2-26 所示，其中以超链接文本的方式给出了 Maple V 帮助系统中所有以“**pl**”开头的主题，包括 **plot**、**plot3d** 等；如果键入的是命令**?yes**，Maple V 将给出如图 2-27 所示的对话框，其中的警告信息说明不能找到关于“yes”的任何帮助。

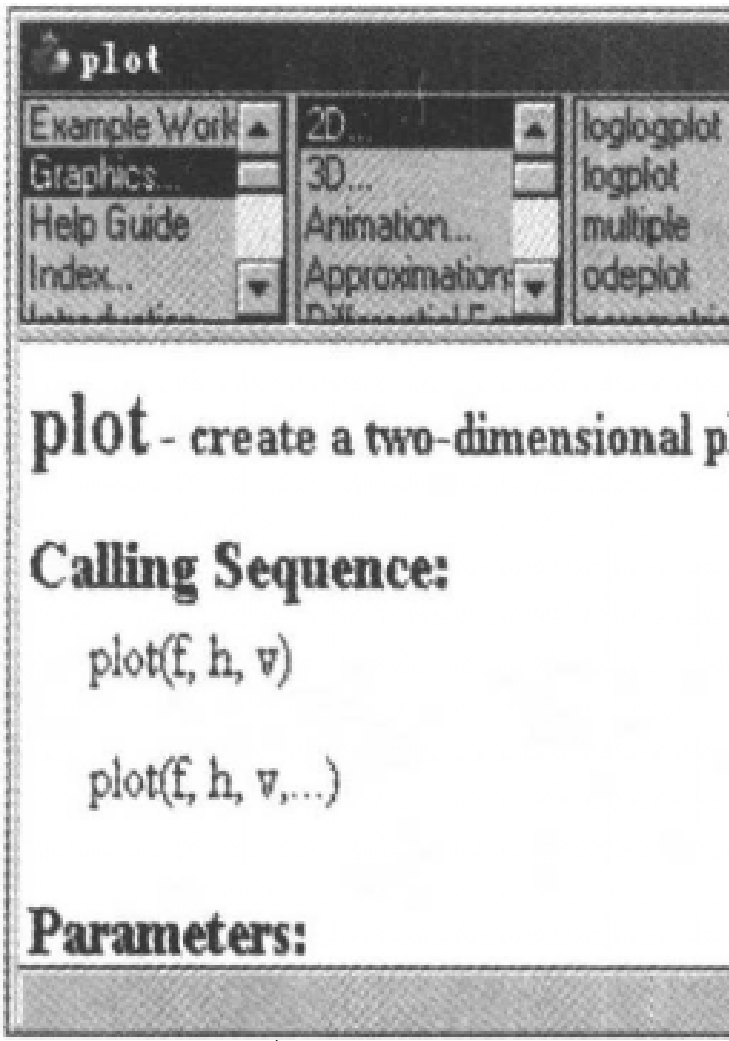


图 2-25　命令 plot 的

图 2-26　命令?pl 打开

图 2-27　命令?yes 得到的警告对话框

如果当前的工作单中存在与所感兴趣的主题对应的单词，用户可以把光标停在该单词上，然后从“Help”菜单中选择“Help on …”选项，对应的快捷键为组合键“Ctrl”+“F1”。这种操作和上面的命令输入操作是等价的。

2.5.4 Maple V 的全文搜索系统

全文搜索是在 Maple V 的所有帮助页中进行字符串匹配以获得帮助信息的一种方法。下面是使用全文搜索系统获得在线帮助的步骤：

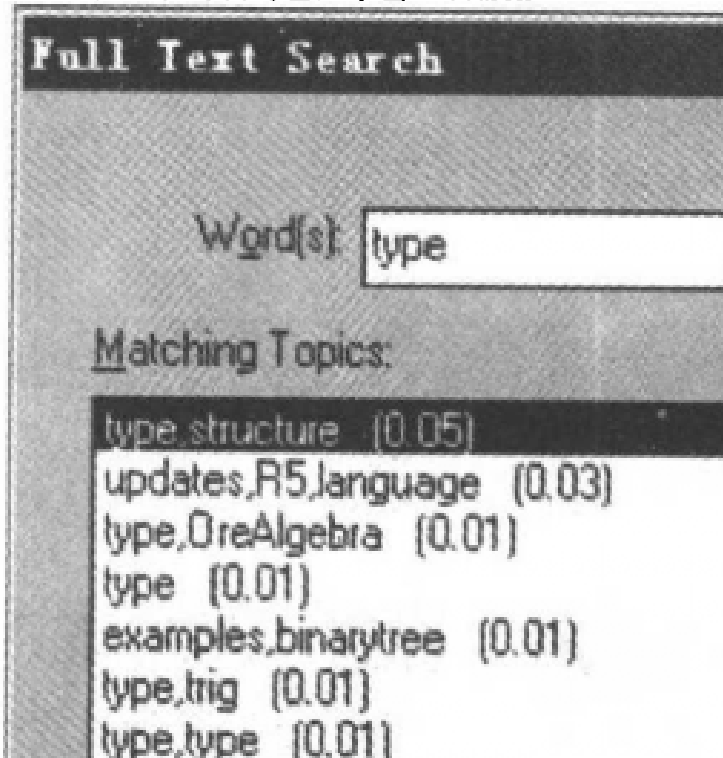
（1）从“Help”菜单里选择“Full Text Search”选项，打开全文搜索对话框，如图 2-28 所示；

图 2-28 Maple V 系统的主题搜索对话框

（2）在全文搜索对话框的“Word(s)”栏里输入要搜索的单词或词组，单击“Search”按钮；

（3）Maple V 在“Matching Topics”栏里列出
主题后的小括号中的数字指出了该帮助页与所搜索

（4）选择感兴趣的主题并双击，即打开该主题



2.5.5 Maple V 的历史搜索系统

除了上面几种获得帮助的方法外，Maple V 的
再次访问先前访问过的帮助页。使用历史搜索系统

（1）从“Help”菜单里选择“History”选项，
示：

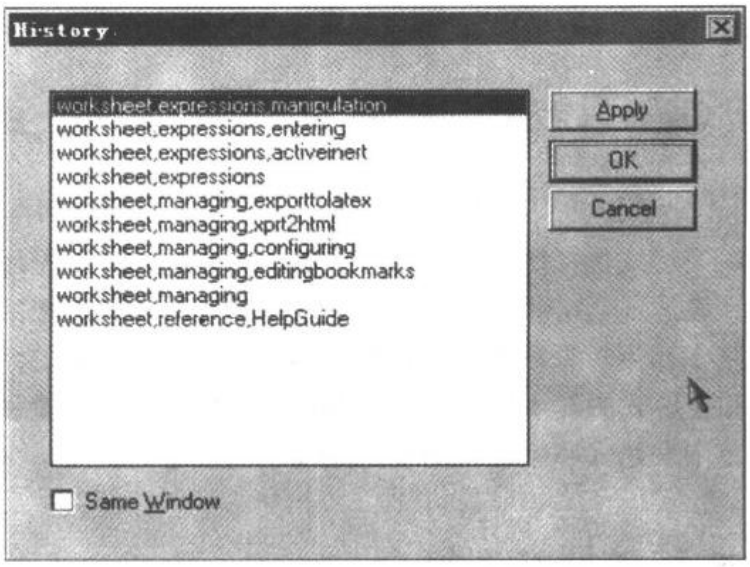


图 2-29　Maple V 系统的历史搜索对话框

（2）从“History”对话框的列表里选择所关心的主题，双击即可再次打开该主题的帮助页。

同时，用户可以使用工具栏中的快捷按钮实现帮助页之间的跳转，图 2-30 给出了各个快捷按钮的作用。

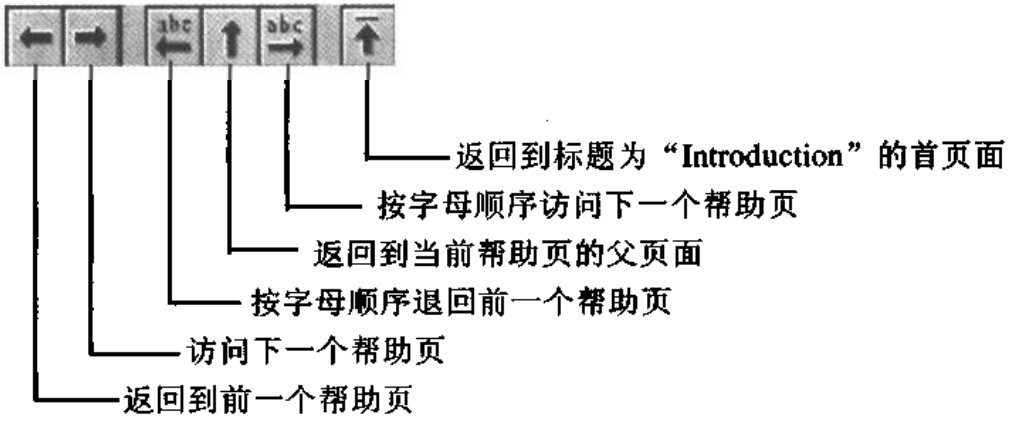


图 2-30　快捷按钮的作用

第 3 章 变量和表达式

在 Maple V 中，表达式包括所有命令或函数的作用对象，变量是用来代表表达式的标识符。数据类型是和表达式紧密相关的，不同的表达式都有各自对应的类型。变量和表达式是 Maple V 中的基本作用对象，在科学计算过程中充当非常重要的角色。

本章介绍变量和表达式的基本知识，包括 Maple V 的常量、变量和表达式的构成及数据类型等，并介绍范围表达式和字符串两种比较简单的表达式。

3.1 表达式和数据类型简介

在 Maple V 中，表达式和数据类型的范围很广。包括数字、字符、方程、不等式及图像在内，所有命令或函数的作用对象都可以称为表达式。根据表达式的构成，可以把表达式分为多个类别，由算术运算符构成的表达式通称为算术表达式，由各种合法字符构成并包含在一对双引号内的字符序列通常称为字符串表达式，由关系运算符构成的表达式通常称为关系表达式，由逻辑运算符构成的表达式通常称为逻辑表达式，由逗号隔开的多个表达式称为表达式序列等等。

数据类型和表达式紧密相关，不同种类的表达式有各自对应的数据类型。整数、分数、浮点数、复数、字符是最基本的数据类型，列表、集合、表、数组、矩阵和向量是 Maple V 中的复合数据结构，表达式是由运算符和这些基本的数据类型或复合数据构成的。与种类繁多的表达式相对应，数据类型的概念也非常广泛，例如未赋值的标识符对应的类型名为 **symbol**，方程或等式对应的数据类型为 **equation**，函数调用对应的数据类型为 **function**，多项式对应的数据类型为 **polynom**，由逗号隔开的多个表达式对应的数据类型为 **exprseq** 等等。Maple V 甚至把 “+”、“*”、“^”、“**”、“!”、“<”、“<=”、“=”、“.” 等几个运算符作为数据类型标识。用户可以使用命令**?type** 和**?whattype** 获得关于 Maple V 数据类型的详细知识。本章只是初步介绍 Maple V 的数据类型，具体内容将在相关章节中介绍。

按照表达式所代表的值的变化情况，可以把表达式分为常量表达式和变量表达式，通常也称为常量或变量。表达式可以是匿名的，也可以是有名的。标识符是用来代表表达式的合法字符序列，使用前不需要预先定义。未赋值的标识符代表其自身，被赋值以后通常具有与所代表的表达式相对应的数据类型。

Maple 提供了一组与表达式和标识符有关的命令，用户可以使用这些命令来查看表达式的类型，分析表达式的结构，引用其中的某个操作数或者构建新的表达式等。表 3.1 中列出了这几个命令的基本格式及作用，具体的用法在随后的几节中介绍。

表 3.1　与表达式有关的常用命令

命令格式	作　用
Whattype(f)	直接获得表达式 f 的数据类型
type(f, t)	分析表达式 f 的类型是否属于 t
nops(f)	返回表达式 f 中主操作数的个数
op(i, f)	选择表达式 f 中的第 i 个操作数
subsop(i=g, f)	用 g 来替换表达式 f 中的第 i 个操作数
has(f, x)	分析表达式 f 中是否包含子表达式 x

3.2　常量和变量

在科学计算过程中经常用到常量和变量，标识符是用来代表常量和变量以及其他各种输入输出对象的符号，对标识符的赋值是计算过程中最基本的操作。本节将介绍 Maple V 中的常量、标识符以及标识符的赋值操作。

3.2.1　常量

在计算过程中值不发生改变的量称为常量。可以用一个标识符来代表常量。常量可以分为不同的类型，包括数字常量、符号常量、字符串常量、系统常量等。数字常量又包括整数常量、分数常量、浮点数常量、复数常量等。例如 **12**、**0** 为整数常量，**3/4**、**-9/7** 为分数常量，**2.3**、**-.234** 为浮点数常量，**1+I**、**-2.2-3*I** 为复数常量，“**abc**”、“**_C1**”为符号常量，“**a**”、“**China**”为字符串常量，**Pi**、**FAIL** 为系统常量等等。下面将主要介绍 Maple V 中的系统常量。

除了系统常量的类型名为 **constant** 外，其他几种常量都不属于 Maple V 中的 **constant** 数据类型，而具有各自对应的类型名，例如整数常量对应的类型名为 **integer**，分数常量的类型名为 **fraction**，浮点数常量的类型名为 **float**，复数常量的类型名为 **complex**，符号常量的类型名为 **symbol**，字符串常量的类型名为 **string** 等。

Maple V 中的系统常量是由 **constants** 序列来定义的，启动 Maple V 后 **constants** 常量序列的初始设定为 ***false***、γ、∞、***true***、***Catalan***、***FAIL*** 和 π，分别用标识符 **false**、**gamma**、**infinity**、**true**、**Catalan**、**FAIL**、**Pi** 来表示。其中 **false**、**true**、**FAIL** 是三个逻辑常量（逻辑假、逻辑真、不确定），**Pi** 代表圆周率 π，**infinity** 代表无穷大∞，**gamma** 代表 γ，**Catalan** 代表 **Catalan** 常量。即：

```
> constants;
```

$$false,\ \gamma,\ \infty,\ true,\ Catalan,\ FAIL,\ \pi$$

读者可以看到，Maple V 的输出结果是用常用的数学符号或者希腊字符来表示的。这

是 Maple V 符号计算特征的体现，在使用 Maple V 的过程中，读者可以进一步认识到这一点。用户输入这些数学符号或者希腊字符时需要用与它们对应的标识符或者其他命令。请看下面的结果：

```
> infinity,Pi,gamma;
```

$$\infty, \pi, \gamma$$

用户可以用命令 **type(name, constant)**来检验标识符 **name** 是否系统常量，如果命令输出为 **true**，则 **name** 是 **constants** 序列中的常量，否则不是。注意 Maple V 严格区分字母的大小写，因此 **pi** 和 **Pi** 是两个不同的标识符。请看下面的程序段，其中包含了两个命令，用组合键“Shift”+“Enter”来换行：

```
> type(Pi,constant),type(Pi,constant);
  type(false,constant);
```

$$true, false$$

$$true$$

另外，还有两个常用的系统常量没有包含在 **constants** 序列中，即虚数单位 *I* 和欧拉常数 **e**，分别用字母 **I** 和 **e** 来表示，但欧拉常数 **e** 在计算过程中只能使用命令 **exp(1)**来表示，而不能用字母 **e** 表示。请看下面的命令结果：

```
> type(I,constant),type(e,constant);
```

$$true, true$$

同时，Maple V 允许用户在 **constants** 序列中添加新的系统常量，或者改变序列 **constants** 中的元素从而改变系统常量的设置。下面的命令即把 **I**、**e**、**a** 添加为 **constants** 序列的元素，因此这时 **a** 也具有 **constant** 数据类型：

```
> constants:=constants,I,e,a;
```

$$constants := false, r, \infty, true, Catalan, FAIL, \pi, I, e, a$$

```
> constants:
```

$$false, r, \infty, true, Catalan, FAIL, \pi, I, e, a$$

```
> type(a,constant);
```

$$true$$

下面的命令则把 **constants** 序列改变为只有一个元素 **Pi**，因此这时 **infinity** 及其他几个原 **constants** 序列中的元素不再具有 **constant** 数据类型：

```
> constants:=Pi;
```

$$constants := \pi$$

```
> constants;
```

$$\pi$$

```
> type(Pi,constant),type(infinity,constant);
                              true, false
```

3.2.2 标识符

标识符是用来代表表达式的字符序列。按照标识符的构成方式，可以把标识符分为简单标识符、复杂标识符和索引标识符三种。标识符在未被赋予其他意义以前，代表其自身，属于符号常量。同时，Maple V 自身的过程控制关键字、运算符、数据类型名、命令或函数名等已经都赋予了特殊了意义，用户一般不能使用这些字符或单词作为标识符使用，Maple V 把这些标识符划分为保留关键字和保护名两类。下面分别介绍三种不同的标识符、符号常量、保留关键字和保护名。

1. 简单标识符

简单标识符指的是比较规则的标识符。和 C 语言、Fortran、Pascal 等高级语言类似，Maple V 规定标识符通常只能由字母、数字和下划线三种字符组成，而且第一个字符必须为字母或下划线。标识符的最大长度与当前的机器系统有关，在 32 位机上为 524271 个字符，在 64 位机上为 34359738335 个字符。下面给出了几个合法的表达式构成的序列：

```
> x,a_long_name_containing_undersores,H20;
              x, a_long_name_containing_undersores, H20
> Unknown,UNKNOWN,UnKnOwN;
                        Unknown, ?, UnKnOwN
```

注意，Maple V 严格区分字母的大小写，因此上面第二个程序段中 **Unknown**、**UNKNOWN**、**UnKnOwN** 是三个不同的标识符，其中 **UNKONWN** 代表未知数符号?。下面的三个标识符是不合法的：

```
> M.D.Lee,#432,3q1;
control character unexpected
```

Maple V 把以下划线开头的标识符用作全局变量名使用，建议用户不要使用以下划线开头的标识符，否则可能会得到莫名其妙的结果或者错误信息。

2. 复杂标识符

复杂标识符指的是其中包含空格、“.”、“,”、“/” 等特殊字符的标识符。这种标识符主要是为了明确标识符的意义，增加可读性。在输入命令或表达式时，这种标识符需要用一对反引号“`”（通常位于“Esc”键的下边）来界定，反引号并不是标识符的组成部分，在输出结果中是不显示的。但如果输入时没有用反引号界定，则是不合法的标识符。看下面的例子：

```
>`c:/Maple V Release 5/lib`,`I am a symbol`;
```

c : /Maple V Release 5/lib, I am a symbol

```
>`I am a symbol,not a string`:=true;
```

I am a symbol, not a string := true

```
>`I am a symbol,not a string:=true;
missing operator or`;`
```

简单标识符用反引号界定后意义和原来一样。如果希望标识符自身包含反引号，可以用两个连续的撇号来表示，或者在反引号前加一个反斜杠来表示。这里的反斜杠称为转义字符，在构成字符串的时候也会用到。看下面的例子：

```
>`back quote is```;
```

back quote is`

```
>`\` is back quote`;
```

`is back quote

3. 索引标识符

Maple V 允许用户使用带下标的标识符，这里称为索引标识符。索引标识符具有这样的形式：**name[sequence]**，其中 **name** 代表任意合法的标识符，**sequence** 代表下标或多个下标构成的序列。其中，**name** 本身也可以是索引标识符，即索引标识符可以带多重下标。索引标识符和其他标识符一样，在未被赋值以前也代表其自身。下面是几个索引标识符的例子：

```
>A[1],A[1,2*j-1];
```

$$A_1, A_{1,2j-1}$$

```
>T[Cu][1,2],T[2,1][Plasma];
```

$$T_{Cu}\ , T_{2,1}$$

$$1,2 \quad Plasma$$

在标识符 **name[sequence]**中，如果 **name** 被赋值为代表字符串或序列、列表、集合、表、数组、矩阵和向量等复合数据结构的标识符，则此时的索引标识符 **name[sequence]** 用来代表字符串或者复合数据结构中的元素，用户不能再使用 **name** 构造其他索引标识符。

4. 符号常量

在 Maple V 中，标识符的使用不需要预先定义或者预先声明，可以直接使用。标识符在没有被赋值之前，代表其自身，称为符号常量；普通标识符和复合标识符的类型名为 **symbol**，索引标识符的类型名为 **indexed**，这两种类型都属于符号常量。在赋值以后，标

识符的数据类型通常与其所代表的表达式类型相同。用来代表系统常量的标识符同时属于 **symbol** 和 **constant** 两种类型。

符号常量是表达式的一种，用户可以使用前面提到的命令 **whattype** 和 **type** 来分析符号常量的类型和数据结构。看下面的命令：

```
> X,T[1,2],Pi;
```

$$x, T_{1,2}, \pi$$

```
> whattype(x),whattype(T[1,2]),whattype(Pi);
```

symbol,indexed,symbol

```
> type(x,symbol),type(T[1,2],symbol),type(Pi,symbol);
```

true,false,true

```
> type(x,indexed),type(Pi,constant);
```

false,true

5. 保留关键字

在 Maple V 中有 7 个运算符关键字和 25 个语句控制字，用户不能使用这些控制字或运算符作为标识符，这些关键字和语句控制字称为保留关键字（reserved word）。表 3.2 中列出了全部 32 个保留关键字。

表 3.2　　Maple V 中的保留关键字

关 键 字	作　用
If、then、elif、else、fi	if 控制语句
For、from、in、by、to、while、do、od	for 循环语句和 while 循环语句
proc、local、global、end、option、options、discription	过程控制关键字
read、save	读操作和保存操作语句
quit、done、stop	Maple V 系统的中止或退出命令
union、minus、intersect	集合运算符
and、or、not	逻辑运算符
mod	取模运算符

保留关键字在 Maple V 中被赋予了特殊的意义，用户不能用它们来代表其他表达式或者赋予其他意义。如果用户试图对这些保留关键字赋值或者用在其他变量中，Maple 将会给出错误信息。下面是把 **if** 直接作为命令 **help** 的参数，试图获得 **if** 的帮助信息时的错误信息：

```
> help(if);
reserved word `if` unexpected
```

除了保留关键字自身的语法规则外，用户唯一可以直接使用保留关键字的命令为帮助命令**?word**，其中 **word** 代表相应的保留关键字。如果必须使用保留关键字作为标识符或者命令的参数，例如希望使用命令 help 获得这些保留关键字的帮助信息，用户可以在这

些关键字的两边用反引号界定。例如，要获得关于 **if** 的帮助信息，可以使用 **help(`if`)**的命令格式，在命令行中键入下面的命令将打开 **if** 的帮助页面，这个命令和**?if** 是等价的：

```
[>help(`if`);
```

6. *保护名*

除了保留关键字外，Maple V 中用来代表命令或函数、数据类型、系统常量等的标识符也不能用作用户的标识符，这些单词或字符称为保护名（protected name）。例如数学函数名 **sin**、**cos**、**abs**、**max**、**min** 等，命令名 **restart**、**expand**、**simplify** 等，系统常量名 **Pi**、**I**、**true**、**false** 等，数据类型名 **symbol**、**integer**、**list**、**set** 等，都属于 Maple V 的保护名。

这些保护名有自己的使用语法，除此之外，用户不能对保护名赋值。如果试图对保护名赋值，通常会得到错误信息。看下面的例子：

```
>sin:=2;
Error,attempting to assign to`sin`which is protected
>Pin:=3.14;
Error,attempting to assign to`Pi`which is protected
```

和保留关键字不同的是，保护名仍然是一种合法的标识符，除了不能对保护名赋值外，其他操作通常是可以的，所以保护名永远只属于 **symbol** 数据类型。看下面的例子：

```
>sin,4+sin;
```

$$\sin, 4+\sin$$

```
>whattype(symbol),whattype(sin);
```

symbol,symbol

如果必须使用这些保护名作为标识符来使用，可以使用命令 **unprotect** 来解除对特定保护名的保护，命令的格式为 **unprotect(protected_name1, protected_name2, ...)**，其中 **protected_name1**、**protected_name2** 等为希望解除保护的保护名。保护名解除保护后，用户可以自由使用，和其他自定义的用户标识符具有相同的使用规则。但是，该保护名原来被赋予的意义就不复存在，即函数名不再代表原来的函数，命令名不能再作为命令使用。在下面的例子中，读者可以看到使用命令 **unprotect** 解除对函数名 **sin** 的保护前后的差别：

```
>'sin'(-Pi)=sin(-Pi);
```

$$\sin(-\pi)=0$$

```
>sin:=Pi;
Error,attempting to assign to`sin`which is protected
>unprotect(sin);
>sin:=pi;
```

$$\sin = \pi$$

```
>'sin'(-Pi)=sin(-Pi);
```

$$\sin(-\pi)=\pi(-\pi)$$

但必须提醒读者的是，保护名被取消保护并被赋予其他意义后，如果想要恢复保护名原来的意义，必须清除 Maple V 系统缓存中的所有信息，这就要用到后面讲到的命令 **restart**。所以 Maple V 不提倡这种操作。

另一方面，用户可以用命令 **protect** 来请求 Maple V 系统对某一用户标识符的保护，防止该标识符被再次赋值或用作其他用途。命令的格式为 **protect('name1', 'name2', ...)**，其中 **name1**、**name2** 等为需要保护的标识符，必须用单引号界定。下面给出了标识符 **myname** 申请保护前后的差别：

```
> myname:="I am to be protected";
                    myname := "I am to be protected"
> protect('myname');
> myname;
                        "I am to be protected"
> myname:="I am now protected";
Error,attempting to assign to `myname`which is protected
```

用户也可以使用 **unprotect** 命令解除对标识符 **myname** 的保护，这时 **myname** 可以被再次赋值或作其他用途：

```
> unprotect('myname');
> myname:="I am unprotected now";
                    myname := "I am unprotected now"
```

3.2.3　标识符的赋值

赋值是 Maple V 中最基本的运算。通过赋值运算，用户可以把表达式、图像等对象用一个标识符来表示，然后使用该标识符代表原来的对象进行后续操作。标识符经赋值后可以执行释放操作使之重新成为自由标识符。用户可以用赋值运算符和命令 assign 两种方法完成赋值操作，下面分别介绍由赋值运算符完成的基本赋值操作和标识符的释放操作，最后介绍命令 **assign** 的使用。

1. 基本的赋值操作

基本的赋值操作是由赋值运算符:=来完成的，赋值运算符由紧接的冒号和等号构成，二者之间不能含有其他字符。由赋值运算符构成的命令也称为赋值语句，是 Maple 中最简单的语句。注意赋值运算符与等号不同，在 Maple V 中等号是用来构成方程和关系表达式中的等式的。

赋值语句的基本格式为：

name := expr;

其中 **name** 代表任意合法的标识符，包括简单标识符、复杂标识符或者索引标识符；

expr 代表任意合法的表达式，包括标识符、数字、字符串、代数表达式、方程或者表达式的序列等，在 **expr** 中不能含有赋值运算符。下面是几个简单的赋值操作：

```
> mystring:="I am a string";
```

$$mystring := \text{"I am a string"}$$

```
> myvalue:=4;
```

$$myvalue := 4$$

```
> T[plasma]:=5000;
```

$$T_{plasma} := 5000$$

```
> `value of Pi`:=evalf(Pi);
```

$$value\ of\ Pi = 3.141592654$$

同时，用户也可以一次对多个标识符赋值，命令格式如下：

name_1,...,name_n:=expr_1,...expr_n;

用来把表达式 **expr_1**，…，**expr_n** 分别赋值给标识符 **name_1**，…，**name_n**。下面的赋值语句同时对三个变量赋值，接着引用了它们的值：

```
> a,b,c:=1,2,3;
```

$$a,b,c = 1,2,3$$

```
> b;
```

$$2$$

为简单起见，这里把只代表自身的标识符称为无值标识符，把赋予了表达式或其他值的标识符称为有值标识符。无值标识符是自由的，当它出现在表达式中时代表未知的变量。用户可以用命令 **assigned(name)**来查看标识符 **name** 是否是有值标识符。如果 **name** 是有值标识符，命令返回逻辑常量 **true**；如果 **name** 是无值标识符，命令返回 **false**。在下面的例子中，**T[plasma]**已被赋值为 **4000**，而 **T**、**new** 没被使用过：

```
> assigned(T[plasma]),assigned(T),assigned(new);
```

$$true, false, false$$

用户既可以对无值标识符赋值，也可以对有值标识符赋值。标识符总是具有最新一次赋值操作时赋予的值，对有值标识符再次赋值后原来的赋值操作自动失效。

在 **name := expr** 中，如果 **expr** 是其他无值标识符，命令执行的结果将使 **name** 代表标识符 **expr**，这时 **name** 成为有值标识符。看下面的例子：

```
> a:=`a symbol`;assigned(a);
```

$$a := a\ symbol$$

$$true$$

```
>a[1];
```

$$a\,symbol_1$$

2. 标识符的释放

在工作单中对标识符赋值后，Maple V 把标识符的值存贮在系统的缓存中，因而对整个工作空间中的任一工作单都是有效的。当标识符不再使用或者需要代表未知数时，用户需要清除当前标识符所代表的意义，这就是标识符的释放。标识符释放后，只代表其自身，成为自由的无值标识符。释放标识符的基本命令为

name := 'name';

其中 **name** 可以是任意的合法标识符，赋值运算符后面的标识符用单引号界定。命令 **evaln(name)**可以执行等价的操作。下面是几个具体的例子：

```
>`a symbol`:=3;
```

$$a\,symbol := 3$$

```
>`a symbol`:='`a symbol`';
```

$$a\,symbol := a\,symbol$$

```
>`a symbol`;
```

$$a\,symbol$$

```
>a[1]:=3
```

$$a_1 := 3$$

```
>a[1]:=evaln(a[1]);a[1];
```

$$a_1 := a_1$$

$$a_1$$

在上面的操作中，单引号的作用是使所界定的标识符代表其自身，命令 **evaln** 的作用是使其参数代表其自身。当用赋值运算符把标识符的自身赋值给自身时，标识符也就成为了无值标识符，即标识符被释放了。

命令 **evaln** 的参数只能是标识符，而单引号还可以用来界定其他表达式，例如由算术运算符构成的算术表达式，或者由关系运算符构成的关系表达式，或者其他表达式和表达式序列。这时单引号的作用是使所界定的表达式代表其自身，即返回的结果与原表达式形式完全一样。Maple V 不对用单引号界定的表达式执行任何化简、变量替换、求值等运算，这里的单引号延缓了对表达式的化简、变量替换、求值等运算。这种由单引号界定的表达式称为缓值表达式。缓值表达式使用户可以书写数学运算中常用的计算等式，这种用法在前面的例子中见到过，下面给出了几个针对性的例子：

```
>x:=2:y:=3:
> 'x+y'=x+y;
```

$$x + y = 5$$

```
> 'sin'(x),'cos(x)';
```

$$\sin(2),\cos(x)$$

```
>"sin"(Pi);
```

$$'\sin'(\pi)$$

```
>%;
```

$$\sin(\pi)$$

```
>%;
```

$$0$$

如果要释放当前所使用的所有标识符，可以使用命令 **restart**。命令 **restart** 的作用是释放 Maple V 系统缓存中的所有信息，使整个工作空间恢复到 Maple V 刚启动时的状态，因此所有的标识符都成为无值标识符。命令 **restart** 没有参数，执行后没有输出。下面给出的例子说明了命令 **restart** 执行前后变量 **myvalue** 值的变化：

```
>myvalue:=evalf(Pi,6);
```

$$myvalue := 3.14159$$

```
>myvalue;
```

$$3.14159$$

```
>restart;
>myvalue;
```

$$myvalue$$

3. 命令 **assign**

命令 **assign** 也可以用来完成赋值操作。命令的格式为：**assign(a=b)**或 **assign(a, b)**，其中 **a** 代表任意的无值标识符，**b** 代表任意的单一表达式，而不能是表达式序列。命令的作用与赋值语句 **a:=b** 等价，不同的是它不给出任何输出。如果 **a** 是有值标识符，则命令将引发错误。看下面的例子：

```
>assign(str,"a atring");
>str;
```

$$\text{"a atring"}$$

```
> a:='a': assign(a=3):
  a;
```

$$3$$

```
> assign(a=4);
Error,(in assign)invalid arguments
```

如果在命令 **assign(a=b)**或 **assign(a,b)**中，**b** 也是无值标识符，则可以使用 **assign** 命令对 **a** 再次赋值，这时 **b** 与 **a** 等值。如果在此之后又使用赋值运算符对标识符 **a** 执行了赋值操作，**b** 的值不会随 **a** 的值而改变。

```
>a:='a':b:='b':assign(a=b);
>assign(a=3);
>a,b;
```

$$3,3$$

```
> a:=5:a,b;
```

$$5,3$$

assign 命令在求解方程组时使用较多，这时其参数可以是方程的集合或列表，下面给出一个简单的例子：

```
>sol:=solve({x+y=2,x-y=3},{x,y});
```

$$sol := \{y = \frac{-1}{2}, x = \frac{5}{2}\}$$

```
>assign(sol);
>x,assigned(x);
```

$$\frac{5}{2}, true$$

3.3　表达式

表达式是 Maple V 中的基本操作对象。运算符是表达式中数字、字母、标识符等对象的连接符号，Maple V 对这些连接符号赋予了特殊的意义，使其能够完成相应的运算或者操作。表达式树是表达式在 Maple V 中的内部描述，其结构决定了表达式的数据类型。范围表达式用来形成序列，在 Maple V 的很多命令中都可以使用。字符串不是科学计算过程中必须的，主要是用作一些对象的描述信息，如图像的标题等。

本节首先简单介绍 Maple V 的运算符，接着介绍用来描述表达式内部结构的表达式树，由此分析表达式的数据类型。最后介绍范围表达式和字符串两种比较简单的表达式。

3.3.1　Maple V 的运算符

本小节简单介绍构成各种表达式的运算符，包括运算符的分类和运算符的优先级和结合性。

1. 运算符的分类

Maple V 中运算符的范围很广，除了常用的算术运算符、关系运算符和逻辑运算符之外，把小数点、逗号以及实现函数定义、复合函数运算、集合运算和类型声明等的符号都作为运算符。运算符的作用对象称为操作数，按照操作数的数量，Maple V 中的运算符可以划分为二元运算符、一元运算符、独立运算符三类。其中，二元运算符需要两个操作数，一元运算符需要一个操作数，而独立运算符则不需要操作数。

赋值运算符、算术运算符、关系运算符、集合运算符都属于二元运算符，逻辑运算符中的与（and）、或（or）也属于二元运算符。另外，小数点、逗号以及用来实现函数定义、复合函数运算和类型声明等的符号也属于二元运算符。表 3.3 中列出了 Maple V 中的二元运算符及其所代表的意义。

表 3.3　　Maple V 中的二元运算符

运算符分类	运算符	运算符的意义	运算符分类	运算符	运算符的意义
算术运算符	+	加号	复合函数运算符	@	单复合
	-	减号		@@	重复复合
	*	乘号	集合运算符	union	集合并集
	/	除号		minus	集合差集
	**	乘方		intersect	集合交集
	^	乘方	二元逻辑运算符	and	逻辑与
	mod	取模		or	逻辑或
关系运算符	<	小于	其他的二元运算符	->	函数定义运算符
	<=	小于等于		..	范围运算符
	>	大于		$	二元序列运算符
	>=	大于等于		.	串联运算符或用作小数点
	=	等于		,	表达式分隔符
	<>	不等于		::	类型定义
赋值运算符	:=	赋值		&<string>	二元不确定算子

一元运算符包括正负号、小数点、阶乘运算符等，表 3.4 中列出了 Maple V 中的一元运算符，它所代表的意义及与操作数作用时的位置。

表 3.4　　Maple V 中的一元运算符

运算符	运算符的意义	运算符的位置
+	正号	前缀
-	负号	前缀
!	阶乘	后缀
$	一元序列运算符	前缀
not	逻辑非	前缀
&<string>	一元不确定算子	前缀
.	小数点	前缀或后缀

Maple V 还提供了三个特殊的独立运算符，它们是：%、%%、%%%，分别代表了

最近三次非空的输出结果或表达式。用户也可以把这三个运算符作为 Maple V 工作空间中的环境变量。这三个运算符为用户再次调用没有名字的输出对象提供了方便，在执行一些临时性的工作时非常有用。在本书的很多例子中都包含了这三个独立运算符的使用，这里不再具体介绍。

2. 运算符的优先级和结合性

表达式在运算或求值过程中各种运算符的执行顺序称为运算符的优先级，而运算符与操作数的结合方向称为运算符的结合性。

总的来说，小数点的优先级最高，算术运算符的优先级次之，赋值运算符的优先级最低，逗号的优先级同样很低；一元运算符的优先级总是高于同类运算符中的二元运算符。就最常用的几种运算符而言，算术运算符的优先级高于关系运算符，关系运算符的优先级高于逻辑运算符。算术运算符、关系运算符和逻辑运算符中的不同运算符的优先级，读者在其他的高级语言中都有接触，这里不再详述。举例来说，在算术运算符中总是先乘除再加减，乘方的优先级最高；关系运算符一般不出现在同一关系表达式中，其优先级是等同的；逻辑运算符中，逻辑非（not）的优先级最高，接着是逻辑与（and）和逻辑或（or）。

二元运算符的结合性一般都是从左到右，一元运算符的结合性与其所处的位置有关。表 3.4 中给出了一元运算符的位置，前缀运算符的结合性为从右到左，后缀运算符的结合性为从左到右。

和其他的高级语言一样，在表达式中可以用圆括号来改变运算符的优先级和结合性。圆括号可以嵌套使用。

3.3.2 表达式树和数据类型

表达式树是表达式在 Maple V 中的内部描述，不同的表达式具有不同的表达式树结构。Maple V 根据表达式树的结构来划分表达式的类型。下面举例介绍表达式树的结构。

考虑下面的表达式：

```
> f:=sin(x)+2*cos(x)^2*sin(x)+3;
```

$$f := \sin(x) + 2\cos(x)^2\sin(x) + 3$$

Maple V 在内部构建了如图 3-1 所示的表达式树，该表达式树的第一个节点即根节点是加号，因此表达式 f 的数据类型标识为`+`（和）。下面是用命令 **whattype** 和 **type** 分析表达式的类型，由于加号“+”是 Maple V 的算术运算符，用作数据类型标识时需要用反引号界定，否则是不合法的。

```
> whattype(f);
                              +
```

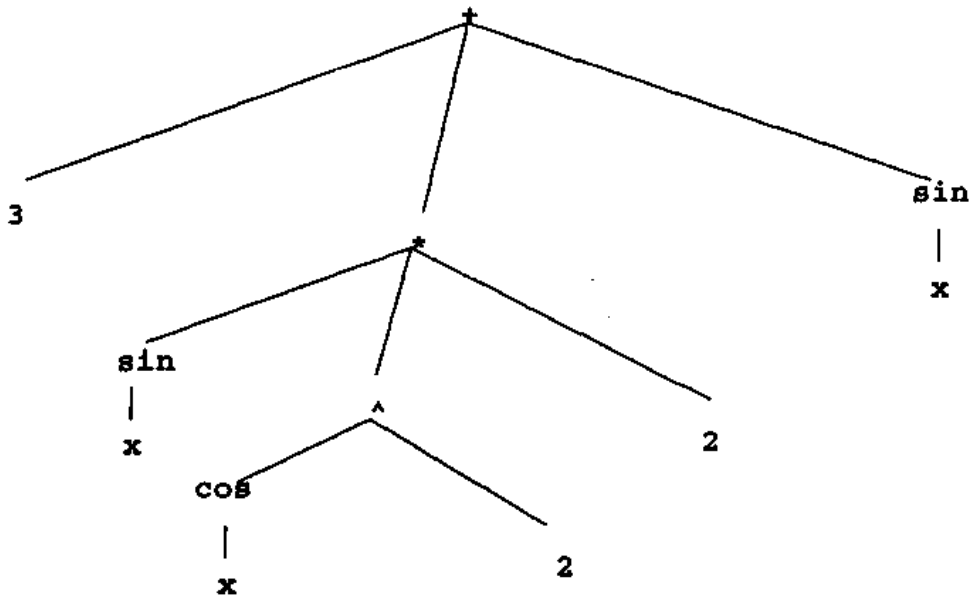


图 3-1 表达式树示例

```
>type(f,`+`);
                                    true
>type(f,+);
`|`unexpected
```

由于减法式子 **a-b** 与加法式子 **a+(-b)**等价，分式 **1/x** 与 **x^(-1)**，**a/b** 与 **a*b^(-1)**等价，因此减号 `-`、除号`/` 不是 Maple V 的数据类型标识。分式 **1/x** 的数据类型为`^`（幂），**a/b** 的数据类型为`*`（积）。看下面的命令：

```
>x:='x':y:='y':
>whattype(x-y),whattype(1/x),whattype(x/y);
                              +,^,*
>type(x-y,`-`);
Error,type`-`does not exist
```

图 3-1 中的表达式树在根节点下有三个分支，分别与表达式中的三项加数 **sin(x)** 、**2*cos(x)^2*sin(x)** 、**3** 对应。每个分支都代表原表达式中的一个子表达式，分支节点的符号或标识代表了该分支对应的子表达式的类型，最底层的节点为叶子节点。根节点下的子表达式称为表达式的主操作数，是根节点对应的运算符或函数作用的对象。

用户可以使用表 3.1 中提供的几个命令来分析表达式的结构，前面给出了命令 **whattype**、**type** 使用的例子，下面以图 3-1 中表示的表达式 **f** 为例，用户可以结合表 3.1 中对命令作用的说明，分析下面的结果：

```
> nops(f);
```

$$3$$

```
> op(1,f),op(2,f),op(3,f);
```

$$\sin(x), 2\cos(x)^2\sin(x), 3$$

```
> has(f,cos(x)),has(f,2);
```

$$\text{true, true}$$

```
> g:=subsop(3=x^2,f);
```

$$g := \sin(x) + 2\cos(x)^2\sin(x) + x^2$$

命令 **op(i,f)**分离出表达式中的子表达式，命令 **subsop(i=g,f)**以表达式 **g** 替换表达式 **f** 中的第 **i** 项，原来的表达式不发生变化。其中，序号 **i** 不能大于表达式 **f** 中操作数的个数 **nops(f)**。命令 **has(f, x)**用来分析表达式 **f** 中是否包含子表达式 **x**。这里的表达式通常指的是单个的表达式，不包括由多个表达式构成的表达式序列。

用户可以用命令 **op** 分离出表达式中的某一子表达式，分析其结构或者进行其他操作。以表达式 **f** 的第二项为例，分析下面的结果：

```
> t:=op(2,f);
```

$$t := 2\cos(x)^2\sin(x)$$

```
> whattype(t),nops(t);
```

$$*, 3$$

```
> op(1,t),op(2,t),op(3,t);
```

$$2, \cos(x)^2, \sin(x)$$

```
> whattype(op(1,t)),whattype(op(2,t)),
  whattype(op(3,t));
```

$$integer, ^{\wedge}, function$$

结果表明子表达式 **t** 是 **2**、**cos(x)^2**、**sin(x)**三个因子的乘积，这三个因子的数据类型分别为 **integer**（整数）、`^`（幂）和 **function**（函数）。由此可见，Maple V 中表达式类型的划分是很细的。

另外，命令 **op** 还有下面几种有用的形式：

- **op(i..j,f)**：返回由表达式 **f** 中第 **i** 到第 **j** 个操作数构成的序列。
- **op(f)**：返回表达式 **f** 中的所有操作数序列，与 **op(1..nops(f),f)**等价。
- **op(0, f)**：返回表达式 **f** 的类型名。唯一的例外是当表达式 **f** 是函数时，返回具体的函数名。

同样以图 3-1 代表的表达式 **f** 为例，看下面几个命令的结果：

```
> op(0,f);
```

$$+$$

```
> op(f);
```

$$\sin(x), 2\cos(x)^2\sin(x), 3$$

```
> op(2..3,f);
```

$$2\cos(x)^2\sin(x), 3$$

```
> op(0,op(1,f));
```

$$\sin$$

3.3.3 范围表达式

范围表达式是由范围运算符和有序表达式构成的表达式，其基本格式为：**expr_1..expr2**。其中“..”为范围运算符，当多个小数点“.”连续出现时，Maple V 仍然认为是范围运算符“..”；**expr_1** 和 **expr_2** 分别代表两个有序的表达式，称为范围表达式的左边界和右边界，通常为数字常量、数字变量或者单个字母。这里的单个字母作为字符串使用，因此必须用双引号界定，否则被认为是变量。

在数字范围表达式中，以左边界为基数，当左边界小于右边界时，对应的增量为 **1**，反之为 **-1**，由此构成了一个数列，数列的最后一个元素值不超过右边界所限定的数。如果范围表达式的两个操作数为复数，则代表了复平面上的一个矩形范围。字母范围表达式是字母表中由左边界字母和右边界字母限定范围内的字母构成的字母序列。

下面给出了几个范围表达式的例子：

```
> r1:=2..3;
```

$$r1 := -2..3$$

```
> r2:=17/4...3/4;
```

$$r2 := \frac{17}{4}..\frac{3}{4}$$

```
> r3:="a".."d";
```

$$r3 = \text{"a"}..\text{"d"}$$

```
> r4:=x1..x2;
```

$$r4 := x1..x2$$

```
> cr:=1+I..3+4*I;
```

$$cr = 1 + I..3 + 4I$$

范围表达式对应的类型名为`..`或 **range**，包含 **expr_1**、**expr2** 两个操作数。用户可以使用表 3.1 中的几个命令来分析范围表达式的结构。看下面的例子：

```
whattype(r1),type(r2,range),type(r4,`..`);
```

$$..., true, true$$

```
>nops(r1),op(0,r2);
```

2,..

```
>op(r3);
```

"a","d"

```
>has(r2,3/4),has(r3,a);
```

true, false

```
>whattype(cr);
```

另外，用户也可以用命令 **lhs**、**rhs** 分别表示范围表达式的左右两个操作数。对范围表达式 **r** 而言，命令 **lhs(r)**与命令 **op(1, r)**等价，命令 **rhs(r)**与命令 **op(2, r)**等价。看下面的结果：

```
>op(1,r1),lhs(r1);
```

-2,-2

```
>op(2,r3),rhs(r3);
```

"d","d"

3.3.4　字符串

在 Maple V 中也可以使用字符串。这里简单介绍字符串的构成和串联运算符的使用，最后简单总结 Maple V 中引号的用法。

1. 字符串的构成

通常，字符串是由一对双引号界定的任意顺序的字符序列。双引号不是字符串的组成部分，而是字符串的界定符。在实际使用中，用户可以定义任意长度的字符串，用命令 **length(s)** 可以计算字符串 **s** 的长度。下面是几个字符串的简单例子：

```
>"Hello,word!";
```

"Hello, word!"

```
>s:="I am a string";
```

s :="I am a string"

```
>length(s);
```

13

当字符串中包含双引号、反斜杠等特殊字符时，需要在这些特殊字符前添加反斜杠作为转义字符。这时 Maple V 忽略用作转义字符的反斜杠。看下面的例子：

```
>"a\"b";
```

"a\"b"

```
> length(%);
                                3
> "a\\"b";
                              "a \\b"
> length(%);
                                3
```

利用方括号和范围表达式可以引用字符串中的任意字符或子字符串。**s[m]**用来引用字符串 **s** 中的第 **m** 个字符，如果 **m** 为负整数，引用的是字符串中倒数第 **m** 个字符。**s[m..n]** 引用字符串中的第 **m** 到第 **n** 个字符，如果 **m** 和 **n** 是负整数，则引用的是字符串中倒数第 **m** 个字符到第 **n** 个字符；其中 **m** 和 **n** 必须全为正整数或全为负整数，且 **m** 不大于 **n**。引用的结果仍然为字符串，用双引号界定。

```
> s[3..6];
                              "am a"
> s[-6..-1];
                             "string"
```

字符串的类型名为 **string**，字符串表达式只有自身一个操作数。看下面的结果：

```
> op(0,s);
                              string
> typ(s,string),nops(s);
                              true,1
> op(s);
                           "I am a string"
```

2. 串联运算符

Maple V 使用小数点“.”作为串联运算符实现标识符或字符串的连接，由此构成新的标识符或字符串。串联运算符是二元运算符，需要左右两个操作数，运算的顺序为从右到左，串联表达式的格式为：**expr1.expr2**。左操作数通常是标识符或字符串，右边的操作数可以是标识符、字符串或者算术表达式。以左操作数为字符串为例，如果右操作数为标识符或字符串，串联表达式的运算结果是由左操作数和由操作数连接构成的新的字符串；如果右操作数为可计算的算术表达式，串联表达式的结果为由左操作数和右操作数的计算结果构成的字符串；如果右操作数中含有未知数，结果以串联表达式的形式给出。看下面的例子：

```
> n:=0:x:='x':
  "p".abc,"p"."abc";
  "p".(2*n-1),"p".(2*x-1);
                                    "pabc","pabc"
                                  "p-1","p".(2x-1)
```

当串联表达式的左操作数为标识符时，也具有类似的运算方式，只是输出的结果为新的标识符，这里不再举例。另外，右操作数也可以是由圆括号界定的序列或者范围表达式确定的序列，输出的结果是左操作数分别与序列中的元素串联后的结果构成的标识符序列或者字符串序列。序列中的算术表达式包含未知数时，以串联表达式的形式输出结果。看下面的例子：

```
> p.(abc,2*n-1,2*x-1);
  p.("e".."f");
                                    p.(abc,-1,2x-1)
                                        pe, pf
```

未运算或者不能运算的串联表达式的类型为`.`，有两个操作数。看下面的例子：

```
> c:='c':c:= p.(2*x-1);
  nops(c), whattype(c );
                                    c := p.(2x-1)
                                        2,.
```

3. 引号的用法

Maple V 中用到了单引号、双引号和反引号三种引号，三者的作用互不相同，简单归纳如下：

- 双引号：用作字符串的界定符，但不作为字符串的一部分。
- 单引号：使所界定的标识符或者表达式代表其自身，用来释放标识符的值或者延缓表达式的计算。
- 反引号：用作复杂标识符的界定符，但不作为标识符的一部分；在需要的时候，使所界定的保留关键字、保护名或运算符成为命令的合法参数。

第 4 章 数和多项式

数是 Maple V 中最基本的对象，数的运算是科学计算中最为基础的内容。多项式是 Maple V 中结构简单但应用最多的表达式。本章首先介绍了数的运算，接着介绍了多项式的有关内容，包括多项式的基本知识、多项式的变量替换和化简等内容。

4.1 数的运算

数字常量是 Maple V 中最简单的表达式。当乘方运算的指数不是整数时，乘方运算的结果可能是难以理解的。Maple V 具有强大的符号计算能力，符号表达式的算术运算是一个基本的体现。

本节首先介绍 Maple V 的数字常量，接着介绍数的基本运算，并对 Maple V 的乘方运算进行重点介绍，最后简单介绍符号表达式的算术运算。

4.1.1 数字常量

数字常量包括整数、分数、浮点数和复数等，其中整数是最基本的数据类型，其他几种数据类型的数字都是由整数构成的。各个数字常量之间可以实现相互的转换。下面分别介绍这几种数字常量以及这几种数字常量之间的转换，最后介绍 Maple V 中的随机数函数。

1. 整数

整数是最简单的数字常量，包括自然数、零和负整数。自然数是由任意顺序的数字构成的，Maple 忽略任何前导的零。自然数的最大长度与系统有关，在 32 位计算机上，自然数的最大长度可以达到 524280 个十进制数字；在 64 位计算机上，最大长度为 38,654,705,646。假设 **natural** 是自然数，**+natural** 和**-natural** 表示带符号的整数，**+natural** 与 **natural** 代表同一个整数，**-natural** 代表负整数。用户可以用命令 **length** 得到整数的长度，负号不计入长度中。下面给出了几个整数的例子：

```
> a:=00002348092;
```

$$a := 2348092$$

```
> length(a);
```

$$7$$

```
>-12345678987654321;
 length(%);
                              -12345678987654321
                                17
```

整数的类型名为 **integer**，表 4.1 中给出了整数的子类型，**type** 命令可以识别这些子类型，**whattype** 命令只能识别 **integer** 类型。Maple V 把数字常量也作为表达式，整数常量表达式只有一个操作数，即整数本身，命令 **op**、**nops** 等可以对整数作用。

表 4.1　　**整数的子类型**

整数分类	类型名	举　例
偶数	even	0、2、-2
奇数	odd	1、-1
素数	prime	2、3、5
正整数	posint	1、2、32
负整数	negint	-1、-3、-9
非负整数	nonnegint	0、2、99
非正整数	nonposint	0、-3、-87

读者可以用命令**?integer** 获得关于整数的帮助信息。下面举例分析整数的类型和结构：

```
>a:2:b:=-7
 Whattype(a),whattype(b);
                                integer,integer
>type(a,even),type(a,odd)type(a,prime);
 type(b,posint),type(b,negint),type(b,prime);
                                true, false,true
                                false,true, false
```

通常，在数字的中间不能含有空格和其他非数字字符，通常所用的数字分节符在 Maple V 中也是不合法的。如果数字的长度超过了当前工作单窗口的宽度，Maple V 将自动换行；如果数字的输出结果不能在一行显示，Maple V 将自动在换行位置补充反斜杠。下面的例子是缩小当前工作单窗口后得到的：

```
>12345678909876543
 210123456789
123456789098765432\
 10123456789
```

```
>length(%);
                29
```

同时，用户可以在数字中手动加入反斜杠，实现数字常量的强制换行或者把数字分为几个容易识别的部分，反斜杠在这里充当续行符或者数字分节符的作用。Maple V 忽略数字中的反斜杠，反斜杠作为转义字符，不作为数字的一部分；输出数字时，如果能在同一行显示，其中的反斜杠自动消失。看下面的例子：

```
>123456789\987654321\
 0123456789;
                1234567899876543210123456789
```

另外，如果在输入命令时必须在变量名或者其他标识符的中间换行，也可以在换行位置处加入反斜杠，Maple V 忽略标识符中的单个反斜杠。

2. 分数

分数也是 Maple V 的一种数字常量，分数具有 $\frac{interger}{natural}$ 的形式，分子为整数，分母为自然数，分子和分母之间没有大于 1 的最大公约数。Maple 能自动对具有 $\frac{interger}{integer}$ 形式的数进行化简约分；化简后分数的分母为正整数，符号归入分子中；如果化简后分母为 1，Maple 自动把结果转化为相应的整数。看下面的例子：

```
>-30/12,567/3456;
```

$$\frac{-5}{2}, \frac{21}{128}$$

```
>144/12;
```

$$12$$

分数的类型名为 **fraction**，分数和整数都是 **rational**（有理数）的子类型。分数作为表达式，有分子和分母两个操作数，操作符为除号“/”。可以使用命令 **whattype**、**type**、**op**、**nops** 等分析分数的类型。同时也可以使用命令 **numer** 和 **denom** 引用分数的分子和分母。看下面的例子：

```
>x:=6/(-4);
```

$$x := \frac{-3}{2}$$

```
> whattype(x),type(x,rational);
                          fraction,true
> nops(x),op(1,x),op(2,x);
  numer(x),denom(x);
                                  2,-3,2
                                  -3,2
```

