目　录

第 1 章 Maple 6 基本知识

提起计算机，人们最先想到的就是它强大的数值计算功能，仅仅是市场中出售的个人计算机（PC），就已经具有每秒钟计算 1000000000（10 亿）次加法的能力。虽然这只是其中央处理器的时钟频率，同它的实际四则运算本领还有差异，但也足以让人脑望尘莫及。可以说计算机的发展已经圆满地完成了其设计的最初使命：将人类从繁重的数值计算中解脱出来。然而仅仅在数学领域，就还有如公式的推导、化简，多项式求和，函数展开等等枯躁复杂的工作需要得到计算机的帮助。这些工作都被统称为符号代数或符号计算。符号代数所涉及的内容十分庞大，从初等代数的一阶方程、方程组，到高等数学中的微分方程、线性代数，几乎涉及到了整个数学领域。实际上符号代数的定义即根据代数规律计算得到的准确结果,而不是依靠浮点数近似来表示的计算结果。随着符号处理系统的发展，这些以前被视为需要人工智能才能完成的工作，已经可以被计算机出色地解决，这又使得人类摆脱了一项繁琐的任务，为进一步发挥人类的创造性思维提供了基础。

1.1 计算机符号代数系统

随着计算机科学的飞速发展，以及工程计算的迫切需求，计算机符号代数系统（Computer Algebra System, CAS）得到了广泛的应用及拓展。一般按功能的不同，将此系统划分为两大类：专用系统和通用系统。

专用符号代数系统是特别为解决物理、化学或数学中某一方面问题而编写的。例如天体物理方面的 CAMAL；针对核物理的 SCHOOHSCHIP；应用于物理化学，计算分子结构、反应动力学的 GAUSSIAN；以及分析微分方程的 DELIA 等等。专用系统由于一般使用针对不同问题的特有符号系统和数据结构，因此各个软件都具有很高的效率和独有的特色。虽然本书主要介绍通用系统中的 Maple 软件，但希望读者根据需要选择适合自己实际工作的软件，有时候适当的专用软件能起到事半功倍的效果。

通用符号系统则是针对普遍的应用领域而开发出来的，包含大量的数据结构与数学函数包。通用系统软件的执行方式一般有两种，一种是输入一条命令、输出一条结果式的命令行方式。虽然不同软件的执行命令不尽相同，但一般都具有数值计算、符号计算、图形处理这 3 种功能；另一种是类似 FORTRAN、C 语言的编程模式。不同的软件编程语言也不尽相同，但能够实现的功能类似。

概括起来，计算机符号代数系统可以应用到以下的领域：

（1）公式推导的工具。符号系统可以出色地完成科研、工程中所遇到的复杂公式推导

与验证等工作，不仅可以避免人工推导过程中可能会出现的错误，而且显著地提高了工作效率。

（2）数学实验。由于推导工作的简化，可以通过对符号的变换与处理，为数学研究提供一种类似“实验室”的环境，通过不同的方法寻找可能出现的结果。

（3）辅助教学。符号代数系统的易学易用为学生和老师提供了很好的学习帮助系统。许多问题借助计算机将得到更快更好的解决。同时也为制作课件和练习系统提供了方便。

1.2 Maple 6 概述

1.2.1 Maple 的发展历史

- 1980 年，加拿大教授 Keith Geddes 与 Gaston Gonnot 在 Waterloo 大学开始设计开发一种数学软件，并被赞助者很快推广到欧美各大学使用。最早使用 B 语言编写，但很快便改用 C 语言。
- 1984 年，Watcom Inc. 代理销售 Maple 3.3。
- 1987 年，Maple 4.0、4.1、4.2 相继发布。
- 1988 年，Waterloo 大学成立自己的公司并直接销售 Maple。
- 1989 年，发布 Maple 4.3。
- 1990 年，包含新图形用户界面(GUI)，添加了三维图形绘制功能的 Maple 5 发布。
- 1994 年，Maple 5 Release 3 发布，并开始兼容 Microsoft®的 Word 文件。
- 1996 年，Release 4 发布。
- 1997 年，Release 5 发布。此版本进一步完善了图形界面，并提供了与 MATLAB 的信息交流界面。
- 1998 年，相继发布 Release 5.1、Release 5.2。
- 2000 年 1 月，在与英国剑桥的“the Numerical Algorithms Groups(NAG)”签定合作协议，获得 NAG 函数库后，增强了数值计算功能的 Maple 6 被推向市场。
- 2000 年 4 月，适合在校学生使用的版本上市。

1.2.2 Maple 6 的新特点

- 在同 NAG（Numerical Algorithms Group）合作，并利用他们的函数库后，Maple 6 中有关线性代数计算的部分得到了显著提高，特别是针对包括浮点数矩阵、向量间的运算在内的数值计算精度及效率作了明显的优化。
- Maple 符号语言的细微调整及提高。
- 对图形处理用户界面进行了调整，添加了新的图形处理功能。
- 同 Microsoft®的 Excel 2000®进行了整和，可以在 Excel 2000 中调用 Maple 6 指令，

同时可以实现两软件间数据的相互拷贝。

- 添加了新的程序包，并对旧有程序进行了修改。
- 对符号语言的部分命令进行了调整，详情请参考联机帮助。

1.2.3　系统要求和安装配置

对 PC 机 Windows 用户：

上理器。

5/98 用户）或 32MB 内存（Windows NT 用户）。

56 色及 640×480 图像分辨率。

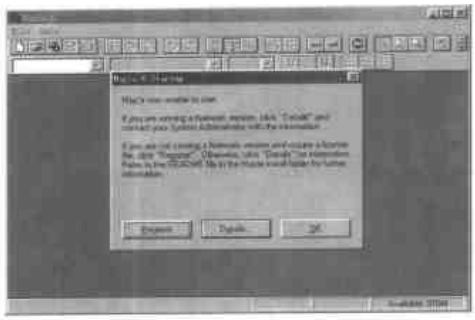
ack 5)、Windows 95/98 或 Windows 2000 操作系统。

。

反，由于两种程序的安装均遵循标准程序，本书将略过此处不提。需要注意的是相对于 Maple V Release 2，Maple 6 将不再提醒安装共享函数库。因为它已将相应的类库与程序相整和，不需要特别的调用过程。Maple 6 安装程序的另一个特点是会询问用户是否以多用户方式安装系统（见图 1-1），并为不同的用户建立各自的配置文件。本书将以单用户模式介绍系统的使用。

图 1-1　Maple 6 多用户安装选择

Maple 6 需要用户在向 Waterloo 公司购买使用协议后才可以使用。否则将出现如图 1-2 所示的警告信息。用户通过信件或在线注册的方式，会获得使用协议文件：*license.dat*。将其拷贝至 Maple 6 安装目录下的 license 子目录后，便可获得程序的使用权。



缺少 license.dat 的启动画面

1.2.4 获得帮助

Maple 自带完善的帮助系统，可以全方位的满足用户在不同情况的需求。本书不可能将 Maple 自带的所有命令全部介绍，所以熟练地使用帮助系统会有助于读者独立完成工作。用户可以通过不同的方法获取帮助信息。如果想对 Maple 进行系统地学习，可以先通过选择主菜单 HELP 选项中的 Introduction 获得对软件的初步了解（见图 1-3）。

图 1-3 Maple 系统帮助菜单

如果想了解 Maple 如何实现某种特定的功能，或是想进一步了解 Maple 程序的某一部分，则可以利用在 Topic Search 选项中输入关键词来获得帮助（见图 1-4）。

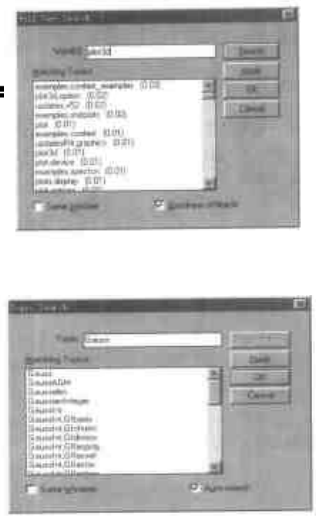


图 1-4　Topic Search 窗口

如  涉及到了某个关键词，则可使用 Full text search（见图 1-5）。

图 1-5　Full Text Search 菜单

如果想了解 Maple 软件中的各个按钮、菜单中选项的具体功能，则可以激活 HELP 菜单中的 Balloon help 选项（见图 1-6），这样用户就会得到鼠标指到位置所对应功能的简单描述。

图 1-6　使用 Balloon Help

如果用户在使用 Maple 的过程中，遗忘了如何设定某个函数的参数，则可以直接输入“？+函数名”来获得对特定函数的帮助。

1.2.5 让你的 Maple 开始工作

首先，让我们完成一项可能需要花费祖冲之几百年时间才能完成的工作：求出 π 的 500 位有效数字。请在 Maple 中输入“evalf(Pi,500);”，注意，需要以“;”作为结束标志，然后回车。不超过一秒钟，你就会看到：

```
> evalf(pi, 500);
3.14159265358979323846264338327950288419716939937510582097494459230781640 \
 62862089986280348253421170679821480865132823066470938446095505822317253 \
 59408128481117450284102701938521105559644622948954930381964428810975665 9 \
 33446128475648233786783165271201909145648566923460348610454326648213393 6 \
 07260249141273724587006606315588174881520920962829254091715364367892590 3 \
 60011330530548820466521384146951941511609433057270365759591953092186117 3 \
 81932611793105118548074462379962749567351885752724891227938183011949 1
```

虽然 500 位有效数字可能对我们没有任何意义，但 Maple 的功能可见一斑。

接下来，让 Maple 帮助不同阶段的学生完成他们的作业。

一道小学四则运算：求(11+(9-8)*7)/6 的值。

```
> (11+(9-8)*7)/6;
```

$$3$$

一道中学的因式分解：展开$(x+2)^4$。

```
> func:=(x+2)^4;
```

$$func := (x+2)^4$$

```
> expand(func);
```

$$x^4+8x^3+24x^2+32x+16$$

一道大学的微积分：求 $\arcsin(5x^2-1)$的导数。

```
> func:=arcsin(5*x^2-1);
```

$$func := \arcsin(5x^2-1)$$

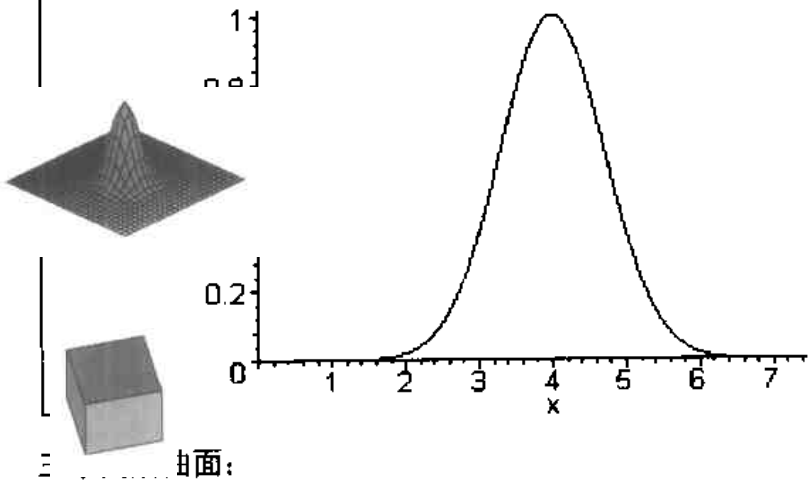
```
> diff(func,x);
```

$$10\frac{x}{\sqrt{-25x^4+10x^2}}$$

最后，让 Maple 制作曲线与曲面。

二维高斯曲线：

```
> plot(exp(-(x-4)^2),x=0..8);
```



三维高斯曲面：

```
> plot3d(exp(-(x-4)^2-(y-4)^2),x=0..8,y=0..8);
```

正六面体：

```
> polyhedraplot([0,0,0],polytype=hexahedron);
```

1.3　Maple 6 的数据类型

Maple 数据类型除了简单的数值类型外，还包括序列(Sequence) 、集合（Set）、列表(List)、

系列(series)、表（table）等数据结构，不同的结构都有各自的应用领域。本节将分别对它们进行介绍。

1.3.1 简单数值类型

Maple 6 支持的简单数值类型包括整数、分数、浮点数和复数。由数字 0~9 以及表示指数的 e，表示复数单位的 I 组合而成。不同于 C、FORTRAN 等程序语言对各种数据类型的长度限制，Maple 不限制整数的大小及浮点数小数点后的位数，只要内存允许，可以在 Maple 中计算出任意大小、任意精度的数值，缺省显示小数点后的 10 位有效数字，并可根据用户的需要随意调整显示位数，如：

```
> evalf(1/3);
                          .3333333333
> evalf(1/3,20);
                    .33333333333333333333
```

如果输入的是分式，Maple 将自动对其进行化简，而且会精确保留其数值。如：

```
> 2+656/32;
```

$$\frac{45}{2}$$

对于浮点数，Maple 将始终按照输入时的精度对它进行处理，而且会将计算结果同算式中小数点后有效位数最多的一个因子保持精度一致，如：

```
> 1.1+2.22+3.333+4.4444;
                          11.0974
```

在使用指数形式的表达式时，注意指数要紧跟在“E”或“e”的后边，如果分开，系统会认为是在执行加减法运算，如：

```
> 5e+3;
                          5000.
>5e  +3;
                            8.
>.2e-2;
                          .002
>.2e  -2;
                          -1.8
```

进行浮点数、分数的混合运算时，Maple 会以 10 位有效数字的浮点数作为默认的显示方式，如：

```
> 3.4+3/4+3^4;
```

85.15000000

对于复数，Maple 会自动将“I”显示在其应该出现的位置，而且可以通过“Re()”，“Im()”两个函数提取出复数的实部与虚部，如：

```
> evalf(-5^(1/3));
```

-1.709975947

```
> evalf((-5)^(1/3));
```

$.8549879733 + 1.480882610\,I$

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

.8549879733,1.480882610

上个例子中，用到了“%”算符，它代表上一行的结果。同时要注意例子中所提示的括号的作用。

除了虚数单位 I 以外，Maple 中还定义了其他一些数学常数，这些数学常数都是精确的，表示的意义与数学中定义的相同。例如圆周率 π 无穷大等。表 1-1 中列出了一些常用的数学常数。

表 1-1　Maple 中的数学常数

数学常数	意　义
I	虚数单位，$I=\sqrt{-1}$
Pi	圆周率
True	逻辑表达式的值，真
False	逻辑表达式的值，假
FAIL	Maple 中表示不确定的值或操作失败等
Catalan	Catalan 常数,值为 0.915965594...
infinity	无穷大
gamma	欧拉常数，值为 0.5772156649...

1.3.2　序列

序列是数学中的一个重要概念。一个序列可以理解成数字按照一定规律排成的有限或无限长的列表。Maple 中以“seq（）”生成一个序列，如：

```
> seq(i,i=1..10);
```

$$1,2,3,4,5,6,7,8,9,10$$

```
> seq(i,i)="Maple");
```

$$\text{"M","A","P","L","E"}$$

```
> seq(i^2,i=i..10);
```

$$1,4,9,16,25,36,49,64,81,100$$

```
> seq(i,i=1+2*x+3*x^2+4*x^3);
```

$$1,2x,3x^2,4x^3$$

```
> seq(degree(i,x),i=1+2*x+3*x^2+4*x^3);
```

$$0,1,2,3$$

Maple 中的序列还涉及到一个有用的算符“$”：

```
> x$4;
```

$$x,x,x,x$$

```
> diff(ln(x),x$4);
```

$$-6\frac{1}{x^4}$$

```
> seq(diff(ln(x),x$n),n=1..5);
```

$$\frac{1}{x},-\frac{1}{x^2},2\frac{1}{x^3},-6\frac{1}{x^4},24\frac{1}{x^5}$$

从这个例子，读者可以发现“$”算符的作用类似于编程中的循环次数。第 1 条命令相当于重复显示 4 次 x，第 2 条命令，相当于求∂ln(x)/∂ ^{4}x，或者说 ln(x)对 x 求 4 次偏导数。第 3 条命令则求出 ln(x)对 x 的 1、2、3、4、5 阶偏导数。由此可见，“$”符号相当于对重复命令的化简，灵活的使用会有效地提高工作效率。

1.3.3 集合

Maple 中的集合对应着数学中的“集合”概念。不同的是 Maple 集合中的元素会按照一定顺序储存，而且系统会自动删除相同的元素。集合以“{}”来定义，集合中的元素以“,”分割。如：

```
> S1:={1,a,2,b,3,c};
```

$$S1:=\{1,2,3,a,b,c\}$$

```
> S2:={4,a,5,B,6,c};
```

$$S2:=\{4,5,6,a,c,B\}$$

```
> S3={c,b,a,3,2,1};
```

$$S3 := \{1,2,3,a,b,c\}$$

在这个例子中，我们用不同的顺序定义了 3 个集合，但读者会发现显示的顺序有一定的规律，即先数字，后字符，并按照 ASCII 字符顺序排列。

接下来，介绍一些对集合操作的命令，包括提取集合元素的“op”，判断集合是否相等的“evalb”，判断某元素是否属于集合的“member”，以及对集合的并（union）、交（intersect）、差（minus）运算。读者可以自行从例子中判断每个命令的使用方法。

```
> op(S1);
```

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

```
> op(2,S1);
```

$$2$$

```
> evalb(S1=S2);
```

$$false$$

```
> evalb(S1=S3);
```

$$true$$

```
> member(a,S1);
```

$$true$$

```
> member(B,S1);
```

$$false$$

```
> S1 union S2;
```

$$\{1,2,3,4,5,6,a,b,c,B\}$$

```
> S1 intersect S2;
```

$$\{a,c\}$$

```
> S1 minus S2;
```

$$\{1,2,3,b\}$$

注意利用“op”命令选取集合元素的时候，对应的位置是经过系统排序后的元素位置，这同用户建立集合时的输入顺序有所不同。

1.3.4　列表

列表同集合类似，也是一组元素的组合。相对于集合，列表更像行列式中的一列元素，它是 Maple 处理向量、矩阵、张量等的基础。列表用“[]”表示，元素以“,”分隔。Maple 将列表理解成数字对象的有序集合，不干涉其生成顺序与包含元素的内容。

请注意以下的例子：

```
> S1:=[12,34,56,A,B,c];
```

$$S1 := [12,34,56,A,B,C]$$

```
> S2:=[67,78,89,d,e,f];
```

$$S2 := [67,78,89,d,e,f]$$

```
> op(S1);
```

$$12,34,56,A,B,C$$

```
> nops(S2);
```

$$6$$

```
> S3:=[op(S1),op(S2)];
```

$$S3 := [12,34,56,A,B,C,67,78,89,d,e,f]$$

```
> S3[3..8];
```

$$[56,A,B,C,67,78]$$

这里使用了几种同集合类似的显示元素的方法，并且显示了如何合并列表。新出现的“nops”命令会显示列表或集合内包含的元素个数。下面，结合序列产生函数“seq（）”，我们让 Maple 生成一些特殊的序列，例如生成以 π 小数点后的 100 位数字为元素的一个列表：

```
> pi_approx:=evalf(pi,100);
```

$$pi\_approx := 3.14159265358979323846264338327950288419716939937\backslash$$
$$51058209749445923078164062862089986280348253421 17068$$

```
> dg:=[seq(trunc(frac(pi_approx*10^n)*10),n=0..99)];
```

$$dg := [1,4,1,5,9,2,6,5,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,$$
$$0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,$$
$$0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,$$
$$0,0,0,0]$$

这里使用了几个新的命令：“frac()”的作用是显示括号内数值的小数部分，“trunc（）”则正好相反，是取得括号内数字的整数部分。注意得到的结果只有 10 位有效数字。这是因为前文曾经提到过，Maple 默认的浮点数显示位数为 10 位，所以系统计算在上个例子的第一步：“frac(Pi_approx*10^0)”时，就只保留了 10 位有效数字。为了达到预期目的，我们将系统默认有效位数“Digits”改成 101 位，就会得到正确结果。在最后，再利用“nops（）”命令验证是否得到了 100 位数字：

```
> Digits:=101;
```

$$Digits := 101$$

```
> dg:=[seq(trunc(frac(Pi_approx*10^n)*10),n=0..99)];
```

$$dg := [1,4,1,5,9,2,6,5,3,5,8,9,7,9,3,2\ 3,8,4,6,2,6,4,3,3,8,3,2,7,$$
$$9,5,0,2,8,8,4,1,9,7,1,6,9,3,9,9,3,7,5,1,0,5,8,2,0,9,7,4,9,4,4,5,$$
$$9,2,3,0,7,8,1,6,4,0,6,2,8,6,2,0,8,9,9,8,6,2,8,0,3,4,8,2,5,3,4,2,$$
$$1,1,7,0,6,8,0]$$

```
> nops(dg);
```

$$100$$

有关列表的使用，会在“线性代数”一章中做进一步介绍。

1.3.5　系列

系列“series（）”所生成的序列就是将括号中的函数进行泰勒展开，同“taylor()”的效果相同。如：

```
> s:=series(exp(x),x);
```

$$s := 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

```
> s:=series(x^5,x=1,3);
```

$$s := 1 + 5(x-1) + 10(x-1)^2 + O((x-1)^3)$$

```
> s:=taylor(x^5,x=1,3);
```

$$s := 1 + 5(x-1) + 10(x-1)^2 + O((x-1)^3)$$

```
> op(s);
```

$$1,0,5,1,10,2,O(1),3$$

1.3.6　数组

类似程序语言中的数组的概念，Maple 也可以定义数组类型的数据格式。代数中的向量、张量、行列式、矩阵在 Maple 中都以类似数组的方式存储。有关数组的具体操作、运算都将在第 3 章“线性代数”中做详细说明，此处，只对数组的定义，元素的引用做简单介绍。

Maple 中，使用“array（）”函数定义数组，利用数组的下标引用数组元素，如：

```
> A:=array(1..5);
```

$$A := \text{array}(1..5,[])$$

```
> print(A);
```

$$[A_1, A_2, A_3, A_4, A_5]$$

```
> B:=array(1..3,1..3,[[1,2,3],[2,3,4],
  []]);
```

$$B := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ B_{3,1} & B_{3,2} & B_{3,3} \end{bmatrix}$$

```
> indices(B);
```

[1,3],[2,3],[2,1],[2,2],[1,1],[1,2]

```
> B[3,2]:=5;
```

$$B_{3,2} := 5$$

```
> print(B);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ B_{3,1} & 5 & B_{3,3} \end{bmatrix}$$

利用默认方式（“[]”）定义的数组，Maple 会自动以“数组名+下标”的方式指定其中的元素。利用“indices（）”函数，可以列表显示数组中有哪些位置的元素已被赋值。利用“array（）”函数定义数组时，可以使用的参数有：“symmetric（对称阵）”、“antisymmetric（反对称阵）”、“diagonal（对角阵）”、“sparse（零阵）”以及“identity（单位阵）”，其详细的使用方法也会在“线性代数”一章中介绍。

1.3.7 表

“表（table）”类型的数据类似于“列表（list）”类型，即元素的集合。但不同的是列表类型中元素必须全部为数值型数据，而表中的元素可以是任何数据类型。Maple 使用“table（）”函数创建一个表，如：

```
> t1:=table([11,22,33]);
```

$t1 := \text{table}([1 = 11, 2 = 22, 3 = 33])$

```
> t1[1];
```

11

```
> t2:=table([first ="number
  one",second ={1,2,3},third =[1,2,3]]);
```

$t2 := \text{table}([second = \{1,2,3\}, third = [1,2,3],$
$first = \text{"number one"}$
$])$

```
> t2[first];
                        "number one"
> t2[second];
                          {1,2,3}
> t2[third];
                          [1,2,3]
> t2[third][1];
                            1
```

从这个例子中，可以发现对没有指定元素名称的表，系统会自动按照数据的输入顺序为其标记序号，用户在以后可以凭此序号引用表中的元素。而且表中的元素还可以嵌套定义，如在上例中定义的第 2 个表中，第 3 个元素为一个列表，它又包含了 3 个元素，利用如最后一条命令的形式，用户可以获得深层次的元素调用。

除去如上例中的对表元素的单独显示外，还可以利用“op（）”命令或“op（op（））”命令显示表中的全部元素。同时，使用函数“indices（）”以及“entries（）”，可以分别显示出表元素的标记或其内容，例如对上例中定义的表“t2”继续操作：

```
> op(t2);
table([second = {1,2,3}, third = [1,2,3],
  first = "number one"
  ])
> op(op(t2));
  [second = {1,2,3}, third = [1,2,3], first = "unmber one"]
>  indices(t2);
              [second],[third],[first]
> entries(t2);
        [{1,2,3}], [[1,2,3]], ["number one"]
```

需要注意的是 Maple 系统对表的储存、显示顺序同用户的输入顺序有差别，虽然不会对 op（）命令有影响，但在使用“indices()”和“entries()”时有可能对用户产生错误引导，希望读者引起注意。

用户只需直接引用已有元素或以表的名字开头输入新元素就可以完成修改、添加、删除表中元素的工作，如下例所示：

```
> t2[first]:="the first number";
              t2first := "the first number"
```

```
> t2[second]:='t2[second]';
```

$$t2_{second} := t2_{second}$$

```
> t2[forth]:=4;
```

$$t2_{forth} := 4$$

```
> op(t2);
```

table([*third* = [1,2,3], *first* = "the first number", *forth* = 4])

第 1 条命令修改了表“t2”中“first”元素的内容，第 2 条命令利用“变量= '变量' ”的命令形式删除了表中的“second”元素，第 3 条命令则直接为表添加了新的元素“forth”，最后，列出了表中的全部内容。可以看出 Maple 对表的操作十分简洁、透明。

最后，对本节所介绍的各种数据类型略作总结。虽然在这一节，我们列出了很多 Maple 支持的各种类型的数据结构，其实在 Maple 的各种程序包中，还包括着很多其他类型的数据结构。读者应该发现，这些数据类型是没有单独存在的意义的，定义它们，只是为今后在处理实际问题时能够方便、直观的引用数据。从后续章节中读者会发现：序列类型是多项式的基础，集合对应着线性空间的基矢量，向量则是用列表的形式定义的，而矩阵则完全是在数组概念的基础上发展起来的，表的概念在编程、文件操作中也有体现。

读者阅读完本节，首要学习目的不是记住使用各种数据结构、定义的方法，而是要学会如何掌握一种新的数据类型的特点，各种函数如何对它进行读写、修改。只有做到能举一反三，才不会因为处理问题时遇到新的数据结构而手忙脚乱，实现对 Maple 的真正融会贯通。

1.4 Maple 6 的语法规则

Maple 语言同常用的进行数值计算的计算机程序语言（如 C、FORTRAN）不同，因为它的特点是符号运算，处理的是没有数值意义的运算方程，所以除去变量、常量、四则运算符等基本的符号系统外，Maple 系统还必须能够判断用户输入的符号表达式的意义，进行符合惯例的顺序处理，以达到用户期望的结果。这一节，我们将简要介绍 Maple 6 的语法规则，希望能为读者掌握后续章节打下基础。

1.4.1 变量与常量

与程序语言类似，为方便计算或者储存中间结果，用户在 Maple 中可以定义自己的变量。变量可以是除系统保留字以外的任意字母开头的字符或字符串，如“a”，“A_1”，“Pi_Approx”等都是合法的变量。用户也可以用“_”作为变量的开始字符，但由于很多系统保留的全局变量一般也是以“_”开头，而系统不会提醒用户的重复定义，因此会造成一些不必要的麻烦，所以一般不提倡以下划线作为变量的初始字符。

Maple 以“:=”定义变量，注意不要将它同“=”相混淆。请看如下的例子：

```
> a=3;
```

$$a=3$$

```
> whattype(a);
```

$$symbol$$

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

$$a:=3$$

```
> whattype(a);
```

$$integer$$

“Whattype()”函数是显示括号内的变量或表达式的系统默认类型。从上例可以看出，仅靠“=”是无法将数值赋值给变量的，只有以“:=”才能定义变量。下面看几个系统默认的变量类型：

```
> f:='f';
```

$$f:=f$$

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

$$symbol$$

```
> f:=1/3;
```

$$f:=\frac{1}{3}$$

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

$$fraction$$

```
> f:="strings";
```

$$f:="strings"$$

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

$$string$$

```
> f:='strings';
```

$$f:=strings$$

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

$$symbol$$

```
> f:=x^2-2*x+1;
```

$$f:=x^2-2x+1$$

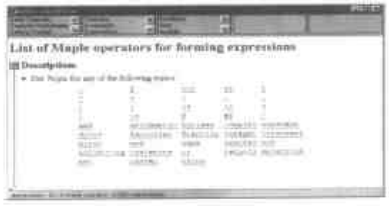
```
> whattype(f);
```

+

```
> solve(f=0,x);
                                1,1
```

从上例中可以看出，双引号 "" 定义的字符串与单引号'定义的字符串所生成的数据，在 MAPLE 中并不相同。双引号将变量定义为一个字符串型数据，而单引号的作用则相当于将变量等同于另一个变量。而且，如此例的第一条命令“a='a'”，用户可以利用这种形式清除以前对此变量的赋值。

字符串（string）类型除了其本身外，不能代表任何其他类型的数值，不能再被赋值。而标识符（symbol）则仅是一种代号（name），可以继续被赋予其他数值。根据操作系统不同，字符串及标识符都有固定的长度限制（32 位平台为 524271 个字符，在 64 位平台为 34359737335 个字符）。一般用户不会超出此限制。

上例  识符“f”定义为一个表达式。利用 whattype()函数，可以显示 如果是加、减运算，则会显示“+”，如果是乘、除运算，则会

利用 表达式：

```
> f:="string":cat(f,'0');
                              "string0"
> whattype(%);
                                string
```

1.4.2 运算符与表达式

如果将 Maple 语言同自然语言（例如英语）相对比的话，它的表达式就相当于一句话，其中的运算符号则类似话中的动词。现实中的一句话需要按照语法规则来组织，动词的位置与形态是语法的基础。在 Maple 中也有类似的情况。虽然 Maple 没有英文中动词时态、变形等规则，但正确的使用运算符号构成表达式依然是掌握它的关键。

Maple 中的运算符，除去基本的四则运算符：“+”、“-”、“*”、“/”外，数值运算符还包括“!”（阶乘）和“**”、“^”（乘方），除此以外，还有关系表达式与逻辑表达式。以下会对它们作简要介绍。有关 Maple 表达式的详细介绍，请参看联机帮助（见图 1-7）。

图 1-7 Maple 中关于表达式的帮助信息

Maple 中操作表达式的函数常用的有 convert、factor、simplify 和 expand。顾名思义，convert 可以将表达式化简成各种形式，具体将视其中参数而定；factor 可以将表达式转化成为多个因子连乘的形式；simplify 的目的是化简表达式，不过有时会起到相反的效果。Expand 则是将整个表达式展开，可以简单的理解成去括号的过程。

关系表达式由“>”（大于）、“<”（小于）、“=”（等于）、“<>”（不等于）、“>=”（不小于）和“<=”（不大于）组成。系统判断关系表达式的成立与否，返回布尔值（Boolean）“TRUE”和“FALSE”，Maple 还为关系表达式的返回值增加了“FAIL”一项，目的是通知用户无法判断表达式是否可能为真（“TRUE”）。关系表达式的应用见下例：

```
> a:=1<2;b:=x=y;c:=x<y;
```

$$a := 1 < 2$$

$$b := x = y$$

$$c := x < y$$

```
>type(a,'boolean');
```

$$true$$

```
>type(b,'equation');
```

$$true$$

```
>type(c,'equation');
```

$$false$$

```
>evalb(a);
```

$$true$$

```
>evalb(b);
```

$$false$$

```
>evalb(c);
```

$$x - y < 0$$

在上一节，我们曾提到使用“whattype()”不能判断变量是否被赋成表达式类型，这里使用新的函数“type（）”则可以判断出变量被赋予的类型，缺点是需要用户首先独立判断变量的类型。“evalb（）”的全称是“evaluate Boolean”，即判断是否为布尔型数据。如果是两个未赋值的变量比较大小，会自动输出以“<0”作为结尾的表达式。

逻辑表达式会使用到“and（与）”、“or（或）”和“not（非）”3 个逻辑关系符号。同样会返回 3 种布尔值：“TRUE”、“FALSE”和“FAIL”。系统按照表 1-2 判断返回值。

表 1-2　Maple 真值表

	and			or			not
	True	False	Fail	True	False	Fail	

续表

	and			or			not
True	True	False	Fail	True	True	True	False
False	False	False	False	True	False	Fail	True
Fail	Fail	False	Fail	True	Fail	Fail	Fail

基本的使用方法如下例：

```
> true and false;
```

$$false$$

```
> 5>3 and 2>3;
```

$$true$$

```
> (x<y) and (not(x>=y));
```

$$x - y < 0 \text{ and not } (y - x \le 0$$

```
>3<>3 or 8<>9;
```

$$true$$

1.5 Maple 6 的函数与函数库调用

函数是数学中最基本的概念。在科学、工程计算过程中，往往会定义各种类型的函数。在 Maple 中，我们可以给表达式、函数、图形赋予任何非系统保留字的名称，目的是方便引用及运算。函数基本上可以分成单变量函数、多变量函数与复合函数。

1.5.1 单变量函数

请读者先观察以下的例子：

```
> f:='f';
```

$$f := f$$

```
> f:=x->5*exp((x-2)^2);
```

$$f := x \to 5e^{((x-2)^2)}$$

```
> f(2);f(3);
```

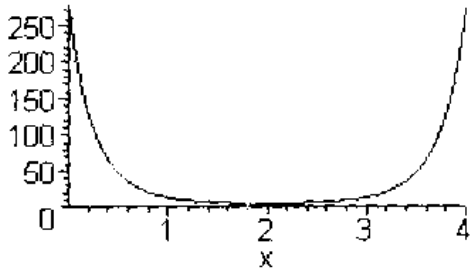
$$5$$

$$5e$$

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

$$13.59140914$$

```
> plot(f(x),x=0..4);
```



例子的第 1 条命令“f: = ' f '”的目的是清除 Maple 内存中以前对变量“f”的定义。这是为防止重复定义造成错误而采取的预防手段。如果用户想完全放弃前面的工作，则可以使用“restart”命令，它将完全清除内存中的数据。注意即使用户重新开启新的工作簿，内存中已定义好的变量依然不会被清除。我们在第 2 条命令中定义了函数 f(x)=5exp((x-2)²)，Maple 中利用“→”来设定自变量，未在函数定义前设定的变量系统会默认为参数。接下来，就可以用 f(x)的形式调用函数了。Maple 不会自动对调用的函数进行化简，而需要以命令“evalf（）”获得近似值，plot()命令可以简单地绘制函数。

1.5.2　多变量函数

定义多个变量的函数，例如 f（x，y），需要使用表达式：“(x,y) →”，见以下的例子：

```
> f:='f':
> f:=(x,y)->1-cos(x^2+y^2);
```

$$f := (x, y) \to 1 - \cos(x^2 + y^2)$$

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

$$1 - \cos(5)$$

```
> f(2*sqrt(pi),3*sqrt(pi)/2);
```

$$1 - \frac{1}{2}\sqrt{2}$$

```
> f(a^2-b^2-a^2);
```

$$1 - \cos\left((a^2 - b^2)^2 + (b^2 - a^2)^2\right)$$

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

$$1 - \cos(2a^4 - 4a^2b^2 + 2b^4)$$

1.5.3　复合函数

通过组合已经定义好的单、多变量函数，就可以获得类似“f(g(x))”的复合函数，Maple 中还可以使用“@”符号定义复合函数。比如 $f_1(f_2(\ldots f_n(x))\ldots)$可以通过“$(f_1@f_2@\ldots.@f_n)(x)$”

来定义。例如：

```
> f:='f':g:='g':h:='h':
> f:=x->x^2+x;g:=x->x+1;h:=x->sin(x)+cos(x);
```

$$f := x \to x^2 + x$$

$$g := x \to x + 1$$

$$h := x \to \sin(x) + \cos(x)$$

```
> f(g(x));(f@g)(x);
```

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

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

```
> (g@h)(pi/2);
```

$$2$$

```
> simplify((f@g@h)(x));
```

$$3 + 2\sin(x)\cos(x) + 3\sin(x) + 3(\cos(x)$$

@符号还有进一步的形式。如果我们定义函数“(f@@10)(x)”，则相当于 f(x) 自身嵌套 10 次，例如：

```
> f:='f':
> f:=x->x^2+x;
```

$$f := x \to x^2 + x$$

```
> expand((f@@3)(x));
```

$$x^8 + 4x^7 + 8x^6 + 10x^5 + 9x^4 + 6x^3 + 3x^2 + x$$

```
> factor(%);
```

$$x(x+1)(x^2+x+1)(x^4+2x^3+2x^2+x+1)$$

1.5.4 调用库函数与加载函数库

在 Maple 6.0 之前，调用 Maple 的非内部库函数，需要使用“readlib()”函数来加载库函数。而 Maple 6 在使用了新的内存管理机制后，取消了这个函数，可以直接调用所有关联库函数而无须提前声明。这一举措消除了用户因为疏忽忘记事先加载某特殊函数而产生错误结果，大大方便了用户的工作。所以用户在使用 Maple 6 后，无须关注某一函数是否是内部库函数，就可以随意使用系统支持的任意函数。同样，以前用来从内存中卸载库函数的命令“unload()”也在 Maple 6 中消失了。

同加载库函数不同，Maple 中还有“函数库（package）”这一概念。函数库是指系统事先按照功能或学术领域划分好的一组函数，它们都有特定的针对性。Maple 的函数库涉及的领域非常广泛，可以满足不同专业人员的需求。用户可以使用“?index[package]”来获得

Maple 自带的函数库列表（见图 1-8）。

Index of descriptions for packages of library functions

Description:

- The following packages are available

DEtools	differential equations tools
Domains	create domains of computation
GF	Galois Fields
GaussInt	Gaussian Integers
Groebner	Groebner basis calculations in skew algebras
LREtools	manipulate linear recurrence relations
LinearAlgebra	linear algebra package based on rtable data structures
Matlab	Matlab Link
Ore_algebra	basic calculations in algebras of linear operators
PDEtools	tools for solving partial differential equations
Spread	Spreadsheets
algcurves	Algebraic Curves
codegen	Code Generation
combinat	combinatorial functions
combstruct	combinatorial structures
context	context sensitive menus
diffalg	differential algebra

图 1-8　Maple 部分函数库

在后续章节中，我们会用到 LinearAlgebra，linalg，diffalg，plots，plottools，geometry，finance 等各种各样的函数库。可以说大量的不同专业的函数库是 Maple 的根基。

利用“with()”命令可以加载特定的函数库。例如：

```
> with(student);
```

[D, *Diff*, *Doubleint*, *Int*, *Limit*, *Lineint*, *Product*, *Sum*, *Tripleint*, *changevar*, *completesquare*, *distance*, *equate*, *integrand*, *intercept*, *intparts*, *leftbox*, *leftsum*, *makaproc*, *middlebox*, *middlesum*, *midpoint*, *powsubs*, *rightbox*, *rightsum*, *showtangent*, *simpson*, *shope*, *summand*, *trapezoid*]

如此我们就可以调用“student”函数库中定义的所有函数了。

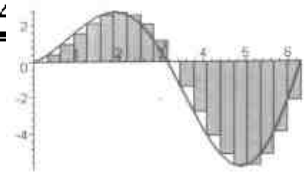
现在简单介绍“student”函数库中的函数，如下例：

```
> with(student):
> isolate(5*x+x*y=1/x+1,x);
```

$$x = \frac{1+\sqrt{21+4y}}{5+y}$$