用户可以使用命令**?fraction** 获得关于分数的帮助信息。

3. 浮点数

浮点数可以分为正的浮点数和负的浮点数，正浮点数的符号可以省略。浮点数可以分成尾数部分和指数部分，尾数部分和指数部分中间不能含有空格；尾数部分不能为空，指数部分可以为空。以正的浮点数为例，尾数部分可以有三种形式：**integer.nature** 、**integer.** 和 **.nature**；但含有指数部分时，尾数部分不能采用 **nature.**的形式。指数部分是用字母 **E** 或者 **e** 前导的整数来表示的，负整数也不用小括号界定；如果指数部分只有字母 **E** 或 **e**，指数默认为 **0**。

下面是合法的浮点数的例子：

```
> 1.2,-2.,+.2;
                                  1.2,-2.,.2
> 2e2,1.2E+2,-.2e-2;
                                200.,120.,-.002
```

下面的浮点数是不合法的：

```
> 1.e2;
missing operator or `;`
> 1.2 e2;
missing operator or `;`
```

下面的两个命令是不同的，第一个命令是由浮点数 **.2** 与 **-1** 构成的加法算式，第二个命令是合法的浮点数：

```
> .2e -1;
                                   -.8
> .2e-1;
                                   .02
```

浮点数的类型名为 **float**，在 Maple V 中 **float** 不是 **rational**（有理数）的子类型，**float** 和 **rational** 都属于 **numeric**（实数）数据类型。浮点数常量作为表达式，包含两个操作数，第一个操作数为所有有效数字和符号构成的整数，第二个操作数为指数。可以使用命令 **whattype**、**type**、**op** 和 **nops** 分析浮点数的类型和结构。看下面的例子：

```
> f:=1e-2;
```

$$f := .01$$

```
> whattype(f),type(f,numeric);
```

$$float, true$$

```
> nops(f),op(1,f),op(2,f);
```

$$2,1,-2$$

可以使用命令 **Float(m,e)**输入浮点数，其中 **m** 代表浮点数的有效数字，必须为整数，否则不能给出结果；**e** 代表以 **10** 为底的指数。命令 **Float** 的首字母必须大写。看下面的例子：

```
> Flot(34,3);
```

$$34000.$$

```
> Float(3.4,2);
```

$$\text{Float}(3.4,2)$$

Maple V 还提供了一些与浮点数有关的函数，下面给出了其中几个最常用的函数：

- **trunc(x)**：当 **x>0** 时，返回比 **x** 小的最大整数；当 **x<0** 时，返回比 **x** 大的最小整数。
- **frac(x)**：返回 **x** 的小数部分。
- **round(x)**：对 **x** 进行四舍五入，返回与 **x** 最接近的整数。
- **floor(x)**：返回不大于 **x** 的最大整数。
- **ceil(x)**：返回不小于 **x** 的最小整数。

下面是这几个函数使用的例子，其中用方括号界定的表达式序列是 Maple V 中的列表数据类型，请参看第 7 章。读者可以比较这几个函数之间的差别：

```
> [trunc(3.6),frac(3.6),round(3.6),floor(3.6)
 ,ceil(3.6)];
```

$$[3,6,4,3,4]$$

```
> [trunc(-3.6),frac(-3.6),round(-3.6);
 floor(-3.6),ceil(-3.6)];
```

$$[-3,-6,-4,-4,-3]$$

4. 复数

默认情况下，Maple V 用大写字母 **I** 来代表复数单位 $\sqrt{-1}$；复数 **a+bi** 的输入格式为 **a+b*I**，**a** 和 **b** 都是实数。复数包括实部和虚部，可以用命令 **Re(z)**和 **Im(z)**分别引用复数的实部或虚部。复数的类型名为 **complex**，实数 **numerical** 是复数 **complex** 的一个子类型。按照实部和虚部的数据类型可以把复数细分为五种类型，表 4.2 中列出了这五种复数类型

的类型名和对类型的描述。用户可以使用 **type**、**op** 和 **nops** 等命令分析复数常量的类型和结构。下面给出了复数的例子：

在上面的例子中，由于复数 **c** 表现为实数部分和虚数部分相加的形式，所以命令 **whattype** 的输出为 `+`；如果复数只含有虚数部分，Maple V 认为是虚部和虚数单位 **I** 的乘积，命令 **whattype** 的输出为 `*`；如果是复数 **I**，由于 **I=(-1)^(1/2)**，命令 **whattype** 的输出为`^`。看下面的例子：

```
> 1+2.*I,2/3+4/3*I,1-1/2*I;
```

$$1.+2.I,\frac{2}{3}+\frac{4}{3}I,1-\frac{1}{2}I$$

```
> c:=3-2*I;
```

$$c:=3-2I$$

```
> Re(c),Im(c);
```

$$3,-2$$

```
> whattype(c),type(c,complex),type(c,complex(integer));
  nops(c),op(1,c),op(2,c);
```

$$+,true,true$$

$$2,3,-2I$$

```
> whattype(-2*I),whattype(I);
```

$$*,\hat{}$$

表 4.2　　复数 a+bi 的五种可能类型

类型名	类型描述
complex(integer)	实部和虚部都是整数，包括 0
complex(fraction)	实部和虚部都是分数
complex(rational)	实部和虚部都是有理数
complex(float)	实部和虚部都是浮点数常量
complex(numeric)	包含上面的每种情况

5. 数字常量之间的强制类型转换

在前面介绍的几种数据类型中，**integer** 是范围最小的数据类型；**complex** 是范围最大的数据类型，它包含了其余的数据类型。整数、分数、浮点数都同属于 **numeric**（实数）类型和 **complex**（复数）类型；整数也可以看成是分母为 1 的分数，整数和分数都可以看作是数字的准确形式；浮点数是数字的近似表示。

在数字运算的过程中，Maple V 可自动进行类型转换：整数转换为分数，分数转换为

浮点数。同时，Maple V 允许用户进行强制类型转换，主要包括整数转换为浮点数、分数转换为浮点数、浮点数转换为分数三种转换。其余的转换通常是有条件的，Maple V 总是以尽量不影响数字的准确度为标准，各种转换之间基本上是等值的。

强制类型转换的命令为 **convert**，命令 **convert** 可以实现 Maple V 中多种表达式之间可能的转换，这里介绍数字常量之间的转换。命令 **convert(int_num, float)**实现整数到浮点数的转换，命令 **convert(fra_num, float)**实现分数到浮点数的转换，命令 **convert(flt_num, fraction)**实现浮点数到分数的转换；其余的如整数到分数的转换是没有意义的，分数到整数的转换和浮点数到整数之间的转换往往是不可行的。看下面的例子：

```
> convert(2,float);
```

$$2$$

```
> convert(0.34,fraction);
```

$$\frac{17}{50}$$

```
> convert(1/3,float);
```

$$.3333333333$$

在 Maple V 中，计算结果的小数位数是由环境变量 **Digits** 的值来设定的。**Digits** 的默认值为 **10**，因此上面的命令将 **1/3** 转换为浮点数时的小数位数是 **10** 位；用户可以通过直接赋值的方式改变 **Digits** 的值。在进行浮点数到分数的转换时，输入浮点数的小数位数会对转换后的分数产生细微的影响，读者可以比较下面的结果：

```
> convert(0.3333,fraction),
  convert(0.333333333,fraction),
  convert(0.3333333331,fraction),
  convert(0.3333333333,fraction);
```

$$\frac{3333}{10000}, \frac{333333333}{1000000000}, \frac{1}{3}, \frac{1}{3}$$

6. 随机数函数

首先介绍随机数函数 **rand**。随机数函数的基本调用方式为 **rand()**，函数的返回值为一个 12 位的非负整数。这种调用是不带参数的，但是小括号不能省略。看下面的例子：

```
> rand();
```

$$321110693270$$

```
> rand();
```

$$343633073697$$

```
>rand;
```

$$rand$$

同时，用户也可以用 **rand(r)**方式调用随机数函数，其中的参数 **r** 通常为范围表达式确定的连续整数段；参数 **r** 也可以是正整数，这时默认的范围表达式为 **0..r-1**；其他的参数都是不合法的。带参数的随机数函数调用的结果实际上定义了一个过程，下面的例子给出了带参数的随机数函数使用的常见方法：

```
>die:=rand(-9..9):
 die();
```

$$-5$$

```
>die();
```

$$3$$

4.1.2　数的基本运算

数的基本运算包括数的加、减、乘、除、乘方等，分别用相应的算术运算符来完成。乘方运算在 4.1.3 节中介绍。命令 **evalf** 是 Maple V 中的一个常用命令，它用来把具有准确形式的表达式转换为浮点数形式的近似表达式。本小节首先介绍数的基本运算规则，接着介绍命令 **evalf** 的使用，最后介绍复数的运算。

1. 基本的运算规则

算术运算符的优先级和数学中的惯例一样，可以使用小括号改变算术运算符的优先级和结合性，使用方括号或者大括号改变计算顺序是不合法的。下面是几个数的运算的例子：

```
>121+542-3*99-100;
 121+(542-3)*(99-100);
```

$$266$$

$$-418$$

```
>3/2+9/7*5/3;
```

$$\frac{51}{14}$$

```
>3.4e-2-2.3*1.5E-3;
```

$$.03055$$

上面给出的例子都是相同数据类型的数字之间的运算。各种数字常量之间可以进行混合运算；在进行混合运算时，不同类型的数字要转换成同一种类型，然后再进行计算。Maple V 将自动进行数据类型的转换，Maple V 总是试图用最准确的形式表示计算的结果，因此数字常量转换的方向是：整数转换为分数，分数转换为浮点数。只有在输入的运算表达式中包含浮点数的时候，Maple V 才采用浮点数的形式表示计算的结果。看下面的例子：

```
> 2+1/3-3/5;
```

$$\frac{26}{15}$$

```
> 2.+1/3-3/5;
```

1.733333333

在进行其他复杂的运算时，Maple V 遵循同样的标准，即总是用尽可能准确的形式表示计算的结果。如果用户输入的表达式不含浮点数，并且没有调用返回浮点数结果的函数或命令，则 Maple V 给出的运算结果通常没有任何近似。当然，这种准确的计算结果可能是用数学中的常用符号来表示的，例如根号、幂、积分符号等；也可能是用 Maple V 的命令本身或者专有格式来表示的，Maple V 能够对这种结果进行后续运算或者操作。这是 Maple V 进行符号计算的最大特征。

2. 命令 evalf

Maple V 的符号计算能力使得计算的结果可以表示为最准确的形式，但很多时候得到的准确计算结果是非常复杂或者是难于理解的，用户可能更希望得到直观的近似结果。Maple V 提供了把准确形式的结果转化为浮点数近似结果的命令，其中最常用的是命令 **evalf**。下面介绍命令 **evalf** 的使用规则。

命令 **evalf** 用来把非浮点数的计算结果或者表达式转化为浮点数形式的近似结果。其基本的命令格式为 **evalf(expr, n)**，其中 **expr** 代表任意的算术表达式，**n** 代表计算结果的精度；**n** 缺省时，采用环境变量 **Digits** 的值。命令 evalf 能够识别 Maple V 的系统常量，如 **Pi**、**exp(1)**和 **gamma** 等；也能够识别 Maple V 的基本数学函数，如 **sin**、**cos**、**exp**、**ln** 等等。命令 evalf 不仅仅局限于实数范围内，它还能够把复数的实部和虚部分别转化为浮点数形式。

命令 **evalf** 的使用还是比较简单的，看下面的例子：

```
> evalf(Pi);
  evalf(Pi,20);
```

3.141592654

3.1415926535897932385

```
> evalf(345678921/123456789);
```

2.799999286

```
> evalf(3/4*x^2+1/3*x-2);
```

$$.7500000000x^2 + .3333333333x - 2.$$

实际上，命令 **evalf** 几乎可以作用于 Maple V 的所有表达式。如果输入的表达式中含有浮点数，Maple V 也常常自动调用命令 **evalf** 来完成浮点数运算并给出浮点数形式的结果。关于命令 **evalf** 作用于其他表达式的情况，在随后的章节中会分别介绍，这里不再举例。

3. 复数的运算

Maple 可以自动地完成复数的各种运算。Maple V 能够把基本的算术运算和数学函数作用在复数上，并按照复数范围内的定义进行计算，计算的结果总是尽可能表示为实部和虚部相加的形式。同时，可以使用命令 **evalc** 把复数运算的结果写成实部和虚部相加的形式，或者使用命令 **Im** 和 **Re** 分离出复数的实部和虚部。看下面的例子：

```
> sqrt(-9),sqrt(-1);
```

$$3I, I$$

```
> (1-I)^4,(3+I)/(4-I);
```

$$-4, \frac{11}{17} + \frac{7}{17} I$$

```
> exp(2+3*I);
```

$$e^{(2+3I)}$$

```
> evalc(%);
```

$$e^2 \cos(3) + I e^2 \sin(3)$$

当命令 **evalf** 对复数表达式作用时，结果通常也表示为实部和虚部相加的形式，实部和虚部都是浮点数实数。看下面的例子：

```
> sqrt(3-I);
  evalf(%);
```

$$\sqrt{3 - I}$$

$$1.755317302 - .2848487846 I$$

另外，可以用命令 **conjugate(expr)**计算复数表达式 **expr** 的共轭复数，**abs(expr)**计算复数表达式的模，**argument(expr)**计算复数表达式的辐角。看下面的例子：

```
> a:=3-4*I;
  conjugate(a);
  abs(a),argument(a);
```

$$a := 3 - 4I$$

$$3 + 4I$$

$$5, -\arctan\left(\frac{4}{3}\right)$$

4.1.3 乘方运算

Maple V 中乘方运算符用“^”或“**”表示，乘方运算符在几种算术运算符中具有最高的优先级。本小节将对 Maple V 中的乘方运算进行较为详细的介绍，包括乘方运算的输入和输出、非整数指数幂的化简规则和负实数的奇数次根等问题。

1. 乘方运算的输入和输出

乘方运算包括底数和指数两个操作数，当指数为分数或者是负数时需要用小括号界定。如果要表示复合乘方运算，需要用小括号规定乘方运算的结合性，**a^b^c** 的形式是不合法的。如果乘方运算的指数为整数，乘方运算的结果通常可以表示为整数、分数等准确的形式；如果乘方运算的结果不能化简为整数、分数等准确形式，输出采用 a^b 的形式表示，可以使用命令 **evalf** 将结果转化为近似值；如果乘方运算中含有浮点数，则结果表示为浮点数形式的近似值。

下面给出了几个乘方运算的简单例子：

```
> 2^3,2.^3;
```

$$8,8.$$

```
> 2^1/3;
  2^(1/3);
```

$$\frac{2}{3}$$

$$\left(\frac{1}{3}\right)$$

$$2$$

```
> evalf(2^(1/3));
```

$$1.259921050$$

```
> 2^(-1);
```

$$\frac{1}{2}$$

```
> 2^(2^2);
```

$$16$$

下面的两个命令是不合法的：

```
> 2^-1;
`-`unexpected
> 2^2^2;
`^`unexpected
```

2. 非整数指数幂的化简规则

如果乘方运算的指数不是整数，其结果的表示将比较困难，这也是 Maple V 中最有争议的一点。对复数 x，非整数指数 y 来说，x^y 的运算规则为 **exp(y*ln(x))**：其中 **exp(x)**是指数函数，**ln(x)**是对数函数。通常，Maple 不会自动把$(x^y)^z$ 化简为 $x^{(yz)}$，因为这种化简可能是不正确的；例如当 y=2，z=1/2 时，$\sqrt{x^2}=x$，并不总是成立的。只有当这种化简对所有的复数 x（有时排除 0 或∞）都绝对成立时，Maple V 才执行这种化简。对于数字常量的乘方运算，可以使用命令 **simplify** 对计算的结果进行化简。看下面的例子：

```
>8^(2/3);
```

$$8^{\left(\frac{2}{3}\right)}$$

```
>simplify(%);
```

$$4$$

如果$-1<a\le 1$ 且 b 为整数或 x 是正数，Maple 能够自动完成化简$(x^a)^b \rightarrow x^{ab}$。下面的例子中，第一个程序段内的两个乘方运算不符合化简的条件，第二个程序段内的三个乘方运算符合化简的条件：

```
>(x^(5/3))^(1/2),(x^(-1))^(1/2);
```

$$\sqrt{x^{\left(\frac{5}{3}\right)}}, \sqrt{\frac{1}{x}}$$

```
>(2^(5/3))^(1/2),(x^(5/2))^3,(x^(1/2))^(3/5);
```

$$2^{\left(\frac{5}{6}\right)}, x^{\left(\frac{15}{2}\right)}, x^{\left(\frac{3}{10}\right)}$$

同时，如果 a、b 不是整数，Maple 也不能自动进行这样的化简：$a^b c^b \rightarrow (ac)^b$，看下面的例子：

```
>2^(1/2)+3^(1/2)-2^(1/2)*3^(1/2);
```

$$\sqrt{2}+\sqrt{3}-\sqrt{2}\sqrt{3}$$

原因是把$\sqrt{2}\sqrt{3}$ 合并为$\sqrt{6}$ 后引入了新的第三个平方根。由于根的计算通常比较困难，因此 Maple V 尽量避免产生新的根。用户可以使用命令 **combine** 来完成这种平方根的合并。

```
>combine(%);
```

$$\sqrt{2}+\sqrt{3}-\sqrt{6}$$

3. 负实数的奇数次根问题

更让人难以理解的是：如果底数 a 为负数，指数是分母为奇数的分数，Maple V 给出的结果将是虚数！看下面的例子：

```
> (-8)^(2/3);
```

$$(-8)^{\left(\frac{2}{3}\right)}$$

```
> evalf(%);
```

$$-2.000000000+3.464101615I$$

这种情况是 Maple V 用 **exp(y*ln(x))**来计算非整数指数的幂 **x^y** 造成的，因为当 **x** 为负数时，**ln(x)**为虚数。但在大多数情况下，用户不希望得到这样的结果，Maple V 提供了命令 **surd** 可以避免这种意外的结果。

命令 **surd** 的基本格式为 **surd(x, n)**，其中 **x** 为复数表达式，通常 **n** 为正整数；命令的输出结果是复数 **x** 的 **n** 次方根中与 **x** 最接近的数；当 **n** 为负整数时，**surd(x, n)=surd(x, -n)^(-1)**。如果 **n=2**，则 **surd(x, n)=sqrt(x)**，**sqrt(x)**是 Maple V 的平方根函数。在实数范围内，如果 **n** 为奇数，**surd(x, n)**返回的是 **x** 的实数根；当 **x>=0** 时，**surd(x, n)=x^(1/n)**；当 **x<0** 时，**surd(x, n)=-(-x)^(1/n)**。看下面的例子：

```
> surd(-8,3),surd(8,3);
```

$$-2,2$$

```
> surd(-27/64,3);
```

$$\frac{-3}{4}$$

```
> surd(-27/64,3)^2;
```

$$\frac{9}{16}$$

```
> sqrt(4),sqrt(3),sqrt(-4);
```

$$2,\sqrt{3},2I$$

4.1.4 符号表达式的算术运算

上面所给的例子中，通常都是数字常量的运算。Maple V 的强大符号计算能力，不仅体现在能够把数字常量的运算结果表示为尽可能准确的形式，而且体现在它能够对作为未知数的符号变量进行运算，并对符号表达式进行化简。包括基本的算术运算在内，数字常量能够完成的运算，符号变量同样可以完成。这里以基本的算术运算为例，介绍符号运算

的一些基本规则。

假设 x 为符号变量或算术表达式，Maple 将自动进行下面的化简：

```
> x-x,x+x,x+0,x*x,x*1,x*0,x/x,0/x,x^0,x^1;
```

$$0, 2x, x, x^2, x, 0, 1, 0, 1, x$$

但是这些化简运算并不是对所有的 x 都是合法的，当 x 为无穷大数时，有些化简是不能进行的，或者与上面的化简规则不同。下面给出了这些例外的情况：

```
> infinity;
```

$$\infty$$

```
> infinity-infinity;
Error,invalid cancellation of infinity
> infinity/infinity;
Error,invalid cancellation of infinity
```

对整数 n、m，数字常量 a、b、c 及符号表达式 x、y、z 来说，Maple V 能根据加法、乘法的结合律、交换律和分配律进行下面的化简：

$$ax+bx \rightarrow (a+b)x$$
$$x^a \times x^b \rightarrow x^{(a+b)}$$
$$a(x+y) \rightarrow ax+ay$$

前两个式子意味着，Maple 能够自动合并多项式中的同类项和乘法算式中的同底数幂。第三个式子意味着，Maple 能把数字常量（包括整数、分数和浮点数）分配到符号多项式的每一项；如果 a 为非数字常量，则不执行这种化简。看下面的例子：

```
> x:='x':y:='y':
  2*x+3*x,x^2*x^(-5),2*(x+y);
```

$$5x, \frac{1}{x^3}, 2x+2y$$

```
> a:='a':b:='b':
  a*x+b*x,x^a*x^b,a*(x+y);
```

$$ax+bx, x^a x^b, a(x+y)$$

在符号表达式中，Maple V 总是默认其中的未知数或者符号变量为实数。因此，对含有符号变量的复数表达式来说，仍然可以使用命令 evalc 将其转换为实部与虚部相加的形式。看下面的例子：

```
> a:='a':1/(a-I);
```

$$\frac{1}{a-I}$$

```
> evalc(%);
```

$$\frac{a}{a^2+1}+\frac{I}{a^2+1}$$

4.2 多项式

变量替换是多项式求值的基本方法。通过多项式的化简，可以把多项式表示为需要的形式。分式是多项式的商，分式和多项式合称为有理式。本节首先介绍多项式的基本知识，接着介绍多项式的变量替换和化简，最后简单介绍分式的有关运算。

4.2.1 多项式的基本知识

在 Maple V 中，多项式是由数字常量、变量和“+”、“-”、“*”、“^”四种算术运算符构成的表达式。在多项式中，未知变量的指数只能是非负整数。多项式可以是匿名的，用户也可以对多项式命名。按照多项式中的变量个数，可以把多项式分为单变量多项式和多变量多项式两种。

1. 单变量多项式

单变量多项式是只包含一个未知变量的多项式，其中不含其他的未知参数。典型的多项式具有多项和的形式，这一点可以通过多项式的展开化简得到，多项式的化简方法在 4.2.3 节中介绍；在展开并化简后的多项式中，各项的系数都是确定的数字常量，变量的指数只能是非负整数，变量的指数代表了该项的次数。下面给出了一个单变量多项式的例子，各项的系数都是整数：

```
> p1:=2*x^3+11*x+15+5*x^2;
```

$$p1 := 2x^3 + 11x + 15 + 5x^2$$

注意到多项式 **p1** 不是按照各项次数的高低排列的。通常情况下，Maple V 按照输入时各项的顺序存储多项式，在输出时同样按照存储的顺序输出各项；如果用户输入的新多项式与工作空间中存储的某个多项式具有完全相同的项，则新输入的多项式直接与已有的多项式等价，因此其各项的顺序与已有的多项式顺序一致。可以使用命令 **sort(p)**对多项式 **p** 排序，输出的结果按照各项次数递减的顺序排列。注意命令 **sort** 作用的结果改变了原多项式中各项的顺序。下面用命令 **sort** 对多项式 **p1** 排序，排序后多项式 **p1** 各项的顺序发生了改变；同时新定义的多项式 **p2** 由于与多项式 **p1** 具有完全相同的项，因此各项的输出顺序与 **p1** 相同：

```
> sort(p1);
```

$$2x^3+5x^2+11x+15$$

```
> p1;
```

$$2x^3+5x^2+11x+15$$

```
> p2:=2*x^3+11*x+15+5*x^2;
```

$$p2:=2x^3+5x^2+11x+15$$

用户可以使用命令 **coeff(p, x, n)**或者 **coeff(p, x^n)**引用以 **x** 为变量的多项式 **p** 中 **x^n** 项的系数，其中 **x** 不能是数字或者两项的积；或者使用命令 **coeffs(p)**或者 **coeffs(p, x)**引用多项式 **p** 中各项的系数构成的序列。以多项式 **p1** 为例，看下面的结果：

```
> coeff(p1,x),coeff(p1,x,3);
```

11,2

```
> coeffs(p1);
```

15,11,2,5

在命令 **coeffs** 的输出中，各项系数的排列顺序与多项式中的顺序不同，用户无法规定这些系数的顺序。为了区分每项的系数，可以在命令 **coeffs** 中加入另一个回返参数，即使用 **coeffs(p, x, 't')**的命令格式，其中参数 **t** 用反引号界定，表示第三个参数应该是无值标识符；命令执行后，在输出各项系数的同时把多项式中的各项以序列的形式赋值给参数 **t**，序列 **t** 中各项的顺序与命令的输出中各项系数的顺序相同。看下面的结果：

```
> coeffs(p1,x,'t1');
```

15,11,2,5

```
> t1;
```

$$1, x, x^3, x^2$$

多项式的次数通常指的是多项式中各项的最高指数，可以使用命令 **degree(p, x)**计算多项式 **p** 中变量 **x** 的最高次数；对单变量多项式而言，可以缺省变量 **x**，命令的结果代表了多项式的次数。看下面的例子：

```
> degree(p1,x),degree(p1);
```

3,3

2. 多变量多项式

多变量多项式是指同时含有多个未知数变量的多项式。同样，各个变量的指数必须是非负整数；各项的次数是项中各个未知变量的指数总和。用户可以用命令 **indets(p)**引用多项式中的未知变量，命令的返回结果以集合的形式表示。下面给出了应用双变量多项式的例子，其中含有 **x** 和 **y** 两个变量：

```
> P2:=x^3+x^2*y^2-y^3+y^4;
```

$$p2 := x^3 + x^2 y^2 - y^3 + y^4$$

```
> indets(p2);
```

$$\{x, y\}$$

和单变量多项式类似，多变量多项式的输出通常也不是按各项次数的高低来排列的。同样，可以使用命令 **sort** 对多变量多项式排序。如果使用命令 **sort(p)**，则按照各项的次数高低和变量名的字典顺序重新排列多项式中的各项；如果在命令 **sort** 中给定了参数列表，则按照各项中给定变量的指数和并结合变量名的字典顺序进行排序；如果在命令中加入了第三个参数 **plex** 选项，则按照变量名的纯字典顺序进行排序。比较下面的结果：

```
> sort(p2,[x,y]);
```

$$x^2 y^2 + y^4 + x^3 - y^3$$

```
> sort(p2,[x,y],plex);
```

$$x^3 + x^2 y^2 + y^4 - y^3$$

命令 **coeff** 和 **coeffs** 也可以对多变量多项式作用，命令的格式基本是类似的；不同的是可以指定多项式的主变量，其余的变量及其指数作为主变量的系数；在指定多个主变量时，需要用列表的形式表示。看下面的例子：

```
> coeffs(p2,[x,y],'t2');
 t2;
```

$$1,1,-1,1$$

$$x^2 y^2, x^3, y^3, y^4$$

```
> coeffs(p2,x,'t2x');
 t2x;
```

$$y^4 - y^3, 1, y^2$$

$$1, x^3, x^2$$

多项式的次数可以用命令 **degree** 获得。命令的调用格式基本是一样的，用户需要用列表或者集合的形式指定所考察的多个变量，这两种指定变量序列的形式决定了命令的输出结果。如果以列表的形式给出所考察的变量序列，Maple V 采用递归的形式调用命令 **degree**；如果以集合的形式给出所考察的变量序列，命令的输出结果为各项中指定变量的指数和的最大值。下面给出了命令 **degree** 对多项式 **p2** 作用的情况：

```
> degree(p2,x),degree(p2,y),degree(p2);
```

$$3,4,4$$

```
> degree(p2,[x,y]);
  degree(p2,{x,y});
                              3
                              4
```

3. 多项式的类型

多项式的类型名为 **polynom**，正如前面所指出的，多项式中未知变量的指数必须是非负整数；否则不属于 **polynom** 数据类型。看下面的例子：

```
> p1,p2;
  type(p1,polynom),type(p2,polynom);
          2x^3 + 5x^2 + 11x + 15, x^2 y^2 + y^4 + x^3 - y^3
                          true, true
> type(x^(-1)+x^7,polynom);
                            false
```

展开并化简以后的多项式通常具有多项的和的形式，命令 **whattype**、**op** 及 **nops** 的作用规则和作用于其他的表达式是一样的，命令 **whattype** 不能返回 **polynom** 类型。看下面的例子：

```
> whattype(p2);
                              +
```

4.2.2 变量替换

多项式只是一种普通的表达式，它本身不具有函数的特征，即多项式不具有算子作用。因此，实现多项式中的变量替换，不能采用函数调用的方式，必须借助于变量替换的命令来完成。在 Maple V 中，实现变量替换的命令有三个：**subs**、**eval** 和 **subsop**；其中命令 **subsop** 的作用对象是表达式中的操作数，实现变量替换时有一定的局限性，其用法在第 3 章中已介绍过，这里不再重复。下面主要介绍命令 **subs** 和 **eval** 的使用，并比较二者的差别，虽然这里主要是以多项式为例，但这两个命令并不局限于对多项式作用，也可以实现其他表达式中的变量替换操作。

1. **subs**

命令 **subs** 的基本格式为 **subs(x=a, expr)**，其中 **expr** 代表任意的 Maple V 表达式，**x=a** 规定了变量替换的规则；**x** 通常为表达式 **expr** 中的变量，但也可以是其中的子表达式。此命令的功能是用表达式 **a** 替换表达式 **expr** 中出现的所有子表达式 **x**；如果 **a** 中含有未知的参数，所给的子表达式必须是命令 **op** 可以分离出的操作数，举例来说，命令 **subs** 不能实现类似于 **subs(x^2=a, x^3)**的转换，因为 **x^2** 不是表达式 **x^3** 的子表达式。

对多项式而言，命令 **subs(x=a, p)**可以把多项式的变量用数字 **a** 或者其他表达式 **a** 替

换，最简单的是对单变量多项式的求值运算。看下面的例子，使用类似于函数调用的方式不能实现多项式的求值运算，借助于命令 **subs** 则可以完成这种求值运算：

```
> p1;
```

$$2x^3+5x^2+11x+15$$

```
> p1(0);
```

$$2x(0)^3+5x(0)^2+11x(0)+15$$

```
> subs(x=0,p1);
```

$$15$$

下面的命令中，由于 **15** 和 **x^2** 都是多项式 **p1** 的子表达式，因此能够完成相应的替换操作：

```
> subs(15=0,p1);
  subs(x^2=0,p1);
```

$$2x^3+5x^2+11x$$

$$2x^3+15+11x$$

多变量多项式也可以执行类似的操作，命令的格式为 **subs(s1, ... , sn, expr)**。其中 **s1**、…、**sn** 可以是单个的等式，也可以是等式构成的列表或集合，代表了变量替换的准则；替换操作从第一个等式 **s1** 开始，同一列表或集合中的等式同时执行替换操作；如果参数中多次规定了对同一子表达式的替换准则，Maple V 采用其中的第一种规定。看下面的例子：

```
> p2;
  subs(x=1,y=-1,p2);
```

$$\frac{x^2y^2+y^4+x^3-y^3}{4}$$

```
> subs(y^3=0,{x^2=0},[y=1],p2);
```

$$1+x^3$$

2. **eval**

命令 **eval** 主要有两个用途，一是执行标识符的求值，二是执行变量替换。这里首先简单介绍命令 **eval** 的求值作用。

通常标识符赋值以后代表相应的表达式，但也有一些对象的名字代表标识符自身，例如函数名、数组名等；这些对象名是有值标识符，通常代表的是一种变换或者映射，例如函数名就具有算子的作用。命令 **eval(name)**是一个完全求值命令，返回值为标识符 **name** 所代表的映射或变换关系。在工作空间中，Maple V 对于递归定义的标识符采用完全求值规则，用户也可以使用命令 **eval(name, n)**返回变量名 **name** 的第 **n** 重定义，同样地，命令

eval(name)返回变量名 **name** 的最终定义。下面给出两个简单的例子，**f** 为函数名，**a** 为递归定义的标识符：

```
> f:=x->x^3-x^2;f;
```

$$f := x \to x^3 - x^2$$

$$f$$

```
> eval(f);
```

$$x \to x^3 - x^2$$

```
> a:=b:b:=c:
  a;
```

$$c$$

```
> eval(a,1);
```

$$b$$

命令 **eval** 的第二个作用是实现变量替换。命令的基本格式为 **eval(expr, x=a)**或 **subs(expr, eqns)**，其中 **expr** 代表任意的表达式，**x=a** 规定了变量替换的规则，**eqns** 代表由类似 **x=a** 的等式构成的集合。通常 **x** 是表达式 **expr** 中的变量，也可以是命令 **op** 分离出的 **expr** 的子表达式，这一点和命令 **subs** 的局限性相同。命令 **eval** 首先完成变量的替换操作，接着对替换后的表达式求值；集合 **eqns** 中规定的替换关系同时执行。下面给出了命令 **eval** 实现变量替换的例子：

```
> p1;
  eval(p1,x=0),eval(p1,15=0);
  eval(p1,{x^2=0,2=-2});
```

$$2x^3+5x^2+11x+15$$

$$15,2x^3+5x^2+11x$$

$$-2x^3+15+11x$$

3. **subs** 和 **eval** 的比较

命令 **subs** 和 **eval** 在执行变量替换时，在很多情况下可以等价。但命令 **subs** 在执行变量替换后通常不能进行完全求值，只能执行简单的运算或化简，而命令 **eval** 可以在变量替换后对替换后的表达式进行完全求值。命令 **eval** 能够完成微分、积分和分段函数等的运算，而命令 **subs** 则不能实现这些运算。下面的例子表明了两个命令的差别：

```
> subs(x=0,abs(x)+sin(x));
```

$$|0| + \sin(0)$$

```
> eval(abs(x)+sin(x),x=0);
```

$$0$$

注意，命令 **subs** 和 **eval** 实现变量替换时，不能发生在函数或者其他表达式的奇异点。

4.2.3 多项式的化简

前面介绍的多项式总是假设它们具有各项和的形式。在实际应用中，用户遇到的多项式可能具有其他形式，例如是多个多项式的乘积或者是多项式的乘方等，可以将这种形式的多项式展开为各项和的形式。在必要时，也可以把具有各项和形式的多项式转化为其他的形式。Maple V 提供的化简命令包括 **expand**（展开）、**factor**（因式分解）和 **collect**（合并同类项）等。这些命令不仅可以化简多项式，也可以用于其他表达式的化简。与命令 **sort** 不同的是，这些命令的执行结果生成新的表达式，而原来的表达式不发生改变。

下面主要以多项式为例介绍这些命令的使用。

1. 多项式的展开 **expand**

命令 **expand** 的主要应用是把乘积展开为各项和的形式。对多项式而言，这种展开通常都可以实现。命令的基本格式为 **expand(expr)**，其中 **expr** 代表任意的代数表达式。命令 **expand** 能够对包含三角函数、对数函数、指数函数、双曲函数、分段函数等运算的表达式进行展开。最常用的命令 **expand** 能够根据三角函数关系式对三角运算的表达式进行展开和化简。

下面给出了一个多项式展开的例子：

```
> expand((x +1)*(x +2));
```

$$x^2 + 3x + 2$$

```
> expand((x -1)^4);
```

$$x^4 - 4x^3 + 6x^2 - 4x + 1$$

另外，可以在命令 **expand** 中加入其他可选参数，规定在展开过程中保持不变的子表达式，命令的格式为 **expand(expr, expr1, expr2, ..., exprn)**。下面给出了一个应用的例子：

```
> expand((x+1)*(y+z)*(x^2-1),x+1);
```

$$(x+1)yx^2 - (x+1)y + (x+1)zx^2 - (x+1)z$$

2. 因式分解 **factor**

命令 **factor** 用来计算具有数字常量系数的多项式的因式，命令的格式为 **factor(expr, K)**。其中 **expr** 代表任意的代数表达式，通常具有多项式的形式；**K** 规定了因式分解的域，可以是用户规定的，也可以是 **real** 或者 **complex** 两个选项等；命令的输出结果为各个因式的乘积。在 **K** 缺省情况下，Maple V 根据所给多项式的系数确定因式分解的域；例如，如果所给多项式的系数都是整数，命令 **factor** 计算出的每个因式都是具有整数系数的不可再分因式，其中的因式可能不是线性的。命令 **factor** 不能对整数进行因数分解。看下面的例子：

```
> factor(6*x^2+18*x-24);
```

$$6(x+4)(x-1)$$

```
> factor(x^3+5);
```

$$x^3+5$$

```
> factor(x^3+5,complex);
```

$$(x+1.709975947)(x-.8549879733+1.480882610I)$$
$$(x-.8549879733-1.480882610I)$$

如果要对整数进行因数分解，可以使用命令 **ifactor**。关于命令 **ifactor** 的使用，读者可以参看相关的帮助。

3. 合并同类项 **collect**

命令 **collect** 能够将多项式中所有具有相同有理数幂的项合并在一起，命令的基本格式为 **collect(expr, x)**。其中 **expr** 代表任意的代数表达式，通常为多项式；**x** 可以是单一的变量，也可以是由未知变量构成的列表或集合。看下面的例子：

```
> f:= a^3*x-x+a^3+a;
```

$$f := a^3x-x+a^3+a$$

```
> collect(f,x);
```

$$(a^3-1)x+a^3+a$$

```
> p:= x*y+a*x*y+y*x^2-a*y*x^2+x+a*x;
```

$$p := xy+axy+yx^2-ayx^2+x+ax$$

```
> collect(p,[x,y]);
  collect(p,[y,x]);
```

$$(1-a)yx^2+((1+a)y+1+a)x$$
$$((1-a)x^2+(1+a)x)y+(1+a)x$$

命令 **collect** 还有另外的命令格式，读者可以用命令**?collect** 获得命令 **collect** 的帮助信息，这里不再介绍。

4.2.4　分式

多项式的加、减、乘和非负整数次乘方运算的结果仍然是多项式，但两个多项式的商有时不能得到多项式的计算结果。分式是用多项式的商来定义的，Maple V 把多项式和分式通称为有理多项式，简称有理式，其中多项式可以看作是分母为 1 的分式。本小节主要介绍有理式的类型分析、分式的化简和变量替换。

1. 有理式的类型分析

有理式的类型名 **ratpoly**，可以使用命令 **type** 确认有理式的数据类型。有理式包含两个操作数，分母和分子；可以使用命令 **op(1, rp)**或命令 **numer** 引用有理式 **rp** 的分子，使

用命令 **op(2, rp)**或 **denom(rp)**等引用有理式的分子和分母。看下面的例子：

```
> f:=(x^2+3*x+2)/(x^2+5*x+6);
```

$$f := \frac{x^2+3x+2}{x^2+5x+6}$$

```
> type(f,ratpoly),type(f,polynom);
```

$$true, false$$

```
> numer(f),denom(f);
```

$$x^2+3x+2, x^2+5x+6$$

2. 分式的化简

在上面定义的分式 **f** 中，分子和分母中包含相同的因式，但 Maple V 不能自动进行化简。这是与有理数运算的不同之处，Maple V 不能自动把有理分式化简为分子和分母没有公因式的状态；只有当 Maple V 能一下子分辨出分子与分母中的公因式时，才自动进行化简。看下面的例子：

```
> nn:=numer(f)*(x-1);
  dd:=denom(f)*(x-1)^2;
```

$$nn := (x^2+3x+2)(x-1)$$

$$dd := (x^2+5x+6)(x-1)^2$$

```
> g:=nn/dd;
```

$$g := \frac{x^2+3x+2}{(x-1)(x^2+5x+6)}$$

命令 **normal** 可以对分式进行化简，使分式的分子和分母具有整系数多项式的形式，而且分子和分母互质。基本的语法格式为 **normal(rat_poly)**，其中 **rat_poly** 代表任意的有理式；如果 **rat_poly** 为多项式，命令 **normal** 通常是不起作用的。下面是对分式 **f** 和 **g** 化简的结果，注意命令 **normal** 作用的结果生成新的有理式，原来的分式不发生改变：

```
> normal(f);
```

$$\frac{x+1}{x+3}$$

```
> f;
```

$$\frac{x^2+3x+2}{x^2+5x+6}$$

```
> normal(g);
```

$$\frac{x+1}{(x+3)(x-1)}$$

```
> normal(nn);
```

$$(x^2+3x+2)(x-1)$$

默认情况下，在命令 **normal** 的输出结果中，分子多项式和分母多项式保持多项式乘积的形式。可以在命令 **normal** 中加入可选项 **expanded**，命令 **normal(rat_poly, expanded)** 的输出结果中分子多项式和分母多项式分别展开为各项和的形式。看下面对分式 g 化简的结果，与前面的化简结果比较：

```
>normal(g,expanded);
```

$$\frac{x+1}{x^2+2x-3}$$

在 4.2.3 节介绍的几个命令中，命令 **expand** 也可以对分式作用，但只能对分式的分子进行展开，并把结果表示为多个分式和的形式；命令 **factor** 对分式作用时，首先把分式化简，然后对分式的分子和分母进行因式分解；命令 **collect** 对分式通常是不起作用的。看下面的结果：

```
>f:=(x+1)^2/((x^2+x)*x);
```

$$f := \frac{(x+1)^2}{(x^2+x)x}$$

```
>expand(f);
```

$$\frac{x}{x^2+x}+2\frac{x}{x^2+x}+\frac{1}{(x^2+x)x}$$

```
>factor(f);
```

$$\frac{x+1}{x^2}$$

```
>factor(g);
```

$$\frac{x+1}{(x+3)(x-1)}$$

```
>collect(g,x);
```

$$\frac{x^2+3x+2}{(x-1)(x^2+5x+6)}$$

3. 分式的变量替换

分式的变量替换方法和多项式的变量替换是类似的，可以通过命令 **subs** 或 **eval** 来完成。下面给出了对分式 g 进行变量替换的例子：

```
>subs(x=2,g);
```

$$\frac{3}{5}$$

```
>eval(g,{x^2=0,x=y});
```

$$\frac{2+3y}{(y-1)(6+5y)}$$

这两个变量替换的命令都不能在分式的奇点处进行变量替换。下面的命令不能被执行，虽然分式 g 化简后 x=-2 不再是奇点：

```
> subs(x=-2,g);
Error,division by zero
> eval(g,x=-2);
Error,division by zero
```

第5章 函 数

函数是基本的数学概念之一。Maple V 提供了 3000 多个系统函数，这些系统函数是 Maple V 实现符号分析、数值计算和图像处理的根本工具。同时，用户也可以定义自己需要的函数，以实现特定的运算或者其他功能。

本章将分别介绍 Maple V 的系统函数和用户函数的定义。

5.1 Maple V 系统函数

Maple V 的系统函数可以分为内部函数和外部函数两大类，内部函数是指 Maple V 启动后自动加载的函数，这些函数对于大多数 Maple V 用户来说都是必须的，包括常用的 Maple V 命令、数学函数等；外部函数是在使用之前需要手动加载的函数，这些函数或者命令是不常使用的。

本节将介绍基本的数学函数，并简单介绍 Maple V 的库函数。

5.1.1 基本数学函数

在科学计算中，绝对值函数、指数函数、对数函数、三角函数和双曲函数等基本数学函数是经常用到的。表 5.1 中列出了这些常用的数学函数和基本的调用格式。这些数学函数的定义和数学中的定义基本是一样的，与数学中的使用方法不同的是，调用这些数学函数时必须把参数放在小括号内。另外，这些数学函数名都是 Maple V 的保护名，用户不能对它们进行赋值操作。

表 5.1　　常用数学函数

数学函数分类	函数的基本调用格式	简单说明
绝对值函数	abs(x)	返回实数的绝对值或复数的模
平方根与开方函数	sqrt(x) surd(x,n)	主要用来计算实数的平方根或 n 次方根
指数函数	exp(x)	参数 x 可以是实数，也可以是复数
对数函数	log[b](x) ln(x)	b 为普通对数函数的底数， 参数 x 可以是实数，也可以是复数
三角函数与反三角函数	sin(x), arcsin(x) cos(x), arccos(x) tan(x), arctan(x) sec(x), arcsec(x) csc(x), arccsc(x) cot(x), arccot(x)	角度单位为弧度

续表

数学函数分类	函数的基本调用格式	简单说明
双曲函数与反双曲函数	sinh(x), arcsinh(x) cosh(x), arccosh(x) tanh(x), arctanh(x) sech(x), arcsech(x) csch(x),arcscs(x) coth(x),arccoth(x)	参数 x 可以是实数，也可以是复数
最大值与最小值函数	max(x1, x2, ..., xn) min(x1, x2, ..., xn)	返回若干实数的最大值或最小值

在 Maple V 中，函数的参数可以是数字常量，包括实数和复数，也可以是用未知参数表示的符号表达式，其中的未知参数默认为实数。Maple V 首先对参数进行化简，然后执行函数运算。如果化简后的参数是整数、分数等有理数或者符号表达式，函数调用的结果将用数学中的常用符号表示或者用进一步的化简结果表示，过程中没有任何近似；如果化简后的参数中含有浮点数，计算结果将用浮点数的近似值表示。用户可以使用命令 **evalf** 把用数学符号表示的结果转化为浮点数形式的近似值，也可以使用命令 **evalc** 把复数运算的结果写成实部和虚部相加的形式。

平方根函数和开方函数在第 4 章中已介绍过，下面分别介绍其他的几种函数。

1. 绝对值函数

函数 **abs** 可以计算实数的绝对值和复数的模。如果其中的参数是未知的参数或者其他的符号表达式，Maple V 用数学中的绝对值符号来表示计算的结果。看下面的例子：

```
> abs(-5.2);
```

$$5.2$$

```
> abs((3+2*I)/(4-3*I));
```

$$\frac{1}{5}\sqrt{13}$$

```
> a:='a':b:='b':
  abs(a+b*I);
```

$$|a+Ib|$$

```
> evalc(%);
```

$$\sqrt{a^2+b^2}$$

2. 指数函数和对数函数

一般情况下，指数函数 **exp(x)**的计算结果用 e^r 的形式表示，其中 **r** 是 **x** 的化简结果；如果 **r** 包含多个操作数，例如分数、复数或者符号表达式等，则放在小括号内；如果 **r=1**，则指数省略，如果 **r=0**，直接化简为 **1**；如果 **r** 具有浮点数的形式，Maple V 将给出浮点数形式的近似结果。看下面的例子：

```
> exp(0),exp(1),exp(1.);
```

$$1, e, 2.718281828$$

```
> exp(1/2),evalf(exp(1/2));
```

$$e^{\left(\frac{1}{2}\right)}, 1.64721271$$

```
> exp(1+I),exp(1.+I);
```

$$e^{(1+I)}, 1.468693940 + 2.287355287\,I$$

```
> evalc(exp(x+I*y));
```

$$e^{x}\cos(y) + I\,e^{x}\sin(y)$$

一般情况下，自然对数函数 **ln(x)**的计算结果用 **ln(r)**的形式表示，其中 **r** 是 **x** 的化简结果；如果 **r=1**，输出结果为 **0**。如果 **r** 具有浮点数的形式，结果用浮点数的近似值表示。而普通对数函数 **log[b](x)**的计算结果用 $\frac{\ln(x)}{\ln(b)}$ 的形式表示，并进行进一步的化简；其中 **b** 缺省时，**log(x)**与 **ln(x)**等价。看下面的例子：

```
> ln(1);
```

$$0$$

```
> ln(-1.);
```

$$3.141592654\,I$$

```
> ln(a+b);
```

$$\ln(a+b)$$

```
> ln(-2);
```

$$\ln(-2)$$

```
> log[2](3);
```

$$\frac{\ln(3)}{\ln(2)}$$

```
> log[b](x);
```

$$\frac{\ln(x)}{\ln(b)}$$

Maple V 能够根据指数函数和对数函数的性质，对指数函数和对数函数的混合运算进行化简。看下面的例子：

```
> ln(exp(1));
```

$$1$$

```
> exp(ln(pi*I+exp(1)));
```

$$I\,\pi + e$$

3. 三角函数和反三角函数

如果所给的角度是$\frac{\pi}{12}$弧度（15 度）的整数倍，Maple V 能够给出三角函数值的准确结果；如果所给的角度是用浮点数表示的，Maple V 将给出三角函数值的近似计算结果；在其他情况下，Maple V 仍然用三角函数调用的形式表示。看下面的例子：

```
> sin(Pi/12),cos(Pi/12),tan(Pi/12);
```

$$\frac{1}{4}\sqrt{6}\left(1-\frac{1}{3}\sqrt{3}\right),\frac{1}{4}\sqrt{6}\left(1+\frac{1}{3}\sqrt{3}\right),2-\sqrt{3}$$

```
> sin(Pi/18),sin(Pi/18),evalf(sin(Pi/18));
```

$$\sin\left(\frac{1}{18}\pi\right),\sin(.05555555556\,\pi),.1736481777$$

在调用函数时，参数必须直接跟在函数名后。数学中三角函数的幂的表示法在 Maple 中将不被识别。比较下面的例子：

```
> sin(Pi/6)^2;
```

$$\frac{1}{4}$$

```
> sin^2(Pi/6);
```

$$\sin^2$$

反三角函数的调用和三角函数的调用类似，Maple V 总是尽力给出准确的计算结果，这种结果可能是用反三角函数本身来表示的。如果反三角函数的参数超过了函数的实数定义域或者是复数，Maple V 将按照复数域中的定义计算反三角函数的值，可能得到复数的计算结果。看下面的例子：

```
> arcsin(1/3);
```

$$\arcsin\left(\frac{1}{3}\right)$$

```
> evalf(%);
```

$$.3398369094$$

```
> arctan(-1);
```

$$-\frac{1}{4}\pi$$

```
> arccos(1.2),arcsin(I);
```

$$.6223625037\,I, I\,\mathrm{arcsinh}(1)$$

双曲函数和反双曲函数的使用方法与三角函数和反三角函数的使用基本类似，这里不再举例介绍。

4. 最大值与最小值函数

最大值函数和最小值函数的使用和上面介绍的函数类似，两个函数都可以带多个参数，通常这些参数应该是实数或者实数表达式。如果 Maple V 不能确定所给参数中的最大值或者最小值，输出结果用函数调用的形式表示，结果中的参数已进行了化简。看下面的例子：

```
>max(3/2,1.49);
```

$$\frac{3}{2}$$

```
>min(3/2,1.49,f(x));
```

$$\min(x^2,1.49)$$

```
>max(I,1);
Error,(in simpl/max)constants must be real
```

5.1.2 Maple V 的函数库

Maple V 的系统函数库可以分为标准库、扩展库、库函数包和共享库四个部分，其中标准库中的函数在 Maple V 启动后可以自动加载，用户可以直接使用；扩展库、库函数包和共享库中的函数在使用之前需要手动加载。下面分别介绍 Maple V 的系统函数库。

1. 标准库

即使是标准库中的函数，在 Maple V 启动后也不是全部加载到系统工作空间中，只有那些系统运行确实需要的函数或命令才被加载到系统的主内存空间中。如果用户输入了一个标准库中目前尚未使用过的函数，系统能自动把函数对应的程序从扩展内存空间加载到系统主内存空间，并执行相应的运算。因此，随着主内存空间中函数的增多，系统的运算速度和性能会逐渐下降。

2. 扩展库

扩展库中包含了不经常使用的函数和命令。在使用这些函数或者命令之前，必须使用命令 **readlib** 从扩展内存空间中加载相应的程序。例如，后面介绍的分析函数不连续点的命令 **discont**：如果不用命令 **readlib(discont)**手动加载，命令 **discont(1/sin(x), x)**的结果仍然保持输入的格式；在加载以后，命令输出了函数 **1/sin(x)**的不连续点构成的集合。看下面的结果：

```
> discont(1/sin(x),x);
```

$$\text{discont}\left(\frac{1}{\sin(x)}, x\right)$$

```
> readlib(discont):
  discont(1/sin(x),x);
```

$$\{\pi - ZI \sim\}$$

3. 库函数包

库函数包是为特定的目的而设计的多个命令或函数的集合。Maple V 提供了 37 个库函数包，每个库函数包的作用都不相同。例如，**orthopoly** 库函数包中主要包含了用于产生正交多项式的函数，**student** 库函数包主要用于学生课堂中的计算，**plots** 库函数包是一个图形工具包，**linalg** 库函数包主要用于矩阵、向量等的操作，**inttrans** 库函数包主要包含了积分变换的函数或命令等。在使用库函数包中的函数或命令之前，必须手动加载这些函数和命令。

以 **orthopoly** 库函数包中的用于生成 Chebyshev 多项式的命令 **T** 为例，用户可以使用下面的两种调用方式：

- 使用 **orthopoly[T](n, x)**的命令格式。
- 首先使用命令 **with(orthopoly)**或者 **with(orthopoly, T)**加载整个 **orthopoly** 库函数包或者只加载其中的命令 **T**，然后直接使用 **T(n, x)**的命令格式。

下面给出了两种命令格式的例子：

```
> orthopoly[T](3,X);
```

$$4x^3 - 3x$$

```
> with(orthopoly);
```

$$[G, H, L, P, T, U]$$

```
> T(3,x);
```

$$4x^3 - 3x$$

```
> with(orthopoly,T);
```

$$[T]$$

使用命令 **with(orthopoly)**后，Maple V 给出了 **orthopoly** 库函数包中的所有函数列表；如果不希望输出函数列表，用户也可以用冒号作为命令的结束标志。如果使用命令 **with(orthopoly, T)**的格式，输出的列表中将只包含所加载的函数 **T**。库函数包或者其中的库函数加载以后，始终存在于 Maple V 的主内存空间中，再次使用时不需要重新加载；当然，如果中途使用了命令 **restart**，或者重新启动了 Maple V，那么以前加载的库函数需要重新加载才能再次使用。

用户可以使用命令 **alias** 或者 **macro** 把 **orthopoly[T]**表示成简单的形式。看下面的例

子：

```
>restart:
alias(T=orthopoly[T]):
T(5,x);
```

$$16x^5 - 20x^3 + 5x$$

```
>alias(T=T):
>macro(T=orthopoly[T]);
T(5,x);
```

$$16x^5 - 20x^3 + 5x$$

上面的命令 **alias(T=T)**用来消除别名 **T** 的定义，同样也可以使用命令 **macro(T=T)**消除缩写 **T** 的定义。

```
>macro(T=T):
T(5,x);
```

$$T(5,x)$$

用户还可以编写自己的函数包，或者向已有的库函数包中添加新的函数或命令，这是由 Maple V 的开放性决定的。关于函数包的编写，本书不再介绍，读者可以参看相关的帮助或其他的 Maple V 手册。

4. 共享库

共享库是由 Maple V 用户或者团体编写并捐献出来的命令、库函数包或者工作单的集合。关于共享库，读者可以用命令**?share** 来获得相关的帮助，这里不再介绍。

5.2 自定义函数

除了 Maple V 提供的系统函数外，用户还可以自定义所需的函数。函数的完整定义是用过程控制语句来完成的。Maple V 提供了一种特殊的函数定义运算符，可以简化常见函数的定义。函数实际上代表了自变量与因变量之间的变换关系，可以看作是一种算子。分段函数是一类重要的函数，Maple V 中提供了专门的命令来定义分段函数。复合函数运算也是经常用到的，可以用复合函数运算符完成。

本小节首先简单介绍 Maple V 的过程控制语句，接着介绍函数定义运算符的使用和函数算子的概念，最后介绍分段函数的定义和复合函数的运算。

5.2.1 过程控制语句简介

过程控制语句的基本关键字为：**proc** 和 **end**，分别标志着过程定义的开始和结束。下面给出过程的完整语法格式：

```
>proc(argseq)
 local lnseq;
 global gnseq;
 options onseq;
 description stringseq;
 statseq
 end;
```

proc(*argseq*)
local *lnseq*;
global gnseq;
option *onseq*;
description *stringseq*;
  statseq
end

其中 **statseq** 是过程的主体部分，由按特定顺序排列的命令构成，规定了过程的功能；**statseq** 中的命令要以分号或冒号结尾，最后一条命令结尾的分号或者冒号可以省略；当 **statseq** 为空时过程不执行任何命令。其余几部分都是可选的，下面简单介绍这几部分的作用：

- **argseq** 代表过程的形式参数序列，在 **argseq** 中可以声明各个形式参数的数据类型；
- **local** 子句声明过程中用到的局部变量，**lnseq** 代表这些局部变量构成的序列；
- **global** 子句声明过程中用到的全局变量，**gnseq** 代表相应的全局变量序列；
- **options** 子句声明过程的特征选项序列 **onseq**，这些选项规定了过程的特殊性质；
- **description** 子句给出过程的描述信息，**stringseq** 代表这些描述信息的字符串序列。

过程定义也可以作为表达式使用，用户可以对过程命名，然后像使用 Maple V 的系统函数一样调用过程来完成特定的功能。

在过程的定义中，可以使用流程控制语句实现复杂的功能。Maple V 提供的流程控制语句包括条件语句和循环语句。条件语句的基本格式为：

if *condexpr* **then** *statseq1* **else** *statseq2* **fi**

其中 **if**、**then**、**else** 和 **fi** 是控制关键字，**condexpr** 是条件表达式，**statseq1** 是条件表达式 **condexpr** 为真时执行的语句序列，**statseq2** 是条件表达式非真时执行的语句序列。

条件语句可以嵌套使用，并且可以加入 **elif** 子句实现多重选择。循环语句包括 **for** 循环语句和 **while** 循环语句两种。**for** 循环语句的基本格式为：

for *indexname* **from** *expr1* **by** *expr2* **to** *expr3* **do** *statseq* **od**

其中 **for**、**from**、**by**、**to**、**do** 和 **od** 是控制关键字，**indexname** 是计数变量，**expr1**、**expr2** 和 **expr3** 分别代表计数变量的初值、增量和终值，**statseq** 代表循环的执行语句序列即循环体。**while** 循环语句的基本格式为：

while *condexpr* **do** *statseq* **od**

其中 **while**、**do** 和 **od** 是控制关键字，**condexpr** 是控制循环的条件表达式，**statseq** 是循环体。**for** 循环语句和 **while** 循环语句还有其他变体，这里不再列出。

这些流程控制语句也可以在过程外部使用，实现条件选择或者循环执行功能。过程的定义和流程控制语句使得用户可以编写出具有复杂功能的程序，关于过程及流程控制语句的详细知识，这里不再具体介绍，读者可以参看相关的帮助或者其他的 Maple V 手册。下面给出了一个过程的例子：

```
>myabs:='myabs':
 myabs:=proc(x)
    if type(x,numeric) then
        if x<0 then -x else x fi;
    else
       'myabs'(x);
    fi;
  end:
```

这个过程不仅可以计算实数常量的绝对值，而且允许所给的实际参数是符号表达式或者其他非数字量。下面是该过程调用的例子：

```
>myabs(7),myabs(-4.3);
                          7,4.3
>myabs(a);
                        myabs(a)
```

5.2.2 用函数定义运算符定义函数

在过程定义的完整格式中，如果主体部分 **statseq** 是单个的表达式或者是一条 **if** 语句，可以使用函数定义运算符简化过程的定义。使用函数定义运算符定义函数（或过程，下同）的基本语法格式为：

f := (argseq)-> statseq;

其中 **f** 代表函数名，**argseq** 代表函数的自变量序列，**statseq** 代表函数因变量与自变量之间的关系表达式，通常为单个算术表达式或者是 **if** 语句。这与下面的过程定义是等价的：

f := proc(argseq) statseq end;

下面分别介绍函数定义的基本知识、函数参数的数据类型、函数名与其他表达式名的差别。

1. 函数定义的基本知识

如果定义的是一元函数，可以省略自变量两边的小括号。下面是一个一元函数的例子：

```
> f:=x->3*x+5;
```

$$f := x \to 3x + 5$$

函数的调用格式和系统函数的调用格式相同，下面是调用函数 f(x)的例子：

```
> f(2),f(-1.);
```

$$11.2$$

如果所给的实际参数是无值标识符或者包含无值标识符的符号表达式，Maple V 通常也能够完成相应的变量替换并进行简单化简。看下面的例子：

```
> f(a+b);f(2*x^2-3);
```

$$3a + 3b + 5$$

$$6x^2 - 4$$

如果在调用函数时给出的参数多于函数的自变量个数，Maple V 将忽略多余的参数；如果所给的参数少于函数的自变量个数，Maple V 将给出错误信息。看下面的例子：

```
> f(a,b-a);
```

$$3a + 5$$

```
> f();
Error,(in f)f uses a lst argument,x,which is missing
```

如果定义的是二元函数或者多元函数，自变量序列必须放在小括号内，否则不能实现预期的功能。下面给出一个二元函数的例子：

```
> g:=(x,y)->x^3-3*x*y^2;
```

$$g := (x, y) \to x^3 - 3xy^2$$

```
> g(1,0),g(2,0.5);
```

$$1,6.50$$

下面的命令定义了一个变量 x 和以 y 为自变量的一元函数的序列，不是一个二元函数：

```
> g:=x,y->x^3-3*x*y^2;
```

$$g := x, y \to x^3 - 3xy^2$$

```
> g(1,0);
```

$$x(1,0), x^3 - 3x$$

用户也可以定义常数函数或者符号常量函数，定义不给出变量序列，但小括号不能省略。下面分别定义了常数函数 f 和符号常量函数 g，并给出了函数调用的情况：

```
> f:=()->0;
```

$$f := 0$$

```
> f(),f(2),f(3,4),f(x);
```

$$0,0,0,0$$

```
> g:=()->x^2*a;
```

$$g := ( ) \to x^2 a$$

```
> g(),g(y),g(2),g(b,c);
```

$$x^2a, x^2a, x^2a, x^2a$$

在 Maple V 中，因变量和自变量之间的关系并不局限于数学中的函数关系，用户也可以定义返回值为列表、集合、向量等对象的函数。看下面的例子：

```
> g:=t->[1-t^2,t^3-1];
```

$$g := t \to [1 - t^2, t^3 - 1]$$

```
> g(2);
```

$$[-3,7]$$

如果希望函数的返回值为表达式序列，定义函数时的函数体 **statseq** 需要用小括号界定，否则不能得到预期的效果。看下面的例子：

```
> f:=t->(2*t,t^3,a);
```

$$f := t \to (2t, t^3, a)$$

```
> f(3);
```

$$6,27,a$$

函数可以嵌套调用，在用户定义的函数中可以调用 Maple V 的系统函数，也可以调用用户自己定义的函数。在下面的例子中，函数 **h(x, y)**的定义中调用了函数 **sin(x)**、**cos(x)** 和 **exp(x)**：

```
> h:=(x,y)->sin(x)*cos(y)+exp(x+y*I);
```

$$h := (x, y) \to \sin(x)\cos(y) + e^{(x+Iy)}$$

```
> h(0,Pi);
```

$$-1$$

```
> x->1-1;
```

$$0$$

```
> (x,y)->h(x,y);
```

$$h$$

Maple V 能够自动对用户定义的函数进行化简，优化函数的定义。看下面的例子：

```
> f:=x->x^2-x*x*2+x*0-2*4+9;
```

$$f := x \to 1 - x^2$$

2. 函数参数的数据类型

在前面的函数定义中，形式参数的数据类型没有声明，因此调用时的实际参数原则上可以是任何数据类型（**anything**）；如果实际参数替换函数的自变量后，其数据类型与函数体 **statseq** 中的运算不匹配，Maple V 将给出错误信息。下面定义了一个函数 **f(x)**，其函数体表达式 **1-x^2** 要求实际参数的类型为数字变量，因此等式 **a=b** 不能作为函数的参数：

```
> f:=x->1-x^2;
```

$$f := x \to 1 - x^2$$

```
> f(a=b);
Error,(in simpl/reloprod) invalid terms in product
```

用户也可以在自变量序列 **argseq** 中同时声明各个自变量的数据类型。类型声明的语法格式为：

parameter::type

其中 **parameter** 代表自变量标识符，**type** 代表自变量的数据类型。各个自变量的类型声明之间仍然用逗号隔开。未声明的自变量默认数据类型为 **anything**，即实际参数可以是任何数据类型。用命令**?type** 可以获得 Maple V 数据类型的详细列表。

调用函数时，Maple V 首先从左到右对每个实际参数进行化简，并同时检查化简后实际参数的数据类型。如果其中某一实际参数的数据类型与声明的数据类型不符，Maple V 立即给出错误信息，函数体不再执行；当没有类型错误时，函数体才会执行。

在下面定义的函数 **evenf** 中，声明自变量的数据类型为偶数（**even**）；因此当给出的实际参数为 **3** 时，Maple V 给出了错误信息：

```
> evenf:=(n::even)->(n/2)!/n!;
```

$$evenf := n :: even \to \frac{\left(\frac{1}{2}n\right)!}{n!}$$

```
> evenf(4);
```

$$\frac{1}{12}$$

```
> evenf(3);
Error,evenf expects its 1st argument,n,to be of type
even,but received 3
```

5.2.3 函数的算子概念

算子规定了一种变换关系，常见的如 Laplace 算子等。本小节简单介绍 Maple V 中函数的算子概念、匿名函数算子和函数名与其他表达式名之间的差别。

1. 函数名与函数算子

在数学中，函数是用映射概念定义的。映射是一种变换关系，对函数而言，它代表从自变量到因变量的变换准则。在 Maple V 中，本质上函数名代表的是这种变换关系，可以称之为函数算子；调用函数实际上是把算子作用于实际参数，返回的结果是对实际参数实施变换准则后的结果。如果函数名未带实际参数，将返回函数名本身。看下面的例子：

```
> f:=x->x^2-x+1;
  f;
```

$$f := x \to x^2 - x + 1$$

$$f$$

Maple V 中的系统函数具有同样的性质。在函数定义的过程中，如果定义的变换关系与当前工作空间中已有的函数算子相同，Maple V 能自动把定义的函数算子赋值为已有的相同函数算子，即认为这两个函数算子是等价的。看下面的例子：

```
> g:=t->f(t);
```

$$g := f$$

```
> h:=x->sqrt(x);
```

$$h := \mathrm{sqrt}$$

上面两个命令的作用与直接把函数算子 f、sqrt 赋值给标识符 g、h 是等价的。赋值以后，标识符 g、h 成为新的函数名，使用这两个新的函数算子标识可以实现所代表的变换关系，下面是调用函数 g(x)和 h(x)的例子：

```
> g(Pi/4);
  h(256);
```

$$\sqrt{2}$$

$$16$$

实际上，可以用已有的函数算子通过赋值直接构造新的函数算子。简单的构造方法包括函数算子的加减乘除以及乘方、开方等，同时，用户可以直接对函数算子计算微分、积

分或者实现函数算子之间的复合运算等。在第 9 章中，可以看到微分算子 **D** 直接对函数算子计算微分，返回新的函数算子。复合运算将在 5.2.5 节中介绍，这里给出几个用已有函数算子直接构造新的函数算子的例子，这些新的函数算子可以实现所规定的变换关系：

```
> f:=sin+cos;
  f(Pi/4);
```

$$f := \sin + \cos$$

$$\sqrt{2}$$

```
> g:=D(f);
  g(Pi/4);
```

$$g := \cos - \sin$$

$$0$$

2. 匿名函数算子

未命名的函数算子称为匿名函数算子。在 Maple V 中，匿名函数算子可以用函数定义运算符表示，也可以用过程控制语句表示。这里主要讨论用函数定义运算符定义的匿名函数算子。下面定义的就是一个匿名函数算子：

```
> x->x^2;
```

$$x \to x^2$$

匿名函数算子可以对参数作用，也可以和其他的函数算子结合甚至计算微分等。看下面的例子：

```
> D(%);
```

$$x \to 2x$$

```
> (x->x^2)(y);
```

$$y^2$$

实际上，匿名函数算子可以出现在其他函数算子可以出现的任意位置。匿名函数算子代表了一种临时的变换关系，在经常使用的时候，对匿名函数算子命名是明智的选择，因为匿名函数算子书写起来往往比较麻烦。

3. 函数名与其他表达式名的区别

函数名和其他表达式名是有差别的。函数名代表着自变量到因变量的变换关系，而其他表达式名往往代表了实际的表达式。比较下面的例子：

```
> e:=x^2-x+1;
  e;
```

$$e := x^2 - x + 1$$

$$x^2 - x + 1$$

```
> f:=x->x^2-x+1;
  f;
```

$$f := x \to x^2 - x + 1$$

$$f$$

函数的调用可以采用函数名后跟参数的形式，参数放在小括号内；函数算子作用于实际参数，并按照所定义的变换准则完成实际参数与自变量的替换。但代表其他表达式的标识符是不能采用这种方法来实现变量替换的。比较下面的结果，Maple V 把表达式 e 中的 x 看作未知的函数算子，因而不能实现预期的变量替换：

```
> e(1);
  f(1);
```

$$x(1)^2 - x(1) + 1$$

$$1$$

在第 4 章中读者已知道，可以使用命令 **subs** 完成多项式中的变量替换。同时，命令 **subs** 可以实现其他许多表达式中的变量替换。看下面的例子：

```
> subs(x=1,e);
```

$$1$$

```
> l:=[t,t^2,t^3];
```

$$l := [t, t^2, t^3]$$

```
> subs(t=2,l);
```

$$[2,4,8]$$

命令 **unapply** 可以实现表达式到函数算子的转换。命令的基本格式为 **unapply(expr, x, y, ...)**，其中 **expr** 代表其他的非函数算子表达式，**x**、**y** 等代表转换后函数的自变量，命令的输出结果规定了一个匿名函数算子，用户可以把输出结果赋值给某一标识符。看下面的例子：

```
> q:=x^2+y^3+1:
  f:=unapply(q,x);
  f(2);
```

$$f := x \to x^2 + y^3 + 1$$

$$5 + y^3$$

```
> unapply(f(x),x);
```

$$x \to x^2 + y^3 + 1$$

```
> g:=unapply(q,x,y);
  g(2,1);
```

$$g := (x, y) \to x^2 + y^3 + 1$$

$$6$$

显然，命令 **unapply** 也可以作为定义函数的一种方法。

5.2.4 分段函数

用函数定义运算符结合条件语句可以定义分段函数。下面给出一个分段函数的定义：

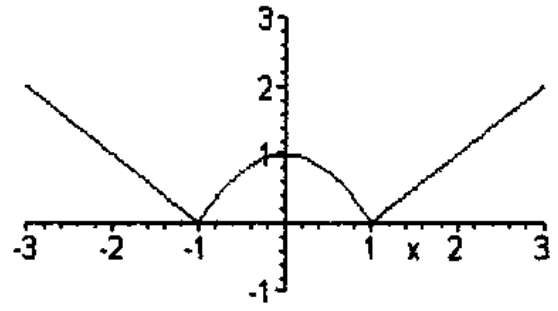
```
> pf:=x->if x>1 then x-1 elif x>-1 then 1-x^2
  else -1-x fi;
pf := proc(x)
option operator, arrow;
    if 1 < x then x - 1 elif -1 < x then 1 - x^2 else -1 - x fi
end
```

Maple V 将函数定义转化为过程的形式输出。由于在函数定义中包含了条件表达式，所以上面定义的分段函数不能够处理实际参数为符号表达式的情形。下面给出了函数 **pf** 调用的情况：

```
> pf(-2),pf(0),pf(2);
                              1,1,1
> pf(a);
Error,(in pf)cannot evaluate boolean
```

下面用命令 **plot** 绘出这个分段函数的图像，关于命令 **plot** 的使用格式，读者可以参看第 13 章或者相关的帮助信息。由于函数 **pf** 不能对未知参数作用，因此在命令 **plot** 中，函数 **pf(x)**需要用单引号界定：

```
>plot('pf(x)',x=-3..3,view=[-3..3,-1..3]);
```



显然，这种定义分段函数的方法比较麻烦。Maple V 提供了命令 **piecewise**，可以定义各种复杂的分段函数。命令的格式为：

piecewise(cond_1,f_1,cond_2,f_2,...,cond_n,f_n,f_otherwise);

其中：**cond_i**（**i=1**，**2**，…，**n**）代表分段的条件，可以是单个的等式或不等式，也可以是包含几个不等式的条件表达式；**f_i** 是在第 **i** 种条件下函数的定义式，**f_otherwise** 代表在前 **i** 种条件都不成立时函数的定义式；在缺省情况下，**f_otherwise=0**。在 **cond_i** 中可

以使用多项式、绝对值函数或者分段函数等，例如“**x^2-4>0 and x>0**”和 “**abs(x)<4**”都可以作为分段的条件。用命令 **piecewise** 定义分段函数时，输出结果仍用命令 **piecewise** 表示。用命令 **piecewise** 定义的函数可以直接处理参数为符号表达式时的情形，对函数调用的结果可以使用命令 **simplify** 化简。

下面用命令 **piecewise** 定义了两个分段函数，其中 **g(x)**的结果不是最简的，可使用命令 **simplify** 对其进行化简，同时给出函数调用时的结果：

```
> f:='f':f:=x->piecewise(x>=0,x^2-1,x<0,-x+1);
```

$$f := x \to \text{piecewise}(0 \le x, x^2 - 1, x < 0, -x + 1)$$

```
> f(2);
```

$$3$$

```
> f(a);
```

$$\begin{cases} a^2 - 1 & 0 \le a \\ -a + 1 & a < 0 \end{cases}$$

```
> g:=x->piecewise(x*x>4 and x<8,f(x))
```

$$g := x \to \text{piecewise}(4 < x^2 \text{ and } x < 8, f(x))$$

```
> g(a);
```

$$\begin{cases} \begin{cases} a^2 - 1 & 0 \le a \\ -a + 1 & a < 0 \end{cases} & 4 - a^2 < 0 \text{ and } a - 8 < 0 \\ 0 & \textit{otherwise} \end{cases}$$

```
> simplify(g(a));
```

$$\begin{cases} -a + 1 & a < -2 \\ 0 & a \le 2 \\ a^2 - 1 & a < 8 \\ 0 & 8 \le a \end{cases}$$

由于不能确定未知参数所处的分段区间，Maple V 仍然用分段的形式表示符号表达式的计算结果。另外，可以使用命令 **assume** 对参数或符号所处的区间或其他性质做出假定。在下面的例子中假定了 **a>0**，于是可以得到 **f(a)=a2-1** 的结果。其中，参数 **a** 后面的“~”表明对变量 a 做了假定：

```
> assume(a>0):f(a);
```

$$a\sim^2 - 1$$

5.2.5 复合函数

复合函数的特征是一个函数的输出是另一个函数的输入，在数学上复合函数的表示为(f∘g)(x)=f(g(x))。在 Maple 中，只要输入 f(g(x))就可以实现函数 f(x)与 g(x)的复合函数运算。看下面的例子，f(x)、g(x)和 h(x)是新定义的函数：

```
> f:=x->x^2+x:
  g:=x->x^3-1:
  h:=x->sin(x)+cos(x):
```

```
> f(g(x));
```

$$(x^3-1)^2+x^3-1$$

```
> expand(%);
```

$$x^6-x^3$$

但这种表示复合函数的方法还是有很大局限性的，例如有数个不同的函数复合在一起或者函数自身复合 n 次时，用这种方法表示是一件很麻烦的事。为方便复合函数的运算，Maple 中提供了@和@@两个复合函数运算符。这两个复合函数运算符的基本格式分别为：

- (f1@f2@f3@...@fn)(x)：用来计算复合函数(f1∘f2∘…∘fn)(x)=f1 (f2…(fn (x)))。
- (f@@n)(x)：用来计算函数 f(x)的 n 次自身复合函数(f∘f∘…∘f)(x)=f(f…f(x))。

显然，(f@g)(x)与 f(g(x))是等价的。看下面的计算结果：

```
>(f@g)(x);
```

$$(x^3-1)^2+x^3-1$$

```
>(f@h)(x);
```

$$(\sin(x)+\cos(x))^2+\sin(x)+\cos(x)$$

```
simplify(%);
```

$$2\sin(x)\cos(x)+\sin(x)+\cos(x)+1$$

```
expand(%%);
```

$$\sin(x)^2+2\sin(x)\cos(x)+\cos(x)^2+\sin(x)+\cos(x)$$

当使用命令 **simplify** 时，Maple 利用三角恒等式对结果进行化简，但对乘积项并未进一步化简；使用命令 **expand** 时，Maple 把结果写为各项和的形式，但没有利用三角恒等式对结果进行整理和化简。为了得到最简的结果，需要使用命令 **combine**。

```
> combine((f@h)(x));
```

$$1+\sin(2x)+\sin(x)+\cos(x)$$

在(f@@n)(x)中，通常 n 为整数。特别地，如果 n=0，(f@@0)(x)返回 x；如果 n=1，(f@@1)(x)返回 f(x)本身；如果 n= -1，(f@@(-1))(x)返回 f(x)的反函数；如果 n 为负整数，记函数 f(x)的反函数为 g(x)，则(f@@n)(x)=(g@@(-n))(x)。看下面的例子：

```
> (sin@@6)(x);
```

$$(\sin^{(6)})(x)$$

```
> expand(%);
```

$$\sin(\sin(\sin(\sin(\sin(\sin(x))))))$$

```
> (sin@@6)(Pi/6);
```

$$(\sin^{(5)})\left(\frac{1}{2}\right)$$

```
> (sin@@1)(x);
```

$$\sin(x)$$

```
> (cos@@(-1))(x);
```

$$\arccos(x)$$

```
> sin@@0;
```

$$() \rightarrow \text{args}$$

其中 sin@@0 的输出结果定义了一个匿名函数() ->args，表示函数的返回值与输入的参数完全相同。

第 6 章 序列和级数

数学中把由数字或算术表达式构成的序列称为数列。Maple V 把数列的概念推广，序列中的元素可以是任意合法的表达式，包括数字、算术表达式、标识符、字符串、方程、不等式和集合等对象，这种由表达式构成的序列称为表达式序列，并简称为序列。序列是 Maple V 中经常用到的一种数据结构。

如果序列中的元素由数字或算术表达式构成，与数学意义上的数列等价，用户可以对数列中的元素求和。数学上把数列的求和式子称为级数，按求和项的数量可以分为无穷级数和有穷级数。在理论分析和科学计算中，级数展开是一种常用的工具，广泛地应用在近似计算和误差分析中。Maple V 定义了自己的级数数据类型，并提供了级数运算和级数展开的许多命令。

本章首先介绍 Maple V 中序列的基本知识和有关运算，接着介绍数列和数列的求和，最后介绍 Maple V 中的级数数据类型、级数展开和级数的运算。

6.1 序列

用户可以用逗号把任意合法的表达式连接起来构成序列，同时 Maple V 提供了用于生成序列的命令。本节介绍序列的基本知识和生成序列的命令。

6.1.1 序列的基本知识

序列是由逗号分隔的表达式的有序排列，其元素可以是除了表达式序列外任意的合法表达式，同一序列的元素其数据类型也可以各不相同。在前面的章节中，很多地方都用到了序列，例如多参数命令的参数序列、多元函数的自变量序列等。把多个有返回值的命令或表达式之间用逗号隔开，作为一条命令输入，输出的结果就是由各个命令或表达式的返回值构成的序列，在前面的例子中很多例子都采用了这种方法。和其他对象一样，用户可以对序列命名。下面给出几个序列的例子：

```
> 1,2,3,4,5;
```

$$1,2,3,4,5$$

```
> seq1:=a<b,a<=b,a=b,a>=b,a>b;
```

$$seq1 := a < b, a \le b, a = b, b \le a, b < a$$

```
> seq2:=2,3/4,-0.3,symbol,"string",[list],{set};
```

$$seq2 := 2, \frac{3}{4}, -3, symbol, "string", [list], \{set\}$$

在构成序列时，总是先对序列中的各个表达式进行运算，由运算的结果构成最终的序列。序列中的元素可以重复出现。不包含任何元素的序列为空序列，用 **NULL** 来表示。注意标识符 **NULL** 是 Maple V 的保护名，不能被赋值。

```
> numseq:=1,1,-1,1,1,-1;
```

$$numseq := 1,1,-1,1,1,-1$$

```
> NULL;
```

```
> nullseq:= NULL;
```

$$nullseq :=$$

```
> NULL:=1;
Error,attempting to assign to 'NULL' which is protected
```

序列中的元素是用逗号隔开的，所以如果把两个序列用逗号隔开将形成一个新的序列。当向序列中添加新的元素时，只需在当前的序列后面用逗号与添加的元素连接即可，这种方法在改变系统常量序列 **constants** 时已经用到。看下面的例子：

```
> nullseq:= NULL,NULL,NULL;
```

$$nullseq :=$$

```
> seq1:=seq1,a<>b;
```

$$seq1 := a < b, a \le b, a = b, b \le a, b < a, a \ne b$$

用户甚至可以把某一个命令的所有参数用一个序列来表示，然后把序列名作为直接参数调用。在下面的例子中即采用了这种方法，把命令 **has** 的两个参数定义为序列 **arg**：

```
> arg:=c[1]*sin(x)+c[2]*cos(x),x;
```

$$arg := c_1 sin(x) + c_2 cos(x), x$$

```
> has(arg);
```

$$true$$

序列的数据类型为表达式序列，类型名为 **exprseq**，用户可以用命令 **whattype** 查看序列的数据类型。但其他命令如 **op**、**nops** 和 **type** 等，其参数的个数是严格限制的，所以不能作用于含有多个元素的序列，这就决定了命令 **type** 不能识别 **exprseq** 数据类型。如果要计算序列 **sequence** 中元素的个数，用户可以使用命令 **nops([sequence])**，其中**[sequence]** 代表由序列 **sequence** 中的元素构成的列表。关于列表的知识，读者可以参阅 7.1 节。看下面的结果：

```
> whattype(seq1),whattype(seq2);
  type(seq1,exprseq);
```

$$exprseq, exprseq$$

```
Error,wrong number (or type) of parameters in function type
```

```
> whattype(NULL);
```

$$exprseq$$

```
>nops([seq1]),nops([NULL]);
```

$$6.0$$

和字符串一样，序列的元素也是用序列名后跟方括号来引用的。在方括号中给出所引用元素的序号或者由范围表达式所确定的序号范围，即可以引用序列中对应的元素。看下面的例子：

```
>seq1[3],seq1[6];
 seq1[1..2];
 seq1[-3..-2];
```

$$a=b,a\neq b$$
$$a<b,a\leq b$$
$$b\leq a,b<a$$

6.1.2 序列的生成命令

除了可以用逗号直接生成序列外，还可以用命令 **seq** 和序列运算符**$**来生成由比较规则的元素构成的序列。规则的元素指的是可以用表达式或命令统一表述的元素。下面分别介绍这两种方法。

1. 命令 seq

用命令 **seq** 可以生成比较规则的序列，常用的命令格式为 **seq(f(i), i=m..n)**，输出的结果为由 **f(m)**、**f(m+1)**、…、**f(n)**的运算结果构成的序列。其中 **f** 为任意的单一表达式，**i** 为变量名；**f(i)**也可以与变量 **i** 无关，这时生成的是由重复元素构成的序列；**m** 和 **n** 为数字常量或字符常量。看下面的几个例子：

```
>sin_seq:=seq(sin(Pi/6*i),i=0..6);
```

$$sin\_seq:=0,\frac{1}{2},\frac{1}{2}\sqrt{3},1,\frac{1}{2}\sqrt{3},\frac{1}{2},0$$

```
>a_seq:=seq(a[i],i=3..5);
```

$$a\_seq:=a_3,a_4,a_5$$

```
>seq(x,i="a".."d");
```

$$x,x,x,x$$

```
>s_seq:=seq("p".i,i="a".."d");
```

$$s\_seq:="pa","pb","pc","pd"$$

在命令 **seq(f(i), i=m..n)**中，**m** 和 **n** 必须是确定的数字常量或字符常量，在特殊情况下，**m** 可以是**-infinity**（-∞），**n** 可以是 **infinity**（∞）。如果 **m>n**，输出结果为 **NULL**。

```
[>seq(i,i=0..-infinity);
```

```
[>seq(i,i=0..-1);
[>|
```

也可以使用 **seq(f(i), i=x)**的命令格式来生成序列，输出的结果是由 **f(op(1, x))**、**f(op(2, x))**、…、**f(op(nops(x), x))**的运算结果构成的序列。其中 **x** 为任意单一的表达式，通称为集合或者列表，也可以是字符串、算术表达式或者命令 **op** 可以作用的其他表达式等。看下面的例子：

```
[>seq(a.i,i=[3,2,1,4,3]);
```

$$a3, a2, a1, a4, a3$$

```
[>seq(x,i="hello");
```

$$x, x, x, x, x$$

```
[>seq(i,i=x^2+y^2-x*y);
```

$$x^2, y^2, -x\,y$$

在上面的两种命令格式中，变量 i 是命令 **seq** 的私有变量，当命令 **seq** 执行完毕后，对当前工作单中同名的变量 **i** 不发生任何影响。

2. 序列运算符**$**

序列运算符**$**也可以用来生成比较规则的序列。命令 **f(i)$i=m..n** 的输出结果和命令 **seq(f(i), i=m..n)**的输出结果相同，两个命令中的符号具有相同的意义。不同的是，命令 **f(i)$i=m..n** 中的变量 **i** 必须为无值标识符，否则将引发错误：命令执行完毕后，变量 **i** 仍保持为无值状态。看下面的例子：

```
[>i:='i':
 i$i=0.1..4;
 i;
```

$$1, 1.1, 2.1, 3.1$$

$$i$$

```
[>(a[i<>b[i])$i=-1..3;
```

$$a_{-1} \neq b_{-1}, a_0 \neq b_0, a_1 \neq b_1, a_2 \neq b_2, a_3 \neq b_3$$

```
[>p.i$i="d".."a";
[>i:=I:
 x$i=2..3;
Error,wrong number (or type) of parameters in function $
```

另外，命令 **i$i=m..n** 可以简化为 **$m..n** 的格式，命令 **expr$n** 与命令 **expr$i=1..n** 等价；但在命令 **expr$n** 中，Maple V 假定 **expr** 与变量 **i** 无关，即使 **expr** 中包含了变量 **i** 也是如此。看下面的例子：

```
> $"a".."e";
```

"a","b","c","d","e"

```
> $2/3..11/3;
```

$$\frac{2}{3},\frac{5}{3},\frac{8}{3},\frac{11}{3}$$

```
> x[i]$4;
```

$$x_i, x_i, x_i, x_i$$

```
> P.i$5;
```

π, π, π, π, π

6.2 数列和求和

数列是序列的一个子集，其元素都是数字或者其他的算术表达式。求和是数列运算中最基本的一种运算。本节介绍数列的定义和相应的求和命令。

6.2.1 数列的定义

由数字或者算术表达式构成的序列称为数列。数列可以看作是以正整数为自变量的一种特殊的函数，引用数列元素的方法可以看作是一种特殊的函数调用。下面以定义的数列 **numseq** 为例，可以把序列名 **numseq** 看作是一个特殊的函数，其自变量为正整数，定义域为孤立的点构成的集合**{1,2,3,4,5,6}**，调用该函数的方法为 **numseq[i]**，其中 **i** 为对应的特殊点。这与定义的函数 **f** 具有相同的作用。

```
> numseq:=seq(cos(Pi/3*i),i=0..5);
```

$$numseq := 1, \frac{1}{2}, \frac{-1}{2}, -1, \frac{-1}{2}, \frac{1}{2}$$

```
> numseq[1];
```

1

```
> f:=(i::positive)->numseq[i];
```

$$f := i :: positive \to numseq_i$$

```
> f(1),f(4);
```

1,−1

但这种近似只适用于包含有限元素的数列，而且是在数列元素已知的情况下。在 6.1.2 节中介绍的生成序列的方法都可以用来生成这种有限数列。

在数学中，经常会遇到无穷数列的例子，通常会用一个通项表达式来代表数列中的每一项。这个通项表达式也是以正整数为自变量的函数，通过这个通项表达式可以写出数列中指定的有限项。在使用命令 **seq(f(i),i=m..n)**或者 **f(i)$i=m..n** 生成有限数列的时候，其中

的 **f(i)**就代表了原无穷数列的通项表达式。用户也可以定义自己的函数作为无穷数列的通项表达式，或者生成需要的有限数列。

在定义数列的通项表达式时，可以使用递推的方法。有名的 Fibonacci 数列就是用递推关系定义的：$f_n=f_{n-1}+f_{n-2}$（n>2），其中 $f_1 = f_2 = 1$ 。在 Maple V 的 **combinat** 函数库中提供了函数 **fibonacci**，用户可以用命令 **fibonacci(i)**调用 Fibonacci 数列中的第 i 项。在使用函数 **fibonacci** 之前要加载 **combinat** 函数库。下面是使用函数 **fibonacci** 的例子：

```
>with(combinat):
>fibonacci(i)$i=1..10;
                          1,1,2,3,5,8,13,21,34,55
>fibonacci(100);
                              354224848179261915075
>length(%);
                                      21
```

最后一个命令的结果表明，Fibonacci 数列的第 100 项是一个具有 21 位数字的整数。

6.2.2　求和

对于由数字或者算术表达式构成的数列，用户可以对其中的元素求和。Maple V 中提供 **add** 和 **sum** 了两个求和命令用来计算数列的各项和。其中，命令 **add** 用来计算有穷数列的和，命令 **sum** 主要用来计算无穷数列的和。另外，与命令 **add** 相对应的是，命令 **mul** 用来计算有限数列的积；与命令 **sum** 相对应，命令 **product** 用来计算无穷数列的积。本小节主要介绍两个求和命令的使用，关于两个求积命令的使用，读者可以参看相关的帮助。

1. **add**

命令 **add** 用来计算确定数列的各项和，确定数列指的是数列的元素和数列元素的个数是给定的。基本的命令格式为：**add(f(i), i=1..n)**，用以计算 **f(1) + f(2) + … + f(n)**。其中 **f(i)**为数列元素的通项表达式，**i** 为数列元素的序号，**n** 为给定的整数。下面给出数列求和的例子：

```
>add(i^2,i=1..5);
                                  55
>seq1:=3,2,1,1,2,3;
                          seq1 := 3,2,1,1,2,3
>add(seq1[i],i=1..nops([seq1]));
                                  12
```

命令 **add(f(i), i=m..n)**是一种更通用的格式，用来计算 **f(m) + f(m+1) + … + f(n)**。其中 **f(i)**代表任意以 **i** 为自变量的代数表达式；**m** 和 **n** 代表有限的数字常量，可以是整数、

分数或浮点数，通常 **m≤n**。如果 **m>n** 则命令的返回结果为 **0**；如果 **m** 和 **n** 是未知的变量，将引发错误。看下面的例子：

```
>add(a[i]*x^i,i=..5);
```

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

```
>add(sin(i*Pi),i=1/6..25/6);
```

$$\frac{1}{2}$$

```
>add(i-1,i=-2.3..10);
```

35.1

```
>add(i,i=infinity..0);
```

0

```
>add(i^2,i=0..n);
Error,unable to execute add
```

命令 **add** 还有一种命令格式：**add(f(i), i=x)**，用来计算数列 **seq(f(i), i=x)**的和，其中的符号 **f**、**i**、**x** 的意义和命令 **seq(f(i), i=x)**中的相应符号意义相同。看下面的例子：

```
[>L:=[1,2,3,4,5]:
```

```
>add(a[i]*(x-i)^i,i=L);
```

$$a_1(x-1) + a_2(x-2)^2 + a_3(x-3)^3 + a_4(x-4)^4 + a_5(x-5)^5$$

```
>add(i,i=x^2*y^(-1)*sin(z));
```

$$x^2 + \frac{1}{y} + \sin(z)$$

和命令 **seq** 类似，命令 **add** 中的变量 **i** 是命令 **add** 的私有变量，命令执行时与工作空间中的同名变量是无关的。

2. **sum**

命令 **sum** 不仅能够完成命令 **add** 所能完成的求和计算，而且能够完成不确定数列包括有穷数列和无穷数列的求和计算。命令 **sum** 通常用于求和公式的计算，在需要计算确定的有穷数列各项和时，建议使用更为有效的命令 **add**。

命令 **sum** 的基本格式为 **sum(f(k), k)**，用来计算由函数 **f(k)**确定的数列的前 **k** 项和。其中 **f(k)**代表数列的通项表达式，**k** 是不确定的正整数变量，但不能是有值标识符。命令返回的结果是以 **k** 为自变量的求和公式，这里记为 **g(k)**，对所有的正整数 **k** 满足：**f(k) = g(k+1)-g(k)**。命令 **sum** 的惰性格式为 **Sum** ，其输出以数学中的求和符号形式来表示所计算的求和表达式，默认情况下求和符号以黑色显示。下面是几个常见数列的求和公式：

```
> Sum('k','k')=sum('k','k');
```

$$\sum_k k = \frac{1}{2}k^2 - \frac{1}{2}k$$

```
> Sum('k^2','k')=sum('k^2','k');
```

$$\sum_k k^2 = \frac{1}{3}k^3 - \frac{1}{2}k^2 + \frac{1}{6}k$$

```
> Sum('k^3','k')=sum('k^3','k');
```

$$\sum_k k^3 = \frac{1}{4}k^4 - \frac{1}{2}k^3 + \frac{1}{4}k^2$$

在上面的例子中，标识符和通项表达式都用单引号界定，防止其中的变量以前的赋值对求和计算产生意外的影响，这在很多情况下都是非常有必要的。下面给出其中第三个求和公式的验证：

```
> sum('k^3','k');
```

$$\frac{1}{4}k^4 - \frac{1}{2}k^3 + \frac{1}{4}k^2$$

```
> g:= unapply(%,k);
```

$$g := k \to \frac{1}{4}k^4 - \frac{1}{2}k^3 + \frac{1}{4}k^2$$

```
> expand(g(k+1)-g(k));
```

$$k^3$$

命令 **sum** 也可以完成很多复杂的求和计算。当命令 **sum** 给出简化的求和结果时以求和符号的形式表示，但求和符号的颜色为蓝色，对应的表达式类型为 **function**。看下面的例子：

```
> Sum('- exp(-k)','k')=sum('- exp(-k)','k');
```

$$\sum_k (-e^{(-k)}) = \frac{e}{(-1+e)e^k}$$

```
> Sum('k*a^k','k')=sum('k*a^k','k');
```

$$\sum_k ka^k = \frac{a^k(ka-k-a)}{(a-1)^2}$$

```
> sum('a[k]*x^k','k');
```

$$\sum_k a_k x^k$$

```
> whattype(%);
```

function

命令 **sum** 还有一种命令格式 **sum('f', 'k'=m..n)**，用来计算 **f(m) + f(m+1) + … + f(n)**，其中 **m** 和 **n** 都是整数或整数变量，通常 **m≤n**。如果用 **g(k)**表示通用的求和公式，命令 **sum('f', 'k'=m..n)**的计算结果为 **g(n+1)-g(m)**。和命令 **add** 不同的是，计算的结果通常不是最简的，用户可以用命令 **expand** 等进一步化简。看下面的例子：

```
> Sum('k^2','k'=0..n)=sum('k^2','k'=0..n);
```

$$\sum_{k=0}^{n} k^2 = \frac{1}{3}(n+1)^3 - \frac{1}{2}(n+1)^2 + \frac{1}{6}n + \frac{1}{6}$$

```
> expand(%);
```

$$\sum_{k=0}^{n} k^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

```
> Sum('a[k]*x^k','k'=0..4)=sum('a[k]*x^k','k'=0..4);
```

$$\sum_{k=0}^{4} a_k x^k = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

在命令 **sum('f', 'k'=m..n)**中，如果 **m** 或 **n** 是分数或浮点数常量，Maple V 首先将命令转化为 **sum('f', 'k'=trunc(m)..trunc(n))**。当 **m>n** 时，如果 **m=n+1**，命令 **sum('f', 'k'=m..n)** 返回值为 **0**；如果 **m>n+1**，命令的返回值为 **-sum('f', 'k'=n+1..m-1)**。看下面的例子：

```
> trunc(-7/3),trunc(4.8);
```

$$-2, 4$$

```
> sum(k,k=-7/3..4.8);
```

$$7$$

```
> sum('k^2','k'=1..n-1);
```

$$\frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n$$

```
> sum('k^2','k'=n..0);
```

$$-\frac{1}{3}n^3 + \frac{1}{2}n^2 - \frac{1}{6}n$$

```
> sum('a[k]*x^k','k'=n+1..n);
```

$$0$$

命令 **sum** 的另一个主要用途是计算无穷数列所有项的和，基本格式为 **sum('f', 'k'=m..infinity)**。下面是两个计算无穷和的例子：

```
> Sum('1/k!','k'=0..infinity)=
      sum('1/k!','k'=0..infinity);
```

$$\sum_{k=0}^{\infty}\frac{1}{k!}=e$$

```
> Sum('1/k^2','k'=1..infinity)=
      sum('1/k^2','k'=1..infinity);
```

$$\sum_{k=1}^{\infty}\frac{1}{k^2}=\frac{1}{6}\pi^2$$

```
> f:=x->sum(c[n]*x^n,n=0..infinity);
```

$$f:=x\to\sum_{n=0}^{\infty}c_n x^n$$

上面的最后一个例子定义了函数 f(x)，如果其各项的系数 c_n 与自变量 x 无关，函数 f(x) 就是数学上最常用的幂级数。但在 Maple V 中，级数作为一种数据类型，与数学上的级数定义是不同的，下一节中将介绍级数的有关知识。

6.3　级数

Maple V 中级数的定义和数学上的级数是不同的，具有自己的运算规则。Maple V 提供了级数展开的命令，允许用户进行级数展开运算。本节首先介绍级数的定义，接着介绍级数的展开和运算规则。

6.3.1　级数的定义

1. 级数的数学定义

在数学上无穷数列 $\{a_n\}_1^{\infty}$ 的和 $\sum_{n=1}^{\infty}a_n$ 称为无穷级数，并简称为级数，其中 a_n 为数列或者级数的通项。前面介绍的求和命令 **sum** 能自动选用各种方法完成求和计算，但不能判断级数收敛与否，对发散的级数可能给出并不正确的计算结果。因此，在计算级数时，应该首先确定级数是否收敛。关于级数的收敛性判断，在数学上有相应的定理，这里不作介绍。

幂级数是科学计算中最常用的级数。幂级数具有下面的形式：

$$\sum_{n=1}^{\infty} a_n = \sum_{n=0}^{\infty} c_n x^n = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots$$

其中的通项 $a_n = c_n x^n$，c_n 是各项的系数，与 x 无关。幂级数可能是收敛的，也可能是不收敛的，这一点取决于 x 的值和各项的系数。

级数展开是研究复杂数学函数的一个重要工具，很多数学函数可以在给定的点展开为无穷级数的形式，但更为有用的是截取无穷级数的前若干项得到的截尾级数。截尾级数可以作为数学函数的一种近似，截尾级数与原数学函数之间的差额称为级数展开的余项。Taylor 级数展开是最常用的级数展开，Taylor 级数具有多项式的性质，在数值逼近、误差分析等领域得到了广泛的应用。

2. Maple V 中的级数

与数学中的级数定义不同，Maple V 中的级数通常是由级数展开命令 **series** 或 **taylor** 得到的。在 6.3.2 节中将会看到，命令 **series** 或 **taylor** 的输出结果具有截尾级数的形式，余项通常由大写字母 O 表示。Maple V 把 O 作为一种特殊的函数使用。级数的类型名为 **series**，**taylor** 是其中的一个子数据类型；如果展开后的级数具有 Taylor 多项式的形式，则同时属于 **series** 和 **taylor** 数据类型。下面用命令 **series** 对正弦函数 **sin(x)**级数展开，用命令 **whattype** 和 **type** 检查输出结果的数据类型：

```
> sin_series:= series(sin(x),x=0,5);
```

$$sin\_series := x - \frac{1}{6}x^3 + O(x^5)$$

```
> whattype(sin_series);
```

$$series$$

```
> type(sin_series,taylor),type(sin_series,polynom);
```

$$true, false$$

上面的展开结果中，最后一项 $O(x^5)$ 是级数展开的余项。虽然函数 **sin(x)**的展开结果具有和多项式类似的形式，但是不属于多项式的数据类型 **polynom**。

在 Maple V 中，级数的各项通常都具有 $c_n(x-a)^n$ 的形式，其中 **x** 为自变量，a 为展开的基准点，c_n 为该项的系数，**n** 为对应的指数且必须为整数。余项通常具有

$O((x-a)^r)$ 的形式，系数定义为 O(1)，对应的指数为整数 r。如果其中某项的指数不是整数，则该表达式不属于级数数据类型，只是作为一般的算术表达式。看下面的例子：

```
> exp_series:=series(exp(x-1),x=1,4);
```

$$exp\_series := 1 + x - 1 + \frac{1}{2}(x-1)^2 + \frac{1}{6}(x-1)^3 + O((x-1)^4)$$

```
> xx_series:=series(x^x,x=0,3);
```

$$xx\_series := 1 + \ln(x)x + \frac{1}{2}\ln(x)^2 x^2 + O(x^3)$$

```
> type(xx_series,series),type(xx_series,taylor);
```

$$true, false$$

```
> ss:=series(sqrt(sin(x)),x=0,4);
```

$$ss := \sqrt{x} - \frac{1}{12}x^{\left(\frac{5}{2}\right)} + O\left(x^{\left(\frac{7}{2}\right)}\right)$$

```
> type(ss,series),whattype(ss);
```

$$false, +$$

命令 **op** 和 **nops** 用来分析级数的结构。命令 **op(0,series)**返回值为各项的底数 **x-a**；**op(2*i-1,series)**返回级数 **series** 中出现的第 **i** 项的系数，**op(2*i,series)**返回第 **i** 项的整数指数。利用上面定义的几个级数，看下面的结果：

```
> op(0,exp_series),op(0,xx_series);
```

$$x-1, x$$

```
> op(exp_series);
```

$$1,0,1,1,\frac{1}{2},2,\frac{1}{6},3,O(1),4$$

```
> op(xx_series);
```

$$1,0,\ln(x),1,\frac{1}{2}\ln(x)^2,2,O(1),3$$

```
> nops(exp_series),nops(xx_series);
```

$$10,8$$

6.3.2 级数展开

Maple V 中提供了两个级数展开的命令，其中命令 **series** 是通用的级数展开命令，用于把给定的函数展开为广义级数的形式；命令 **taylor** 是专用的 Taylor 级数展开命令，用于将给定的函数展开为 Taylor 多项式的形式。下面分别对这两个命令进行介绍。

1. 广义级数展开

基本的命令格式为：**series(expr,x=a, n)**，用来对以 **x** 为自变量的代数表达式 **expr** 在 **x=a** 处进行级数展开，余项的阶数为 **n**。其中 **n** 为确定的非负整数，在缺省时余项的阶数由系统变量 **Order** 的值决定。系统变量 **Order** 的默认值为 **6**，用户可以重新设置 **Order** 的值。如果第二个参数为 **x**，Maple V 默认在 **x=0** 处进行级数展开。

命令 **series** 的输出结果具有截尾级数的结构，余项 $O((x-a)^r)$ 代表了展开的近似程度，r 给出了展开后余项的实际阶数。需要指出的是，r 与命令 **series(expr,x=a, n)**中的指定阶数 **n** 可能是不一样的，因为 Maple V 自动略去系数为 **0** 的展开项。如果展开后的级数和原来的函数或表达式具有相同的形式，在命令 **series** 的输出结果中将不包含余项，例如对次数较低的多项式函数进行级数展开时，其输出结果中即不包含余项。下面是几个级数展开的例子：

```
> series(x+1/x,x=1,4);
```

$$2+(x-1)^2-(x-1)^3+O((x-1)^4)$$

```
> series(a*x^3+b*x^2+c*x+d,x);
```

$$d+cx+bx^2+ax^3$$

```
> whattype(%),type(%,polynom);
```

$$series, false$$

```
> Order;
```

$$6$$

```
> series(x/(1-x-x^2),x);
```

$$x+x^2+2x^3+3x^4+5x^5+O(x^6)$$

```
> Order:=8:series(x/(1-x-x^2),x);
```

$$x+x^2+2x^3+3x^4+5x^5+8x^6+13x^7+O(x^8)$$

上面的几个例子，级数展开的结果都具有 Taylor 级数的形式。命令 **series** 是一个广义的级数展开命令，命令 **series** 的结果也可以是 Laurent 级数或者更为普通的级数。广义级数展开的第 **i** 项的系数 **coeff[i]**满足：当 **x** 趋近于 **a** 时，对任意的 **eps>0**，

k1*(x-a)^eps < |coeff[i]| < k2/(x-a)^eps

其中，**k1** 和 **k2** 是常数。或者说，广义级数的各项系数可能与 **x** 有关，但是系数的增长要比该项的幂的增长速度慢。看下面的例子：

```
> s1:=series(exp(x)/x,x=0,4);
```

$$s1 := x^{-1} + 1 + \frac{1}{2}x + \frac{1}{6}x^2 + O(x^3)$$

```
> type(s1,series),type(s1,taylor),type(s1,laurent);
```

$$true, false, true$$

```
> op(s1);
```

$$1,-1,1,0,\frac{1}{2},1,\frac{1}{6},2,\frac{1}{24},3,O(1),4$$

```
> series(x^x,x=0,5);
```

$$1 + \ln(x)x + \frac{1}{2}\ln(x)^2 x^2 + \frac{1}{6}\ln(x)^3 x^3 + \frac{1}{24}\ln(x)^4 x^4 + O(x^5)$$

```
> sx1:=series(1/(x-1)^3,x=1);
```

$$sx1 := (x-1)^{-3}$$

```
> op(0,sx1),whattype(sx1);
```

$$x-1, series$$

```
> op(sx1);
```

$$1,-3$$

通常命令 **series** 输出结果的数据类型为级数，类型名为 **series**，其特征为各展开项的指数为整数；但是如果展开项的指数只能用分数或小数表示，则展开得到的表达式不属于级数数据类型。

如果命令中所给的展开基准点为无穷大，即 **series(expr, x=infinity, n)**，则 Maple V 对表达式 **expr** 沿渐近线进行展开。这时的展开结果也不属于级数数据类型。看下面的例子：

```
> series(x^3/(x^4+4*x-5),x=infinity);
```

$$\frac{1}{x} - 4\frac{1}{x^4} + 5\frac{1}{x^5} + O\left(\frac{1}{x^7}\right)$$

```
> series(sin(1/x),x=infinity);
```

$$\frac{1}{x} - \frac{1}{6}\frac{1}{x^3} + \frac{1}{120}\frac{1}{x^5} + O\left(\frac{1}{x^6}\right)$$

```
> whattype(%),type(%,series);
```

$$+, false$$

另外，命令 **series** 可以直接把未计算的积分表达式进行级数展开。看下面的例子：

```
> Int(exp(x^3),x);
```

$$\int e^{(x^3)}dx$$

```
> series(%,x);
```

$$x+\frac{1}{4}x^4+O(x^7)$$

2. Taylor 级数展开

在理论分析和科学应用中，Taylor 级数展开是最常用的一种近似处理方法。Taylor 级数的命令为 **taylor**，基本的格式为 **taylor(expr, x=a, n)**，用来把表达式 **expr** 在 **x=a** 这一点展开为 Taylor 多项式的形式，指定的余项阶数为 **n**。当 **n** 缺省时，余项的阶数由系统变量 **Order** 的值确定。下面是几个 Taylor 级数展开的例子：

```
> taylor(exp(x),x,4);
```

$$1+x+\frac{1}{2}x^2+\frac{1}{6}x^3+O(x^4)$$

```
> whattype(taylor1),type(taylor1,taylor);
```

series,true

```
> taylor(f(x),x);
```

$$f(0)+D(f)(0)x+\frac{1}{2}(D^{(2)})(f)(0)x^2+\frac{1}{6}(D^{(3)})(f)(0)x^3+\frac{1}{24}(D^{(4)})(f)(0)x^4+\frac{1}{120}(D^{(5)})(f)(0)x^5+O(x^6)$$

和命令 **series** 一样，用户也可以用命令 **taylor** 把未计算的表达式展开为 Taylor 多项式的形式。看下面的例子：

```
> Order:=6:int(f(x),x);
```

$$\int f(x)dx$$

```
> taylor(%,x);
```

$$f(0)x+\frac{1}{2}D(f)(0)x^2+\frac{1}{6}(D^{(2)})(f)(0)x^3+\frac{1}{24}(D^{(3)})(f)(0)x^4+\frac{1}{120}(D^{(4)})(f)(0)x^5+O(x^6)$$

Taylor 级数展开是有条件的，有的函数不能够进行 Taylor 级数展开；这时命令 **taylor** 将给出相应的错误信息。看下面的例子：

```
> taylor(1/x,x=0);
Error,does not have a taylor expansion,try series()
```

6.3.3　级数的运算

级数是一种新的数据类型，其运算的规则和其他数据类型也不同的。为了实现更多的运算，用户可以把级数转化为相应的多项式。

1. 基本的运算规则

前面指出，级数和多项式是两种不一样的数据类型，因此许多对多项式适用的操作对级数来说都是不适用的。例如，除了展开的基准点外，命令 **subs** 不能用来对级数中的自变量进行替换；命令 **expand** 也不能完成级数的化简。看下面的例子：

```
> sin_series:=series(sin(x),x);
```

$$sin\_series := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^6)$$

```
> subs(x=0,sin_series);
```

$$0$$

```
> subs(x=1/2,sin_series);
Error,invalid substitution in series
```

```
> sin_series*sin_series;
```

$$\left(x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x6)\right)^2$$

```
> expand(%);
```

$$\left(x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^6)\right)^2$$

另一方面，用户可以再次用命令 **series** 对级数构成的表达式进行化简。用户也可以计算级数的微分、积分，求解包含级数的方程等，这些运算的结果仍然具有级数的形式。看下面的例子：

```
> series(%,x);
```

$$x^2 - \frac{1}{3}x^4 + O(x^6)$$

```
> exp_series:=series(exp(x),x,4);
```

$$exp\_series := 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + O(x^4)$$

```
> diff(exp_series,x);
```

$$1+x+\frac{1}{2}x^2+O(x^3)$$

```
> solve(y=sin_series,x);
```

$$y+\frac{1}{6}y^3+\frac{3}{40}y^5+O(y^6)$$

方程 y= sin_series 的求解结果和反正弦函数 arcsin(x)的级数展开结果是相同的，即：

```
> series(arcsin(y),y,6);
```

$$y+\frac{1}{6}y^3+\frac{3}{40}y^5+O(y^6)$$

2. 级数与多项式的转化

为了计算方便，用户可以使用命令 **convert** 将级数转换为其他的数据类型。常见的是把级数转化为多项式，命令的格式为 **convert(series_expr, polynom)**。看下面的例子：

```
> sin_series;
```

$$x-\frac{1}{6}x^3+\frac{1}{120}x^5+O(x^6)$$

```
> sin_poly:=convert(sin_series,'polynom');
```

$$sin\_poly := x-\frac{1}{6}x^3+\frac{1}{120}x^5$$

```
> whattype(sin_series),whattype(sin_poly);
```

$$series,+$$

反过来，用户只能使用命令 **series** 把多项式转化为级数，命令 **convert** 不能实现这种操作。

第 7 章　列表、集合和数组

Maple V 中提供了序列、列表、集合、数组、矩阵、向量等多种复合数据结构，其中序列在第 6 章中已作了介绍，矩阵和向量作为数组的特殊类型将放在第 11 章中介绍。在 Maple V 中，经常遇到使用列表、集合和数组的例子，本章简单介绍这三种复合数据结构。