```
> distance([a,b],[c,d]):
```

$$\sqrt{(b-d)^2+(c-a)^2}$$



```
*x+sin(x),x=0..2*Pi,20);
```

```
*z,x,y,z);
```

$$\iiint 2x + xy + yz \, dx \, dy \, dz$$

```
> value(%);
```

$$x^2yz + \frac{1}{4}x^2y^2 + \frac{1}{4}y^2z^2x$$

“student”函数库是 Maple 的程序员专门针对大、中学生常用函数而综合的。这里列出的四个函数：“isolate”，“distance”，“leftbox”，“tripleint”顾名思义，分别对应着“分离变量”、“求两点间距离”、“柱状图近似”以及“三重积分”。

相对于 Maple 5，Maple 6 对函数库又做了很大调整，将一些常用函数转换为内部函数，方便了用户调用。同时也增加了新的函数库，扩展了 Maple 的功能。读者可以阅读联机帮助获得有关函数库内函数的具体使用方法。最后，简单列出一些 Maple 中的常用到的函数库（见表 1-3）：

表 1-3　Maple 6 常用函数库

codegen	代码生成函数库
combinat	组合函数函数库
PDEtools	偏微分方程函数库
diffalg	微分代数函数库
geometry	欧氏几何函数库
group	群论相关函数库
inttrans	积分变换函数库
LinerAlgebra	基于 rtable 结构的线性代数函数库
linalg	基于数列结构的线性代数函数库
Matlab	Matlab 链接包

续表

函　数　库	说　明
plots	画图函数库
plottools	基本画图函数
simplex	线性优化函数库
stats	统计函数库
student	学生用数学函数库
tensor	张量及广义相对论函数库

第 2 章　初等数学运算

Maple 的特点在于符号运算，即利用符号形式对变量、函数、方程进行处理，并得到准确的代数结果。但它仍然具有便捷、强大的浮点运算能力。尤其是 Maple 6 在获得了 NAG 的函数库后，其浮点运算能力更得到了显著的增强。一目了然、迅速准确的数据处理能力使得它在国外被广泛用作数学教学的辅助工具。虽然在中国将 Maple 推广到初等数学教学

为一个功能强大的计算器，辅助科技、工程人员的初等数学解决策略，希望能让 Maple 尽快为您服务。

簿

在开始这一章之前，先要介绍 Maple 中的最基本单元：工作簿（worksheet）。用户在 Maple 上的绝大多数操作都是在工作簿中完成的，而 Maple 则会将处理信息在工作簿中显示。作为预备知识，让我们先了解一下工作簿的基本功能。

2.1.1　建立工作簿

这一步非常简单。Maple 会在启动主程序的时候自动建立一个空白工作簿并将其最大化显示，如图 2-1 所示。

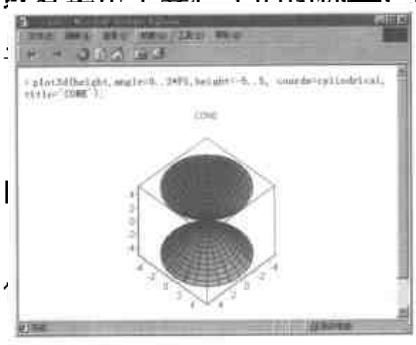
图 2-1　Maple 初始界面

如果用户希望建立新的工作簿，可以通过主控菜单中的 File->New 来进行，或者直接

在当前工作簿中按组合键 Ctrl+N，或者直接单击工具栏中的图标。建立多个工作簿可以使 Maple 并行处理多个过程（也有在同一工作簿中并行运算的方法，会在“Maple 6 程序设计”一章中介绍），也可以方便多个处理过程间的互相调用。

2.1.2　保存工作簿

为方便用户保存工作结果，Maple 工作簿有多种储存方式。如果简单地选择 File->Save 或者单击工具栏中的图标，Maple 会将工作簿缺省保存为*.mws(maple worksheet)格式，



文件，就会将它自动载入 Maple 主程序。

le->Export，Maple 工作簿还可以被储存成：

*.txt)。注意此格式不会保留任何图形方式的结果，但会近似

。

xt Markup Language, *.html)。为方便用户建立主页形式的文

的全部内容转换成超文本格式的文件（见图 2-2）。

aple Explorer, *.tex)。在安装了 Maple Explorer 插件的浏览器上可以直接调用 Maple 程序显示此种格式的文件。

（4）RTF 格式（Rich-text format, *.rtf）。Windows 标准文件格式，可以被 Microsoft Word 打开（见图 2-3）。工作表中所有用户输入部分保存为文本格式，Maple 的处理结果保存为图片格式。

图 2-2　Internet Explorer 5.0 显示的 Maple 工作簿

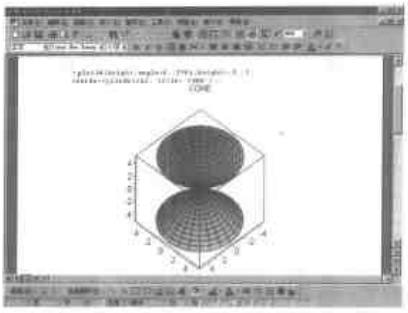


图 2-3　Microsoft Word 97 显示的 Maple 工作簿

2.1.3　自动保存

类似 Word 中的自动保存功能，Maple 同样可以自动保存用户的当前工作进度。选择 Option->Autosave 选项，会弹出自动保存参数设置菜单（见图 2-4）。当用户激活自动保存选项，并设置自动保存间隔后，Maple 会间隔固定的时间保存当前工作簿中的全部结果，并将其以“工作簿名+_MAS.mws”的形式保存在工作簿上次保存的目录。

图 2-4　自动保存设定对话框

自动保存的间隔时间从上次修改此工作簿的时间开始计算。注意有以下几种情况时，即使已被激活，Maple 也不会自动保存工作簿：

（1）工作簿没有被命名。即用户建立此工作簿后，从未人工保存过。

（2）在上次人工或自动保存后，此工作簿未被修改过。

（3）使用 Maple Explorer 调用的工作簿。

2.2　数值计算

Maple 几乎可以完成任何普通用户会涉及到的数值计算任务，绝非普通计算器可以比拟。下面就让我们一步一步地来发掘 Maple 的数值计算功能。

2.2.1　简单数值计算

先请读者观察以下的例子，虽然很短小，但涉及了很多常用技巧：

```
> 65+54-sqrt(43*(32/21)^10);
```

$$119-\frac{33554432}{4084101}\sqrt{43}$$

```
> evalf(%);%%^2;
```

$$65.12495503$$

$$\left(119-\frac{33554432}{4084101}\sqrt{43}\right)^2$$

```
> %%%:%-119;
```

$$-\frac{33554432}{4084101}\sqrt{43}$$

第一条命令，包含了基本的运算过程：加、减、乘、除、乘方、开方。一条希望被执行的 Maple 指令要以“;”作为结尾，而后按回车，命令程序分析并执行此指令。注意第一行 Maple 的处理结果中包含根式，这说明 Maple 语言只会对用户的输入信息做基本的化简处理，不会破坏其精度。

第二条指令的头一步，是希望获得化简结果的近似解，使用了“eval（）”系列指令中的“evalf()”。这个系列共包括指令 eval，evala，evalb，evalc，evalf，evalhf，evalm，evaln，evalr，目的是获得一个确定数值，读者可以参考联机帮助获得相关命令的说明。此处使用的 evalf（）是获得括号内数值的浮点数数值，是最常用的一个指令。其默认精度为小数点后 10 位有效数字。“%”符号是上一个 Maple 运行结果的替代符号，类似的“%%”则表示倒数第二次结果。可以对此代号进行任何合理的处理。这条指令中共有两个“;”，读者可以看出只有回车后 Maple 才会以“;”作为分割符处理输入数据。

第三条指令用到了“%”系列的最长指代范围：向上的第三次结果。同时它以“:”作为两条指令的分割符。“:”同“;”的区别是 Maple 不会向用户显示“:”所引导命令的结果，但会将它储存在内存中。所以在下一条语句以“%”调用上一次结果后，会得到没有显示的结果。

Maple 包括的初等数值运算指令如表 2-1 所示。

表 2-1 Maple 操作符

操 作 符	意 义
!	阶乘
+	加
-	减
*	乘
/	除
**、^	乘方
mod	取模

Maple 中，以下的几个函数常用来对浮点数进行操作：

（1）Trunc：将选择的数值化简成为最接近“0”的整数，或者说就是“取整”。

（2）Round：将选择的数值化简成邻近的整数，或者说是“四舍五入”。

（3）Frac：取出选择数值的小数部分。

（4）Floor：求小于选择数值的最小整数，或者等于原整数。

（5）Ceil：求大于选择数值的最大整数，或者等于原整数。

举例对其中的“ROUND（）”函数做简要说明，请注意以下的例子：

```
> round(5.5);
                              6
> round(5.49);
                              5
> round(5,49999999999);
                              5
> round(5.499999999999);
                              6
```

这个例子依然是提醒大家注意系统的有效位数，当输入数值超过有效位数后，系统会自动将其四舍五入后再进行处理，所以会出现以上的结果。虽然平时可能会忽视这一点，但当进行精细计算或者进行大数乘除运算时，希望大家能保持警惕。

还有需要提醒的一个问题是如果在输入数据中包含浮点数，Maple 系统会自动以“evalf”方式计算结果。例如下例：

```
>2^(2.5);
                          5.656854249
```

```
> 2^(5/2);
```

$$4\sqrt{2}$$

```
> 2.5^(1/2);
```

1.581138830

顺便介绍一下 Maple 中有关复数的运算。在上一章曾经提到，Maple 会自动处理涉及复数的运算，如计算$(-8)^{1/3}$等，而且可以用 Re（），Im（）两个函数取出复数的实部与虚部。这里再介绍一个复数运算的基本函数 polar()或 convert(expression,polar)，可将复数转化成极坐标形式。请看下例：

```
> a:=(-8)^0.3;
```

$$a := 1.096846065 + 1.509679093\,I$$

```
> Re(a);Im(a);
```

1.096846065

1.509679093

```
> convert(a,polar);
```

polar(1.866065983,.9424777959)

```
> polar(a);
```

polar(1.866065983,.9424777959)

在所显示的极坐标中，第一个参数是极轴半径，即所表示的复数的模；第二个参数表示在极坐标中的倾角，显示的是角度的反正切（arctg）值。注意到由于在例子的第一步输入的是$-8^{0.3}$，所以以后的过程显示都为是近似值，如果第一步输入的是$-8^{3/10}$，以后的处理结果都将会是精确值。

2.2.2　连加与连乘

相信很多人小时候都听说过高斯是如何计算从 1 加到 100 的。现在我们让 Maple 来完成这项任务：

```
> sum(i,i=1..100);
```

5050

```
> add(i,i=1..100);
```

5050

也许有人会问，为什么会设计两个函数来完成同一种功能？请看以下的例子：

```
> sum(i,i=1..n);
```

$$\frac{1}{2}(n+1)^2 - \frac{1}{2}n - \frac{1}{2}$$

```
> factor(%);
```

$$\frac{1}{2}n(n+1)$$

```
> sum(i^2,i=1..n);
```

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

```
> factor(%);
```

$$\frac{1}{6}n(n+1)(2n+1)$$

```
> sum(a[i]*x^i,i=0..4);
```

$$a_0+a_1x+a_2x^2+a_3x^3+a_4x^4$$

```
> sum(a[i]*x^i,i=0..n);
```

$$\sum_{i=0}^{n}a_ix^i$$

```
> add(i,i=1..n);
Error, unable to execute add
```

借助多项式化简函数“factor（）”的帮助，函数“sum（）”让我们得到了自然数求和递推公式以及自然数平方和递推公式。可以看出“sum（）”等价于求和符号“Σ”。然而“add（）”函数却不能完成将连加推广到任意数“n”这项功能，因为“add（）”函数有自己独特的功能，即对数组元素进行累加，如下例：

```
> S:=[seq(i,i=1..10)];
```

$$S:=[1,2,3,4,5,6,7,8,9,10]$$

```
> add(i^2, i=S);
```

$$385$$

```
> add(a[i]*i,i=S);
```

$$a_1+2a_2+3a_3+4a_4+5a_5+6a_6+7a_7+8a_8+9a_9+10a_{10}$$

```
> sum(i^2,i=S);
Error, (in sum) second argument must be a name or
name=a..b or name=RootOf
```

最后要说明的是 Maple 中还有另外一个函数“Sum（）”，它同“sum（）”不只区别在首字母是大写形式。使用“Sum（）”只会得到求和式的解析表示，而不会得到数值解，即完全相当于求和符号“Σ”，但可以利用“Times New Roman（）”求出数值解。请看下例：

```
> Sum((1+2*i^3)/(i+4)^5,i=6..7);
```

$$\sum_{i=6}^{7}\frac{1+2i^3}{(i+4)^5}$$

```
> value(%);
```

$$\frac{138435083}{16105100000}$$

```
> sum((1+2*i^3)/(i+4)^5,i=6..7);
```

$$\frac{138435083}{16105100000}$$

如果不定义“sum（）”中变量i的取值范围，则系统自动会选择从1到i-1，如下例：

```
> sum(i,i);
```

$$\frac{1}{2}i^2-\frac{1}{2}i$$

```
> sum(i,i=0..i-1);
```

$$\frac{1}{2}i^2-\frac{1}{2}i$$

最后，有限求和推广到无限求和. Maple使用“infinity”和“-infinity”代表“+∞”和“-∞”，例如求表达式 $\sum_{i=0}^{\infty}\frac{1}{i!}$：

```
> sum(1/i!,i=0..infinity);
```

$$\sum_{i=0}^{\infty}\frac{1}{i!}$$

```
> value(%);
```

$$e$$

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

$$2.718281828$$

读者可以同时从上例中对比得出“value（）”与“evalf（）”两个命令的区别。

对于连乘函数“prudoct（）”与“mul（）”，其作用及区别完全同“sum”与“add（）”一一对应，同样“product（）”也有大写形式“Product（）”，这里就不再举例。

2.3　多项式及其运算

多项式是初等数学的基本组成元素，它可以定义成为由多个单项式依靠基本算符连接而成的算式。Maple 中针对多项式的函数有很多，这一节我们将循序渐进地对它们一一介绍。

2.3.1 基本运算

包括四则运算“+ - * /”及乘方“^”，用户可以简单的对多项式进行基本运算操作，如：

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

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

```
> b:=2*x;
```

$$b := 2x$$

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

$$true$$

```
> type(a+b,polynom);
```

$$true$$

```
> a+b;
```

$$x^2 - 1 + 2x$$

```
> a*b;
```

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

```
> a^b;
```

$$(x^2 - 1)^{(2x)}$$

```
> a/(x+1);
```

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

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

$$x - 1$$

```
> type(a*sin(x),polynom(anything,x));
```

$$false$$

这里使用了“type（）”命令来判断变量所属的类型。最后一条命令是判断表达式“（x^2-1）*sin(x)”中的变量“x”是否可以取任意数值。当然返回值是“false”。

2.3.2 多项式的展开、化简与因式分解

这是 Maple 强大的符号处理功能在初等代数中的第一次体现。快速准确的对多项式进行化简与因式分解常常会对实际问题起到关键的帮助。涉及到的命令有“expand（）”展开、“sort”按阶数排序、“factor（）”提取公因式、“simplify（）”化简等。请读者从以下的例子中观察这几个函数的使用方法：

```
> a:=(x-1)^8;expand(a);
```

$$a := (x-1)^8$$

$$x^8 - 8x^7 + 28x^6 - 56x^5 + 70x^4 - 56x^3 + 28x^2 - 8x + 1$$

```
> b:=((x-1)^3+x*(x-1)^2+1);sort(b);
```

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

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

```
> expand(%);sort(%);
```

$$2x^3 - 5x^2 + 4x$$

$$2x^3 - 5x^2 + 4x$$

从这里可以看出 expand()会自动按照变量的阶数排序显示，sort()只对原始公式排序。factor()函数会对多项式提取公因式。基本用法如下：

```
> p1:=x^2+y^2-2*x*y;
```

$$p1 := x^2 + y^2 - 2xy$$

```
> factor(p1);
```

$$(x-y)^2$$

```
> p2:=a^3+b^3+c^3-3*a*b*c;
```

$$p2 := a^3 + b^3 + c^3 - 3abc$$

```
> factor(p2);
```

$$(b+a+c)(b^2 - ab - bc - ac + a^2 + c^2)$$

在默认设置下，factor()只会将多项式在整数范围内分解因式，如果用户想进一步在实数或复数范围内分解，就要设置参数，如：

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

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

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

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

simplify()是最常用的一条命令。基本用法如下：

```
> e:='e':f:='f';
```

$$f := f$$

```
> e:=sin(x)^2+cos(x)^2;
```

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

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

$$1$$

让我们看一个在实际过程中遇到的 Maple 对复杂公式的化简：

```
> k:=-(g+t*diff(g,t))/(2*f+t*diff(f,t));
```

$$k := -\frac{1}{2}\left(\frac{-1-\frac{1}{t}+\frac{(1+t)e^{(-2t)}}{t}-e^{(-t)}(1+t)}{1+e^{(-t)}(1+t)}+t\left(\frac{-\frac{1}{t^2}-\frac{(1+t)e^{(-2t)}}{t^2}+\frac{e^{(-2t)}}{t}-\frac{2(1+t)e^{(-2t)}}{t}+e^{(-t)}(1+t)-e^{(-t)}}{1+e^{(-t)}(1+t)}-\frac{\left(-1-\frac{1}{t}+\frac{(1+t)e^{(-2t)}}{t}-e^{(-t)}(1+t)\right)\left(-e^{(-t)}(1+t)+e^{(-t)}\right)}{(1+e^{(-t)}(1+t))^2}\right)\right)/f$$

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

$$\frac{1}{2}\frac{1+2e^{(-t)}+3e^{(-t)}t+e^{(-3t)}+2e^{(-3t)}t+4e^{(-2t)}t+e^{(-2t)}t^2+2e^{(-2t)}+e^{(-3t)}t^2}{(1+e^{(-t)}+e^{(-t)}t)^2 f}$$

然而有时候 Maple 自动化简的结果也并不会令人满意，如继续对上例操作：

```
> subs(R=t/k,E);
>
```

$$-\frac{1}{2}k^2+\frac{k^2-k-\frac{k}{t}+\frac{k(1+t)e^{(-2t)}}{t}+k(k-2)(1+t)e^{(-t)}}{1+e^{(-t)}\left(1+t+\frac{1}{3}k^2r^2\right)}$$

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

$$-\frac{1}{2}k(-3kt-3e^{(-t)}tk-3e^{(-t)}t^2k+k^3te^{(-t)}r^2+6t+6-6e^{(-2t)}-6e^{(-2t)}t+12e^{(-t)}t+12e^{(-t)}t^2)/(t(3+3e^{(-t)}+3e^{(-t)}t+e^{(-t)}k^2r^2))$$

但养成经常输入“simplify(%)”的习惯会给用户带来意想不到的好处。

2.3.3　其他常用函数

Maple 中提供了 50 余个可以对多项式或表达式操作的函数，先看一个例子：

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

$$p := x^4 - x^2y^2 + x^3y + xy^3 - 2y^4$$

```
> sort(p,y);
```

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

```
> coeff(p,y);
```

$$x^3$$

```
> coeffs(p,y);
```

$$x^4, x^3, -2, -x^2, x$$

```
> op(P);
```

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

```
> Q:='Q':Q:=x^2-4*x*y^2+2*x*y-8*y^3;
```

$$Q := x^2 - 4xy^2 + 2xy - 8y^3$$

```
> gcd(P,Q);
```

$$x + 2y$$

sort()函数在前面曾经提过，它按照选定参数的由高到低的阶数对原多项式重新排序。

coeff()函数可以提取多项式中特定变量的系数，但它不会处理变量的其他阶数。

coeffs()函数则可以按照指定变量阶数从低到高的顺序显示其系数。

gcd()则可以显示出多组多项式的最大公因子（greatest common divisor），同它类似的还有 lcm()(least common multiple)，它用来求最小公倍数。

op()函数的缺省作用是按照原始输入顺序提取出多项式中的所有单项并显示。它的完全表达形式为 op(I,exprssion)当“I”为大于 0 的自然数时，它返回表达式中的第 I 项，当 I=0 时，它返回表达式的类型。例如“symbol”“+”“*”“<”等，同 whattype(expression)的返回值相同。与之相关的函数还有“nops()”：用以显示表达式中算子个数；“subs()”：用以替换表达式中的某一项。请看以下的例子：

```
> f:='f':f:=x+y^2+x*y-exp(x);
```

$$f := x + y^2 + xy - e^x$$

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

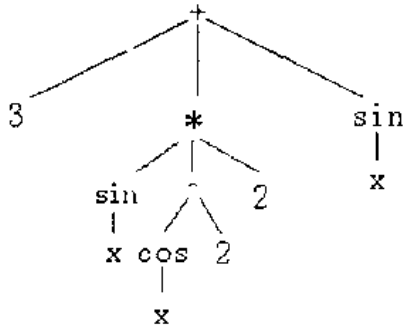
$$+$$

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

$$y^2$$

```
> nops(f);
                          4
> op(0..4,f);
                    + x, y^2, xy, -e^x
> subsop(2=sin(y),f);
                  x + sin(y) + xy - e^x
> op(0..4,%);
                    +, x, sin(y), xy, -e^x
```

这里顺便对 Maple 处理表达式的顺序作一简要介绍。根据数学运算符的优先级，系统自动对表达式建立一套树状处理顺序表，并存储在内部。最后进行的运算符在树的根部，同一运算级的项位于树的同一层，对每一项依次递归构造运算符的表达式树。例如运算函数：sin(x)+2*sin(x)*cos(x)^2+3，表达式树如下：



再看一个有关 subs()，替代函数的例子：

【例 2-1】 已知函数 E:

$$E=-\frac{1}{2}k^2+\frac{k^2-k-R^{-1}(1+kR)e^{-2kR}+k(k-2)(1+kR)e^{-kR}}{1+e^{-kR}(1+kR+k^2R^2/3)}$$

同时 E 还可以表示成 $E=k^2F(t)+kG(t)$，$t=kR$，求 F(t),G(t)。

第一步，用 subs()进行变量替换 R->t/k:

```
> E:='E';
                          E := E
> E:=-1/2*k^2+(k^2-k-1/R+1/R*(1+k*R)*exp(-2*k*R)+k*(k-2)
*(1+k*R)*exp(-k*R))/(1+exp(-k*R)*(1+k*R+k^2*R^2/3));
```

$$E:=-\frac{1}{2}k^2+\frac{k^2-k-\frac{1}{R}+\frac{(1+kR)e^{(-2kR)}}{R}+k(k-2)(1+kR)e^{(-kR)}}{1+e^{(-kR)}\left(1+kR+\frac{1}{3}k^2R^2\right)}$$

```
> subs(R=t/k, E);
```

$$-\frac{1}{2}k^2+\frac{k^2-k-\dfrac{k}{t}+\dfrac{k(1+t)\,e^{(-2t)}}{t}+k(k-2)(1+t)\,e^{(-t)}}{1+e^{(-t)}\left(1+t+\frac{1}{3}t^2\right)}$$

第二步，将 k、k^2 分别替换为 0、1，得到 F（t）：

```
> F:='F';
```

$$F := F$$

```
> F:=subs(k^2=1,k=0,%%);
```

$$F := -\frac{1}{2}+\frac{1}{1+e^{(-t)}\left(1+t+\frac{1}{3}t^2\right)}$$

注意替换的顺序，如果先替换 k，则所有 k^2 项也会被替换。

类似的，将 k、k^2 分别替换为 1、0，则可以获得 G（t）。

同多项式相关的常用函数还有：

degree：　求多项式的阶数。

collect：　将多项式按特定项幂次展开。

content：　按所给变量提出表达式中的公因式。

divide：　判断多项式相除是否能精确除尽，如果可能，保存其相除结果。

2.4　初等代数变换

这一章主要对 Maple 中的 convert（）函数进行初步介绍。convert（）函数在高等数学中也有非常重要的应用，如进行坐标变换、拉普拉斯（Laplace）变换、贝塞耳（Bessel）变换、傅立叶（Fourier）变换、矩阵变换等，有些会在后续章节中介绍。此节只介绍简单代数变换。

请看以下两组例子：

```
> convert(1000,binary);
```

1111101000

```
> convert(1000,hex);
```

3E8

```
> convert(`3E8`,decimal,hex);
```

1000

```
> convert(1.23456,fraction);
```

$$\frac{3858}{3125}$$

```
> convert(1234/5678,float);
```

$$.2173300458$$

前三条命令是对数值进行进制转换：十进制转换为二进制；十进制转换为十六进制；十六进制转换为十进制。注意输入十六进制时需要以``（即键盘上与~符号同处一个键的符号）括起数据。后两条命令是进行浮点数与分数间的互相转换。再看一个例子：

```
> f:='f':f:=seq(x[i]^i,=1..4);
```

$$f := x_1, x_2^2, x_3^3, x_4^4$$

```
> convert([f],`*`);
```

$$x_1 x_2^2 x_3^3 x_4^4$$

```
> g:='g':g:=(x^6-1)/((x-1)^6);
```

$$g := \frac{x^6-1}{(x-1)^6}$$

```
> convert(g,parfrac,x);
```

$$1+\frac{6}{(x-1)^5}+\frac{15}{(x-1)^4}+\frac{20}{(x-1)^3}+\frac{15}{(x-2)^2}+\frac{6}{x-1}$$

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

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

```
> simplify(g);
```

$$\frac{x^5+x^4+x^3+x^2+x+1}{(x-1)^5}$$

第一条命令将一个序列转化为连乘的形式，同样，将参数改换为“+”就可以得到序列的连加形式。第二条命令将一个多项式转化成为分式形式，有利于对原多项式进行积分运算。注意此结果有异于利用 factor()函数及 simplify()函数处理后的结果，不同的方法针对的问题各不相同。

初等变换还包括对包含根式的分式对其分母进行有理化，用到 rationalize（）函数，例如：

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

$$\frac{1+2^{\left(\frac{1}{3}\right)}}{1-2^{\left(\frac{1}{3}\right)}}$$

```
> rationalize(%);
```

$$-\left(1+2^{\left(\frac{1}{3}\right)}\right)\left(1+2^{\left(\frac{1}{3}\right)}+2^{\left(\frac{2}{3}\right)}\right)$$

```
> [(x+y)/(x+sqrt(y)),x*y/(x+sqrt(x+sqrt(3)))];
```

$$\left[\frac{x+y}{x+\sqrt{y}},\frac{xy}{x+\sqrt{x+\sqrt{3}}}\right]$$

```
>rationalize(%);
```

$$\left[\frac{(x+y)(x-\sqrt{y})}{x^2-y},\frac{xy(x-\sqrt{x+\sqrt{3}})(x^2-x+\sqrt{3})}{x^4-2x^3+x^2-3}\right]$$

最后，介绍一下对分式的通分与化简。Maple 中与之相关的命令有 normal()、numer()和 denom()。如下例：

```
> f:='f':f:=1/x+1/x^2+1/x^3;
```

$$f:=\frac{1}{x}+\frac{1}{x^2}+\frac{1}{x^3}$$

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

$$\frac{x^2+x+1}{x3}$$

```
> numer(f);
```

$$x^2+x+1$$

```
> denom(f);
```

$$x^3$$

normal()的功能在一定程度上同 simplify()类似，但它一般只进行简单的通分与合并同类项，有时并不能将表达式完全化简。numer()的功能是提取表达式的分子，denom()则是提取其分母。读者不难发现，这两个函数都是在将原表达式通分后再执行的。

2.5　三角函数及其变换

由于三角函数在初等数学中起着关键作用，所以我们单独用一节来介绍 Maple 中与三角函数相关的命令。希望读者能尽快掌握它们的使用方法并应用到实际问题中。

2.5.1　三角函数及反三角函数

三角函数及其反函数在 Maple 中对应的命令如表 2-2 所示：

表 2-2　Maple 中的三角函数

sin(x)	正　弦	cos(x)	余　弦
arcsin(x)	反正弦	arccos(x)	反余弦
sec(x)	正割	csc(x)	余割
arcsec(x)	反正割	arccsc(x)	反余割
tan(x)	正切	cot(x)	余切
arctan(x)	反正切	arccot(x)	反余切

默认情况下，系统会以弧度值作为计算参数（1 弧度=180/π 角度;1°≈0.0175 弧度），利用系统提供的基本函数，用户可以完成如下的计算：

```
> sin(0);
```

$$0$$

```
> cos(-Pi);
```

$$-1$$

```
> sin(Pi/3);
```

$$\frac{1}{2}\sqrt{3}$$

```
> sec(1);
```

$$\sec(1)$$

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

$$1.850815718$$

```
> sec(1.0);
```

$$1.850815718$$

```
> sin(arccos(1/2));
```

$$\frac{1}{2}\sqrt{3}$$

用户如果希望计算角度，则需要以其同 π 的比例作为参数输入，例如第三条命令中的“Pi/3”，对任意角度 θ，通用形式为“θ*Pi/180”。同样，Maple 在默认情况下不会计算三角函数的近似值，只会将特殊的角度化简为实数形式，常规方法是用 evalf()函数显示近似值，但用户可以简单地将输入数据变化为类似例中第六条命令所给出的形式，从而直接获得近似实数解。对于角度，也可以用“θ*Pi/180.0”的形式输入。

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

$$360\frac{degrees}{\pi}$$

```
> convert(Pi/6,degrees);
```

$$30degrees$$

```
> convert(90*degrees,radians);
```

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

Maple 可以利用 convert（）函数进行“角度<->弧度”间的相互转化，如：

读者会发现此种方法输入起来十分复杂，而且结果也不直观，相信中国读者均会利用上文所给出的互换公式自行计算。

反三角函数同三角函数的用法类似，它会以弧度值返回。需要指出的是，反三角函数同样可以用复合函数“sin@@(-1)”的形式得到。如下例：

```
> arcsin(1/2);
```

$$\frac{1}{6}\pi$$

```
> arcsin(0.5);
```

$$.5235987756$$

```
> (sin@@(-1))(1/2);
```

$$\frac{1}{6}\pi$$

```
> (sin@@(-2))(1/2);
```

$$\arcsin\left(\frac{1}{6}\pi\right)$$

利用三角函数与反三角函数的组合，我们会得到一些基本的三角恒等变化公式，例如：

```
> sin(arctan(x));
```

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

2.5.2　恒等变换与三角函数展开

三角函数的恒等变换是高中数学的重点之一。Maple 可以帮助用户完成“积化和差”；“和角、差角”；“半角、倍角”等一系列工作。请看下例：

```
> sin(x)*cos(x);combine(%);
```

$$\sin(x)\cos(x)$$

$$\frac{1}{2}\sin(2x)$$

```
> sin(x+y);expand(%);
```

$$\sin(x+y)$$

$$\sin(x)\cos(y)+\cos(x)\sin(y)$$

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

$$\sin(x+y)$$

```
> cot(x-y);expand(%);
```

$$\cot(x-y)$$

$$\frac{-\cot(x)\cot(y)-1}{\cot(x)-\cot(y)}$$

```
> sin(x)^2;combine(%);
```

$$\sin(x)^2$$

$$\frac{1}{2}-\frac{1}{2}\cos(2x)$$

```
> cos(3*x);expand(%);
```

$$\cos(3x)$$

$$4\cos(x)^3-3\cos(x)$$

不难发现，这里只使用了两个命令“combine（）”与“expand（）”，这两个命令可以简单地理解为 combine()将三角函数间的运算关系变换为角度间的运算关系，而 expand()则正好相反，将角度间的运算关系转换为函数间的关系。希望读者能灵活运用它们。

利用 convert（）函数，Maple 可以将三角函数方程完全以 tan(x)的形式表达，或化为只含有 sin(x)、cos(x)的方程。如下例所示：

```
> sec(x)*cot(x)+sin(x)*cos(x);convert(&,tan);
```

$$\sec(x)\cot(x)+\sin(x)\cos(x)$$

$$\frac{1+\tan\left(\frac{1}{2}x\right)^2}{\left(1-\tan\left(\frac{1}{2}x\right)^2\right)\tan(x)}+\frac{2\tan\left(\frac{1}{2}x\right)\left(1-\tan\left(\frac{1}{2}x\right)^2\right)}{\left(1+\tan\left(\frac{1}{2}x\right)^2\right)^2}$$

```
> convert(%, sincos);
```

$$\sec(x)\cot(x)+\sin(x)\cos(x)$$

$$\frac{\left(1+\frac{(1-\cos(x))^2}{\sin(x)^2}\right)\cos(x)}{\left(1-\frac{(1-\cos(x))^2}{\sin(x)^2}\right)\sin(x)}+\frac{2(1-\cos(x))\left(1-\frac{(1-\cos(x))^2}{\sin(x)^2}\right)}{\sin(x)\left(1+\frac{(1-\cos(x))^2}{\sin(x)^2}\right)^2}$$

```
> simplity(%);
```

$$-\frac{\cos(x)^3-\cos(x)-1}{\sin(x)}$$

作为另一种转换方式，Maple 可以进行三角函数与指数间的互换，如下例所示：

```
> convert(sec(x),exp);simplify(%);
>
```

$$2\frac{1}{e^{(Ix)}+\frac{1}{e^{(Ix)}}}$$

$$2\frac{e^{(Ix)}}{e^{(2Ix)}+1}$$

```
> sin(x)*cos(x)+tan(x);convert(%,exp);
```

$$\sin(x)\cos(x)+\tan(x)$$

$$-\frac{1}{2}I\left(e^{(Ix)}-\frac{1}{e^{(Ix)}}\right)\left(\frac{1}{2}e^{(Ix)}+\frac{\frac{1}{2}}{e^{(Ix)}}\right)+\frac{-I\left((e^{(Ix)})^2-1\right)}{(e^{(Ix)})^2+1}$$

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

$$\frac{\frac{1}{4}I(e^{(4Ix)}+6e^{(2Ix)}+1)(-1+e^{(-2Ix)})}{e^{(2Ix)}+1}$$

```
> convert(1/4*exp(x)^2-1/4/exp(x)^2,tring);
```

$$\frac{1}{4}(\cosh(x)+\sinh(x))2-\frac{1}{4}\frac{1}{(\cosh(x)+\sinh(x))^2}$$

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

$$\sinh(x)\cosh(x)$$

同样，利用 convert(expression, ln)函数，可以将反三角函数以自然对数的形式展开，如：

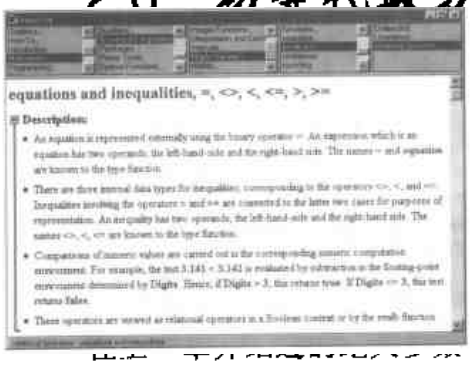
```
> convert(arcsin(x),ln);
```

$$-I\ln(\sqrt{1-x^2}+Ix)$$

```
> convert(arctan(x),ln);
```

$$\frac{1}{2}I(\ln(1-Ix)-\ln(1+Ix))$$

2.6 初等代数方程



不等式，可以说是初等代数中最基本的问题。利用等号
而形成的算术形式被定义成等式，类似的，利用小于号
“<>”连接起来的多项式或数值被定义为不等式（Maple
于等于号“>=”连接起来的多项式调换位置后转变为使
个过程不受用户的控制）。

Maple 中处理、化简多项式的函数都可以应用到对方程的变形和处理中。同时 Maple 还提供了丰富的求解方程、不等式、判断函数类型等函数以方便用户的工作。在这一节，我们将着重介绍如何获得方程的解，以及 Maple 中涉及方程的一些基本二维图形显示功能。Maple 还可以求多项式的零点，这个功能完全相当于求方程的解，这里也将一并介绍。要获得对 Maple 同方程、不等式相关的帮助，用户可以输入“?equation”，会见到如图 2-5 所示的画面。

图 2-5 输入“?equation”后的画面

2.6.1　一元方程的求解

只有一个未知数的方程就是一元方程。Maple 可以求一元方程的精确解，当系统利用通常的方法无法获得方程的精确解析解时，系统默认以“RootOf 表达式”的形式表示，但 Maple 还可以对这种形式的根进一步求其数值解。

虽然简单获得一个方程的根在 Maple 中十分简单，但我们希望读者能够对如何完成方程的恒等变形有一定了解，为真正的“精通”Maple 打好基础，所以下面我们将利用“配方”法求解初等代数中最基本的一元二次方程：“$ax^2+bx+c=0$”，其中将涉及到几乎所有的恒等变换函数，请看下例：

(1)
```
> restart;
```

(2)
```
> eq:=a*x^2+b*x+c=0;
```
$$eq := ax^2 + bx + c = 0$$

(3)
```
> student[completesquare](eq,x);
```
$$a\left(x+\frac{\frac{1}{2}b}{a}\right)^2-\frac{1}{4}\frac{b^2}{a}+c=0$$

(4)
```
> isolate(%,(x+b/(2*a))^2);
```
$$\left(x+\frac{\frac{1}{2}b}{a}\right)^2=\frac{\frac{1}{4}\frac{b^2}{a}-c}{a}$$

(5)
```
> lhs(%)=normal(rhs(%));
```
$$\left(x+\frac{\frac{1}{2}b}{a}\right)^2=-\frac{1}{4}\frac{-b^2+4ca}{a^2}$$

(6)
```
> map(sqrt,%);
```
$$\sqrt{\left(x+\frac{\frac{1}{2}b}{a}\right)^2}=\frac{1}{2}\sqrt{-\frac{-b^2+4ca}{a^2}}$$

(7)
```
> isolate(%,x);
```
$$x=\sqrt{-\frac{1}{4}\frac{-b^2+4ca}{a^2}}-\frac{1}{2}\frac{b}{a}$$

(8)
```
> map(simplify,%);
```
$$x=\frac{1}{2}\frac{\sqrt{\frac{-b^2+4ca}{a^2}}a-b}{a}$$

现在我们一步步分析上例中使用的函数：

第一步，“restart”，曾经在第一章提到过这个函数。运行“restart”清空内存是在用户准备进行常过程推导或运算之前最应该做的工作。不注意的话经常会被重复定义函数带来的错误所困扰。

第二步，“eq:=a*x^2+b*x+c=0”，定义一个方程。方程的定义同第 1 章介绍的定义函数的方法类似。定义方程目的是为今后引用方便，注意在定义方程的时候系统不会自动区分未知函数与它的系数。

第三步，“student[completesquare](eq,x)”，调用“student”程序包中的“completesquare()”函数对方程进行配方。同样，在第一章曾经简单介绍过“student”程序包，包含在其中的函数几乎都是初等数学或微积分、线性代数中的常用函数，可以说是专为大学生准备的。曾有书对它进行专门介绍（《Calculus with Maple V》，作者是 J. S. Devitt，于 1993 年由 Brooks/Cole Publishing Co.出版发行）。其中一些函数的基本功能如表 2-3 所示：

表 2-3 Student 函数包列表

函 数	功 能	函 数	功 能
D	微分(Diff 简化形式)	Diff	微分
Int	积分	Limit	求极限
Minimize	求最小值	Maxmize	求最大值
Product	连乘	Sum	连加
Doubleint	双重积分	Tripleint	三重积分
changevar	积分中的变量替换	combine	组合变形
completesquare	配方	distance	求两点距离
Equate	定义等式	Extrema	寻找方程的最值
Integrand	返回方程的最小值	summand	返回方程的最大值
Intparts	分步积分	Isolate	分离变量
Leftbox	柱状图拟合曲线	intersect	计算两条曲线的交点

其中很多函数都会在后续章节中介绍，这里只希望读者对此函数库的概貌有所了解。

第四步，“isolate(%,(x+b/(2*a))^2)”，将已配方的方程分离变量。使用 isolate()函数的时候要注意：

（1）它不会对所指定参数的不同阶区别对待，所以如果将它直接应用于一个方程，*Maple* 将返回第一个它可以求出的解，如果不能求出精确解，将以“Rootof()”的形式返回。如：

```
> isolate(x^2+3*x-4=0,x);
```

$$x=-4$$

```
> isolate(x^5+1=0,x);
```

$$x=\text{RootOf}(\_Z^5+1)$$

（2）isolate（）不会自动寻找方程中是否包含某项因式，所以如果希望做因式分解，还需要使用 factor()函数。如：

```
> factor(x^2+3*x-4=0);
```

$$(x+4)(x-1)=0$$

```
> isolate(x^5+1=0,x);
Error, (in isolate) x^2+3*x-4 = 0, does not contain,
x-1
```

第五步，“lhs(%)=normal(rhs(%))”，利用 normal 函数对方程的右式进行化简。这里使用两个函数“lhs()”，“rhs()”分别取出方程等号的左右两项（“left hand side”，“right hand side”）。希望读者注意：**使用这组函数时有可能破坏等式的成立**。

第六步，“map(sqrt,%)”，将“sqrt()”函数同时作用在等式的两边。由于是对方程两边同时操作，所以它不会影响等式的成立。map（）函数不只能对等式进行映射处理，在线性代数一章，我们会经常使用它对整个列表、数组、矩阵的每一个元素调用某一函数。具体操作步骤会在那时介绍。

第七步，“isolate(%,x)”，继续分离变量，将方程化简为变量 x 的表达式。

第八步，“map(simplify,%)”，将等式左右化简。

虽然结果并不是我们所熟知的一元二次方程标准解 $x=\dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$，但通过如此复杂的步骤，读者应该了解到各种函数的基本使用方法，相当于为今后用 Maple 解决复杂问题做一些基本训练。

其实，Maple 为解方程提供了一个简单的的函数：solve()，上个例子可以通过如下命令一步解决：

```
> restart:eq:=a*x^2+b*x+c=0;
```

$$eq:=ax^2+bx+c=0$$

```
> solve(eq,x);
```

$$\frac{1}{2}\frac{-b+\sqrt{b^2-4ac}}{a},\frac{1}{2}\frac{-b-\sqrt{b^2-4ac}}{a}$$

solve(equation,x)是一个“傻瓜”函数，用户只需设定等式（equation）和需要求解的变

量（x）两个参数，就可以直接获得方程的解。利用 solve()获得方程的解可以有多种返回形式，如下例所示：