7.1　列表

列表是 Maple V 中经常使用的一种复合数据结构，许多命令的参数或输出结果都采用列表的形式来表示。本节首先介绍列表的定义，接着系统地介绍列表的基本操作。

7.1.1　列表的基本知识

用一对方括号界定一个序列就构成了列表。列表中的元素按给定的顺序排列，可以是除了序列以外的任何数据类型，并且可以重复出现。用户可以用 6.1 节中介绍的序列生成方法构造相应的列表，只需用方括号界定生成的序列即可。和序列一样，用户可以对列表命名，列表的名字不能再用来构造其他的索引标识符。用户可以使用列表名后跟方括号的办法引用列表中的一个或多个元素，给定的元素下标不能超过元素的个数。下面给出几个列表的例子：

```
> list1:=[seq(x[i],i=1..4)];
```

$$list1 := [x_1, x_2, x_3, x_4]$$

```
> list1[2];
```

$$x_2$$

```
> list1[5];
Error,invalid subscript selector
```

```
> list2:=[$"a".."f"];
```

$$list2 := ["a","b","c","d","e","f"]$$

```
> list3:={p.(2..4),"i am string in a list"];
```

$$list3 := [p2, p3, p4, \text{"i am string in a list"}]$$

```
> list3[4],list3[4][3..11];
```

"i am string in a list","am string"

如果引用了列表中的多个元素，则返回的结果是由这多个元素构成的新的列表。看下

面的例子：

```
> list2[2..4];
```

["b","c","d"]

```
> list3[-2..-1];
```

[*p4*,"i am string in a list"]

Maple V 用系统变量 **NULL** 表示空序列，不含任何元素的空列表用**[NULL]**表示，或者直接用不包含任何元素的一对方括号表示。下面是一个空的列表：

```
>'emptylist':=[];
```

empty list := [ ]

通过直接对列表的某个元素赋值，用户可以改变列表中的相应元素值。看下面的例子：

```
> L:=[1,2,3];
```

$L := [1,2,3]$

```
> L[1]:=3;
  L;
```

$L_1 := 3$

[3,2,3]

但用户不能把元素赋值为 **NULL** 或者释放元素值，下面的命令都是错误的：

```
> L[1]:='L[1]';
Error,lhs of a table should be a NAME or a TABLEREF
> L[1]:=NULL;
Error,expression sequences cannot be assigned to lists
```

借助于命令 **map**，用户可以对列表中的元素执行共同的运算或者其他操作，返回的结果构成新的列表。下面给出一个简单的例子：

```
> map(x->x^2,L);
```

[1,4,9]

列表的类型名为 **list**，用户可以用 **whattype**、**type**、**op**、**nops** 等命令分析列表的类型和结构。看下面的例子：

```
> whattype(list1),op(0,list3);
```

list, *list*

```
> nops(list3),nops('empty list');
```

4,0

```
> type('empty list',list);
```

true

7.1.2　列表的基本操作

本小节将比较系统地介绍列表的基本操作，包括列表的长度，列表元素的确认、条件选择、删除、插入及替换，列表结构的重排等。下面将分别介绍这些基本操作。

1. 列表的长度

Maple V 把列表作为一种特殊的表达式，列表中的每个元素为该表达式的操作数。列表的长度定义为其元素的个数，因此可以用命令 **nops** 获得列表的长度。很显然，空列表的长度为 **0**。下面首先定义了一个列表 **color0**，并用命令 **nops** 来考察其长度。

```
> colorl:=[red,orange,yellow,green,blue,indigo,purple];
```

$$colorl := [red, orange, yellow, green, blue, indigo, purple]$$

```
> nops(colorl);
```

$$7$$

2. 列表元素的确认

命令 **member** 是一个逻辑函数，基本的命令格式为 **member(x, lst)**，其中 **x** 为任意的单个表达式，**lst** 代表任意的列表。如果 **x** 是列表 **lsl** 中的元素，命令返回 **true**，否则返回 **false**。

```
> member(black,colorl);
```

$$false$$

```
> member(indigo,colorl);
```

$$true$$

用户也可以使用 **member(x, lst, k)**的格式，其中 **k** 是无值标识符。为防止变量 **k** 原来已赋值，可以在 **k** 的两边用单引号界定，即使用 **member(x, lst, 'k')**的格式。如果 **x** 为列表 **lst** 的元素，则命令 **member(x, lst, 'k')**把列表中第一个元素 **x** 的序号赋值给变量 **k**，并返回 **true**；否则返回 **false**，**k** 仍处于原来的状态。看下面的例子：

```
> k:='k':k:=0:
> member(black,colorl,'k'),k;
```

$$false, 0$$

```
> member(indigo,colorl,k);
Error,wrong number (or type) of parameters in function member
> member(indigo,colorl,'k'),k;
```

$$true, 6$$

3. 列表元素的条件选择

命令 **select(f, lst)**可以选择列表 **lst** 中符合条件 **f** 的元素，并构成一个新的列表。其中 **f** 为逻辑函数，给出了查找的条件；如果 **f** 有参数，需使用 **select(f, lst, b1, b2,..., bn)**的格

式，其中 **b1**、**b2**、…、**bn** 为参数列表。看下面的例子：

```
> die:=rand(-10..10):
  list0=['die()'$10];
```

$$list0 := [7,8,10,-6,-8,-5,7,6,-6,0]$$

```
> select(isprime,list0);
```

$$[7,7]$$

```
> select(type,list0,nonnegint);
```

$$[7,8,10,7,6,0]$$

```
> select(has,[[x,z],y,cos(x),x],x);
```

$$[[x,z],\cos(x),x]$$

上面的例子中用到了 **isprime**、**type**、**has** 三个逻辑函数，读者可以参看相关的帮助，这里不再介绍。

4. 列表元素的替换

用户可以用命令 **subsop(i=newelement, lst)**把列表 **lst** 中的第 **i** 个元素替换为新的元素 **newelement**，从而构成一个新的列表。看下面的例子：

```
> subsop(1=white,color1);
```

$$[white, orange, yellow, green, blue, indigo, purple]$$

```
> color1;
```

$$[red, orange, yellow, green, blue, indigo, purple]$$

```
> color1:=subsop(6=white,color1);
```

$$color1 := [red, orange, yellow, green, blue, white, purple]$$

```
> color1;
```

$$[red, orange, yellow, green, blue, white, purple]$$

另外，用户可以用命令 **subs(oldelement = newelement, lst)**把列表 **lst** 中的元素 **oldelement** 全部用新的元素 **newelement** 来替换，从而构成一个新的列表。看下面的例子：

```
> color1:=subs(white=indigo,color1);
```

$$color1 := [red, orange, yellow, green, blue, indigo, purple]$$

```
> color1;
```

$$[red, orange, yellow, green, blue, indigo, purple]$$

5. 列表元素的删除与插入

用户可以用命令 **subsop** 把列表中的某个元素用空值 **NULL** 替换，相当于删除了该元素，从而构成新的列表。或者可以用引用列表元素的方法从原来的列表中取出若干个元素构成新的列表。看下面的例子：

```
> color2:=subsop(6=NULL,7=NULL,color1);
```

$$color2 := [red, orange, yellow, green, blue]$$

```
> color3:=[op(6..7,color1)];
```

$$color3 := [indigo, purple]$$

与命令 **select** 相反，命令 **remove** 用于删除列表中符合给定条件的元素，返回由余下的元素构成的新的列表。如果给定的条件是无参数的逻辑过程或逻辑函数，命令的格式为 **remove(f, lst)**，其中 **f** 为返回逻辑值的过程或函数；如果过程或函数 **f** 需要参数，命令的格式为 **remove(f, lst, b1,..., bn)**，其中 **b1**、…、**bn** 为 **f** 的参数序列。看下面的例子：

```
> remove(isprime,list0);
```

[8,10,–6,–8,–5,6,–6,0]

```
> remove(type,list0,negint);
```

[7,8,10,7,6,0]

```
> list0;
```

[7,8,10,–6,–8,–5,7,6,–6,0]

```
remove(has,[x^2+y,xy,y^2-1,[cos(x),sin(y)]],x);
```

$$[xy, y^2 - 1]$$

使用列表元素引用的方法，可以在原列表的任何位置插入新的元素，从而构成新的列表。在下面的例子中，分别在列表 **color1** 的开头、结尾和中间插入了一个元素：

```
> [crimson,op(color1)];
```

$$[crimson, red, orange, yellow, green, blue, indigo, purple]$$

```
> [op(color1),crimson];
```

$$[red, orange, yellow, green, blue, indigo, purple, crimson]$$

```
> [op(1..3,color1),crimson,op(4..7,color1)];
```

$$[red, orange, yellow, crimson, green, blue, indigo, purple]$$

6. 列表的合并

使用命令 **op** 引用列表中的元素可以把多个列表合并为一个新的序列。下面的例子把上面定义的两个列表 **color2** 和 **color3** 合并为新的列表 **color4**，与 **color1** 是相同的：

```
> [op(color2),op(color3)];
```

$$[red, orange, yellow, green, blue, indigo, purple]$$

下面的命令不能实现列表的合并：

```
> [color2,color3];
```

$$[[red, orange, yellow, green, blue], [indigo, purple]]$$

7. 列表的重排

命令 **sort(lst)**可以对列表 **lst** 中的元素重新排序，并生成新的列表。如果列表 **lst** 是的数列构成的，则 Maple V 按照数字的大小升序排列；如果列表 **lst** 是字符串列表，则按字

典顺序排列；其他的情况下，Maple V 将按照元素的机器地址顺序排列，这时元素的顺序在每次启动 Maple V 时都有可能不同。

```
>sort([3,2,5,6,0]);
```

[0,2,3,5,6]

```
>sort(color1);
```

[*blue, green, indigo, orange, purple, red, yellow*]

```
>color1;
```

[*red, orange, yellow, green, blue, indigo, purple*]

在命令 **sort** 中也可以用第二个参数规定排序的标准，即使用 **sort(lst, order)**的形式。其中，**order** 的取值可以是 **numeric**、**lexorder**、**adress**，分别代表以数字、字典、机器地址的顺序排序；同时 **order** 的取值也可以是用户定义的过程名，详细的用法可以用命令**?sort** 获得命令 **sort** 的更多信息。

```
>sort(colort1,address);
```

[*green, blue, orange, yellow, indigo, purple, red*]

如果要实现列表元素的翻转，用户可以使用类似下面的命令格式，用列表的名字替换其中的 **color1** 即可。

```
>[seq(color1[nops(color1)-i+1],i=1..nops(color1))];
```

[*purple, indigo, blue, green, yellow, orange, red*]

```
>[op(2..nops(color1),color),op(1,color1)];
```

[*orange, yellow, green, blue, indigo, purple, red*]

7.2 集合

集合也是 Maple V 中的一种复合数据结构，在 Maple V 的命令中经常用到以集合形式表示的参数。在数学上，集合之间有交集、并集和差集三种基本运算，用户也可以对集合中的元素执行共同的运算或者共同的操作。本节简单介绍集合的基本知识和基本运算。

7.2.1 集合的基本知识

与数学中的集合表示方法一样，用一对花括号{}界定一个序列就构成了集合。集合中的元素可以是除了序列之外的任何数据类型。和序列、列表不同的是，集合中的元素不能重复出现，而且是无序的，用户不能控制集合中元素的顺序；Maple V 将自动删除集合中重复的元素，并按照系统内部的物理地址排序输出正式的集合。

看下面的例子：

```
>s1:={4,2,1,5,3,2};
```

$s1 := [1,2,3,4,5]$

```
>s2:={x,z,y,x,y};
```

$$s2 := \{x, y, z\}$$

```
>lst:=[x,z,y,x,y]
```

$$lst := [x, y, z, x, y]$$

```
>s3:={sin(x),tan(x),cos(x),sec(x),csc(x)};
```

$$s3 := \{\sin(x), \tan(x), \cos(x), \sec(x), \csc(x)\}$$

```
>s4:={sin(x),cos(x),tan(x),csc(x),sec(x),cos(x)};
```

$$s4 := \{\sin(x), \tan(x), \cos(x), \sec(x), \csc(x)\}$$

虽然集合 **s2** 和列表 **lst** 输入时的元素是完全一样的，但最终结果中的元素是不一样的，集合 **s2** 中删除了重复的元素；而集合 **s3** 和 **s4** 输入时的元素不一样，但实际上是同一个集合。

用户可以采用与列表类似的方法引用集合中的一个或多个元素。如果引用的是多个元素，返回的是由所引用的元素构成的新的集合，但不能使用赋值语句对集合的元素赋值。看下面的例子：

```
>s1[2];
```

$$2$$

```
>s1[1]:=6;
Error,cannot assign to a set
```

```
>s3[-4..-2];
```

$$\{\tan(x), \cos(x), \sec(x)\}$$

```
>s3[-4..-2];
```

$$\{\tan(x), \cos(x), \sec(x)\}$$

没有任何元素的集合为空集，空集可以用**{NULL}**表示，或者直接用不包含任何元素的一对花括号表示。集合的数据类型名为 **set**，集合的每个元素对应着一个操作数，用户可以用 **whattype**、**type**、**op**、**nops** 等命令分析集合的类型和结构。看下面的例子：

```
>'empty set':={};
```

$$empty\ set := \{\ \}$$

```
>whattype(s1),type('empty set',set);
```

$$set, true$$

```
> nops(s2);
```

$$3$$

```
>op(s3);op(s4);
```

$$\sin(x), \tan(x), \cos(x), \sec(x), \csc(x)$$
$$\sin(x), \tan(x), \cos(x), \sec(x), \csc(x)$$

另外，用户也可以对集合执行与列表类似的其他操作，包括使用命令 **subs** 或 **subsop**

替换集合中的元素，使用命令 **member** 对给定的元素确认是否是集合中的元素，使用命令 **select** 或 **remove** 按照给定的条件选择或者删除集合中的元素等。这些命令的结果都生成了新的集合，这里不再一一介绍。但是，集合中的元素对用户而言是无序排列的，因此用户不能使用命令 **sort** 对集合中的元素排序。

7.2.2 集合的基本运算

集合间的基本运算包括交集、并集和差集，这三种运算的定义与数学中的定义是一样的。借助于命令 **map**，用户可以对集合中的元素执行共同的运算和操作。本小节主要介绍集合的三种基本运算，并介绍命令 **map** 对集合的作用以及集合的子集运算。

1. 集合的基本运算

Maple V 中提供了 **intersect**、**union**、**minus** 三个集合运算符。命令 **s1 intersect s2** 返回集合 **s1** 与 **s2** 的交集，即由集合 **s1** 和 **s2** 的公共元素构成的集合；命令 **s1 union s2** 返回集合 **s1** 与 **s2** 的并集，即由集合 **s1** 和 **s2** 的所有元素构成的集合；命令 **s1 minus s2** 返回集合 **s1** 与 **s2** 的差集，即由属于集合 **s1** 但不同时属于集合 **s2** 的元素构成的集合。下面给出了几个简单的集合运算的例子：

```
>{a,b} union {b,c};
```

$$\{a,c,b\}$$

```
>{a,b} intersect {b,c};
```

$$\{b\}$$

```
>{a,b} minus {b,c};
```

$$\{a\}$$

在这三种集合运算中，交集运算符 **intersect** 和并集运算符 **union** 的操作数可以交换位置，命令的输出结果相同；差集运算符的操作数互换后，输出的结果通常要发生变化。下面是对上面的三个集合运算的操作数互换位置后的情形：

```
>{b,c} intersect {a,b};
```

$$\{b\}$$

```
>{b,c}union{a,b};
```

$$\{a,b,c\}$$

```
>{b,c}minus{a,b};
```

$$\{c\}$$

在集合的混合运算表达式中，交集 **intersect** 的优先级最高，并集 **union** 和差集 **minus** 的优先级相同，计算的顺序为从左到右。用户也可以使用圆括号改变这三种集合运算的次序。读者可以比较下面的例子：

```
>a union b union a;
```

$$a \text{ union } b$$

```
>{a,b} minus {b,c} minus {a};
                                    { }

>{a,b} intersect {a,b} union {a};
                                    {a,b}

>{a,b} union {a} intersect {b,c};
                                    {a,b}

>({a,b} union {a}) intersect {b,c};
                                    {b}

>{a,b} minus {a} intersect {b,c};
                                    {a,b}

>{b,c} intersect {a,b} minus {a};
                                    {b}
```

三个集合运算符也可以采用另一种命令格式：即用命令`**intersect**`**(s1, s2, ...)**实现集合 **s1**、**s2** 等的交集，命令`**union**`**(s1, s2, ...)**实现集合 **s1**、**s2** 等的并集，命令`**minus**`**(s1, s2)**实现集合 **s1** 与 **s2** 的差集。前两个命令都可以有多个参数，且参数序列可以交换顺序而不影响输出的结果，命令` **minus**`**(s1, s2)**中只能有两个参数，且改变参数顺序后输出结果通常是不同的。三个集合运算符 **intersect**、**union**、**minus** 是 Maple V 的保护名，使用上述命令格式时需要用反引号界定。看下面的例子：

```
>'union'({b,c},{a,b},{a});
                                {a,b,c}

>'minus'({a,b},{b,c});
                                {a}
```

在集合运算表达式中，不能包含非集合类型的操作数。在集合的运算表达式中，如果含有未知的集合，命令的输出结果仍然使用集合的运算符表示。看下面的例子，其中集合外的标识符 **a** 代表未知的集合：

```
>'intersect'({a,b},{b,c},a);
                        a intersect {b}

>'intersect'({a,b},a,{b,c});
                        a intersect {b}
```

2. 命令 **map** 对集合的作用

命令 **map** 也可以对集合对象作用，使集合中的元素执行共同的运算或者其他操作。下面给出一个简单的例子：

```
> f:=x→x*exp(x):
  map(f,{a,b,c});
```

$$\{be^{b}, ae^{a}, ce^{c}\}$$

```
> d:={Pi,exp(1),1/3}:
  map(evalf,d);
```

{2.718281828,.3333333333,3.141592654}

用户也可以使用命令 **map** 对集合中的元素执行其他运算或操作，这里不再举例。

3. 集合的子集

集合的子集是由集合中的部分或全部元素构成的集合。如果集合的元素个数为 n，包括空集在内所有子集的个数为 2^n，非空子集的个数为 2^n-1。Maple V 在 **combinat** 函数库中提供了命令 **powerset**，可以得到集合的所有子集。命令的格式为 **powerset(s)**，返回的结果是由集合 s 的所有子集构成的集合；如果参数 s 是正整数，Maple V 默认是由前 s 个正整数构成的集合。注意在使用命令 **powerset** 之前要用命令 **with(combinat)**加载 **combinat** 函数库，或者使用 **combinat[powerset](s)**的形式。看下面的例子：

```
> powerset(3);
```

{{},{1},{1,2,3},{2,3},{3},{1,3},{2},{1,2}}

```
> powerset({sin(x),cos(x)});
```

$$\{\{\},\{\sin(x),\cos(x)\},\{\sin(x)\},\{\cos(x)\}\}$$

7.3 数组

数组是一种常用的数据结构，和其他许多编程语言一样，Maple V 允许用户创建一维数组、二维数组及多维数组。在使用 Maple V 进行科学计算的过程中，通常更多地使用数组中的两个特例，即矩阵和向量。本节简单介绍数组的创建以及数组元素的引用和更新，关于矩阵和向量的知识将在第 11 章中进行介绍。

7.3.1 数组的创建

1. 一维数组的创建

创建数组的命令为 **array**，其基本格式为 **array(bounds, list)**。其中，**bounds** 是整数范围表达式的序列，给出了数组每一维的索引范围；**list** 是一组等式的列表，等式的左边是元素的索引，等式的右边是与该索引对应的元素值。

一维数组是最简单的数组，创建一维数组的命令格式为 **array(a..b, lst)**。其中 **a..b** 给出了一维数组的下标范围，可以是任意的连续整数段；**lst** 以列表的形式给出列表中的部分或者全部元素值，列表中的元素具有 **index=entry** 或者**(index)=entry** 的形式，**index** 代

表元素的下标，**entry** 代表对应的元素值；用户也可以在列表 **lst** 中按照数组中的顺序给出其中前几个或全部元素值。数组的元素可以是除表达式序列、等式和不等式外任意的单个表达式，包括字符串、列表、集合和数组等，各个元素可以具有不同的数据类型。

如果数组的下标不是从 **1** 开始的，命令 **array** 的输出具有类似下面的格式：

```
> array(-1..1,[-1=a,1="a"]);
array(-1..1,[
   (-1) = a
   (0) = ?0
   (1) = "a"
   D
```

```
> A := array(-1..1,[1,2]);
A := array(-1..1,[
   (-1) = 1
   (0) = 2
   (1) = A1
   D
```

输出的开始行和结束行表示了输出对象的类型和结构，中间的每行输出对应数组的一个元素，具有**(index)=(value)**的形式；其中未知的元素用数组的名字加相应下标的形式表示，如果数组是未命名的，则用加下标的问号表示其中未知的元素。如果创建数组时元素都是未知的，则可以直接使用 **array(a..b)**的格式，看下面的例子：

```
> array(-1..1);
 array(-1..1,[
   (-1) = ?-1
   (0) = ?0
   (1) = ?1
   D
```

下标从 **1** 开始的一维数组代表 Maple V 中的一维向量，输出的结果和列表的形式类似，但并不属于列表数据类型，看下面的例子：

```
> array(1..4);
                    [?1, ?2, ?3, ?4]
```

```
> V := array(1..4,[1=4,3=2,2=3]);
 type(V,list);
                    V := [4,3,2,V4]
                          false
```

如果一维数组的元素都是已知的，并且下标从 1 开始，则可以直接使用 **array(lst)**的

格式。其中列表 **lst** 中的元素可以采用 **index=entry** 的形式，但 **index** 的值不能超过所给的元素个数；用户也可以在列表 **lst** 中按顺序直接给出数组中的元素值。看下面的例子：

```
> array([1=2,3=2,2=3]);
                          [2,3,2]
> array([2,3,2]);
                          [2,3,2]
```

2. 二维数组和多维数组的创建

Maple V 按照行的顺序存储多维数组中的元素，这和 C 语言等高级语言是类似的，读者可以参阅 C 语言手册理解这种存储结构，这里不再介绍。

创建二维数组和多维数组，可以采用与创建一维数组类似的命令格式。基本的格式为 **array(bounds, lst)**，其中参数 **bounds** 以序列的形式依次给出每一维下标的范围，各个下标的范围可以是任意的连续整数段；**lst** 以列表的形式给出了数组的部分或全部元素值，其中的元素可以具有**(index)=entry** 的形式，**index** 代表各个下标值构成的序列，**entry** 代表相应的元素值；用户也可以在列表 lst 中按照行的顺序直接给出数组每一维中的部分或全部元素，其中的每一维元素分别包含在一个子列表中。如果数组的元素值都是未知的，则可以使用 **array(bounds)**的格式。

创建二维数组或者多维数组时，命令 **array** 的输出结果和创建一维数组时类似，看下面的例子：

```
> array(-1..1,1..2,[[1,2],[3],[5]]);
array(-1..1,1..2,[
  (-1,1) = 1
  (-1,2) = 2
  (0,1) = 3
  (0,2) = ?0,2
  (1,1) = 5
  (1,2) = ?1,2
  D
```

如果二维数组或者多维数组的元素值都已知，并且每一维下标都从 **1** 开始，用户可以在命令 **array** 中缺省下标的范围序列，直接使用 **array(lst)**的命令格式。其中 **lst** 中的元素也可以有两种表示形式，但所给出的元素个数与数组的结构应该是匹配的，这里不再具体介绍，读者可以根据一维数组的情况类推。看下面的例子：

```
>d:=array([[[1,2],[3,4]],[[5,6],[7,8]],
                [[9,10],[11,12]]]);
d := array(1..3,1..2,1..2,[
  (1,1,1) = 1
  (1,1,2) = 2
  (1,2,1) = 3
  (1,2,2) = 4
  (2,1,1) = 5
  (2,1,2) = 6
  (2,2,1) = 7
  (2,2,2) = 8
  (3,1,1) = 9
  (3,1,2) = 10
  (3,2,1) = 11
  (3,2,2) = 12
  D
```

下标都从 **1** 开始的二维数组又可以称为矩阵，用命令 **array** 创建矩阵时的输出结果采用数学上的矩阵表示格式打印，具体将在第 11 章中介绍，下面给出一个简单的例子：

```
>M:=array(1..2,1..3,[[1,3],[2,4]]);
```

$$M := \begin{bmatrix} 1 & 3 & M_{1,3} \\ 2 & 4 & M_{2,3} \end{bmatrix}$$

和表一样，用户可以用命令 **indices(A)**和命令 **entries(A)**分别引用数组 **A** 中元素的下标和元素值，结果的输出是以下标列表或者元素值列表的序列形式显示。同样，在输出的列表序列中，用户不能控制数组元素的下标列表和对应元素列表的显示顺序，但是二者的位置是一一对应的。下面是这两个命令使用的例子

```
>a:=array([[1,2],[3,4],[5,6]]):
 indices(a);
                    [3,1],[2,1],[2,2],[1,2],[3,2],[1,1]
>entries(a);
                              [5],[3],[4],[2],[6],[1]
```

3. *特殊结构的数组*

特殊结构的数组通常是针对二维数组而言的，主要包括对称数组、反对称数组、对角数组、稀疏数组等。用户可以在命令 **array** 中加入相应的特征选项，创建特殊结构的数组，即使用 **array(CF, bounds, list)**的命令格式。其中 **CF** 代表特征选项，常用的特征选项包括 **symmetric**（对称）、**antisymmetric**（反对称）、**sparse**（稀疏）、 **diagonal**（对角）、**identity**（单位）等，用户也可以自定义其他的特征选项。这些特征选项主要用于创建特殊结构的

矩阵或者向量，在第 11 章中会有比较详细的介绍，这里给出几个简单的例子，其中包括一个反对称数组 **asa** 和一个对称矩阵 **sa**：

```
> asa:=array(antisymmetric,-2..2,-2..2);
```

$$asa := array(antisymmetric,-2..2,-2..2,[\ ])$$

```
> asa[-1,-1],asa[-1,2],asa[2,-1];
```

$$0, asa_{-1,2}, -asa_{-1,2}$$

```
> asa[1,-2]+asa[-2,1];
```

$$0$$

```
> sa:=array(symmetric,1..2,1..2,[[1,x],[x,x^2]]);
```

$$sa := \begin{bmatrix} 1 & x \\ x & x^2 \end{bmatrix}$$

4. 数组名

代表数组名的标识符可以是任意合法的标识符，但不能再用来构成其他索引标识符。与其他对象不同的是，数组名的值是其自身，而不是所代表的数组；如果要显示所代表的数组的结构或者其中的全部元素，可以使用命令 **eval(A)**或者 **op(A)**。这一点是和类别、集合等复合数据结构不同。看下面的例子，**V** 和 **M** 分别是前面定义的一维数组和二维数组：

```
> v;
  op(v);
```

$$V$$

$$[4,3,2,?_4]$$

```
> M;
  eval(M);
```

$$\mathrm{M}$$

$$\begin{bmatrix} 1 & 3 & ?_{1,3} \\ 2 & 4 & ?_{2,3} \end{bmatrix}$$

```
> nops(V),nops(M);
```

$$1,1$$

虽然数组名的值是其自身，但它代表的是数组，用户还是可以使用数组名引用数组中的元素，参看 7.3.1 节。

5. 数组的类型与结构分析

数组的类型是与数组的创建命令联系在一起的，命令 **array** 的完整格式为 **array(CF, bounds, lst)**，因此数组的类型名为 **array**，用户可以用命令 **type(A, array)**确认数组 **A** 的

类型是否是 **array**（数组）。由于 Maple V 把自身赋值给数组名，因此如果 **A** 为数组名，则命令 **whattype(A)**的返回值为 **symbol**。看下面的例子：

```
> da:=array(diagonal,0..1,0..1);
  da[1,0];
```

$$da := \text{array}(diagonal,0..1,0..1,[\ ])$$

$$0$$

```
> whattype(da),type(da,array);
```

$$symbol, true$$

```
> whattype(array(1..4,[1,0,0]));
```

$$array$$

如果一维数组的下标从 **1** 开始，则同时属于 **array** 和 **vector**（向量）两个数据类型；如果二维数组的下标都从 **1** 开始，则同时属于 **array** 和 **matrix**（矩阵）两个数据类型。具体将在第 11 章中介绍。

如果把创建数组的命令 **array(CF, bounds, lst)**看作数组表达式，则该表达式包含三个操作数；当特征选项 **CF** 缺省时，第一个操作数为空序列 **NULL**。由于数组名的特殊性，命令 **op** 和 **nops** 对数组名作用和对代表的数组表达式作用时，结果是不同的。读者可以分析下面的结果：

```
> op(da);
```

$$\text{array}(diagonal,0..1,0..1,[$$
$$(0,0) = ?_{0,0}$$
$$(0,1) = 0$$
$$(1,0) = 0$$
$$(1,1) = ?_{1,1}$$
$$D$$

```
> op(op(da));
  op(1,op(da)),op(3,op(da));
```

$$diagonal,0..1,0..1,[\ ]$$

$$diagonal,[\ ]$$

7.3.2　数组元素的引用、更新与运算

数组元素的引用规则和列表、集合元素的引用是类似的，用户可以通过对数组中的元素赋值，从而改变更新数组中的元素。使用命令 **map** 可以对数组中的元素执行共同的运算或者操作。本小节简单介绍数组元素的引用和更新。

1、数组元素的引用

在 Maple V 中，数组元素的引用采用数组名后跟方括号的方法，在方括号中依次给出

所引用元素的下标，各个下标之间用逗号隔开。这和 C 语言中用小括号引用数组的元素是不同的。看下面的例子：

```
> A:=array(-1..1,1..2,[[1,2],[3],[5]]):
  A[-1,1],A[0,1],A[1,2];
```

$$1,3,A_{1,2}$$

在引用数组的元素时，给定的下标个数必须与数组的维数匹配，下标的数值必须处于数组定义时规定的下标范围内，否则将引发错误。看下面的例子：

```
> A[0];
Error,array defined with 2 indices,used with 1 indices
> A[0,0];
Error,2nd index,0,smaller than lower array bound 1
```

通常情况下，用户不能一次引用数组的一行或者一列元素。但对于矩阵和向量来说，一般可以对其中的行列进行整体操作，具体将在第 11 章中介绍。

另外，用户可以使用命令 **indices(A)**和 **entries(A)**分别得到数组 **A** 中所有已知元素的下标和元素值，这两个命令的输出结果是一一对应的。看下面的例子：

```
> indices(A);
  entries(A);
```

$$[1,1],[0,1],[-1,1],[-1,2]$$
$$[5],[3],[1],[2]$$

2. *数组元素的更新*

当数组的结构确定后，用户不能再改变数组的结构，除非重新定义数组的结构。但是用户可以通过对数组中的元素赋值的办法来更新数组中的元素值。用户既可以对数组中未知的元素赋值，也可以改变数组中已知的元素，或者把已知的元素设为未知的元素。

用户可以引用数组中的某个元素，并对其进行赋值操作，赋值以后数组中的元素值随之发生改变，但用户不能把数组的元素赋值为序列。看下面的例子：

```
> op(A);
array(-1..1,1..2,[
   (-1,1) = 1
   (-1,2) = 2
   (0,1) = 3
   (0,2) = ?0,2
   (1,1) = 5
   (1,2) = ?1,2
   D
```

```
> A[0,2]:=4;
  A[1,1]:='A[1,1]';
```

$$A_{0,2} := 4$$

$$A_{1,1} := A_{1,1}$$

```
> op(A);
array(-1..1,1..2,[
   (-1,1)=1
   (-1,2)=2
   (0,1)=3
   (0,2)=4
   (1,1)=?1,1
   (1,2)=?1,2
   D
```

3. **map** 命令的使用

前面已多次用到了命令 **map**，命令 **map** 也可以对数组中的元素执行共同的操作或者运算。下面给出一个对数组元素执行微分运算的例子：

```
> F:=array([[x^3-x,ln(x^2-1)],[exp(x),cos(x^2)]]);
```

$$F := \begin{bmatrix} x^3 - x & \ln(x^2-1) \\ e^x & \cos(x^2) \end{bmatrix}$$

```
> map(diff,F,X);
```

$$\begin{bmatrix} 3x^2 - 1 & 2\dfrac{x}{x^2-1} \\ e^x & -2\sin(x^2)x \end{bmatrix}$$

用户也可以使用命令 **map** 对矩阵中的元素进行其他的共同操作或者运算，这里不再举例。

第8章 代数方程

方程是数学中一个重要的课题，方程可以分为代数方程和微分方程两大类。微分方程的求解将在第 12 章中介绍，本章主要讨论了代数方程的求解。Maple V 提供了许多用于方程求解的工具，不仅可以得到代数方程的分析解，而且可以得到代数方程的数值解。

本章分别介绍代数方程的分析求解和数值求解，主要集中于两个基本求解命令 **solve** 和 **fsolve** 的使用。

8.1 方程的分析解

方程的分析解是指没有任何近似的解。Maple V 提供的命令 **solve** 是一个功能强大的求解工具，用它可以得到多种方程的分析解，并可以用它来求解不等式和方程组。**RootOf** 表达式可以将复杂方程的解表示为统一的形式。

本节首先介绍 Maple V 中的方程与不等式的概念，接着介绍 **RootOf** 表达式的基本知识，最后分别介绍方程、不等式和方程组的求解。

8.1.1 代数方程与代数不等式

Maple V 中，等式和不等式是由六个关系运算符连接两个表达式构成的，代数方程和代数不等式分别是等式和不等式的一个子集。本节简单介绍等式和不等式的知识，并因此引出代数方程和代数不等式的概念。

1. 等式和方程

在 Maple V 中，等式是由等号连接两个表达式构成的，等式中只能含有一个等号，在所连接的两个表达式中不能再含有等号或者不等号。等式也是一种特殊的表达式，其类型名为 **equation** 或`=`，二者是等价的，用户可以对方程命名。等式有两个操作数，分别称为左操作数和右操作数，这两个操作数既可以是算术表达式，也可以是字符串、列表、集合等构成的其他表达式。用户可以使用命令 **type**、**whattype**、**op**、**nops**、**has** 等命令来分析等式的结构和数据类型，同时也可以用命令 *lhs* 和 *rhs* 引用等式的左操作数和右操作数。

下面给出几个等式的例子：

```
> e:= a = b;
```

$$e := a = b$$

```
> s_equ:="a"="b";
```

$$s\_equ := "a" = "b"$$

```
> whattype(e),type(s_equ,equation);
```

$$=, true$$

```
> nops(e),op(e);
```

$$2, a, b$$

```
> lhs(e),rhs(s_equ);
```

$$a, "b"$$

如果等式的两个操作数都是算术表达式，这样的等式就是数学意义上的方程。方程是本章的主要讨论对象。下面给出一个方程的例子：

```
> eqn:=sqrt(x^2-4*x+3)=x+1;
```

$$eqn := \sqrt{x^2 - 4x + 3} = x + 1$$

```
> type(eqn,`=`),nops(eqn),op(0,eqn);
```

$$true, 2, =$$

```
> lhs(eqn),rhs(eqn);
```

$$\sqrt{x^2 - 4x + 3}, x + 1$$

2. 不等式和代数不等式

和等式类似，不等式是由不等号连接两个表达式构成的，不等式中只能含有一个不等号，包括<>、<、<=、>、>=五个运算符，在所连接的两个表达式中不能再含有等号或者不等号。Maple V 把所有的不等式归类为三种类型，对应的类型名为：`<>`、`<`、`<=`，其中把含有 >、>= 的不等式分别转化为由<、<=构成的不等式，并分别归入`<`、`<=`两个类型中。同样，不等式有两个操作数，分别称为左操作数和右操作数，这两个操作数可以是算术表达式，也可以是字符串等。下面给出两个不等式的例子：

```
> ne1:=f(x)<g(x);
```

$$ne1 := f(x) < g(x)$$

```
> whattype(ne1),nops(ne1);
```

$$<, 2$$

```
> ne2:="b">="a";whattype(ne2);
```

$$ne2 := "a" \le "b"$$

$$<=$$

```
> lhs("b">="a"),lhs("b"<>"a");
```

$$"a", "b"$$

如果等式或者不等式出现在由逻辑运算符连接构成的布尔表达式中，或者作为命令

evalb 的参数，等式和不等式就被认为是关系表达式，其返回值为 **true**、**false** 或者 **FAIL**。如果 Maple V 不能确定不等式的返回值，将以不等式的形式返回。命令 **evalb** 和关系表达式主要用于编程，这里不再详细介绍，下面是使用命令 **evalb** 对关系表达式求值的例子：

```
> evalb(a=b),evalb("a"="b"),evalb([a]=[a]);
  evalb(a<b),evalb([a]<>[b]),evalb("b">="a");
```

$$false, false, true$$

$$a-b<0, true, true$$

由不等号连接两个算术表达式构成的不等式和数学意义上的不等式是相同的，这里称为代数不等式。代数不等式的求解和代数方程的求解是紧密相关的。Maple V 中提供的命令 **solve** 是个功能强大的求解工具，既可以用来求解代数方程或方程组，也可以用来求解代数不等式或代数不等式组。本章主要介绍代数方程的求解，并简单介绍代数不等式及代数不等式组的求解。

8.1.2 RootOf 表达式

RootOf 表达式是含有命令 **RootOf** 的表达式，它使得代数方程的所有的根都可以表示为统一的形式，可以认为是方程的根的隐式格式。命令 **allvalues** 可以把 **RootOf** 表达式转化为显式的根的形式。下面分别介绍命令 **RootOf** 和 **allvalues** 的使用及 **RootOf** 表达式的类型。

1. RootOf 命令

RootOf 命令除了可用来表示方程求解、特征值计算或者有理函数积分的计算结果外，本身也可以直接用来求解代数方程，并把求解结果表示为 **RootOf** 表达式的形式。

命令 **RootOf** 的基本格式为 **RootOf(expr, x)**，其中 **expr** 代表所求解的方程，如果 **expr** 是不具有方程形式的代数表达式，求解的方程默认为 **expr=0**。如果 **expr** 中只含有一个未知数，则可以省略命令中的参数 **x**。如果 **expr** 是由一次多项式构成的方程，命令 **RootOf** 直接给出方程的根；如果方程 **expr** 比较复杂，例如超越方程、高次方程等，命令 **RootOf** 把方程的所有的根表示为 **RootOf(f(_Z))**的形式，其中 **f(_Z)**是以全局变量_Z 为自变量的函数。看下面的例子：

```
> RootOf(a*x+b,x);
```

$$-\frac{a}{b}$$

```
> RootOf(x^2+1=0);
```

$$\mathrm{RootOf}(\_Z^2+1)$$

```
> RootOf(cos(x)-x,x);
```

$$\mathrm{RootOf}(\cos(\_Z)-\_Z)$$

同时，用户也可以在命令 **RootOf** 中指定方程求根的范围。命令的格式为 **RootOf(expr,**

x, c) 或者 **RootOf(expr, x, a..b)**，其中数字 **c** 限定命令在 **x=c** 附近对方程进行求解；如果 **a** 和 **b** 为实数，**a..b** 限定命令在区间**[a,b]**内对方程进行求解，要求 **a<b**；如果 **a** 和 **b** 为复数，**a..b** 限定命令在复平面上由复数 **a** 和 **b** 确定的矩形范围内进行求解，要求 **a** 必须是矩形的左下角对应的复数，**b** 是矩形的右上角对应的复数。看下面的例子：

```
> rt_c:=RootOf(x^2=2,x,-1.41);
```

$$rt\_c := \mathrm{RootOf}(\_Z^2 - 2, -1.41)$$

```
> rt_ab:=RootOf(x^3-2,x,-0.7-1.1*I..-0.6-1.0*I)
```

$$rt\_ab := \mathrm{RootOf}(\_Z^3 - 2, -.7 - 1.1I .. -.6 - 1.0I)$$

2. **RootOf** 表达式

Maple V 把含有命令 **RootOf** 的表达式通称为 **RootOf** 表达式，其中只调用一次 **RootOf** 命令的表达式称为简单 **RootOf** 表达式，由多个简单 **RootOf** 表达式的和、差、积、商或者其他运算构成的表达式称为复杂 **RootOf** 表达式。和其他的函数调用一样，简单 **RootOf** 表达式同时属于 **function** 和 **RootOf** 两个类型；复杂 **RootOf** 表达式则把简单 **RootOf** 作为其中的一个操作数，整个表达式的类型与表达式的结构有关。看下面的例子：

```
> rt_c;
  whattype(rt_c),type(rt_c,RootOf);
```

$$\mathrm{RootOf}(\_Z^2 - 2, -1.41)$$

$$function, true$$

Maple V 允许用户使用 **evalf**、**diff**、**series**、**simplify** 等命令对 **RootOf** 表达式进行相应的运算。用户也可以用命令 **alias** 定义 **RootOf** 表达式的缩写形式。看下面的例子：

```
> evalf(rt_c);
```

$$-1.414213562$$

```
> alias(beta=RootOf(y^3-x^2+1,y)):
  diff(beta+beta^2/(x-1),x);
```

$$\frac{2}{3}\frac{x}{\beta^2} + \frac{4}{3}\frac{x}{\beta(x-1)} - \frac{\beta^2}{(x-1)^2}$$

```
> int(%,x);
```

$$\beta + \frac{\beta^2}{x-1}$$

```
> RootOf(y*exp(y)-x,y);
```

$$\mathrm{RootOf}(\_Ze^{-Z} - x)$$

```
> series(%,x);
```

$$x - x^2 + \frac{3}{2}x^3 - \frac{8}{3}x^4 + \frac{125}{24}x^5 + O(x^6)$$

3. **allvalues** 命令

在实际应用中，总是希望得到方程的根的显式格式，用函数 **RootOf** 表示的求解结果有时不能满足用户的需求。命令 **allvalues** 能够识别 **RootOf** 表达式中的方程的根，并给出 **RootOf** 表达式的显式的计算结果。命令 **RootOf** 和 **allvalues** 结合使用通常可以得到代数方程的根的显式形式。

命令 **allvalues** 的基本格式为 **allvalues(RootOf_expr)**，其中 **RootOf_expr** 可以是单一的 **RootOf** 表达式，也可以是由 **RootOf** 表达式构成的列表、集合、表或数组等复合数据类型，但不能是表达式序列。命令 **allvalues** 总是试图给出 **RootOf** 表达式中方程的所有的根，对于次数不超过 **4** 的多项式方程通常可以得到准确的根。如果不能给出准确的求解结果且方程中没有其他的符号常量，命令 **allvalues** 采用数值的方法获得方程的数值解。看下面的例子：

```
> allvalues(2*RootOf(_Z^3-1)-1);
```

$$1, -2 + I\sqrt{3}, -2 - I\sqrt{3}$$

```
> allvalues(RootOf(cos(x)-x));
```

$$7390851332$$

另外，在使用命令 **allvalues** 时，命令 **RootOf** 中所限定的求根的范围通常是不起作用的。看下面的例子：

```
> rt_c;
  allvalues(rt_c);
```

$$\mathrm{RootOf}(\_Z^2 - 2, -1.41)$$

$$\sqrt{2}, -\sqrt{2}$$

如果参数 **RootOf_expr** 是复杂 **RootOf** 表达式，命令 **allvalues(RootOf_expr)**的返回结果是由其中的多个简单 **RootOf** 表达式所代表的方程的根交叉组合生成的表达式序列。在下面的例子中，由于方程 **x^2-1=0** 有 2 个根，方程 **x^4-1=0** 有 4 个根，调用 **allvalues** 函数时输出的结果是把这两个方程的根交叉组合后计算得到的，在结果序列中包含了 8 个值。

```
> e:=RootOf(_Z^2-1)+1/RootOf(_Z^4-1)^2;
```

$$e := \mathrm{RootOf}(\_Z^2 - 1) + \frac{1}{\mathrm{RootOf}(\_Z^4 - 1)^2}$$

```
> allvalues(e);
```

$$2, 2, 0, 0, 0, 0, -2, -2$$

下面是命令 **RootOf** 嵌套使用的复杂 **RootOf** 表达式，读者可以分析其结果：

```
> solve(x^4-1);
```

$$\text{RootOf}(-\text{RootOf}(\_Z^2+1)+\_Z^2)$$

```
> allvalues(%);
```

$$\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2},-\frac{1}{2}\sqrt{2}-\frac{1}{2}I\sqrt{2},\frac{1}{2}\sqrt{2}-\frac{1}{2}I\sqrt{2},-\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2}$$

在复杂 **RootOf** 表达式中，如果多个简单 **RootOf** 表达式求解的是同一方程，则这几个简单 **RootOf** 表达式是同步求解的，即不同的根之间不再交叉组合计算。同时，用户也可以在命令 **allvalues** 中加入 **independent** 选项，从而限定求解同一方程的简单 **RootOf** 表达式是独立的。比较下面的结果：

```
> f:=RootOf(_Z^2-1)-1/RootOf(_Z^2-1);
```

$$f:=\text{RootOf}(\_Z^2-1)-\frac{1}{\text{RootOf}(\_Z^2-1)}$$

```
> allvalues(f);
```

$$0,0$$

```
> allvalues(f,'independent');
```

$$0,2,-2,0$$

8.1.3　一元方程的分析解

含有一个未知数的方程称为一元方程。在很多情况下，命令 **solve** 可以给出一元方程的分析解。本小节将分别介绍基本的求解格式，复杂求解结果的表示形式，最后介绍周期函数方程的求解，以及以函数名为未知数的方程的求解。

1. 基本的命令格式

一元方程求解的基本命令格式为 **solve(eqn, var)**，命令返回的结果是由一元方程 **eqn** 的所有分析解构成的序列，**var** 代表了方程中的未知数。如果给出的参数 **eqn** 是不具有方程形式的算术表达式，则默认的方程为 **eqn=0**；如果方程 **eqn** 中只有一个未知数，则命令中的参数 **var** 可以缺省。下面给出几个简单的例子：

```
> solve(x^2-x=6);
```

$$-2,3$$

```
> solve(x^3+x^2+x+1);
```

$$-1,I,-I$$

```
> eq:=x^4-5*x^2+6*x=2;
```

$$eq:=x^4-5x^2+6x=2$$

```
> solve(eq,x);
```

$$-1+\sqrt{3},-1-\sqrt{3},1,1$$

用户也可以把未知数 **var** 用集合**{var}**的形式表示，这时的输出结果是具有**{var=value}**形式的集合的序列，每个集合代表了方程的一个解。看下面的例子：

```
> solve(eq,{x});
```

$$\{x=-1+\sqrt{3}\},\{x=-1-\sqrt{3}\},\{x=1\},\{x=1\}$$

在命令 **solve(eqn, var)**中，方程 **eqn** 中也可以包含其他的符号常量，求解的结果以这些符号常量为参数表示未知数 **var**。如果方程 **eqn** 中含有多个符号常量，并缺省了参数 **var**，命令将对其中的所有符号常量求解，输出结果以集合的形式表示。看下面的例子：

```
> solve(f=m*a,a);
```

$$\frac{f}{m}$$

```
> solve(f=m*a);
```

$$\{f=ma,a=a,m=m\}$$

如果方程中包含了开方运算，Maple V 将自动补充辅助变量，将方程转化为相应的方程组进行求解。通常，命令 **solve** 得到的解都能满足原方程，用户可以用命令 **subs** 来验证原方程是否成立。下面是几个包含开方运算的方程的例子：

```
> solve(x+sqrt(x)+x^(1/3)=3,x);
```

$$1$$

```
> solve(sqrt(x+1)-sqrt(x-1)=a,x);
```

$$\frac{1}{4}\frac{4+a^4}{a^2}$$

```
> eq:='eq':eq:=sqrt(x)+sqrt(x+1)=3
  solve(eq,{x});
```

$$\{x=\frac{16}{9}\}$$

```
> subs(%,eq);
```

$$\frac{1}{9}\sqrt{16}\sqrt{9}+\frac{1}{9}\sqrt{25}\sqrt{9}=3$$

```
> combine(%);
```

$$3=3$$

当方程中存在浮点数或者包含浮点运算时，命令 **solve** 首先将方程中的浮点数转化为近似的有理数即整数或者分数，然后求解转化后的方程，最后把求解的结果转化为浮点数。如果方程中同时包含浮点数和未知的参数，命令 **solve** 的求解结果通常以函数 **RootOf** 的形式表示。下面给出两个包含浮点数的方程求解的例子：

```
> solve(x^4+x+1.0,x);
```

$$-.7271360845-.4300142883I,-.7271360845+.4300142883I,$$
$$.7271360845-.9340992895I,.7271360845+.9340992895I$$

```
> solve(x^2-3*a*x+0.01,x);
```

$$\frac{1}{10}\,\mathrm{RootOf}(\_Z^2-30a\_Z+1)$$

```
> allvalues(%);
```

$$\frac{3}{2}a+\frac{1}{10}\sqrt{225a^2-1},\frac{3}{2}a-\frac{1}{10}\sqrt{225a^2-1}$$

如果方程 **eqn** 无解，或者命令 **solve** 不能找到方程的解，则命令 **solve** 的输出结果为空序列 **NULL**。如果命令 **solve** 在求解过程中漏掉了方程 **equation** 的某些解，Maple V 将把全局变量_**SolutionsMayBeLost** 赋值为 **true**。下面是一个简单的无解方程的例子：

```
> solve(x-3=x-2);
> whattype(%);
```

exprseq

2. 复杂的求解结果

命令 **solve** 能够给出多种方程的分析解。对于次数不超过 4 的一元方程来说，命令 **solve** 能够以显式的格式给出方程的分析解。但是符号计算的缺点就是结果的形式复杂，从下面的例子可以再次感觉到这一点。其中求解了一个三次方程 **eqn1**，并把求解的结果表示为列表的形式，分别打印了方程的三个根：

```
> eqn1:=x^3-5*x^2+x=1;
```

$$eqn1:=x^3-5x^2+x=1$$

```
> soll:=[solve(eqn1,x)]:nops(soll);
```

3

```
> soll[1];
```

$$\frac{1}{3}(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}+\frac{22}{3}\frac{1}{(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}}+\frac{5}{3}$$

```
> soll[2];
```

$$-\frac{1}{6}(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}-\frac{11}{3}\frac{1}{(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}}+\frac{5}{3}+\frac{1}{2}I\sqrt{3}\left[\frac{1}{3}(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}-\frac{22}{3}\frac{1}{(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}}\right]$$

```
> soll[3];
```

$$-\frac{1}{6}(116+6\sqrt{78}^{\left(\frac{1}{3}\right)}-\frac{11}{3}\frac{1}{(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}}+\frac{5}{3}-\frac{1}{2}I\sqrt{3}\left[\frac{1}{3}(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}-\frac{22}{3}\frac{1}{(116+6\sqrt{78})^{\left(\frac{1}{3}\right)}}\right]$$

可见方程的分析解形式是很复杂的，其中包括两个复数根。把复杂的分析解转化为浮点数的形式，就简单多了。看下面的结果：

```
> evalf(soll);
```

$$[4.835975919,.082012041+.4472779002\,I,.082012041-.4472779002\,I]$$

四次多项式方程的分析解虽然可以用方程的系数来表示，但这种表示通常非常复杂；在默认情况下，命令 **solve** 总是用 **RootOf** 表达式的形式表示四次多项式方程的解。用户可以设置全局变量**_EnvExplicit** 的值为 **true**，这时命令 **solve** 以显式格式给出四次多项式的所有分析解；如果**_EnvExplicit** 的值为 **false**（默认值），方程的所有非有理数解都以 **RootOf** 表达式的形式表示。下面以一个简单的四次方程为例，显示全局变量**_EnvExplicit** 的作用：

```
> _EnvExplicit:=true;
```

$$_EnvExplicit := true$$

```
> solve(x^4+1);
```

$$\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2},-\frac{1}{2}\sqrt{2}-\frac{1}{2}I\sqrt{2},\frac{1}{2}\sqrt{2}-\frac{1}{2}I\sqrt{2},-\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2}$$

```
> _EnvExplicit:=false;
```

$$_EnvExplicit := false$$

```
> solve(x^4+1);
```

$$\text{RootOf}(-\text{RootOf}(\_Z^2+1)+\_Z^2)$$

```
> allvalues(%);
```

$$\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2},-\frac{1}{2}\sqrt{2}-\frac{1}{2}I\sqrt{2},\frac{1}{2}\sqrt{2}-\frac{1}{2}I\sqrt{2},-\frac{1}{2}\sqrt{2}+\frac{1}{2}I\sqrt{2}$$

次数高于 4 的多项式方程或者复杂的超越方程，其分析解通常会更为复杂，命令 **solve** 自动以 **RootOf** 表达式的形式给出求解结果。用户可以用命令 **allvalues** 计算出方程的显式分析解，但有时得到的结果是非常复杂的，用户可以直接用 **evalf** 等命令将求解结果转化为近似值。下面给出了一个七次多项式方程的例子，求解的结果是两个 **RootOf** 表达式构成的序列，为了用命令 **allvalues** 得到显式解，把求解结果写作列表的形式，命令 **allvalues** 可得到方程的数值解：

```
> expr:=x^7-2*x^6-4*x^5-x^3+x^2+6*x+4:
  [solve(expr)];
```

$$[2\text{RootOf}(\_Z^2-\_Z-1),\text{RootOf}(\_Z^5-\_Z-1)]$$

```
> allvalues(%);
```

$$[1+\sqrt{5},-.7648844336-.3524715460I],[1+\sqrt{5},-.7648844336+.3524715460I],$$
$$[1+\sqrt{5},.1812324445-1.083954101I],[1+\sqrt{5},.1812324445+1.083954101I],$$
$$[1+\sqrt{5},1.167303978],[1-\sqrt{5},-.7648844336-.3524715460I],$$
$$[1-\sqrt{5},-.7648844336+.3524715460I],[1-\sqrt{5},.1812324445+1.083954101I],$$
$$[1-\sqrt{5},.1812324445+1.083954101I],[1-\sqrt{5},1.167303978]$$

3. 周期函数方程的解

周期函数方程指的是含有三角函数或者其他周期函数的方程。在很多情况下，周期函数方程的解有无穷多个，但命令 **solve** 通常只能给出其中有限的几个分析解。用户可以设置全局变量**_EnvAllSolutions** 的值为 **true**，命令 **solve** 将返回周期函数方程的所有分析解。这些分析解通常可以表述为统一的形式，其中需要用到用来代表特定整数的变量，Maple V 创建了一些系统变量，如**_Zn~**（**n=1**，**2**，…，下同）代表整数，**_NNn~**代表非负整数，**_Bn~**代表二进制数 **0** 和 **1**。这些系统变量不是全局变量，虽然这些系统变量代表了多个值，包含这些系统变量的表达式仍只作为一个表达式。下面给出几个周期函数方程求解的例子：

```
> eqsin:=sin(x)^2-2*sin(x)-3=0;
```

$$eqsin := \sin(x)^2-2\sin(x)-3=0$$

```
> s1:=[solve(eqsin)];
```

$$s1 := \left[-\frac{1}{2}\pi, \arcsin(3)\right]$$

```
> EnvAllSolutions:=true:
  s2:=[solve(eqsin)];
```

$$s2 := \left[-\frac{1}{2}\pi + 2\pi\_ZI\sim, \arcsin(3) - 2\arcsin(3)\_BI\sim + 2\pi\_ZI\sim + \pi\_BI\sim\right]$$

```
> nops(s2);
```

$$2$$

4. *以函数名为未知数的方程*

命令 **solve** 甚至可以求解以函数名为未知数的方程。命令的返回结果是未知函数的定义，通常是以过程的定义形式给出。下面给出一个这样的例子：

```
> solve(f(x)^2-3*f(x)+2*x,f);
```

$$\text{proc}(x)\text{RootOf}(\_Z\text{^}2 - 3*\_Z + 2*x)\text{end}$$

在实际应用中，用户可能会遇到用递归关系定义的函数，Maple V 提供了一个专门的命令 **rsolve** 用来获得这种函数的显式格式。关于命令 **rsolve** 的使用，读者可以参看相关的帮助信息，这里不再介绍。

8.1.4 不等式的求解

不等式的求解和方程的求解是相关的，命令 **solve** 也可以求解比较简单的不等式。下面给出两个简单的例子：

```
> solve(x^2>=0,x);
```

$$x$$

```
> solve((x-1+a)*(x-2+a)*(x-3+a)<0,{x});
```

$$\{x < 1 - a\}, \{2 - a < x, x < 3 - a\}$$

```
> assume(a,real);
  solve(a*x<b,{x});
```

$$\left\{\text{signum}(a\sim)x < \frac{\text{signum}(a\sim)b}{a\sim}\right\}$$

在上面的例子中，**signum** 是符号函数，当自变量为正数时函数值为 **1**，当自变量为

负数时函数值为-1，当自变量为 0 时函数值为 0。在不等式的求解中，经常使用函数 **RealRange** 和 **Open** 表示求解的结果。其中，**RealRange(a, b)**表示实数域内的闭区间**[a, b]**，如果要表示开区间，则使用函数 **Open** 来表示，即 **RealRange(Open(a), Open(b))**表示开区间**(a, b)**。看下面的例子：

```
> solve(x+1/x>0,x);
```

$$\text{RealRange}(\text{Open}(0),\infty)$$

```
> solve((x-2)*(x-3)<0,x);
```

$$\text{RealRange}(\text{Open}(2),\text{Open}(3))$$

同时，不等式的求解结果也经常采用 **RootOf** 表达式来表示，用户可以用命令 **allvalues** 对这样的结果求值，但不能使用命令 **evalf** 将结果转化为浮点数形式。看下面的例子：

```
> solve(x^2-2<0);
```

$$\text{RealRange}(\text{Open}(\text{RootOf}(\_Z^2-2,-1.414213562)),\ \text{Open}(\text{RootOf}(\_Z^2-2,1.414213562)))$$

```
> allvalues(%);
```

$$\text{RealRange}(\text{Open}(\sqrt{2}),\text{Open}(\sqrt{2})),\text{RealRange}(\text{Open}(-\sqrt{2}),\text{Open}(-\sqrt{2}))$$

8.1.5　方程组的求解

命令 **solve** 也可以用来求解方程组。基本格式为 **solve(eqns, vars)**，其中 **eqns** 代表以集合形式表示的方程组，**vars** 代表未知数构成的集合；当参数 **vars** 缺省时，命令 **solve** 对方程组中的所有未知数求解。求解的每组结果以集合的形式表示，集合中的元素具有 **var=value** 的形式，给出了各个未知数的解；如果方程组有多组解，求解结果以序列的形式给出得到的每一组解。下面分别介绍线性方程组和非线性方程组的求解。

1. 线性方程组

线性方程组是最简单的方程组，命令 **solve** 采用的基本方法为高斯消元法。同时，针对大型稀疏问题，在常规的高斯消元法基础上设计了一些特殊的运算法则，这些运算复杂对一般的稠密问题仍然是非常有效的。用户可以用命令 **subs** 验证所得到的解的合理性。下面是一个三元一次方程组的例子：

```
> eqns:={u+v+w=1,3*u+v=3,u-2*v-w=0};
```

$$eqns := \{u+v+w=1, 3u+v=3, u-2v-w=0\}$$

```
> sols:=solve(eqns);
```

$$sols := \{w=\frac{-2}{5}, v=\frac{3}{5}, u=\frac{4}{5}\}$$

```
> subs(sols,eqns);
```

$$\{1=1, 3=3, 0=0\}$$

某些超定的方程组可能是无解的，命令 **solve** 返回空序列 **NULL**。欠定的方程组有无穷多个解，命令 **solve** 选择其中的一个或多个未知数作为参数，其余的未知数用这些参数来表示。看下面的例子：

```
> eqns1:={2*x-2*y-2*z=-2,-x+y+3*z=0,-3*x+3*y-2*z=1};
```

$$eqns1 := \{2x-2y-2z=-2,-x+y+3z=0,-3x+3y-2z=1\}$$

```
> solve(eqns1,{x,y,z});
```

```
> eqns2:={-2*x+2*y-2*z=-2,3*x-2*y+2*z=2,x+3*y-3*z=-3}
```

$$eqns2 := \{3x-2y+2z=2, x+3y-3z=-3,-2x+2y-2z=-2\}$$

```
> solve(eqns2);
```

$$\{y=z-1, x=0, z=z\}$$

用户可以在方程组中同时给定未知数所满足的不等式，这时的求解结果同时满足方程组中的每个方程和不等式。比较下面的两个例子：

```
> solve({x^2*y^2=0,x-y=1});
```

$$\{y=-1, x=0\},\{y=-1, x=0\},\{x=1, y=0\},\{x=1, y=0\}$$

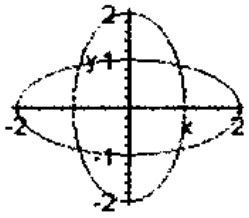
```
> solve({x^2*y^2=0,x-y=1,x<>0});
```

$$\{x=1, y=0\},\{x=1, y=0\}$$

2. 非线性方程组

非线性方程组的求解通常是比较复杂的。命令 **solve** 总是试图给出方程组的分析解，在必要的时候自动用 **RootOf** 表达式表示求解的结果。下面是一个非线性方程组的例子，求解两个椭圆的交点坐标。首先用 **plots** 函数库中的命令 **contourplot** 绘制了两个椭圆的图像：

```
> plots[contourplot]({4*x^2+y^2,x^2+4*y^2},
  x=-4..4,y=-4..4,contours=[4],numpoints=2000);
```



这两个椭圆有四个交点，下面是用命令 **solve** 的求解结果：

```
> solve({4*x^2+y^2=4,x^2+4*y^2=4},{x,y});
```

$$\{x=2\text{RootOf}(5\_Z^2-1), y=2\text{RootOf}(5\_Z^2-1)\}$$

命令 **allvalues** 可以把求解的结果表示为显式的格式，但由于上面的求解结果中含有两个相同的简单 **RootOf** 表达式，如果要得到四个交点的坐标，在 **allvaules** 命令中需使用 **independent** 选项。比较下面的结果：

```
> allvalues(%,independent);
```

$$\{y=\frac{2}{5}\sqrt{5},x=\frac{2}{5}\sqrt{5}\},\{y=-\frac{2}{5}\sqrt{5},x=\frac{2}{5}\sqrt{5}\},\{x=-\frac{2}{5}\sqrt{5},y=\frac{2}{5}\sqrt{5}\},$$
$$\{x=-\frac{2}{5}\sqrt{5},y=-\frac{2}{5}\sqrt{5}\}$$

```
> allvalues(%%);
```

$$\{x=\frac{2}{5}\sqrt{5},y=\frac{2}{5}\sqrt{5}\},\{x=-\frac{2}{5}\sqrt{5},y=-\frac{2}{5}\sqrt{5}\}$$

8.1.6　恒等式的成立条件

总是成立的等式称为恒等式。有些等式在某些特定的条件下可以成为恒等式，这些条件称为恒等式的成立条件。通常这些等式中含有需要确定的未知参数，确定了这些未知参数的值，等式就成为恒等式。命令 **solve** 可以确定恒等式中的未知参数。

命令的基本格式为 **solve(identity(eqn, x), vars)**，其中 **x** 是恒等式的变量，**vars** 是以集合形式表示的未知参数，**x** 不能是参数集合 **vars** 中的元素；等式 **eqn** 中必须同时包含恒等式的变量 **x** 和所求解的未知参数 **vars**，如果 **eqn** 是单一的算术表达式，则默认的等式为 **eqn=0**。返回的结果以集合形式表示恒等式成立时未知参数的值，集合中的元素具有 **var=value** 的形式；如果有多组参数值使恒等式成立，则返回的结果以集合的序列表示，每一个序列给出了一组满足恒等式的参数值；如果恒等式不可能成立，或者命令 **solve** 不能找到满足恒等式的参数值，返回的结果为空序列 **NULL**。

下面给出几个求解恒等式成立条件的例子：

```
> solve(identity(sin(x)=cos(a*x+b),x),{a,b});
```

$$\{a=1,b=-\frac{1}{2}\pi\},\{a=-1,b=\frac{1}{2}\pi\}$$

```
> solve(identity(3/(x^2+4*x+4)=A*(x+B)^P,x),{A,B,P});
```

$$\{B=2,A=3,P=-2\}$$

```
> solve(identity(y/x=a*(x+c)^b,x),{a,b,c});
```

$$\{b=-1,a=y,c=0\}$$

8.2　方程的数值解

用户可以使用命令 **solve** 并结合命令 **evalf** 或 **allvalues** 把方程的分析解转化为近似解。这种方法通常是比较麻烦的，Maple V 提供的命令 **fsolve** 是另一个功能很强的求解工具，

可以直接用来计算方程或者方程组的数值解。在数值求解的过程中，可以借助于 Maple V 的图像工具，使求解过程更为快捷有效。

本节首先以简单方程的数值解为例，介绍命令 **fsolve** 的基本格式；接着分别介绍复杂方程和方程组的数值求解；最后介绍方程求解和函数图像的结合。

8.2.1 简单方程的数值解

简单方程主要指的是具有多项式形式的方程，本小节以多项式方程的求解为例介绍命令 **fsolve** 的基本求解格式。

命令 **fsolve** 的基本格式为 **fsolve(eqn, var)**，其中 **eqn** 代表所求解的方程，在方程中不能含有其他的未知参数，且不能是不等式，如果不具有方程的形式，则默认方程为 **eqn=0**；**var** 代表要求解的未知数，通常也可以缺省。命令 **fsolve** 返回所得到的数值解构成的序列，数值解的精度是由全局变量 **Digits**（默认值为 **10**）的值决定的，用户可以改变 **Digits** 的值，从而改变求解结果的精度；如果未知数以**{var}**的集合形式给出，命令的返回结果是具有**{var=value}**形式的集合构成的序列，每个集合代表一个求解结果。

多项式方程是比较简单的方程，命令 **fsolve** 通常会给出多项式方程包括高次多项式方程的所有实数解。如果多项式方程是病态的，则命令 **fsolve** 有可能会漏掉方程的某些解。下面是两个多项式方程求解的例子：

```
> fsolve(x+2=x^2);
```

$$-1.000000000, 2.000000000$$

```
> Digits;
  Digits:=6;
```

$$10$$

$$Digits := 6$$

```
> poly:=23*x^5+105*x^4-10*x^2+17*x:
  fsolve(poly,{x});
```

$$\{x=-4.53617\},\{x=-.637181\},\{x=0\}$$

命令 **fsolve** 只是给出五次多项式方程 **poly=0** 的三个实数解。用户可以在命令中加入 **complex** 选项，则把求解的范围扩展到复数域，即采用 **fsolve(eqn, var, complex)**的格式，其中的参数 **var** 也不可缺省。看下面的例子：

```
> fsolve(poly,{x},complex);
```

$$\{x=-4.53617\},\{x=-.637181\},\{x=0\},\{x=.304066-.404062\,I\},\{x=.304066+.404062\,I\}$$

对多项式方程而言，用户可以在命令 **fsolve** 中加入 **maxsols=n** 选项，即采用 **fsolve(eqn,**

var, maxsols=n)的格式，命令返回最先得到的 **n** 个解；或者加入 **avoid=s** 选项，集合 **s** 中的元素具有 **var=value** 的形式，即采用 **fsolve(eqn, var, avoid=s)**的格式，命令 **fsolve** 在求解过程中排除集合 **s** 中列出的解的可能性。这两个选项和 **complex** 选项可以同时使用，三者的顺序没有限制，在命令中必须给出未知数 **var**；其中的 **complex** 选项和 **avoid** 选项对其他的方程也是有效的。看下面的例子：

```
> fsolve(poly,{x},maxsols=2);
```

$$\{x=-4.53617\},\{x=-.637181\}$$

```
> fsolve(poly,{x},complex,maxsols=4);
```

$$\{x=-4.53617\},\{x=-.637181\},\{x=0\},\{x=.304066-.404062\,I\}$$

```
> fsolve(poly,{x},avoid={x=0});
```

$$\{x=-4.53617\},\{x=-.637181\}$$

```
> fsolve(poly,{x},avoid={x=0},maxsols=3,complex);
```

$$\{x=-4.53617\},\{x=-.637181\},\{x=.304066-.404062\,I\}$$

如果只关心一定区间内方程的解的情况，用户可以在命令 **fsolve** 中指定方程求解的范围。命令的格式为 **fsolve(eqn, var, a..b)**或者 **fsolve(eqn, var=a..b)**，返回的解位于区间**[a, b]**内；如果在区间**[a, b]**内不存在方程 **eqn** 的解，则命令返回空序列 **NULL**。如果指定的范围表达式 **a..b** 代表复平面上的矩形区域，求解结果中将同时包含互为共轭复数的解。这个格式不仅对多项式方程有效，对其他的方程也是有效的。看下面的例子：

```
> fsolve(poly,{x=-5..-3});
```

$$\{x=-4.53617\}$$

```
> fsolve(poly,{x=1..2});
> fsolve(poly,{x=0.3+0.4*I..0.4+0.5*I},complex);
```

$$\{x=.304066-.404062\,I\},\{x=.304066+.404062\,I\}$$

方程求解范围的指定通常要借助于函数图像，关于方程求解与函数图像的结合，将在 8.2.4 节中介绍。

8.2.2　复杂方程的数值解

对于含有三角函数、指数函数以及对数函数等函数的超越方程或者其他类型的复杂方程来说，命令 **fsolve** 可以采用数值的方法在实数范围内进行求解，也可以在命令 **fsolve** 中加入 **complex** 选项，把方程求解的范围扩展为复数域；或者加入 **avoid** 选项排除指定的某些解；同时也可以指定方程求解的范围。命令 **fsolve** 可以求解多种形式的复杂方程，命令的返回结果通常只有一个，而且 **maxsols** 选项是不起作用的。看下面的例子：

```
> fsolve(sin(x)=cos(2*x));
                        -1.5707963217
> fsolve(sin(x)=cos(2*x),x,maxsols=3);
                        -1.5707963217

> fsolve(sin(x)=cos(2*x),x=2..4);
                         2.671993878
> fsolve(sin(x),x,avoid={x=0,x=Pi,x=-Pi},-10..10);
                                -6.283185307

> f:=10-(ln(v+(v^2-1)^(1/2))-ln(3+(3^2-1)^(1/2)));
                    f := 10 - ln(v + √(v^2 - 1)) + ln(3 + √8)
> fsolve(f,v);
                                64189.82535
```

在使用命令 **fsolve** 求解复杂方程的过程中，用户可以指定数值求解的初始值。命令的格式为 **fsolve(eqn, var=value)**，其中 **value** 代表指定的初始值。命令 **fsolve** 根据 **value** 的值选择合适的数值方法计算方程的解，所返回的计算结果通常位于初始值的附近。这种格式对多项式方程是无效的。看下面的例子：

```
> fsolve(sin(x),x=1);
                                    0
> fsolve(sin(x),x=1,avoid={x=0});
                             -3.141592654

> fsolve(x^2-1);
                        -1.000000000,1.
```

在复杂方程的求解过程中，借助于函数图像可以更准确地指定方程的求解范围或者求解的初始值，从而得到特定的解。

8.2.3 方程组的数值解

命令 **fsolve** 不仅能够得到单个方程的数值解，而且能够采用数值方法对方程组进行求解。命令的基本格式为 **fsolve({eqns}, {vars})**，要求方程组 **eqns** 中方程的个数和未知数的个数相同，其中不能含有其他的未知参数，也不能是不等式；参数 **vars** 给定所求解的未知数，通常也可以省略。

通常情况下，命令 **fsolve** 在实数范围内求解方程组；用户可以在命令中加入 **complex** 选项，把求解范围扩展到复数域；也可以指定方程组中某个未知数的初始值或者求解区间。命令通常只能得到方程组的一组解，并以集合的形式表示；**maxsols** 选项和 **avoid** 选项通

常是不起作用的。

看下面的例子：

```
> fsolve({4*x^2+y^2=4,x^2+4*y^2=4},{x,y});
                    {y = -.8944271910, x = .8944271910}
> fsolve({4*x^2+y^2=4,x^2+4*y^2=4},{x=0.8,y});
                    {x = .8944271910, y = -.8944271910}

> fsolve({4*x^2+y^2=4,x^2+4*y^2=4},{x=0.8,y=0.8});
                    {y = .8944271910, x = .8944271910}
> fsolve({x-y^2=1,x^2-y=1},{x,y},avoid={y=0});
                    {y = 0, x = 1.000000000}

> f:='f':f:=sin(x+y)-exp(x)*y=0:
  g:='g':g:=x^2-y=2:
  fsolve({f,g},{x,y},{x=-1..1,y=-2..0});
                    {x = -.6687012050, y = -1.552838698}
```

但在方程组的求解命令中，不能同时指定某个未知数的初始值和另一个未知数的求解区间，下面的命令将引发错误。

```
> fsolve({4*x^2+y^2=4,x^2+4*y^2=4},
  {x=0.8,y=0.7..0.9});
Error,(in fsolve) fsolve cannot solve on,y=.7...9
```

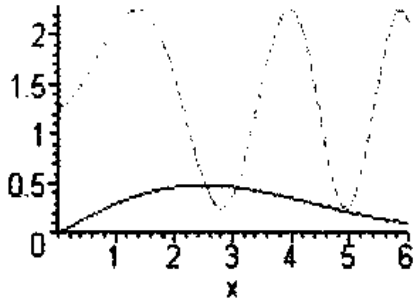
8.2.4　方程求解与函数图像的结合

Maple V 提供了强大的数据可视化功能，几乎可以绘出所有函数或表达式的图像。在方程求解的过程中，借助于函数图像可以直观地把握方程解的数量及对应的区间，对命令 **fsolve** 的求解结果进行检验。借助于函数图像，同时可以更准确地指定方程求解的范围或者求解的初始值，对于复杂方程的求解往往是非常有用的。

在绘制方程对应的函数图像时，可以把方程的左边和右边分别作为一个独立的函数，方程的求解相当于寻找这两个函数图像的交点坐标。如果方程的一边为 **0**，则相当于寻找函数图像与水平坐标轴的交点。根据绘制的函数图像，用户在命令 **fsolve** 中可以比较准确地限定方程求解的范围或者求解的初始值。

下面给出一个求解两个函数图像交点的例子。首先给出两个函数 **f(x)**和 **g(x)**的定义，然后在同一坐标系中绘出两个函数的图像：

```
> f:='f':g:='g':
  f:=x→exp(-(x/4)^2)*sin(x/Pi):
  g:=x→sin(x^(3/2))+5/4:
  plot({f(x),g(x)},x=0..6,linestyle=[1,4]);
```



这两个函数图像有两个交点，下面的命令可以用来求解两个交点的横坐标，但需要指定两个交点横坐标的区间范围或者求解的初始值，否则只能得到其中的一个解：

```
> fsolve(f(x)=g(x),x);
                              3.067531493
> fsolve(f(x)=g(x),x,2..3);
                              2.526235765
> fsolve(f(x)=g(x),x=3);
                              3.067531493
```

用户可以把两个横坐标的值代入其函数中，得到相应的函数值，即图像交点的纵坐标。另外，用户可以直接求解由两个函数构成的方程组，直接得到两个交点的横坐标和纵坐标，看下面的结果：

```
> fsolve({y=f(x),y=g(x)},{x=2.5,y});
                    {y = .4833283425, x = 2.526235765}
> fsolve({y=f(x),y=g(x)},{x=3,y});
                    {x = 3.067531493, y = .4601298630}
```

第 9 章　极限和微分

极限、微分和积分是数学分析的基础内容，是理论研究和工程应用的重要工具。Maple V 中提供了相关的命令，可以完成常见的极限和微积分运算。本章主要介绍极限和微分的计算，积分的计算将在第 10 章中介绍。

9.1　极限

极限是微积分中的一个基本概念，是数学分析和工程应用的一个重要工具。本节介绍极限的各种计算，包括基本的极限计算、左极限和右极限、复变函数的极限以及多元函数的极限几个部分。

9.1.1　基本的极限计算

计算极限的命令为 **limit**，其基本的格式为 **limit(f(x), x=a)**，用来计算 $\lim_{x\to a} f(x)$，其中 **f(x)**代表以 **x** 为自变量的算术表达式；**a** 可以是实数，也可以是复数。通常，命令 **limit** 在实数范围内计算函数 **f(x)**在 **x=a** 点的极限；如果 **a** 是复数，命令 **limit** 在复数范围内计算函数的极限，参看 9.1.3 节。

下面给出几个极限计算的例子：

```
> limit(x^sin(x),x=0);
```

$$1$$

```
> limit(1+x^2,x=-1);
```

$$2$$

```
> frac1:=(2*x^2+25*x+72)/(72-47*x-14*x^2):
  eval(frac1,x=-9/2);
Error,division by zero
```

```
> limit(frac1,x=-9/2);
```

$$\frac{7}{79}$$

在命令 **limit(f(x), x=a)**中，算术表达式 **f(x)**中可以包含其他的未知参数，极限点 **a** 也可以是未知的无值标识符，在 Maple V 可以顺利地完成这种符号极限的运算，但极限点 **a** 不能是自变量 **x**。看下面的例子：

```
> f:='f':a:='a':
  limit(f(x),x=a);
```

$$f(a)$$

```
> f:=x->x^3+5*x^2+1;
  val:='val':limit(f(x),x=val);
```

$$f := x \to x^3 + 5x^2 + 1$$

$$val^3 + 5val^2 + 1$$

```
> limit(a*x,x=infinity);
```

$$\mathrm{signum}(a)\infty$$

```
> limit(f(x),x=x);
Error, (in limit) invalid arguments
```

如果函数 **f(x)**在 **x=a** 处的极限值依赖于 **x** 趋近于 **a** 的方向，或者左右两边的极限不相等，命令返回 **undefined**；**undefined** 是 Maple V 的保护名，用户不能对其赋值。看下面的例子：

```
> limit(1/x,x=0);
```

undefined

```
> limit(abs(x)/x,x=0);
```

undefined

```
> whattype(%);
```

symbol

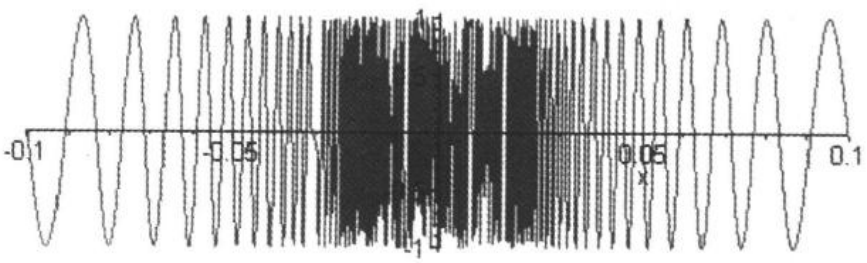
```
> undefined:=1;
Error,attempting to assign to `undefined` which is protected
```

如果函数 **f(x)**在 **x=a** 点的极限不存在，但在 **x=a** 点的邻域上函数值在有限的区间内变化，命令 **limit(f(x), x=a)**返回这一区间对应的范围表达式，这并不意味着函数 **f(x)**在 **x=a** 点的极限在区间中每一点取值的概率相同。下面给出函数 **f(x)=sin(Pi/x)**在 **x=0** 点的极限计算结果，并绘出函数在区间**[-0.1, 0.1]**上的函数图像，读者可以观察函数在 **x=0** 点附近的变化情况：

```
> limit(sin(Pi/x),x=0);
```

$$-1..1$$

```
> plot(sin(Pi/x),x=-0.1..0.1);
```



命令 **Limit** 是 **limit** 的惰性格式，命令 **Limit(f(x), x=a)**返回由极限符号表示的极限表

达式，极限符号以黑色显示，可以用命令 **value** 或 **evalf** 等得到最终的结果。

```
>Limit(sin(x),x=0);
```

$$\lim_{x \to 0} \sin(x)$$

```
>Limit(exp(x),x=infinity);
```

$$\lim_{x \to \infty} e^{x}$$

```
>value(%);
```

$$\infty$$

9.1.2　左极限与右极限

命令 **limit** 也可以计算左极限和右极限，命令的格式分别为 **limit(f, point, left)**和 **limit(f, point, right)**。下面的结果表明，$\frac{1}{x}$、$\frac{|x|}{x}$ 在 **x=0** 处的左右极限是不相等的：

```
>limit(1/x,x=0,right);
```

$$\infty$$

```
>limit(1/x,x=0,left);
```

$$-\infty$$

```
>limit(abs(x)/x,x=0,right);
```

$$1$$

```
>limit(abs(x)/x,x=0,left);
```

$$-1$$

命令 **limit** 也可以计算当自变量趋近于无穷大时的极限，其中 **limit(f(x), x=infinity)**用来计算左极限 $\lim_{x \to \infty} f(x)$，**limit(f(x), x=-infinity)**用来计算右极限 $\lim_{x \to (-\infty)} f(x)$，这两个极限都是单边极限，命令 **limit(f(x), x=infinity, right)**和 **limit(f(x), x=-infinity, left)**是不合法的。如果要计算实变函数 **f(x)**在 **x→±∞**时的极限值，需要采用 **limit(limit(f(x), x=infinity, real)**的格式。看下面的例子：

```
>limit(exp(-x),x=infinity,left);
```

$$0$$

```
>limit(exp(-x),x=infinity,right);
Error, (in limit) inconsistent direction with infinities
```

```
>t:=x*exp(x)/(1+exp(x));
```

$$t := \frac{x e^{x}}{1+e^{x}}$$

```
> limit(t,x=infinity);
```

$$\infty$$

```
> limit(t,x=-infinity);
```

$$0$$

```
> limit(t,x=infinity,real);
```

undefined

9.1.3 复变函数的极限

命令 **limit** 也可以用来计算复变函数的极限。在命令 **limit(f(x), x=a)**中，如果极限点 **a** 是复数，命令 **limit** 可用来在复数范围内计算复变函数 **f(x)**的极限，这时自变量 **x** 趋近于极限点 **a** 的方向为复平面上的任意方向。看下面的例子：

```
> limit(exp(z),z=1+1*I);
```

$$e^{(1+I)}$$

```
> evalf(%);
```

$$1.468693940 + 2.287355287\,I$$

```
> limit(exp(-z),z=a+b*I);
```

$$e^{(-a-I\,b)}$$

```
> limit(z^2-z+6,z=1+I);
```

$$5 + I$$

如果所给的极限点 **a** 不是复数，命令 **limit(f(x), x=a)**总是默认为在实数范围内计算函数的极限 $\lim_{x\to a} f(x)$；如果希望在复数范围内求函数 **f(x)**在 **x** 趋近于 **a** 时的值，需要在命令中加入 **complex** 选项，即使用 **limit(f(x), x=a, complex)**的格式。当 **a** 是无穷大或者是函数 **f(x)**的奇异点时，极限 $\lim_{x\to a} f(x)$ 在实数范围内可能是不存在的，但在复数范围内是存在的。看下面的例子：

```
> limit(1/x,x=0,complex);
```

$$\infty$$

```
> limit(1/x,x=0);
```

undefined

```
limit(-x,x=infinity,complex);
```

$$\infty$$

```
> limit(-x,x=infinity);
```

$$-\infty$$

```
> limit(1/(x-I-1),x=I+1);
```

$$undefined$$

```
> limit(1/(x-I-1),x=I+1,complex);
```

$$\infty$$

9.1.4　多元函数的极限

命令 **limit** 也可以用来求多元函数的极限。命令的基本格式为 **limit(f, {x=a, y=b,...})**，其中 **f** 代表多元函数，**x**、**y**、…分别代表函数的自变量，**a**、**b**、…代表所求的极限点。看下面的例子：

```
> limit(x^2+y^2-9,{x=0,y=3});
```

$$0$$

```
> limit((x^2-y^2)/(x^2+y^2),{x=0,y=0});
```

$$undefined$$

```
> limit(x+1/y,{x=0,y=infinity});
```

$$0$$

```
> limit(x*y,{x=0,y=infinity});
```

$$undefined$$

```
> limit((sin(x^2)-sin(y^2))/(x-y),{x=0,y=0});
```

$$\text{limit}\left(\frac{\sin(x^2)-\sin(y^2)}{x-y},\{x=0,y=0\}\right)$$

通常，**limit** 是在实数范围内计算函数的极限。用户也可以在命令中加入 **left** 或 **right** 选项，规定极限的方向；也可以加入 **real** 或 **complex** 选项，规定极限计算的范围。所加入的这些选项对函数的每个自变量都是起作用的。这些选项的使用和计算一元函数的极限时类似，这里不再举例介绍。

9.2　微分

微分是数学分析的基础内容，微分计算在科学计算和工程应用中经常用到。Maple V 提供了微分计算的命令，用户可以完成通常遇到的微分计算。本节将通过一阶微分、偏微分和高阶微分的计算以及隐函数的微分计算，介绍 Maple V 的三个常用的微分命令。

9.2.1　一阶微分

Maple V 提供了 **diff** 和 **D** 两个命令，都可以完成微分运算。本节以一元函数的微分计算为例，介绍这两个命令的基本使用，最后将对这两个命令进行比较。

1. diff

命令 **diff** 的常用格式为 **diff(f(x), x)**，用来计算函数 **f(x)**对 **x** 的一阶微分。**f(x)**也可以是与 **x** 无关的算术表达式，这时的微分结果为 **0**。看下面的例子：

```
> diff(cos(1),x);
```

$$0$$

```
> diff(0,x);
```

$$0$$

```
> diff(sin(x),x);
```

$$\cos(x)$$

```
> diff(sin(x),y);
```

$$0$$

```
> diff(x^5+x^3-x+1,x);
```

$$5x^4+3x^2-1$$

如果命令 **diff** 无法给出最终的计算结果，例如当 **f(x)**是未定义的函数时，输出结果用数学中的偏微分符号表示，其中的微分符号默认为蓝色显示。

```
> f:='f':diff(f(x),x);
```

$$\frac{\partial}{\partial x}\mathrm{f}(x)$$

命令 **Diff** 是 **diff** 的惰性格式，其输出结果是用偏微分符号表示的微分算式，其中的微分符号默认为以黑色显示。把命令 **Diff** 和 **diff** 结合使用，可以把计算过程写为等式。看下面的例子：

```
> Diff(exp(x),x)=diff(exp(x),x);
```

$$\frac{\partial}{\partial x}\mathrm{e}^x=\mathrm{e}^x$$

```
> Diff(x^2,x)=diff(x^2,x);
```

$$\frac{\partial}{\partial x}x^2=2x$$

```
> Diff(x*sin(cos(x)),x)=diff(x*sin(cos(x)),x);
```

$$\frac{\partial}{\partial x}x\sin(\cos(x))=\sin(\cos(x))-x\cos(\cos(x))\sin(x)$$

```
> Diff(ln(x/(1+x^2)),x,)=diff(ln(x/(1+x^2)),x);
```

$$\frac{\partial}{\partial x}\ln\left(\frac{x}{1+x^2}\right)=\frac{\left(\frac{1}{1+x^2}-2\frac{x^2}{(1+x^2)^2}\right)(1+x^2)}{\mathrm{x}}$$

Maple V 能够根据微分的性质自动完成各种复杂的微分运算。下面给出了两个未定义的函数 **f** 和 **g**，并用命令 **diff** 和 **Diff** 表示了和、差、积、商的微分运算的基本性质：

```
> f:='f':g:='g':
  Diff(c*f(x),x)=diff(c*f(x),x);
```

$$\frac{\partial}{\partial x}\mathrm{f}(x)c=\left(\frac{\partial}{\partial x}\mathrm{f}(x)\right)\mathrm{c}$$

```
> Diff(f(x)+g(x),x)=diff(f(x)+g(x),x);
```

$$\frac{\partial}{\partial x}(\mathrm{f}(x)+\mathrm{g}(x))=\left(\frac{\partial}{\partial x}\mathrm{f}(x)\right)+\left(\frac{\partial}{\partial x}\mathrm{g}(x)\right)$$

```
> Diff(f(x)*g(x),x)=diff(f(x)*g(x),x);
```

$$\frac{\partial}{\partial x}\mathrm{f}(x)\mathrm{g}(x)=\left(\frac{\partial}{\partial x}\mathrm{f}(x)\right)g(x)+\mathrm{f}(x)\left(\frac{\partial}{\partial x}\mathrm{g}(x)\right)$$

```
> Diff(1/g(x),x)=diff(1/g(x),x);
```

$$\frac{\partial}{\partial x}\frac{1}{\mathrm{g}(x)}=-\frac{\frac{\partial}{\partial x}\mathrm{g}(x)}{\mathrm{g}(x)^2}$$

同时，Maple V 能根据复合函数微分的链式法则完成复合函数的微分运算。下面是复合函数微分运算的例子，其中 **f(g(x))**和**(f@g)(x)**是等价的。

```
> Diff(sin(exp(x)),x)=diff(sin(exp(x)),x);
```

$$\frac{\partial}{\partial x}\sin(\mathrm{e}^x)=\cos(\mathrm{e}^x)\mathrm{e}^x$$

```
> Diff(f(g(x)),x)=diff(f(g(x)),x);
```

$$\frac{\partial}{\partial x}\mathrm{f}(\mathrm{g}(x))=\mathrm{D}(f)(\mathrm{g}(x))\left(\frac{\partial}{\partial x}\mathrm{g}(x)\right)$$

```
> Diff((f@g)(x)),x)=diff((f@g)(x),x);
```

$$\frac{\partial}{\partial x}\mathrm{f}(\mathrm{g}(x))=\mathrm{D}(f)(\mathrm{g}(x))\left(\frac{\partial}{\partial x}\mathrm{g}(x)\right)$$

2. D

命令 **D** 的基本格式为 **D(f)**，其中 f 代表函数算子，包括已定义的函数名如 **sin**、**cos** 等，未知的函数名 **f**、**g**，或者包含函数运算符“**->**”的函数定义表达式；命令返回的结果仍然是这样的函数算子。命令 **D** 直接对函数算子作用，表示了从一元函数 **f** 到一元函数 **D(f)**的变换关系。用户可以用 **D(f)(a)**表示函数 **f** 的微分在自变量为 **a** 时的值，其中 **a** 可以

是任意给定的数字、变量或表达式。看下面的例子：

```
> D(exp+cos^2+Pi+tan);
  %(Pi);
```

$$\exp - 2\sin\cos + 1 + \tan^2$$

$$e^{\pi} + 1$$

```
> f:=x->x^2;
  D(f);
  D(f)(x);
```

$$f := x \to x^2$$

$$x \to 2x$$

$$2x$$

如果 f 是个未确定的函数算子，命令 **D(f)**的输出结果仍用算子 **D** 来表示。

```
> f:='f':D(f);
```

$$D(f)$$

```
> D(f)(0); # the first derivative of f evaluated at 0
```

$$D(f)(0)$$

在下面的例子中，由于把 t 设为常量，因此 Maple V 把 cos(t)作为常量，这时命令 **D(cos(t))** 的结果为零。

```
> restart:constants:=constants,t;
```

$$constants := false, \gamma, \infty, true, Catalan, FAIL, \pi, t$$

```
> D(cos(t));
```

$$0$$

当 f、g 用来表示函数名时，**c*f**，**f+g**，**f*g** 和 **1/g** 都是合法的函数算子，下面用命令 **D** 给出了和、差、积、商微分运算的基本性质，形式上显得更简洁一些。

```
> f:='f':g:='g':
  'D(c*f)'=D(c*f);
```

$$D(cf) = D(c)f + cD(f)$$

```
>'D(f+g)'=D(f+g);
```

$$D(f+g) = D(f) + D(g)$$

```
>'D(f*g)'=D(f*g);
```

$$D(fg) = D(f)g + f\,D(g)$$

```
>'D(1/g)'=D(1/g);
```

$$D\left(\frac{1}{g}\right)=-\frac{D(g)}{g^2}$$

要用命令 **D** 表示复合函数的微分时，需要用 **f@g** 表示函数 **f** 与 **g** 的复合函数算子。下面给出复合函数计算的例子：

```
>'D(f(g))'=D(f(g));
```

$$D(f(g))=D(f(g))$$

```
>'D(f@g)'=D(f@g);
```

$$D(f@g)=(D(f))@gD(g)$$

```
>'D(sin@exp)(x)'=D(sin@exp)(x);
```

$$D(\sin@\exp)(x)=\cos(e^x)e^x$$

3. **D** 和 **diff** 的关系

和命令 **diff(f(x), x)**等价的形式为 **D(f)(x)**，都可用来计算函数 **f(x)**对自变量 **x** 的微分，二者之间存在这样的关系：**D(f)(x) =unapply(diff(f(x),x),x)**。看下面的例子：

```
>Diff(tan(x),x)=diff(tan(x),x);
```

$$\frac{\partial}{\partial x}\tan(x)=1+\tan(x)^2$$

```
>Diff(tan(x),x)=D(tan)(x);
```

$$\frac{\partial}{\partial x}\tan(x)=1+\tan(x)^2$$

命令 **D(f)**中的 **f** 始终作为函数算子，在 **D(f(x))**中，**f(x)**被看作是一个函数算子，和命令 **diff(f(x),x)**是完全不同的。比较下面的例子：

```
>D(sin(x));
```

$$D(\sin(x))$$

```
>diff(sin(x),x);
```

$$\cos(x)$$

```
>D(exp(-x^2))(x);
```

$$D\left(e^{(-x^2)}\right)(x)$$

```
>diff(exp(-x^2),x);
```

$$-2xe^{(-x^2)}$$

同样，如果 **f** 是函数算子，**D(f)**和 **diff(f, x)**的结果也是不同的。在命令 **diff(cos, x)**中，Maple V 认为 **x** 与函数算子 **cos** 没有关系。

```
> diff(cos,x);
```

$$0$$

```
> D(cos);
```

$$-\sin$$

9.2.2 偏微分与高阶微分

命令 **diff** 和 **D** 都可以用来计算函数的高阶微分和多元函数的偏微分。

1. diff

在计算函数的高阶微分和多元函数的偏微分时，命令 diff 的格式为 **diff(a, x1, x2, ..., xn)** 或者 **diff(a, [x1, x2, ..., xn])**，两个命令是等价的。其中 **a** 代表某一代数表达式，**x1**、**x2**、…、**xn** 代表微分的方向自变量，可以是相同的方向。例如对二元函数 **f(x, y)**而言，求二阶偏微分的命令为 **diff(f(x, y), x, y)**或者 **diff(f(x, y),[x, y])**，可以理解为命令 **diff** 的嵌套，即 **diff(f(x, y),x, y)=diff(diff(f(x, y),x), y)**。另外，命令 **diff(a, [])**返回算术表达式 **a** 自身。

下面给出偏微分和高阶微分计算的几个例子：

```
> f:='f':diff(f(x,y),[x,y]);
  diff(f(x,y),x,y);
```

$$\frac{\partial^2}{\partial y\,\partial x}\mathrm{f}(x,y)$$

$$\frac{\partial^2}{\partial y\,\partial x}\mathrm{f}(x,y)$$

```
> Diff(x*y*z,x,y)=diff(x*y*z,x,y);
```

$$\frac{\partial^2}{\partial y\,\partial x}xyz = z$$

```
> Diff(x^2+y^2-xy,[x,y])=diff(x^2+y^2-xy,[x,y]);
```

$$\frac{\partial^2}{\partial y\,\partial x}(x^2+y^2-xy)=0$$

```
> Diff(x^6+3*x^4-7*x^2,x,x,x,x)=
          diff(x^6+3*x^4-7*x^2,x,x,x,x);
```

$$\frac{\partial^4}{\partial x^4}(x^6+3x^4-7x^2)=360x^2+72$$

```
> Diff(x*y*z,[])=diff(x*y*z,[]);
```

$$\mathrm{Diff}(x\,y\,z,[\,]) = x\,y\,z$$

在计算函数的高阶微分或者高阶偏微分时，可以使用序列运算符**$**将命令表达为更简

洁的形式，下面的两个命令是等价的：

```
> diff(g(x,y),x,x,x,y,y);
```

$$\frac{\partial^5}{\partial y^2 \partial x^3} g(x,y)$$

```
> diff(g(x,y),x$3,y$2);
```

$$\frac{\partial^5}{\partial y^2 \partial x^3} g(x,y)$$

2. **D**

对多元函数 **f(x1,x2,...,xn)**而言，命令 **D[i](f)**用来计算函数 **f** 对第 **i** 个变元的偏微分。用户可以用命令 **D** 对多个变元同时求偏导数，例如命令 **D[i,j](f)**用来计算函数对第 **i** 和第 **j** 个变元的偏微分，且 **D[i,j](f)=D[i](D[j](f))**；而命令 **D[](f)**将返回函数算子本身。看下面的例子：

```
> g:='g':g:=(x,y)->x^5*y^5;
```

$$g := (x,y) \to x^5 y^5$$

```
> D[1](g),D[2](g);
```

$$(x,y) \to 5x^4 y^5, (x,y) \to 5x^5 y^4$$

```
> D[](g),D[](g)(x,y);
```

$$g, x^5 y^5$$

```
> D[1,2](g);
```

$$(x,y) \to 25x^4 y^4$$

如果函数 **f** 是一元函数，命令 **D[1](f)**和命令 **D(f)**是等价的；如果 **f** 是多元函数，命令 **D(f)**是不合法的，Maple V 将给出相应的错误信息。在下面的例子中，**g(x, y)**是前面定义的二元函数，因此 Maple V 给出了错误信息：

```
> D(g);
Error, (in D/procedure) function must be unary
```

命令 **D** 也可以用来求高阶微分。对多元函数算子 **f** 而言，命令 **D[in, jm](f)**分别对第 **i** 个变元求 **n** 阶微分，再对第 j 个变元求 m 阶微分。同时也可以把各个微分的变元按顺序列出，即命令 **D[1,1,2,2,2](f)**和命令 **D[1$2,2$3](f)**是等价的。下面是对二元函数 g(x, y)计算高阶偏微分的例子：

```
> D[1,1,2,2,2](g);
```

$$(x,y) \to 1200 x^3 y^2$$

```
> Diff(g(x,y),x$2,y$3)=D[1$2,2$3](g)(x,y);
```

$$\frac{\partial 5}{\partial y^3 \partial x^2} x^5 y^5 = 1200 x^3 y^2$$

如果对一元函数求高阶微分，则命令 **D[1$n](f)**和命令**(D@@n)(f)**是等价的，其中@@是复合函数运算符，Maple V 把命令 D 作为特殊的变换算子，可以和函数 f 构成复合的函数算子。看下面的例子：

```
> f:='f':(D@@n)(f);
```

$$(D^{(n)})(f)$$

```
> D[1$n](f);
```

$$D_{1\$n}(f)$$

```
> f:=x->x^7;
```

$$f := x \to x^7$$

```
> D[1$5](f);
```

$$x \to 2520x^2$$

```
> (D@@5)(f);
```

$$x \to 2520x^2$$

3. **D** 和 **diff** 的转换

命令 **convert** 可以实现命令 **D** 和 **diff** 的转换。命令 **convert(expr, D)**把用偏微分符号表示的微分表达式 **expr** 转换为用 **D** 表示的微分表达式；命令 **convert(expr, diff, x)**把用 **D** 表示的微分表达式 **expr** 转换为用偏微分符号表示的微分表达式。看下面的例子：

```
> df:=diff(y(x),x$2):
  convert(df,D);
```

$$(D^{(2)})(y)(x)$$

```
> dd:=D(y)(x)-a*D(z)(x):
  convert(dd,diff,x);
```

$$\left(\frac{\partial}{\partial x} y(x)\right) - a\left(\frac{\partial}{\partial x} z(x)\right)$$

4. 偏微分的可交换性

通常偏微分具有可交换性，即 $\frac{\partial^2}{\partial y \partial x} f(x,y) = \frac{\partial^2}{\partial y \partial x} f(x,y)$，因此，命令 **diff(f(x,y), x, y)** 和命令 **diff(f(x,y), y, x)**是等价的，命令 **D[i, j](f)**和命令 **D[j, i](f)**是等价的。

```
> D[i,j](f),D[j,i](f);
```

$$D_{i,j}(f), D_{i,j}(f)$$

```
> diff(f(x,y),x,y)-diff(f(x,y),y,x);
```

$$0$$

9.2.3　隐函数的微分

隐函数是指由包含因变量和自变量的方程隐式定义的函数。Maple V 提供了命令 **implicitdiff**，用来计算隐函数的微分。本小节首先将介绍简单的隐函数微分，接着介绍较为复杂的隐函数微分。

1. 简单的隐函数微分计算

当因变量和自变量的函数关系由一个方程确定时，隐函数的微分计算相对简单一些。对于由方程 **f** 定义的一元隐函数 **y(x)**来说，命令 **implicitdiff(f, y(x), x)**用来计算因变量 **y** 对自变量 **x** 的微分 **dy/dx**。其中 **f** 代表关于 **y** 和 **x** 的方程，如果算术表达式 **f** 不具有方程的形式，则默认为方程右边为 **0**。如果方程 **f** 中只包含因变量 **y** 和自变量 **x**，命令也可以简写为 **implicitdiff(f, y, x)**的形式。下面给出几个隐函数微分计算的例子：

```
> f:='f':f:=x^2+y^3=1;
```

$$f := x^2 + y^3 = 1$$

```
> implicitdiff(f,y,x);
```

$$-\frac{2}{3}\frac{x}{y^2}$$

```
> implicitdiff(f,x,y);
```

$$-\frac{3}{2}\frac{y^2}{x}$$

```
> f:='f':f:=a*x^3*y-2*y/z=z^2;
```

$$f := ax^3y - 2\frac{y}{z} = z^2$$

```
> implicitdiff(f,y(x),x);
```

$$-3\frac{ax^2yz}{ax^3z-2}$$

如果方程 **f** 中同时包含多个未知变量，则命令 **implicitdiff(f, y, x)**默认因变量 **y** 与其中的所有未知变量有关，即 **y** 被默认为多元函数，命令的计算结果代表因变量 **y** 对其中有关自变量 **x** 的偏微分。用户也可以指定因变量 **y** 与方程 **f** 中哪些变量有关，即指定函数 **y** 的自变量，命令的格式为 **implicitdiff(f, y(x1, ... , xj), xi)**，计算结果代表因变量 **y** 对自变量 **xi** 的偏微分；其中指定的自变量 **x1**、…、**xj** 也可以不在方程中出现，微分的方向 **xi** 应该是自变量序列中的成员。看下面的例子：

```
> implicitdiff(f,y(x,z),x);
```

$$-3\frac{ax^2yz}{ax^3z-2}$$

```
> implicitdiff(R=P*V/T,P,T);
```

$$\frac{P}{T}$$

```
> implicitdiff(R=P*V/T,P(V,T),T);
```

$$\frac{P}{T}$$

命令 **implicitdiff** 也可以用来计算隐函数的高阶偏微分，命令格式为 **implicitdiff(f, y, x1, ... , xk)**，其中 **x1**、…、**xk** 代表微分变元，微分变元序列不能包含在列表中，计算结果表示因变量 **y** 对这些微分变元的偏微分；如果对同一变元计算多阶微分，可以使用序列运算符“**$**”来规定微分的阶数。下面给出几个计算隐函数高阶微分的例子：

```
> implicitdiff(y=exp(x)+sin(x),y,x,x);
```

$$e^x-\sin(x)$$

```
> implicitdiff(y=exp(x)+sin(x),y,x$2);
```

$$e^x-\sin(x)$$

```
> implicitdiff(y=exp(x)*sin(z),y,x$2,z$3);
```

$$-e^x-\cos(z)$$

```
> implicitdiff(a*x^3*y-2*y/z=z^2,y(x,z),x,z);
```

$$6\frac{ax^2(2y-z^3)}{a^2x^6z^2-4ax^3z+4}$$

下面的命令是不合法的：

```
> implicitdiff(y=exp(x)+sin(x),y,[x,x]);
Error,implicitdiff expects its 3rd argument,u,to be of type {name,
set(name)},but received [x,x]
```

如果所计算的隐函数微分不存在，例如在方程 **f** 中不包含因变量 **y**，命令 **implicitdiff(f, y, x)**将返回系统常量 **FAIL**。

```
> implicitdiff(x^2*y+y^2=1,z,x);
```

FAIL

2. 复杂的隐函数微分

如果隐函数是由多个方程确定的，其微分运算相对复杂些。对于由多个方程同时确定的多个函数来说，**implicitdiff** 有下面四种复杂的命令格式：

- **implicitdiff({f1, ..., fm}, {y1, ..., yn}, u, x)**
- **implicitdiff({f1, ..., fm}, {y1, ..., yn}, u, x1 , ..., xk)**
- **implicitdiff({f1, ..., fm}, {y1, ..., yn}, {u1, ..., ur}, x)**
- **implicitdiff({f1, ..., fm}, {y1, ..., yn}, {u1,..., ur}, x1,...,xk)**

其中方程系{f1, ..., fm}定义了 n 个隐函数{y1, ..., yn}，y1、…、yn 可以是因变量名，也可以同时规定自变量序列，即具有 y(x1, ..., xm)的形式；u 是其中的一个隐函数，{u1, ... , ur}代表了其中的 r 个隐函数，u 或者 u1、…、ur 必须是{y1,..., yn}中的成员；x 或者 x1、…、xk 规定了微分的变元，如果在命令中规定了隐函数的自变量，则微分变元 x 或者 x1、…、xk 必须是自变量中的成员。第一种和第三种命令格式计算的是隐函数的一阶微分，第二种和第四种命令格式计算的是隐函数的高阶微分。

前两种命令格式的输出直接给出微分的计算结果。后两种命令的输出以集合的形式表示，集合中的元素具有 Dexpr = ans 的形式，Dexpr 代表由算子 D 表示的微分表达式，ans 代表相应的微分计算结果。用户也可以在命令中加入 notation=Diff 选项，规定集合中的元素以 diffexpr = ans 的形式表示，其中 diffexpr 代表由数学中的偏微分符号表示的微分表达式。缺省选项是 notation = D。

下面的例子是使用前两种命令形式的情况：

```
> f:='f':f:=a*sin(u*v)+b*cos(w*x)=c;
```

$$f := a\sin(u\,v)+b\cos(w\,x)=c$$

```
> g:='g':g:=u+v+w+x=z;
```

$$g := u+v+w+x=z$$

```
> h:='h':h:=u*v+w*x=z;
```

$$h := u\,v+w\,x=z$$

```
> implicitdiff({f,g,h},{u,v,w},u,z);
```

$$\frac{a\cos(u\,v)\,u\,x+b\sin(w\,x)\,x\,u-b\sin(w\,x)\,x-a\cos(u\,v)\,u}{x\,(-a\cos(u\,v)\,v+a\cos(u\,v)\,u+b\sin(w\,x)\,u-b\sin(w\,x)\,v)}$$

```
> implicitdiff({f,g,h},{u(x,z),v(x,z),w(x,z)},u,z);
```

$$\frac{-a\cos(u\,v)\,u+a\cos(u\,v)\,x\,u-b\sin(w\,x)\,x+b\,x\sin(w\,x)\,u}{(b\sin(w\,x)+a\cos(u\,v))\,(-v+u)\,x}$$

```
> implicitdiff({g,h},{u(x,z),v(x,z),w(x,z)},{u,v,w},z);
```

$$\{D_2(u)=\frac{xD_2(v)-uD_2(v)+1-x}{-v+x},\ D_2(w)=-\frac{-vD_2(v)+v+uD_2(v)-1}{-v+x},\ D_2(v)=D_2(v)\}$$

```
> implicitdiff({g,h},{u(x,z),v(x,z),w(x,z)},{u,v,w},z,
  notation=Diff);
```

$$\left\{\left(\frac{\partial}{\partial z}u\right)_x=\left(\frac{\partial}{\partial z}u\right)_x,\left(\frac{\partial}{\partial z}v\right)_x=-\frac{-\left(\frac{\partial}{\partial z}u\right)_x x+\left(\frac{\partial}{\partial z}u\right)_x v-1+x}{u-x},\right.$$

$$\left.\left(\frac{\partial}{\partial z}w\right)_x=\frac{\left(\frac{\partial}{\partial z}u\right)_x v-u\left(\frac{\partial}{\partial z}u\right)_x+u-1}{u-x}\right\}$$

如果方程组{f1, ... , fm}是关于因变量{y1, ... , yn}的超定方程，命令 **implicitdiff** 的输出结果可能为 **FALL**。

```
> implicitdiff({f,g,h},{u(x,z),v(x,z)},u,z);
                                    FAIL
```

关于命令 **implicitdiff** 的更多信息，读者可以用命令**?implicitdiff** 参看 Maple V 的在线帮助。

第10章　积　　分

积分作为数学分析的基础内容，在科学计算和工程技术领域中有着广泛的应用。积分主要包括定积分和不定积分，积分变换是积分的一个重要应用。积分函数是用积分定义的函数，Maple V 提供的标准积分函数不仅可以快捷地完成相应的积分运算，而且常常出现在其他复杂函数积分的计算结果中。