```
> eq:=x^4-*x^2+6*x=2;
```

$$eq := x^4 - 5x^2 + 6x = 2$$

```
> solve(eq,x);
```

$$1,1,-1+\sqrt{3},-1-\sqrt{3}$$

```
> solve(eq,{x});
```

$$\{x=1\},\{x=1\},\{x=-1+\sqrt{3}\},\{x=-1-\sqrt{3}\}$$

```
> sov:={solve(eq,x);
```

$$sov := \{1,-1+\sqrt{3},-1-\sqrt{3}\}$$

```
> sov[2];
```

$$-1+\sqrt{3}$$

未加任何参数前，系统的默认返回形式为一组数列，用户也可以利用“{}”获得更形象的返回形式，如第三条命令。同时，Maple 也会将解转化为集合形式，在这种情况下，重复的解会被舍去。

利用 subs()函数、eval()函数，或者联合使用 lhs()函数、rhs()函数，我们都可以获得对方程根的检验。请看下例：

```
> eq:='eq':eq:=x^4+3*x^3+6*x=3*x^3+5*x^2+2;
```

$$eq := x^4 + 3x^3 + 6x = 3x^3 + 5x^2 + 2$$

```
> solve(eq,x);
```

$$1,1,-1+\sqrt{3},-1-\sqrt{3}$$

```
> subs(x=1,eq);
```

$$10=10$$

```
> eval(eq,x=1);
```

$$10=10$$

```
> subs(x=-1+sqrt(3),eq);
```

$$(-1+\sqrt{3})^4+3(-1+\sqrt{3})^3-6+6\sqrt{3}=$$
$$3(-1+\sqrt{3})^3+5(-1+\sqrt{3})^2+2$$

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

$$-8+8\sqrt{3}=-8+8\sqrt{3}$$

注意 subs()函数同 eval()函数的参数位置正好相反。

在一些情况下，我们无法获得方程的精确解，可以利用作图法寻找近似解。例如对上

例中的方程略作变形后求解：

```
> eq:='eq':eq:=x^4+3*x^3+6*x=3x*^3-5*x^2+2;
```

$$eq := x^4 + 3x^3 + 6x = 3x^3 - 5x^2 + 2$$

```
> solve(eq,x);
```

$$\text{RootOf}(\_Z^4 + 6\_Z + 5\_Z^2 - 2, index = 1),\ \text{RootOf}(\_Z^4 + 6\_Z + 5\_Z^2 - 2, index = 2),\ \text{RootOf}(\_Z^4 + 6\_Z + 5\_Z^2 - 2, index = 3),\ \text{RootOf}(\_Z^4 + 6\_Z + 5\_Z^2 - 2, index = 4)$$

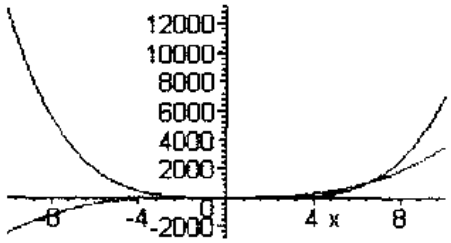
```
> f:='f':f:=lhs(eq);
```

$$f := x^4 - 3x^3 + 6x$$

```
> g:='g':g:=rhs(eq);
```

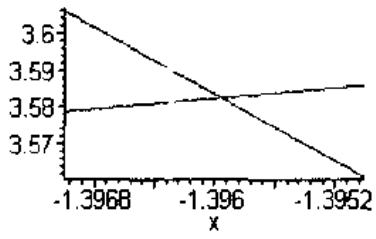
$$g := 3x^3 + 5x^2 + 2$$

```
> plot({f,g},x,color=[red,blue]);
```



利用对感兴趣区域的逐渐放大，也可以获得较精确的数值解，如对上例多步放大后，我们可以获得下图：

```
> plot({f,g},x=-1.397..-1.395,color=[red,blue]);
```



注意到这种方法只能获得方程的实数解，如果要获得方程的所有数值解，就需要借助函数 fsolve()的帮助，例如同样求解上例中的方程：

```
> fsolve(eq,x);
```

$$-1.395958206, 6.624899317$$

```
> fsolve(eq,{x},complex);
```

$$\{x=-1.395958206\},\{x=.3855294444-.2600540183\,I\},$$
$$\{x=.3855294444+.2600540183\,I\},\{x=6.624899317\}$$

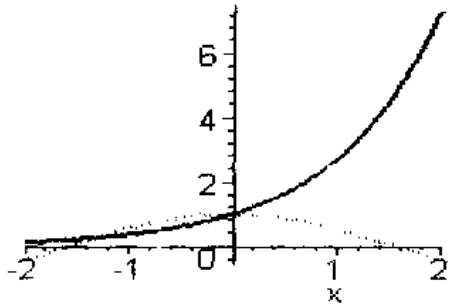
然而 fsolve()有时并不能返回方程的所有解，例如解超越方程 $e^x=\cos(x)$时：

```
> fsolve(exq(x)=cos(x),x);
```

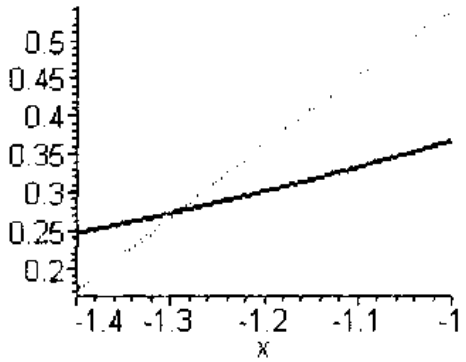
$$0.$$

虽然 x=0 是原方程的解，但通过作图，我们发现此方程还有另解：

```
> plot({exp(x),cos(x)},x=-2..2,linestyle=[1,2],
  thickness=2);
```




```
> plot({exp(x),cos(x)},x=-1.4..-1,linestyle=[1,2],
  thickness=2);
```



只要对 fsolve()指定新的求值范围，就可以获得此解的近似值：

```
> fsolve(exp(x)=cos(x),x=-2..-1);
```

$$-1.29269519$$

2.6.2 线性方程组与多元函数求解

在下一章线性代数部分，会有关于线性方程组的深入讨论。这里只简单地介绍 Maple 中线性方程组的基本解决方法，为帮助读者解决简单的多元函数问题提供参考。请看一个简单例子：

```
> eq1:=a+2*b+3*c=14;
```

$$eq1 := a + 2b + 3c = 14$$

```
> eq2:=a+3*b+4*c=19;
```

$$eq2 := a + 3b + 4c = 19$$

```
> eq3:=a+4*b+2*c=15;
```

$$eq3 := a + 4b + 2c = 15$$

```
> sol:=solve({eq1,eq2,eq3},{a,b,c});
```

$$sol := \{b = 2, c = 3, a = 1\}$$

```
> subs(sol,eq1);
```

$$14 = 14$$

```
> eval({eq1,eq2,eq3},sol);
```

$$\{15 = 15, 14 = 14, 27 = 27\}$$

```
> map(subs,[sol],{eq1,eq2,eq3});
```

$$[\{15 = 15, 14 = 14, 27 = 27\}]$$

这里我们先定义了三个方程："eq1，eq2，eq3"，利用 solve()函数，并指定好需要求解的变量，系统便自动返回解的集合。再利用三种不同的方法，将解代入原方程进行验证。读者可以根据自身的喜好选择验证方法。对于没有惟一解的方程组，Maple 会随即选择一个指定的变量作为参数并给出解的表达式，如下例所示：

```
> eq1:=a+2*b+3*c=4;
```

$$eq1 := a + 2b + 3c = 4$$

```
> eq2:=2*a+3*b+4*c=5;
```

$$eq2 := 2a + 3b + 4c = 5$$

```
> eq3:=3*a+4*b+5*c=6;
```

$$eq3 := 3a + 4b + 5c = 6$$

```
> sol:=solve{eq1, eq2, eq3},{a,b,c};
```

$$sol := \{a = -2 + c, c = c, b = 3 - 2c\}$$

```
> eval({eq1,eq2,eq3},sol);
```

$$\{4 = 4, 6 = 6, 5 = 5\}$$

对于多元函数的高次方程，用户同样可以无所顾忌地使用 solve()函数进行求解：

```
> a:='a':eq1:='eq1':eq1:=x^2*y^2=a;
```

$$eq1 := x^2 y^2 = a$$

```
> eq2:='eq2':eq2:=x=y;
```

$$eq2 := x = y$$

```
> solve(eq1,{x,y});
```

$$\{y = y, x = \frac{\sqrt{a}}{y}\}, \{y = y, x = -\frac{\sqrt{a}}{y}\}$$

```
> assume(a>0);
> solve(eq1,{x,y});
```

$$\{y = y, x = \frac{I\sqrt{-a\sim}}{y}\}, \{y = y, x = \frac{-I\sqrt{-a\sim}}{y}\}$$

在这个例子中，方程中加入了参数“a”。Maple 系统会在默认情况下将参数设定为大于零的实数，所以会得到如第三步所求出的解。利用“assume（）”函数，就可以对参数设定取值范围，获得用户所期望的结果。

再介绍一个函数“allvalues()”，请看下例：

```
> eq1:='eq1':eq1:=x^2+y^2=8;
```

$$eq1 := x^2 + y^2 = 8$$

```
> eq2:='eq2':eq2:=y=x^2;
```

$$eq2 := y = x^2$$

```
> solve({eq1,eq2},{x,y});
```

$$\{x = \text{RootOf}(-\text{RootOf}(\_Z + \_Z^2 - 8, label = \_L1) + \_Z^2, label = \_L2),$$
$$y = \text{RootOf}(\_Z + \_Z^2 - 8, label = \_L1)\}$$

```
>allvalues(%);
```

$$\{y = -\frac{1}{2} + \frac{1}{2}\sqrt{33}, x = \frac{1}{2}\sqrt{-2 + 2\sqrt{33}}\},$$
$$\{y = -\frac{1}{2} + \frac{1}{2}\sqrt{33}, x = -\frac{1}{2}\sqrt{-2 + 2\sqrt{33}}\},$$
$$\{y = -\frac{1}{2} - \frac{1}{2}\sqrt{33}, x = \frac{1}{2}\sqrt{-2 - 2\sqrt{33}}\},$$
$$\{y = -\frac{1}{2} - \frac{1}{2}\sqrt{33}, x = -\frac{1}{2}\sqrt{-2 - 2\sqrt{33}}\},$$

```
> fsolve({eq1,eq2},{x,y});
```

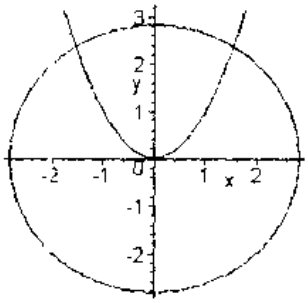
$$\{y = 2.372281323, x = -1.540221193\}$$

注意利用 allvalues()并不常会得到 solve()函数无法求解方程的解析解，而且读者会发现，相对于 fsolve()有可能返回不完全的方程组的解，allvalues 则有时会得到伪解。所以选择哪个函数进一步计算 solve()无法解决的方程需要一定技巧。同时，经常利用 eval()函数或图解法对方程进行验证也是 Maple 用户的良好习惯。

对多元函数，以前介绍的 plot 函数将无法画图。这时需要利用 plots 程序库中的函数“implicitplot()”，如检验上例中涉及的函数：

```
> plot({x^2+y^2=8,y=x^2},x=-Pi..Pi,y=-Pi..Pi);
plotting error,empty plot
```

```
> with(plots):implicitplot({x^2+y^2=8,y=
  x^2},x=-Pi..Pi,y=-Pi..Pi);
```



本书会在“Maple 的绘图功能”中详细介绍 plots 程序库中的函数。

在“student”程序包中，提供了类似 solve()的函数“intersect()”，它的标准形式为：

（1）intercept(eqn1)。

（2）intercept(eqn1, eqn2, {x, y})。

这里 eqn1，eqn2 分别代表所定义的方程，{x,y}代表方程中未知数的名称。此函数的功能是求两组方程曲线的交点。如果未指定第二个方程（如形式一），则系统会缺省定义方程二为“x=0”，如：

```
> intercept(y=x^2-8);
```

$$\{y=-8, x=0\}$$

```
> intercept(x^2+y^2=20,x=2*y);
```

$$\{x=4, y=2\},\{x=-4, y=-2\}$$

```
> intercept(x^3+y^2=20,y=x,{x,y});
```

$$\{x=\text{RootOf}(2\_Z^3-25, label=\_L7), y=\text{RootOf}(2\_Z^3-25, label=\_L7)\}$$

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

$$\{x=2.320794417, y=2.320794417\},$$
$$\{y=-1.160397208+2.009866922I, x=-1.160397208+2.009866922\,I\},$$
$$\{x=-1.160397208-2.009866922I, y=-1.160397208-2.009866922\,I\}$$

2.6.3　不等式的求解

同样利用 solve()函数，我们可以对不等式进行求解，Maple 会自动将解化简。对多变量的不等式组，只需指定所求变量即可。如：

```
> solve({x<5,x<6,3<x},x);
```

$$\{x<5, 3<x\}$$

```
> solve({-4<x,x<7,x>-10,5>x},x);
```

$$\{-4<x, x<5\}$$

```
> solve({x-y<5,x+y<4},{x,y});
```

$$\{x+y<4, x<\frac{9}{2}, x-y<5\}$$

```
> ineq:=x+y+4/(x+y)<10;
```

$$ineq := x+y+\frac{4}{x+y}<10$$

```
> solve(ineq,{x,y});
```

$$\{x<-y\}, \{5-\sqrt{21}-y<x, x<5+\sqrt{21}-y\}$$

注意最后一个不等式的解是以集合+序列的形式表示，序列是由集合组成的，每个集合之间是“或”的关系，而每个集合内的元素之间则是与的关系。如果用语言表示上述形式，为：“$x<-y$，或$5-\sqrt{21}-y<x<5+\sqrt{21}-y$”。

对于含绝对值的不等式，Maple 会默认使用“RealRange（实数范围内的解）”方式表示方程解的区间，如下例所示：

```
> imeq1:=abs(x^2)<8;ineq2:=abs(x^2)<=8;
```

$$ineq1 := |x|^2<8$$

$$ineq2 := |x|^2\le 8$$

```
> solve(ineq1,x);solve(ineq2,x);
```

RealRange(Open($-2\sqrt{2}$), Open($2\sqrt{2}$))

RealRange($-2\sqrt{2}$, $2\sqrt{2}$)

```
> solve(ineq2,{x});
```

$$\{-2\sqrt{2}\le x, x\le 2\sqrt{2}\}$$

最后，再举一个应用作图法求解的例子：

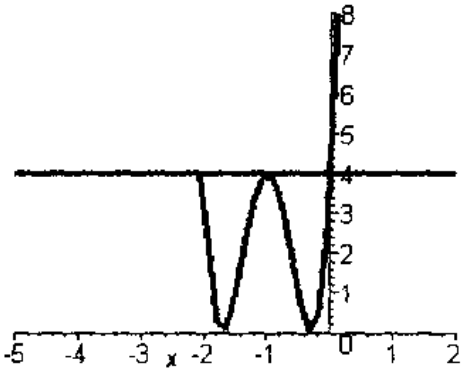
```
> eq:='eq':eq:=abs((x+abs(x+2))^2-2)^2=4;
```

$$eq := |(x+|x+2|)^2-2|^2=4$$

```
> solve(eq,{x});
```

$$\{x=0\}, \{x=-1\}, \{x\le -2\}$$

```
> plot({abs((x+abs(x+2))^2-2)^2,4},x=-5..2,
  0..8,color=[red,blue],thickness=3);
```



2.6.4　多项式求根

多项式求根的原理同求解方程的差别很小，相当于将多项式等于零后求此方程的解。MAPLE 中使用函数 roots()求解多项式的根，roots()以列表[n,m]的形式返回，n 相当于多项式的根，m 相当于此根的阶数，即是几重根。如：

```
> p:='p':p:=2*x^3+11*x^2+12*x-9;
```

$$p := 2x^3 + 11x^2 + 12x - 9$$

```
> roots(p,x);
```

$$\left[\left[\frac{1}{2},1\right],[-3,2]\right]$$

```
> factor(p);
```

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

```
> solve(p=0,x);
```

$$\frac{1}{2},-3,-3$$

使用 roots()的时候要注意，系统一般只处理可以在实数范围内因式分解至最简形式的多项式，例如如果输入“roots(x^4-4,x);”，系统不会返回任何值。必须在指定返回参数后才能得到正确结果，如：

```
> roots(x^4-4,x);
```

$$[\,]$$

```
> roots(x^4-4,sqrt(2));
```

$$[[\sqrt{2},1],[-\sqrt{2},1]]$$

```
> roots(x^4-4,{sqrt(2),I});
```

$$[[\sqrt{2},1],[-\sqrt{2},1],[-I\sqrt{2},1],[I\sqrt{2},1]]$$

另外，还需要指出的是，Maple 中还有另一条指令“root()”，它的作用是对表达式求 N 次方根，希望读者不要将其同 roots()混淆。例如：

```
> root[3](3.0);
```

$$1.442249570$$

```
> 3.0^(1/3);
```

$$1.442249570$$

```
> root(8,3);
```

$$2$$

```
> root(-2,2);
```

$$I\sqrt{2}$$

```
> root[2](x^2-2*x-3);
```

$$\sqrt{x^2-2x-3}$$

```
> root(x^2-2*x-3,2);
```

$$\sqrt{x^2-2x-3}$$

第 3 章　线性代数

线性代数，尤其是矩阵理论，是高等代数的核心部分。在高等量子力学、几何光学、运筹学、微分方程组的求解等应用领域中，线性代数都发挥着举足轻重的作用。从这一章开始，我们将开始利用 Maple 解决线性代数问题。除去系统的内部函数库外，Maple 还提供了两个程序包："linalg package" 和 "LinearAlgebra package"，它们分别提供了大量辅助函数，用于解决同线性代数相关的复杂问题。本章会以介绍 linalg 中的函数为主。

本章将依次介绍向量、数组、矩阵的基本概念，如何实现各种矩阵运算，以及如何利用矩阵理论重新求解线性方程组，最后，将介绍一些在实际问题中会碰到的线性代数问题并给出它们的解决办法。

3.1　向量运算

向量（vector）：既有大小又有方向的量。它是 "线性空间" 的基本组成概念，是研究有方向、位置关系的空间的重要工具。有关向量及线性空间的概念，请参考线性代数方面书籍。同样，在第 4 章 "微积分运算"、第 5 章 "微分方程"，本书都将假定读者了解相关的知识，不会涉及理论的介绍与公式推导。

3.1.1　向量的生成

在 Maple 中，用表示列表的 "[]" 符号标记向量。同时用户也可以将向量看做是一种特殊的列表，只是它的下标从 1 开始。因此有关列表的运算同样适用于向量间的运算。但通过一般过程定义的列表将不具备向量的性质，如：

```
> array(1..3),[1,2,3]);
```

$$[1,2,3]$$

```
> type(%,vector);
```

$$true$$

```
> type(%%,array);
```

$$true$$

```
> A:=array(0..3,[a,b,c,d]):
> type(A,vector);
```

$$false$$

```
> type(A,array);
```

$$true$$

利用“array(1..n,[])”，我们就可以定义一个 N 维的向量。利用 linalg 程序包中的“linalg[vector]”，或者直接使用系统自带的 vector 函数，也可以定义向量，如：

```
> vector([1,2,3]);
```

$$[1,2,3]$$

```
> linalg[vector](4,[1,x,x^2,x^3]);
```

$$[1, x^2, x^3]$$

```
> type(%,vector);
```

$$true$$

```
> type(%%,array);
```

$$true$$

可以看出向量（vector）这种数据类型在 Maple 系统中同列表（array）这种数据类型是完全对应的，但列表类型不完全是向量，即向量相当于列表的一个子集。

再看几个同向量相关的例子：

```
> v:='v':v:=vector(5,8);
```

$$v := [8,8,8,8,8]$$

```
> u:='u':u:=vector([4,5,6,7,8,9]);
```

$$u := [4,5,6,7,8,9]$$

```
> linalg[vectdim](v);
```

$$5$$

```
> linalg[vectdim](u);
```

$$6$$

```
> v;
```

$$v$$

```
> eval(v);
```

$$[8,8,8,8,8]$$

```
> print(v);
```

$$[8,8,8,8,8]$$

```
> u[4]:=0;
```

$$u_4 := 0$$

```
> print(u);
```

$$[4,5,6,0,8,9]$$

读者可以从头两条命令中看出 vector()函数的使用方法，如果未指定向量中的元素，而按照 vector（n,m）的形式输入，系统将自动产生一个 n 维向量，它的每个元素都是 m。使用 linalg 程序包中的“vectdim”函数，会得到所求向量的维数。由于直接输入“v;”系统不会返回向量的内容，查看向量值就需要使用“eval（）”函数或“print（）”函数。同 C 语言类似，Maple 中也有“printf（）”函数，同样可以利用%s、%a、%g、\n 等参数获得标准格式输出。相关问题会在有关 Maple -C 程序接口一章中介绍。引用、更改向量中的参数也很简单，如上例中倒数第二条命令所示。

Maple 还可以使用变量作为向量的元素，利用 MAP 函数可对其所有或部分元素进行操作，并利用两种作图函数获得不同显示方式的图形，如：

```
> u:='u':u:=vector([1,x^2,x^3,x^4]);
```

$$u := [1, x^2, x^3, x^4]$$

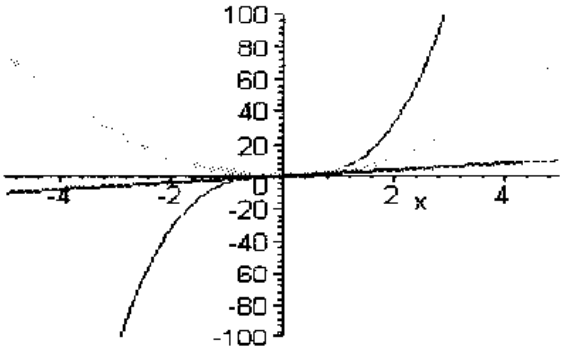
```
> map(diff,[u[1],u[2]],x)
```

$$[0, 2x]$$

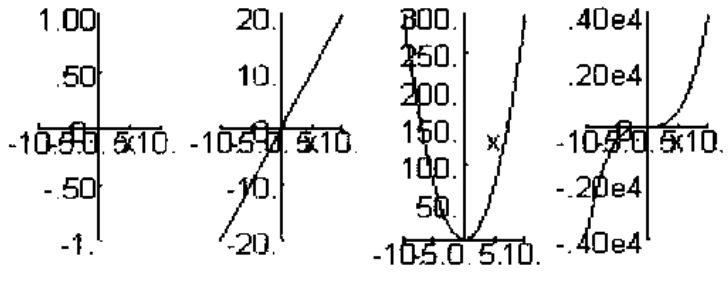
```
> w:='w':w:=map(diff,u,x);
```

$$w := [0, 2x, 3x^2, 4x^3]$$

```
> map(plot,w,x);
> plot(w,x=-5..5,-100.100,thickness=[1,2,4]);
```




```
> plots[display](map(plot,w,x=-10..10));
```



3.1.2　向量的运算

向量间的基本运算是线性运算，即两个或多个向量间的加、减运算。作为一种复合结

构的数据，用户不能直接对向量进行加法或减法，而需要调用函数“evalm()”(evaluate a matix expression)，如下例所示：

```
> v:='v':v:=vector([1,2,3]);
```

$$v := [1,2,3]$$

```
> u:='u':u:=vector([x,x^2,y]);
```

$$u := [x, x^2, y]$$

```
> w:='w':w:=vector([a,b,c,d]);
```

$$w := [a, b, c, d]$$

```
> z:'z':z:=v+u;
```

$$z := v + u$$

```
> eval(z);
```

$$v + u$$

```
> z:='z':z:=evalm(v+u);
```

$$z := [1 + x, 2 + x^2, 3 + y]$$

```
> evalm(5*u);
```

$$[5x, 5x^2, 5y]$$

```
> evalm(5*u);
Error, (in linalg[matadd]) vector dimensions incompatible
```

可以看出，维数不相同的向量之间是无法进行运算的。读者可能会注意，evalm 的英文名称(evaluate matrix)直译过来是“对矩阵求值”，因此这时 Maple 系统是将向量作为一个一维矩阵对待的。有关 evalm（）函数，我们还会在矩阵运算一节继续介绍。

evalm（）函数只能对向量完成线性运算（即加、减运算。上例中的 5*u 也相当于是对向量 u 的连加运算），向量间的“乘法”涉及两类：内积与外积，或者说点积与叉积。在 Maple 如何解决向量间的“乘法”之前，先需要向读者介绍如何利用 Maple 计算向量的模与夹角。

向量的模即向量的的长度，其数值等于构成向量的各个元素平方和的算术平方根，对三维向量 V（V1，V2，V3），有$|V|=(|V1|^2+|V2|^2+|V3|^2)^{1/2}$。在 Maple 中，可以使用 linalg 函数库中的“norm”函数计算向量的模，如：

```
> restart;
> with(linalg);
  Warning, the protected names norm and trace have been
  redefined and unprotected
> u:=vector([1,2,3]);
```

$$u := [1,2,3]$$

```
> norm(u,1);norm(u,2);
```

$$\sqrt{14}$$

```
> v:=vector([a,b,c]);
```

$$v := [a, b, c]$$

```
> norm(v);
```

$$\max(|c|,|b|,|a|)$$

```
> norm(v,3);
```

$$(|a|^3 + |b|^3 + |c|^3)^{\left(\frac{1}{3}\right)}$$

倒数第二步显示了默认参数时 norm 的返回值，即求向量中各分量绝对值的最大值。最后一步可以看出 norm 函数的一般计算公式。显然，以 norm(v,2)的形式输入就可以获得向量 *v* 的模。注意在载入 linalg 程序库的时候，会提示保留函数 norm 被重新定义了。Maple 中的保留函数 norm 的定义为：norm(a,n,v) = sum(abs(c)^n for c in [coeffs(a,v)])^(1/n)，即对一个表达式中变量的常数系数按照 linalg[norm]的公式计算结果。注意的是保留函数的 norm 只对多项式进行计算，而 linalg 程序包中的 norm 的对象则只是矩阵。

计算向量间的夹角相对简单，在 Maple 中可以使用 linalg 程序库中的 angle（）函数获得两个向量间夹角的反余弦值，如继续上例的计算：

```
> w:=vector([0,0,1]);
```

$$w := [0,0,1]$$

```
> angle(u,w);
```

$$\arccos\left(\frac{3}{14}\sqrt{14}\right)$$

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

$$.6405223126$$

```
> angle(v,w);
```

$$\arccos\left(\frac{c}{\sqrt{a^2+b^2+c^2}}\right)$$

两向量（*V1*，*V2*）间的内积（点积）定义为：

$$\overline{V1} \bullet \overline{V2} = |V1||V2|\cos(\theta)$$

其中 θ 是向量 V1、V2 的夹角。即两个向量的内积等于这两个向量的模以及向量夹角余弦的乘积，其结果是一个数值（因此内积还有“数量积”的叫法）。Maple 使用函数 *dotprod* 计算同维数向量间的内积。如下例，其中使用了 vector() 函数的默认表示方法：

“V=vector(n)”，即定义一个 n 维的任意向量(V_1,V_2,…Vn),V_1,…Vn 可以是任意数值：

```
> v1:=vector(3);
```

$$v1 := \mathrm{array}(1..3,[])$$

```
> v2:=vector(3);
```

$$v2 := \mathrm{array}(1..3,[])$$

```
> evalm(v1+v2);
```

$$[v1_1 + v2_1, v1_2 + v2_2, v1_3 + v2_3]$$

```
> v3:=vector([1,2,3]);
```

$$v3 := (1,2,3)$$

```
> with(linalg);
Warning, the protected names norm and trace have been
redefined and unprotected
```

```
> dotprod(v1,v2);
```

$$v1_1\overline{(v2_1)} + v1_2\overline{(v2_2)} + v1_3\overline{(v2_3)}$$

```
> dotprod(v1,v3);
```

$$v1_1 + 2v1_2 + 3v1_3$$

从这个例子中，可以发现其实计算向量的内积很简单，只是将两个向量中的对应分量相乘后再相加即可。读者应该能够自行从向量内积的定义中推导出此结果。内积的一个推论是：如果两个非零向量的内积为零，则这两个向量互相垂直。 利用 dotprod()函数可以简单的验证此结果，这里就不再举例了。

两向量（***V1***，***V2***）间的外积，或称叉积，其运算结果为另一向量，此向量的方向分别同两原始向量垂直，其大小（模）等于以 ***V1***，***V2*** 为边的平行四边形的面积，对应的计算公式为：

$$|\vec{V}1 \times \vec{V}2| = |V1||V2|\sin(\theta)$$

Maple 使用函数 crossprod（）计算叉积，如继续刚才的例子：

```
> crossprod(v1,v2);
```

$$[v1_1 v2_3 - v1_3 v2_2, v1_3 v2_1 - v1_1 v2_3, v1_1 v2_2 - v1_2 v2_1]$$

```
> crossprod(v1,v3);
```

$$[3v1_2 - 2v1_3, v1_3 - 3v1_1, 2v1_1 - v1_2]$$

从这里不难发现内积的计算公式。内积的计算利用行列式会更方便，有关的内容会在后边提到。读者需要注意的是内积只是在三维空间的一个定义，如果在 crossprod（）函数中出现其他维数的向量，Maple 会返回错误信息：

```
> v4:=vector(2);
```

$$v4 := \text{array}(1..2,[\ ])$$

```
> v5:=vector(4);
```

$$v5 := \text{array}(1..4,[\ ])$$

```
> crossprod(v4,v4);
Error, (in crossprod) invalid arguments
> crossprod(v4,v1);
Error, (in crossprod) invalid arguments
> crossprod(v5,v4);
Error, (in crossprod) invalid arguments
> crossprod(v1,v1);
```

$$[0,0,0]$$

在 linalg 程序库中，同向量相关的函数还有“normalize”：对向量进行归一化；以及“matadd”：类似 evalm（），可以对两个向量进行线性运算。使用方法如下：

```
> v1:=vector([1,2,3]);
```

$$v1 := [1,2,3]$$

```
> v2:=vector([2,3,4]);
```

$$v2 := [2,3,4]$$

```
> normalize(v1);
```

$$\left[\frac{1}{4}\sqrt{14},\frac{1}{7}\sqrt{14},\frac{3}{14}\sqrt{14}\right]$$

```
> matadd(v1,v2,2,-2);evalm(2*v1-2*v2);
```

$$[-2,-2,-2]$$

$$[-2,-2,-2]$$

由于 matadd 函数只能对两个向量进行运算，而且其表达方式并不直观，所以一般不建议用户使用。

3.1.3　向量的转化

本章的第一节简要介绍了向量在 Maple 中所对应的数据类型，从下例中，读者能系统的了解 Maple 是如何定义向量的数据类型的，以及如何利用 convert（）函数将向量转化为其他类型的函数：

```
> v:=vector([1,2,2,3,]);
```

$$v := [1,2,2,3]$$

```
> type(v,vector);
```

$$true$$

```
> type(v,array);
```

$$true$$

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

$$false$$

```
> whattype(a);
```

$$symbol$$

```
> convert(v,set);
```

$$\{1,2,3\}$$

本节的最后，对 LinearAlgebra 程序包的 Vector 函数作一简要介绍。Vector（）函数所定义的向量是 LinearAlgebra 程序包的基础元素，它对应一种名为“rtable”的数据结构，此种数据结构有其特有的储存方式，此处不做介绍。相应的还有 Matrix（）函数也对应这种结构。此程序包中的函数将只处理此种类型的数据（这些函数的一个共同特点是首字母大写，类似情况我们在介绍 sum 函数时也曾遇到）。Vector（）以类似矩阵中的行向量、列向量的方式定义向量，如：

```
> v:=Vector[row]([1,2,3,]);
```

$$V := [1,2,3]$$

```
> W:Vector([3,2,1]);
```

$$W := \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

```
> whattype(v);
```

$$Vector_{row}$$

用户可以在 Vector（）函数生成向量时添加多种限制条件，常用的有“Vector(n,expr)”和“Vector([],readonly=true)”（其他参数请参考联机帮助），同时，Vector（）类型的向量之间可以直接进行线性运算，如：

```
> F:= (j) -> x^(j-1):
  Vector[row](3,F);
```

$$[1, x, x^2]$$

```
> A:=Vecotor[row]([1,2,3],readonly=true);
>
```

$$A := [1,2,3]$$

```
> A[2]:=2;
Errow, cannot assign to a read-only Vector
```

```
> B:=Vecotor[row]([5,6,7]);
```

$$B := [5,6,7]$$

```
> C:=A+B;
```

$$C := [6,8,10]$$

```
> whattype(%%);
```

$$Vector_{row}$$

利用 Vector（）函数定义的向量不能在 linalg 程序库的函数中调用，反之亦然。而利用 convert 函数，则可以使向量或矩阵在两种类型间转换，如继续上例：

```
> c:=convert(C,vector);
```

$$c := [6,8,10]$$

```
> type(c,vector);
```

$$true$$

```
> type(c,rtable);
```

$$false$$

```
> convert(c,vector);
```

$$[6,8,10]$$

```
> type(%,rtable);
```

$$true$$

```
> whattype(%%);
```

$$Vector_{row}$$

总体上说，LinearAlgebra 函数库具备 linalg 函数库的所有功能，同时也具有更规范、更完善的数据存储格式，以及更准确、更迅速的数值计算能力。相对来说，linalg 函数库更适合于抽象的线性代数问题的推导工作，而 LinearAlgebra 函数库则更适合于对数值型矩阵的处理与运算。

3.2 矩阵的生成与修改

从这一节开始，我们将开始进行对线性代数的核心部分——矩阵的学习。矩阵的运算是线性代数的基本内容。虽然矩阵（matrix）这个单词是 1850 年由英国数学家 Sylvester（西尔维斯特）首先提出的，但不谦虚地说，第一次利用它解决线性方程组问题的依然是中国人（在最早的东汉初年的《九章算术》，以及 1303 年朱世杰的著作中，都曾详细介绍过类似的方法）。矩阵的出现将一系列杂乱无章的数据用数表联系成一个整体，并统一参与运算，推动了现代数学甚至计算机理论的发展，具有不可估量的作用。

本书的第 1 章介绍 Maple 的数据结构时曾经提到，Maple 在缺省方式下，以数组的形式储存矩阵（在一些特殊程序库中，还有其他的储存方式，此处不做讨论），它们以顺序的方式在内存中存储（类似 C 语言中数组的储存方式）。下面，我们从建立一个矩阵开始 Maple 的线性代数解决策略。

3.2.1 创建简单矩阵的几种方法

在第 1 章介绍 Maple 数据类型一节中，我们说过用户在利用“array（）”函数建立一个数组的过程实际上也是建立矩阵的过程。这里我们先简要复习一下：

```
> restart;
```

```
> A:=array(1..3,1..3):print(A);
```

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}$$

```
> B:=array(1..3,1..3,[[1,2,3],[],[]]);;
```

$$B := \begin{bmatrix} 1 & 2 & 3 \\ B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,1} & B_{3,2} & B_{3,3} \end{bmatrix}$$

```
> C:=array([[1,2,3],[2,3,4]]);
```

$$C := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

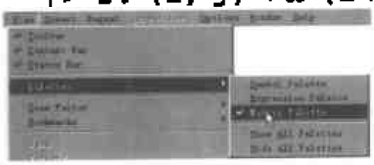
在此例中，我们使用了 array 的三种形式定义矩阵。读者应该能清楚地分辨每种形式的作用。需要提醒读者注意的是，如果在定义矩阵的同时为其元素赋值，一定要用“[]”将元素括起来，因为 Maple 是将此参数按向量集合的形式传递给为数组开辟的内存，错误的形式将导致参数传递错误。

在 linalg 程序库中，有一个函数 matrix（）专门负责矩阵的生成，其用法同 array（）函数类似，只是在定义矩阵的维数时不需要以“1..n”的形式了，如：

```
> with(linalg):
> matrix(3, 3, [[], [1, %?, 3], [1, 2, 3]]);
```

$$\begin{bmatrix} ?1,1 & ?1,2 & ?1,3 \\ 1 & ? & 3 \\ 1 & 2 & 3 \end{bmatrix}$$

```
> f:=(i, j)->x^(i+j-1):
```



```
f);
```

$$A := \begin{bmatrix} x & x^2 \\ x^2 & x^3 \end{bmatrix}$$

第一条  、“linalg”程序库（以“:”结尾可以省去大量的无用信息显示）。第二条命令创建×3 矩阵，其三个行向量分别以三种形式输入，希望读者能总结出区别。第三条命令使用了一个技巧，即 Maple 在创建矩阵的时候，会自动以“i，j”为坐标变量标记矩阵元素。所以我们创建一个以“i，j”为变量的函数，就可以被系统自动按照位置赋值而获得所期望的矩阵形式。希望读者能灵活掌握此种为矩阵赋值的方法，这样会给工作带来一定的便利。

Maple 中创建矩阵的第三种形式是利用系统工具栏。用户可以从主菜单中的 View->Palletes->Matrix Palletes 选择显示矩阵的默认工具栏，如图 3-1 所示：

图 3-1　选择 Matrix Palette 工具栏

选中后，会出现图 3-2 所示面板：

图 3-2　Matrix Palette 工具栏

用户点击面板上的图标，就会产生对应维数的矩阵。例如点击  图标，Maple 就会在工作簿中显示：

```
> Matrix([[%?, %?, %?], [%?, %?, %?]]);
```

用户就可以在高亮的地方开始输入数组的第一个元素。例如按如下输入，回车后系统会自动生成一个 2×3 的矩阵：

```
> Matrix([[1, %?, 3], [%?, 2, %?]]);
```

$$\begin{bmatrix} 1 & ? & 3 \\ ? & 2 & ? \end{bmatrix}$$

另外一种常用的矩阵生成方法是利用简单的程序，使用“for”循环生成矩阵的元素，有关 Maple 的中的编程问题有单独的章节介绍，此处只简单的列出程序清单，读者可自行研究：

```
> restart;
> M:=matrix(3,3);
```

$$M := \text{array}(1..3,1..3,[\,])$$

```
> for i to 3 do
>   for j to 3 do
>   M[i,j]:=x^i+j*x
>   od
> od;
> print(M);
```

$$\begin{bmatrix} 2x & 3x & 4x \\ x^2+x & x^2+2x & x^2+3x \\ x^3+x & x^3+2x & x^3+3x \end{bmatrix}$$

3.2.2 几种特殊矩阵的创建方法

为方便用户建立矩阵，Maple 系统支持几种特殊矩阵的简易输入，主要是利用为“array()”函数添加不同参数而实现的。在这里我们为读者列出几种简单特殊矩阵的创建方法，希望能有助于读者的工作。

Maple 支持的可以简化输入的特殊矩阵有：“symmetric：对称矩阵”；“antisymmetric 非对称矩阵”；“identity：单位矩阵”；“sparse：零矩阵”和“diagonal：对角矩阵”。它们的创建方法如下所示：

```
/**********symmetry**********/
> A:=array(symmetric,1..3,1..3);
```

$$A := \mathrm{array}(symmetric, 1..3, 1..3, [\,])$$

```
> print(A);
```

$$\begin{bmatrix} A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,1} & A_{3,2} & A_{3,3} \end{bmatrix}$$

```
> A[1,2]:=10;A[3,1]:=8;
```

$$A_{1,2} := 10$$

$$A_{3,1} := 8$$

```
> print(A);
```

$$\begin{bmatrix} A_{1,1} & 10 & 8 \\ 10 & A_{2,2} & A_{2,3} \\ 8 & A_{3,2} & A_{3,3} \end{bmatrix}$$

被定义成“symmetric”后，在给矩阵赋值的时候就会如上例所示，对角元素会同时被赋值，确保用户不会错误地输入矩阵元素。

非对称矩阵的生成方式同对称矩阵类似，为：

```
/**********antisymmetry**********/
> B:=array(antisymmetric,1..3,1..3);
```

$$B := \mathrm{array}(antisymmetric, 1..3, 1..3, [\,])$$

```
> print(B);
```

$$\begin{bmatrix} 0 & B_{1,2} & B_{1,3} \\ -B_{1,2} & 0 & B_{2,3} \\ -B_{1,3} & -B_{2,3} & 0 \end{bmatrix}$$

单位矩阵和零矩阵的创建相对容易，如下所示：

```
/**********sparse**********/
> C:=array(sparse,1..3,1..3);
```

$$C := \mathrm{array}(sparse, 1..3, 1..3, [\,])$$

```
> print(C);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
/**********identity**********/
```

```
> J:=array(identity,1..3,1..3);
```

$$J := \text{array}(identity, 1..3, 1..3, [\,])$$

```
> print(J);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

利用类似的方法，用户也可以建立对角矩阵：

```
/**********diagonal**********/
```

```
> Dia:=array(diagonal, 1..3, 1..3);
```

$$Dia := \text{array}(diagonal, 1..3, 1..3, [\,])$$

```
> Dia[2,2]:=4
```

$$Dia_{2,2} := 4$$

```
> prit(Dia);
```

$$\begin{bmatrix} Dia_{1,1} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & Dia_{3,3} \end{bmatrix}$$

3.2.3 访问矩阵元

在创建矩阵之后，下一步我们来介绍如何修改、提取矩阵中的元素。上一节的例子中，实际已经对矩阵元素进行过赋值。同数组一样，对矩阵元素的显示、修改是以“矩阵名+[i, j]”的形式完成的。其中 i，j 分别代表矩阵元素的行坐标与列坐标。简单的赋值方法此处就不再做介绍。下面我们继续利用第一节最后所创建的变量矩阵 M，通过 for 循环以相同规则改变矩阵中的所有元素，并生成新的矩阵 N。作为基本的编程方法，for（）函数的原理此处就不作说明：

```
> print(M);
```

$$\begin{bmatrix} 2x & 3x & 4x \\ x^2+x & x^2+2x & x^2+3x \\ x^3+x & x^3+2x & x^3+3x \end{bmatrix}$$

```
> N:=matrix(3,3);
```

$$N := \text{array}(1..3, 1..3, [\,])$$

```
> for i to 3 do
>   for j to 3 do
>    N[i,j]:=diff(M[i,j],x)
>   od
> od;
```

```
> print(N);
```

$$\begin{bmatrix} 2 & 3 & 4 \\ 2x+x & 2x+2 & 2x+3 \\ 3x^2+x & 3x^2+2 & 3x^2+3 \end{bmatrix}$$

前面的章节中曾介绍过函数“map（）”，它的作用也是将某种运算规则统一作用于某一表达式。利用 map（）函数，我们可以更方便地完成上例所实现的功能：

```
> map(diff,M,x);
```

$$\begin{bmatrix} 2 & 3 & 4 \\ 2x+x & 2x+2 & 2x+3 \\ 3x^2+x & 3x^2+2 & 3x^2+3 \end{bmatrix}$$

在第 2 章的第 5 节，我们曾使用过 map 函数，它的完全形式为：

MAP（函数 F，作用对象，函数 F 对应的参数）

MAP 的实际作用就相当于作用对象中的每个元素都使用同一函数作用。读者可阅读帮助文件获得有关 MAP 函数的更详细说明。

除了修改矩阵中的单独元素外，在 Maple 的 linalg 程序库中，还提供了可以对矩阵的行、列向量进行提取的函数：row（）和 col（）。它们的用法如下例所示：

```
> T:=matrix(3,3,[[1,1,1],[2,2,2],[3,3,3]]);
```

$$T := \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

```
> with(linalg):
Warning, the protected names norm and trace have been
redefined and unprotected
```

```
> row(T,2);
```

$$[2,2,2]$$

```
> type(row)(T,1),vector);
```

$$true$$

```
> col(T,2);col(T,2):=vector[col]([2,4,6]);
```

$$[1,2,2]$$

$$\mathrm{col}(T,2) := [2,4,6]$$

```
> print(T);
```

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

```
> col(T,2);
```

$$[2,4,6]$$

如上例所示，row（）与 col（）函数只能对矩阵的行、列向量进行读取，无法直接为整行、整列的向量重新赋值。强行赋值只会造成函数的错误输出，请读者注意。

Linalg 程序库还类似的提供提取子矩阵的函数 submatrix()，在分别指定所需要提取矩阵的行、列范围后，它能自动生成新的矩阵。如：

```
> submatrix(T,1..2,1..3);
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}$$

```
> submatrix(T,1..1,1..3);
```

$$[1 \quad 1 \quad 1]$$

```
> submatrix(T,2..3,1..3);
```

$$\begin{bmatrix} 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix}$$

注意，即使仅提取一行元素，也需要以“n..n”的形式书写，而不能直接写成“n”。

3.2.4 矩阵的初等变换与修改

在本节的开始，我们曾介绍过矩阵的起源是为了方便线性方程组的求解过程。最基本的求解方法是消元法，它利用对多元方程进行初等变换而获得未知数的独立解。类似的，我们也定义了矩阵的初等变换，包括：

（1）用一个非零数乘矩阵的某一行。

（2）将一行的 k 倍加到另一行（k 为非零实数）。

（3）交换矩阵中两行的位置。

Maple 的 linalg 程序库中，分别提供了三个函数：“mulrow（）”、“addrow（）”和“swaprow（）”来完成上述的三种初等变换。与对行向量的处理相对应的，还有三个函数可以对列向量完成同样的变换：“mulcol（）”、“addcol（）”和“swapcol（）”。下面将以两两一组的方式介绍这六个函数的适用方法，全部变换都在矩阵 M 的基础上完成，M 的定义为：

```
> M:=matrix(3,3,[1,2,3,4,5,6,7,8,9]);
```

$$M := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

1. 非零数乘矩阵的某一行或列："mulrow()"，"mulcol()"

此组函数的参数输入顺序为：矩阵名，目的变换行（列），所乘系数。

```
> mulrow(M,1,3);
```

$$\begin{bmatrix} 3 & 6 & 9 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> mulrow(M,2,x);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4x & 5x & 6x \\ 7 & 8 & 9 \end{bmatrix}$$

```
> mulrol(M,3,v);
```

$$\begin{bmatrix} 1 & 2 & 3v \\ 4 & 5 & 6v \\ 7 & 8 & 9v \end{bmatrix}$$

此组函数的参数输入顺序为：矩阵名，目的变换行（列），所乘系数。

2. 将一行（列）的 k 倍加到另一行（列）："addrow()"，"addcol()"

此组函数参数输入顺序为：矩阵名，原始行（列），目的行（列），所乘倍数。
利用这两个函数分别完成对矩阵的上三角变换与下三角变换。先变换为上三角矩阵：

```
> addrow(M,1,2,-4);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> addrow(%,1,3,-7);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -12 \end{bmatrix}$$

```
> addrow(%,2,3,-2);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 0 \end{bmatrix}$$

再变换为下三角矩阵：

```
> addrow(M,1,3,-3);
```

$$\begin{bmatrix} 1 & 2 & 0 \\ 4 & 5 & -6 \\ 7 & 8 & -12 \end{bmatrix}$$

```
> addrow(%,1,2,-2);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 4 & -3 & -6 \\ 7 & -6 & -12 \end{bmatrix}$$

```
> addcol(%,2,3,-2);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 4 & -3 & 0 \\ 7 & -6 & 0 \end{bmatrix}$$

读者可以看出，不管是变换为上三角或是下三角，矩阵对角线上的元素是保持不变的，当然它们的积也是相等的。有一条定理为：矩阵上三角化或下三角化后，其对角线上元素的乘积等于矩阵的行列式的值。这里的 M 矩阵的行列式等于零，不易检验这条定理，读者可以自行设计简单矩阵来验证它。

3. 交换两行或两列："swaprow()"，"swapcol()"

函数参数输入顺序为：矩阵名，交换行（列）序号 1，序号 2：

```
> swaprow(M,1,2);
```

$$\begin{bmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> swapcol(M,1,2);
```

$$\begin{bmatrix} 2 & 1 & 3 \\ 5 & 4 & 6 \\ 8 & 7 & 9 \end{bmatrix}$$

读者需要注意的是以上的几种变换，均不会改变原始矩阵。如果需要对矩阵进行连续变换，则需要以“%”作为矩阵名进行后续步骤（如第二组命令）。此方法的缺点是如果某一步出现错误，修正错误会打乱原始运算顺序，用户须时刻注意“%”的引用对象的变化。当然也可以为中间过程的矩阵定义新的矩阵名，代价是相对复杂的输入过程以及可能会造成的重复定义。用户可以根据实际问题的需要以及自己的习惯选择不同的输入、运算过程。

初等变换的特点是不影响矩阵对应行列式的大小，也不会改变矩阵的维数。如果用户想改变矩阵的结构，linalg 程序库还提供了其他函数辅助用户的工作。相关的函数包括：“extend()：扩展矩阵”，“delrow()、delcol（）：删除行、列”，“augment/concat（）：将两个矩阵横向连接”，“stackmatrix（）：将两个矩阵纵向连接”以及“copyinto（）：将一个矩阵拷贝进另一个矩阵”。以下将分别介绍这几个函数。几组例子均是按顺序生成的。

（1）extend(A,m,n,x)：矩阵 A 扩充 m 行，n 列，并在扩充位置填入元素 x：

```
> A:=matrix(3,3,[1,2,3,2,4,7,3,7,14]);
```

$$A:=\begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 7\\ 3 & 7 & 14\end{bmatrix}$$

```
> with(linalg):
Warning, the protected names norm and trace have
 been redefined and unprotected
```

```
> extend(A,1,3,x);
```

$$\begin{bmatrix}1 & 2 & 3 & x & x & x\\ 2 & 4 & 7 & x & x & x\\ 3 & 7 & 14 & x & x & x\\ x & x & x & x & x & x\end{bmatrix}$$

（2）delrow(A，r..s)、delcol（A,r..s）:删除矩阵 A 中的 r 到 s 行（列）：

```
> delrows(%,4..4);
```

$$\begin{bmatrix}1 & 2 & 3 & x & x & x\\ 2 & 4 & 7 & x & x & x\\ 3 & 7 & 14 & x & x & x\end{bmatrix}$$

```
> delcols(%,4..6);
```

$$\begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 7\\ 3 & 7 & 14\end{bmatrix}$$

注意这两个函数中的第二个参数必须是一组范围结构的数据，即使只想删除一行，也需要以类似上例中“delcols（%，5..5）”的形式输入。

（3）augment/concat(A,B,…)：将矩阵 B 及其后的矩阵或向量按顺序连接在矩阵 A 的右方：

```
> B:=matrix(3,2,[4,5,11,16,25,41]);
```

$$B := \begin{bmatrix} 4 & 5 \\ 11 & 16 \\ 25 & 41 \end{bmatrix}$$

```
> augment(A,B);
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 7 & 11 & 16 \\ 3 & 7 & 14 & 25 & 41 \end{bmatrix}$$

```
> v:=vector(3,[6,22,63]);
```

$$V := [6,22,63]$$

```
> C:=concat(%%,V);
```

$$C := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 7 & 11 & 16 & 22 \\ 3 & 7 & 14 & 25 & 41 & 63 \end{bmatrix}$$

用户一定要注意，在使用 augment/concat()函数，以及下面介绍的 stackmatrix()函数时，预先一定要确定所连接的矩阵、向量的横向或纵向维数相同，否则，系统会给出如下出错提示：

```
> E:=matrix(2,6,[4,11,25,50,91,154,5,16
  41,91,182,336]);
```

$$E := \begin{bmatrix} 4 & 11 & 25 & 50 & 91 & 154 \\ 5 & 16 & 41 & 91 & 182 & 336 \end{bmatrix}$$

```
> concat(C,E);
Error, (in concat) incompatible matrix/vector
 dimensions
```

（4）Stackmatrix(A,B,…):将矩阵 B 及其后矩阵或向量按顺序连接到矩阵 A 下方：

```
> F:=stackmatrix(C,E);
```

$$F := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 7 & 11 & 16 & 22 \\ 3 & 7 & 14 & 25 & 41 & 63 \\ 4 & 11 & 25 & 50 & 91 & 154 \\ 5 & 16 & 41 & 91 & 182 & 336 \end{bmatrix}$$

（5）copyinto(B,A,m,n):将 B 矩阵由 A 矩阵的第 m 行，第 n 列起拷贝进 A 矩阵：

```
> G:=array(sparse,1..3,1..3);print(G);
```

$$G := \text{array}(sparse, 1..3, 1..3, [\,])$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> copyinto(F,G,1,1,);
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 7 \\ 3 & 7 & 14 \end{bmatrix}$$

```
> copyinto(G,F,4,4);
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 7 & 11 & 16 & 22 \\ 3 & 7 & 14 & 25 & 41 & 63 \\ 4 & 11 & 25 & 0 & 0 & 0 \\ 5 & 16 & 41 & 0 & 0 & 0 \end{bmatrix}$$

从此例读者可以看出，使用 copyinto()函数时，不会破坏原始矩阵的维数。如果被拷贝矩阵超出了拷贝范围，系统将自动截去出界部分。

以上我们主要以 linalg 程序库作为介绍对象。作为 Maple 自带的另一个处理线性代数问题的 LinearAlgebra 程序库，虽然采取不同的数据结构，但同样包含了可以完成类似功能的函数。我们利用一个例子介绍其中的几个简单函数，读者可以通过阅读联机帮助来获得进一步的信息。

```
> with(LinearAlgebra):
```

```
> A:=<<1,2,3,>|<4,5,6>|<7,8,9>|<10,11,12>>;
```

$$A := \begin{bmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{bmatrix}$$

```
> B:=DeleteColumn(A,4);
```

$$B := \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
> ColumnOperation(B,[1,3]);
```

$$\begin{bmatrix} 7 & 4 & 1 \\ 8 & 5 & 2 \\ 9 & 6 & 3 \end{bmatrix}$$

```
> ColumnOperation(B,1,3);
```

$$\begin{bmatrix} 3 & 4 & 7 \\ 6 & 5 & 8 \\ 9 & 6 & 9 \end{bmatrix}$$

此例中，第一条命令建立了一个以 rtable 为数据格式的 3×4 矩阵 A，并为其各个元素赋值；第二条命令删除了矩阵 A 的第四列，并将其命名为矩阵 B；第三条命令，将矩阵 B 的第三列同第一列交换位置，第四条命令，将矩阵 B 的第一列乘三。可以看出，ColumnOperation()函数能完成 linalg 函数库中的 addcol()与 swampcol()两个函数的功能。同它类似的，还有函数 RowOperation()列上。同时，将矩阵 B 替换为新生成的矩阵，即为矩阵 B 重新赋值。

如果为 ColumnOperation 函数或 RowOperation 函数添加参数“inplace=true”，则可以在生成新矩阵的同时，将它赋值给旧矩阵名。如：

```
> RowOperation(B,[2,3],2,inplace=true):print(B);
```

$$\begin{bmatrix} 1 & 4 & 7 \\ 8 & 17 & 26 \\ 3 & 6 & 9 \end{bmatrix}$$

3.3 矩阵的运算及其特征值求解

仿照第 2 章的顺序，同时也是数学研究的特点，在了解了一个对象自身的基本特点后，研究方向将集中在对象之间的运算关系上。在本节，我们将首先介绍矩阵间四则运算的实

现方法，再一步步的研究一个矩阵对应行列式的值、它的相关矩阵的求法，以及矩阵对角化的实现。阅读完这一节，读者将可以利用 Maple 完成所有同矩阵相关的基本变换。同时也可以了解 Maple 处理线性代数问题的特点。

3.3.1　矩阵间的基本运算

矩阵间的基本运算包括矩阵相加、矩阵相减、矩阵数乘、矩阵的乘方及矩阵相乘。Maple 系统利用内部函数“evalm（）”实现这些基本的矩阵运算，如下例所示：

```
> restart;
> A:=matrix(2,2,[1,2,3,4]);
```

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

```
> B:=matrix(2,2,[1,2,2,4]);
```

$$B := \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

```
> evalm(A+B);
```

$$\begin{bmatrix} 2 & 4 \\ 5 & 8 \end{bmatrix}$$

```
> evalm(2*A);
```

$$\begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

```
> evalm(A^2);
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

```
> evalm(A&*B);
```

$$\begin{bmatrix} 5 & 10 \\ 11 & 22 \end{bmatrix}$$

```
> evalm(B&*A);
```

$$\begin{bmatrix} 7 & 10 \\ 14 & 20 \end{bmatrix}$$

可以发现矩阵同数值的运算基本上没有任何区别，只是在进行矩阵间的乘法时，需要以“&*”作为运算符，否则 Maple 将提示系统错误。在进行矩阵运算时，需要注意矩阵的维度是否满足如下要求：

（1）两矩阵相加，其维度必须严格相等。

（2）两矩阵 A，B 相乘，即 A*B 时，A 矩阵的列向量数必须等于 B 矩阵的行向量数。

（3）只能对方阵（行向量数等于列向量数）进行乘方运算。

如果用户此时已经习惯了使用 linalg 程序库，可以依旧使用它所提供的函数完成上述功能。相关的函数为："matadd：矩阵间加减"，"scalarmul：矩阵数乘"，以及"multiply：矩阵相乘"。依旧使用上例中创造的矩阵，举例如下：

```
> with(linalg):
Warning, the protected names norm and trace
have been redefined and unprotected
```

（1）matadd(A,B,m,n)/matadd(m*A,n*B)：输出 m*A+n*B 的结果。m,n 一般为实数：

```
> matadd(2*A,-3*B);
```

$$\begin{bmatrix} -1 & -2 \\ 0 & -4 \end{bmatrix}$$

```
> matadd(A,B,2,-3);
```

$$\begin{bmatrix} -1 & -2 \\ 0 & -4 \end{bmatrix}$$

（2）scalarmul(A,m)：输出 m*A 的结果。m 非矩阵：

```
> scalarmul(A,-3);
```

$$\begin{bmatrix} -3 & -6 \\ -9 & -12 \end{bmatrix}$$

```
> scalarmul(A,3*x);
```

$$\begin{bmatrix} 3x & 6x \\ 9x & 12x \end{bmatrix}$$

（3）multiply(A,B,...)：按顺序计算 A*B*...的结果：

```
> multiply(A,B);
```

$$\begin{bmatrix} 5 & 10 \\ 11 & 22 \end{bmatrix}$$

```
> multiply(B,A);
```

$$\begin{bmatrix} 7 & 10 \\ 14 & 20 \end{bmatrix}$$

由于矩阵间相乘有顺序问题，相反的顺序会造成不同的结果。所以用户在使用 multiply 函数时，务必注意矩阵的输入顺序。

multiply 不仅可以计算矩阵间的乘法，如果满足运算条件，它还可以计算“矩阵*向量”或“向量*矩阵”，如：

```
> v:=vector([5,6]);
```

$$V := [5,6]$$

```
> multiply(A,V);
```

$$[17,39$$

```
> multiply(V,A);
```

$$[23,34$$

在计算矩阵与向量的乘法时，系统会自动按照向量在算式中的位置将其在行向量与列向量间进行转换，无须用户自己指定。

3.3.2　行列式的求解

行列式也是一个重要的数学工具，它利用简单的排列化简了一组数间的运算关系，同时利用其自身的性质，又可以化简计算过程。行列式也是计算矩阵的逆、特征值的基础。在 linalg 程序库中，使用函数“det（）”计算矩阵的行列式。注意只能对方阵计算行列式。如：

```
> A:=matrix(3,3,[1,2,3,2,4,7,3,7,14]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 7 \\ 3 & 7 & 14 \end{bmatrix}$$

```
> det(A);
```

$$-1$$

```
> B:=matrix(3,3,[1,0,0,0,4,0,0,0,10]);
```

$$B := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

```
> det(B);
```

$$40$$

在高能物理或一些其他领域中，还会遇到求高阶（384~1024 阶）矩阵的逆的情况。这里就涉及到求高阶矩阵的行列式问题。如果依然靠行列式的展开公式计算，会遇到天文数字般的计算量。此时，Maple 会自动在高斯消元法（Gaussian elimination）或最小二乘展开（minor expansion）间选择计算方法，或一起使用。

3.3.3 矩阵的转置

设矩阵 A 为 m×n 阶矩阵，将 A 的行、列互换而得的 n×m 阶矩阵 B 称为 A 的转置，记为 A^T。在 Maple 中，用函数“transpose（）”实现对矩阵的转置，如：

```
> A:=matrix(3,2,[1,-1,0,4,4,2]);
```

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

```
> At:=transpose(A);
```

$$At := \begin{bmatrix} 1 & 0 & 4 \\ -1 & 4 & 2 \end{bmatrix}$$

```
> multiply(A,At);
```

$$\begin{bmatrix} 2 & -4 & 2 \\ -4 & 16 & 8 \\ 2 & 8 & 20 \end{bmatrix}$$

```
> multiply(At,A);
```

$$\begin{bmatrix} 17 & 7 \\ 7 & 21 \end{bmatrix}$$

从这里也可以观察出矩阵相乘的顺序对最终结果的影响。我们再利用一个例子验证转置的一个性质：$(A*B)^T=A^T*B^T$：

```
> B:=matrix(2,3,[1,1,2,2,3,3]);
```

$$B := \begin{bmatrix} 1 & 1 & 2 \\ 2 & 3 & 3 \end{bmatrix}$$

```
> transpose(multiply(A,B));
```

$$\begin{bmatrix} -1 & 8 & 8 \\ -2 & 12 & 10 \\ -1 & 12 & 14 \end{bmatrix}$$

```
> multiply(transpose(B),transpose(A));
```

$$\begin{bmatrix} -1 & 8 & 8 \\ -2 & 12 & 10 \\ -1 & 12 & 14 \end{bmatrix}$$

3.3.4　矩阵的秩

在介绍矩阵的秩之前，先介绍向量的秩。一组向量中的最大线性无关向量的数目称为此组向量的秩。在线性代数中，一般是使用高斯消元法化简多个向量组成的矩阵，观察结果而得出最大线性无关数。在 Maple 中，利用“basis”函数就可以简单的判断一组向量中的最大线性无关组。如：

```
> v1:=vector([1,-1,2,4]);
```

$$V1 := [1,-1,2,4]$$

```
> v2:=vector([0,3,1,2]);
```

$$V2 := [0,3,1,2]$$

```
> v3:=vector([3,0,7,14]);
```

$$V3 := [3,0,7,14]$$

```
> v4:=vector([1,-1,2,0]);
```

$$V4 := [1,-1,2,0]$$

```
> v5:=vector([2,1,5,0]);
```

$$V5 := [2,1,5,0]$$

```
> basis([v1,v2,v3,v4,v5]);
```

$$[V1,V2,V4]$$

利用“rowspace”，“colspace”两个参数，basis 函数还可以判断一个矩阵中行向量或列向量中的最大线性无关组，如：

```
> M:=matrix(4,5,[6,4,1,-1,2,1,0,2,3,-4,1
  4,-9,-16,22,7,1,0,-1,3]);
```

$$M := \begin{bmatrix} 6 & 4 & 1 & -1 & 2 \\ 1 & 0 & 2 & 3 & -4 \\ 1 & 4 & -9 & -16 & 22 \\ 7 & 1 & 0 & -1 & 3 \end{bmatrix}$$

```
> basis(M,'rowspace');
```

$$[[6,4,1,-1,2],[1,0,2,3,-4],[7,1,0,-1,3]]$$

```
> basis(M,'colspace');
```

$$[[6,1,1,7],[4,0,4,1],[1,2,-9,0]]$$

类似的，利用函数“rowspace()”及“colspace()”，可以获得矩阵行或列向量构成空间的基础解系。利用 basis 函数获得的基础解系，是由矩阵的列向量构成的，而使用 rowspace

或 colspace 函数获得的基解系，则是规范型向量，且都以“1”为首元素。如继续上例：

```
> rowspace(M);
```

$$\left\{\left[0,0,1,\frac{23}{15},\frac{-98}{45}\right],\left[0,1,0,\frac{-8}{15},\frac{23}{45}\right],\left[1,0,0,\frac{-1}{15},\frac{16}{45}\right]\right\}$$

```
> colspace(M);
```

$$\{[0,0,0,1],[0,1,-5,0],[1,0,1,0]\}$$

本书涉及有关线性空间的概念，如果读者想了解相关概念，请参看有关书籍。

在矩阵 A 中任取 k 行、k 列而组成的 k 阶方阵称为矩阵 A 的一个 k 阶子块（submatrix），矩阵 A 中非零子块的最高阶数称为矩阵的秩（rank）。利用矩阵的秩，可以从理论上分析出线性方程组有解的条件，以及如何表达有无穷多解的线性方程组解的结构。有关线性方程组的问题，会在 3.4 节单独讨论，这里先简单介绍如何利用 Maple 求出一个已知矩阵的秩。

在 Maple 中，利用函数 rank（A），可以方便的求出矩阵 A 的秩，如：

```
> N:=matrix(5,3,[2,3,1,1,-2,4,-1,1,-3,1,
  -3,5,1,4,-2]);
```

$$N := \begin{bmatrix} 2 & 3 & 1 \\ 1 & -2 & 4 \\ -1 & 1 & -3 \\ 1 & -3 & 5 \\ 1 & 4 & -2 \end{bmatrix}$$

```
> rank(N);
```

$$2$$

```
> basis(N,'colspace');
```

$$[[2,1,-1,1,1],[3,-2,1,-3,4]]$$

```
> colspace(N);
```

$$\left\{\left[0,1,\frac{-5}{7},\frac{9}{7},\frac{-5}{7}\right],\left[1,0,\frac{-1}{7},\frac{-1}{7},\frac{6}{7}\right]\right\}$$

```
> basis(N,'rowspace');
```

$$[[2,3,1],[1,-2,4]]$$

```
> rowspace(N);
```

$$\{[1,0,2],[0,1,-1]\}$$

实际上，利用线性变换，对矩阵上三角化，或称“高斯约当消元法（gauss-jord elimination）”，就可以确定矩阵的秩，以及构成矩阵行、列向量的最大线性无关组。相关函

数，我们会在线性方程组一节做具体介绍。

3.3.5　逆矩阵

若矩阵 B 满足 A*B=I（I 为单位矩阵），则矩阵 B 称为矩阵 A 的逆矩阵。记做 A^{-1}。

以上是线性代数中逆矩阵的定义，求逆矩阵是线性代数中的基本问题之一。先介绍逆矩阵的一个推论，这个推论是：

$$\mathbf{AA^*=|A|I}$$

即一个矩阵同它的伴随矩阵（A^*，adjoin matrix）相乘，结果为单位矩阵乘以此矩阵的对应行列式的值。矩阵的伴随矩阵定义为：

$$A^* = \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}$$

这里 A_{ij} 为矩阵 A 的代数余子式，即矩阵 A 除去第 i 行，第 j 列的元素后所剩余的矩阵的 $(-1)^{(i+j)}$ 倍。有关此推论的详细推导过程，请读者参看相关书籍。

利用 Maple 编写一小段程序，可以求出矩阵的伴随矩阵：

```
> restart;
> with(linalg):
> A:=matrix(3,3,[1,2,3,2,6,9,3,9,18]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 6 & 9 \\ 3 & 9 & 18 \end{bmatrix}$$

```
> B:=matrix(3,3);
```

$$B := \text{array}(1..3,1..3,[\ ])$$

```
> for i to 3 do
>   for j to 3 do
>   B[i,j]:=det(minor(A,j,i))*(-1)^(i+j)
>   od
> od;
> print(B);
```

$$\begin{bmatrix} 27 & -9 & 0 \\ -9 & 9 & -3 \\ 0 & -3 & 2 \end{bmatrix}$$

这里用到了一个函数“minor（A,i,j）”，它的作用是求出矩阵 A 的余子式 M_{ij}，再将它乘以$(-1)^{(i+j)}$，就可以得到矩阵 A 的代数余子式 A_{ij}，利用它们按顺序组成新的矩阵，即在上例获得的矩阵 B，即为所求的矩阵 A 的伴随矩阵。实际上，Maple 中提供函数“adj/adjoin()”可以直接获得矩阵的伴随矩阵，即上例中的循环过程可以化简为：

```
> B:=adj(A);
```

$$B := \begin{bmatrix} 27 & -9 & 0 \\ -9 & 9 & -3 \\ 0 & -3 & 2 \end{bmatrix}$$

再将 A 的伴随矩阵除以其行列式，就可以获得矩阵 A 的逆矩阵 A^{-1}：

```
> C:=evalm(B/det(A));
```

$$C := \begin{bmatrix} 3 & -1 & 0 \\ -9 & 1 & \frac{-1}{3} \\ 0 & \frac{-1}{3} & \frac{2}{9} \end{bmatrix}$$

```
> evalm(A&*C);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

这个例子只是为读者介绍一些 Maple 处理线性代数问题的基本方法，同时又介绍了两个函数“minor”和“adj/adjoin()”，如果用户希望计算行列式 A 的逆矩阵，只需使用函数“inverse(A)”，甚至“evalm(A^(-1))”就可以了。如继续上例：

```
> inverse(A);
```

$$\begin{bmatrix} 3 & -1 & 0 \\ -1 & 1 & \frac{-1}{3} \\ 0 & \frac{-1}{3} & \frac{2}{9} \end{bmatrix}$$

```
> evalm(A^(-1));
```

$$\begin{bmatrix} 3 & -1 & 0 \\ -1 & 1 & \frac{-1}{3} \\ 0 & \frac{-1}{3} & \frac{2}{9} \end{bmatrix}$$

3.3.6　矩阵的特征值和特征向量

设 A 是 n 阶方阵，若存在数 λ 及非零向量 **X**，使得：

$$\mathbf{AX}=\lambda\mathbf{X}$$

则称 λ 是 A 的特征值(eigenvalue)，**X** 是属于特征值 λ 的特征向量（eigenvector）。由特征值的定义，我们不难推出矩阵 A 的所有特征值，须满足公式 $|\lambda\mathbf{I}-\mathbf{A}|=0$，我们可以利用此公式计算矩阵的特征值，例如：

```
> A:=matrix(3,3,[3,-1,1,2,0,1,1,-1,2]);
```

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

```
> J:=array(identity,1..3,1..3):print(J);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> f:=det(evalm(x*J-A));
```

$$f:=x^3-5x^2+8x-4$$

```
> solve(f=0,x);
```

$$1,2,2$$

即矩阵 A 的特征值为 1，2（两重）。将特征值代入方程（$\lambda\mathbf{I}-\mathbf{A}$）$\mathbf{X}=\mathbf{0}$，就可以获得特征值对应的特征向量：λ=1 对应向量[0,1,1]，λ=2 对应向量[1,1,0]（两重）。此处略过具体步骤。同上例类似，Maple 提供了函数“eigenvals()”、“eigenvects()”，可以直接获得矩阵对应的特征值与特征向量。如继续操作上例的矩阵 A：

```
> eigenvals(A);
```

$$1,2,2$$

```
> eigenvects(A);
```

$$[2,2,\{[1,1,0]\}],[1,1,\{[0,1,1]\}]$$

eigenvals()函数缺省返回的是矩阵 A 对应的所有实特征值，eigenvects()函数以“**[特征值，此特征值对应维数，{对应特征向量的集合}]**”的形式返回。所有结果一目了然。

矩阵中，同特征值相关的性质还有“特征多项式”。特征多项式即满足方程“$|\lambda \mathbf{I}-\mathbf{A}|=0$”的多项式，我们可以利用它求出矩阵的特征值。在 linalg 程序库中，函数“charpoly()”可以完成此项任务。使用此函数时，须将此函数的第二个参数指定为特征多项式中的变量名。如继续操作上例中的矩阵 A：

```
> charpoly(A,x);
```

$$x^3-5x^2+8x-4$$

```
> factor(%);
```

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

```
> solve(%=0,x);
```

$$1,2,2$$

利用以上提到的几个函数，我们可以验证两个同特征值相关的定理：

1.
$$\sum_{i=1}^{n}\lambda_i=\sum_{i=1}^{n}a_{ii}=trA$$

即矩阵的特征值之和，等于其对角线上的元素之和，又称矩阵的“迹（trace）”：

```
> A:=matrix(3,3,[7,4,-1,4,7,-1,-4,-4,4]);
```

$$A:=\begin{bmatrix}7 & 4 & -1\\ 4 & 7 & -1\\ -4 & -4 & 4\end{bmatrix}$$

```
> eigenvals(A);
```

$$12,3,3$$

```
> 12+3+3;
```

$$18$$

```
> B:=0;
```

$$B:=0$$

```
> for i to 3 do
> B:=A[i,i]+B
> od;
```

$$B := 7$$

$$B := 14$$

$$B := 18$$

linalg 程序库提供函数“trace”计算矩阵的迹，即其对角线上元素的和。例如可以将上例中的循环部分改写为：

```
> trace(A);
```

$$18$$

2. $$\prod_{i=1}^{n} \lambda i = |A|$$

即矩阵所有特征值的乘积，等于其对应行列式的值，如：

```
> A:=matrix(3,3,[1,2,3,4,5,6,7,8,9]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> eigenvals(A);
```

$$0, \frac{15}{2} + \frac{3}{2}\sqrt{33}, \frac{15}{2} - \frac{3}{2}\sqrt{33}$$

```
> det(A);
```

$$0$$

```
> B:=matrix(3,3,[3,-1,1,2,0,1,1,-1,2]);
```

$$B := \begin{bmatrix} 3 & -1 & 1 \\ 2 & 0 & 1 \\ 1 & -1 & 2 \end{bmatrix}$$

```
> eigenvals(B);
```

$$1,2,2$$

```
> det(B);
```

$$4$$

3.3.7　相似矩阵与矩阵的对角化

先对相似矩阵的概念作一简单介绍：

设 A，B 是两个 n 阶方阵，如果存在一个 n 阶可逆矩阵 P，使得 $\mathbf{P^{-1}AP=B}$，则称矩阵 B 相似（similar）于矩阵 A，记做 B~A。

Maple 提供函数“issilimlar（A，B，P）”帮助用户判断两个矩阵 A，B 是否相似，如果判断结果为真，则将使得两矩阵相似的正交矩阵保存至 P 中，如：

```
> A:=matrix(3,3,[9,-5,4,-3,-1,-16,-6,6,-12]);
```

$$A:=\begin{bmatrix} 9 & -5 & 4 \\ -3 & -1 & -16 \\ -6 & 6 & -12 \end{bmatrix}$$

```
> B:=matrix(3,3,[4,4,4,8,-4,-4,-4,8,-4]);
```

$$B:=\begin{bmatrix} 4 & 4 & 4 \\ 8 & -4 & -4 \\ -4 & 8 & -4 \end{bmatrix}$$

```
> issimilar(A,B,P);
```

$$true$$

```
> print(P);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ \frac{5}{4} & -\frac{3}{4} & \frac{7}{4} \\ 0 & -\frac{1}{2} & -\frac{3}{4} \end{bmatrix}$$

为验证 P 是否满足条件，利用相似矩阵的公式进行验证：

```
> evalm(P^(-1) &* B &* P);
```

$$\begin{bmatrix} 9 & -5 & 4 \\ -3 & -1 & -16 \\ -6 & 6 & -12 \end{bmatrix}$$

通过验证公式，我们发现 Maple 是按照公式 $\mathbf{P^{-1}BP=A}$ 计算两矩阵是否相似的。这并不妨碍用户计算公式 $\mathbf{P^{-1}AP=B}$，因为只须将 Maple 计算得出的 P 取逆，就可以获得用户所求的正交矩阵 P。

矩阵的对角化过程，实际就是寻找一个矩阵的对角化的相似矩阵的过程。将矩阵对角化，目的是为了能寻找到一组简单的基矢量表示线性空间。因为相似矩阵可以代表同一个

线性变换，利用简单的对角阵，可以方便线性空间的计算过程。

在线性代数中，可以证明如果一个矩阵存在相似对角矩阵，则此对角阵必是由此矩阵的特征值在对角元上的排列而组成的。由一些判定条件可以确定一个矩阵是否能被对角化，在 Maple 中，可以利用 issimilar 函数来验证这些条件。例如：

```
> A:=matrix(3,3,[1,2,2,2,1,2,2,2,1]);
```

$$A := \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

```
> eigenvals(A);
```

$$5, -1, -1$$

```
> B:=diag(5,-1,-1);
```

$$B := \begin{bmatrix} 5 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

```
> issimilar(B,A,P);
```

$$true$$

```
> print(P);
```

$$\begin{bmatrix} 1 & -1 & 2 \\ 1 & 2 & -5 \\ 1 & -1 & 3 \end{bmatrix}$$

```
> evalm(P^(-1)&*A&*P);
```

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

实现矩阵的对角化，还可以方便对原始矩阵的计算过程。例如计算上例中矩阵 A 的 100 次方，直接计算会面临 100 个矩阵相乘的算式。利用将矩阵 A 的对角化，可以简化为计算 $\mathbf{P}^{-1}\mathbf{B}^{100}\mathbf{P}$。由于矩阵 B 是对角阵，则其 100 次方等于对角线上的元素的 100 次方，在左右分别乘 $\mathbf{P}^{-1}$、$\mathbf{P}$，则计算过程将大大化简　利用此方法，我们可以求得结果为：

```
[2629536350736706018039095217609287432244021450363410015900929768880209,
 2629536350736706018039095217609287432244021450363410015900929768880208,
 2629536350736706018039095217609287432244021450363410015900929768880208]
[2629536350736706018039095217609287432244021450363410015900929768880208,
 2629536350736706018039095217609287432244021450363410015900929768880208,
 2629536350736706018039095217609287432244021450363410015900929768880208,
```

[262953635073670601803909521760928743224402145036341001590092976888020 8]
262953635073670601803909521760928743224402145036341001590092976888020 8,
262953635073670601803909521760928743224402145036341001590092976888020 8,

有兴趣的读者还可以逐次计算 A 的奇、偶次方，会发现一些有趣的规律。

3.4 解线性方程组

本书的 2.6 节曾介绍过使用函数 solve（）求解线性方程组问题。solve（）函数是一种“傻瓜”函数。用户完全不需理会求解过程，系统会自动返回方程的解。这种函数对于初等代数中遇到的二元一次或三元一次方程组很有效，但却无法帮助用户解决线性代数中的一些问题，如方程的基础解系、线性空间的基等等。在这一节中，我们将利用矩阵理论，再次分析线性方程组问题。

3.4.1 Gauss-Jordan 消元法

在初中数学中，解决二元一次或三元一次方程组的基本方法就是消元法，但消元的过程完全凭经验与感觉，而 Gauss-Jordan 消元法则提供了一个标准的消元步骤，使解线性方程组有迹可寻，有法可依。

Gauss 消元法的求解思路是构造方程组的增广矩阵，利用矩阵的行变换将增广矩阵变换为上三角矩阵后，一步步回代即可求得方程组的解。Gauss-Jordan 消元法是 Gauss 消元法的变形。它们的差别在于选择行变换系数的方法略有不同。Gauss 消元法只将系数矩阵变化至上三角阵，而 Gauss-Jordan 消元法则是将方程的系数矩阵完全变形为单位阵，从而直接获得方程的解。使用 linalg 程序库中提供的函数 gausselim 可完成 Gauss 消元，gaussjord 可完成 Gauss-Jordan 消元。例如，我们重复在第 2 章解线性方程组时使用的例子：

$$\begin{cases} x_1 + 2x_2 + 3x_3 = 14 \\ x_1 + 3x_2 + 4x_3 = 19 \\ x_1 + 4x_2 + 2x_3 = 15 \end{cases}$$

在 Maple 中，可以按照如下方法完成对方程组的求解：

```
> restart;
> with(linalg):
Warning, the protected names norm and trace have been
redefined and unprotected
```

```
> A:=matrix(3,3,[1,2,3,1,3,4,1,4,2]);
```

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

```
> a:=vector([14,19,15]);
```

$$a:=[14,19,15]$$

```
> G:=augment(A,a);
```

$$G:=\begin{bmatrix}1 & 2 & 3 & 14\\ 1 & 3 & 4 & 19\\ 1 & 4 & 2 & 15\end{bmatrix}$$

```
> gausselim(G);
```

$$\begin{bmatrix}1 & 2 & 3 & 14\\ 0 & 1 & 1 & 5\\ 0 & 0 & -3 & -9\end{bmatrix}$$

```
> gaussjord(G);
```

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

```
> backsub(%);
```

$$[1,2,3]$$

这里使用 augment 命令扩充方程的系数矩阵 A，目的是向读者解释方程的增广矩阵是如何产生的。最后使用函数 gaussjord 获得的结果就对应着原始方程组的解，即 X_1=1，X_2=2，X_3=3。最后，利用函数 backsub（）获得了向量形式的解。