本章首先介绍不定积分和定积分的计算，接着介绍两种常用的积分技术，最后简单介绍积分变换和积分函数的定义和命令格式。

10.1　不定积分和定积分

不定积分和定积分的计算在高等数学中都是经常遇到的，Maple V 提供的积分命令 **int** 可以同时完成不定积分和定积分两种运算。本节将分别介绍不定积分和定积分的计算，并简单介绍多重积分的计算。

10.1.1　不定积分

1. 简单的不定积分

不定积分 $\int f(x)dx$ 的计算命令为 **int(f(x),x)**，其中 **f(x)**为被积函数，可以是任意的算术表达式；**x** 为积分变量，不能是常数或常量，也不能是包含运算符的表达式。如果被积函数 **f(x)**与积分变量 **x** 无关，则被认为是常数或常量。对于确定的函数表达式而言，命令 **int** 通常给出准确的积分结果；如果积分函数是未定义的，则输出用积分符号 $\int$ 和微分符号 d 表示的积分表达式，两个数学符号也是以蓝色显示。与命令 **int** 对应的惰性命令为 **Int**，命令 **Int** 的输出结果是用积分符号 $\int$ 和微分符号 d 表示的积分表达式，两个数学符号以黑色显示。

下面给出几个不定积分计算的简单例子：

```
> Int(sin(x),x)=int(sin(x),x);
```

$$\int \sin(x)dx = -\cos(x)$$

```
> Int(1/x,x)=int(1/x,x);
```

$$\int \frac{1}{x}dx = \ln(x)$$

```
> f:='f':f:=x^2*(1-x^3)^4:
  int(f,x);
```

$$\frac{1}{15}x^{15}-\frac{1}{3}x^{12}+\frac{2}{3}x^{9}-\frac{2}{3}x^{6}+\frac{1}{3}x^{3}$$

```
> Int(exp(-x^2),x)=int(exp(-x^2),x);
```

$$\int e^{(-x^2)}dx=\frac{1}{2}\sqrt{\pi}\,\mathrm{erf}(x)$$

```
> f:='f':int(f(x),x);
```

$$\int f(x)dx$$

```
> g:='g':g:=y->y^3*ln(y)^2:
  int(g(x),x);
```

$$\frac{1}{4}x^{4}\ln(x)^{2}-\frac{1}{8}x^{4}\ln(x)+\frac{1}{32}x^{4}$$

```
> int(g(y),x);
```

$$y^{3}\ln(y)^{2}x$$

在 Maple V 中，函数名和代表算术表达式的标识符是不同的。如果标识符 **f** 不是函数名，**f(x)**就将构造新的表达式，命令 **int(f(x), x)**无法给出积分结果；如果要对 **f** 代表的算术表达式积分，直接使用命令 **int(f, x)**即可。看下面的例子：

```
> f:='f':f:=sin(x)*cos(x)^2:
> int(f,x);
```

$$-\frac{1}{3}\cos(x)^{3}$$

```
> int(f(x),x);
```

$$\int \sin(x)(x)\cos(x)(x)^{2}dx$$

2. 复杂的不定积分

积分是微分的逆过程，有些积分运算比较复杂或者难以找到原函数，命令 **int** 有时可能无法得到积分的结果。但是，Maple V 还是能够计算出通常遇到的很多种复杂函数的积分。

下面给出三个较为复杂的积分运算的例子，用户可以使用命令 **simplify** 等化简积分的结果：

```
>#复杂积分运算之一
Int(sin(x)*ln(x),x)=int(sin(x)*ln(x),x);
```

$$\int \sin(x)\ln(x)dx = -\cos(x)\ln(x) + \mathrm{Ci}(x) + \frac{1}{2}I\pi - \frac{1}{2}I\pi\,\mathrm{csgn}(x)$$

```
>#复杂积分运算之二
Int(1/(sin(x)^2+2),x)=int(1/(sin(x)^2+2),x);
```

$$\int \frac{1}{\sin(x)^2+2}dx = \frac{\arctan\left(2\dfrac{\tan\left(\frac{1}{2}x\right)}{\sqrt{6}+\sqrt{2}}\right)}{\sqrt{6}+\sqrt{2}} + \frac{1}{3}\frac{\arctan\left(2\dfrac{\tan\left(\frac{1}{2}x\right)}{\sqrt{6}+\sqrt{2}}\right)\sqrt{3}}{\sqrt{6}+\sqrt{2}} - \frac{1}{3}\frac{\arctan\left(2\dfrac{\tan\left(\frac{1}{2}x\right)}{\sqrt{6}-\sqrt{2}}\right)\sqrt{3}}{\sqrt{6}-\sqrt{2}} + \frac{\arctan\left(2\dfrac{\tan\left(\frac{1}{2}x\right)}{\sqrt{6}-\sqrt{2}}\right)}{\sqrt{6}-\sqrt{2}}$$

```
>simplify(%);
```

$$-\int \frac{1}{-3+\cos(x)^2}dx = \frac{1}{3}\frac{\sqrt{3}\left(\arctan\left(\dfrac{\tan\left(\frac{1}{2}x\right)\sqrt{2}}{\sqrt{3}+1}\right)+\arctan\left(\dfrac{\tan\left(\frac{1}{2}x\right)\sqrt{2}}{\sqrt{3}-1}\right)\right)\sqrt{2}}{(\sqrt{3}+1)(\sqrt{3}-1)}$$

```
>#复杂积分运算之三
Int(x/(sin(x)+2),x)=int(x/(sin(x)+2),x);
```

$$\int \frac{x}{\sin(x)+2}dx = -\frac{1}{3}\ln(-2I+I\sqrt{3})\ln\left(-\frac{I(e^{(Ix)}+2I-I\sqrt{3})}{-2+\sqrt{3}}\right)\sqrt{3} + \frac{1}{3}\ln(-2I-I\sqrt{3})\ln\left(\frac{I(e^{(Ix)}+2I+I\sqrt{3})}{2+\sqrt{3}}\right)\sqrt{3} + \frac{1}{3}\mathrm{dilog}\left(-\frac{Ie^{(Ix)}}{-2+\sqrt{3}}\right)\sqrt{3} - \frac{1}{3}\mathrm{dilog}\left(\frac{Ie^{(Ix)}}{2+\sqrt{3}}\right)\sqrt{3}$$

在上面的例子中，积分结果中出现的函数 **Ci(x)**代表余弦积分函数，**dilog(x)**代表对数积分函数，余弦积分函数和对数积分函数的定义请参看 10.3.2 节。**csgn(x)**是复数平面上的符号函数，其定义如下：

$$csgn(z)=\begin{cases}1, \mathrm{Re}(z)>0 \text{ 或者 } \mathrm{Re}(z)=0 \text{ 且 } \mathrm{Im}(z)>0\\ -1, \mathrm{Re}(z)<0 \text{ 或者 } \mathrm{Re}(z)=0 \text{ 且 } \mathrm{Im}(z)<0\end{cases}$$

3. 被积函数的分离

Maple V 在 **student** 函数库中提供了命令 **integrand**，用户可以从用积分符号表示的积分表达式中分离出被积函数。命令的格式为 **integrand(expr)**，**expr** 代表任意的积分表达式或者多个积分表达式构成的算术表达式。通常情况下，命令 **intergrand(expr)**直接返回积分表达式中的积分函数；如果 **expr** 中包含多个积分表达式，则命令 **integrand(expr)**返回所有被积函数的集合；如果 **expr** 中不包含积分命令或积分符号，命令 **integrand(expr)** 返回空集。注意，在使用命令 **integrand** 之前要加载 **student** 函数库。

下面给出几个简单的例子：

```
> with(student):f:='f':
  integrand(Int(f(x),x));
warning,new definition for D
```

$$f(x)$$

```
> Int(-16/9*sin(3*x)*exp(-4*x),x);
```

$$\int -\frac{16}{9}\sin(3x)e^{(-4x)}dx$$

```
> integrand(%);
```

$$-\frac{16}{9}\sin(3x)e^{(-4x)}$$

```
> integrand(x+y);
```

$$\{\ \}$$

```
> integrand(Doubleint(f(x,y),x,y)+Int(g(x),x));
```

$$\{g(x), f(x,y)\}$$

Doubleint 是双重积分命令，读者可以参看 10.1.3 节的内容。

10.1.2 定积分

命令 **int(f(x),x=a..b)**用来计算定积分 $\int_a^b f(x)dx$，其中参数 **f(x)**和 **x** 与不定积分的命令 **int(f(x), x)**的规定是一样的，**a**、**b** 给出了积分的上限和下限。惰性命令 **Int** 也有类似的命令格式。命令 **int** 通常可以得到准确的积分结果；如果被积函数或者积分限中包含浮点数，则积分的结果用浮点数表示。

1. 简单的定积分计算

如果积分区间是数字常量，被积函数在积分区间上是确定的连续函数，被积函数在整

个积分区间上没有不连续的点，这种积分叫定积分。定积分的计算通常是比较简单的，下面给出几个简单的例子：

```
> f:='f':f:=x^3*exp(-4*x):
  int(f,x=1..2)
```

$$-\frac{379}{128}e^{(-8)}+\frac{71}{128}e^{(-4)}$$

```
> g:='g':g:=x→exp(2*x)*sin(2*x)^2:
  int(g(x),x=Pi..2*Pi);
```

$$\frac{1}{5}e^{(4\pi)}-\frac{1}{5}e^{(-2\pi)}$$

```
> Int(sin(x),x=0..Pi)=int(sin(x),x=0..Pi);
```

$$\int_0^{\pi}\sin(x)dx=2$$

值得注意的是，Maple V 对负数开方奇数次时不能给出实根，下面的结果是不可接受的，因为在实数范围内的积分竟然出现了复数：

```
> int(1/x^(2/3),x=-8..-1);
```

$$\frac{3}{2}+\frac{3}{2}I\sqrt{3}-\frac{3}{2}8^{\left(\frac{1}{3}\right)}-\frac{3}{2}I8^{\left(\frac{1}{3}\right)}\sqrt{3}$$

这种情况下，如果要得到合理的计算结果，需要使用命令 **surd**。看下面的结果：

```
> int(1/surd(x,3)^2,x=-8..-1);
```

$$38^{\left(\frac{1}{3}\right)}-3$$

```
> simplify(%);
```

$$3$$

在计算无穷积分时，用 **infinity** 表示正无穷大•，**-infinity** 表示负无穷大-•。除了是数字常量或者无穷大之外，定积分的积分限也可以是未知的符号表达式，或者任意的不含积分变量的算术表达式。下面给出两个这样的例子：

```
> Int(exp(-x),x=0..infinity)=
            int(exp(-x),x=0..infinity);
```

$$\int_0^{\infty}e^{(-x)}dx=1$$

```
> Int(exp(-x),x=-infinity..infinity)=
              int(exp(-x),x=-infinity..infinity);
```

$$\int_{-\infty}^{\infty} e^{(-x)}dx = \infty$$

```
> Int(sin(x),x=t-1..t^2+1)=int(sin(x),x=t-1..t^2+1);
```

$$\int_{t-1}^{t^2+1} \sin(x)dx = -\cos(t^2+1)+\cos(t-1)$$

```
> Int(-x*exp(-x^2),x=0..y)=int(-x*exp(-x^2),x=0..y);
```

$$\int_0^y -xe^{(-x^2)}dx = \frac{1}{2}e^{(-y^2)} - \frac{1}{2}$$

定积分的上限值也可以比其下限值小；很明显，当定积分的积分限互换后，计算结果的符号也随着改变。看下面的例子：

```
> Int(exp(-x),x=0..infinity)=
              int(exp(-x),x=0..infinity);
```

$$\int_0^{\infty} e^{(-x)}dx = 1$$

```
> Int(exp(-x),x=infinity..0)=
              int(exp(-x),x=infinity..0);
```

$$\int_{\infty}^{0} e^{(-x)}dx = -1$$

下面的命令是错误的：

```
> Int(exp(-x),x=infinity..-infinity)=
              int(exp(-x),x=infinity..-infinity);
Error, (in limit) inconsistent direction with infinities
```

2. 被积函数的连续性

被积函数的连续性是影响积分结果和能否积分的重要因素。默认情况下，Maple V 将自动检查被积函数的连续性。一种较为特殊的情况是，不连续点刚好是积分的上限或下限，这时 Maple V 将自动利用极限概念完成积分运算。读者可以分析下面的计算结果：

```
> Int(1/(x*sqrt(x^2-1)),x)=int(1/(x*sqrt(x^2-1)),x);
```

$$\int \frac{1}{x\sqrt{x^2-1}}dx = -\arctan\left(\frac{1}{\sqrt{x^2-1}}\right)$$

```
> limit(f,x=infinity);
  limit(f,x=1,right);
```

$$0$$

$$-\frac{1}{2}\pi$$

```
> Int(1/(x*sqrt(x^2-1)),x=1..infinity)
        =int(1/(x*sqrt(x^2-1)),x=1..infinity);
```

$$\int_1^{\infty} \frac{1}{x\sqrt{x^2-1}}dx = \frac{1}{2}\pi$$

如果被积函数在积分区间内存在不连续点，Maple V 将以该不连续点为界把积分区间分为两部分，定积分变成两部分积分的和，被积函数在两个子区间内都是连续的。如果存在多个不连续点，按同样的方法可把定积分分成多个子区间上的积分的和。

以被积函数$\frac{1}{x^2}$为例，在积分区间[-1,1]上 x=0 为不连续点，

$$\int_{-1}^{1} \frac{1}{x^2}dx \neq -\frac{1}{x}\Big|_{-1}^{1} = -2$$

Maple V 把该定积分分成两部分来计算，

$$\int_{-1}^{1} \frac{1}{x^2}dx = \int_{0}^{1} \frac{1}{x^2}dx + \int_{-1}^{0} \frac{1}{x^2}dx$$

读者可以分析下面的计算结果：

```
> Int(1/x^2,x=-1..0)=int(1/x^2,x=-1..0);
```

$$\int_{-1}^{0} \frac{1}{x^2}dx = \infty$$

```
> Int(1/x^2,x=0..1)=int(1/x^2,x=0..1);
```

$$\int_{0}^{1} \frac{1}{x^2}dx = \infty$$

```
> Int(1/x^2,x=-1..1)=int(1/x^2,x=-1..1);
```

$$\int_{-1}^{1}\frac{1}{x^2}dx=\infty$$

用户可以强制 Maple V 把被积函数作为连续函数对待，只需在命令 **int** 中加入选项 **continuous** 即可，命令的格式为 **int(f(x), x, continuous)**。看下面的结果，与上面的计算结果是不同的：

```
> Int(1/x^2,x=-1..1)=
        int(1/x^2,x=-1..1,continuous);
```

$$\int_{-1}^{1}\frac{1}{x^2}dx=-2$$

也可以在命令 **int** 中加入 **CauchyPrincipalValue** 选项，这时 Maple V 认为每个不连续点的左极限和右极限相等，即认为自变量从左右两边接近不连续点的速率相同。这意味着正负无穷大的极限可以相互抵消。看下面的例子：

```
> int(1/x^3,x=-1..2;
```

$$\int_{-1}^{2}\frac{1}{x^3}dx$$

```
> int(1/x^3,x=-1..0);
  int(1/x^3,x=0..2;
```

$$-\infty$$

$$\infty$$

```
> int(1/x^3,x=-1..2,'CauchyPrincipalValue');
```

$$\frac{3}{8}$$

选项 **continuous** 和 **CauchyPrincipalValue** 的含义是不同的，尽管有时使用两个选项得到的结果相同。下面给出了使用两个选项结果不同的例子：

```
> Int(1/abs(x),x)=int(1/abs(x),x);
```

$$\int\frac{1}{|x|}dx=\frac{x\ln(x)}{|x|}$$

```
> int(1/abs(x),x=-1..1,'CauchyPrincipalValue');
```

$$\infty$$

```
> int(1/abs(x),x=-1..1,continuous);
```

$$I\pi$$

在上面的例子中，当使用 **continuous** 选项时，Maple V 把-1 和 1 分别代入积分后的

函数计算，并把计算域扩展为复数空间，这时 **ln1=0**，**ln(-1)=** πi，因此 Maple V 给出的输出结果为 $I\pi$。

3. 定积分的数值计算

用户可以使用命令 **evalf(int(f, x=a..b))**将命令 **int** 的计算结果转化为浮点数的近似形式。看下面的例子：

```
> int(1/abs(x),x=-1..1,continuous):evalf(%);
```

$$3.141592654\,I$$

```
> int(exp(-x^2)*cos(x^3),x=0..Pi);
```

$$\int_0^{\pi} e^{(-x^2)}\cos(x^3)\,dx$$

```
> evalf(%);
```

$$.6940531229$$

更为有用的是，命令 **evalf(Int(f, x=a..b))**或者 **evalf(Int(f, x=a..b, digits, flag))**采用数值积分的方法，直接得到积分的数值结果。其中 **digits** 规定数值积分的精度，缺省时采用系统变量 **Digits** 的值；**flag** 规定数值积分的方法，在缺省时，采用 Clenshaw-Curtis 积分法。数值积分结果的相对误差通常小于 **0.5*10^(1-Digits)**或 **0.5*10^(1-digits)**。看下面的例子：

```
> evalf(Int(exp(-x^2)*cos(x^3),x=0..Pi^(1/3)));
```

$$.7015656956$$

```
> evalf(Int(sin(x)*ln(x),x=0..1));
```

$$-.2398117420$$

```
> Int(exp(v-v^2/2)/(1+1/2*exp(v)),v=
  -infinity..infinity);
```

$$\int_{-\infty}^{\infty} \frac{e^{\left(v-\frac{1}{2}v^2\right)}}{1+\frac{1}{2}e^{v}}\,dv$$

```
> evalf(%,20);
```

$$1.80557706229704967 88$$

当 **flag** 选项缺省时，如果在积分区间内或积分区间附近存在奇异点，Maple V 将采用自适应的双指数积分法或者符号分析中的相关技术完成数值积分。用户可以在 **flag** 选项中规定命令 **evalf** 所采用的积分方法。**flag** 选项有三种取值：

- **_CCquad** 规定只使用 **Clenshaw-Curtis** 积分法，且不采用奇异点处理技术；
- **_Dexp** 规定只使用自适应的双指数积分法，且不采用 Clenshaw-Curtis 积分法和奇异点处理技术；

- **_NCrule** 规定只采用自适应的 Newton-Cotes 规则即 quanc8 积分法，这种方法是固定精度的，当精度要求很高时不再适用。

看下面的例子：

```
> evalf(Int(1/GAMMA(x),x=0..2,20,_NCrule));
```

$$1.62637839868614061 45$$

10.1.3 多重积分

双重积分和多重积分可以通过命令 **int** 或 **Int** 的嵌套来实现。看下面的例子：

```
> Int(Int(sin(x)*y,x),y)=int(int(sin(x)*y,x),y);
```

$$\iint \sin(x)\,y\,dx\,dy = -\frac{1}{2}y^2\cos(x)$$

```
> Int(Int(Int(x*y*Z,x),y),z)=int(int(int(x*y*z,x),y),z);
```

$$\iiint x\,y\,z\,dx\,dy\,dz = \frac{1}{8}x^2y^2z^2$$

```
> Int(Int(exp(x)*sin(y),x=0..cos(y)),y=Pi/6..Pi/4)
  =int(int(exp(x)*sin(y),x=0..cos(y)),y=Pi/6..Pi/4);
```

$$\int_{\frac{1}{6}\pi}^{\frac{1}{4}\pi}\int_{0}^{\cos(y)} e^x\sin(y)dx\,dy = -e^{\left(\frac{1}{2}\sqrt{2}\right)} + \frac{1}{2}\sqrt{2} + e^{\left(\frac{1}{2}\sqrt{3}\right)} - \frac{1}{2}\sqrt{3}$$

```
> lhs(%) = evalf(rhs(%));
```

$$\int_{\frac{1}{6}\pi}^{\frac{1}{4}\pi}\int_{0}^{\cos(y)} e^x\sin(y)dx\,dy = .1904090720$$

同时，**student** 函数库中的命令 **Doubleint** 和 **Tripleint** 可分别用来计算双重积分和三重积分。这两个命令都是惰性命令，可以结合命令 **value** 和 **evalf** 得到最终的积分结果。命令 **Doubleint** 和 **Tripleint** 的使用格式是类似的。以命令 **Doubleint** 为例，有下面三种命令格式：

- **Doubleint(g, x, y)**：计算被积函数 **g** 的双重不定积分；
- **Doubleint(g, x, y, Domain)**：在给定的积分域 **Domain** 上计算函数 **g** 的双重积分，其中 **Domain** 是可选项，代表积分域的名字；
- **Doubleint(g, x = a..b , y = c..d)**：用来计算双重定积分。

在使用这两个命令之前需要加载 **student** 函数库，或者使用 **student[Doubleint]**或 **student[Tripleint]**的形式调用函数。下面给出几个使用命令 **Doubleint** 和 **Tripleint** 完成双

重积分和三重积分的例子：

```
> with(student):
  Doubleint(x^2*y^3,x,y)=value(Doubleint(x^2*y^3,x,y));
warning,new definition for D
```

$$\iint x^2 y^3 dx\,dy = \frac{1}{12} y^4 x^3$$

```
> f:='f';g:='g':
  Tripleint(f*g,x,y,z,Omega);
```

$$\iiint_{\Omega} fg\,dx\,dy\,dz$$

```
> Doubleint(exp(x)*sin(y),x=0..cos(y),y=Pi/6..Pi/4)=
  evalf(Doubleint(exp(x)*sin(y),x=0..cos(y),
  y=Pi/6..Pi/4));
```

$$\int_{\frac{1}{6}\pi}^{\frac{1}{4}\pi}\int_{0}^{\cos(y)} e^x \sin(y)\,dx\,dy = .1904090710$$

10.2　积分技术

换元积分法和分部积分法是两种基本的积分技术，Maple V 提供的集成命令 **int** 能自动选择合适的积分技术完成常见的积分运算。但在必要的时候，用户也可以手动完成积分运算。本节将分别介绍换元积分法和分部积分法的计算过程。

10.2.1　换元积分法

换元积分法是根据复合函数微分的链式法则而来的。Maple V 在 **student** 函数库中提供了 **changevar** 命令，可以实现换元积分法。命令 **changevar** 的基本格式为 **changevar(s, f)**，其中 **s** 具有 **h(x) = g(u)**的形式，规定了原积分变量 **x** 和新积分变量的函数关系，也可以是隐函数关系；**f** 代表由积分符号或者命令 **Int** 本身表示的积分表达式，包括定积分和不定积分。如果在 **s** 中还包含其他变量，可采用 **changevar(s, f, u)**的命令格式。在使用命令 **changevar** 前首先加载 **student** 函数库，或者使用 **student[changevar]**的形式调用。下面给出换元积分法的一般例子：

```
> f:='f':g:='g':h:='h':
  h:=f@g:
  D(h)(x);
```

$$D(f)(g(x))D(g)(x)$$

```
> Int(D(h)(x),x)=h(x);
```

$$\int D(f)(g(x))D(g)(x)dx = f(g(x))$$

```
> student[changevar](g(x)=u,%,u);
```

$$\int D(f)(u)du = f(u)$$

在换元积分过程中，新积分变量 **u** 有多种选择，一个较好的选择是 **D(g)(x)**刚好是被积函数的因子，**g(x)**是某个已知函数 **f(x)**的参数。下面用一个例子介绍换元积分法的计算过程：

（1） **tocompute** 是要计算的积分表达式：

```
> f:='f':f:=x^2*sin(x^3)*cos(cos(x^3)):
  tocompute:=Int(f,x);
```

$$tocompute := \int x^2 \sin(x^3)\cos(x^3))dx$$

（2）**afterchange** 是换元后的积分表达式，下面使用命令 **value** 得到积分结果，并赋值给变量 **midresult**：

```
> afterchange:=
      student[changevar](cos(x^3)=u,tocompute,u);
```

$$afterchange := \int -\frac{1}{3}\cos(u)\,du$$

```
> midresult:=value(afterchange);
```

$$midresult := -\frac{1}{3}\sin(u)$$

（3）命令 **subs(x=a, expr)**是用表达式 **a** 来替换表达式 **expr** 中的变量 **x**，下面用命令 **subs** 得到所求的最终积分结果 **finalresult**：

```
> finalresult:=subs(u=cos(x^3),midresult);
```

$$finalresult := -\frac{1}{3}\sin(\cos(x^3))$$

在大多数情况下，命令 **int** 在完成积分运算的过程中能够自动使用这些积分技术。下面给出用命令 **int** 直接积分得到的结果：

```
> Int(f,x)=int(f,x);
```

$$\int x^2\sin(x^3)\cos(\cos(x^3))dx=-\frac{1}{3}\sin(\cos(x^3))$$

在计算定积分时，命令 **changevar** 能自动改变换元后的积分限，从而保证换元前后的积分结果保持一致。看下面的例子：

```
> Int(sqrt(1-x^2),x);
```

$$\int\sqrt{1-x^2}\,dx$$

```
> student[changevar](x=sin(u),
            Int(sqrt(1-x^2),x),u);
```

$$\int\sqrt{1-\sin(u)^2}\cos(u)\,du$$

命令 **changevar** 也可以对多重积分进行换元操作，命令的格式为 **changevar(s, intexpr, v)**。其中 **s** 代表原积分变量与新积分变量之间的函数关系方程组，用集合的形式表示；**intexpr** 代表双重积分或者多重积分的积分表达式；**v** 是新积分变量的列表。看下面的例子：

```
> Doubleint(1,x,y);
 changevar({x=r*cos(t),y=r*sin(t)},
                    Doubleint(1,x,y),[t,r]);
```

$$\iint 1\,dx\,dy$$

$$\iint |r|\,dt\,dr$$

10.2.2　分部积分法

分部积分法是根据积的微分性质发展来的：由于 $D(uv)=D(u)v+uD(v)$，所以 $\int u\,dv=u\,v-\int v\,du$。student 函数库中的命令 **intparts** 可以用来对积分表达式实施分部积分。命令的格式为 **intparts(f, u)**。其中 **f** 代表由积分符号或者命令 **Int** 本身表示的积分表达式，可以是不定积分或定积分；**u** 代表从被积函数中分离出来的因子。和命令 changevar 一样，使用命令 **intparts** 之前要加载 **student** 函数库。

下面给出分部积分的一般形式：

```
> f:='f':g:='g':
 I1:=Int(f(x)*D(g)(x),x);
```

$$I1:=\int f(x)D(g)(x)\,dx$$

```
> I1=student[intparts](I1,f(x));
```

$$\int f(x)\,D(g)(x)\,dx = f(x)\,g(x) - \int\left(\frac{\partial}{\partial x}f(x)\right)g(x)\,dx$$

在计算积分的过程中，命令 **int** 能够自动选择合适的积分技术计算出最终的结果。用户也可以使用 **intparts** 命令，按照分部积分法的步骤手动完成积分运算。下面给出了一个较为复杂的例子，使用命令 **int** 可以得到最终的积分结果：

```
> f:='f':f:=sin(3*x)*exp(-4*x):
  Int(f,x)=int(f,x);
```

$$\int \sin(3x)e^{(-4x)}dx = -\frac{3}{25}e^{(-4x)}\cos(3x) - \frac{4}{25}\sin(3x)e^{(-4x)}$$

用户也可以按照下面的步骤手动完成上述的积分运算：

（1）用 **toint** 表示要计算的积分表达式：

```
> toint:=Int(f,x);
```

$$toint := \int \sin(3x)e^{(-4x)}dx$$

（2）使用命令 **intparts** 进行分部积分，首先加载 **sthdent** 函数库，Maple V 给出了算子 **D** 被重新定义的警告信息。忽略该信息，用 **step1** 表示分部积分后的结果：

```
> with(student):
  step1:=intparts(toint,exp(-4*x));
warning,new definition for D
```

$$stip1 := -\frac{1}{3}e^{(-4x)}\cos(3x) - \int\frac{4}{3}e^{(-4x)}\cos(3x)\,dx$$

（3）注意到 **setp1** 中的第二部分用常规积分方法仍无法计算，用命令 **intparts** 进行第二次分部积分，用 **step2** 表示第二次分部积分的结果：

```
> step2:=intparts(step1,exp(-4*x));
```

$$step2 := -\frac{1}{3}e^{(-4x)}\cos(3x) - \frac{4}{9}\sin(3x)e^{(-4x)} + \int -\frac{16}{9}\sin(3x)e^{(-4x)}dx$$

（4）**step2** 的第三部分中包含了要计算的积分表达式 **toint** $=\int \sin(3x)e^{(-4x)}dx$ 把上面的分部积分过程写为下面的等式，并用 **intequ** 来表示：

```
> intequ:=toint=step2;
```

$$intequ := \int \sin(3x)e^{(-4x)}dx =$$

$$-\frac{1}{3}e^{(-4x)}\cos(3x) - \frac{4}{9}\sin(3x)e^{(-4x)} + \int -\frac{16}{9}\sin(3x)e^{(-4x)}dx$$

（5）注意到只要把 $\int \sin(3x)\,e^{(-4x)}\,dx$ 从方程中分离出来就大功告成了，但是 Maple V 不能直接把 $\int \sin(3x)\,e^{(-4x)}\,dx$ 作为未知数求解；在方程的左右两边同时减去方程右边中的第三项 $\int -\frac{16}{9}\sin(3x)\,e^{(-4x)}\,dx$，并使用命令 **combine** 合并同类项，用 **newlhs** 和 **newrhs** 分别表示合并同类项后等式的左边和右边：

```
> newlhs:=combine(lhs(intequ)-op(3,rhs(intequ)));
```

$$newlhs := \int \frac{25}{9}\sin(3x)e^{(-4x)}dx$$

```
> newrhs:=(rhs(intequ)-op(3,rhs(intequ));
```

$$newrhs := -\frac{1}{3}e^{(-4x)}\cos(3x) - \frac{4}{9}\sin(3x)e^{(-4x)}$$

（6）在等式的两边同乘 **9/25**，并用命令 **combine** 把积分号内外的常数合并，得到了最终的积分结果，和前面用命令 **int** 直接计算的结果是相同的：

```
> combine(newlhs*(9/25)=newrhs*(9/25));
```

$$\int \sin(3x)e^{(-4x)}dx = -\frac{3}{25}e^{(-4x)}\cos(3x) - \frac{4}{25}\sin(3x)e^{(-4x)}$$

在积分运算的过程中，可以同时使用换元积分法和分部积分法。在下面的例子中，同时应用换元积分法和分部积分法手动完成了反正切函数的定积分：

```
> I4:=Int(arctan(x),x=0..1);
  I4:=intparts(I4,arctan(x));
```

$$\int_0^1 \arctan(x)dx = \frac{1}{4}\pi - \int_0^1 \frac{x}{1+x^2}dx$$

```
> lhs(%)=changevar(u=x^2+1,rhs(%),u);
```

$$\int_0^1 \arctan(x)dx = \frac{1}{4}\pi - \int_1^2 \frac{1}{2}\frac{1}{u}du$$

```
> lhs(%)=value(rhs(%));
```

$$\int_0^1 \arctan(x)\,dx = \frac{1}{4}\pi - \frac{1}{2}\ln(2)$$

10.3 积分变换和积分函数

积分变换的种类很多，Maple V 在 **inttrans** 函数库中提供了各种积分变换的实现命令。积分函数是用积分定义的函数，Maple V 提供了许多标准的积分函数。本节将介绍其中最常用的积分变换：Laplace 变换和 Fourier 变换，同时将对几种常用的标准积分函数进行简单介绍。

10.3.1 Laplace 变换和 Fourier 变换

Maple V 提供的积分变换命令都包含在 **inttrans** 函数库中，读者可以用命令**?inttrans** 获得 **inttrans** 函数库的更多信息。Laplace 变换和 Fourier 变换的命令分别为 **laplace** 和 **fourier**，在使用之前需要用命令 **with(inttrans)** 加载 **inttrans** 函数库，或者使用 **inttrans[laplace]**或 **inttrans[fourier]**的格式调用。表 10.1 中给出了 Laplace 变换和 Fourier 变换的数学定义和在 Maple V 中的命令格式。

表 10.1　　常用的积分变换

积分变换种类	积分变换的定义	Maple V 中的命令格式
Laplace 变换	$\int_0^{\infty} f(t)\, e^{(-s\,t)}\, dt$	**laplace(f(t), t, s)**
Fourier 变换	$\int_{-\infty}^{\infty} f(t)\, e^{(-i\,s\,t)}\, dt$	**fourier(f(t), t ,s)**

1. Laplace 变换

包括指数函数、三角函数、误差函数、Bessel 函数等诸多函数在内的函数都可以进行 Laplace 变换。与命令 **laplace** 相对应的 Laplace 逆变换的命令为 **invlaplace**，同样也包含在 **inttrans** 函数库中。下面给出几个 Laplace 变换与 Laplace 逆变换的例子：

```
> with(inttrans):
  laplace(t^2+sin(t)=y(t),t,s);
```

$$2\frac{1}{s^3}+\frac{1}{s^2+1}=\text{laplace}(y(t),t,s)$$

```
> invlaplace(%,s,t);
```

$$t^2+\sin(t)=y(t)$$

```
> laplace(diff(y(t),t$2)-y(t)=sin(2*t),t,s);
```

$$s\,(s\,\mathrm{laplace}(y(t),t,s)-y(0))-D(y)(0)-\mathrm{laplace}(y(t),t,s)=2\frac{1}{s^2+4}$$

```
> invlaplace(%,s,t);
```

$$\left(\frac{\partial^2}{\partial t^2}y(t)\right)-y(t)=\frac{1}{2}\sqrt{4}\sin(\sqrt{4}\,t)$$

```
> simplify(%);
```

$$\left(\frac{\partial^2}{\partial t^2}y(t)\right)-y(t)=\sin(2t)$$

如果积分变换后的结果中包含卷积，Maple V 将使用全局变量**_U1**、**_U2** 等作为积分变量。看下面的例子：

```
> F:='F':
  invlaplace(s/(s-1)*laplace(F(t),t,s),s,t);
```

$$F(t)+\int_0^t F(\_UI)e^{(t-\_UI)}d\_UI$$

2. Fourier 变换

Fourier 变换广泛应用在 Fourier 分析、信号处理等领域里。包括复数指数函数、多项式函数、正余弦函数、脉冲函数等在内的函数都可以进行 Fourier 变换。与命令 **fourier** 相对应，**inttrans** 函数库中的命令 **invfourier** 可以实现 Fourier 逆变换。

下面给出几个 Fourier 变换和 Fourier 逆变换的例子：

```
> with(inttrans):f:='f':
  fourier(diff(f(x),x$4),x,w);
```

$$w^4\,\mathrm{fourier}(f(x),x,w)$$

```
> invfourier(%,w,x);
```

$$(D^{(4)})(f)(x)$$

```
> fourier(t*exp(-3*t)*Heaviside(t),t,w);
```

$$\frac{1}{(3+Iw)^2}$$

```
> invfourier(%,w,x);
```

$$xe^{(-3x)}\mathrm{Heaviside}(x)$$

上面的例子用到了脉冲函数 **Heaviside(x)**。和 Laplace 变换一样，如果变换后的结果中包含卷积，Maple V 也将使用全局变量_U1、_U2 等作为积分变量。看下面的例子：

```
> F:=int(g(x)*h(t-x),x=-infinity..infinity);
  fourier(3*F,t,w);
```

$$F := \int_{-\infty}^{\infty} g(x)h(t-x)\,dx$$

$$3\,\mathrm{fourier}(g(t),t,w)\,\mathrm{fourier}(h(t),t,w)$$

```
> invfourier(%,w,x);
```

$$\frac{3}{4}\frac{\int_{-\infty}^{\infty} 4\pi^2 g(-\_UI)\,h(x+\_UI)\,d\_UI}{\pi^2}$$

```
> combine(%);
```

$$\int_{-\infty}^{\infty} 3\,g(-\_UI)\,h(x+\_UI)\,d\_UI$$

Fourier 正弦变换和 Fourier 余弦变换是 Fourier 变换的一种变体，二者的定义为 $F(s)=\sqrt{\frac{2}{\pi}}\int_0^{\infty} f(t)\sin(st)\,dt$ 、 $F(s)=\sqrt{\frac{2}{\pi}}\int_0^{\infty} f(t)\cos(st)\,dt$ ，对应的命令分别为 **fouriersin** 和 **fouriercos**。这两个命令也包含在 **inttrans** 函数库中，使用格式和命令 **fourier** 类似，这里不再介绍。下面给出几个应用的例子：

```
> fouriersin(t/(t^2+1),t,s);
```

$$\frac{1}{2}\sqrt{2}\sqrt{\pi}\,e^{(-s)}$$

```
> fouriersin(%,s,x);
```

$$\frac{x}{1+x^2}$$

```
> fouriercos(1/(t^2+1),t,s);
```

$$\frac{1}{2}\sqrt{2}\sqrt{\pi}\,e^{(-s)}$$

```
> fouriercos(%,s,x);
```

$$\frac{x}{1+x^2}$$

10.3.2 常用积分函数

为了方便用户，Maple V 还提供了许多常用的积分函数包括误差函数、椭圆积分函数、三角积分和双曲积分函数、Fresnel（菲涅尔）积分函数、Dawson（道森）积分函数等，下面将简单介绍其中比较常用的几种积分函数。

1. 误差函数

Maple V 中提供了三个误差函数的命令，其中包括最常用的误差函数 **erf**。表 10.2 中给出了三个误差函数的定义及其命令格式。

表 10.2　　Maple V 的误差函数

误差函数	误差函数的定义	Maple V 中的命令格式
基本误差函数 erf(x)	$\operatorname{erf}(x)=\frac{2\int_0^x e^{(-t^2)}dt}{\sqrt{\pi}}$	erf(x)
补偿误差函数 erfc(x)	erfc(x)=1-erf(x)	erfc(x)
复数误差函数 erfi(x)	$\operatorname{erfi}(x)=\frac{2\int_0^x e^{(t^2)}dt}{\sqrt{\pi}}$	erfi(x)

这三个命令函数都是完备的，不存在奇异点。命令中的 x 可以是任意的代数表达式，命令的使用都比较简单，下面给出了几个例子：

```
> erf(3);
```

$$\operatorname{erf}(3)$$

```
> evalf(%);
```

$$.9999779095$$

```
erfc(3.);
```

$$.00002209049700$$

```
erfi(-x);
```

$$-\operatorname{erfi}(x)$$

其中，命令 **erfc** 还有一种格式 **erfc(n,x)**，用来计算补偿误差函数的积分，其中 n≥-1。命令 **erfc(n, x)**对应的定义为：

$\operatorname{erfc}(-1,x)=\frac{2e^{(-x^2)}}{\sqrt{\pi}}$，n≥0 时 $\operatorname{erfc}(n,x)=\int_x^{\infty}\operatorname{erfc}(n-1,t)\,dt$，**erfc(0, x)=erfc(x)**。

下面给出了这种命令格式的两个例子：

```
>erfc(5,0);
```

$$\frac{1}{60}\frac{1}{\sqrt{\pi}}$$

```
>diff(erfc(5,x),x);
```

$$-\mathrm{erfc}(4,x)$$

2. 椭圆积分函数

椭圆积分是指具有 $\int_a^b R(x, y^{1/2})dx$ 形式的积分，其中 R 是关于 x 和 $y^{1/2}$ 的分式函数，y 通常是一个三次或四次多项式。Maple V 中提供了常用的三类椭圆积分的函数，在表 10.3 中给出了这些椭圆积分函数的定义与用法。

表 10.3　　Maple V 中的椭圆积分函数

描　　述	定　　义	Maple V 中的命令格式
第一类不完全椭圆积分	$\mathrm{EllipticF}(z,k)=\int_0^z \frac{1}{\sqrt{1-t^2}\sqrt{1-k^2t^2}}dt$	**EllipticF(z,k)**
第一类完全椭圆积分	EllipticK(k)=EllipticF(1,k)	**EllipticK(k)**
第一类补偿完全椭圆积分	$\mathrm{EllipticCK}(k)=\mathrm{EllipticF}(1,\sqrt{1-k^2})$	**EllipticCK(k)**
第二类不完全椭圆积分	$\mathrm{EllipticE}(z,k)=\int_0^z \frac{\sqrt{1-k^2t^2}}{\sqrt{1-t^2}}dt$	**EllipticE(z, k)**
第二类完全椭圆积分	EllipticE(k)=EllipticE(1,k)	**EllipticE(k)**
第二类补偿完全椭圆积分	$\mathrm{EllipticCE}(k)=\mathrm{EllipticE}(1,\sqrt{1-k^2})$	**EllipticCE(k)**
第三类不完全椭圆积分	$\mathrm{EllipticPi}(z,a,k)=\int_0^z \frac{1}{(1-at^2)\sqrt{1-t^2}\sqrt{1-k^2t^2}}dt$	**EllipticPi(z, a, k)**
第三类完全椭圆积分	EllipticPi(a,k)=EllipticPi(1,a,k)	**EllipticPi(a, k)**
第三类补偿完全椭圆积分	$\mathrm{EllipticCPi}(a,k)=\mathrm{EllopticPi}(1,a,\sqrt{1-k^2})$	**EllipticCPi(a, k)**

表 10.3 中的椭圆积分函数虽然可以在复数范围内使用，相应的参数可以是复数表达式，但更多是应用在实数范围内，这时其中的参数 **k** 和 **a** 满足：**0<k<1**，**0<a<1**。在计算

其他形式的椭圆积分时，Maple V 常常用表中的几个标准椭圆积分函数表示计算结果。下面给出几个简单的应用：

```
> EllipticF(0.2,0.3);
```

.2014795901

```
> EllipticE(0.3);
```

1.534833465

```
> EllipticCPi(0.2,0.3);
```

3.032020785

```
> Int(sqrt(1+2*sin(x)),x=0..Pi/2)=
      int(sqrt(1+2*sin(x)),x=0..Pi/2);
```

$$\int_0^{\frac{1}{2}\pi}\sqrt{1+2\sin(x)}\,dx=-\text{EllipticK}\left(\frac{1}{2}\sqrt{3}\right)+\text{EllipticF}\left(\frac{1}{3}\sqrt{2}\sqrt{3},\frac{1}{2}\sqrt{3}\right)$$
$$+\text{EllipticPi}\left(\frac{3}{4},\frac{1}{2}\sqrt{3}\right)-\text{EllipticPi}\left(\frac{1}{3}\sqrt{2}\sqrt{3},\frac{3}{4},\frac{1}{2}\sqrt{3}\right)$$

```
> Int(sqrt(1+2*sin(x)),x=0..Pi/2)=evalf(rhs(%));
```

$$\int_0^{\frac{1}{2}\pi}\sqrt{1+2\sin(x)}\,dx=2.343854231$$

3. 三角积分和双曲积分函数

表 10.4 中给出了 Maple V 提供的三角积分和双曲积分函数的定义和命令格式，其中的 r 为 Euler 常数。这些积分函数也经常出现在复杂函数的积分运算结果中。

表 10.4　　三角积分和双曲积分函数

积分函数	积分函数的定义	Maple V 中的命令格式
正弦积分函数(1) Si(x)	$\text{Si}(x)=\int_0^x\frac{\sin(t)}{t}\,dt$	Si(x)

续表

积分函数	积分函数的定义	Maple V 中的命令格式
余弦积分函数 Ci(x)	$\mathrm{Ci}(x)=\gamma+\ln(x)+\int_0^x\frac{\cos(t)-1}{t}dt$	Ci(x)
正弦积分函数(2) Ssi(x)	$\mathrm{Ssi}(x)=\mathrm{Si}(x)-\frac{\pi}{2}$	Ssi(x)
双曲正弦积分函数 Shi(x)	$\mathrm{Shi}(x)=\int_0^x\frac{\sinh(t)}{t}dt$	Shi(x)
双曲余弦积分函数 Chi(x)	$\mathrm{Chi}(x)=\gamma+\ln(x)+\int_0^x\frac{\cosh(t)-1}{t}dt$	Chi(x)

函数 Si(x)、Ssi(x)、Shi(x)是完备的，命令中的 x 可以是任意的代数表达式；而 Ci(x)和 Chi(x)在原点存在奇异，命令中的 x 可以是除了原点之外的代数表达式。下面给出这几个命令的例子：

```
> Si(12345.67890);
```

$$1.570738760$$

```
> Ssi(12345.67890);
```

$$-.00005756635677$$

```
> Shi(Pi);
```

$$\mathrm{Shi}(\pi)$$

```
> evalf(%);
```

$$5.469640347$$

```
> Chi(1.+I);
```

$$.8821721806+1.283547193I$$

```
> Ci(0);
Error, (in Ci) singularity encountered
```

4. 指数积分函数和对数积分函数

表 10.5 中将给出 Maple V 提供的指数积分函数和对数积分函数的定义以及对应的命令格式。

表 10.5　指数积分和对数积分函数

函数种类	误差函数的定义	Maple V 中的命令格式
指数积分函数 **Ei(n,x)**	$\mathrm{Ei}(n,x)=\int_1^{\infty}\frac{e^{(-xt)}}{t^n}dt$	**Ei(n,x)** 其中 **n≥0**
柯西积分函数 **Ei(x)**	$\mathrm{Ei}(x)=\int_{-\infty}^{x}\frac{e^t}{t}dt$ x < 0时, Ei(x) = −Ei(1,−x)	**Ei(x)**
对数积分函数 **Li(x)**	$\mathrm{Li}(x)=\int_0^{x}\frac{1}{\ln(t)}dt$, x > 1	**Li(x)** 其中 **x>1**
对数积分函数 **dilog(x)**	$\mathrm{dilog}(x)=\int_1^{x}\frac{\ln(t)}{1-t}dt$	**dialog(x)**

这几个命令都比较简单，使用时需注意各个命令的使用条件。下面给出了几个例子：

```
> Ei(1);evalf(%);
```

$$\mathrm{Ei}(1)$$

$$1.895117816$$

```
> Ei(1,1.);
```

$$.2193839344$$

```
> Ei(-1.);
```

$$-.2193839344$$

```
> Li(10.);
```

$$6.165599505$$

在 Maple V 中还有许多标准的积分函数，读者可以参看相关的帮助，了解更多的积分函数的信息，这里不再介绍。

第 11 章 矩阵和向量

矩阵和向量是高等代数中的一个重要课题，同时又是理论研究和工程应用的重要工具。Maple V 中定义了矩阵和向量两种数据类型，允许用户对矩阵和向量执行常见的各种运算和操作，其中包括矩阵特征值和特征向量的计算等。本章将介绍矩阵和向量的基本知识和运算。

11.1 矩阵和向量的基本知识

在 Maple V 中，矩阵和向量是与数组相关的两种数据结构，向量对应于下标从 1 开始的一维数组，矩阵对应于两个下标都从 1 开始的二维数组。矩阵的行列操作和块操作在矩阵的初等变换中经常用到。

本节首先介绍矩阵和向量的定义，接着介绍矩阵的行列操作和矩阵的块操作。

11.1.1 矩阵和向量的定义

矩阵和向量是数组的特例，用户可以用定义数组的方法定义矩阵和向量，这是定义矩阵和向量的基本方法。用户也可以使用命令 **matrix** 和 **vector** 分别定义矩阵和向量。另外，**linalg** 函数库中的命令 **randmatrix** 和 **randvector** 分别用来生成随机矩阵和随机向量。

本小节主要介绍矩阵和向量的定义方法，包括矩阵和向量的 **array** 定义、命令 **matrix** 和 **vector** 的使用、特殊矩阵的定义以及随机矩阵和随机向量的生成方法，最后简单介绍矩阵元素的引用和赋值。

1. 矩阵和向量的 array 定义

在 7.3 节中比较详细地介绍了命令 **array** 的用法，用户可以参照其中的命令格式定义矩阵和向量。Maple V 采用数学中的矩阵表示法表示矩阵，用户可以使用命令 **op**、**eval** 或者 **print** 打印矩阵的结构，但命令 **print** 的输出结果和命令 **op**、**eval** 的输出结果略有差别。下面给出一个矩阵的例子：

```
> A:=array(1..3,1..4);
```

$$A := \text{array}(1..3,1..4,[\ ])$$

```
> print(A);
```

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

```
> op(A);
```

$$\begin{bmatrix} ?_{1,1} & ?_{1,2} & ?_{1,3} & ?_{1,4} \\ ?_{2,1} & ?_{2,2} & ?_{2,3} & ?_{2,4} \\ ?_{3,1} & ?_{3,2} & ?_{3,3} & ?_{3,4} \end{bmatrix}$$

Maple V 用类似列表的形式表示向量，虽然具有行向量的形式，但在运算过程中 MapleV 把向量作为列向量使用。下面是一个向量的例子：

```
> V:=array(1..4,[1,0,0]);
```

$$V := [1,0,0,V_4]$$

```
> V;
  print(V);
```

$$V$$

$$[1,0,0,V_4]$$

矩阵的类型为 **matrix**，向量的类型为 **vector**，二者都是数组的子类型，属于 **vector** 类型的数组不同时属于 **matrix** 数据类型。用户可以用命令 **type** 分析矩阵和向量的类型。但用来代表矩阵和向量的标识符和其他的对象名是不同的，用户不能使用命令 **whattype** 得到矩阵名或者向量名的真正类型。看下面的例子：

```
> whattype(A);
  type(A,matrix),type(V,vector);
```

symbol

true,true

```
> whattype(array(1..4,[1,0,0]));
  type(V,array);
```

array

true

2. matrix 和 vector

命令 **matrix** 的基本格式为：**matrix(m, n, L)**，其中 **m** 和 **n** 分别代表矩阵的行数和列数，二者必须同时出现；**L** 代表由已知元素值构成的列表，列表中的元素按照行的顺序排列。如果矩阵中的元素都是未知的，可以使用 **matrix(m, n)**的命令格式。如果矩阵的元素都是已知的，用户可以使用 **matrix(dlst)**的命令格式，**dlst** 以嵌套列表的形式给出矩阵的元素值，**dlst** 中的每一个子列表代表矩阵的一行元素值，各行元素按照行的顺序排列，Maple V 根据 **dlst** 的构成确定矩阵的行数和列数。下面是两个用命令 **matrix** 定义的矩阵：

```
> matrix(2,2,[5,4,6,3]);
```

$$\begin{bmatrix} 5 & 4 \\ 6 & 3 \end{bmatrix}$$

```
> matrix([[5,4],[6,3]]);
```

$$\begin{bmatrix} 5 & 4 \\ 6 & 3 \end{bmatrix}$$

如果矩阵只有一行或者一列，采用 **matrix(dlst)**的命令格式时，参数 **dlst** 也必须采用嵌套列表的形式来表示。用这种方式定义的行向量和列向量，被看作是只有一行或只有一列的矩阵，不属于 Maple V 的向量数据类型，Maple V 中的向量只能使用命令 **array** 和 **vector** 定义。看下面的例子：

```
> hv:=matrix([[1,2,3]]);
```

$$hv := [1 \quad 2 \quad 3]$$

```
> lv:=matrix([[1],[2],[3]]);
```

$$lv := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

```
> whattype(hv),whattype(lv);
  type(hv,vector),type(lv,vector);
  type(hv,matrix),type(lv,matrix);
```

symbol, symbol

false, false

true, true

命令 **vector** 的基本格式为：**vector(n, [x1, x2, ..., xk])**，其中 **n** 给出了向量的长度，代数表达式 **x1**、**x2**、…、**xk** 给出了部分或全部的元素值，且 $k \le n$，其余的元素是未知的。如果 **n** 省略，Maple V 默认向量的长度等于所给的元素个数；如果所有元素都是未知的，可以使用 **vector(n)**的格式。下面是两个向量的例子：

```
> v:=vector(4,[1]);
```

$$v := [1, v_2, v_3, v_4]$$

```
> vector(4);
  whattype(%),type(%,vector),type(%,matrix);
```

$$[?_1, ?_2, ?_3, ?_4]$$

array, true, false

另外，用户也可以在命令 **matrix** 或 **vector** 中给定以元素的下标为变量的函数，由给定的函数确定矩阵或向量中的元素。在命令 **matrix(m, n, f)**中，**f** 代表给定的二元函数，生成矩阵中的元素值对应于二元函数 **f** 在相应下标点的取值；在命令 **vector(n, f)**中，**f** 代表给定的一元函数，生成向量中的元素值对应于函数 **f** 在相应下标点的取值。其中给定的函数 **f** 也可以是常数函数，最常用的是 **f≡0** 或者 **f≡1**。下面给出这种命令格式的两个例子：

```
> f:=(i,j)->x^(i+j-1):
  A:= matrix(2,2,f);
```

$$A := \begin{bmatrix} x & x^2 \\ x^2 & x^3 \end{bmatrix}$$

```
> zerovec:= vector(5,0);
```

$$zerovec := [0,0,0,0,0]$$

3. 特殊矩阵

特殊矩阵包括单位矩阵、对称矩阵、反对称矩阵、对角矩阵、稀疏矩阵等，这些矩阵在元素的构成上形成了比较特殊的结构。在命令 **array** 中加入特征选项，可以定义这些特殊结构的矩阵。命令的基本格式为 **array(CF, 1..m, 1..n, L)**的命令格式，特征选项 **CF** 的常用取值有 **identity**、**symmetric**、**antisymmetric**、**diagonal**、**sparse** 五种，分别对应于单位矩阵、对称矩阵、反对称矩阵、主对角矩阵和稀疏矩阵。下面的例子中，分别定义了单位矩阵 **IM**、稀疏矩阵 **SM** 和对角矩阵 **DM**：

```
> IM:= array(identity,1..3,1..3);
```

$$IM := \text{array}(identity, 1..3, 1..3, [\,])$$

```
> print(IM);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> SM:= array(sparse,1..3,1..4,[(2,3)=3,(1,1)=8,
        (3,1)=7,(2,2)=11]);
```

$$SM := \begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & 11 & 3 & 0 \\ 7 & 0 & 0 & 0 \end{bmatrix}$$

```
> DM:= array(diagonal,1..3,1..4,[(1,1)=2,(2,2)=3,
  (3,3)=7]);
```

$$DM := \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 7 & 0 \end{bmatrix}$$

在定义向量时，如果使用命令 **array(identity, 1..n)**，定义的是一个全 **1** 的 **n** 维向量；如果使用命令 **array(sparse, 1..n, lst)**，则生成的是一个 **n** 维的稀疏向量，即列表 **lst** 中未给出的元素为 **0**。其他的几个特征选项在定义向量时是不起作用的。看下面的例子：

```
> array(identity,1..3);
```

$$[1.1.1]$$

```
> array(sparse,1..3,[2=1]);
```

$$[0,1,0]$$

```
> array(symmetric,1..3,[2=1]);
```

$$[?_1,1,?_3]$$

需要指出的是，这些特征选项只能在命令 **array** 中使用，不能用在命令 **matrix** 和 **vector** 中。

4. 随机矩阵和随机向量

命令 **randmatrix** 和 **randvector** 分别用来生成随机矩阵和随机向量。在使用这两个命令之前，需要用命令 **with(linalg)**加载 **lianlg** 函数库。