“backsub（A，v，n）”函数很有意思，它会自动判别用户输入参数的意义，如果输入的参数 v 是一个向量，则系统自动将 A 定义成方程组的系数矩阵，系统返回 A*x=v 的解；如果 v 对应一个矩阵，则系统自动对此矩阵的列向量分别求解，并将解的集合再组成类似的矩阵返回；如果用户没有声明 v，或 v 被定义为一个空值，则系统自动将 A 定义为一个增广矩阵，并求出它的解；如果 A 已经是一个经过 gaussjord 函数处理过的增广矩阵，则 backsub 会以向量形式返回它的解。注意利用 backsub 处理的矩阵必须是上三角或梯形矩阵，否则系统会报错。一般是利用此函数计算由 Gauss 消元法生成的矩阵对应的解，如计算一个由系统利用“randmatrix()”函数随机生成的增广矩阵：

```
> A:=randmatrix(3,4);
```

$$A := \begin{bmatrix} -50 & -12 & -18 & 31 \\ -26 & -62 & 1 & -47 \\ -91 & -47 & -61 & 41 \end{bmatrix}$$

```
> F := gausselim(A);
```

$$F := \begin{bmatrix} -50 & -12 & -18 & 31 \\ 0 & \frac{-629}{25} & \frac{-706}{25} & \frac{-771}{50} \\ 0 & 0 & \frac{2699}{37} & \frac{-1071}{37} \end{bmatrix}$$

```
> backsub(F);
```

$$\left[\frac{-67093}{91766}, \frac{97113}{91766}, \frac{-1071}{2699}\right]$$

需要向读者指出的是，Gauss 消元法同 Gauss-Jord 消元法虽然作用相同，但由于消元步骤不同，两者处理浮点数运算时会得到不同精度的结果，如计算如下增广矩阵：

```
> B:=matrix(4,5,[1.1348,3.8326,1.1651,3.4017,9.5342
  ,0.5301,1.7875,2.5330,1.5435,6.3941,3.4129,4.9317
  ,8.7643,1.3142,18.4231,1.2371,4.9998,10.6721,0.01
  47,16.9237]);
```

$$B := \begin{bmatrix} 1.1348 & 3.8326 & 1.1651 & 3.4017 & 9.5342 \\ .5301 & 1.7875 & 2.5330 & 1.5435 & 6.3941 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 & 18.4231 \\ 1.2371 & 4.9998 & 10.6721 & .0147 & 16.9237 \end{bmatrix}$$

```
> gausselim(B);
```

$$\begin{bmatrix} 3.4129, & 4.9317, & 8.7643, & 1.3142, & 18.4231 \\ 0, & 3.212168933, & 7.495237059, & -.4616681385, & 10.24573785 \\ 0, & 0, & -6.86567487, & 3.279883134, & -3.585814349 \\ 0, & 0., & 0, & .9072689895, & .9072689918 \end{bmatrix}$$

```
> gaussjord(B);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1.000000000 \\ 0 & 1 & 0 & 0 & .9999999991 \\ 0 & 0 & 1 & 0 & 1.000000000 \\ 0 & 0 & 0 & 1 & 1.000000001 \end{bmatrix}$$

```
> backsub(%%); ##gausselim
```

$$[1.000000000, .9999999972, 1.000000001, 1.0000000003]$$

```
> backsub(%%); ##gaussjord
```

$$[1.000000000, .9999999991, 1.000000000, 1.0000000001]$$

此方程的精确解是[1，1，1，1]。由此可以看出两种方法所得出结果的精确程度。对系数为浮点数，但解有可能为整数的运算，我们推荐使用函数 linesolve，例如从上例的增广矩阵中提取系数矩阵和右向量后，用此函数求解线性方程组，读者就可以明显的比较出它同 Gauss 系列消元法的差别：

```
> C:=delcols(B,5..5);
```

$$C := \begin{bmatrix} 1.1343 & 3.8326 & 1.1651 & 3.4017 \\ .5301 & 1.7875 & 2.5330 & 1.5435 \\ 3.4129 & 4.9317 & 8.7643 & 1.3142 \\ 1.2371 & 4.9998 & 10.6721 & .0147 \end{bmatrix}$$

```
> V:=delcols(B,1..4);
```

$$V := \begin{bmatrix} 9.5342 \\ 6.3941 \\ 18.4231 \\ 16.9237 \end{bmatrix}$$

```
> transpose(linsolve(C,V));
```

$$[1 \quad 1 \quad 1 \quad 1]$$

作为解方程的最后一步，建议用户养成验根的习惯，即将解的向量表示同系数矩阵相乘，并判断结果是否同原始方程相等。如验证上例的计算结果：

```
> evalm(C&*transpose(%));
```

$$\begin{bmatrix} 9.5342 \\ 6.3941 \\ 18.4231 \\ 16.9237 \end{bmatrix}$$

```
> evalm(%-V);
```

$$\begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}$$

3.4.2 系数矩阵的提取与线性方程组的生成

在介绍使用最小二乘法解决线性方程组问题之前，先介绍一些同线性方程组相关的简化输入的函数，以方便用户的工作。

（1）提取线性方程组的系数矩阵和增广矩阵：

genmatrix(eqns, vars)
genmatrix(eqns, vars, flag)
genmatrix(eqns, vars, b)

genmatrix（）函数中，参数“eqns”代表一组线性方程组的集合或列表，用户通过指定“vars”来确定方程组中的变量。如果未指定第三个参数，则系统自动生成原始方程的系数矩阵。如果指定第三个参数为标志符“flag”，则系统会将原始方程组等号右边的系数变成生成的系数矩阵的最右一列，即生成原始方程组的增广矩阵。如果指定第三个参数为向量“b”，则系统将原始方程组等号右的参数赋值给向量“b”，生成一列向量。函数具体的使用方法如下：

【例 3-1】提取方程组：$\begin{cases} 3x_1 + 2x_2 + 3x_3 = 44 \\ 2x_1 + 3x_2 + 6x_3 = 29 \\ 6x_1 - 4x_2 + 5x_3 = 53 \end{cases}$ 的系数矩阵。

```
> eq1:=3*x1+2*x2+3*x3=44;
```

$$eq1 := 3x1 + 2x2 + 3x3 = 44$$

```
> eq2:=2*x1+3*x2+6*x3=29;
```

$$eq2 := 2x1 + 3x2 + 6x3 = 49$$

```
> eq3:=6*x1-4*x2+5*x3=53;
```

$$eq3 := 6x1 - 4x2 + 5x3 = 53$$

```
> with(linalg):
Warning, the protected names norm and trace have been
redefined and unprotected
```

```
> A:=genmatrix({eq1,eq2,eq3},{x1,x2,x3});
```

$$A := \begin{bmatrix} 3 & 2 & 3 \\ 2 & 3 & 6 \\ 6 & -4 & 5 \end{bmatrix}$$

```
> genmatrix({eq1,eq2,eq3},{x1,x2,x3},flag);
```

$$\begin{bmatrix} 3 & 2 & 3 & 44 \\ 2 & 3 & 6 & 29 \\ 6 & -4 & 5 & 53 \end{bmatrix}$$

```
> genmatrix({eq1,eq2,eq3},{x1,x2,x3},v);
```

$$\begin{bmatrix} 3 & 2 & 3 \\ 2 & 3 & 6 \\ 6 & -4 & 5 \end{bmatrix}$$

```
> print(v);
```

$$[44, 29, 53]$$

建立方程组的增广矩阵，还可以利用本章 3.2 节所介绍的函数 augment，如合并例中建立的系数矩阵 A 与右向量 v：

```
> augment(A,v);
```

$$\begin{bmatrix} 3 & 2 & 3 & 44 \\ 2 & 3 & 6 & 29 \\ 6 & -4 & 5 & 53 \end{bmatrix}$$

（2）利用系数矩阵、增广矩阵生成线性方程组：

geneqns(A, x)

geneqns(A, x, b)

参数 A 代表系数矩阵，x 为未知数列表，如果指定了向量 b，则以 A 为系数矩阵，向量 b 的元素为方程组等号右方的值生成方程组，否则等号右方将全为零。如：

```
> geneqns(A,x);
```

$$\{3x_1 + 2x_2 + 3x_3 = 0,\ 2x_1 + 3x_2 + 6x_3 = 0,\ 6x_1 - 4x_2 + 5x_3 = 0\}$$

```
> geneqns(A,x,v);
```

$$\{6x_1 - 4x_2 + 5x_3 = 52,\ 3x_1 + 2x_2 + 3x_3 = 44,\ 2x_1 - 3x_2 + 6x_3 = 29\}$$

3.4.3 线性方程组的最小二乘解

到目前为止，我们求解的线性方程组不是定解问题，就是有无穷多的解。然而在处理实际问题的过程中，尤其是利用线性方程组处理实验数据的过程中，经常会遇到方程个数比未知数多，而且有矛盾的方程组，但我们依然希望能得到一组可能的解。这时候，就需要使用最小二乘法（least square）来求解。我们先对最小二乘法的原理做简单介绍：

例如解方程组：

$$\begin{cases} 3x = 10 \\ 4x = 11 \\ 2x = 9 \end{cases}$$

很明显，没有任何有理数可以同时满足这三个方程。而且由于各个方程的地位都是平等的，不可能指定必须满足哪个方程，哪个方程可以不满足，因此无法通过舍去某个方程来获得惟一定解。但我们依然希望由这组方程获得一个解，这时，可以通过寻找一个使得每个方程误差的平方和最小的数值来满足这个方程组，即找到一个最“接近”标准解的数值。使用平方和的目的，是为了消除正负号不同而带来的影响。因此，解上述方程组的问题就转化为寻找满足函数：

$$Y=(3X-10)^2+(4X-11)^2+(2X-9)^2$$

同时又使得 Y 最小的 X 的数值。对此一元二次方程，通过解析几何很容易确定其最低点的位置，但由于今后会面对多元问题，所以希望能用矩阵表示。所以改写方程组为 **ax=b**,其中 $\mathbf{a}=(3,4,2)^T$,$\mathbf{b}=(10,11,9)^T$，则：

$$\begin{aligned} Y&=\|\mathbf{ax}-\mathbf{b}\|^2=(\mathbf{ax}-\mathbf{b},\mathbf{ax}-\mathbf{b}) \\ &=(\mathbf{ax}-\mathbf{b})^T(\mathbf{ax}-\mathbf{b}) \\ &=\mathbf{a}^T\mathbf{a}x^2-2\mathbf{a}^T\mathbf{b}x+\mathbf{b}^T\mathbf{b} \end{aligned}$$

由于这是 X 的二次三项式，则最小值对应 $x=\mathbf{a}^T\mathbf{b}/\mathbf{a}^T\mathbf{a}$。分别代入 **a**，**b**，得到 x=98/50=1.96。

这是在一元问题时的结果，同样，可以把此公式推广到多元函数的线性方程组，对于方程组：**AX=b**，其中 **A** 为系数矩阵，**b** 为方程的右向量，可以推出其最小二乘解为满足方程 $\mathbf{A}^T\mathbf{AX}=\mathbf{A}^T\mathbf{b}$ 的解。即方程的最小二乘解 $\mathbf{X}=(\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{b})$。

先观察一个例子：

【例 3-2】 在最小二乘意义下，求如下一组数据的最佳拟合直线方程。

X	1	2	3	4	5	6
Y	1.3	1.8	2.2	2.9	3.4	4.3

请看如何利用 Maple 解决此问题。

（1）设直线方程为 y=kx+b，代入实验数据。

```
with(linalg):
```

```
> eqns:={k+b=1.3,2*k+b=1.8,3*k+b=2.2,4*k+b
  =2.9,5*k+b=3.4,6*k+b=4.3};
```

$$eqns := \{k+b=1.3, 2k+b=1.8, 3k+b=2.2, 4k+b=2.9, 5k+b=3, 4.6k+b=4.3\}$$

（2）生成原始方程组的系数矩阵及右向量。

```
> A:=genmatrix(eqns,[k,b],v);
```

$$A := \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \\ 6 & 1 \end{bmatrix}$$

```
> print(v);
```

$$[1.3, 1.8, 2.2, 2.9, 3.4, 4.3]$$

（3）定义系数矩阵的转置。

```
> At:=transpose(A);
```

$$At := \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

（4）按照最小二乘公式，计算得出最小二乘解。

```
> sols:=evalm(inverse(evalm(At&*A))&*evalm
  (At&*v));
```

$$sols := [.585714286, .60000000]$$

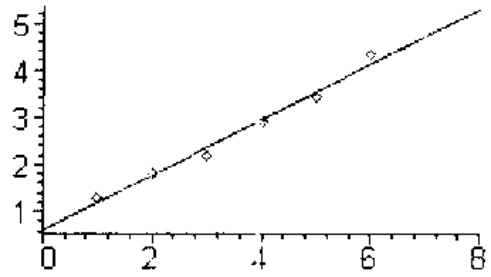
（5）作图确定拟合结果。

```
> with(plots):
```

```
> p1:=plot([[1,1.3],[2,1.8],[3,2.2],[4,2.9
  ],[5,3.4],[6,4.3]],style=point):
```

```
> p2:=plot(0.585714286*x+0.6,x=0..8,color=
  blue):
```

```
> display(p1,p2);
```



从上图中，读者不难观察出最小二乘直线拟合的效果。

Maple 提供的函数 leastsqrs（）可以化简上例中第四步的表达式，甚至可以将第一步至第四步完全替代。leastsqrs（）函数的完全形式为：

leastsqrs(A, b)

leastsqrs(S, v)

leastsqrs(A, b)形式的函数调用可以返回在最小二乘意义下满足方程 A x =b 的向量解。即解向量 x 使得$\|A\ x - b\|^2$最小。leastsqrs(S, v)形式的函数调用则直接利用最小二乘法求解方程列表 S，v 为用户指定的方程组 S 中的变量名，并以集合的形式返回所求未知数的解，如使用此函数操作上例中定义的变量：

```
> leastsqrs(eqns,{b,k});
```

$$\{b = .5999999984, k = .5857142861\}$$

```
> leastsqrs(A,v);
```

$$[.5857142861, .5999999984]$$

读者会发现使用 leastsqrs()函数同输入最小二乘公式而计算得出的结果精度不同（leastsqrs 返回值有 10 位有效数字，而公式计算得出的有效位数一为 8 位，一为 9 位），这是由于 Maple 在两种过程中，采取了不同的计算方法，使用公式会在分步计算的过程中约减有效位数，而使用函数则会在最后一步一次处理出数据。所以建议读者尽可能的使用 Maple 系统定义好的各种函数，以求获得最大的计算精度。

最小二乘法可以给出矛盾方程的最佳解，然而根据最小二乘解的定义公式：$X=(A^TA)^{-1}(A^Tb)$，却不能为有无穷多解的方程给出最优解。因为此公式仅是对 A 的线性变换，无法改变系数矩阵的秩，即无法将不满秩的矩阵变形为满秩矩阵，从而给出线性方程组的定解。但有些时候，即使未知数的个数多于方程的个数，人们依然希望可以获得一组有意义的解，这时，就需要使用**最优最小二乘法**来求解方程，在 Maple 中，可以使用 leastsqrs 函数的第三个参数 optimize，关于如何使用，请看下一节。

3.4.4　线性方程组的最优解

我们先来看一个利用最小二乘法解无定解线性方程组的例子：

【例 3-3】求解方程组：$\begin{cases} x_1 - 2x_2 + x_3 = -5 \\ x_1 + 5x_2 + -7x_3 = 2 \\ 3x_1 + x_2 + -5x_3 = 1 \end{cases}$

```
with(linalg):
> eqs:={x1-2*x2+x3=-5,x1+5*x2-7*x3=2,3*x1+x2
  -5*x3=1
```

$$eqns := \{x1 + 2x2 + x3 = -5, x1 + 5x2 - 7x3 = 2, 3x1 + x2 - 5x3 = 1\}$$

```
> leastsqrs(eqs,{x1,x2,x3});
```

$$\{x1 = -\frac{3}{7} + \frac{9}{7}\_t_1, x2 = \frac{11}{14} + \frac{8}{7}\_t_1, x3 = \_t_1\}$$

由于是非定解问题，leastsqrs 函数以变量“_t1”作为参数表示了解的集合。但这不是普通的无穷多解方程，比如我们利用以前介绍过的 solve 函数再解此方程，发现 Maple 没有返回任何数值，再用 Gauss 消元法化简系数矩阵：

```
> solve(eqs,{x1,x2,x3});
> genmatrix(eqs,{x1,x2,x3},flag);
```

$$\begin{bmatrix} -2 & 1 & 1 & -5 \\ 5 & -7 & 1 & 2 \\ 1 & -5 & 3 & 1 \end{bmatrix}$$

```
> gausselim(%);
```

$$\begin{bmatrix} -2 & 1 & 1 & -5 \\ 0 & \frac{-9}{2} & \frac{7}{2} & \frac{-21}{2} \\ 0 & 0 & 0 & 9 \end{bmatrix}$$

这时我们发现了问题所在，即利用消元法化简后，第三个方程变为了 $0_x1+0_x2+0_x3=9$，是明显的矛盾方程，因此普通的 solve 函数是无法获得原方程的解析解的。我们再来仔细分析一下此时 leastsqrs 函数的求解过程。

利用最小二乘的定义，我们计算线性方程组：$\mathbf{A^TA\ X=A^Tb}$

```
> A:=genmatrix(eqs,{x1,x2,x3},v);
```

$$A := \begin{bmatrix} -2 & 1 & 1 \\ 5 & -7 & 1 \\ 1 & -5 & 3 \end{bmatrix}$$

```
> B:=evalm(transpose(A)&*A);
```

$$B := \begin{bmatrix} 30 & -42 & 6 \\ -42 & 75 & -21 \\ 6 & -21 & 11 \end{bmatrix}$$

```
> b:=evalm(transpose(A)&*v);
```

$$b := [21, -24.0]$$

```
> gausselim*augment(B,b));
```

$$\begin{bmatrix} 6 & -21 & 11 & 0 \\ 0 & -72 & 56 & -24 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> solvs:=backsub(%);
```

$$solvs := \left[\frac{7}{6}+\frac{8}{9}\_t_1, \frac{1}{3}+\frac{7}{9}\_t_1, \_t_1\right]$$

果然，我们看到此时获得的解集同直接使用 leastsqrs 函数获得的一样。此结果告诉我们，有一组无穷多解可以同时使上述线性方程组的误差最小。如果此时用户依然希望可以获得一组有意义的解，则需要使用最优化方法筛选此无穷多解的集合。同最小二乘原理类似，我们依然选择平方和最小这一性质。不过此时计算的是解的平方和。例如计算上例中的解的平方和：

$$S = (\frac{7}{6}+\frac{8}{9}\_t_1)^2+(\frac{1}{3}+\frac{7}{9}\_t_1)^2+(\_t_1)^2$$

利用 Maple，我们可以获得它的最小值：

```
> sum(solvs[i]^2,i=1..3);
```

$$\left(\frac{7}{6}+\frac{8}{9}\_t_1\right)^2+\left(\frac{1}{3}+\frac{7}{9}\_t_1\right)^2+t_1^2$$

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

$$\frac{53}{36}+\frac{70}{27}\_t_1+\frac{194}{81}\_t_1^2$$

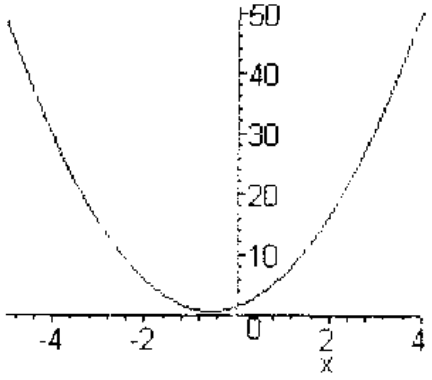
```
> subs(_t[1]=x,%);
```

$$\frac{53}{36}+\frac{70}{27}x+\frac{194}{81}x^2$$

```
> minimize(%,x,location);
```

$$\frac{299}{388},\left\{\left[\{x=\frac{105}{194}\},\frac{299}{388}\right]\right\}$$

```
> plot(%%,x=-5..4);
```



利用函数 minimize，我们求得了最小二乘解的平方和函数的最低点，同时也得到了对应的解 x=-105/194（即_t_1），由此就可以得到另两个由_t_1 表示的解对应的数值：

$$\begin{cases} x_1 = \dfrac{133}{194} \\ x_2 = -\dfrac{17}{194} \\ x_3 = -\dfrac{105}{194} \end{cases}$$

然而这样做毕竟很麻烦。显然，Maple 提供了可以完全化简以上的步骤的函数，即在 leastsqrs 函数中，加入参数 optimize，就可以将以上的步骤省略成一条命令：

```
> leastsqrs(eqs,{x1,x2,x3},optimize);
```

$$\{x2=\frac{133}{194},x3=\frac{-17}{194},x1\frac{-105}{194}\}$$

虽然解的顺序同我们手工推导的顺序不同，但对最终结果没有影响。

至此，我们已将 Maple 中同线性代数相关的基本知识全部介绍完，下一节，我们将介绍一些线性代数在实际问题中的应用，希望能使读者迅速将所学内容为己所用。

3.5 线性代数的实际应用

作为数学的一个重要分支，线性代数在社会的各个领域都得到了广泛的应用。常用的利用涉及几何、物理、经济、管理、运筹学、社会学、人口学、遗传学、生物学等诸方面。由于笔者的经验所限，不可能涉及应用到线性代数的所有领域。在这一节，我们将讨论几种利用线性代数分析实际问题的方法，同时借助 Maple 解决问题。如果读者能通过这一节，

对应用线性代数的方法有些感性的了解，那么我们的目的也就达到了。

3.5.1 几何学应用

1．确定过定点的曲线

线性方程组理论中有一个基本结论：未知数个数与方程个数相同的线性齐次方程组有非零解的充要条件是系数行列式等于零。利用此性质，我们可以确定过定点的曲线方程。例如某圆过三点（x1,y1），（x2,y2），（x3,y3），设其方程为：

$$a_1(x^2+y^2)+a_2x_1+a_2y+a_4=0$$

则通过代入三点的坐标，可以获得关于 a_1,a_2,a_3,a_4 的线性方程组，因为它们不全为零，则线性方程组的系数行列式必为零，即：

$$\begin{vmatrix} x^2+y^2 & x & y & 1 \\ x_1^2+y_1^2 & x_1 & y_1 & 1 \\ x_2^2+y_2^2 & x_2 & y_2 & 1 \\ x_3^2+y_3^2 & x_3 & y_3 & 1 \end{vmatrix}=0$$

【例 3-4】求过三点(1,3),(1,-7),(6,-2)的圆。

```
> A:=matrix(4,4,[x^2+y^2,x,y,1,x1^2+y1^2,x1,y1
  ,1,x2^2+y2^2,x2,y2,1,x3^2+y3^2,x3,y3,1]);
```

$$A:=\begin{bmatrix} x^2+y^2 & x & y & 1 \\ x1^2+y1^2 & x1 & y1 & 1 \\ x2^2+y2^2 & x2 & y2 & 1 \\ x3^2+y3^2 & x3 & y3 & 1 \end{bmatrix}$$

```
> f:=det(A):subs(x1=1,y1=3,x2=1,y2=-7,x3=6,
  y3=-2,f=0);
```

$$-100x+200y+50x^2+50y^2-1000=0$$

```
> student[completesquare](%,x);
```

$$50(x-1)^2-1050+200y+50y^2=0$$

```
> student[completesquare](%,y);
```

$$50(y+2)^2-1250+50(x-1)^2=0$$

虽然结果未化简，但很容易的我们就获得了圆的曲线方程。

如果读者感觉4×4的矩阵太简单，则请再看一个求圆锥曲线方程的例子：

【例 3-5】已知某圆锥曲线通过五个点：(5.764,0.648),(6.268,1.202),(6.759,1.823),(7.168,2.526),(7.408,3.360)，求此圆锥曲线。

类似的，我们可以获得此圆锥曲线需满足的行列式：

```
> A:=matrix(6,6,[x^2,x*y,y*2,x,y,1,33.224,3.
  735,0.420,5.764,0.648,1,39.514,7.556,1.445
  ,6.286,1.202,1,45.684,12.322,3.323,6.759,1
  .823,1,51.380,18.106,6.381,7.168,2.526,1,5
  5.950,25.133,11.290,7.480,3.360,1]);
```

$$A:=\begin{bmatrix} x^2 & xy & y^2 & x & y & 1 \\ 33.224 & 3.735 & .420 & 5.764 & .648 & 1 \\ 39.514 & 7.556 & 1.445 & 6.286 & 1.202 & 1 \\ 45.684 & 12.322 & 3.323 & 6.759 & 1.823 & 1 \\ 51.380 & 18.106 & 6.381 & 7.168 & 2.526 & 1 \\ 55.950 & 25.133 & 11.290 & 7.480 & 3.360 & 1 \end{bmatrix}$$

```
> det(A)=0,
```

$$.0024613994x^2 - .0027973729xy + .0026507505y^2 - .014145081x - .00229097y + .00923859 = 0$$

```
> evalf(det(A),5);
```

$$.00824x^2 - .05320xy + .12751y^2 + .0815x + .2006y + .551$$

利用Maple，可以轻松计算复杂的6阶行列式。

2. 最小二乘法的曲线拟合

在上一节，我们曾介绍了最小二乘法的直线拟合，作为此方法的推广形式，曲线的最小二乘法拟合同直线情况类似。例如有n个数据点（x_1,y_1），$(x_2,y_2),\ldots,(x_n,yn)$满足m次多项式：

$$y=a_0+a_1x+a_2x^2+\cdots+a_mx^m$$

则定义：

$$Y=\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \qquad M=\begin{pmatrix} 1 & x_1 & \cdots & x_1^m \\ 1 & x_2 & \cdots & x_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^m \end{pmatrix} \qquad Y=\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix}$$

可证明U的最小二乘解为：$\mathbf{U}=(\mathbf{M}^{\mathrm{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathrm{T}}\mathbf{Y}$。

【例 3-6】求拟合六点(−1,−11),(1,−5),(−2,−15),(0,4),(3,8),(7,15)的最佳三次多项式。

（1）生成相关系数方程组。

```
> f1:a0+a1*x+a2*x^2+a3*x^3:
> f:=f1-y;
```

$$f := a0 + a1x + a2x^2 + a3x^3 - y$$

```
> eq1:=subs(x=-1,y=-11,f);
  eq2:=subs(x=1,y=-5,f);
  eq3:=subs(x=-2,y=-15,f);
  eq3:=subs(x=0,y=4,f);
  eq5:=subs(x=3,y=8,f);
  eq6:=subs(x=7,y=15,f);
```

$$eq1 := a0 - a1 + a2 - a3 + 11$$

$$eq2 := a0 + a1 + a2 + a3 + 5$$

$$eq3 := a0 - 2a1 + 4a2 - 8a3 + 15$$

$$eq3 := a0 - 4$$

$$eq5 := a0 + 3a1 + 9a2 + 27a3 - 8$$

$$eq6 := a0 + 7a1 + 49a2 + 343a3 - 15$$

（2）最小二乘法求解方程系数。

```
> evalf(leastsqrs({eq1,eq2,eq3,eq4,eq5},{a0,a1
  ,a2,a3}),5);
```

$$\{a3 = .66195, a1 = 1.5193, a0 = -2.5499, a2 = -1.3565\}$$

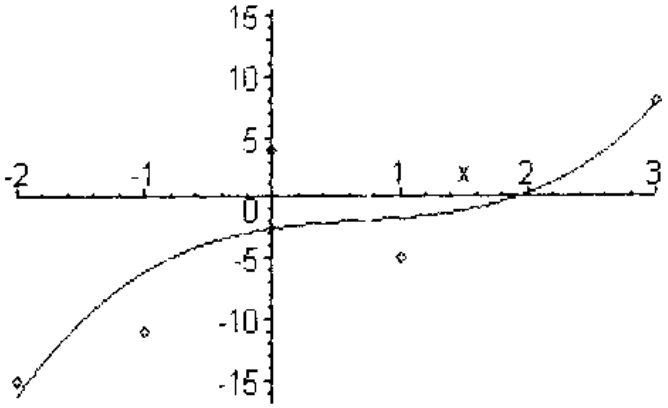
```
> f2:=subs(a0=-2.5499,a1=1.5193,a2=-1.3565,a3=
  0.6619,f1);
```

$$f2 := -2.5499 + 1.5193x - 1.3565x^2 + .6619x^3$$

（3）验证方程的拟合效果。

```
> with(plots):
> p1:=plot(f2,x=-2..3):
> p2:=plot([[-1,-11],[1,-5],[-2,-15],
  [0,4],[3,8],[7,15]],style=point):
```

```
> display(p1,p2);
```



3.5.2 物理学应用

在本章的第3节，我们曾提到在高能物理或高等量子力学中，会遇到求高阶（384~1024阶）矩阵的逆的情况。由于其公式过于复杂，我们在这里只介绍简单的线性代数应用：一个利用矩阵解决平衡温度分布问题。

作为热平衡理论的初步研究，我们假定研究对象是有限长均匀物质。分子运动的简化模型告诉我们，当均匀物质处于热平衡时，其中任意点 P 的温度，等于以它为圆心，完全包含在此物质中的任意球面上的温度的平均值。为进一步的简化，我们把研究对象缩小为二维有限宽平板。利用此平均值性质，虽然可以唯一确定平板的温度分布，但计算过程并不容易。利用有限元的方法，即将平板以相交的平行线划分为四边形网格，则可以将此平均值性质应用于网格的交点上，近似的结论为平板内部网格点的温度，是其四个相临格点温度的平均值。这样，就可以将原本无限的问题化简为有限元。同时减小网格间的距离，由此而得出的近似值也就更接近于实际的温度分布。

看如下的一个例子：

【例 3-7】利用有限元法求处于热平衡状态的四边温度分别为 0℃，0℃，1℃，2℃的梯形薄板内的温度分布。

如图 3-3 所示，将梯形划分为 6×4 的网格。内部格点温度分别设为 $t_1,t_2,\ldots,t_9$：

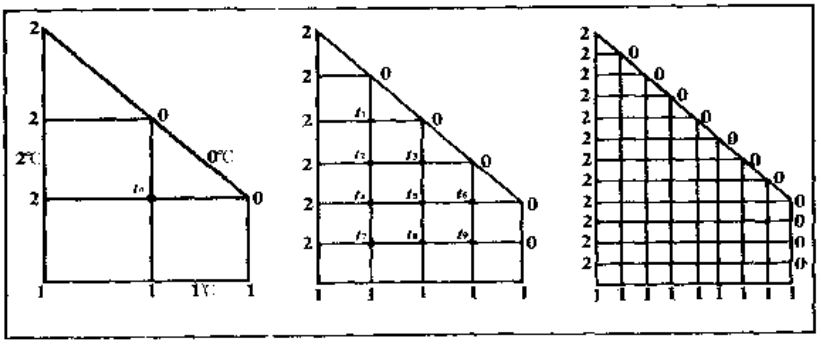


图 3-3 薄板上的有限元温度分布

$$t_1=\frac{1}{4}(t_2+2+0+0),\qquad t_2=\frac{1}{4}(t_1+t_3+t_4+2)$$

$$t_3=\frac{1}{4}(t_2+t_5+0+0),\qquad t_4=\frac{1}{4}(t_2+t_5+t_7+0)$$

$$t_5=\frac{1}{4}(t_3+t_4+t_6+t_8),\qquad t_6=\frac{1}{4}(t_5+t_9+0+0)$$

$$t_7=\frac{1}{4}(t_4+t_8+1+2),\qquad t_8=\frac{1}{4}(t_5+t_7+t_9+1)$$

$$t_9=\frac{1}{4}(t_6+t_8+1+0)$$

这是九个未知数，九个方程的线性方程组，写成矩阵形式为：

$$\mathbf{t=Mt+b}$$

可以将它化简成：

$$\mathbf{t=(I-M)^{-1}b}$$

让我们利用 MAPLE 来求解此方程组：

```
> with(linalg):
Warning, the protected names norm and trace have
been redefined and unprotected
```

```
> eq1:=(t2+2):eq2:=(t1+t3+t4+2):eq3:=)t2+t5):
  eq4:=(t2+t5+t7+2):eq5:=(t3+t4+t6+t8):eq6:=(
  t5+t9):eq7:=(t4+t8+1+2):eq8:=(t5+t7+t9+1):e
  q9:=(t6+t8+1):
```

```
> M:=evalm(genmatrix([eq1,eq2,eq3,eq4,eq5,eq6
  eq7,eq8,eq9],[t1,t2,t3,t4,t5,t6,t7,t8,t9],
  b));
```

$$M:=\begin{bmatrix}0&1&0&0&0&0&0&0&0\\1&0&1&1&0&0&0&0&0\\0&1&0&0&1&0&0&0&0\\0&1&0&0&1&0&1&0&0\\0&0&1&1&0&1&0&1&0\\0&0&0&0&1&0&0&0&1\\0&0&0&1&0&0&0&1&0\\0&0&0&0&1&0&1&0&1\\0&0&0&0&0&1&0&1&0\end{bmatrix}$$

```
> print(b);
                    [-2,-2,0,-2,0,0,-3,-1,-1]
> J:=array(identity,1..9,1..9):
> t:=evalm(inverse(evalm(J-M/4))&*(-1/4*b));
```

$$t := \left[\frac{5452}{6949}, \frac{7910}{6949}, \frac{52463}{111184}, \frac{144177}{111184}, \frac{20823}{27796}, \frac{36305}{111184}, \frac{18061}{13898}, \frac{100223}{111184}, \frac{7741}{13898}\right]$$

```
> evalf(%,5);
[.78457,1.1383,.47186,1.2967,.74914,.32653,1.2995,.90142,
    .55699]
```

注意在计算过程中，Maple 的“genmatrix”函数是将方程组中的所有常数移至等号右边后再赋值给向量 b 的，所以在最后的公式中，需要对向量 b 反号。同时，为简化输入，我们将方程的系数矩阵统一扩大了 4 倍，因此在最后的计算过程中又将 M，b 统一除以 4。

高中物理中，解决力学平衡问题一般是利用四边形法则，或利用受力分解的方法，即将任意方向上的力都分解为水平分量和垂直分量，再将平衡点的受力问题化简为水平方向上的力学平衡与垂直方向上的力学平衡。在学习了线性代数中的向量理论后，相信都会转而使用向量间的运算来处理相关问题。Maple 中有丰富的向量运算函数，可以帮助用户完成所希望完成的运算。由于相关问题都比较简单，在本章的第一节也对向量间的运算做了详细的说明，我们在这里就不再赘述。

3.5.3　运筹学应用

在日常生活和生产管理中，经常会遇到如何有效、合理的利用有限的人力、物力、财力等资源，得到最佳经济效益的问题，即如何确定最佳的分配方案。这就是运筹学中线性规划问题所要解决的对象。为一件任务建立相应的数学模型，通过适当的变形与运算获得有意义的解，这就是线性规划的基本处理过程。举个简单的例子：

【例 3-8】某工厂生产 A、B 两种机器，生产一台 A 需要的原料为：煤 9 吨，钢 4 吨，木 3 立方米，A 单价 7 万元；生产一台 B 需要的原料为煤 4 吨，钢 5 吨，木 10 立方米，B 单价 12 万元。则利用现有原料：煤 360 吨，钢 200 吨，木 300 立方米，如何生产才能获得最大效益？

设总共生产 A x_1 台，B x_2 台，则 x_1，x_2 需满足如下方程：

$$9x_1+4x_2 \le 360$$
$$4x_1+5x_2 \le 200$$

$$3x_1+10x_2 \le 300$$
$$x_1 \ge 0, x_2 \ge 0$$

同时，需要使得总效益 $S=7x_1+12x_2$ 最大。

传统的线性规划方法是上述方程添加几个未知数，将不等式转化为标准型：

$$9x_1+4x_2+x_3 = 360$$
$$4x_1+5x_2+x_4 = 200$$
$$3x_1+10x_2+x_5 = 300$$
$$x_1 \ge 0, x_2 \ge 0, x_3 \ge 0, x_4 \ge 0, x_5 \ge 0,$$

再利用单纯形法求解方程组，过程十分复杂。请看如何利用 Maple 解决此类问题：

```
> eq1:=9*x1+4*x2+x3=360;eq2:=4*x1+5*x2+x4=200;
  eq3:=3*x1+10*x2+x5=300;S:=7*x1+12*x2;
```

$$eq1 := 9x1 + 4x2 + x3 = 360$$
$$eq2 := 4x1 + 5x2 + x4 = 200$$
$$eq3 := 3x1 + 10x2 + x5 = 300$$
$$S := 7x1 + 12x2$$

```
> with(simplex):
> maximize(S,{eq1,eq2,eq3},NONNEGATIVE);
```

$$\{x5 = 0, x4 = 0, x3 = 84, x2 = 24, x1 = 20\}$$

```
> subs(%,S);
```

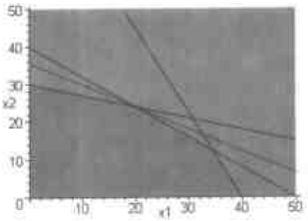
$$428$$

简单的几步，就可以获得所希望的结果。为验证此结果，可以使用画图程序包中附带的对不等式作图的函数“inequal”：

```
> ieq1:=9*x1+4*x2=360;ieq2:=4*x1+5*x2<=200;ieq3:
  =3*x1+10*x2<=300;
```

$$eq1 := 9x1 + 4x2 \le 360$$
$$eq2 := 4x1 + 5x2 \le 200$$
$$eq3 := 3x1 + 10x2 \le 300$$

```
> with(plots):
> implicitplot(7*x1+12*x2=428,x1=0..50,x2=0..50):
> inequal({ieq1,ieq2,ieq3},x1=0..50,x2=0..50,opti
  onsexcluded=(color=gray,thickness=2,symbol=circ
  le)):
```




```
6);
```

有关这里使用的两个新函数："implicitplot"和"inequal"，我们都会在第 8 章"Maple 的绘图功能"中做详细介绍。读者在这里可以先对其功能有一定的感性了解。从我们获得的图形中可以看出，利用 maximize 函数获得的直线"$7X_1+12X_2=428$"正好落在三个不等式所围成的凸四边形最外边的顶点处，就是同时满足此组不等式的最大解。

线性代数在运筹学的应用还涉及对策论，指派问题等方面，由于其过程相对来说十分繁琐，一般需要大量的人工化简，所以不在本书中涉及。但一般情况下所涉及的函数本书基本都已介绍过，至于能否灵活使用它们，还要看读者对 Maple 运用的熟练程度以及是否有一定的创造性思维。

3.5.4 计算机图形学

坐标变换是矩阵最基本的应用，而图形的坐标变换更是计算机图形处理的重要内容。基本变换包括对图形的投影、比例缩放、平移、旋转、错切和透视等。这里仅对其中的几个做简单介绍，想深入了解的读者可以参考计算机图形学方面的书籍。

1. 旋转变换

二维旋转变化矩阵为：$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$。

【例 3-9】将顶点坐标为（3，0），（6，2），（4，3）的三角形绕原点逆时针旋转 45°。

```
> with(plots):
> loopplot:=proc(L)
>     plot([op(L),L[1]],args[2..nargs]);
> end:
```

```
> L1:=[[3,0],[6,0],[7/2,2]];
```

$$L1 := \left[[3,0],[6,0],\left[\frac{7}{2},2\right]\right]$$

```
> p1:=loopplot(L1):
> trans:=x->array([[cos(x),sin(x)],[-sin(x),
  cos(x)]]);
```

$$trans := x \rightarrow \mathrm{array}([[\cos(x),\sin(x)],[-\sin(x),\cos(x)]])$$

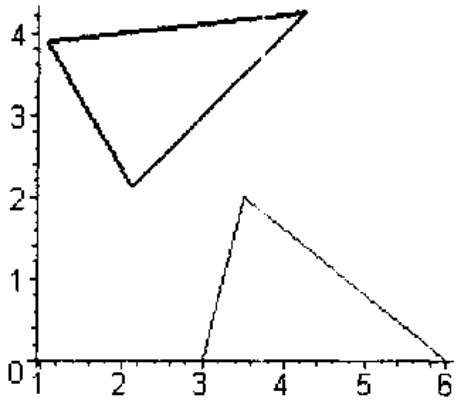
```
> evalm(L1&*trans(Pi/4));
```

$$\begin{bmatrix} \frac{3}{2}\sqrt{2} & \frac{3}{2}\sqrt{2} \\ 3\sqrt{2} & 3\sqrt{2} \\ \frac{3}{4}\sqrt{2} & \frac{11}{4}\sqrt{2} \end{bmatrix}$$

```
> L2:=evalf([[3/2*sqrt(2), 3/2*sqrt(2)],
  [3*sqrt(2), 3*sqrt(2)], [3/4*sqrt(2),
  11/4*sqrt(2)]],5);
```

$$L2 := [[2.1213,2.1213],[4.2426,4.2426],[1.0606,3.8890]]$$

```
> p2:=loopplot(L2,thickness=2):
> display(p1,p2,scaling=CONSTRAINED);
```



在这个例子中，我们使用了很多还未介绍的技术：创建自定义函数“loopplot”用到了“过程”，这部分会在第 7 章“Maple 6 程序设计”中做详细介绍；Display 函数会在第 8 章“Maple 的绘图功能”中具体介绍。如果读者对上例有不理解的地方，请参看相应的章节。

三维旋转变换比二维图形的旋转要复杂得多，但可以将其看成是多个二维变换的合成。最简单的方法是将一个三维旋转变换视为三个绕坐标轴的二维旋转变换，因此变换矩阵可以分为三组：

绕 X 轴旋转 α 角变换：

$$(x^*,y^*,z^*)=(x,y,z)\begin{pmatrix}1 & 0 & 0\\ 0 & \cos\alpha & \sin\alpha\\ 0 & -\sin\alpha & \cos\alpha\end{pmatrix}$$

绕 Y 轴旋转 β 角变换：

$$(x^*,y^*,z^*)=(x,y,z)\begin{pmatrix}\cos\beta & 0 & -\sin\beta\\ 0 & 1 & 0\\ \sin\beta & 0 & \cos\beta\end{pmatrix}$$

绕 Z 轴旋转 γ 角变换：

$$(x^*,y^*,z^*)=(x,y,z)\begin{pmatrix}\cos\gamma & \sin\gamma & 0\\ -\sin\gamma & \cos\gamma & 0\\ 0 & 0 & 1\end{pmatrix}$$

三维矩阵的多重旋转就是对上述变换进行连乘。最后的公式相对比较复杂，这里就不再列出。读者可以利用 Maple 将上述三个矩阵连乘就会获得相应结果。

2. 透视投影变换

此类变换只涉及三维图形。投影的作用就是将不好直接显示的三维图象投射到二维平面上，从而获得可以在常用显示设备上显示的平面图形。一般可以把投影分为平行投影与透视投影（中心投影）。当利用三维物体上各点在某方向上的平行投影线与投影平面的交点来组合新的投影时，产生的就是平行投影图像，投影图像与投影中心间的距离对投影效果没有任何影响；当投影中心到投影平面的距离有限时，通过物体不同点的投影线彼此不平行的投影被定义为透视投影（perspective projection）。透视投影由于相似于人眼平时观察物体的效果，一般有很好的三维立体感，但不足之处是会破坏原始图像各点间的实际距离和比例。平行投影的三维效果不强，但符合原始比例，适合于对物体的多角度观察。这里我们只对透视投影做简单介绍。

透视投影一般分为三类，即一点透视、两点透视和三点透视。将一组平行线在远端投影成一个点，此点被称为灭点。顾名思义，一点透视即只会产生一个灭点的透视方法，两点、三点透视即分别对应两个、三个灭点。以两点透视为例，我们来看一下透视投影的效果。两点透视的变换矩阵为：

$$T_2=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\l&m&n&1\end{pmatrix}\begin{pmatrix}\cos\theta&\sin\theta&0&0\\-\sin\theta&\cos\theta&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix}\begin{pmatrix}1&0&0&0\\0&1&0&-\frac{1}{y_v}\\0&0&1&0\\0&0&0&1\end{pmatrix}$$

这里（l,m,n）对应三维物体的坐标平移，目的是使它更接近 XYZ 坐标系的原点，θ 角对应物体绕 Z 轴的旋转角度，y_v 对应视点（0, y_v,0）。读者可以发现透视变换矩阵实际上是几种坐标变换的组合。变换这几个参数，我们会直接感受其对透视效果的影响。下面看一下 Maple 如何对一个正方体做两点透视变换：正方体的坐标为(0,0,0), (0,2,0), (0,2,2),(0,0,2),(2,0,0),(2,2,0),(2,2,2),(2,0,2)。

变换参数为：(l,m,n)=(0,0,2); θ =45°; y_v=6。

```
> restart;
> with(linalg):
Warning, the protected names norm and trace have been
redefined and unprotected
> A:=matrix(8,3,[[0,0,0],[0,2,0],[0,2,2],[0,0,2]
   ,[2,0,0],[2,2,0],[2,2,2],[2,0,2]]):
> vector(8,1):
> A1:=concat(A,%);
```

$$A1:=\begin{bmatrix}0&0&0&1\\0&2&0&1\\0&2&2&1\\0&0&2&1\\2&0&0&1\\2&2&0&1\\2&2&2&1\\2&0&2&1\end{bmatrix}$$

```
> T1:=(l,m,n)->matrix(4,4,[[1,0,0,0,],[0,1,0,0],[0
  ,0,1,0],[l,m,n,1]]);
T1 := (l, m, n) →
   matrix(4,4,[[1,0,0,0],[0,1,0,0],[0,0,1,0],[l, m, n,1]])
```

```
> T2:=a->matrix(4,4,[[cos(a),sin(a),0,0],[-sin(a)
  ,cos(a),0,0],[0,0,1,0],[0,0,0,1]]);
```

$$T2 := a \to \text{matrix}(4,4,\ [[\cos(a),\sin(a),0,0],[-\sin(a),\cos(a),0,0],[0,0,1,0],[0,0,0,1]])$$

```
> T3:=y->matrix(4,4,[[1,0,0,0],[0,1,0,-1/y],[0,0,1,
  0],[0,0,0,1]]);
```

$$T3 := y \to \text{matrix}\left(4,4,\left[[1,0,0,0],\left[0,1,0,-\frac{1}{y}\right],[0,0,1,0],[0,0,0,1]\right]\right)$$

```
> T:=evalm(T1(0,0,-2.2)&*T2(Pi/4)&*T3(6));
```

$$T := \begin{bmatrix} \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} & 0 & -\frac{1}{12}\sqrt{2} \\ -\frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} & 0 & -\frac{1}{12}\sqrt{2} \\ 0 & 0 & 1 & 0 \\ 0. & 0. & -2.2 & 1. \end{bmatrix}$$

```
> A2:=evalm(A1&*T):
```

```
> v:=col(A2,4);
```

$$v := \left[1.,1.-\frac{1}{6}\sqrt{2},1.,-\frac{1}{6}\sqrt{2},1.,1.-\frac{1}{6}\sqrt{2},-\frac{1}{3}\sqrt{2}+1.,-\frac{1}{3}\sqrt{2}+1.,1.-\frac{1}{6}\sqrt{2}\right]$$

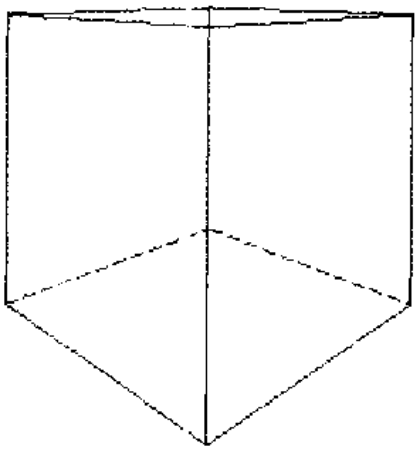
```
> B:=delcols(delcols(A2,2..2),3..3);
```

$$B := \begin{bmatrix} 0. & -2.2 \\ -\sqrt{2} & -2.2 \\ -\sqrt{2} & -2 \\ 0. & -2 \\ \sqrt{2} & -2.2 \\ 0 & -2.2 \\ 0 & -.2 \\ \sqrt{2} & -.2 \end{bmatrix}$$

```
> P:=array(1..8);
```

$$P := \text{array}(1..8,[\ ])$$

```
> for i to 8 do
>  P[i]:=convert(evalf(evalm(row(B,i)/v[i]),5),list)
> od:
> L1:=plot([P[2],P[1],P[4],P[1],P[5]],linestyle=3):
> L2:=plot([P[2],P[6],P[5],P[8],P[4],P[3],P[2]]):
> L3:=plot([P[6],P[7],P[3],P[7],P[8]]):
> plots[display](L1,L2,L3,axes=NONE;
```



3.6 特殊函数矩阵

作为线性代数部分的最后一节，我们对 Maple 中附带的特殊函数矩阵做简要介绍。在近世代数中，针对不同问题，数学家们创造了很多特殊行列式和特殊矩阵，例如贾克比（Jacobian）行列式、范德门（Vandermonde）行列式、西耳伯特（Hilbert）矩阵等等。Maple 将其中一些行列式、矩阵定义成函数形式，方便了用户对它们的创立。有关这些特殊矩阵的具体性质及用途，请有兴趣的读者参看相应的数学手册。在这一节，我们只对函数本身的形式做初步介绍。

3.6.1 随机矩阵

利用函数 randmatrix（m，n，option），系统可以产生 m×n 的随机数矩阵。用户可以通过指定“option”参数来选择矩阵的形状。其参数同 array 函数中可选择的参数类似，有：“symmetry：对称矩阵”；“antisymmetric：非对称矩阵”；“sparse：零矩阵”以及“unimodular：上三角矩阵”，缺省时系统自动以参数生成“dense：一般矩阵”。

需要提醒读者的是，由于 Maple 随机数发生函数有一定问题，所以按顺序产生的随机数矩阵完全一样，如：

```
> restart;
```

```
> with(linalg):randmatrix(4,4);
```

Warning, the protected names norm and trace have been redefined and unprotected

$$\begin{bmatrix} -85 & -55 & -37 & -35 \\ 97 & 50 & 79 & 56 \\ 49 & 63 & 57 & -59 \\ 45 & -8 & -93 & 92 \end{bmatrix}$$

```
> restart;
> with(linalg):randmatrix(4,4);
```

Warning, the protected names norm and trace have been redefined and unprotected

$$\begin{bmatrix} -85 & -55 & -37 & -35 \\ 97 & 50 & 79 & 56 \\ 49 & 63 & 57 & -59 \\ 45 & -8 & -93 & 92 \end{bmatrix}$$

3.6.2 斐波那契（Fibonacci）矩阵

Fibonacci 常数定义为 $N_{n+1}=N_n+N_{n-1}$（$N_0=1$，$N_1=1$），则此数列为：1,1,2,3,5,8,13,21……，相应的，定义 Fibonacci 矩阵为：

$$U_{n+1}=\begin{pmatrix} U_n & V_n \\ V_n & 0 \end{pmatrix}$$

其中 $U_0=1$，$U_1=1$，$U_2=\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$，$V_n=U_{n-1}+V^{T}_{n-1}$。

前几阶 Fibonacci 矩阵如下：

```
> fibonacci(2);
```

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

```
> fibonacci(3);
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

```
> fibonacci(4);
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

同 Fibonaci 矩阵相关的，还可以定义矩阵

$$Q \equiv \begin{pmatrix} F_2 & F_1 \\ F_1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \Rightarrow Q^n \equiv \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

其中 Fn 为 n 阶 Fibonacci 数。读者可以利用 combinat 函数库中的 Fibonacci 函数来创建 n 阶 Fibonacci 数，比如：

```
> with(combinat):
> seq(fibonacci(i),i=0..10);
```

0,1,1,2,3,5,8,13,21,34,55

再并利用对 Q_1 求 n 阶矩阵来验证此公式。

3.6.3 希耳伯特（Hilbert）矩阵

希耳伯特（Hilbert）矩阵中的各个元素的定义为：$H_{ij}=(i+j-x)^{-1}$,用户可以利用 linalg 程序库中的 hilbert（n，x）函数来创建 n 阶希耳伯特矩阵，如果用户未指定 x，则系统默认 x=1. Hilbert 矩阵的逆矩阵元也有类似的公式表达：

$$(H^{-1})_{ij} = \frac{(-1)^{i+j}}{i+j-1} * \frac{(n+i-1)!(n+j-1)!}{[(i-1)!(j-1)!]^2 (n-i)!(n-j)!}$$，如：

```
> hilbert(2);
```

$$\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} \end{bmatrix}$$

```
> hibert(3);
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

```
> inverse(%);
```

$$\begin{bmatrix} 9 & -36 & 30 \\ -36 & 192 & -180 \\ 30 & -180 & 180 \end{bmatrix}$$

对于矩阵元为 $H_{ij}=(i+j-x)^{-1}$ 的矩阵，其对应的行列式也有其特定的规律，感兴趣的读者可以查阅数学手册，这里仅介绍其三阶矩阵：

```
> hilbert(3,x+1);
```

$$\begin{bmatrix} \frac{1}{1-x} & \frac{1}{2-x} & \frac{1}{3-x} \\ \frac{1}{2-x} & \frac{1}{3-x} & \frac{1}{4-x} \\ \frac{1}{3-x} & \frac{1}{4-x} & \frac{1}{5-x} \end{bmatrix}$$

```
> det(%);
```

$$-4\,\frac{1}{(-1+x)(-3+x)^3(-5+x)(-4+x)^2(-2+x)^2}$$

3.6.4　托普里斯（Toeplitz）行列式

Toeplitz 行列式的定义为：

元素 R_k（N 个数：k=-1,...N）组成一个行列式：

$$T=\begin{pmatrix} r_1 & r_2 & \cdots & r_{n-1} & r_n \\ r_2 & r_3 & \cdots & r_n & r_1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ r_n & r_1 & \cdots & r_{n-2} & r_{n-1} \end{pmatrix}$$

Toeplitz 矩阵实际上是一种特殊的对称矩阵，利用 linalg 函数库中的 Toeplitz（v）函数，可以对列表 v 按照如上规律组成矩阵。例如：

```
> v:=[a,a+1,a+2,a+3,a+4]:
> toeplitz(v);
```

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

让我们继续观察 Toeplitz 矩阵的行列式有什么规律：

```
> gausselim(%);
```

$$\begin{bmatrix} a & a+1 & a+2 & a+3 & a+4 \\ 0 & -2\frac{a+1}{a} & -2\frac{a+1}{a} & -\frac{2a+2}{a} & -2\frac{a+3}{a} \\ 0 & 0 & -4\frac{a+1}{2a+1} & -2\frac{2a+3}{2a+1} & -4\frac{a+2}{2a+1} \\ 0 & 0 & 0 & -\frac{2a+3}{a+1} & -2\frac{a+2}{a+1} \\ 0 & 0 & 0 & 0 & -4\frac{a+3}{2a+3} \end{bmatrix}$$

```
> det(%);
```

$$16a+32$$

3.6.5 范德门（Vandermonde）行列式

上小节介绍的 toeplize 行列式实际是一种特殊的 Vandermonde 行列式，此行列式的定义为：

$$V=\begin{vmatrix} 1 & 1 & \cdots & 1 \\ a_1 & a_2 & \cdots & a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{n-1} & a_2^{n-1} & \cdots & a_n^{n-1} \end{vmatrix}=\prod_{1\le j\le i\le n}(a_j-a_i)$$

Maple 使用函数 vandermonde(v)定义的范德门行列式为上述定义的转置，但不影响其结果，v 是“$a_1 \ldots a_n$”的列表。如：

```
> vandermonde([a,b,c,d]);
```

$$\begin{bmatrix} 1 & a & a^2 & a^3 \\ 1 & b & b^2 & b^3 \\ 1 & c & c^2 & c^3 \\ 1 & d & d^2 & d^3 \end{bmatrix}$$

```
> det(%);
```

$$bc^2d^3 - d^2c^3b - d^3cb^2 + ca^3d^2 + c^3a^2d - b^3dc^2 - c^2d^3a + c^3d^2a + d^3ca^2 - ca^3d^2 - c^3a^2d + a^3dc^2 + d^3ab^2 - d^2ab^3 - bd^3a^2 + bd^2a^3 + b^3da^2 - da^3b^2 - b^2c^3a + c^2b^3a + c^3ba^2 - c^2ba^3 - cb^3a^2 + a^3cb^2$$

```
> factor(%);
```

$$-(c-d)(a-d)(a-c)(-d+b)(b-c)(b-a)$$

3.6.6 向量的 Wronskian 矩阵

利用函数 Wronskian()，用户可以创建向量的 Wronskian 矩阵，函数的完全形式为：

Wronskian(f,v)

其中 f 为一向量或列表形式，v 为一变量。

Wronskian()函数创建的矩阵，其中的第(i,j)位置的元素，等于 diff(f[j],v(i-1))的结果，即向量 f 的第 j 个元素，对变量 v 的 i-1 次微商。比如：

```
> A:=vector([x,x^2,x^3]);
```

$$A := [x, x^2, x^3]$$

```
> Wr := wronskinan(A, x);
```

$$W := \begin{bmatrix} x & x^2 & x^3 \\ 1 & 2x & 3x^2 \\ 0 & 2 & 6x \end{bmatrix}$$

```
> det(Wr);
```

$$2x^3$$

```
> B :=vector([exp(x), cos(x), cosh(x), ln(x)]);
```

$$B := [e^x, \cos(x), \cosh(x), \ln(x)]$$

```
> Wr2 := wronskian(B,x);
```

$$Wr2 := \begin{bmatrix} e^x & \cos(x) & \cosh(x) & \ln(x) \\ e^x & -\sin(x) & \sinh(x) & \frac{1}{x} \\ e^x & -\cos(x) & \cosh(x) & -\frac{1}{x^2} \\ e^x & \sin(x) & \sinh(x) & 2\frac{1}{x^3} \end{bmatrix}$$

3.6.7 多项式的 Hessian 矩阵

函数 hessian()的完全形式为：

hessian(expr,vars)

由 hessian 函数生成的矩阵，其中位于（i,j）位置的元素，等于 diff(expr,vars[i],vars[j])的结果，比如：

```
> hessian(f(x,y), [x, y]);
```

$$Wr2 := \begin{bmatrix} \frac{\partial^2}{\partial x^2} f(x,y) & \frac{\partial^2}{\partial y\,\partial x} f(x,y) \\ \frac{\partial^2}{\partial y\,\partial x} f(x,y) & \frac{\partial^2}{\partial y^2} f(x,y) \end{bmatrix}$$

```
> hessian(f(x, y, z), [x, y, z]);
```

$$\begin{bmatrix} \frac{\partial^2}{\partial x^2} f(x,y,z) & \frac{\partial^2}{\partial y\,\partial x} f(x,y,z) & \frac{\partial^2}{\partial z\,\partial x} f(x,y,z) \\ \frac{\partial^2}{\partial y\,\partial x} f(x,y,z) & \frac{\partial^2}{\partial y^2} f(x,y,z) & \frac{\partial^2}{\partial z\,\partial x} f(x,y,z) \\ \frac{\partial^2}{\partial z\,\partial x} f(x,y,z) & \frac{\partial^2}{\partial z\,\partial x} f(x,y,z) & \frac{\partial^2}{\partial z^2} f(x,y,z) \end{bmatrix}$$

```
> hessian(x*y*z, [x, y, z]);
```

$$\begin{bmatrix} 0 & z & y \\ z & 0 & x \\ y & x & 0 \end{bmatrix}$$

第4章 微积分运算

微积分是现代高等数学的基础，Maple 提供了相当丰富的函数来处理微积分的事务。本章我们通过一些实例来看看 Maple 是如何处理函数极限、序列、级数、微分和积分（包括不定积分、定积分、积分变换以及特殊函数的积分和数值积分）的。

4.1 极限

微积分是以极限为基础的，正因为有了极限的严格理论，微积分才摆脱了单纯几何与运动的直观理解和解释，成为一门有严密理论基础的科学。下面让我们来看看如何用 Maple 来求极限。

4.1.1 一元函数的极限

在 Maple 中求极限的函数是 limit（或 Limit），完整的函数表达式是：

limit(f(x), x=a [,dir]);

Limit(f(x), x=a [,dir]);

其中 f 是代数表达式（即我们要求极限的函数），x 是代数表达式的自变量，a 是极限点，可以是一个常数，也可以是正无穷（infinity）和负无穷(-infinity)，dir 是极限的方向（left 和 right）或是 real 和 complex，它是一个可选变量，当参数空时，系统自动取实数。到底如何使用，我们先来看一下下面的例子。

【例 4-1】求极限 $\lim_{x\to 0} x\sin\frac{1}{x}$。

```
> f:=x*sin(1/x);
```

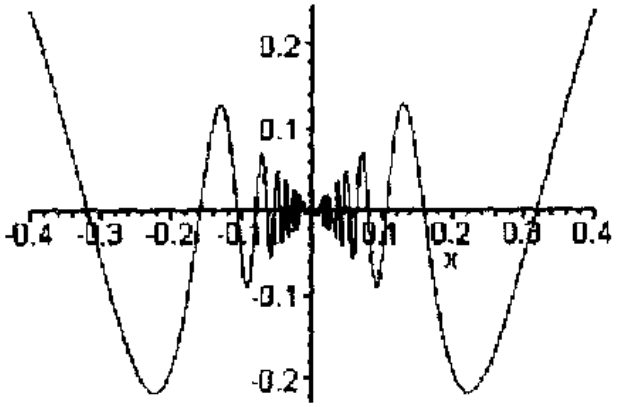
$$f := x\sin\left(\frac{1}{x}\right)$$

```
> limit(f,x=0);
```

$$0$$

用 Maple 很快就求出了函数的极限，为了更直观地显示出极限的值，我们用 Maple 的画图功能验证一下。

```
> plot(f,x=-0.4..0.4);
```



从图中可以看出，函数 f 的极限确实是 0。上面的极限是一个两边的极限，下面让我们看看 limit 函数的 dir 参数的功能：

```
> limit(f,x=0,left);
                        0
> limit(f,x=0,right);
                        0
> limit(f,x=0,real);
                        0
> limit(f,x=0,complex);
```

$$\lim_{x \to 0, \text{complex}} x \sin\left(\frac{1}{x}\right)$$

从上面的例子可以很明显的看出 dir 参数的功能，在本例中，由于复数不能求三角函数，所以函数在自变量 x 以复数形式趋于 0 时不存在。

上面是一个很简单的函数，可以直观的看出。下面让我们再来看一个比较复杂的例子。

【例 4-2】求 $\lim_{x\to\infty}\left(\frac{\sqrt{1+x^4}-\sqrt[3]{x^6-2x^2}}{x^2\tan(1/x)\sin(1/x)(1-\cos(1/x))}\right)$。

这里我们先定义待求的函数，然后再利用极限函数 limit 来求极限：

```
> f:='f':f:=x->(sqrt(1+x^4)-(x^6-2*x^2)^(1/3))/(x^
  2*tan(1/x)*sin(1/x)*(1-cos(1/x)));
```

$$f := x \to -\frac{\sqrt{1+x^4}-(x^6-2x^2)^{\left(\frac{1}{3}\right)}}{x^2\tan\left(\frac{1}{x}\right)\sin\left(\frac{1}{x}\right)\left(1-\cos\left(\frac{1}{x}\right)\right)}$$

我们要求的极限是无穷远处的，所以使用参数infinity。

```
> limit(f,x=infinity);
```

$$\frac{7}{3}$$

上面两个例子中函数的极限都是跟变量的方向无关的，下面我们再来看一个在不同的方向上极限不同的例子。

【例 4-3】 求极限 $\lim_{x\to 0}\frac{1}{x}$。

```
> limit(1/x,x=0);
```

undefined

系统没有给出我们所需要的极限，这是因为我们只是给定了极限的点，并没有指出自变量如何趋于极限点（是正向还是负向），而在这个例子中当 x 从负值趋于零时，极限为+∞而 x 从正值逼近 0 时，极限是-∞。在这种情况下用 Maple 进行计算就必须指定方向。

```
> limit(1/x,x=0,left);
```

$$-\infty$$

4.1.2　多元函数的极限

多元函数求极限的公式与一元函数的完全相同，只是参数的含义稍有区别。具体的函数表达式为：

```
limit(expr(x1,x2,...,xn),{x1=a1,x2=a2,...,xn=an},dir);
Limit(expr(x1,x2,...,xn),{x1=a1,x2=a2,...,xn=an},dir);
```

其中 expr 是一个多元函数表达式，而大括号中提供的是极限点处各自变量所取的值，如果其中的 x_i 没有取值时，系统将其保留不动，请看下面的例子。

```
> limit(x+y,x=1,left);
```

$$1+y$$

```
> limit(x+y,{x=1,y=1});
```

$$2$$

第三个参数同样是一个可选参数，可取 left，right，real 或 complex，在多自变量时，各自变量取相同的方向，并且当各自变量以不同的方向趋进所求的点得到的值不相等时，系统将给出一个 undefined 信息。如：

```
> limit((x^2-y^2)/(x^2+y^2),{x=0,y=0});
```

undefined

```
> limit(x*y,{x=0,y=infinity});
```

undefined

4.1.3 复变函数的极限

在 Maple 中没有专门用来求复变函数的极限的函数，但是它的 limit 函数支持复数操作，我们仍然可以很方便的求出复变函数的极限。其中有两种方法：第一种就是利用 4.1.2 中提到的用多元函数求极限的方法，将复数的实部和虚部当成两个独立的自变量来求解，这种方法在理论上任何时候都适用。

```
> z:x+y*I;
```

$$z := x + Iy$$

```
> f:=(abs(z))^2;
```

$$f := |x + Iy|^2$$

```
> limit(evalc(f),{x=1,y=1});
```

$$2$$

第二种方法就是在所求的函数 f(z)只有自变量 z，而没有单独的 x 和 y 出现时，可以用一元函数求极限的方法来求。

```
> f:='f':z:='z':
  f:=z->(z+4)/(z-4);
  limit(f(z),z=-4+4*I);
```

$$f := z \to \frac{z+4}{z-4}$$

$$\frac{1}{5} - \frac{2}{5}I$$

显然当函数中有单独的 x，y 出现时，用第二种方法不太合适。

4.1.4　函数的连续性

函数的连续性在高等代数中有很重要的意义，有时对比较复杂的函数表达式用手工计算既费时又很容易出错，用 Maple 中提供的专门函数（iscont）来判断函数是否连续就比较简单了，具体的函数表达式如下：

iscont(expr,x=a..b,dir);

其中 expr 是一个代数表达式，即我们所要判断的函数表达式。x=a..b 用来表示需要判断的自变量所在的区间，a 和 b 都只能取实数，当 a 比 b 大时，系统会自动将其转换。dir 是一个可选的变量，可以取 open、closed 或什么都不选，用来表示是在开区间中判断函数的连续性还是在闭区间中判断，默认值是在开区间中进行，当在闭区间中判断连续性是将检查起始点（a）和终止点（b）的连续性。

当表达式在给定的区间中连续时，函数返回 true，否则函数返回 false，而当函数 iscon 不能判断表达式的连续性时，将返回 FAIL（注意无法判断和不连续时的区别）。如：

```
> iscont(1/x,x=0..1);
                        true
> iscont(1/x,x=0..1,open);
                        true
> iscont(1/x,x=0..1,closed);
                        false
> iscont(1/(exp(x)+b),x=1..2);
                        FAIL
```

Maple 除了提供一个判断函数表达式是否连续的函数外，还提供两个函数来寻找函数表达式的不连续点，它们是 discont 和 fdiscont。discont 的具体表达式为：

discont(f, x);

其中 f 是所求的函数表达式，x 是自变量名。discont 将返回所有的不连续点，包括正无穷和负无穷，在返回的值当中通常会包括_Zn~、_NNn~和_Bn~等符号，表示当返回的表达式中的这些变量取整数（_Zn~）、非负整数（_NNn~）或二进制数（_Bn~）0 和 1 时，函数 f 在这些点都不连续。例如：

```
> discont(1/(x-1),x);
                        {1}
```

```
> discont(1/(sin(x)-(1/2)^(1/2)),x);
```

$$\{\frac{1}{4}\pi - \frac{1}{2}\pi\_B1 \sim + 2\pi\_Z1 \sim$$

第二个例子表示当_B1~取 0 或 1，_Zn~取任意整数时，函数表达式在上述点都不连续。

fdiscont 函数比较复杂，它带有 5 个参数，其具体表达式为：

fdiscont(f, domain, res, ivar, eqns):

f 是代数表达式；domain 是求解的区间；res 是用户期望得到的值的精度；ivar 是独立变量的名称；eqns 是一个可选的等式，用来设置系统运算的参数，在一般情况下我们可以不设这个参数而使用系统默认的参数，但有一个比较重要的等式是'newton'=b_newt(b_newt 取 true 或 false)，这个等式用来设置运算和最后返回值选取的方法，假如参数 res 没有设置的话，一般要用到这个等式，我们用以下几个例子来说明：

```
> fdiscont(1/(x^2-1),x=-2..2);
[-1.00024078055225774..-.9992218137113746014,
   .9995577174433442O6..1.000645590070947441
> fdiscont(1/(x^2-1),x=-2..2,10^(-10));
[-1.000000000237218241..-.999999999352049081,
   .999999999661888681..1.000000000762145241
> fdiscont(1/(x^2-1),x=-2..2,newton=true);
                    [-1..1.]
```

注意在 fdiscont 中要设置精度，否则有可能出现意想不到的结果。

4.2 序列与级数

序列是按自然数的顺序排列的一列数学对象，也可以看成是自然数集上的一个函数。而级数是无穷和，是有限个数学对象或函数求和的推广，在函数表示、近似计算、微分方程求解等很多方面都有不可替代的作用。Maple 提供了很多函数来处理这些问题，本节我们从创建序列入手，介绍用 Maple 处理序列和级数的方法，在介绍序列的过程中我们还将顺带介绍列表和集合，因为这对我们处理序列有帮助。

4.2.1 创建一个序列

序列的创建多种多样，平时求解方程时得到的解其实就是一个序列：

```
> 4*x^6+(16-4*a)*x^5+(12*a-75)*x^4+(63-9*a)*x^3;
```

$$4x^6 + (16 - 4a)x^5 + (12a - 75)x^4 + (63 - 9a)x^3$$

```
> SqA:=solve(%,x);
```

$$SqA := 0,0,0,-7+a,\frac{3}{2},\frac{3}{2}$$

现在 SqA 就是一个包含六个元素的序列。当然我们也能用直接的办法来创建一个数列：

```
> SqB:=111,2^x,x^3,sin(4),a;
```

$$SqB := 111,2x,x^3,\sin(4),a$$

这是一个五元序列，假如我们在创建序列 SqB 的时候用 SqA 代替最后一个元素 a，我们得到的将不是像上面那样包含五个元素的序列，而是十元序列，如下：

```
> SqB:=111,2^x,x^3,sin(4),SqA;
```

$$SqB := 111,2x,x^3,\sin(4),0,0,0,-7+a,\frac{3}{2},\frac{3}{2}$$

这是因为序列在 Maple 中通常是被看成是多个元素的，当生成序列时 Maple 通常会将先前序列中的元素组合到新序列中。

Maple 除了支持可以用两种最直接的方法来生成序列外，还提供专门的函数 seq 来生成序列。函数 seq 的具体形式如下：

```
seq(f, i = m..n);
seq(f, i = x);
```

其中第一个参数 f 是一个代数表达式，第二个参数是设定自变量 i 的取值范围，可以是区间，也可以是另外一个序列或集合、列表等。

```
> x:=seq(i^2,i=1..5);
```

$$x := 1,4,9,16,25$$

```
> seq(i mod 5,i=x);
```

$$1,4,4,1,0$$

第一个序列是直接计算函数在区间[1，5]上的整数得到的值的集合，而第二个序列在计算序列的值时自变量用到了第一个序列的值。序列的通项不光是数值形的，也可以生成是字符串形的：

```
> seq(1,i="a".."f");
```

"a","b","c","d","e","f"

4.2.2 序列的基本运算

1. 赋值操作

由于序列是多个元素的集合，在进行各种运算时都是对多个元素进行操作，而 Maple 又把序列看成是一个多元素对象，因此有些普通数值和符号运算规则需要稍做修改，否则系统将报告“参数错误”的信息，比如：

```
> SqC:=a,b,a+b,a-b,a*b,a/b;
```

$$SqC := a, b, a+b, a-b, ab, \frac{a}{b}$$

```
> subs(a=3,b=4,SqC);
Error, wrong number (or type) of parameters in function subs
```

系统给出了“wrong number (or type) of parameters in function subs”(在函数 subs 中参数的数值或类型错误)，这是因为 subs 只能包含一个代数表达式，而 Maple 却将序列 SqC 看成是多个代数表达式。为了解决这个问题，我们不得不先将序列转换成一个列表（list），操作完成以后再转换成列表，先看下面的例子：

```
> op(subs(a=3,b=4,[SqC]));
```

$$3, 4, 7, -1, 12, \frac{3}{4}$$

上面的 op 函数的功能是提取列表的各元素得到一个序列，这个函数的具体用法我们将在稍后讲到。

2. 函数运算

上面对序列进行赋值运算时是将序列转换为列表，按照同样的思路，当我们想将一个列表的元素当成一个函数的自变量的值进行运算，这样的方法似乎也是可行的。但是事实上并不如此，例如：

```
> exp([%]);
Error, exp expects its 1st argument, x to be of type
algebraic, but received [3, 4, 7, -1, 12, 3/4]
```

对于这种困境，我们似乎无能为力了，幸运的是 Maple 提供了一个函数 map，可用来解决同时对多个对象进行操作的问题，具体表达式为：

map(fcn, expr, arg2, ..., argn);

fcn 为操作名，expr 是任何一个表达式，argi 是一个可选操作名。请看下面的例子：

```
> op(map(exp,[SqC]));
```

$$e^{a}, e^{b}, e^{(a+b)}, e^{(a-b)}, e^{(ab)}, e^{\left(\frac{a}{b}\right)}$$

用 map 函数可同时对多个对象进行同一种操作，在上面就是将列表里的元素都取指数。

3. 从序列中按位置寻找元素

序列是一个有序元素的集合，每一个元素在序列中都有特定的位置，因此可以进行直接寻找，这种方法与在数组得到元素的方法几乎相同，例如：

```
> SqC[1];
```

$$a$$

```
> SqC[2..4];
```

$$b, a+b, a-b$$

还有一种方法是用 Maple 提供的函数 op 来提取，但是一个比较麻烦的问题是 op 函数只能用来提取列表或集合里的某一个或某一段元素，对序列则无能为力，在对序列进行操作前还必须将其转换为列表或集合。op 函数的具体表达式有以下几种格式：

```
op(i, e);
op(i..j, e);
op(e);
op(list, e);
```

e 代表表达式，包括列表和集合，i，j 表示元素的位置（也可以是区间），当 i 为 0 时将给出表达式 e 的类型，请看例子：

```
> op(1,{SqC});
```

$$a$$

```
> op(1..3,[SqC]);
```

$$a, b, a+b$$

```
> op(0,[SqC]);
```

$$list$$

```
> op(0,{SqC});
```

$$set$$

在使用 op 函数时需要注意的是不能将其直接作用在序列上，否则会出现“参数的数值或类型有误”的错误返回信息，例如：

```
> op(1,SqC);
Error, wrong number (or type) of parameters in function op
```

这是因为 op 函数总是将参数转换成序列的形式，当本身参数为序列时，op 就会转换失败。

4. 判断元素是否在序列中

要判断序列中是否包含某个元素可以使用 Maple 提供的 member 函数，但是很不幸的是这个函数与 op 函数一样不能直接作用于序列上，我们同样得将序列先转换成列表或集合。member 带三个参数，具体的表达式为：

member(x, s, 'p');

x 是任何一个元素的表达式，s 是列表或集合的名称，'p'是一个可选的参数，用来存储所找到的元素在列表或集合中的位置。当元素在集合或列表中时，member 函数返回 true，p 的值等于元素的位置，否则返回 false，p 值不变。

```
> member(a,[SqC],'p');
                              true
> p;
                               1
> member(aa,[SqC],'q');
                             false
> q;
                               q
```

5. 寻找最大和最小值

用 Maple 提供的函数 min，max 可以很容易得到序列中元素的最小值和最大值：

```
> SqD:=seq(n!,n=1..5);
                    SqD := 1,2,6,24,120
> min(SqD);
                               1
> min(SqD);
                              120
```

假如列表中元素的最大值或最小值有两个或两个以上时，函数也将只返回元素的值，而对最大值的个数置之不理：

```
> max(2,10,10,10,5,6);
                              10
```

在判断含有变量的元素的大小时需要注意变量的正负，在默认情况下，系统认为变量

是实数，下面这个例子中 Maple 就不能给出最大值：

```
> max(6,a+6,2*a+6);
```

$$\max(6, a+6, 2a+6)$$

但是假如我们假定 a 是一个正数的话，函数就能给出准确的判断：

```
> assume(a,positive);
> max(6,a+6,2*a+6);
```

$$2a+6$$

6. 寻找满足特定条件的元素

利用函数 select 可以提取表达式中满足一定条件的元素，其具体表达式是：

$$\text{select}(f,e,b_1..b_n);$$

f 是一个值为布尔型的表达式，e 是一个表达式，$b_1..b_n$ 是一些可选的参数，用来为表达式 f 提供一个附加的判断条件：

```
> SqE:=seq(i,i=100000..100100);
> op(select(isprime,[SqE]));
```

$$100003,100019,100043,100049,100057,100069$$

```
> SqF:=2*exp(a*x),sin(x),ln(y);
```

$$SqF := 2e^{(a*x)}, \sin(x), \ln(y)$$

```
> op(select(has,[SqF],x));
```

$$2e^{(a*x)}, \sin(x)$$

与 select 函数相对应的是函数 remove，其功能是提取不满足条件的元素，用法与 select 函数完全相同。还有一个函数是 selectremove，其功能刚好是 select 和 remove 的总和，即得到的序列是上面两个函数得到的序列的和，用法与上面两个函数都是相同的。

```
> op(remove(has,[SqF],x));
```

$$\ln(y)$$

```
> selectremove(has,[SqF],x);
```

$$[2e^{(a*x)}, \sin(x)], [\ln(y)]$$

7. 序列各元素之间的代数运算

假如我们想将一个序列中的所有的项都相加，我们该怎么办呢？这在级数中经常要用到。在高级语言中我们可以编个循环程序来进行运算，而在 Maple 中很简单，可以用 convert 函数来运算，convert 的功能很强大，这里我们仅举例说明其对序列的代数运算功能：

```
> SqG:=20$10;
```

$$SqG := 20,20,20,20,20,20,20,20,20,20$$

```
> convert([SqG],`+`);
```

$$200$$

```
> SqH:=seq(x^n,n=1..5);
```

$$SqH := x, x^2, x^3, x^4, x^5$$

```
> convert([[SqH],`*`);
```

$$x^{15}$$

从上面的例子可以看到，函数 convert 的功能很简单，它可以只带两个参数，第一个参数是列表或集合，第二个参数是各元素之间的运算关系。这里需要注意的一点是在加号和乘号两边的两撇（“`”）是与符号“~”位于同一键的符号，而不是单引号。

4.2.3 级数的定义与展开

上一小节我们讨论了序列的求和，假如序列的求和是无穷项的，那么这个和就是级数，也就是说级数是序列（也可称为数列）的无穷求和，假如序列部分项求和收敛，则称为级数收敛。

1. 幂级数展开

在高等数学中经常要用到级数展开来运算，其中最常见的就是 Taylor 展开，在 Maple 中是用函数 series 来实现的，其具体表达式是：

series(expr, eqn);
series(expr, eqn, n);

expr 是需要展开的表达式，eqn 是一个等式(例如 x = a) 或是一个变量名 (例如 x)，第三个参数 n 是级数展开的阶数(n-1)，可以是任意一个非负整数，当第三个参数为空时，系统会根据全局变量 Order 来决定展开的阶数，系统默认值是 Order=6，展开的阶数是 Order-1。下面先让我们来看几个例子：

```
> series(exp(x),x);
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + O(x^6)$$

```
> series(exp(x),x,10);
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + \frac{1}{5040}x^7 + \frac{1}{40320}x^8 + \frac{1}{362880}x^9 + O(x^{10})$$

要展开 10 阶，我们也可以通过设定全局变量的值来实现：

```
> Order:=10;
```

$$Order := 10$$

```
> series(exp(x),x);
```

$$1+x+\frac{1}{2}x^2+\frac{1}{6}x^3+\frac{1}{24}x^4+\frac{1}{120}x^5+\frac{1}{720}x^6+\frac{1}{5040}x^7+\frac{1}{40320}x^8+\frac{1}{362880}x^9+O(x^{10})$$

假如想将结果写成多项式，我们可以用函数 convert 将高阶项去掉：

```
> ps:=convert(%,polynom);
```

$$ps := 1+x+\frac{1}{2}x^2+\frac{1}{6}x^3+\frac{1}{24}x^4+\frac{1}{120}x^5+\frac{1}{720}x^6+\frac{1}{5040}x^7+\frac{1}{40320}x^8+\frac{1}{362880}x^9$$

假如我们想在点 x=a 处展开，我们只需将第二个参数设成 x=a 即可：

```
> f:=1/(x-2);
```

$$f := \frac{1}{x-2}$$

```
> series(f, x=1, 4);
```

$$-1-(x-1)-(x-1)^2-(x-1)^3+O((x-1)^4)$$

2. 泰勒 (Taylor) 展开

Maple 提供两个函数 taylor、mtaylor 来分别处理一元表达式和多元表达式的展开，它们的参数设置与幂级数展开 series 相同，具体使用方法如下，从中可以看出 Taylor 展开其实也是一种幂级数展开：

```
> taylor(1/(x-1),x,4);
```

$$-1-x-x^2-x^3+O(x^4)$$

```
> readlib(mataylor):
> mtaylor(exp(x)+y^2,{x=0,y=0},3);
```

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

3. Laurent 级数展开

高等数学中除了经常用到幂级数展开外，还会用到 Luarent 级数展开，其函数为 luarent，具体表达式为：

laurent(f, x=a, n);
laurent(f, x, n);

函数所包含的参数的含义与幂级数和 Taylor 展开相同，f 是待展开的代数表达式，x 或 x=a 设定变量名和展开点，n 设定展开的阶数，是一个非负整数，可选可不选，下面看一下具体的例子：

```
> laurent(1/(x*sin(x)),x=0);
```

$$\text{laurent}\left(\frac{1}{x\sin(x)}, x=0\right)$$

在上面的例子中我们想用 Laurent 函数来展开 1/(xsin(x))，但是 Maple 却并没有按我们的要求来展开，这是为什么呢？原因就在于默认情况下，系统并不知道 luarent 展开的具体形式，为了使其写成具体表达式，我们需要在函数前面调用 with(numapprox)：