命令 **randmatrix** 用来生成随机矩阵，其基本的命令格式为 **randmatrix(m, n, options)**。其中 **m** 和 **n** 代表随机矩阵的行数和列数；可选参数 **options** 代表随机矩阵的特征选项，缺省情况下的默认值为 **dense**，其他几个常用的特征选项有 **sparse**、**symmetric**、**antisymmetric**、**unimodular** 等，分别对应于稀疏随机矩阵、对称随机矩阵、反对称随机矩阵、随机酉矩阵（复数域）；可选参数 **options** 也可以具有 **entries = f** 形式的等式，其中 **f** 给定了矩阵中元素的生成规则，在缺省情况下 **entries = rand(-99, 99)**，用来生成所有元素都为二位随机整数的随机矩阵。下面给出几个随机矩阵的例子：

```
> with(linalg):
warning,new definition for norm
warning,new definition for trace
```

```
> rm1:=randmatrix(2,3);
```

$$rm1 := \begin{bmatrix} -85 & -55 & -37 \\ -35 & 97 & 50 \end{bmatrix}$$

```
> rm2:=randmatrix(2,2,antisymmetric);
```

$$rm2 := \begin{bmatrix} 0 & 79 \\ -79 & 0 \end{bmatrix}$$

```
> poly:=()→randpoly(x,terms=3):
  randmatrix(2,2,entries=poly);
```

$$\begin{bmatrix} -91x^3 - 47x^2 - 61 & 53x^5 - x^3 + 94 \\ -84x^4 + 19x^3 - 50x & 49x^4 + 78x^2 + 17 \end{bmatrix}$$

最后一个例子中用到的命令 **randpoly** 用来生成随机多项式，第一条命令定义了一个无参数函数 **poly**，其返回值为以 **x** 为自变量的次数不超过 **5** 的项数为 **3** 的随机多项式。

命令 **randvector** 用来生成随机向量，基本的命令格式为 **randvector(n, entries=f)**，其中 **f** 规定了向量中元素的生成规则，缺省情况下 **entries=rand(-99, 99)**，用来生成所有元素都是二位整数的随机向量。下面是两个随机向量的例子，**poly** 是前面定义的函数：

```
> randvector(3);
```

$$[72,-99,-85]$$

```
> randvector(2,entries=poly);
```

$$\left[66x^4-29x^2-91,-72x^5-87x^4+79x^2\right]$$

5. 元素的引用和更新

矩阵和向量作为特殊的数组，其中元素的引用和更新方法与数组元素的引用和更新方法完全一样。如果 **mat** 定义为矩阵，则 **mat[i, j]**代表矩阵 **mat** 中第 **i** 行第 **j** 列的元素。看下面的例子：

```
> A:=linalg[randmatrix](3,3,entries=rand(-9..9));
```

$$A:=\begin{bmatrix}4 & -5 & -5\\ 3 & -5 & 8\\ 5 & -1 & 0\end{bmatrix}$$

```
> A[2,2],A[1,3]+A[3,1];
```

$$-5,0$$

在引用矩阵或者向量中的元素时，不能在范围表达式中同时引用多个元素。如果要引用矩阵或向量中的多个元素，读者可以参看 11.1.2 节中介绍的行列操作。虽然向量的表示和列表类似，但其元素的引用规则是完全不同的。看下面的例子，**L** 为列表，**V** 为对应的向量，分析其中的结果：

```
> L:=[1,2,3];
```

$$L:=[1,2,3]$$

```
> L[1..2];
```

$$[1,2]$$

```
> V:=array(L);
```

$$V:=[1,2,3]$$

```
> V[1..2];
```

$$V_{1..2}$$

如果矩阵和向量的元素值与下标之间有一定的函数关系，用户可以在命令 **matrix** 或者 **vector** 中给定对应的二元函数或者一元函数。用户也可以首先定义出矩阵或向量的框架，然后使用 **for** 循环语句对其中的元素赋值。关于 **for** 循环语句的使用，读者可以使用命令**?for** 参看相关的帮助，这里不再介绍。下面给出一个使用 **for** 循环语句对矩阵元素赋值的例子：

```
> A:=array(1..3,1..4):
  for i to 3 do
  for j to 4 do
     A[i,j]:=i^2+j^2
  od   od:
```

```
> print(A);
```

$$\begin{bmatrix} 2 & 5 & 10 & 17 \\ 5 & 8 & 13 & 20 \\ 10 & 13 & 18 & 25 \end{bmatrix}$$

11.1.2 矩阵的行列操作

在 Maple V 的 **linalg** 函数库中，包含了许多用于矩阵的行列整体操作的命令，其中包括行与列的整体引用、交换、数乘、线性组合和删除等。本小节简单介绍这几种操作。注意在使用这些命令之前，需要加载 **linalg** 函数库。

下面定义了一个随机矩阵 **A**，本节主要以矩阵 **A** 为例介绍行列的基本操作：

```
> with(linalg):
  A:=randmatrix(3,3,entries=rand(-9..9));
```

$$A := \begin{bmatrix} 3 & -4 & -5 \\ -2 & -1 & -9 \\ 5 & 8 & 4 \end{bmatrix}$$

1. *行、列的引用*

命令 **row(mat, i)**引用矩阵 **mat** 的第 **i** 行元素，命令 **col(mat, j)**引用矩阵 **mat** 的第 **j** 列元素；其中的 **mat** 代表矩阵，命令的输出都是以向量的形式表示的。命令 **row(mat, i1..i2)**引用矩阵 **mat** 的第 **i1** 到 **i2** 行元素，命令 **col(mat, j1..j2)**引用矩阵 **mat** 的第 **j1** 到 **j2** 列元素；输出的结果是由所引用的行或列构成的向量的序列，其中 **i1<=i2**，**j1<=j2**，否则输出为 **NULL**。

下面给出使用命令 **row** 和 **col** 的例子：

```
> row(A,3);
```

[5,8,4]

```
> row(A,2..1);
```

```
> col(A,1..2);
```

[3,−2,5],[−4,−1,8]

命令 **row** 和 **col** 只能用来引用矩阵的行和列，Maple V 不接受采用这种引用矩阵元素的方法对矩阵的行列整体赋值。看下面的例子：

```
> row(A,3):=[2,4,3]
```

$$\mathrm{row}(A,3):=[2,4,3]$$

```
> print(A);
```

$$\begin{bmatrix} 3 & -4 & -5 \\ -2 & -1 & -9 \\ 5 & 8 & 4 \end{bmatrix}$$

2. 行、列的交换

命令 **swaprow(mat, rowi, rowj)**用来实现矩阵 **mat** 第 **rowi** 行元素与第 **rowj** 行元素的交换，并生成新的矩阵；命令 **swapcol(mat, coli, colj)**用来实现矩阵 **mat** 第 **coli** 列元素与第 **colj** 列元素的互换，并生成新的矩阵。下面的例子中交换了矩阵 **A** 中的第 **1** 列和第 **3** 列，交换后原来的矩阵不发生变化：

```
> swapcol(A,1,3);
```

$$\begin{bmatrix} -5 & -4 & 3 \\ -9 & -1 & -2 \\ 4 & 8 & 5 \end{bmatrix}$$

```
> print(A);
```

$$\begin{bmatrix} 3 & -4 & -5 \\ -2 & -1 & -9 \\ 5 & 8 & 4 \end{bmatrix}$$

3. 行、列的数乘

用户可以对矩阵中的某一行或者某一列中的元素同时乘以一个数，生成新的矩阵，原来的矩阵不发生改变。命令 **mulrow(mat, rowi, expr)**对矩阵 **mat** 的第 **rowi** 行元素同时乘以算术表达式 **expr**，命令 **mulcol(mat, colj, expr)**对矩阵 **mat** 的第 **colj** 列元素同时乘以算术表达式 **expr**，输出新形成的矩阵。看下面的例子：

```
> mulrow(A,2,Pi);
```

$$\begin{bmatrix} 3 & -4 & -5 \\ -2\pi & -\pi & -9\pi \\ 5 & 8 & 4 \end{bmatrix}$$

4. 行、列的线性组合

用户可以对矩阵中的行和列进行线性组合，构造出新的矩阵。命令 **addrow(mat, rowi, rowj, m)**用矩阵 **mat** 的第 **rowi** 行元素的 **m** 倍与第 **rowj** 行元素的和替换矩阵中的第 **rowj** 行元素，生成一个新的矩阵；命令 **addcol(mat, coli, colj, m)**则实现列之间的线性合并，其中 **m** 可以是任意的算术表达式（标量）。看下面的例子：

```
> addcol(A,1,2,x);
```

$$\begin{bmatrix} 3 & 3x-4 & -5 \\ -2 & -2x-1 & -9 \\ 5 & 5x+8 & 4 \end{bmatrix}$$

5. 行、列的删除

用户可以删除矩阵的若干行或若干列，构造出新的矩阵。命令 **delrows(mat, m..n)**删除矩阵 **mat** 中的第 **m** 行到第 **n** 行元素，命令 **delcols(mat, m..n)**删除矩阵 **mat** 中的第 **m** 列到第 **n** 列元素；命令中的第二个参数必须是范围表达式，命令生成由余下的行或列构成的矩阵，原来的矩阵不发生变化。看下面的例子：

```
> delrows(A,2..2);
```

$$\begin{bmatrix} 3 & -4 & -5 \\ 5 & 8 & 4 \end{bmatrix}$$

```
> delcols(A,1..2);
```

$$\begin{bmatrix} -5 \\ -9 \\ 4 \end{bmatrix}$$

11.1.3 矩阵的块操作

前面介绍的是矩阵的行列操作，在 **linalg** 函数库中，同时包含了一些实现矩阵块操作的命令，包括矩阵的分块引用、联合、扩展及转置等。本小节简单介绍这些命令的使用。同样，在使用这些命令之前需要加载 **linalg** 函数库。

1. 矩阵的分块引用

命令 **submatrix** 可以引用矩阵中的任意子矩阵，基本的命令格式为 **submatrix(mat, Rrange, Crange)**。其中 **mat** 代表矩阵，**Rrange** 和 **Crange** 给出了所引用元素的行与列的下标范围；命令的输出结果是由矩阵 **mat** 的这些行与列交叉位置上的元素构成的子矩阵。看下面的例子：

```
> with(linalg):
  A:=randmatrix(3,3,entries=rand(-9..9));
```

$$A := \begin{bmatrix} -3 & 8 & -9 \\ 7 & 5 & 0 \\ 4 & 4 & 2 \end{bmatrix}$$

```
> submatrix(A,2..3,1..2);
```

$$\begin{bmatrix} 7 & 5 \\ 4 & 4 \end{bmatrix}$$

```
> submatrix(A,1..3,1..1);
```

$$\begin{bmatrix} -3 \\ 7 \\ 4 \end{bmatrix}$$

更为灵活地，用户也可以使用 **submatrix(mat, Rlist, Clist)**的格式。其中 **Rlist** 和 **Clist** 以列表形式给出了所引用元素的行列位置下标，新构成的子矩阵中位置下标为**[i, j]**的元素来源于矩阵 **mat** 中位置下标为**[Rlist[i], Clist[j]]**的元素。看下面的例子：

```
> submatrix(A,[3,1],[2,3]);
```

$$\begin{bmatrix} 4 & 2 \\ 8 & -9 \end{bmatrix}$$

```
> M:=array(1..4,1..4);
```

$$M := \text{array}(1..4,1..4,[\ ])$$

```
> submatrix(M,[3,1],[2,3]);
```

$$\begin{bmatrix} M_{3,2} & M_{3,3} \\ M_{1,2} & M_{1,3} \end{bmatrix}$$

2. 矩阵的联合

命令 **augment** 或 **concat** 可以把两个或多个行数相同的矩阵联合为一个新的矩阵。基本的命令格式 **augment(mat_seq)**或者 **concat(mat_seq)**，其中 **mat_seq** 代表行数相同的矩阵的序列。看下面的例子：

```
> B:=matrix([[1,2],[3,4],[5,6]]);
```

$$\mathrm{B} := \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
> augment(A,B);
```

$$\begin{bmatrix} 3 & -4 & -5 & 1 & 2 \\ -2 & -1 & -9 & 3 & 4 \\ 5 & 8 & 4 & 5 & 6 \end{bmatrix}$$

在命令 **augment** 和 **concat** 中的参数也可以包含向量，向量的维数应该与矩阵的行数相

同，Maple V 把向量作为结果矩阵中的一列。看下面的例子：

```
> b:=vector(3);
```

$$b := \text{array}(1..3,[\ ])$$

```
> concat(A,b);
```

$$\begin{bmatrix} 3 & -4 & -5 & b_1 \\ -2 & -1 & -9 & b_2 \\ 5 & 8 & 4 & b_3 \end{bmatrix}$$

3. 矩阵的扩展

矩阵的扩展指的是在原矩阵基础上添加新的行或新的列构成新的矩阵，使用的命令为 **extend(mat, m, n, val)**，其中 **m** 和 **n** 代表新增加的行数和列数，**val** 代表新增加的元素的值。参数 **val** 缺省时，新增加的元素用带下标的?或者矩阵的名称表示。看下面的例子：

```
> extend(A,1,1,0);
```

$$\begin{bmatrix} 3 & -4 & -5 & 0 \\ -2 & -1 & -9 & 0 \\ 5 & 8 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> extend(A,0,1);
```

$$\begin{bmatrix} 3 & -4 & -5 & ?_{1,4} \\ -2 & -1 & -9 & ?_{2,4} \\ 5 & 8 & 4 & ?_{3,4} \end{bmatrix}$$

4. 块对角矩阵

命令 **diag(Bmat_seq)**用来生成块对角矩阵，其中 **Bmat_seq** 以序列的形式给出了结果矩阵对角线上的矩阵块或者元素值。命令 **BlockDiagonal** 与 **diag** 等价，具有相同的参数格式。看下面的例子

```
> diag(A,x);
```

$$\begin{bmatrix} 3 & -4 & -5 & 0 \\ -2 & -1 & -9 & 0 \\ 5 & 8 & 4 & 0 \\ 0 & 0 & 0 & x \end{bmatrix}$$

```
> diag(lambda1,lambda2);
```

$$\begin{bmatrix} \lambda 1 & 0 \\ 0 & \lambda 2 \end{bmatrix}$$

5. 矩阵的转置

命令 **transpose(mat)**可以实现矩阵 **mat** 的转置，并直接给出转置后的结果矩阵。看下面

的例子：

```
> At := transpose(A);
```

$$At := \begin{bmatrix} 3 & -2 & 5 \\ -4 & -1 & 8 \\ -5 & -9 & 4 \end{bmatrix}$$

命令 **transpose(V)**也可以实现向量 **V** 的转置，但结果仍以函数调用的格式显示。看下面的例子：

```
> v := array([1,0,-1]);
  evalm(transpose(v));
```

$$v := [1,0,-1]$$

$$\text{transpose}(v)$$

11.2　矩阵和向量的运算

在 Maple V 中，矩阵和向量能够进行常见的各种运算。其中特征值和特征向量的计算是矩阵代数中的一个重要课题。本节首先介绍矩阵和向量的基本运算，最后介绍矩阵的特征值和特征向量的计算。

11.2.1　矩阵的基本运算

本节介绍矩阵的基本运算，包括矩阵的加法、数乘、乘法、乘方以及初等函数对矩阵的作用法则和矩阵的行列式、逆矩阵、范数计算等。在介绍这些基本运算之前，首先介绍对矩阵表达式求值的命令 **evalm**。

首先定义下面的矩阵 **B**：

```
> B := array([[3,0,2],[-3,2,2],[2,-3,3]]);
```

$$B := \begin{bmatrix} 3 & 0 & 2 \\ -3 & 2 & 2 \\ 2 & -3 & 3 \end{bmatrix}$$

本小节主要以这两个矩阵为例介绍矩阵的基本运算。在矩阵运算中，Maple V 用 **0** 代表任意阶数的零矩阵，用**&*()**代表任意阶数的单位矩阵，其中**&***是矩阵乘法运算符。为了使用方便，用下面的命令将单位矩阵用 **Id** 来表示：

```
>alias=(Id=&*()):
```

1. 命令 **evalm**

命令 **evalm** 用来对矩阵表达式求值，基本格式为 **evalm(matrixexpr)**，其中 **matrixexpr**

代表包含矩阵运算或者向量运算的表达式。命令 **evalm** 首先对矩阵表达式 **matrixexpr** 化简，然后求值。在后面的例子中，读者可以看到命令 **evalm** 的使用，这里不再专门举例。

2. 矩阵的数乘

数字和矩阵的乘法称为矩阵的数乘，运算的法则是用数字和矩阵中的每个元素相乘，结果仍为矩阵；当数字为 **0** 时，数乘的结果为 **0**。数乘的运算符用乘号*表示，**c*M** 代表数字 **c** 与矩阵 **M** 的数乘表达式，其中 **c** 可以是任意的算术表达式；当 **c= -1** 时，可以简写为**-M**。由于矩阵名的特殊性，用户需要用命令 **evalm(c*M)**对数乘表达式求值。这种运算对向量来说也是成立的。下面是矩阵数乘运算的例子：

```
> 0*B;
```

$$0$$

```
> 2*B;
```

$$2B$$

```
> evalm(%);
```

$$\begin{bmatrix} 6 & 0 & 4 \\ -6 & 4 & 4 \\ 4 & -6 & 6 \end{bmatrix}$$

```
> evalm(-B);
```

$$\begin{bmatrix} -3 & 0 & -2 \\ 3 & -2 & -2 \\ -2 & 3 & -3 \end{bmatrix}$$

```
> evalm(sin(x)*B);
```

$$\begin{bmatrix} 3\sin(x) & 0 & 2\sin(x) \\ -3\sin(x) & 2\sin(x) & 2\sin(x) \\ 2\sin(x) & -3\sin(x) & 3\sin(x) \end{bmatrix}$$

另外，**linalg** 函数库的命令 **scalarmul** 也可以实现矩阵的数乘运算，基本的命令格式为 **scalarmal(mat, expr)**，参数 **expr** 代表任意的算术表达式。看下面的计算结果：

```
> scalarmul(B,x^2*cos(x));
```

$$\begin{bmatrix} 3x^2\cos(x) & 0 & 2x^2\cos(x) \\ -3x^2\cos(x) & 2x^2\cos(x) & 2x^2\cos(x) \\ 2x^2\cos(x) & -3x^2\cos(x) & 3x^2\cos(x) \end{bmatrix}$$

3. 矩阵的加法

矩阵的加法运算符用加号表示，**A+B** 表示矩阵 **A** 和 **B** 的加法，其中 **A** 和 **B** 是具有相同阶数的矩阵，运算的结果是由矩阵 **A** 和矩阵 **B** 对应位置的元素相加后得到的元素。用户需

要使用命令 **evalm(A+B)**对矩阵的加法表达式求值。矩阵的减法用 **A-B** 来表示，和 **A+(-B)**是等价的。矩阵的加法同样适用于向量的加法。

下面是两个矩阵相加的例子：

```
> evalm(A + B);
```

$$\begin{bmatrix} 5 & 9 & -5 \\ -10 & -1 & -5 \\ 0 & -10 & 11 \end{bmatrix}$$

```
> evalm(A - B);
```

$$\begin{bmatrix} -1 & 9 & -9 \\ -4 & -5 & -9 \\ -4 & -4 & 5 \end{bmatrix}$$

在矩阵加法中，**A+c** 与 **A+c*Id** 是等价的，其中 **A** 代表矩阵，**c** 为任意的算术表达式，**Id** 代表单位矩阵。看下面的例子：

```
> evalm(A - 2);
```

$$\begin{bmatrix} 0 & 9 & -7 \\ -7 & -5 & -7 \\ -2 & -7 & 6 \end{bmatrix}$$

4. 矩阵的乘法

矩阵乘法运算符用**&***表示，乘号*不能实现矩阵的乘法；矩阵 **A** 与 **B** 的乘法用 **A&*B** 表示，其中 **A** 的列数和矩阵 **B** 的行数相等。用户需要使用命令 **evalm(A&*B)**对矩阵的乘法求值，得到结果矩阵。通常矩阵的乘法不满足交换律，比较下面的计算结果：

```
> evalm(A&*B);
```

$$\begin{bmatrix} -35 & 39 & 1 \\ -26 & 15 & -41 \\ 31 & -38 & 6 \end{bmatrix}$$

```
> evalm(B&*A);
```

$$\begin{bmatrix} 2 & 13 & -5 \\ -24 & -47 & 23 \\ 19 & 6 & 31 \end{bmatrix}$$

虽然向量在 Maple V 中具有行向量的形式，但它在矩阵运算中是作为列向量使用的。矩阵乘法运算符**&***也可以实现矩阵和向量的乘法，其中矩阵的列数和向量的维数相等。下面是矩阵和向量相乘的例子：

```
> v:=array([1,0,-1]);
```

$$v := [1,0,-1]$$

```
> evalm(A&*v);
```

$$[9,0,-10]$$

矩阵乘法 **A&*B** 也可以用**&*(A, B)**的形式表示，这种格式在计算多个矩阵的连乘时通常更为有效。看下面的例子：

```
> &*(A,B,Id);
```

$$\&*(A, B, Id)$$

```
> evalm(%);
```

$$\begin{bmatrix} -35 & 39 & 1 \\ -26 & 15 & -41 \\ 31 & -38 & 6 \end{bmatrix}$$

同时，**linalg** 函数库的命令 **multiply** 也可以实现矩阵的乘法。命令的格式为 **multiply(A, B, ...)**，命令中可以包含多个参数，相邻两个矩阵或向量的列数和行数必须满足矩阵乘法的规则。看下面的例子：

```
> multiply(A,B);
```

$$\begin{bmatrix} -35 & 39 & 1 \\ -26 & 15 & -41 \\ 31 & -38 & 6 \end{bmatrix}$$

```
> v:=array([1,0,-1]);
  multiply(v,transpose(v));
```

$$v := [1,0,-1]$$

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

5. 矩阵的幂

矩阵的幂仍然用 **A^n** 来表示，**A** 代表行数和列数相等的正方矩阵，**n** 通常为整数。当 **n=0** 时，**A^0** 的值为 **1**；当 **n= -1** 时，**A^(-1)**代表矩阵的逆矩阵，这时 **A** 必须是行列式不为 **0** 的非奇异矩阵；当 **n** 为正整数时，**A^n** 代表矩阵 **A** 的 **n** 次乘法；当 **n** 为负整数时，**A^n** 代表矩阵 **A** 的逆矩阵多次相乘。

看下面的计算结果：

```
> A^0;
```

$$1$$

```
> evalm(A^0);
```

$$1$$

```
> evalm(A^(-1));
```

$$\begin{bmatrix} \frac{-73}{183} & \frac{-23}{183} & \frac{-28}{61} \\ \frac{70}{183} & \frac{2}{183} & \frac{21}{61} \\ \frac{43}{183} & \frac{-4}{183} & \frac{19}{61} \end{bmatrix}$$

```
> evalm(A^2);
```

$$\begin{bmatrix} -45 & 40 & -133 \\ 21 & -5 & 14 \\ 29 & -53 & 127 \end{bmatrix}$$

6. *初等函数对矩阵的作用*

在定义了矩阵的加法、数乘、乘法和乘方等基本运算后，许多的初等函数包括多项式函数都可以依据这些基本运算规则对矩阵作用。运算的规则通常是用初等函数分别作用于矩阵的每个元素，生成新的矩阵。用户需要使用命令 **evalm** 得到结果矩阵，使用命令 **evalf** 可以把结果转化为浮点数形式。

看下面的例子：

```
> evalm(abs(B));
```

$$\begin{bmatrix} 3 & 0 & 2 \\ 3 & 2 & 2 \\ 2 & 3 & 3 \end{bmatrix}$$

```
> evalm(exp(B));
```

$$\begin{bmatrix} e^{3} & 1 & e^{2} \\ e^{(-3)} & e^{2} & e^{2} \\ e^{2} & e^{(-3)} & e^{3} \end{bmatrix}$$

```
> evalf(%);
```

$$\begin{bmatrix} 20.08553692 & 1. & 7.389056099 \\ .04978706837 & 7.389056099 & 7.389056099 \\ 7.389056099 & .04978706837 & 20.08553692 \end{bmatrix}$$

当多项式函数作用于矩阵时，Maple V 用矩阵替换其中的自变量，而不是用函数作用于矩阵的每个元素。如果希望多项式函数对矩阵的每个元素作用，需要使用命令 **map**。读者可以比较下面的两个计算结果：

```
> f:=x→x^2-1:
  evalm(f(B));
```

$$\begin{bmatrix} 12 & -6 & 12 \\ -11 & -3 & 4 \\ 21 & -15 & 6 \end{bmatrix}$$

```
> map(f,B);
```

$$\begin{bmatrix} 8 & -1 & 3 \\ 8 & 3 & 3 \\ 3 & 8 & 8 \end{bmatrix}$$

另外，由于矩阵名的特殊性，Maple V 中的许多命令在作用于矩阵时往往得不到预期的效果；如果希望这些命令分别作用于矩阵中的元素，可以借助于命令 **map**。看下面的例子：

```
> F:=matrix([[sin(x),x^2+x+3],[exp(x),
  cos(x^2)]]);
```

$$F := \begin{bmatrix} \sin(x) & x^2+x+3 \\ e^x & \cos(x^2) \end{bmatrix}$$

```
> evalm(diff(F,x));
```

$$0$$

```
> map(diff,F,x);
```

$$\begin{bmatrix} \cos(x) & 2x+1 \\ e^x & -2\sin(x^2)x \end{bmatrix}$$

7. 行列式、逆矩阵

行列式和逆矩阵都是针对正方矩阵而言的，正方矩阵是行数和列数相等的矩阵。在 **linalg** 函数库中，命令 **det** 用以计算正方矩阵的行列式，命令 **inverse** 用以计算非奇异矩阵的逆矩阵，与命令 **evalm(A^(-1))**的作用等价。在使用这两个命令之前需要加载 **linalg** 函数库，用户也可以使用命令 **alias** 简化命令的调用格式。看下面的例子：

```
> alias(inv=linalg[inverse]):
  alias(det=linalg[det]):
> det(A);
```

```
>inv(A);
```

$$\begin{bmatrix} -\frac{73}{183} & -\frac{23}{183} & -\frac{28}{61} \\ \frac{70}{183} & \frac{2}{183} & \frac{21}{61} \\ \frac{43}{183} & -\frac{4}{183} & \frac{19}{61} \end{bmatrix}$$

8. 矩阵的范数

包含于 **linalg** 函数库中的命令 **norm** 用来计算矩阵的范数，基本的格式为 **norm(mat normname)**，其中 **mat** 代表矩阵，**normname** 的取值为 **1**、**2**、**infinity** 或者 **frobenius**，分别对应于矩阵的 1-范数、2-范数、无穷范数和 Frobenius 范数。这个命令和用来计算多项式系数绝对值和的 **norm** 命令是不同的，使用这个命令之前需要加载 **linalg** 函数库；加载 **lianlg** 函数库后的警告信息说明 **norm** 有了新的定义。下面给出矩阵范数计算的例子：

```
>with(linalg):
warning,new definition for norm
warning,new definition for trace
```

```
>norm(A,1);
```

$$22$$

```
>norm(A,infinity);
```

$$18$$

```
>norm(A,'frobenius');
```

$$\sqrt{358}$$

11.2.2　向量的基本运算

在 Maple V 中可以实现向量加法、数乘运算，初等函数也可以对向量作用，在实现这些运算时，通常要借助于命令 **evalm** 的作用计算出结果向量。这些基本运算的法则和矩阵的运算法则是类似的，这里不再举例介绍。本节主要介绍向量的其他基本运算，包括向量的点积、范数计算和标准化等。本节介绍的命令都是 **linalg** 函数库中的命令，在使用之前需要用命令 **with(linalg)**加载 **linalg** 函数库。

1. 向量的点积

向量的点积通常也称为向量的内积，是两个向量对应元素乘积的和，两个向量应具有相同的维数。**linalg** 函数库有两个命令 **dotprod** 和 **innerprod**，都可以计算向量的点积或内积。

命令 **dotprod** 的基本格式为 **dotprod(u, v)**，其中 **u** 和 **v** 是相同维数的向量（或相同长度

的标量元素列表），命令的计算域为复数，对应的运算法则为 **sum(u[i]*conjugate(v[i]), i=1..n)**，其中 **n** 为向量的维数或者列表的长度，conjugate 是取共轭复数的命令。用户也可以使用命令 **dotprod(u, v, 'orthogonal')**的格式，这时的运算法则为 **sum(u[i]*v[i], i=1..n)**。如果 **u** 和 **v** 的元素都是实数，这两种命令格式是等价的。看下面的例子，首先定义两个随机向量 **u** 和 **v**：

```
> u:=linalg[randvector](3,entries=rand(-5..5));
```

$$u := [4,-1,0]$$

```
> v:=linalg[randvector](3,entries=rand(-5..5));
```

$$v := [5,1,5]$$

```
> with(linalg):
warning,new definition for norm
warning,new definition for trace
> dotprod(u,v);
```

$$19$$

```
> a:=[1,I];b:=[I,1];
```

$$a := [1, I]$$

$$b := [I, 1]$$

```
> dotprod(a,b);
```

$$0$$

```
> dotprod(a,b,'orthogonal');
```

$$2I$$

命令 **innerprod** 可以采用与 **dotprod** 类似的命令格式计算向量的点积。另外，命令 **innerprod** 也可以采用 **innerprod(u, A1,A2,...,An, v)**的格式，其中 **u** 和 **v** 代表向量（或标量元素列表），**A1**、**A2**、…、**An** 代表矩阵，这些向量和矩阵之间应该满足矩阵的乘法规则，计算的结果为这些向量和矩阵的乘积，通常也把这种运算称为向量的广义内积。下面是一个广义内积的例子：

```
> L:=[0,1];
```

$$L := [0,1]$$

```
> A:=array([[1,3,5],[2,4,6]]);
```

$$A := \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

```
> innerprod(I,A,u);
```

$$4$$

2. 向量的范数

数学上最常用的向量范数为：

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \quad p \geq 1$$

通常称为向量的 p-范数。**linalg** 函数库中的命令 **norm** 也可以用来计算向量的范数。命令的基本格式为 **norm(v, normname)**，其中：**v** 为给定的向量，**normname** 的取值可以为任意大于 **1** 的实数，对应于上面的 p-范数；**normname** 的取值也可以为 **infinity** 或者 **frobenius**，对应于无穷范数（$p=\infty$）和 F-范数（Frobenius 范数）。向量的 F-范数和 2-范数是等价的，通常也称为向量的模。**norm(v, 2)**或者 **norm(v, frobenius)**和 **sqrt(dotprod(v, v))**的计算结果是相等的。

下面给出范数计算的例子：

```
> norm(v,infinity);
```

$$5$$

```
> norm(u,2);
```

$$\sqrt{17}$$

```
> sqrt(dotprod(u,u));
```

$$\sqrt{17}$$

3. 向量的标准化

标准化向量指的是模为 1 的向量。**linalg** 函数库中的命令 **normalize** 可以把给定的向量转化为标准化向量。命令的格式为 **normalize(v)**，和命令 **evalm(v/norm(v, frobenius))**是等价的。下面的命令分别把向量 **u** 和 **v** 转化为标准化向量：

```
> normalize(u);
```

$$\left[\frac{4}{17}\sqrt{17}, -\frac{1}{17}\sqrt{17}, 0\right]$$

```
> normalize(v);
```

$$\left[\frac{5}{51}\sqrt{51}, \frac{1}{51}\sqrt{51}, \frac{5}{51}\sqrt{51}\right]$$

11.2.3　矩阵的特征值和特征向量

首先介绍有关矩阵特征值和特征向量的几个数学术语。设 A 为 n×n 的实矩阵，v 为非零的 n 维向量，则满足 $A\ v=\lambda v$ 的数λ称为矩阵 A 的特征值，相应的向量 v 称为矩阵 A 的特征向量。如果 I 为 n×n 的单位矩阵，则 $A-\lambda I$ 称为矩阵 A 的特征矩阵；令其行列式值为 0，相应的方程$|A-\lambda I|=0$ 称为矩阵 A 的特征方程，矩阵 A 的特征值是其特征方程的根；该方程左边的关于λ的多项式称为矩阵 A 的特征多项式。由矩阵 A 的特征向量构成的空间通常也称为矩阵 A 的特征空间，可能是低于 n 维的空间。

首先定义两个 3×3 的矩阵 **A** 和 **B**，本小节主要以这两个矩阵为例介绍矩阵特征值和特征向量的计算。本小节介绍的命令包含在 **linalg** 函数库中，在使用之前要加载 **linalg** 函数库。

```
> with(linalg):
  A:=matrix(3,3,[4,-3,4,1,-3,1,-1,-1,-5]
warning,new definition for norm
warning,new definition for trace
```

$$A:=\begin{bmatrix}4 & -3 & 4\\ 1 & -3 & 1\\ -1 & -1 & -5\end{bmatrix}$$

```
> B:=array([[1,5,-1],[-3,6,-6],[-4,1,7]]);
```

$$B:=\begin{bmatrix}1 & 5 & -1\\ -3 & 6 & -6\\ -4 & 1 & 7\end{bmatrix}$$

1. 矩阵的特征矩阵和特征多项式

linalg 函数库的命令 **charmat** 和 **charpoly** 分别用来计算矩阵的特征矩阵和特征多项式。基本的命令格式分别为 **charmat(mat, lambda)**和 **charpoly(mat, lambda)**，其中 **mat** 代表行数和列数相等的正方实矩阵，**lambda** 代表特征值变量。

看下面的计算结果：

```
> polyA:=charpoly(A,x);
```

$$polyA:=x^3+4x^2-9x-36$$

```
> charmat(A,lambda);
```

$$\begin{bmatrix}\lambda-4 & 3 & -4\\ -1 & \lambda+3 & -1\\ 1 & 1 & \lambda+5\end{bmatrix}$$

2. 矩阵的特征值

使用命令 **charpoly** 得到矩阵的特征多项式后，通过求解相应的特征方程可以得到矩阵的特征值。下面的例子中，使用命令 **solve** 求解矩阵 **A** 的特征方程 **polyA=0**，得到了矩阵 **A** 的三个特征值**-4**、**-3**、**3**：

```
> solve(polyA);
                          -4,-3,3
```

同时，Maple V 在 **linalg** 函数库中提供了命令 **eigenvals**，可以直接计算矩阵的特征值。命令的基本格式为 **eigenvalues(mat)**，其中 **mat** 代表正方矩阵，命令输出由矩阵 **mat** 的特征

值构成的序列。下面用命令 **eigenvals** 计算矩阵 **A** 的特征值，和求解特征方程得到的结果是一致的：

```
> eigenvals(A);
```

$$-4,-3,3$$

通常，命令 **eigenvals** 能够以准确的形式给出矩阵的特征值，甚至对符号矩阵也能给出特征值的符号表达式；矩阵的元素中有浮点数，命令 **eigenvals** 的计算结果也用浮点数来表示，精度由环境变量 **Digits** 确定。看下面的几个例子：

```
> C:=array(1..2,1..2);
```

$$C := \text{array}(1..2,1..2,[\ ])$$

```
> eigenvals(C);
```

$$\frac{1}{2}C_{1,1}+\frac{1}{2}C_{2,2}+\frac{1}{2}\sqrt{C_{1,1}2-2C_{1,1}C_{2,2}2+C_{2,2}2+4C_{2,1}C_{1,2}},\ \frac{1}{2}C_{1,1}+\frac{1}{2}C_{2,2}-\frac{1}{2}\sqrt{C_{1,1}2-2C_{1,1}C_{2,2}2+C_{2,2}2+4C_{2,1}C_{1,2}}$$

```
> eigenvals(array([[1,2],[3,4]]));
```

$$\frac{5}{2}+\frac{1}{2}\sqrt{33},\frac{5}{2}-\frac{1}{2}\sqrt{33}$$

```
> eigenvals(array([[1.0,2],[3,4]]));
```

$$-.3722813233,5.372281323$$

矩阵特征值的计算和多项式方程的求解相关，对于低于 5 阶的矩阵来说，命令 **eigenvals** 通常都可以给出特征值的显式结果；但当矩阵阶数高于 4 时，Maple V 可能无法给出所有特征值的显式结果，因此无法给出所有的特征值。用户可以在命令 **eigenvals** 中加入 **implicit** 选项，即采用 **eigenvals(mat, 'implicit')**的命令格式，输出结果将采用 **RootOf** 表达式的形式表示，与命令 **allvalues** 结合即可以得到矩阵 **mat** 的所有特征值。看下面的例子：

```
> eigenvals(C,'implicit');
```

$$\text{RootOf}(\_Z^2+(-C_{1,1}-C_{2,2})\_Z+C_{1,1}C_{2,2}-C_{2,1}C_{1,2})$$

```
> eigenvals(B,'implicit');
```

$$\text{RootOf}(-252+72\_Z-14\_Z^2+\_Z^3)$$

命令 **Eigenvals** 是 **eigenvals** 的惰性格式，Maple V 启动后自动加载 **Eigenvals** 命令，因此使用前不需要加载 **linalg** 函数库；同时，选项**'implicit'**是不起作用的。与命令 **evalf** 结合，命令 **Eigenvals** 可以得到矩阵的所有特征值的近似值，但矩阵中不能含有未知变量。命令的格式为 **evalf(Eigenvals(mat))**，返回的结果是由矩阵的所有特征值构成的列表。看下面的例子：

```
> evalf(Eigenvals(B));
```

$$[2.430581441 + 4.654771544\,I, 2.430581441 - 4.654771544\,I, 9.138837126]$$

另外，命令 **eigenvals** 也可以求解广义的特征值问题，广义特征值满足的方程为$|\lambda A-B|=0$。命令的格式为 **eigenvals(A, B)**，具体的使用方法这里不再举例。

3. 矩阵的特征向量

矩阵的特征向量和矩阵的特征值是成对出现的，惰性命令 **Eigenvals** 还有一种命令格式，可以同时得到矩阵的特征值和特征向量。用户可以采用下面的命令格式：

mat_vals := Eigenvals(mat , 'mat_vecs')

命令执行后，用户可以使用命令 **evalf(mat_vals)**得到矩阵 **mat** 的特征值；其中的参数 **mat_vecs** 返回由矩阵 **mat** 的特征向量构成的矩阵，矩阵 **mat_vecs** 的元素都用浮点数表示，每一列对应着列表 **mat_vals** 中相应下标处特征值的特征向量。

下面是矩阵 **A** 的特征值和特征向量的计算结果，并在最后验证了矩阵 **A** 的第一个特征值和相应的特征向量所满足的条件 $Av=\lambda v$。

```
> vals_A:= Eigenvals(A,'vecs_A');
```

$$vals\_A := \text{Eigenvals}\left(\begin{bmatrix} 4 & -3 & 4 \\ 1 & -3 & 1 \\ -1 & -1 & -5 \end{bmatrix}, vecs\_A\right)$$

```
> print(vecs_A);
```

$$\begin{bmatrix} .9899494936 & 1.178511302 & -1.000000000 \\ .1414213562 & 1.178511302 & -.5714285715 \\ -.1414213562 & -1.178511302 & 1.571428572 \end{bmatrix},$$

```
> vals_A:= evalf(vals_A);
```

$$vals\_A := [3.000000000, -3.000000000, -4.000000000]$$

```
> v1:= linalg[submatrix](vecs_A,1..3,1..1);
```

$$v1 := \begin{bmatrix} .9899494936 \\ .1414213562 \\ -.1414213562 \end{bmatrix}$$

```
> evalm(A&*v1)= evalm(vals_A[1]*v1);
```

$$\begin{bmatrix} 2.969848480 \\ .4242640688 \\ -.4242640690 \end{bmatrix} = \begin{bmatrix} 2.969848481 \\ .4242640686 \\ -.4242640686 \end{bmatrix}$$

同时，Maple V 在 **linalg** 函数库中提供了命令 **eigenvects**，用来计算矩阵的特征值和特征向量。命令 **eigenvects** 的格式和命令 **eigenvals** 类似，也可以使用 **implicit** 选项，输出结果以序列的形式同时给出矩阵的特征值和特征向量，序列的每个元素具有类似**[ei, mi, {v1, v2, ...}]**的列表形式；其中 **ei** 代表矩阵的特征值，**mi** 代表特征值 **ei** 的重数，集合中的元素 **v1**、**v2**、…代表对应于特征值 **ei** 的特征向量。

下面用命令 **eigenvects** 计算矩阵 **A** 的特征值和特征向量，可以从计算结果中分离出矩阵的特征值、特征值的重数和相应的特征向量。看下面的结果，读者可以与前面的结果进行比较：

```
> eig := eigenvects(A);
```

$$eig := \left[-4,1,\left\{\left[1,\frac{4}{7},\frac{-11}{7}\right]\right\}\right],[-3,1,\{[-1,-1,1]\}],[3,1,\{[-7,-1,1]\}]$$

```
> eig_vals:=[eig[i][1]$i=1..3];#矩阵A的特征值
```

$$eig\_vals := [-4,-3,3]$$

```
> eig_valm:=[eig[i][2]$i=1..3];#矩阵A特征值的重数
```

$$eig\_valm := [1,1,1]$$

```
> eigvec1:=eig[3][3][1];#对应于特征值3的特征向量
```

$$eigvec1 := [-7,-1,1]$$

```
> type(eigvec1,vector);
```

$$true$$

命令 **eigenvects** 也可以计算简单符号矩阵的特征值和特征向量，这里不再举例。

第 12 章 微 分 方 程

微分方程是指含有未知函数的导数或微分的方程，包括常微分方程和偏微分方程两大类。和代数方程求解未知数不同，微分方程的求解是要寻找满足微分方程及定解条件的函数。微分方程的求解是科学计算中一个相当复杂的课题。

Maple V 提供了功能强大的工具用于微分方程的求解。源于其强大的符号计算能力，Maple V 能够给出许多常见微分方程的分析解；同时，Maple V 也能够使用各种数值方法获得微分方程的数值解。Maple V 提供的求解工具是很多的，这里不能一一介绍，本章将介绍其中的一些基本求解工具。

常微分方程和偏微分方程是两类不同的微分方程，所用的求解方法也是不尽相同的，偏微分方程的求解相对更为复杂。本章主要介绍常微分方程的求解，并简单介绍偏微分方程的求解方法。

12.1 常微分方程

常微分方程是关于一元函数的微分方程，和偏微分方程相比是一种相对简单的微分方程。Maple V 提供了功能强大的求解工具 **dsolve**，用于常微分方程及常微分方程组的求解，同时在 **DEtools** 函数库中提供了许多其他的辅助命令用于微分方程的求解。用户可以使用这些工具获得常见的常微分方程和常微分方程组及给定定解条件的定解问题的分析解和数值解。

本节首先介绍微分方程的表示方法，接着介绍命令 **dsolve** 及其辅助命令的使用方法，主要讨论一阶常微分方程、高阶常微分方程、常微分方程组及定解问题的分析解，最后简单介绍常微分方程的数值解。

12.1.1 微分方程的表示

微分方程是关于未知函数及其导数的方程。在 Maple V 中，通常要用 **y(x)**的格式表示微分方程中的未知函数，而不能简单的表示为 **y**；同时，与两个微分的命令 **diff** 和 **D** 对应，可以用 **diff(y(x),x)**和 **D(y)(x)**两种方法来表示数学上的微分 **y'**。函数 **y(x)**的高阶微分也可以有 **diff** 和 **D** 两种表示法。在下面的例子中，同一常微分方程有两种不同的表示方法：

```
> Dode:=D(y)(x)=sqrt(y(x)^2+1);
```

$$Dode := \mathrm{D}(y)(x) = \sqrt{y(x)^2 + 1}$$

```
> diffode:=diff(y(x),x)=sqrt(y(x)^2+1);
```

$$diffode := \frac{\partial}{\partial x} y(x) = \sqrt{y(x)^2 + 1}$$

用户可以用命令 **type(equ, `ODEtools/ODE`)**来检验给定的微分等式 **equ** 是否常微分方程，或者用命令 **type(equ,`PDEtools/PDE`)**检验给定的微分等式是否偏微分方程。常微分方程和偏微分方程的类型名要用反引号界定，因为其中包含了斜杠。

```
> type(Dode,`ODEtools/ODE`),
  type(diffode,`ODEtools/ODE`);
```

$$true, true$$

由于命令 **D** 直接对函数的映射法则作用，与函数的自变量无关，所以在表示微分方程时具有一定的模糊性，可能引发意外的错误，因此这里建议读者用命令 **diff** 来表示微分方程。

另外，用户也可以用命令 **alias（y=y(x)）** 把 **y** 等价为关于自变量 **x** 的函数 **y(x)**，这时可以用 **diff(y, x)**或者 **D(y)**来表示未知函数 **y(x)**的微分 **y'**，即：

```
> alias(y=y(x)):
> diffode:=diff(y,x)=sqrt(y^2+1);
```

$$diffode := \frac{\partial}{\partial x} y = \sqrt{y^2 + 1}$$

12.1.2　一阶常微分方程的分析解

一阶常微分方程具有 F(y, y',x)=0 的形式，是最简单的常微分方程。其中 y 代表以 x 为自变量的未知函数，y'代表函数 y(x)对自变量 x 的一阶导数。一阶常微分方程，包括可分离型微分方程、齐次微分方程、线性微分方程和全微分方程等，通常都可以得到准确的分析解。本小节通过介绍一阶常微分方程的求解，介绍命令 **dsolve** 的基本格式及输出结果的表示。

命令 **dsolve** 的基本格式为：**dsolve(ODE, y(x))**，其中：**ODE** 代表单个的常微分方程；**y(x)** 给定了要求解的未知函数，如果方程 **ODE** 中只包含一个未知函数，则 **y(x)**可以缺省。默认情况下，命令的输出结果以 **y(x)=F(x, _Cn)**的显函数格式给出未知函数的求解结果，其中系统变量**_Cn**（**n=1**，**2**，…）代表任意的常数，对应于任意常数**_C1** 的不同值，所得到的求解结果代表了一系列的函数。如果难以从求解结果中分离出因变量，或者因变量的分离需要用分数幂的形式表示，或者因变量只能用 **RootOf** 的形式给出，Maple V 采用 **F(y(x),x, _Cn)=0** 的隐函数格式表示未知函数的求解结果。和命令 **solve** 一样，用户也可以设置环境变量 **_EnvExplicit** 的值强制命令 **dsolve** 输出显式的求解结果。

下面给出几个微分方程求解的例子：

```
> ODE:=diff(y(x),x)=k*x;
```

$$ODE := \frac{\partial}{\partial x} y(x) = kx$$

```
> dsolve(ODE);
```

$$y(x)=\frac{1}{2}kx^2+\_CI$$

```
> ODE1:=x*diff(y(x),x)=y(x)*ln(x*y(x))-y(x);
```

$$ODE1:=x\left(\frac{\partial}{\partial x}y(x)\right)=y(x)\ln(x\,y(x))-y(x)$$

```
> dsolve(ODE1);
```

$$y(x)=\frac{e^{(xe^{(-\_CI)})}}{x}$$

同时，用户也可以在命令 **dsolve** 中给定 **explicit** 选项，规定在任何情况下都以显函数格式表示求解的结果；或者给定 **implicit** 选项，规定命令 **dsolve** 的求解结果以隐函数格式表示。以前面定义的常微分方程 **ODE1** 为例，下面是以隐函数格式表示的求解结果：

```
> dsolve(ODE1,implicit);
```

$$\ln(x)-\_CI-\ln(\ln(x\,y(x)))=0$$

如果有多个函数满足一阶常微分方程，在计算结果中将以序列的形式表示。下面给出一个这样的例子：

```
> ODE2:=x*(x+1)*diff(y(x),x)=y(x)^3;
```

$$ODE2:=x(x+1)\left(\frac{\partial}{\partial x}y(x)\right)=y(x)^3$$

```
> dsolve(ODE2,y(x),implicit);
```

$$\frac{1}{y(x)^2}+2\ln(x)-2\ln(x+1)-\_CI=0$$

```
> dsolve(ODE2,y(x));
```

$$y(x)=\frac{1}{\sqrt{\_CI+2\ln\left(\frac{x+1}{x}\right)}},\quad y(x)=-\frac{1}{\sqrt{\_CI+2\ln\left(\frac{x+1}{x}\right)}}$$

在求解微分方程的过程中，如果 Maple V 不能给出准确的积分结果或者用使用积分表示更为方便，在命令 **dsolve** 的输出结果中将出现由积分符号表示的函数形式。这里的积分符号通常以黑色显示，对应于惰性命令 **Int** 的输出形式，其中的积分变量用**_a** 等系统变量来表示。用户可以用命令 **value** 等得到积分完成后的结果。同时，如果命令 **dsolve** 不能消除求解过程中用到的参数，求解的结果可能以参数方程的形式表示，这时函数的因变量和自变量包含于同一个列表中，其中的参数用系统变量**_T** 来表示。看下面的例子：

```
> ODE2 := y(x)/x = F(diff(y(x),x));
```

$$ODE2 := \frac{y(x)}{x} = F\left(\frac{\partial}{\partial x} y(x)\right)$$

```
> dsolve(ODE2);
```

$$y(x) = F(\text{RootOf}(F(\_Z) - \_Z))\, x, \left[x(\_T) = e^{\left(\int \frac{\frac{\partial}{\partial \_T} F(\_T)}{\_T - F(\_T)} d\_T\right)} \_C1,\ y(\_T) = F(\_T)\, e^{\left(\int \frac{\frac{\partial}{\partial \_T} F(\_T)}{\_T - F(\_T)} d\_T\right)} \_C1\right]$$

对于一阶常微分方程来说，用户也可以在命令 **dsolve** 中同时给定 **parametric** 和 **implicit** 选项，限定求解的结果以参数方程的形式表示。由于命令 **dsolve** 会尽力消除求解过程中用到的参数，因此只给定 **parametric** 选项时求解的结果可能同样会以显函数的格式表示。看下面的例子：

```
> diffode := diff(y(x),x) = sqrt(y(x)^2 + 1);
```

$$diffode := \frac{\partial}{\partial x} y(x) = \sqrt{y(x)^2 + 1}$$

```
> dsolve(diffode, parametric);
```

$$y(x) = \frac{1}{2}\sqrt{-2 + e^{(-2x+2\_C1)} + e^{(2x-2\_C1)}},\quad y(x) = -\frac{1}{2}\sqrt{-2 + e^{(-2x+2\_C1)} + e^{(2x-2\_C1)}}$$

```
> dsolve(diffode, implicit, parametric);
```

$$[y(\_T) = \sqrt{-1 + \_T^2}, x(\_T) = \ln(\_T + \sqrt{(\_T - 1)(\_T + 1)}) + \_C1],\quad [y(\_T) = -\sqrt{-1 + \_T^2}, x(\_T) = -\ln(\_T + \sqrt{(\_T - 1)(\_T + 1)}) + \_C1]$$

在命令中加入 **type=series** 选项，可以规定命令 **dsolve** 寻找常微分方程的级数解，即使用 **dsolve(ODE, y(x), type=series)**的格式。求解的结果属于 Maple V 中的级数数据类型，级数通常是在原点处进行展开的。下面给出一个级数解的例子：

```
> ODE3:=diff(y(t),t,t)+diff(y(t),t(^2=0;
```

$$ODE3 := \left(\frac{\partial^2}{\partial t^2} y(t)\right) + \left(\frac{\partial}{\partial t} y(t)\right)^2 = 0$$

```
> dsolve(ODE3,y(t),type=series);
```

$$y(t) = y(0) + D(y)(0)t - \frac{1}{2}D(y)(0)^2 t^2 + \frac{1}{3}D(y)(0)^3 t^3 - \frac{1}{4}D(y)(0)^4 t^4 + \frac{1}{5}D(y)(0)^5 t^5 + O(t^6)$$

12.1.3 两个辅助的命令

本小节介绍微分方程求解中两个常用的辅助命令，命令 **DESol** 是 Maple V 表示微分方程解的一种特殊方式，命令 **odetsest** 用来验证命令 **dsolve** 所得到的求解结果是否满足原来的微分方程。

1. DESol

和命令 **RootOf** 是代数方程的解的一种表示法一样，**DESol** 是 Maple V 中微分方程的解的一种表示法。命令 **DESol** 使得用户可以很方便地对微分方程中的未知函数进行操作，包括 **diff**、**int**、**evalf**、**series**、**simplify** 等命令在内的许多 Maple V 命令都可以对 DESol 表达式直接作用。

用户可以直接使用命令 **DESol** 表示微分方程的解，基本的命令格式为 **DESol(DE, y(x))**，其中 **DE** 为关于未知函数 **y(x)**的微分方程，可以使用命令 **D** 和 **diff** 两种方式表示；当微分方程 **DE** 中只有一个未知函数时，**y(x)**可以缺省。下面给出两个以 **DESol** 表示的微分方程的解：

```
> de1:=DESol(D(y)-y,y);
```

$$de1 := \mathrm{DESol}(\{D(y) - y\}, \{y\})$$

```
> de2:=DESol(diff(y(x),x)-y(x),y(x));
```

$$de2 := \mathrm{DESol}\left(\left\{\left(\frac{\partial}{\partial x} y(x)\right) - y(x)\right\}, \{y(x)\}\right)$$

```
> de1(x)-de2;
```

$$0$$

```
> diff(de2,x)-de2;
```

$$0$$

对于线性常微分方程来说，如果命令 **dsolve** 不能够找到满足微分方程的函数，将以 **DESol** 的格式表示求解的结果。这也可以作为 Maple V 表示复杂函数的一种隐函数格式，虽然这

不能提供比原来的常微分方程更多的信息，但用户可以很方便地使用微分方程中的未知函数。读者可以用命令**?DESol** 获得命令 **DESol** 的更多信息。

2. odetest

在微分方程的求解过程中可能会产生伪解，命令 **odetest** 可以检验命令 **dsolve** 得到的求解结果是否原微分方程的解，所用的方法是把得到的解代入微分方程中并化简。

命令的基本格式为 **odetest(sol,ODE,y(x))**，其中：**sol** 代表所检验的求解结果，可以是显函数的形式，也可以是隐函数的形式；**ODE** 代表原来的常微分方程；**y(x)**代表所求解的未知函数，当微分方程 **ODE** 中只有一个未知函数时，可以缺省。如果所检验的解是原微分方程的解，命令的返回结果为 **0**，否则将返回所得到的余项表达式。如果返回的结果不是 **0**，并不能由此肯定所验证的解不是原微分方程的解，用户可以使用命令 **expand**、**combine** 等命令对余项表达式进一步化简。

下面给出两个微分方程求解和检验的例子：

```
> ode1:=diff(y(x),x)-y(x)^2+y(x)*sin(x)-cos(x);
```

$$ode1 := \left(\frac{\partial}{\partial x}y(x)\right) - y(x)^2 + y(x)\sin(x) - \cos(x)$$

```
> sol1:=dsolve(ode1);
```

$$sol1 := y(x) = \sin(x) - \frac{e^{(-\cos(x))}}{\_C1 + \int e^{(-\cos(x))}dx}$$

```
> odetest(sol1,ode1);
```

$$0$$

```
> ode2:=diff(y(x),x)=F((y(x)-x*ln(x))/x)+ln(x);
```

$$ode2 := \frac{\partial}{\partial x}y(x) = F\left(\frac{y(x) - x\ln(x)}{x}\right) + \ln(x)$$

```
> sol2:=dsolve(ode2,implicit)
```

$$sol2 := \ln(x) + \int^{\frac{y(x)}{x}-\ln(x)} \frac{1}{\_a + 1 - F(\_a)}d\_a - \_C1 = 0$$

```
> odetest(sol2,ode2);
```

$$0$$

12.1.4　高阶常微分方程的分析解

高阶常微分方程是关于未知函数及其高阶微分的方程，在科学计算和工程应用中经常遇到，具有 $F(y, y', y'', \ldots, y^{(n)}, x)=0$ 的形式。其中，y 是以 x 为自变量的函数，y'、y''、...、$y^{(n)}$分

别代表函数 y(x)的一阶微分、二阶微分及高阶微分。

在 Maple V 中，命令 **dsolve** 也可以用来求解高阶常微分方程，在很多情况下可以得到高阶常微分方程的分析解。同时，也可以使用命令 **odetest** 检验所得到的求解结果是否是原微分方程的解。下面给出几个高阶常微分方程的求解结果，并使用命令 **odetest** 验证了解的正确性：

```
> hode1:=diff(y(x),x,x)=
        x^n*n*(n-1+x^n*n-cos(x)*x)/x^2*y(x)
              +cos(x)*diff(y(x),x);
```

$$hode2 := \frac{\partial^2}{\partial x^2} y(x) = \frac{x^n n(n-1+x^n n-\cos(x)x)y(x)}{x^2} + \cos(x)\left(\frac{\partial}{\partial x} y(x)\right)$$

```
> sol1:=dsolve(hode1);
```

$$sol1 := y(x) = \left(\int \frac{e^{\sin(x)} \_C1}{\left(e^{(x^n)}\right)^2} dx + \_C2\right) e^{(x^n)}$$

```
> odetest(sol1,hode1);
```

$$0$$

```
> alias(y=y(x)):
> hode2:=9*diff(y,x$4)-6*diff(y,x$3)+
        46*diff(y,x$2)-6*diff(y,x)+37*y=0
```

$$hode2 := 9\left(\frac{\partial^4}{\partial x^4} y\right) - 6\left(\frac{\partial^3}{\partial x^3} y\right) + 46\left(\frac{\partial^2}{\partial x^2} y\right) - 6\left(\frac{\partial}{\partial x} y\right) + 37y = 0$$

```
> sol2:=dsolve(hode2);
```

$$sol2 :=$$

$$y = \_C1\sin(x) + \_C2\cos(x) + \_C3e^{\left(\frac{1}{3}x\right)}\cos(2x) + \_C4e^{\left(\frac{1}{3}x\right)}\cos(2x)$$

```
> odetest(sol2,hode2);
```

$$0$$

同时，在很多情况下，命令 **dsolve** 并不能成功地得到高阶常微分方程的分析解。这时 Maple V 会力图降低微分方程的阶数，这时给出的结果以命令 **ODESolStruc** 的格式表示，这个结果使得用户可以借助于其他工具如级数展开等进行更为深入的处理，如果得到了降阶后的微分方程的解，用户可以用命令 **DEtools** 函数库中的命令 **buildsol** 获得原微分方程的解。

12.1.5　常微分方程组

命令 **dsolve** 不仅可以求解单个的常微分方程，而且可以求解许多线性和非线性的常微分方程组。常微分方程组和代数方程组一样，用集合的形式表示，因此常微分方程组求解的基本命令格式为 **dsolve({sysODE}, {funcs})**。其中的两个参数都以集合的形式表示，**sysODE** 代表常微分方程组的多个方程序列，**funcs** 代表所求解的多个未知函数序列。下面分别介绍线性常微分方程组和非线性常微分方程组的求解情况。

1. 线性常微分方程组的求解

线性常微分方程组的求解比较简单，命令的输出结果以集合的形式表示。下面给出一个线性常微分方程组求解的例子：

```
> lsys:={diff(x(t),t)=y(t),diff(y(t),t)=-x(t)};
```

$$lsys := \{\frac{\partial}{\partial t}\mathrm{y}(t) = -\mathrm{x}(t), \frac{\partial}{\partial t}\mathrm{x}(t) = \mathrm{y}(t)\}$$

```
> sol1:=dsolve(lsys,{x(t),y(t)});
```

$$sol1 := \{\mathrm{x}(t) = \cos(t)\_C1 + \sin(t)\_C2, \mathrm{y}(t) = -\sin(t)\_C1 + \cos(t)\_C2\}$$

用户也可以用命令 **odetest** 检验所得解的正确性，由于集合的元素互不相同，因此如果所得到的解是原微分方程组的解，则命令 **odetest** 的输出结果是只包含 **0** 的集合。看下面的结果：

```
> odetest(sol1,lsys);
```

$$\{0\}$$

2. 非线性常微分方程组的求解

非线性常微分方程组的求解是比较复杂的，因为通常所求解的多个未知函数是相互耦合的。命令 **dsolve** 在求解非线性常微分方程组时，总是试图把原来的非线性常微分方程组分解为几个尽可能简单的子方程组或者是单个常微分方程，各个子方程组或常微分方程之间没有耦合关系，然后对各个子方程组的常微分方程进行分析求解。如果成功地把原来的常微分方程组分解为相互独立的常微分方程，这意味着通过求解各个常微分方程即可以得到原方程组的解；如果其中的一个常微分方程不能求解，Maple V 直接求解下一个常微分方程，所以在最后的输出结果中可能仍旧包含没有求解的常微分方程。

为了加速非线性常微分方程组的求解过程，通常非线性常微分方程组的求解结果以集合的序列形式表示，其中的每个序列中包含了其中某一个未知函数的所有解，后面一个集合中的未知函数的解中可能包含前一个集合中的未知函数。这时不能用命令 **odetest** 来检验所得解的正确性。看下面的例子：

```
> nlsys:={diff(f(x),x)=cos(f(x)),
          diff(g(x),x)=-f(x)^(1/2),
          diff(h(x),x,x)=g(x)/f(x)};
```

$$nlsys := \left\{\frac{\partial}{\partial x}g(x) = -\sqrt{f(x)}, \frac{\partial}{\partial x}f(x) = \cos(f(x)), \frac{\partial^2}{\partial x^2}h(x) = \frac{g(x)}{f(x)}\right\}$$

```
> sol2:=dsolve(nlsys,{f,g,h});
```

$$sol2 := \left\{f(x) = \arctan\left(\frac{e^{(2x+2\_C4)}-1}{e^{(2x+2\_C4)}+1}, 2\frac{e^{(x+\_C4)}}{e^{(2x+2\_C4)}+1}\right)\right\},$$
$$\{g(x) = \int -\sqrt{f(x)}dx + \_C3\}, \left\{h(x) = \int\int \frac{g(x)}{f(x)}dx + \_C1dx + \_C2\right\}$$

```
> odetest(sol2,nlsys);
Error,odetest expects its 2nd argument,ODE,to be of type
{set(`ODEtools/ODE'),`ODEtools/ODE`},but received{g(x)=
Int(-f(x)^(1/2),x)+_C3}
```

如果希望求解结果中各个未知函数表达式不包含其他的未知函数，用户可以在命令 **dsolve** 中给定 **explicit** 选项，即使用 **dsolve({sysODE}, {funcs}, 'explicit')**的命令格式，这时的求解结果把各个未知函数的表达式包含在同一个集合中。看下面的例子：

```
> sol3:=dsolve(nlsys,{f,g,h},explicit);
```

$$sol3 := \left\{h(x) = \int\int \frac{\int -\sqrt{\arctan\left(\frac{e^{(2x+2\_C4)}-1}{e^{(2x+2\_C4)}+1}, 2\frac{e^{(x+\_C4)}}{e^{(2x+2\_C4)}+1}\right)}dx + \_C3}{\arctan\left(\frac{e^{(2x+2\_C4)}-1}{e^{(2x+2\_C4)}+1}, 2\frac{e^{(x+\_C4)}}{e^{(2x+2\_C4)}+1}\right)}dx + \_C1dx + \_c2,\right.$$
$$g(x) = \int -\sqrt{\arctan\left(\frac{e^{(2x+2\_C4)}-1}{e^{(2x+2\_C4)}+1}, 2\frac{e^{(x+\_C4)}}{e^{(2x+2\_C4)}+1}\right)}dx + \_C3,$$
$$\left.f(x) = \arctan\left(\frac{e^{(2x+2\_C4)-1}}{e^{(2x+2\_C4)}+1}, 2\frac{e^{(x+\_C4)}}{e^{(2x+2\_C4)}+1}\right)\right\}$$

```
> odetest(sol3,nlsys);
                                  {0}
```

12.1.6 定解问题的解

在微分方程的求解结果中通常包含代表任意常数的系统变量**_Cn**（**n=1**，**2**，…），这样

的解称为微分方程的通解。在很多时候需要给出其他一些条件，以确定通解中的任意常数，这些条件称为定解条件，满足定解条件的解称为特解。给定了定解条件的微分方程的求解问题，通常也称为定解问题。

在命令 **dsolve** 中同时给出微分方程和定解条件，即可求解对应的定解问题。积分变换是求解定解问题的一种常用方法。本小节主要介绍定解问题的基本求解格式，同时简单介绍常微分方程的积分变换求解。

1. 基本求解格式

求解定解问题的基本命令格式为 **dsolve({ODE, ICs}, y(x))**，其中 **ODE** 为所求解的关于未知函数 **y(x)**的微分方程；**ICs** 为定解条件，给出了未知函数 **y(x)**或相应的微分在给定的点上满足的条件，定解条件的个数应该少于微分方程的阶数。下面给出一个定解问题的解，由于微分方程是二阶的，只给出了一个定解条件，因此在求解结果中仍然包含系统变量**_C1**：

```
> ode1:=diff(y(t),t,t)+diff(y(t),t)^2=0;
```

$$ode1 := \left(\frac{\partial^2}{\partial t^2} y(t)\right) + \left(\frac{\partial}{\partial t} y(t)\right)^2 = 0$$

```
> sol11:=dsolve({ode1,y(0)=3},y(t));
```

$$sol11 := y(t) = \ln(t + \_C1) - \ln(\_C1) + 3$$

在定解条件中，未知函数的导数经常用命令 **D** 来表示，同时也可以表示为命令 *diff* 的形式。定解条件中的特定点可以是符号量，例如 **diff(y(a),a)=1** 表示未知函数 **y(x)**在 **x=a** 这一点的导数必须为 **1**。所给的定解条件也可以是非线性的。看下面的计算结果：

```
> sol12:=dsolve({ode1,y(0)=3,D(y)(0)=0},y(t));
```

$$sol12 := y(t) = 3$$

```
> sol13:=dsolve({ode1,diff(y(a),a)=A},y(t));
```

$$sol13 := y(t) = \ln\left(t - \frac{-1 + A\,a}{A}\right) + \_C2$$

```
> sol14:=dsolve({ode1,diff(y(a),a)-y(b)=A,
                 exp(y(b))=B},y(t));
```

$$sol14 := y(t) = \ln\left(t - \frac{-1 + \ln(B)\,a + A\,a}{\ln(B) + A}\right) - \ln\left(-\frac{-b\ln(B) - b\,A - 1 + \ln(B)\,a + A\,a}{\ln(B) + A}\right) + \ln(B)$$

下面的结果则表明上面得到的四个解都是微分方程 **ode1** 的解：

```
> map(odetest,[sol11,sol12,sol13,sol14],ode1);
                        [0,0,0,0]
```

如果求解的是常微分方程组的定解问题，需要采用 **dsolve({sysODE,ICs}, {funcs})**的格

式。其中，**sysODE** 代表要求解的常微分方程组，**ICs** 代表给出的定解条件，**funcs** 代表要求解的未知函数；**sysODE** 和 **ICs** 必须包含在同一个集合中。这时的每个定解条件中可以同时包含多个未知函数在特定点的耦合关系。看下面的例子：

```
> sys:={diff(y(t),t)=-x(t),diff(x(t),t)=y(t)};
```

$$sys := \{\frac{\partial}{\partial t} y(t) = -x(t), \frac{\partial}{\partial t} x(t) = y(t)\}$$

```
> IC_1:={x(a)=A,diff(x(a),a)=B};
```

$$IC\_1 := \{\frac{\partial}{\partial a} x(a) = B, x(a) = A\}$$

```
> dsolve(sys union IC_1,{x(t),y(t)});
```

$$\{y(t) = -\sin(t)(-B\sin(a) + \cos(a)A) + \cos(t)(\cos(a)B + A\sin(a)),$$
$$x(t) = \cos(t)(-B\sin(a) + \cos(a)A) + \sin(t)(\cos(a)B + A\sin(a))\}$$

在上面的命令中用到了集合的并集运算符 **union**，用来把常微分方程组和定解条件合并为一个集合。读者可以用命令 **combine** 把求解的结果进行化简：

```
> sol1:=
    combine(dsolve(sys union IC_1,{x(t),y(t)}),trig);
```

$$sol1 := \{x(t) = B\sin(t-a) + A\cos(t-a), y(t) = B\cos(t-a) - A\sin(t-a)\}$$

```
> odetest(sol1,sys);
```

$$\{0\}$$

2. 用积分变换法求解

积分变换既可以计算微分方程的通解，也可以计算定解问题的特解，但更多地用在定解问题的求解中。在命令 **dsolve** 中加入 **methods=transform** 选项，从而规定用积分变换的方法求解给定的常微分方程或者定解问题。其中 **transform** 代表 **laplace**、**fourier**、**fouriercos**、**fouriersin** 等几种积分变换方法中的一种。看下面的例子：

```
> ode2:=diff(y(t),t$2)+5*diff(y(t),t)+6*y(t)=0;
```

$$ode2 := \left(\frac{\partial^2}{\partial t^2} y(t)\right) + 5\left(\frac{\partial}{\partial t} y(t)\right) + 6\,y(t) = 0$$

```
> dsolve({ode2},y(t),
        method=laplace);
```

$$y(t) =$$
$$-2e^{(-3t)}y(0) - e^{(-3t)}D(y)(0) + 3e^{(-2t)}y(0) + e^{(-2t)}D(y)(0)$$

```
>dsolve({ode2,y(0)=0,D(y)(0)=1},y(t),
        method=laplace);
```

$$y(t) = -e^{(-3t)} + e^{(-2t)}$$

应该注意，由于积分变换是有条件的，因此并不是所有的微分方程都可以用积分变换的方法来求解。

另外，求解定解问题时，如果规定了求解的方法为积分变换法或者规定了求解的结果为数值解、级数解，定解条件中的所有导数必须表达为命令 D 的形式，并且每个定解条件只能与一个未知函数有关，所有的定解条件必须是未知函数或者其导数的线性形式。

12.1.7　常微分方程的数值解

常微分方程的求解是个非常复杂的课题，很多情况下都难以找到常微分方程和常微分方程组的分析解，数值解提供了研究问题的另一种方法。

在 Maple V 中，命令 **dsolve** 也可以用数值方法获得常微分方程的解，只需在命令中给出 **numeric** 选项即可。命令的基本格式为：**dsolve(deqns, funcs, numeric)**或者 **dsolve(deqns, funcs, type=numeric)**，其中 **deqns** 代表由常微分方程或常微分方程组即定解条件构成的集合，**funcs** 代表所求解的未知函数或者多个未知函数构成的集合。Maple V 自动选择合适的数值方法计算常微分方程的解，默认情况下以过程的形式给出函数的求解结果。

下面给出一个简单的例子：

```
>deq:=diff(y(x),x)=sin(2*x-y(x)):
>sol:=dsolve({deq,y(0)=0.5},y(x),numeric);
```

$$sol := \mathbf{proc}(rkf45\_x)..\mathbf{end}$$

结果中的 **rkf45_x** 表明了求解时所用的数值方法为 **rkf45**。求解的结果定义了一个与未知函数有关的以 **rkf45_x** 为形式参数的过程，用户可以像调用其他函数一样调用该过程。通过给定过程的实际参数，即所求的未知函数的自变量值，用户可以得到在相应点处函数的数值，结果以列表的形式同时给出自变量和因变量的值。以上面的求解结果 **sol** 为例，看下面的调用结果：

```
>sol(1);
```

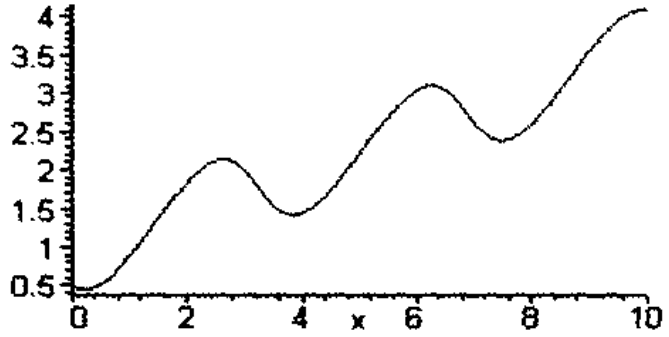
$$[x = 1, y(x) = .8758947797345141]$$

```
>sol(1)[2];
```

$$y(x) = .8758947797345141$$

更为一般地，可以用下面的命令绘制出未知函数的图像：

```
> plot('rhs(sol(x)[2])',x=0..10);
```



在用命令 **dsolve** 计算常微分方程的数值解时，其中还可以包含其他选项，分别用来规定求解的数值方法、迭代求解的初值、求解结果的表示方法等。

12.2 偏微分方程

偏微分方程是以二元或者多元函数为未知函数的微分方程，偏微分方程的求解要比常微分方程的求解复杂得多。Maple V 中用于偏微分方程求解的基本命令为 **pdsolve**，同时在 **PDEtools** 函数库中提供了许多其他命令，用于偏微分方程的分析求解。用户可以使用 Maple V 提供的这些求解工具得到大多数常见偏微分方程的分析解。

本节简单介绍偏微分方程的基本求解工具，主要是命令 **pdsolve** 的使用。关于 **PDEtools** 函数库中的其他求解工具，读者可以参看相关的帮助信息。

12.2.1 求解偏微分方程的基本工具

命令 **pdsolve** 是 Maple V 提供的用于求解偏微分方程的基本工具。命令 **pdsolve** 试图寻找满足偏微分方程的分析解，对偏微分方程的类型、阶数及自变量的个数都没有特别的限制。目前，命令 **pdsolve** 只能用于单个偏微分方程中单个未知函数的求解，在以后的 Maple V 版本中，命令 **pdsolve** 有望实现对偏微分方程组的分析求解。

命令 **pdsolve** 的基本格式为 **pdsolve(PDE, f)**，其中 **PDE** 代表所求解的单个偏微分方程，**f** 代表所求的未知函数，可以用函数的名字或者是 **f(x)**的形式。如果所给的偏微分方程 **PDE** 中只包含一个未知函数，命令的第二个参数 **f** 可以缺省。对于大多数一阶偏微分方程及波动方程、泊松方程等常见的二阶或高阶偏微分方程来说，命令 **pdsolve** 都可以使用分离变量法等标准方法获得分析解；在命令的输出结果中经常引入类似_**Fn**（**n=1**，**2**，…）的函数，例如 **f(x, y, z)= _F1(x) + _F2(y) + _F3(z)**，这些函数代表关于其所带参数的任意函数。下面给出了两个简单的例子，其中 **PDE1** 是一阶偏微分方程，**PDE2** 是一维波动方程：

```
> PDE1:=x*diff(f(x,y),y)-y*diff(f(x,y),x)=0;
```

$$PDE1 := x\left(\frac{\partial}{\partial y}\mathrm{f}(x,y)\right) - y\left(\frac{\partial}{\partial x}\mathrm{f}(x,y)\right) = 0$$

```
> sol1:=pdsolve(PDE1);
```

$$sol1 := \mathrm{f}(x,y) = \_\mathrm{F1}(x^2 + y^2)$$

```
> PDE2:=diff(u(x,t),x$2)=diff(u(x,t),t$2);
```

$$PDE2 := \frac{\partial^2}{\partial x^2}\mathrm{u}(x,t) = \frac{\partial^2}{\partial t^2}\mathrm{u}(x,t)$$

```
> sol2:=pdsolve(PDE2);
```

$$sol2 := \mathrm{u}(x,t) = \_\mathrm{F2}(x+t) + \_\mathrm{F1}\left(\frac{1}{2}x - \frac{1}{2}t\right)$$

和命令 **odetest** 类似，用户可以用命令 **pdetest** 检验所得的解是否满足原来的偏微分方程。命令 **pdetest** 的格式为 **pdetest(sol, PDE)**，其中 **sol** 为所检验的解，**PDE** 代表相应的偏微分方程。下面给出上面两个解的验证情况：

```
> pdetest(sol1,PDE1);
                                0
> pdetest(sol2,PDE2);
                                0
```

在更多的情况下，命令 **pdsolve** 并不能用标准方法求解所给的偏微分方程，这时命令 **pdsolve** 会根据偏微分方程的结构采用一种启发式的运算法则对偏微分方程分离变量，然后对分离变量后的方程分别进行求解。如果不能得到偏微分方程的通解形式，命令 **pdsolve** 将把求解的结果表达为函数 **PDESolStruc** 的形式。即使这样，由于偏微分方程的复杂性，命令 **pdsolve** 所能求解的偏微分方程还是很有限的。

另外，用户可以在命令 **pdsolve** 中加入一些控制选项，用来规定求解的方法、输出结果的形式等。

第 13 章 绘 图

数据可视化也是 Maple V 的一大主要功能。虽然 Maple V 的最大特点是其无与伦比的符号计算能力，Maple V 同样具有很强的数据可视化功能。Maple V 提供的绘图工具，不仅能够绘制常见函数的二维或者三维图形，而且能够绘制各种复杂的空间曲线、空间曲面等复杂的三维图形；同时，在 Maple V 中还能够绘制等高线、等高面、向量图等特殊图形，甚至可以制作简单的动画。在 Maple V 中不仅能够绘制显式定义的函数的图像，而且能够绘制隐函数形式的函数图像。

由于篇幅所限，本章只对 Maple V 的基本绘图功能进行简单介绍，包括基本的二维绘图和三维绘图，以及 Maple V 的图形输出对象的简单操作等。

13.1 二维绘图

二维绘图的基本命令为 **plot**，命令 **plot** 是个功能强大的绘图工具。命令 **plot** 不仅可以用来绘制直角坐标系或者极坐标系中的单个函数的图像，而且能同时绘制多个函数的图像；不仅能够用来绘制普通的函数图像，而且能够绘制参数曲线。

本节首先介绍二维绘图的基础知识和绘图命令的常用选项，接着介绍在同一坐标系中绘制多个函数图像的方法，最后简单介绍极坐标系中的二维绘图和参数曲线的绘制。

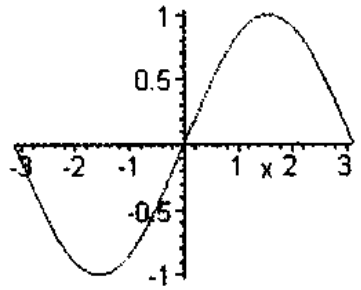
13.1.1 二维绘图的基础知识

本小节的内容包括基本的命令格式、图形输出对象的输出位置、绘图区与图像大小的改变以及图形输出对象的工作环境。

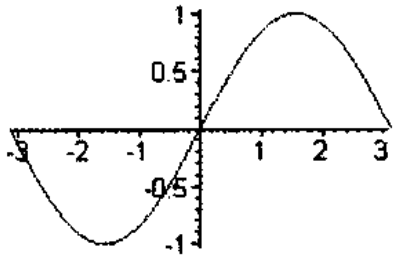
1. 基本的命令格式

命令 **plot** 的基本格式为 **plot(f(x), x=a..b)**，其中 **f(x)**代表任意的单变量的算术表达式，在 **f(x)**中不能含有其他未知参数，**x=a..b** 规定了函数图像的横坐标的范围。如果 **f** 代表一元函数名，其函数图像可以用 **plot(f, a..b)**的简化命令格式。下面的例子绘出了正弦函数的图像，两种命令格式是等价的：

```
> plot(sin(x),x=-Pi..Pi);
```




```
> plot(sin,-Pi..Pi);
```



2. 图形输出对象的输出位置

在默认情况下，函数图像的输出位于当前工作单中命令行的下边，并居中显示。用户可以改变图像的输出设置，从“options”菜单的“Plot Display”子菜单中选择“Window”选项，如图 13-1 所示，可以使绘制的图像输出到图像窗口中。改变设置后，再次执行下面的命令，图像将输出到一个未命名的空白工作单中，如图 13-2 所示，图像孤立地存在于空白工作单的中间；该空白工作单称为 Maple V 的图像窗口，图像窗口具有普通工作单的所有特性。改变这个工作单窗口的大小，工作单中的图像大小也随着改变。

图 13-1　改变图像的输出位置

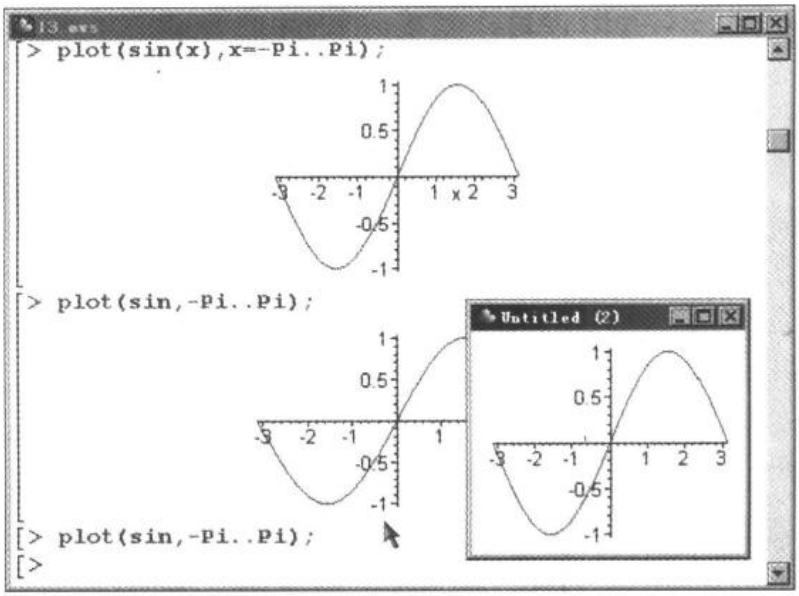


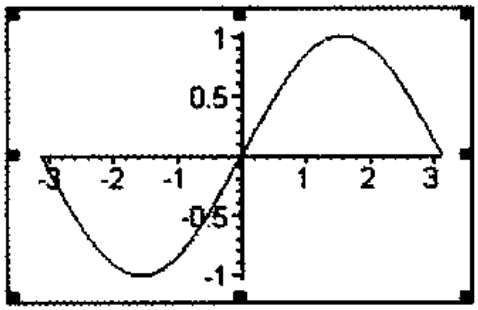
图 13-2　图像的输出位置

本章采用 Maple V 的默认设置，图像的输出都位于当前工作单中的输出行。

3. 绘图区与图像大小的改变

Maple V 图像属于 Maple V 中的图形类输出对象，单击图形类输出对象，图像处于选中状态，在图像的周围出现一个矩形框，矩形框的周边有 8 个控制点，矩形框界定的区域称为 Maple V 的绘图区。在下面的例子中，绘制的三角函数图像处于选中状态：

```
> plot(sin,-Pi..Pi);
```



通过矩形框上的 8 个控制点，可以改变绘图区的大小，从而改变图像的显示大小。当鼠标位于控制点上时，鼠标变成双向箭头的形状，此时按下鼠标并向某一方向滑动，绘图区的大小随着改变。

4. 图形对象的工作环境

当图像处于选中状态时，Maple V 对菜单栏和上下文关联栏进行了调整，菜单栏增添了“Styles”、“Axes”、“Projection”三个菜单项，上下文关联栏也相应调整为对二维图像操作的快捷按钮，如图 13-3 所示。同时，如果在绘图区单击鼠标右键，弹出的上下文菜单与当

前的图像对象适应，其中也包含了“Styles”、“Axes”、“Projection”菜单项，如图 13-4 所示。

图 13-3　二维图像对应的上下文关联栏

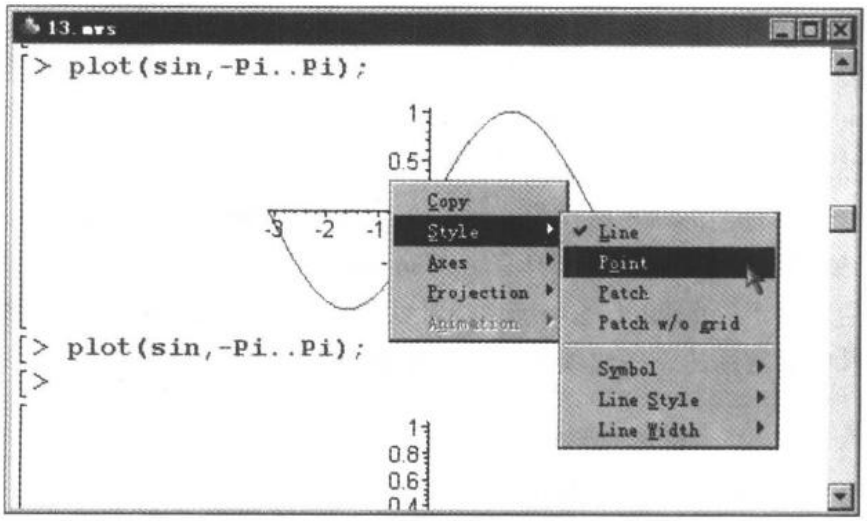
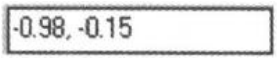


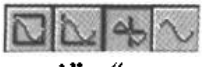
图 13-4　图像的弹出菜单

“Styles”菜单用来规定图像的样式、线形、线宽和孤立点的符号；默认情况下，图像的样式为“Line”（曲线），线形为“Solid”（实线）。“Axes”菜单用来规定坐标轴的形式，有“Boxed”、“Framed”、“normal”、“none”四个选项，默认情况下为“normal”，对应于传统的坐标轴形式。“Projection”菜单用来规定两个坐标轴的显示比例，“Constrained”对应着两个坐标轴按 1:1 显示，默认设置是“Unconstrained”选项，Maple V 自动选择两个坐标轴的显示比例使图像合理显示。上下文关联栏中的快捷按钮是与这三个菜单选项对应的，Maple V 把上下文关联栏分成了四段，下面给出这四段各自的作用：



：其中的数字显示了当前光标的位置坐标，随着光标位置的改变，其中的数字随着发生改变。其中坐标系是图像本身的坐标系，有效范围是在绘图区内。


：用于选择图像的样式，与“Style”菜单项中的四个选项“Line”、“Point”、“Patch”、“Patch w/o grid”相对应。


：用于选择坐标轴的形式，与“Axes”菜单中的四个选项“Boxed”、“Framed”、“normal”、“none”对应。


：在“Constrained”与“Unconstrained”两个选项之间切换。

用户可以从菜单中选择不同的选项，或者单击上下文关联栏中的快捷按钮，改变当前图形对象的特征。按照这种途径对图形对象特征的设定，通常只对当前的图形对象有效；如果绘图命令再次执行，或者绘制其他的图像时，生成的图像的特征仍然采用默认的设定。

更为灵活地，Maple V 允许用户在命令中加入不同的选项，直接规定图像的特征。关于命令 **plot** 的选项，将在 13.1.2 节中介绍。

13.1.2 绘图命令的常用选项

在 13.1.1 节中介绍了对图形对象进行控制的菜单项和工具栏，使用菜单项或工具栏规定图形对象的特征，可以使绘图命令更为简洁。但一方面，这些菜单项或工具栏所规定的选项是很有限的；另一方面，利用菜单项或工具栏对图形对象所作的设定，在绘图命令再次执行时不能保持下来，常需要重新设定。因此，菜单项或工具栏只能作为绘制图形对象的辅助工具。在命令中加入不同的选项，直接规定图形对象的特征，常常是很有必要的。

在 **plot** 中加入可选项的命令格式为 **plot(f(x), x=a..b, option1, option2, ...)**，其中 **option1**、**option2**、…代表所加入的可选项，都具有 **option=value** 的形式，**option** 代表选项名，**value** 代表相应的取值。关于命令 **plot** 的可选项，可以用命令**?plot[options]**获得详细的帮助信息。下面分类介绍其中最常用的一些选项的作用，并给出了一个具体的例子。

1. 坐标轴

与坐标轴有关的常用选项有下面几个：

- **axes**：规定坐标轴的类型，可以有 **FRAME**、**BOXED**、**NORMAL**、**NONE** 四种取值，分别对应于框架坐标轴（  ）、矩形坐标轴（）、普通坐标轴（）和无坐标轴（）；默认的取值为 **NORMAL**。
- **view=[xmin..xmax, ymin..ymax]**：用来指定图像在屏幕上显示部分的横坐标和纵坐标范围。默认情况下，显示整个曲线。
- **coords=name**：规定坐标系的类型。在默认情况下，所有的二维图形的坐标系为笛卡尔直角坐标系；用户也可以改变坐标系的类型，其他可能的取值包括 **logarithmic**（对数坐标）、**polar**（极坐标）等 14 种。
- **label=[x, y]**：规定坐标轴的名字，**x** 和 **y** 必须是字符串。默认情况下，采用与所绘制图像的原函数相对应的自变量和因变量的名字。
- **scaling**：规定两个坐标轴的比例，有 **CONSTRAINED**（1:1）、**UNCONSTRAINED**（Maple V 自动选择）两种取值，默认的取值为 **UNCONSTRAINED**。

2. 图线样式

规定图线样式的常用选项有：

- **style**：规定图线上取样点之间的连接样式，可能的取值为 **LINE**、**POINT**、**PATCH**、**PATCHNOGRID** 中的一种。默认的取值为 LINE，即曲线连接。
- **linestyle=n**：规定连线的线形，**n** 的取值为 **1**、**2**、**3**、**4** 四者之一，分别对应于实线、虚线、短划线、点划线；默认的取值为 **n=1**，实线。
- **symbol=s**：规定图线中取样点的符号，**s** 可能的取值为 **BOX**、**CROSS**、**CIRCLE**、

POINT、**DIAMOND** 中的一种，分别对应于小方块、十字、小圆圈、点、菱形，默认的符号为 **DIAMOND**。

- **thickness=n**：规定连线的宽度，**n** 的取值可能为 **0**、**1**、**2**、**3**，默认值为 **0**。

3. 颜色

color=c：用来规定图线的颜色。**c** 的取值可以采用常见的各种颜色，也可以采用 **RGB** 函数定义等。默认情况下，第一条函数图线的颜色为红色。

4. 标题

用户可以在命令 **plot** 中指定图像的标题，对应的选项为：**title=t**，其中 **t** 必须是字符串。

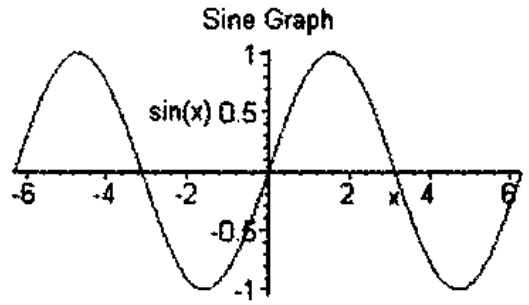
5. 其他

在命令 **plot** 中还可以规定其他的选项，例如：**font** 选项规定其中的文本对象的样式；**numpoints** 选项用来规定取样点的最少个数等，这里不再介绍。

6. 选项使用举例

上面所介绍的选项，在命令中的顺序没有特别的限制，但应该出现在自变量的范围表达式之后。下面给出了函数 **sin(x)**在区间[$-\pi/-2\pi$,2π]上的图像，在命令中用到了 **title** 选项和 **labels** 选项：

```
>plot(sin(x),x=-2*Pi..2*Pi,title="Sine
 Graph",labels=["x","sin(x)"]);
```



13.1.3　在同一坐标系中绘制多个函数图像

命令 **plot** 也可以用来在同一坐标系中绘制多个函数的图像。以 **f(x)**、**g(x)**、**h(x)**三个函数为例，在同一坐标系中绘制三个函数图像可以采用下面的两种命令格式：

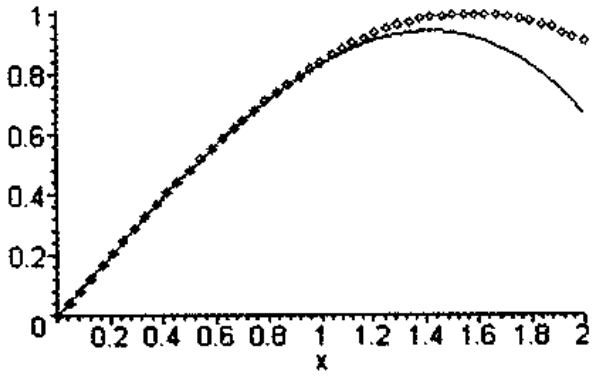
- **plot({f(x), g(x), h(x)}, x = a..b, options)**
- **plot([f(x), g(x), h(x)], x = a..b, options)**

其中给定的自变量的范围对三个函数来说是共同的，**options** 为可选项，13.1.2 节中介绍的选项在这里都可以使用。这些选项可以分别规定每一个函数图像的特征，对应于同一选项的取值包含在同一列表中；或者用一个取值规定三个函数的共同特征。需要注意的是，由于集

合中的元素是无序的，因此在可选项的取值列表中规定的特征对应的函数可能与输入时希望对应的函数不同。

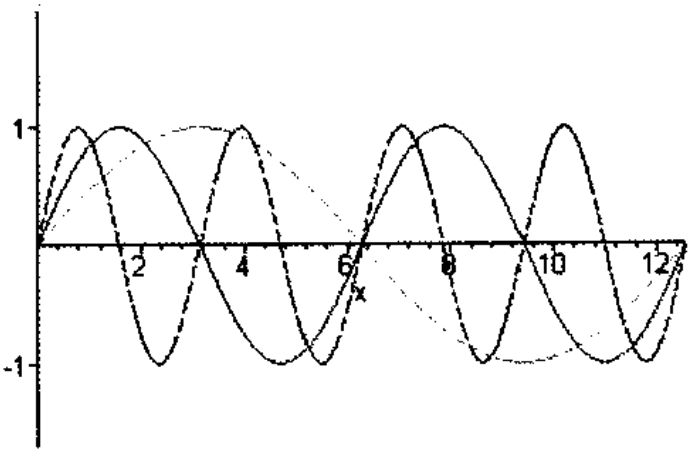
下面的例子中，在同一坐标系中绘出了函数 **sin(x)**和 **x-x^3/6** 的图像，自变量的限定范围为**[0, 2]**；两个函数图像的颜色都为蓝色；函数 **sin(x)**的图像是孤立的用菱形符号表示的点的阵列，函数 **x-x^3/6** 的图像是线形。看下面的结果：

```
> plot([sin(x),x-x^3/6],x=0..2,color=blue,
  style=[point,line]);
```



下面的例子在同一坐标系中绘出了函数 **sinx**，**sin2x** 和 **sin(x/2)**的图形。其中的选项 **color=[red,blue,green]** 表示三个函数图形的颜色分别为红色、蓝色和绿色；选项 **linestyle=[1,3,4]**表示三个函数图形的线形分别为实线、点线和点划线；选项 **ytickmarks=[-1,1]** 表示 **y** 坐标轴上标出的数字为**-1** 和 **1**；选项 **view=[0..4*Pi,-2*Pi..2*Pi]**表示图形显示区域的范围为：$0 \le x \le 4\pi$，$-2 \le y \le 2\pi$。看下面的结果：

```
> plot([sin(x),sin(2*x),sin(x/2)],x=0..4*Pi,
  color=[red,blue,green],linestyle=[1,3,4],
  ytickmarks=[-1,1],view=[0..4*Pi,-2..2]);
```



实际上，用户可以对图像对象命名。**plots** 函数库中的命令 **display** 可以在同一坐标系中同时打印多个二维图形。这也是在同一坐标系中绘制多个函数图像的一种方法。命令的基本

格式为 **display(L)**，其中 **L** 代表由多个图像名的标识符构成的列表或集合。命令 display 在使用之前需要加载到工作空间中。

下面的例子给出了使用 **display** 命令绘制圆的过程。函数 **y1** 和 **y2** 分别代表圆的上下两个半圆，**p1** 和 **p2** 是对应的曲线，命令 **disply({p1,p2})**用来把 **p1**、**p2** 显示在同一个坐标系中，**p1** 与 **p2** 合在一起构成了完整的圆。

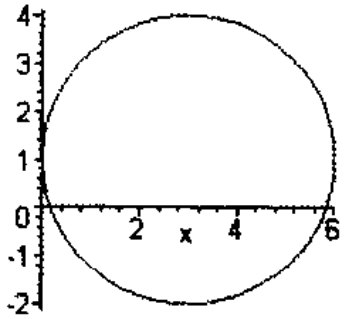
```
> y1:=`y1`:y1:=x->1+sqrt(9-(x-3)^2);
```

$$y1 := x \to 1 + \sqrt{9-(x-3)^2}$$

```
> y2:=`y2`:y2:=x->1-sqrt(9-(x-3)^2);
```

$$y2 := x \to 1 - \sqrt{9-(x-3)^2}$$

```
> with(plots):
> p1:=plot(y1(x),x=0..6):
> p2:=plot(y2(x),x=0..6):
> display({p1,p2});
```



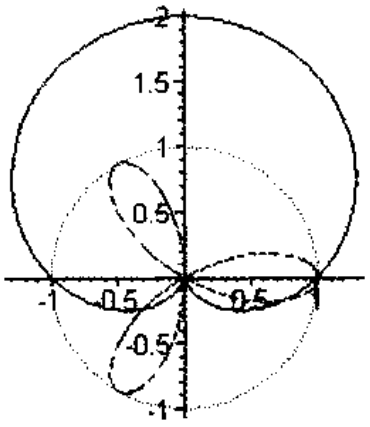
13.1.4　极坐标系中的图形

命令 **plot** 也可以在极坐标系中绘制给定函数的图像。命令的基本格式为 **plot(f(theta), theta=a..b, coords=polar, options)**，其中 **f(theta)**代表以 **theta** 为自变量的函数，**options** 代表除 **coords** 选项外的其他选项。同时，命令 **plot** 也可以在同一极坐标系中绘制多个函数的图像，命令的格式和在直角坐标系中绘制多个函数图像的格式类似。

下面的例子在同一极坐标系中绘出了三个函数 r=1+2sinθ、r=1、r=cos3θ的图像：

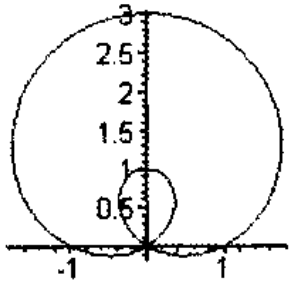
另外，Maple V 在 **plots** 函数库中提供了命令 **polarplot**，用来绘制极坐标系(r,θ)中的函数图像。命令的基本格式为 **polarplot(f(theta), theta=alpha..belta, options)**，其中的选项 **options** 与命令 **plot** 中的选项基本是类似的。命令 **polarplot** 在使用之前需要加载到当前的工作空间中。

```
>plot([1+sin(theta),1,cos(3*theta)],
 theta=0..2*Pi,coords=polar,color=[black,red,blue],
 linestyle=[1,2,4],resolution=600);
```



下面的例子中，使用命令 **polarplot** 绘制极坐标系中的函数 $r=1+2\sin\theta$ 的图像，看下面的结果：

```
>with(plots):
>polarplot(1+2*sin(theta),theta=0..2*Pi);
```



命令 **polarplot** 也可用来在同一极坐标系中绘制多个函数的图像，其命令格式和用命令 **plot** 绘制多个函数图像的格式类似，这里不再介绍。

13.1.5 参数曲线

参数曲线是由参数方程确定的图像。命令 **plot** 可以绘制参数曲线。对于参数方程

$$\begin{cases} x = x(t) \\ y = y(t) \end{cases}, a \leqslant t \leqslant b$$

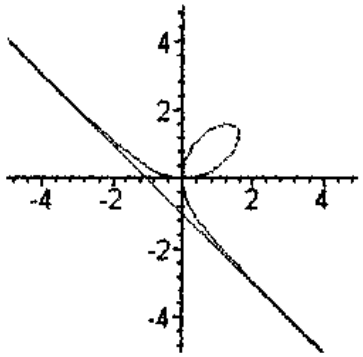
命令 **plot([x(t),y(t),t=a..b], h, v, options)**用来绘制由（**x(t), y(t)**）确定的参数曲线。其中，**h** 和 **v** 具有范围表达式的形式，分别代表绘图时横坐标和纵坐标的显示范围；**options** 代表可选项，可以规定图形对象的特征。

笛卡尔薄层是由参数方程

$$\begin{cases} x=3at/(1+t^3) \\ y=3at^2/(1+t^3) \end{cases}$$

定义的参数曲线。下面的例子绘制出了 a=1 时的笛卡尔薄层。

```
>x:=`x`:x:=t->3*t/(1+t^3):
 y:=`y`:y:=t->3*t^2/(1+t^3):
 plot([x(t),y(t),t=-10..10],-5..5,-5..5,
 numpoints=100,resolution=600);
```

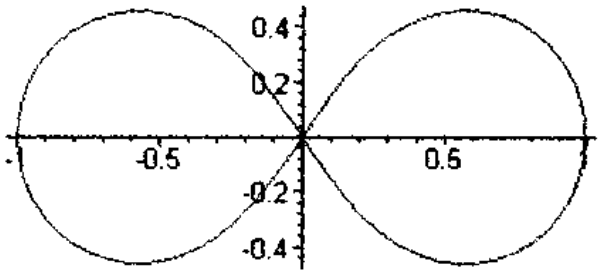


同样，可以在极坐标系中绘制参数曲线。命令 **plot([r(t), theta(t), t=a..b], cords=polar)** 可以用来在极坐标系中绘制出由参数方程

$$\begin{cases} r=\cos t \\ \theta=\sin t \end{cases}, 0\leqslant\theta\leqslant 2\pi$$

确定的参数曲线。看下面的例子：

```
>plot([cos(t),sin(t),t=0..2*Pi],coords=polar);
```



13.2　三维绘图

三维绘图的命令为 **plot3d**。命令 **plot3d** 不仅可以绘制普通的二元函数的图像，而且能够绘制由参数方程确定的参数曲面等。本节首先介绍简单的三维绘图，接着介绍参数曲面的

绘制。

13.2.1 简单的三维绘图

本小节介绍三维绘图的基本命令格式，并简单介绍三维图形对象的基本操作。

1. 三维绘图的基本命令格式

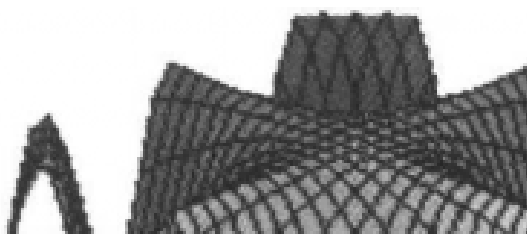
对于普通的二元函数 **f(x,y)**来说，命令 **plot3d(f(x,y), x=a..b, y=c..d)**能够用来绘制出函数 **f(x, y)**当 $a \leqslant x \leqslant b$ 且 $a \leqslant y \leqslant d$ 时的曲面。其中，函数 **f(x, y)**不能含有除自变量 **x** 和 **y** 外的其他未知参数；**a** 和 **b** 必须是确定的实数；**c** 和 **d** 可以是确定的实数，也可以是关于自变量 **x** 的算术表达式，其中也不能含有除 x 外的其他未知参数。如果 **f** 为二元函数的函数名，可以使用等价的命令 **plot(f, a..b, c..d)**绘制函数 **f(x, y)**的图像。在命令 **plot3d** 中，也可以加入不同的选项，规定输出图像的特征。

下面给出一个二元函数的例子，两种命令格式输出的结果是一致的：

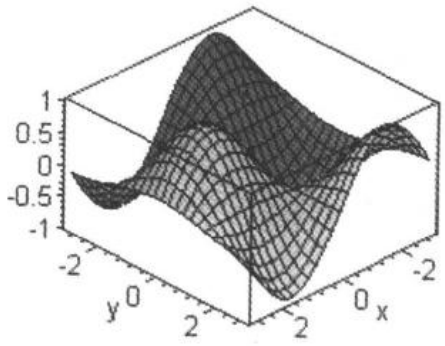
```
> f:=(x,y)->cos(x*y):
  plot3d(f(x,y),x=-3..3,y=-3..3);
```

```
> plot3d(f,-3..3,-3..3);
```

在默认情况下，三维图像的输出不显示坐标轴。下面的命令中包含了 **axes=boxed** 选项，因此输出的图像包含在矩形体内，矩形体表明图像所处的三维坐标系：



```
> plot3d(sin(x)*cos(y),x=-3..3,y=-3..3,
  axes=boxed);
```



2. 三维图形对象的简单操作

和二维图形输出对象一样，在三维图形对象上单击鼠标左键，三维图形对象周围出现了带控制点的黑色矩形框，三维图形对象处于选定状态。用户可以拖动矩形框上的控制点改变绘图区的大小，从而改变当前图形对象的显示尺寸。

三维图形对象与二维图形对象的一个重要不同点在于，三维图形对象是立体的，观察的角度不同，所看到的图像显示也不尽相同。选中三维图形对象后，可以把光标移到绘图区，按住鼠标左键并滑动鼠标可以改变三维图形对象的观察角度。

当选定三维的图形输出对象时，系统集成界面的菜单栏和上下文关联栏以及当前对象的弹出菜单都调整为与三维图形对象相适应的配置。在菜单栏中出现了“Style”、“Color”、“Axes”、“Projection”，其中包含了用来规定三维图形对象特征的菜单项。图 13-5 给出了选定三维图形对象时的上下文关联栏。

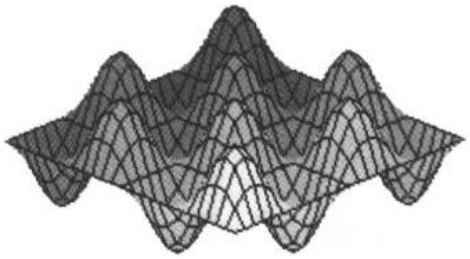
图 13-5　三维图形对象的上下文关联栏

在上下文关联栏中，角度 θ 和 Φ 的值显示了当前三维图形对象的观察角度。当观察角度改变时，两个角度的值会随着发生变化。更为方便的是，用户可以直接在两个角度对应的域中输入特定的角度，或者利用域右边的加减箭头选定所需的角度，当这些角度改变后，三维图形对象会自动旋转到对应的角度。

可以利用菜单栏或者上下文关联栏中的快捷按钮来规定三维图形对象的显示特征，这些菜单选项或快捷按钮通常是和命令 **plot3d** 中的可选项相对应的。

下面以对一个三维图形对象的操作为例，介绍对三维图形对象的简单操作。首先用命令 plot3d 绘制出下面的三维图像：

```
> plot3d(sin(x)*sin(y),x=0..4*Pi,y=-2*Pi..2*Pi);
```



以下是对该三维图形对象进行具体操作的步骤，读者可以在执行每一步操作的时候，观察图形对象显示的变化情况。

（1）从“Projection”菜单中确认选择了“No Perspective”选项和“Unconstrained”选项，这两项是 Maple 的默认操作；

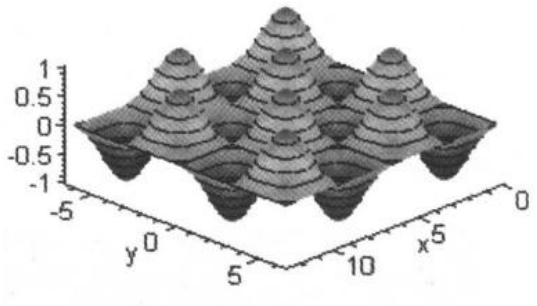
（2）从“Axes”菜单中选择“Framed”选项，或者单击工具栏中的快捷按钮 。

（3）从“Color”菜单中选择“Z (Hue)”选项。

（4）从“Style”菜单中选择“Patch and contour”选项，或者单击工具栏中的快捷按钮 。

执行上面的四步操作后，图形对象的显示如下所示：

```
> plot3d(sin(x)*sin(y),x=0..4*Pi,y=-2*Pi..2*Pi);
```



13.2.2 参数曲面

命令 **plot3d** 也可以用来绘制由参数方程确定的参数曲面。命令的格式为 **plot3d([f(s, t), g(s, t), h(s, t)], s=a..b, t=c..d)**，其中 **f(s, t)**、**g(s, t)**、**h(s, t)**代表以 **s**、**t** 为自变量的二元函数，**s**、**t** 为参数；命令的输出是由对应的参数方程确定的参数曲面。如果 **f**、**g**、**h** 代表二元函数的函数名，则命令 **plot([f, g, h], a..b, c..d)**与前面的命令格式等价。在这两种命令格式中，也可以加入合法的可选项，控制图形对象的显示特征。

$$\begin{cases} x=f(s,t) \\ y=g(s,t),\ a\leqslant s\leqslant b, c\leqslant t\leqslant d \\ z=h(s,t) \end{cases}$$

二次曲面是和二维圆锥曲线相对应的三维对象。在绘制二次曲面的过程中，由于三个

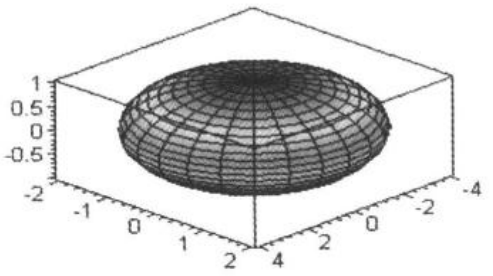
坐标变量之间的关系是隐式的，往往需要转化为参数方程的形式来表示，然后绘制出对应的参数曲面，即得到所求的二次曲面。这里举一个椭圆球面和双曲面的例子，用命令 **plot3d** 绘制出其对应的图像。

椭圆球面的方程为：$\frac{1}{16}x^2+\frac{1}{4}y^2+z^2=1$，对应的参数方程为：

$$\begin{cases} x=4\cos t\cos r \\ y=2\cos t\sin r,(-\pi/2\leqslant t\leqslant \pi/2),(-\pi\leqslant r\leqslant \pi) \\ z=\sin t \end{cases}$$

下面是绘制椭圆球面的命令和输出的图形：

```
> plot3d([4*cos(t)*cos(r),2*cos(t)*sin(r),sin(t)],
  t=-Pi/2..Pi/2,r=-Pi..Pi,axes=boxed);
```



双曲面的方程为：$\frac{1}{16}x^2+\frac{1}{4}y^2-z^2=1$。

$$\begin{cases} x=4\cos t\cos r \\ y=2\cos t\sin r,(-\pi/2\leqslant t\leqslant \pi/2),(-\pi\leqslant r\leqslant \pi)。 \\ z=\tan t \end{cases}$$

由于函数 **sec(t)**和 **tan(t)**在 t=± π/2 时没有定义，同时在 **t** 接近± π/2 时，对应的三个坐标值都很大，因此在下面的绘图命令中，把参数 **t** 的范围取为[- π/3, π/3]。看下面的结果：

```
> plot3d([4*sec(t)*cos(r),2*sec(t)*sin(r),tan(t)],
  t=-Pi/3..Pi/3,r=-Pi..Pi,axes=boxed);
```