```
> with(numapprox):
  laurent(1/(x*sin(x)),x=0,10);
```

$$x^{-2}+\frac{1}{6}+\frac{7}{360}x^2+\frac{31}{15120}x^4+\frac{127}{604800}x^6+O(x^7)$$

4. 泊松（possion）级数展开

possion 展开的函数是 possion，具体表达式如下：

poisson(f, v);
poisson(f, v, n);

f 是带展开的代数表达式，v 是待展开的变量名或变量集合，n 是展开的阶数：

```
> f :=sin(3*w+x)cos(2*w-y);
```

$$f := \sin(3w+x)\cos(-2w+y)$$

```
> poisson(f,[x,y],3);
```

$$\frac{1}{2}\sin(5w)+\frac{1}{2}\sin(w)+\left(\frac{1}{2}\cos(w)+\frac{1}{2}\cos(5w)\right)x+\left(\frac{1}{2}\cos(w)-\frac{1}{2}\cos(5w)\right)y+\left(\frac{1}{2}\sin(5w)-\frac{1}{2}\sin(w)\right)yx+\left(-\frac{1}{4}\sin(5w)-\frac{1}{4}\sin(w)\right)x^2+\left(-\frac{1}{4}\sin(5w)-\frac{1}{4}\sin(w)\right)y^2$$

5. 傅立叶 (Fourier) 展开

与上面三种级数展开有点区别，其具体表达式如下：

fourier(expr, t, w);

expr 是待展开的代数表达式、等式表达式或等式集合，t 是 fourier 变换时需要变换掉的参数，w 是变换后的变量名。在使用这个函数时要注意，它并不直接返回具体的代数表达式，而是用一些特殊函数表示：

```
> with(inttrans):
> fourier(exp(I*t*x),t,w);
```

$$2\pi\,\mathrm{Dirac}(-w+x)$$

4.2.4 级数的基本运算

级数是一个无穷求和，求和是它的基本运算，也是判断其是否收敛的重要手段。Maple 中没有专门为级数设计的求和函数，但我们可以用普通的求和函数来进行求和运算，这个函数就是 sum 或 Sum，共有四种用法，这里我们只介绍对级数求和，其具体表达式如下：

sum(f,k=m..n);
Sum(f,k=m..n);

其中 f 为待求和的级数的通项表达式，k 为求和变量，求和将由 m 到 n:

```
> Sn:=1/x^n;
```

$$Sn := \frac{1}{x^n}$$

```
> sum(Sn,n=1..infinity);
```

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

用 Maple 很容易就求出了级数的无穷和，下面我们再来看看级数在某一区间的求和是什么样子的：

```
> sum(Sn,n=4..10;
```

$$\frac{1}{x^4}+\frac{1}{x^5}+\frac{1}{x^6}+\frac{1}{x^7}+\frac{1}{x^8}+\frac{1}{x^9}+\frac{1}{x^{10}}$$

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

$$\frac{x^6+x^5+x^4+x^3+x^2+x+1}{x^{10}}$$

4.3 微分

在本章的第一节我们曾经介绍过极限，微分的概念就是从极限中来的。例如函数 f(x) 的微分（即导数）的定义就是：lim(f(x+h)-f(x))。因此可以用求极限的方法来求函数的微分。

4.3.1 一元函数的微分运算

回想高等数学课本上关于求导数的内容，几乎无一例外的都是用求极限的方法求导数来作为微分章节的开场白，我们现在也来看看用本身的定义来求导是什么样的：

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

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

```
> limit((f(x+h)-f(x))/h,h=0);
```

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

在 Maple 中用导数的定义很容易求出导数，但是这样还是比较麻烦，除了要多写一个变量外，还要多加一个 f(x+h)，由于求导几乎是高等数学中最常见的问题，所以应该有一个尽量简单的函数来处理这类问题，在 Maple 中是用函数 diff 和 Diff 来求微分的。对于求一元函数的微分来说，这两个函数的表达式都比较简单，具体形式如下：

```
diff(f,x);
Diff(f,x);
```

f 是待求导的代数表达式，x 是变量名。注意：这两个函数的返回值有所不同，Diff 返回没有经过计算的结果，它的返回值只是一个求微分的表达式，而 diff 返回的是经过运算后得到的结果。我们先利用下面这个例子来熟悉一下这个函数：

```
> Diff(x/x^2+1),x);
```

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

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

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

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

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

在这个例子中我们用到的代数表达式与上一个例子的相同，观察结果发现这两种方法得到的结果是完全相同的。

上面我们求得了一元函数的一阶导数，如果我们要求高阶导数怎么办？按照平时笔算的方法是对一阶导数再求导得到二阶导数，对二阶导数求导得到三阶导数，...依次类推。这种方法理论上在 Maple 中进行运算是可行的，但是有点麻烦，特别是在求导的阶数比较高时。幸运的是在 Maple 中可以直接求高阶导数，只要将函数 diff 中的第二个参数多写几遍，比如 diff(f,x,x,x)就是求三阶导数，当然利用 Maple 中$符号的特殊功能我们能将上面那个式子写成更简单的形式 diff(f,x$3)。下面看几个例子：

```
> simplify(diff(x/(x^2+1),x,x,x));
```

$$-6\frac{x^4-6x^2+1}{(x^2+1)^4}$$

```
> simplify(diff(x/(x^2+1),x$3));
```

$$-6\frac{x^4-6x^2+1}{(x^2+1)^4}$$

我们可以用平时的算法来验证一下：

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

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

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

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

```
> diff(%,x);
```

$$-6\frac{x}{(x^2+1)^2}-\frac{8x^3}{(x^2+1)^3}$$

```
> simplify(diff(%,x));
```

$$-6\frac{x^4-6x^2+1}{(x^2+1)^4}$$

可以看到这两种方法得到的结果是一样的。

在一元函数的微分运算中还可以用一个函数 D。函数 D 并不仅仅用于一元函数求微分，也能进行各种代数表达式、不带自变量的表达式、各表达式之间各种代数运算后的微分。

现在我们只是介绍一下在一元函数的微分运算中的用法，其他的用法在后面将详细介绍。在用于一元函数的微分运算时其表达式如下：

D(f)(x);

其中 f 是一个代数表达式，x 是对应于求导的变量名。用在这种情况下，函数 D 的功能与 diff(f,x)是相同的，例如：

```
> D(x->x/(x^2+1))(x);
```

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

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

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

在写函数 D 的第一个参数时，我们用“x->”给出了变量名，这样函数就能有针对的对代数表达式进行求导，假如我们不写的话，函数 D 运算的时候将会将代数表达式中的自变量 x 当成是一个函数来求导，而不是一个变量，这从下例中很容易看出来。

```
> D(x/(x^2+1))(x);
```

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

4.3.2 多元函数的偏微分运算

有了一元函数微分运算的基础，我们进行多元函数的偏微分运算就容易多了，因为在 Maple 中它们用到的函数是相同的，只是参数稍有不同。例如要对代数表达式 f(x,y)同时求 x,y 的导数：即求 $\frac{\partial}{\partial x}\left(\frac{\partial}{\partial y}\mathrm{f}(x,y)\right)$，我们只需用函数 diff(f,x,y)即可：

```
> f(x,y):=(x)^2*(y^2);
```

$$\mathrm{f}(x,y):=x^2y^2$$

```
> simplify(diff(f(x,y),x,y));
```

$$4xy$$

假如要对其中的某个变量进行多次求导，我们只要在函数 diff 中第一个参数后连续加就行了，比如求：$\frac{\partial}{\partial y}\left(\frac{\partial}{\partial x}\left(\frac{\partial}{\partial y}\left(\frac{\partial}{\partial x}\mathrm{f}(x,y)\right)\right)\right)$。我们只需将第一个参数代数表达式后面的参数设成 x，y，x，y 即可，从下面的例子中我们能够得到较充分的了解。

```
> f(x,y):=x^6*y^6;
```

$$f(x, y) := x^6 y^6$$

```
> diff(f(x,y),x,y,x,y);
```

$$900x^4 y^4$$

当然我们也能用函数 D 来实现此功能，下面我们来详细介绍此函数的用法，它的具体表达式如下：

D(f);

D[i](f);

f 是一个代数表达式，参数 i 是一个正整数，用来标识求偏微分的自变量的位置。

当函数 D 为第一种形式时，其作用等效于 unapply(diff(f,x),x)，可以对一个不带变量的表达式(比如：sin，exp 等)进行求解，利用这一性质我们可以很轻易地得到各种代数表达式之间的进行基本运算后的求偏微分规则，比如：

$$\frac{\partial}{\partial x}\frac{f}{g} = \frac{\left(\frac{\partial}{\partial x} f\right) g - \left(\frac{\partial}{\partial x} g\right) f}{g^2}$$

下面我们来看几个例子：

```
> D(cos);
```

$$-\sin$$

```
> D(ln);
```

$$a \to \frac{1}{a}$$

```
> D(h/k);
```

$$\frac{\mathrm{D}(h)}{k} - \frac{h\mathrm{D}(h)}{k^2}$$

```
> D(h@k);     #compute the diff(h(k(x)),x);
```

$$((\mathrm{D}(h))@k)\,\mathrm{D}(k)$$

```
> D(sin@y);
```

$$(\cos@y)\mathrm{D}(y)$$

当 D 为第二种形式时，它适合于对代数表达式进行偏微分运算，功能与函数 diff 相同。其实它的最普遍的表达式是 D[i,j,m..](f)，用来求多元函数的偏微分。

```
> f:='f':
> f:=(x,y,z)->x^6*y^6*z^6:
> D[1,2,3](f);
```

$$(x, y, z) \to 216x^5 y^5 z^5$$

```
> D[2,1,1](f);
```

$$(x, y, z) \rightarrow 180x^4y^5z^6$$

```
> D[1,2,2](f)(0,0,0);
```

$$0$$

```
> D[1$2,2](f);
```

$$(x, y, z) \rightarrow 180x^4y^5z^6$$

假如函数 D 的第一个参数（即方括号[]中的参数）不存在的话，函数将不进行任何运算，只是返回的原代数表达式的表达式。

```
> D[ ](f);
```

$$f$$

4.3.3 隐函数的微分运算

用解析表达式描述 y 对于 x 的函数关系，可以有两种不同的方式。形如 y=f(x)的描述方式，称为显函数(explicit function)。初等函数都可以用显函数方式表示。如果 y 对于 x 的依赖关系是由一个方程式 F(x,y)=0 所确定，则称其为隐函数(implicit function)。在高等数学中，有很多隐函数都不能解出初等表达式，这就产生了一个问题：如何求出 dy/dx 呢？我们很容易想到 Maple 是否提供了专门的函数来处理这种问题，遗憾的是我们在 Maple 中找不到这样的函数。但是我们可以利用现有的函数来解决这种问题。

在高等数学的教材上，一般都是用一元函数的复合函数求导法则来求隐函数的导数。顺着这条思路我们可以想到在 Maple 中用 D 函数来求得隐函数的导数，这时将 x，y 都是当成一个函数，我们必须将 D(x)设成 1，然后再解出那个方程即可。为了说明得更清楚，我们举例说明：

【例 4-4】求解等式 $2x^2-2xy+y^2+x+2y+1=0$ 中 y 的导数，其中 y 是关于 x 的函数。

```
> Eq:='Eq':
> Eq:=2*x^2-2*x*y+y^2+x+2*y+1=0;
```

$$Eq := 2x^2 - 2xy + y^2 + x + 2y + 1 = 0$$

```
> dEq:=D(Eq);
```

$$dEq := 4\mathrm{D}(x)x - 2\mathrm{D}(x)y - 2x\mathrm{D}(y) + 2\mathrm{D}(y)y + \mathrm{D}(x) + 2\mathrm{D}(y) = 0$$

```
> dEq2:=subs(D(x)=1,dEq):
> isolate(dEq2,D(y));
```

$$\mathrm{D}(y) = \frac{-4x + 2y - 1}{-2x + 2y + 2}$$

当然我们也可以用 diff 函数来求解，不过需要额外增加一个过程，那就是用 y(x)代替

原等式中的 y，这是为了是函数 diff 能够将 y 当成是一个函数而不是一个变量，否则用 diff 对 y 求导将得到 0。具体过程如下：

```
> dEq:=diff(Eq,x);
```

$$dEq := 4x - 2y + 1 = 0$$

```
> Eq:='Eq':
> Eq:=2*x^2-2*x*y(x)+y(x)^2+x+2*y(x)+1=0;
```

$$Eq := 2x^2 - 2xy(x) - y(x)^2 + x + 2y(x) + 1 = 0$$

```
> dEq:=diff(Eq,x);
```

$$dEq := 4x - 2y(x) - 2x\left(\frac{\partial}{\partial x} y(x)\right) + 2y(x)\left(\frac{\partial}{\partial x} y(x)\right) + 1 + 2\left(\frac{\partial}{\partial x} y(x)\right) = 0$$

```
> isolate(dEq,diff(y(x),x));
```

$$\frac{\partial}{\partial x}(x) = \frac{-4x + 2y(x) - 1}{-2x + 2y(x) + 2}$$

假如我们没有将 y 写成 y(x)，求导时将出现致命错误：

```
> Eq:='Eq':
> Eq:=2*x^2-2*x*y+y^2+x+2*y+1=0;
```

$$Eq := 2x^2 - 2xy + y^2 + x + 2y + 1 = 0$$

```
> dEq:=diff(Eq,x);
```

$$dEq := 4x - 2y + 1 = 0$$

所以用 diff 求解隐函数的微分时一定要注意待求变量的形式。

在本章中我们已经学习了如何利用 Maple 来求代数表达式的极限和微分，现在我们举一个实际生活中的例子来综合一下所学的东西。

【例 4-5】 已知氢分子离子的反键态能量如下形式：

$$Eg := -\frac{1}{2}k^2 + \frac{k^2 - k - \frac{1}{R} + \frac{(1+kR)\mathrm{e}^{(-2kR)}}{R} - k(k-2)(1+kR)\mathrm{e}^{(-kR)}}{1 - \mathrm{e}^{(-kR)}\left(1 + kR + \frac{1}{3}k^2R^2\right)}$$

（1）证明 Eg 可以写成 $Eg=k^2F(t)+kG(t)$ 的形式，并求出 F(t)和 G(t)，其中 t=kR；

（2）假如已知在 $\frac{\partial}{\partial k} Eg=0$ 情况下导致 $k=\frac{-G(t)-t\left(\frac{\partial}{\partial t}G(t)\right)}{-F(t)+t\left(\frac{\partial}{\partial t}F(t)\right)}$，求解当 t 分别趋于 0 和正无穷时 k 的值。

先来看看第一问题，这是一个初等代数问题，其证明就是一个移项的过程，用笔算很

麻烦，假如用 Maple 则要简单得多。

先输入能量的表达式，并用 t/k 替代 R：

```
> Eg:=-1/2*k^2+(k^2-k-1/R+1/R*(1+k*R)*exp(-2*k*R)-k(k
  -2)(1+k*R)*exp(-k*R))/(1-exp(-k*R)*(1+k*R+k^2*R^2/3
  ));
```

$$Eg := -\frac{1}{2}k^2 + \frac{k^2 - k - \frac{1}{R} + \frac{(1+kR)\mathbf{e}^{(-2kR)}}{R} - k(k-2)(1+kR)\mathbf{e}^{(-kR)}}{1-\mathbf{e}^{(-kR)}\left(1+kR+\frac{1}{3}k^2R^2\right)}$$

```
> subs(R=t/k,Eg);
```

$$k^2 + \left(-\frac{1}{2} + \frac{1}{1-\mathbf{e}^{(-t)}\left(1+t+\frac{1}{3}t^2\right)}\right) + \frac{k\left(-1-\frac{1}{t}+\frac{(1-t)\mathbf{e}^{(-2t)}}{t}-\mathbf{e}^{(-2t)}\right)}{1-\mathbf{e}^{(-t)}\left(1+t+\frac{1}{3}t^2\right)}$$

从上面的等式就可以看出等式能够写成 $k^2F(t)+kG(t)$的形式，为了得到 F(t)和 G(t)，我们可以设 $k^2=1$，k=0 或 $k^2=0$，k=1：

```
> F:=simplify(subs(k^2=1,k=0,%));
```

$$F := -\frac{1}{2}\frac{3+3\mathbf{e}^{(-t)}+3\mathbf{e}^{(-t)}t+\mathbf{e}^{(-t)}t^2}{-3+3\mathbf{e}^{(-t)}+3\mathbf{e}^{(-t)}t+\mathbf{e}^{(-t)}t^2}$$

```
> G:=simplify(subs(k^2=0,k=1,%%));
```

$$G := 3\frac{t+1-\mathbf{e}^{(-2t)}+\mathbf{e}^{(-2t)}t+\mathbf{e}^{(-2t)}t}{t(-3+3\mathbf{e}^{(-t)}+3\mathbf{e}^{(-t)}t+\mathbf{e}^{(-2t)}t^2)}$$

下面来求解第二个问题，首先求 F 和 G 的导数，再用 limit 函数求出 t 在零和正无穷处的值：

```
> F1:=simplify(diff(F,t));
```

$$F1 := -3\frac{\mathbf{e}^{(-t)}t(1+t)}{(-3+3\mathbf{e}^{(-t)}+3\mathbf{e}^{(-t)}t+\mathbf{e}^{(-t)}t^2)^2}$$

```
> G1:=simplify(diff(G,t));
```

$$G1 := 3(3 + 3e^{(-3t)} + 6e^{(-3t)}t^3 + 12e^{(-3t)}t^2 + 9e^{(-3t)}t + e^{(-t)}t^4 + e^{(-3t)}t^4 - 3e^{(-t)} + 3e^{(-t)}t^2 - 3e^{(-t)}t - 3e^{(-2t)} - 6e^{(-2t)}t - 9e^{(-2t)}t^2 - 2e^{(-2t)}t^3 + 2t^3e^{(-t)})/(t^2(-3 + 3e^{(-t)} + 3e^{(-t)}t + e^{(-t)}t^2)^2)$$

```
> limit(-(G+t*G1)/(2*F+t*F),t=0);
```

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

```
> limit(-(G+t*G1)/(2*F+t*F),t=infinity);
```

$$0$$

用 Maple 很容易就完成了原本需要花好几个小时的工作，从中也可以看出 Maple 的强大功能。

4.4 不定积分与定积分

积分是微分的逆变换，在高等数学中占有举足轻重的地位，在 Maple 中有一个专门的函数 int 或 Int 被用来处理这种问题。

4.4.1 不定积分

计算不定积分时函数 int(或 Int)的具体表达式如下：

```
int(expr, x);
Int(expr, x);
```

expr 是待求积分的代数表达式，x 是积分变量，与本章前面介绍的函数类似，函数 int 和 Int 的返回值略有不同，int 返回运算以后的结果，而 Int 只是返回一个表达式。下面我们来看几个例子：

```
> int(1/(x^2+x),x);
```

$$\ln(x) - \ln(x+1)$$

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

$$x\ln(x) - x$$

从上面的两个例子来看，函数 int 求不定积分时并不像我们平时计算时在结果后面加上一个常数 C。上面是用 int 来运算，我们也可以用 Int 来进行运算，不过应该在 Int 语句后面再加一条语句：value(%)；将返回的表达式求出值：

```
> Int(1/(x^2+x),x):
> %=value(%);
```

$$\int \frac{1}{x^2+x}\,dx = \ln(x) - \ln(x+1)$$

用求微分函数可以验证一下积分的正确性：

```
> simplify(diff(ln(x)-ln(x+1),x));
```

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

很明显结果与我们原先的代数表达式是完全相同的，这说明了我们的求解是正确的，下面我们再看看定积分。

4.4.2 定积分

求定积分与求不定积分的函数相同，只是参数比较多，具体表达式为：

int(f,x=a..b);
Int(f,x=a..b);

f 是待积分的代数表达式，x 是自变量名，a 和 b 为积分的上下限。

下面是几个常见的定积分例子：

```
> int(sin(x),x=0..Pi);
```

$$2$$

```
> Int(x,x=-10..10):
> value(%);
```

$$0$$

```
> int(Dirac(x),x=-infinity..infinity);
```

$$1$$

```
> f(x):=sqrt(1+x^2):
> int(f(x),x);
```

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

```
> assume(n,integer):
```

```
> simplify(int(x^n*exp(-x^2),x=-infinity..infinity));
```

$$\frac{1}{2}\Gamma\left(\frac{1}{2}n+\frac{1}{2}\right)(1+(-1)^n)$$

4.4.3 多重积分

多重积分就是对一个多元的代数表达式的多个自变量进行积分，在 Maple 中只提供专门的函数来处理二重积分（Doubleint）和三重积分(Tripleint)。但从多重积分的性质和我们平时运算多重积分的算法来看，我们也可以多次利用一元函数的定积分来求得，比如：

```
> int(int(x/y,x=1..2),y=1..2);
```

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

```
> int(int(x/y,y=1..2),x=1..2);
```

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

二重积分 Doubleint 有三种函数表达式，分别代表三种情况，下面分别介绍这三种用法：

1. 二重函数的不定积分

Doublieint(f(x,y),x,y);

f(x,y)是待积的代数表达式，x 和 y 都是自变量，用法很简单，但有一点需要注意，由于函数 Doubleint 不在内部函数库里，因此在使用的时候一定要先调用 student 程序包（这个程序包里包含了 Doubleint 函数和后面将用到的 Tripleint 函数，以及其他一些函数）。

```
> with(student):
> Doubleint(x/y,x,y);
```

$$\iint\frac{x}{y}\,dx\,dy$$

```
> value(%);
```

$$\frac{1}{2}x^2\ln(y)$$

从上面的例子可以看出 Doubleint 函数并不直接给出积分结果，必须调用 value 函数来附加计算。

2. 在区域内的积分

Doubleint(f(x,y),x,y,D);

f(x,y)是待积分的代数表达式，x，y 是积分变量，D 是积分区域。

```
> Doubleint(x/y,x,y,D);
```

$$\iint_D \frac{x}{y}\,dx\,dy$$

这个函数只是返回一个表达式，且不能进行运算，为了得到某一个区域内的积分的结果，我们需要利用下面二重定积分的性质。

3. 二重函数的定积分

Doubleint(f(x,y),x=a..b,y=m..n);

f(x,y)是待求积分的代数表达式，x，y 是积分变量，a，b 和 m，n 分别是变量 x，y 积分的上下限。先来看一个 x，y 的积分上下限彼此无关的例子：

```
> with(student):
> Doubleint(x/y,x=-1..1,y=-1..1);
```

$$\int_{-1}^{1}\int_{-1}^{1} \frac{x}{y}\,dx\,dy$$

```
> value(%);
```

$$0$$

这种情况很简单，但是我们通常情况下都会碰到某一个区域内的积分，这时 y 的积分上下限中包含 x，或 x 的积分上下限中包含 y。这个问题看起来比较复杂，在 Maple 中的处理还是比较简单的，与我们平时笔算的过程差不多，都是将某一个含变量积分上下限的变量先积分，最后积分上下限中不带变量的那个变量。比如我们要将函数 f(x,y)在区域 $x^2+y^2<1$ 内积分，我们所选的策略是将 y 在$[-(1-x^2)^{1/2}, (1-x^2)^{1/2}]$，x 在[-1，1]上积分：

```
> with(student):
  f(x,y):=(x/y)^2:
> Doubleint(f(x,y),y=1-sqrt(1-x^2)..1+sqrt(1-x^2),
  x=-1..1);
```

$$\int_{-1}^{1}\int_{1-\sqrt{1-x^2}}^{1+\sqrt{1-x^2}} \frac{x^2}{y^2}\,dy\,dx$$

```
> value(%);
```

$$\pi$$

在计算这种定积分的时候千万要注意，变量积分的顺序是固定的，如果先对变量 y 积分就应该将变量 y 写在第一个，不要将积分顺序颠倒，否则将得不到正确答案，比如我们将 x 和 y 的积分位置互换：

```
> value(Doubleint(f(x,y),x=-1..1,y=1-sqrt(1-x^2)..
  1+sqrt(1-x^2)));
```

$$-\frac{4}{3}\frac{\sqrt{1-x^2}}{(1+\sqrt{1-x^2})(-1+\sqrt{1-x^2})}$$

上面并没有给出我们期望得到的值，从这点也可以看出 Doubleint 其实是利用我们本节刚开始介绍的方法进行的。

有了二重积分的基础，三重积分的使用就很简单了，除了多一个参数外，其它没什么区别，只是在计算定积分时可能需要点变化。

```
> with(student):
> Doubleint(x/y,x=-1..1,y=-1..1);
```

$$\int_{-1}^{1}\int_{-1}^{1}\frac{x}{y}\,dx\,dy$$

```
> value(%);
```

$$0$$

```
> Doubleint(x/y,x,y,D);
```

$$\iint_{D}\frac{x}{y}\,dx\,dy$$

```
> with(student):
  f(x,y,z):=x^2*y^2*z^2;
```

$$f(x,y,z):=x^2y^2z^2$$

```
> Tripleint(f(x,y,z),x,y,z);
```

$$\iiint x^2y^2z^2\,dx\,dy\,dz$$

```
> value(%);
```

$$\frac{1}{27}x^3y^3z^3$$

```
> Tripleint(f(x,y,z),x,y,z,D);
```

$$\iiint_{D} x^2y^2z^2\,dx\,dy\,dz$$

最后我们来看一个 f(x,y,z)在 $x^2+y^2+z^2<1$ 区域上的积分，其积分方法与上面所介绍的在区域 $x^2+y^2<1$ 上的二重积分相同。

```
> Tripleint(x^2*y^2*z^2,x=-sqrt(1-y^2-z^2)..
  sqrt(1-y^2-z^2),y=-sqrt(1-z^2)..sqrt(1-z^2
  ),z=-1..1);
```

$$\int_{-1}^{1}\int_{-\sqrt{1-z^2}}^{\sqrt{1-z^2}}\int_{-\sqrt{1-y^2-z^2}}^{\sqrt{1-y^2-z^2}} x^2y^2z^2\,dx\,dy\,dz$$

```
> value(%);
```

$$\frac{4}{945}\pi$$

4.4.4 变量代换法和分部积分法

Maple 能直接解决很多积分问题，但这还不是它的全部。在很多情况下 Maple 还能帮我们解决变量代换和分步积分的问题。

1. 变量代换法

在前面我们曾经使用过函数 subs 来替换代数表达式中的某些变量，现在我们使用函数 changevar 来进行积分中的变量代换。这个函数不仅仅是对变量进行替换，而且能对积分的变量形式进行替换。函数 changevar 的具体表达形式如下：

changevar(s, f);
changevar(s, f, u);
changevar(t, g, v);

s 是一个等式表达式，形式如：h(x) = g(u)，x 是变量 u 的函数；f 是形如 Int(F(x), x = a..b) 的代数表达式；u 是新的积分变量的名称；t 是定义了多个变量的等式的集合，比如 h(x)=g(u,t) 这个参数可以是多个等式，利用大括号{}包含起来就行，这种主要是在多重积分时要使用到；g 是双重或三重积分；v 跟 u 类似，是新的积分变量，不过 v 可以是多个变量的列表。现在我们举例说明。

（1）用 u 替代 $\cos x^3$ 运算 $\int x^2 \sin x^3 \cos x^3\,dx$。

```
> f:='f':
> f:=x^2*sin(x^3)*cos(cos(x^3)):
> tocompute:=Int(f,x);
```

$$tocompute := \int x^2 \sin(x^3)\cos(\cos(x^3))\,dx$$

```
> with(student):
```

```
> aftersurb:=changevar(cos(x^3)=u,tocompute);
```

$$aftersurb := \int -\frac{1}{3}\cos(u)\,du$$

现在我们成功的进行了替代，假如要计算出结果并且将其还原为自变量为 x 的形式，则只需调用函数 subs 即可。

```
> subs(u=cos(x^3),value(aftersurb));
```

$$-\frac{1}{3}\sin(\cos(x^3))$$

上面是一个不定积分的例子，假如是定积分又会如何呢？从实例可以看出，我们使用 Maple 是幸运的，函数 changevar 不光将积分变量进行了替换，在定积分中它还会自动替换积分的上下限。

（2）用 u 替换 5+cosx 来计算 $\int_{\frac{\pi}{2}}^{\pi} \frac{sinx}{(5+cosx)^2}\,dx$。

首先我们定义积分形式：

```
> f:='f':
> f:=x->sin(x)/(5+cos(x))^2:
> tocompute:=Int(f(x),x=Pi/2..Pi);
```

$$tocompute := \int_{\frac{1}{2}\pi}^{\pi} \frac{\sin(x)}{(5+\cos(x))^2}\,dx$$

然后我们调用函数 changevar 进行变量代换，最后进行积分。

```
> with(student):
> aftersurb:=changevar(5+cos(x)=u,tocompute);
```

$$aftersurb := \int_4^5 \frac{1}{u^2}\,du$$

```
> value(aftersurb);
```

$$\frac{1}{20}$$

最后让我们来看一个多重积分变量代换的例子。

（3）用 x=rsint，y=rcost 的变换来计算积分：$\iint x^2+y^2\,dx\,dy$。

```
> f:='f':
> f:=(x,y)->x^2+y^2:
> with(student):
```

```
> tocompute:=Doubleint(f(x,y),x,y);
```

$$tocompute := \int\int x^2 + y^2 \, dx \, dy$$

```
> aftersub:=changevar({x=r*sin(t),y=r*cos(t)}
  ,tocompute,[t,r]);
```

$$aftersub := \int\int r^2 \, |r| \, dt \, dr$$

在多重积分的变量代换中，参数 v 是必选的，否则 Maple 将不知道哪些是积分变量，哪些是常量。

2. 分步积分法

假如我们想知道某一个积分的分步积分过程是什么样的，最直接的方法就是利用手中的笔进行运算，利用我们所学的公式推导一番，这种方法在积分比简单的情况也许是一个比较不错的主意。但是假如积分很复杂，笔算就有可能错误百出，利用 Maple 提供的 intparts 函数就能很好的解决这个问题。这个函数只有一个表达形式，使用起来也很方便，它的具体表达形式如下：

intparts(f, u);

f 是一个形如 Int(u*dv, x)的代数表达式，u 是在分步积分时待微分的代数表达式，为了有一个比较形象的了解我们还是来看一个例子。

【例 4-6】 用分步积分法来求积分：$\int e^{(ax)} \cos(bx) dx$，其中 a 和 b 都是正数。

首先是输入设定 a 和 b 都是正数，然后再输入积分表达式，为了保证 f 是重新定义的，我们在最前面将它本身赋值给 f。

```
> f:='f';
> assume(a,positive):
> assume(b,positive):
> f:=x->exp(a*x)*sin(b*x):
> tocompute:=Int(f(x),x);
```

$$tocompute := \int e^{(a \sim x)} \sin(b \sim x) \, dx$$

第二步我们用 intparts 函数来分步积分，由于这个函数不属于 Maple 的内部函数，而是在 student 程序包里，因此使用之前我们调用了函数 with(student)。

```
> with(student):
> stepone:=intparts(tocompute,exp(a*x));
```

$$stepone := -\frac{e^{(a \sim x)} \cos(b \sim x)}{b \sim} - \int -\frac{a \sim e^{(a \sim x)} \cos(b \sim x)}{b \sim} dx$$

```
> steptwo:=simplify(intparts(stepone,exp(a*x)));
```

$$steptwo := -\frac{e^{(a\sim x)}\cos(b\sim x)b\sim - e^{(a\sim x)}a\sim \sin(b\sim x) + a\sim^2 \int e^{(a\sim x)}\sin(b\sim x)dx}{b\sim^2}$$

在做完第二步分步积分后发现，在结果中包含有我们刚开始所要求的积分表达式，这样我们利用 isolate 函数就能很容易的得到积分的值。

```
> isolate(tocompute=steptwo, tocompute)
```

$$\int e^{(a\sim x)}\sin(b\sim x)dx = \frac{-e^{(a\sim x)}\cos(b\sim x)b\sim + e^{(a\sim x)}a\sim \sin(b\sim x)}{b\sim^2 + a\sim^2}$$

最后我们用直接积分来验证一下上面的结果，很明显答案是正确的。

```
> simplify(value(Int(f(x),x)));
```

$$\frac{e^{(a\sim x)}(-b\sim\cos(b\sim x) + a\sim\sin(b\sim x))}{b\sim^2 + a\sim^2}$$

4.5 积分变换与特殊函数

积分变换在高等数学中经常碰到，比如量子力学波函数在坐标空间和动量空间的互换、光谱的分解等等。在本节中我们将来看看如何利用 Maple 来实现 Laplace 变换、Fourier 变换、快速 Fourier 变换以及它们的逆变换。除此以外，我们在本节还将介绍一些在工程中经常用到的特殊函数的积分。

4.5.1 Laplace 变换及其逆变换

Laplace 变换是将函数 f(t)变换成函数 $F(x)=\int_0^\infty f(t)e^{(-tx)}dt$ 的一种变换。在 Maple 中进行此变换的函数是 laplace，它的具体表达形式如下：

```
laplace(expr, t, s);
```

其中 expr 是待变换的代数表达式，t 是将被变换掉的自变量，s 是变换后的自变量。这个函数不属于 Maple 的内部函数，它存在于 inttrans 程序包中，因此使用这个函数的时候应先调用函数 with(inttrans)。下面我们先来看一个例子：

```
> with(inttrans):
```

```
> laplace(f(t),t,s);
```

$$\text{laplace}(f(t), t, s)$$

函数并没有像我们希望的那样给出积分的形式，这让我们很沮丧。但不要着急，回想以前用过的 convert 函数，现在看看有什么效果。

```
> convert(%,int);
```

$$\int_0^{\infty} f(t)e^{(-ts)}dt$$

现在我们得到了想要的结果，令人满意。下面我们再来看一个具体的例子，这个例子是将函数 exp(-at)进行 Laplace 变换。

```
> f:='f':
> f:=exp(-a*t):
> with(inttrans):
> convert(laplace(f,t,s),int);
```

$$\frac{1}{s+a}$$

Laplace 变换的逆变换叫做逆 Laplace 变换，在 Maple 中的函数是 invlaplace，其具体表达式如下：

invlaplace(expr, s, t);

其中 expr 是带变换的代数表达式，s 是将被变换掉的自变量，t 是变换后的变量。下面我们来看几个例子。

```
> with(inttrans):
> invlaplace(1/(s+a),s,t);
```

$$e^{(-at)}$$

```
> invlaplace(s^2/(s^2+a^2)^(3/2), s, t);
```

$$\text{BesselJ}(0,at) - t\,\text{BesselJ}(1,at)a$$

```
> invlaplace(s/(s-1)*laplace(F(t), t, s), s, t);
```

$$F(t) + \int_0^t F(\_U1)e^{(t-\_U1)}d\_U1$$

第一个例子变换的是上面 Laplace 变换的结果，得到的结果与我们所给的初值是吻合的。

4.5.2 Fourier 变换及其逆变换

Fourier 变换是将函数 f(t)变换为函数：$F(w)=\int_0^{\infty} f(t)e^{(-Itw)}dt$ 的一种变换，它在波函数坐标空间和动量空间的变换中经常用到。在 Maple 中处理这个变换的是函数 Fourier，它的具体函数表达式如下：

fourier(expr, t, w);

其中 expr 是待变换的代数表达式，t 是将被变换掉的变量，w 是变换后的变量。与函数 laplace 一样，这个函数也存在于 inttrans 程序包中。下面我们先来看一个普遍的例子。

```
> with(inttrans):
> f:='f':
> fourier(f(t), t, w);
```

$$\text{fourier}(f(t), t, w)$$

```
> convert(%,int);
```

$$\int_{-\infty}^{\infty} f(t)e^{(-Itw)}dx$$

从上面可以看出 fourier 函数与 laplace 函数很相像，都不直接给出积分的结果，接下来再来看几个具体的例子。

```
> fourier(exp(I*b*t),t,w);
```

$$2\pi\,\text{Dirac}(-w+b)$$

```
> assume(a>0):
  fourier(3/(a^2+t^2),t,w);
```

$$3\frac{e^{(a\sim w)}\pi\,\text{Heaviside}(-w)}{a\sim}+\frac{3e^{(-a\sim w)}\pi\,\text{Heaviside}(w)}{a\sim}$$

跟 laplace 变换一样，fourier 变换也有逆变换，其函数是 invfourier，具体的函数表达式如下：

invfourier(expr, w, t);

其中 expr 是待变换的代数表达式，w 是将被变换掉的变量，t 是变换后的变量。我们还是用一个具体的例子来看看它的用法，这个例子是将上面 fourier 变换得到的一个结果进行逆变换。

```
> invfourier(2*Pi*Dirac(-w+b),w,t);
```

$$e^{(Ibt)}$$

4.5.3 快速 Fourier 变换及其逆变换

在信号分析与处理中经常要用到离散 Fourier 变换，为了减少运算量，能够对信号进行实时分析，就要用到快速 Fourier 变换及其逆变换。在 Maple 中有专门的函数来处理这类事务。它们分别是 FFT 和 iFFT。这两个函数的具体表达式为：

FFT(m,x,y);
iFFT(m,x,y);

其中 m 是一个非负整数，用来指示变换的维数（2^m 维）；x 和 y 都是 2^m*1 的矩阵，x 用来保存输入部分的实数部分的序列以及 Fourier 变换（或其逆变换）后输出的实数部分的序列，而 y 用来保存输入部分的虚数部分的序列以及 Fourier 变换（或其逆变换）后输出的虚数部分的序列。

这两个函数运行完以后不直接给出变换的结果，而是给出了变换的维数（2^m），为了得到变换的结果，我们可以用 print 函数来实现，请看下面的结果：

```
> x := array([7.,5.,6.,9.]):
  y := array([0,0,0,0]):
  FFT(2,x,y);
```

4

```
> print(x);
```

[27.,1.,-1.,1.]

```
> print(y);
```

[0.,4.,-0.,-4.]

我们用 FFT 实现了长度为 $2^2=4$ 的函数的 Fourier 变换，下面我们用 iFFT 对上面得到的结果再进行一次变换：

```
> iFFT(2,x,y);
```

4

```
> print(x);
```

[7.000000000,5.000000000,6.00000000,9.000000000]

```
> print(y);
```

[0.,0.,0.,0.]

很明显得到的结果就是我们原先输入的结果。为了有一个更深刻的理解，下面再来看一个 m=3，且虚部（即 y 的元素）不都为零的例子。

```
> x := array([1.,2.,3.,4.,5.,6.,7.,8.]):
  y := array([8.,7.,6.,5.,4.,3.,2.,1.]):
  FFT(3,x,y);
```

$$8$$

```
> print(x);
```

[36.,5.656854242,0.,−2.343145750,−4.,−5.656854246,−8.,−13.65685425]

```
> print(y);
```

[36.,13.65685425,8.,5.656854264,4.,2.343145750,0.,−5.656854242]

```
> iFFT(3,x,y);
```

$$8$$

```
> print(x);
```

[1.000000000,2.000000001,3.000000000,4.000000002,5.000000000,
5.000000000,7.000000000,7.999999998]

```
> print(y);
```

[8.000000000,7.000000000,6.000000000,4.999999996,4.000000000,
3.000000000,2.000000000,1.000000004]

4.5.4　三角积分函数和双曲积分函数

三角函数、双曲函数、对数函数以及指数函数在实际的应用中极为常见，我们经常要对它们进行求导和积分的处理，本节我们用对三角函数和双曲函数求积分来巩固一下积分的知识。一般来说这些函数的积分结果都比较复杂，一般不是初等的代数表达式．下面我们先通过几个例子来看看三角函数的积分性质。

```
> int(sin(x)*cos(x),x);
```

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

```
> int(tan(x^2),x);
```

$$-Ix-I\int -2\frac{1}{\left(e^{(Ix^2)}\right)^2+1}dx$$

```
> int(cot(x),x=0..1);
```

$$\infty$$

```
> int(sin(x^3),x=1..2);
```

$$\int_1^2 \sin(x^3)dx$$

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

$$.2181032316$$

第一个例子很简单，类似的积分在前面已经碰到过，第二个例子的不定积分的结果很复杂，Maple 不能直接给出初等代数表达式的结果，而只能用一个比较简单的积分式子来表示。第三个例子是一个定积分，从形式上就可以看出结果应该是正无穷；从运算过程来看，$\sinh x^3$ 的积分没有普遍的形式，Maple 是利用极限求和的方法来得到最后结果的。

```
> int(sinh(x^2),x);
```

$$-4I\sqrt{\pi}\,\mathrm{erf}(Ix)-\frac{1}{4}\sqrt{\pi}\,\mathrm{erf}(x)$$

```
> int(x^2*csch(x^2),x=0..infinity);
```

$$\int_0^{\infty} x^2\mathrm{csch}(x^2)\,dx$$

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

$$1.496625634$$

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

$$x\,\mathrm{arcsinh}(x)-\sqrt{x^2+1}$$

4.5.5 对数积分和指数积分

对数积分在工程运用中比三角函数的积分还常见，比如 $x^n e^{-x^2}$ 的积分随处可见，下面我们就来看看这个积分。

```
> assume(n,integer):
> int(x^n*exp(-x^2),x=0..infinity);
```

$$\frac{1}{2}\Gamma\left(\frac{1}{2}n\sim+\frac{1}{2}\right)$$

最后我们来看两个简单的对数积分。

```
> int(ln(x^2)/x,x);
```

$$\frac{1}{4}\ln(x^2)^2$$

```
> int(x^3*ln(x^3-2*x^2-3),x);
```

$$\frac{1}{4}x^4\ln(x^3-2x^3+3)-\frac{3}{16}x^4-\frac{1}{6}x^3-\frac{1}{2}x^2+\frac{1}{4}x-\frac{1}{4}\ln(x+1)$$
$$+\frac{9}{8}\ln(x^2-3x+3)+\frac{9}{4}\sqrt{3}\arctan\left(\frac{1}{3}(2x-3)\sqrt{3}\right)$$

4.6 数值积分

在前面关于积分的介绍中，我们曾经碰到过很多不能给出普遍解的积分表达式，但在工程运算中，我们总是希望得到一个可以应用的数值，这就要求我们能够进行数值积分。其实在前面我们已经碰到过用 evalf 来求出定积分的近似数值解，下面我们再来看一个例子：求代数表达式$(4x^2-9)^{1/2}/x^3$在区间[4,10]之间的积分值。

按照我们前面的计算方法是先将上面的式子用 int 函数进行积分运算：

```
> int(sqrt(4*x^2-9)/(x^3),x=4..10);
```

$$-\frac{1}{200}\sqrt{391}-\frac{2}{3}\arctan\left(\frac{3}{391}\sqrt{391}\right)+\frac{1}{32}\sqrt{55}+\frac{2}{3}\arctan\left(\frac{3}{55}\sqrt{55}\right)$$

这是一个很复杂的式子，要得到实用的值，我们还必须用 evalf 来得到近似的数值解。

```
> evalf(%);
                    .2887732708
```

在 Maple 中除了 int 函数可以求数值积分外，还提供了很多其他函数来处理数值积分，这些函数跟 int 不一样，它们是专门用来进行数值积分的，下面我们来详细介绍这些函数，同时对它们的结果进行比较。

1. leftsum 函数

顾名思义，函数 leftsum 是将代数表达式分割成很多矩阵，使矩阵的左顶点的值等于函数在该点的值，然后将这些矩阵的面积（如果矩阵在 x 轴下方，则面积为负）相加而得到数值积分的值。Maple 提供了一个函数 leftbox 用图形方式来显示积分的规则，使我们能够清楚的看到这个函数是如何进行运算的。

函数 leftsum 的具体的函数表达式如下：

leftsum(f(x),x=a..b)

leftsum(f(x), x=a..b, n)

其中 f(x)包含一个自变量 x 的代数表达式，x 是积分变量，a 和 b 分别是数值积分的上

下限，n 是一个可选的变量，用来设置系统运算时使用矩形的个数，默认值为 4。如果待积的代数表达式是一个参数，其将返回一个积分规则的代数表示。

函数 leftbox 的具体函数表达式如下：

leftbox(f(x), x=a..b, <plot options>)

leftbox(f(x), x=a..b, n, 'shading'=<color>, <plot options>)

前面的几个参数的含义与函数 leftsum 的相同，倒数第二个参数用来设置填充矩阵的颜色，最后一个参数则是设置绘画待积的函数表达式的曲线的性质（包括颜色），具体的设置请参看关于绘图的那一章。

下面我们用这两个函数来做一个积分。

【例 4-7】 求函数 $x^4\ln x$ 在区间[2,4]的数值积分值。

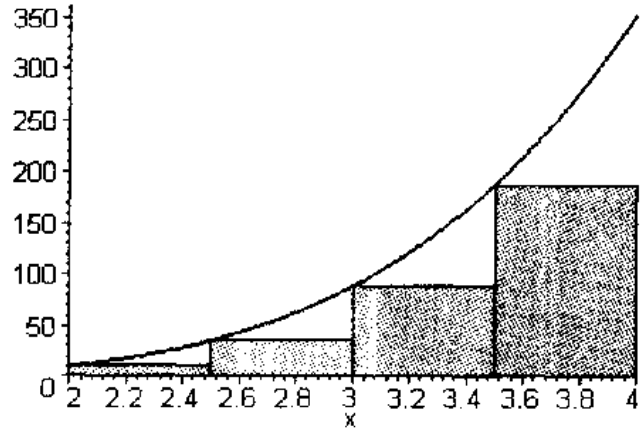
首先我们先不设被积函数的具体表达式，看看这种数值积分规则的代数表达式是什么样的，由于这两个函数都不是 Maple 的内部函数，而是在 student 程序包里，因此我们在最开始先包含这个程序包。

```
> with(student):
  f:='f':leftsum(f,x=a..b);
```

$$\left(\frac{1}{4}b-\frac{1}{4}a\right)\left(\sum_{i=0}^{3} f\right)$$

然后我们用 leftbox 函数来显示这种积分的积分规则，首先我们用默认的矩阵个数(4个)。

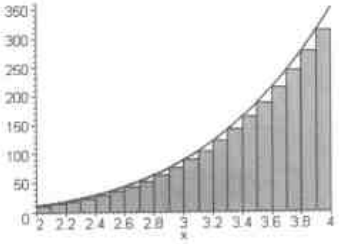
```
> f:=x->x^4*ln(x):
> leftbox(f(x),x=2..4);
```



最后我们用函数 leftsum 来看看最后的积分结果是多少：

```
> evalf(leftsum(f(x),x=2..4));
                              161.9316500
```

我们使用默认的积分矩阵个数（4 个）的时候得到了积分结果 161.9316500。但从上面的那个图中可以看出这种结果是很粗略的，为了得到较精确的结果就必须增加积分的矩阵个数，下面我们将矩阵个数分别设成 20，100，1000 和 10000，看看结果有什么变化，同时也是为了将这个积分方法与后面介绍的方法进行比较，在矩阵个数是 20 的时候，我们还



，其他三个就不画出图形了。

```
(f(x),x=2..4,20));
                              222.9307976
(f(x),x=2..4,100));
                              236.3718902
> evalf(leftsum(f(x),x=2..4,1000));
                              236.4532718
> evalf(leftsum(f(x),x=2..4,10000));
                              239.7625644
> leftbox(f(x),x=2..4,20));
```

用这种方法进行运算时，积分矩阵个数的设置对最后的结果还是有很大的影响的，显然积分矩阵个数越大，结果也越准确，因此假如要得到较精确的数值，建议设置积分矩阵个数而不使用系统的默认值。

2. middlesum 函数

从函数名就可以看出这个函数的积分方法是跟函数 leftsum 是相似的，只是选取矩阵时是使矩阵顶边的中点与待积代数表达式在该点的值相等。相应的，Maple 也提供了 middlebox

函数来显示积分矩阵的选取，这两个函数的具体表达式如下：

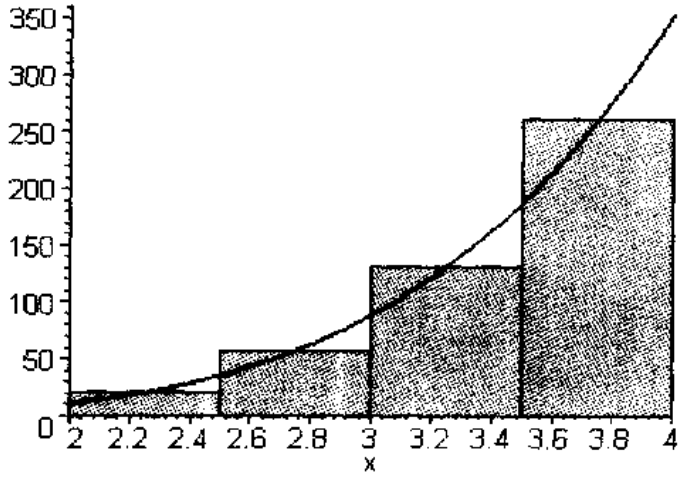
middlesum(f(x),x=a..b)
middlesum(f(x), x=a..b, n)
middlebox(f(x), x=a..b, <plot options>)
middlebox(f(x), x=a..b, n, 'shading'=<color>, <plot options>)

很明显这些参数跟对应的 leftsum 和 leftbox 的参数是完全一样的，其含义我们就不一一介绍了。这里只是用实际的例子来看看这两个函数的规则和功能，这个例子就是上一小节的例子 $x^4\ln x$ 在[2, 4]的积分值，在这之前我们先不给函数 f 赋值，而是调用函数 middlesum 看看它的积分规则：

```
> with(student):
  f:='f':
  f:=x->f(x):
  middlesum(f(x),x=a..b);
```

$$\left(\frac{1}{4}b-\frac{1}{4}a\right)\left(\sum_{i=0}^{3}f\left(a+\left(i+\frac{1}{2}\right)\left(\frac{1}{4}b-\frac{1}{4}a\right)\right)\right)$$

```
> f:='f':  f:=x->x^4*ln(x):
> middlebox(f(x),x=2..4);
```




```
> evalf(middlesum(f(x),x=2..4));
```

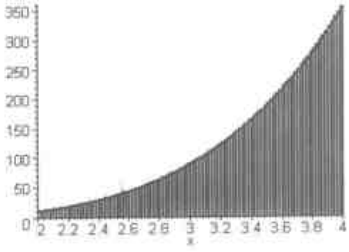
235.7594079

```
> evalf(middlesum(f(x),x=2..4,100));
```

239.7904648

```
> evalf(middlesum(f(x),x=2..4,1000));
                              239.7968784
> evalf(middlesum(f(x),x=2..4,10000));
                              239.7969426
```

与 leftsum 的结果比较，可以发现在相同的积分矩阵个数的情况下，这种方法得到结果要相对精确，用这种方法可以用较小的积分矩阵得到较精确的值，但这也不是绝对的，哪个方法更准确跟待求积分的代数表达式的单调性有关。



sum 函数，顺理成章就应该有这个函数，同样也应该有 rightbox 这两个函数，它们的具体函数表达式也是类似的：

ghtsum(f(x),x=a..b);
..ghtsum(f(x), x=a..b, n);
rightbox(f(x), x=a..b, <plot options>);
rightbox(f(x), x=a..b, n, 'shading'=<color>, <plot options>);

还是用上面例子中的那个代数表达式进行积分，结果如下：

```
> with(student):
> f:='f':
> f:=x->x^4*ln(x):
> rightbox(f(x),x=2..4,100);
```

```
> evalf(rightsum(f(x),x=2..4));
                              333.8321508
> evalf(rightsum(f(x),x=2..4,100));
                              243.2479102
```

```
> evalf(rightsum(f(x),x=2..4,1000));
                              240.1408738
> evalf(rightsum(f(x),x=2..4,10000));
                              239.8313246
```

从上面三种类似的数值积分方法的结果来看，middlesum 是效果最好的，而 leftsum 和 rightsum 则各有千秋。除了这三种常见的积分方法外，Maple 还提供了另外两个专门的数值积分函数，下面我们将较为具体的介绍这两个函数。

4．simpson 函数

此函数在计算数值积分时利用 Simpson 规则进行运算，它的具体函数表达式如下：

simpson(f(x),x=a..b);
simpson(f(x), x=a..b, n);

f(x)是待积的代数表达式，x 是积分变量，a 和 b 分别是积分上下限，n 是函数在积分过程中使用矩阵的个数，这是一个可选量，默认值为 4。当待积分的代数表达式是符号参数时，函数将返回一个运算规则：

```
> with(student):
  f:='f':
  simpson(f,x=a..b,4);
```

$$\frac{1}{3}\left(\frac{1}{4}b-\frac{1}{4}a\right)\left(2f+4\left(\sum_{i=1}^{2}f\right)+2\left(\sum_{i=1}^{1}f\right)\right)$$

由于函数 simpson 不是 Maple 的内部函数，而是在 student 程序包中，所以我们在开头先包含 student 程序包。从结果中我们可以看出函数是如何运算的。现在我们设置一下运算矩阵的个数，看看结果会有什么影响：

```
> with(student):
  f:='f':
  simpson(f,x=a..b,10);
```

$$\frac{1}{3}\left(\frac{1}{10}b-\frac{1}{10}a\right)\left(2f+4\left(\sum_{i=1}^{5}f\right)+2\left(\sum_{i=1}^{4}f\right)\right)$$

从上面可以看出当运算矩阵个数增加时，运算量也将增加，当然结果的精确度也将得到提高。下面来看一个具体的例子，还是利用上面的那个代数表达式进行积分：

```
> with(student):
  f:='f':f:=x->x^4*ln(x):
  simpson(f(x),x=2..4,4);
```

$$\frac{8}{3}\ln(2)+\frac{128}{3}\ln(4)+\frac{2}{3}\left(\sum_{i=1}^{2}\left(\frac{3}{2}+i\right)^4\ln\left(\frac{3}{2}+i\right)\right)+\frac{1}{3}\left(\left(\sum_{i=1}^{1}(2+i)^4\ln(2+i)\right)\right)$$

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

239.8497168

最后我们将运算矩阵设成20，看看最后的数值有什么不同：

```
> evalf(simpson(f(x),x=2..4,20));
```

239.7970276

从上面可以看出使用不同运算矩阵个数，其结果还是有点出入的，假如我们需要比较精确的答案，建议自己设置矩阵个数，而不是利用系统的默认值。

5. trapezoid 函数

trapezoid 是一个利用 Trapezoidal 规则来进行数值积分的函数，其具体的函数表达式如下：

trapezoid(f(x), x=a..b);
trapezoid(f(x), x=a..b, n);

很明显这个函数的格式与上面那几个函数的格式是完全相同的，并且参数的含义也是一样的，这里就不再详细介绍了，我们还是先不设置被积函数的具体代数表达式，来看看这种积分的规则是什么样的：

```
> with(student):
  f:='f':
  trapezoid(f,x=a..b);
```

$$\frac{1}{2}\left(\frac{1}{4}b-\frac{1}{4}a\right)\left(2f+2\left(\sum_{i=1}^{3}f\right)\right)$$

这个函数使用的积分规则比 simpson 要简单，但比前三个复杂。下面我们还是来利用上面那个代数表达式的数值积分来了解这个函数，先使用系统默认的运算矩阵个数（4个）：

```
> with(student):
  f:='f':
  f:=x->x^4*ln(x):
  trapezoid(f(x),x=2..4);
```

$$4\ln(2)+\frac{1}{2}\left(\sum_{i=1}^{3}\left(2+\frac{1}{2}i\right)^4\ln\left(2+\frac{1}{2}i\right)\right)+64\ln(4)$$

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

247.8819004

显然这个结果跟实际结果差别比较大，为了提高精度，我们提高运算矩阵个数试一试：

```
> evalf(trapezoid(f(x),x=2..4,20));
```

240.1208476

```
> evalf(trapezoid(f(x),x=2..4,100));
```

239.8099001

```
> evalf(trapezoid(f(x),x=2..4,1000));
```

239.7970728

```
> evalf(trapezoid(f(x),x=2..4,10000));
```

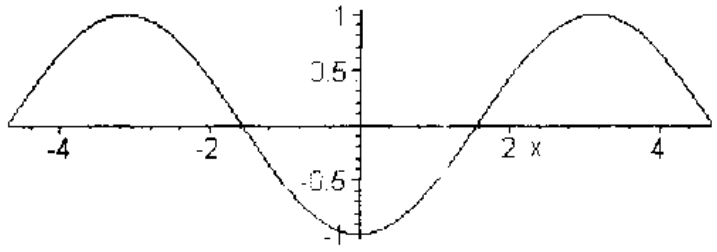
239.7969445

从上面的结果可以看出，当运算矩阵大于 100 时，结果就相差不大了。看上面 5 个函数的结果，我们能够大致看出在相同的运算矩阵的情况下，simpson 的运算精度是最高的，trapezoid 次之，其他三个差不多。但由于被积的函数在积分空间内是单调递增的，不太有说服力，我们下面再来看一个周期震荡的函数的数值积分值，来比较这 5 个函数以及直接积分（用 int 函数和 evalf 函数）的结果哪个精度更高。

【例 4-8】求函数-cosx 在区间[-3/2 π, 3/2 π]上的积分。

为了有一个直观的理解，我们先用画图函数画出这个函数的图形：

```
> f:='f':
  f:x->-cos(x):
  plot(f(x),x=-3/2*Pi..3/2*Pi);
```



然后我们用 int 函数进行不定积分：

```
> int(f(x),x);
                              -sin(x)
```

这是一个能得到普遍积分结果的代数表达式，很显然利用这个结果我们能得到准确的结果：

```
> -sin(3/2*Pi)-(-sin(-3/2*Pi));
                               2
```

有了这个准确的结果，我们就能将后面的不同方法计算得到的结果进行比较了，下面我们用定积分进行运算：

```
> int(f(x),x=-3/2*Pi..3/2*Pi);
                               2
```

显然这个积分也能得到准确值，下面我们再用系统默认的运算矩阵个数调用上面五个函数分别进行计算：

```
> with(student):
  evalf(leftsum(f(x),x=-3/2*Pi..3/2*Pi));
                           .9759677138
> evalf(middlesum(f(x),x=-3/2*Pi..3/2*Pi));
                           2.550326538
> evalf(rightsum(f(x),x=-3/2*Pi..3/2*Pi));
                           .9759677138
> evalf(simpson(f(x),x=-3/2*Pi..3/2*Pi));
                           2.872086610
> evalf(trapezoid(f(x),x=-3/2*Pi..3/2*Pi));
                           .9759677138
```

对于变化比较明显的代数表达式，用以上几个函数在 4 个运算矩阵进行运算时得到的结果均不理想，这是一个比较头疼的问题，为了进行进一步的比较，我们将默认的运算矩阵设成 20：

```
> evalf(leftsum(f(x),x=-3/2*Pi..3/2*Pi,20));
                           1.962851274
> evalf(middlesum(f(x),x=-3/2*Pi..3/2*Pi,20));
                           2.018626072
```

```
> evalf(rightsum(f(x),x=-3/2*Pi..3/2*Pi,20));
                              .962851274
> evalf(simpson(f(x),x=-3/2*Pi..3/2*Pi,20));
                              2.000562758
> evalf(trapezoid(f(x),x=-3/2*Pi..3/2*Pi,20));
                              1.962851274
```

而运算矩阵设成 100 时结果如下：

```
> evalf(leftsum(f(x),x=-3/2*Pi..3/2*Pi,100));
                              1.998519340
> evalf(middlesum(f(x),x=-3/2*Pi..3/2*Pi,100));
                              2.000740413
> evalf(rightsum(f(x),x=-3/2*Pi..3/2*Pi,100));
                              1.998519340
> evalf(simpson(f(x),x=-3/2*Pi..3/2*Pi,100));
                              2.000000878
> evalf(trapezoid(f(x),x=-3/2*Pi..3/2*Pi,100));
                              1.998519340
```

很显然，在相同的运算矩阵的情况下，simpson 函数得到的结果是最好的。

第5章 微分方程

在工程研究的函数是反映客观世界中量与量之间的一种依赖关系，或者反映某种物理、生物、社会等发展过程的数量关系，在大量的实际问题中，这些函数往往不能直接获得。但是经常可以建立起函数与其导数之间的某种关系。这种包含了函数及其导数的方程式就称为微分方程。科学与工程中的中的许多现象的内在规律，都需要用微分方程来描述。微分方程是用数学理论解决实际问题的重要渠道之一。有些有规律的方程可以用相关的规则进行求解，假如方程比较简单，求解可能比较容易，但大部分的微分方程都很难求解，即使能也是比较麻烦的，因此我们需要有一个强有力的工具来帮助我们来求解微分方程。Maple不仅提供了丰富的函数来直接求解微分方程，而且还提供了一些函数来帮助我们来理解微分方程的性质，使微分方程的求解变得容易。本章我们将从求常微分方程的解析解开始，用实例来介绍怎么用 Maple 来求各种微分方程的解。

5.1 常微分方程

如果方程中的未知函数是一元函数，则称这样的微分方程为常微分方程，而微分方程中出现的最高阶导数的阶数称为这个微分方程的阶。这两个概念在本节中要经常用到，所以我们先介绍了它们。常微分方程是微分方程中最简单的一种，大部分方程的形式都不复杂，但是求解过程就不像它们的形式那样了，通常是要确定方程的形式，然后利用前人总结的规律进行求解。Maple 提供了一套比较完善的程序包来帮助我们解微分方程，在这一节中我们将介绍 Maple 函数的用法。

5.1.1 常微分方程的解析解

我们求解微分方程的时候，总是希望能得到一个解析解，本小节我们先来介绍一个求解微分方程功能很强大的函数 dsolve，然后介绍如何用 Maple 提供的函数来判断微分的形式，同时介绍用 Maple 提供的解某种特定微分方程的函数来解决问题，在这过程中读者还将看到用 Maple 中专门为微分方程设计的画图函数描述出来的结果。

1. 函数 dsolve 在常微分方程中的应用

我们先来看 Maple 中求解微分方程最常用的一个函数 dsolve。在 Maple 中这是一个用途最广的函数，几乎可以求解所有的微分方程和方程组，既能求解解析解，也能求解数值

解，本节是介绍常微分方程的解析解的，所以我们在这一小节中只介绍函数 dsolve 在求解析解时的具体表达式，其他形式的函数表达式我们将在后面几小节中详细介绍。它的具体表达式如下：

```
dsolve(ODE);
dsolve(ODE, y(x), extra_args);
```

第一个参数 ODE 是一个常微分方程(Ordinary Differential Equation)；参数 y(x)表示包含一个自变量的待求函数，在求解常微分方程时，这个参数有时也可以省略；第三个参数 extra_arg 是一个可选的参数，主要是用来设置最后解析解的形式或求解过程中一些积分的设置，它的选值很广，我们不准备将所有的参数一一列举，下面我们只介绍几个常用的参数。

（1）explicit：这个参数要求函数求出最后具有变量独立的函数的具体表达形式，而不是一个隐函数的形式。

（2）implicit：这个参数要求函数得到的解可以是隐函数的形式。这在隐函数的解的形式比较简单而显函数的解很复杂的时候很有用，能够避免系统强制将解转换成显函数的形式。

（3）useInt：这个参数使函数在运算过程中用 Int 函数代替 int 函数（即在运算过程中用积分表达式代替积分结果，这两个函数的差别在上一章中提到过），这个参数对加快运算速度有很大的帮助。

（4）parametric：这个参数要求函数将最后的解析解表达成另外一个自变量的形式，比如{x=x(t)，y=y(t)}，这个参数要跟参数 implicit 合用，并且只适用于一阶常微分方程，并且并不是每个方程都适合用这个参数的，我们在后面的例子中将看到这一点。

这些参数的位置很灵活，可以放在除第一参数位置外的任何位置，并且它们的组合也很灵活，可以单独使用，也可几个参数合用，只要在中间用逗号隔开，而且参数并不一定需要写在一块，也可以分开，我们将在下面用具体的例子来说明。

【例 5-1】求微分方程(dy/dx)(y^2+1)+cosx=0 的解析解。

首先我们先写出微分方程的具体形式：

```
> eq:='eq':
  eq:=diff(y(x),x)*(1+y(x)^2)+cos(x)=0;
```

$$eq := \left(\frac{\partial}{\partial x} y(x)\right)(y(x)^2 + 1) + \cos(x) = 0$$

注意：由于我们要求解的函数 y 是关于 x 的函数，因此在整个等式中就需要将所有的 y 都写成 y(x)的形式，不然 Maple 会将 y 看成一个单独的变量。当然上面的那个等式中的微分形式也可以用函数 D 来表示，即写成 eq:=D(y)(x)*(y(x)^2+1)+cos(x)=0。有了这个等式，我们再用 dsolve 求一个用显函数表示的解。

```
> sol1:=dsolve(eq,explicit);
```

$$sol1 := y(x) = \frac{1}{2}\left(\left(-12\sin(x) - 12\_C1 + 2\sqrt{34 - 18\cos(2x) + 72\sin(x)\_C1 + 36\_C1^2}\right)^{\left(\frac{2}{3}\right)} - 4 \right) \Big/ \left(-12\sin(x) - 12\_C1 + 2\sqrt{34 - 18\cos(2x) + 72\sin(x)\_C1 + 36\_C1^2}\right)^{\left(\frac{1}{3}\right)},$$

这是一个很恐怖的解（并且远没有结束，为了节约篇幅，我们在这儿只列出第一个解，还有两个更长的解未列出，如果读者感兴趣，可以代入验证看一看），我们一时很难看出来这个解具体是什么形式的，其实这是函数在将最后的解转换成显函数形式时造成的，假如我们将参数 explicit 改成 implicit，结果将变得简单明了。

```
> sol2:=dsolve(eq,implicit,y(x));
```

$$sol2 := \sin(x) + y(x) + \frac{1}{3}y(x)^3 + C1 = 0$$

表示成隐函数的形式后，最后的解就成了初等函数的形式，比原来要直观得多。下面我们看看加上前面介绍的第二个参数 y(x)后的效果，可以看到结果是一样的，然后读者可以调整 implicit 参数的位置，验证一下这个参数的位置的多样性。

最后我们再来看看加上第三个参数关于 useInt 的属性是什么效果：

```
> sol3:=dsolve(eq,implicit,y(x));
```

$$sol3 := \int \cos(x)dx - \int^{y(x)} -1 - a^2 d\_a + \_C1 = 0$$

从上面可以看出，当加上参数 useInt 以后，函数用一个积分形式代替原解中的结果，这在求解过程中积分比较复杂时很有用。为了避免复杂的积分结果掩盖了一些解的信息，我们可以用这个参数，不但可以加快系统的运算速度，还能使最后的结果给出比较多的信息。最后我们将参数 parametric 加到函数 dsolve 中：

```
> sol4:=dsolve(eq,implicit,y(x),parametric);
```

$$sol4 :=$$

我们很惊讶的发现函数没有给出任何结果，这是因为微分方程的解实在太复杂了，函数找不到用参数表示的方法，下面我们用一个比较简单的例子来说明设置参数 parametric 以后的结果，大家很容易从结果中看出表示的方法，这里就不在详细介绍了。

```
> dsolve(diff(y(x),x)+y(x)=0,implicit,parametric);
```

$$[y(\_T) = \_T, x(\_T) = -\ln(\_T) + \_C1]$$

2. 用函数 odetest 检验常微分方程的解

千辛万苦终于求得了微分方程得解析解，为了保险起见，我们往往想验证一下结果得正确性。为了方便用户，Maple 专门提供了一个检验微分方程结果正确性的函数 odetest。它的具体函数表达式如下：

```
odetest(sol,ODE);
odetest(sol,ODE,y(x));
odetest(solsys,sysODE);
```

参数 sol 是待检验的微分方程的解，ODE 是一个常微分方程，y(x)是常微分方程中未确定的函数，这个参数有时是必须的，为了保险起见我们一般把它写上；最后一个函数表达式在用于判断微分方程组的解时有用，我们将在下一节中介绍。

函数 odetest 既能对显函数表示的解进行判断，也能对隐函数表示的解进行判断；在判断过程中首先将给定的解代入常微分方程中通过微分运算进行简化，然后判断两边是否相等，如果相等，则返回 0，否则返回那个化简了的代数表达式。下面我们来看一些具体的例子，主要判断上面的那些解，为了说明给定的结果不是给定方程的解时，函数 odetest 是如何处理的，我们在最后用一个随意取的代数表达式去测试，由于 odetest 函数在 Detools 程序包中，我们在开始使调用 with(Detools):

```
> with(DEtools):
  odetest(sol1[1],eq,y(x));
```

$$0$$

```
> odetest(sol2,eq,y(x));
```

$$0$$

```
> odetest(sol4,eq,y(x));
```

$$0$$

```
> y(x)=x^2;
```

$$y(x) = x^2$$

```
> odetest(%,eq,y(x));
```

$$2x^5 + 2x + \cos(x)$$

从判断的结果中得知，我们上面求得的常微分方程的解都是正确的，这也是很显然的。由于代数表达式 y=x^2 不是微分方程的解，函数判断后给出了表达式代入后化简的结果。需要注意的是，上面代入的表达式中所有的 y 都要写成 y(x)的形式，否则函数将提示找不

到 y(x)的信息，这与一开始写微分方程时的规则是一样的。

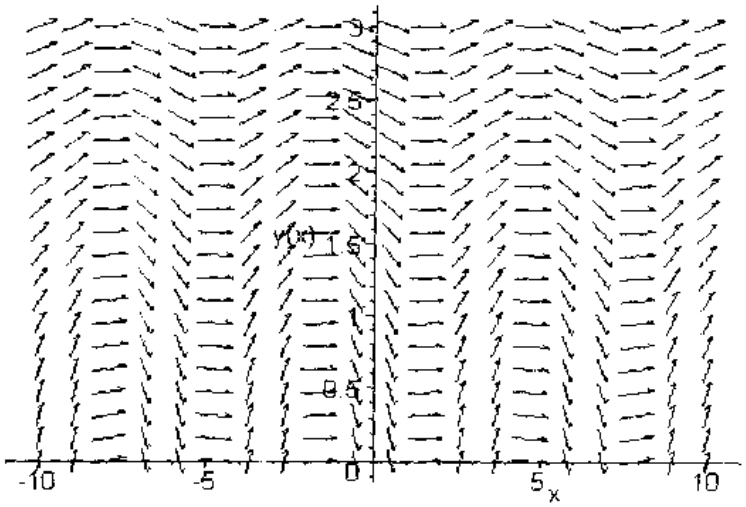
3. 用 DEplot 函数来显示微分方程的解的图像

上面我们不光用函数 dsolve 求出了微分方程的解，而且还用 odetest 验证了解的正确性，现在我们再来介绍一个函数 Deplot，它是一个专门用来求微分方程的图形解的函数，它可以直接来求微分方程的图形解，它的函数形式很复杂，在本节中我们只来介绍用于常微分方程的具体函数表达式：

DEplot(deqns, vars, trange, eqns);
DEplot(deqns, vars, trange, inits, eqns);

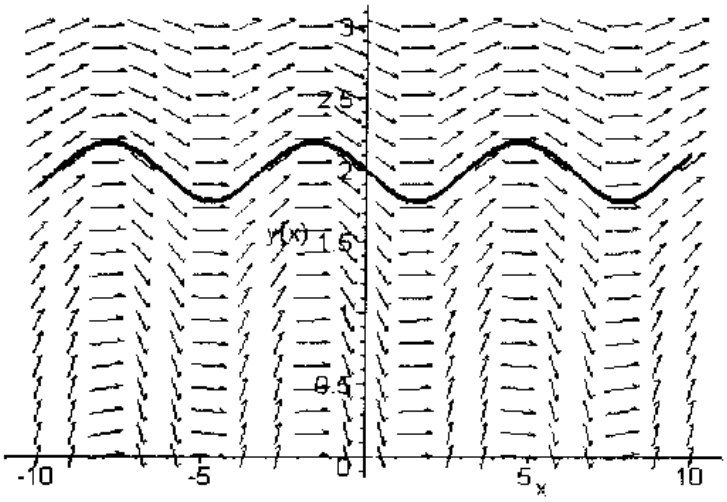
参数 deqns 是待求的微分方程或微分方程组，本节中就是指微分方程；第二个参数 vars 是一个非独立的变量或变量的集合，比如 y(x)；第三个参数 trange 设定变量的取值范围；第四个参数 eqns 是一个等式形式的条件，如：linecolor=blue(设定图像中线的颜色为蓝的)，stepsize=0.05（变量的取值步长）；参数 inits 是微分方程的初值条件，如果有多个初值条件，用集合表示，并且每个初值条件都必须用方括号包括起来，如果不给定初始值，函数将等间隔的取最后解中的常数为某值。请看下面的例子：

```
> with(DEtools):
  DEplot(eq,y(x),x=-10..10,y=0..3,linecolor=blue
  ,stepsize=0.05);
```



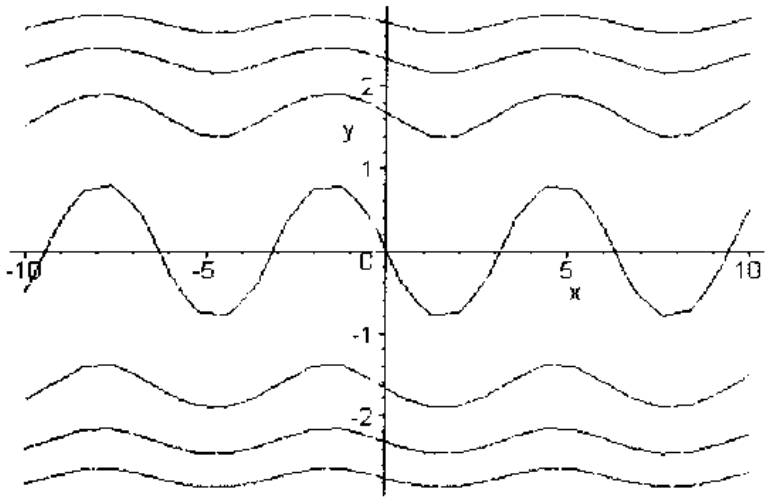
这个例子中没有为微分方程设定初始值，所以函数用一些箭头来表示解的趋势，假如我们设方程的初始值是 y(0)=2。图像将变为：

```
> with(DEtools):
  DEplot(eq,y(x),x=-10..10,[[y(0)=2]],y=0..3,lin
  ecolor=blue,stepsize=0.05);
```



图形中的粗线就是满足初始条件的解的图形，假如我们对这个图形是不是我们刚才的求得的解的图形有疑问的话，我们可以用另外一个作图函数 contourplot 来验证一下，这个函数在 plots 程序包里：

```
> with(plots):
  f:=sin(x)+1/3*y^3+y:
  contourplot(f,x=-10..10,y=-3..3,coloring=[red,blue]
  );
```



显然这个图形跟刚才是一样的，说明我们的求解没有错。

4. 用各种特定函数求解各种微分方程

Maple 除了提供直接求解微分方程的函数以外，还提供了一些帮助我们求解的函数.

比如 odeadvisor，这是一个能够给出微分方程的形式的函数，假如我们想判断一个微分方程到底是可分离变量的方程、一阶线性微分方程、全微分方程还是普通的微分方程，我们就可以用这个函数。现在我们来介绍这个函数最简单的用法，它的具体表达式如下：

odeadvisor(ODE):

参数 ODE 是待判断的常微分方程，这个函数将返回给定常微分方程的形式，其中最常见的几个返回参数和中文含义如下：

separable	常微分方程是一个可分离变量的方程
linear	常微分方程是一个线性微分方程
homogeneous	常微分方程是一个齐次微分方程
bernoulli	常微分方程是一个贝努里方程，它是一阶线性微分方程的一种
exact	常微分方程是一个全微分方程
Abel	常微分方程是一个阿贝尔（Abel）方程
quadrature	常微分方程是一个积分形式的微分方程（比如 dy=xdx），是可分离变量中最简单的一种

下面用此函数来判断一下本节开始处用到的微分方程(这个函数也在 DEtools 程序包中)：

```
> with(DEtools):
  eq:='eq':
  eq:=diff(y(x),x)*(y(x)^2+1)+cos(x)=0:
> odeadvisor(eq);
                        [_separable]
```

下面我们利用这个得力的工具来求解各种各样的常微分方程：

（1）可分离变量的常微分方程。这是我们在高等数学中最早见到的微分方程，也是微分方程中最容易求解的一种，方程的形式为 dy/dx=f(x)/g(y)或者 f(x)dx=g(y)dy。这种方程除了可以用上面提到的函数 dsolve 求解外，我们还可以调用 Maple 中专门求可分离变量的常微分方程的函数 separablesol 来求解，这个函数只有一种表达式：

separablesol(1ode,v);

参数 1ode 是待求的一阶常微分方程，v 是微分方程中的应变量，我们还是举例说明这个函数的用法：求微分方程 x*y(1-x*dy/dx)=x+y*dy/dx 的解。

首先我们定义微分方程，导数用函数 diff 表示：

```
> eq:='eq':
  eq:=x*y(x)*(1-x*diff(y(x),x))=x+y(x)*diff(y(x),x);
```

$$eq := xy(x)\left(1 - x\left(\frac{\partial}{\partial x} y(x)\right)\right) = x + y(x)\left(\frac{\partial}{\partial x} y(x)\right)$$

当然定义上面的微分方程也可以输入 eq:=x*y(x)*(1-x*D(y)(x))=x+y(x)*D(y)(x)，结果都是一样的，在这里我们再提醒一句：在微分方程中一定要将 y 写成 y(x)的形式，表明 y 是 x 的函数。然后我们用 odeadvisor 函数来判断微分方程的形式：

```
> with(DEtools):
  odeadvisor(eq);
```

$$[\_separable]$$

函数返回参数 separable，说明给定的微分方程是可分离变量的微分方程，我们可以用函数 separablesol 来求解：

```
> separablesol(eq,y(x));
```

$$y(x) = \text{LambertW}(\sqrt{x^2+1}\_C1\,e^{(-1)}) + 1$$

我们看到：函数 separablesol 返回解的形式是一个集合的形式，这是由于我们并没有给微分方程定初值，函数得到的解是一个通解，当可取任意值的常数时，方程能得到无穷多个解。下面来比较一下用 dsolve 函数得到的解，我们将看到它们是完全相同的：

```
> dsolve(eq);
```

$$\{y(x) = \text{LambertW}(\sqrt{x^2+1}\_C1\,e^{(-1)}) + 1\}$$

这个解与用 separablesol 函数求得的解的唯一区别就是没有写成集合的形式。除了用这两种方法可以求得可分离变量的微分方程的解以外，在高等数学中还有一种解法——分离变量法，其求解方法为：

将可分离变量的微分方程写成 f(x)dx=g(y)dy，然后两边同时积分得到微分方程的隐式解：

$$\int f(x)\,dx = \int g(y)\,dy + C$$

上面 C 是积分常数，将 y 解出就可得到显式解。下面我们就用这种方法来求上面那个微分方程的解：

首先我们解出 dy/dx 的形式，为分离变量做准备：

```
> solve(eq,diff(y(x),x);
```

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

上面的代数表达式，我们很容易就能找到 f(x)和 g(y):

```
> f:='f':
  g:='g':
  f:=x->x/(x^2+1);
  g:=y->y/(y-1);
```

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

$$g := y \to \frac{y}{(y-1)}$$

然后对两边的代数表达式（即 f(x),g(y)）进行积分:

```
> LHS:=int(f(x),x);
  RHS:=int(g(y),y);
```

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

$$RHS := y + \ln(y-1)$$

最后将左右两边划上等号就得到了微分方程的解，为了说明这个解是方程的通解，在等式的右边加上一个积分常数_C1:

```
> LHS=RHS+C1;
```

$$\frac{1}{2}\ln(x^2+1) = y + \ln(y-1) + \_C1$$

现在得到的是微分方程的隐式解，为了与上面的结果进行比较，用 solve 函数将其解出:

```
> solve(%,y);
```

$$\text{LambertW}\left(e^{\left(\frac{1}{2}\ln(x^2+1)-1-\_C1\right)}\right)+1$$

很明显这个结果与上面的结果是相同的。

（2）一阶齐次常微分方程。假如一阶常微分方程能够写成 dy/dx=f(y/x)的形式，我们就把这个方程称为一阶齐次微分方程。与可分离变量的微分方程类似，一阶齐次微分方程也可以用一个专门的函数来求解，现在让我们来看看这个函数的具体形式：

genhomosol(1ode,v);

参数 lode 是待求的一阶齐次常微分方程，v 是微分方程中的应变量。这个函数属于 DEtools 程序包，用之前应先载入这个程序包。

还是用例子来说明一阶常微分方程的解法和函数 genhomosol 的用法：

【例 5-2】 求一阶常微分方程 $\frac{\partial}{\partial x}y=\frac{y}{x}+\tan\left(\frac{y}{x}\right)$ 的通解。

还是跟前面一样，首先定义微分方程式：

```
> eq:='eq':
> eq:=diff(y(x),x)=y(x)/x+tan(y(x)/x);
```

$$eq:=\frac{\partial}{\partial x}y(x)=\frac{y(x)}{x}+\tan\left(\frac{y(x)}{x}\right)$$

然后用函数 odeadvisor 来判断微分方程的形式：

```
> with(DEtools):
  odeadvisor(eq);
```

$$[[\_homogeneous, class\ A], [\_lst\_order, \_with\_linear\_symmetries], \_dAlembert]$$

这个函数除了返回参数 homogeneous，确认微分方程式是齐次方程外，还返回此微分方程是一阶（1st_order）、线性对称（with_linear_symmetries）方程的信息。确认待求微分方程是一个一阶齐次微分方程后，我们用函数 genhomosol 来求解：

```
> genhomosol(eq,y(x));
```

$$\left\{y(x)=\arctan\left(\frac{\sqrt{-(-1+\_C1x^2)\_C1}\,x}{-1+\_C1x^2}\right)x,\ y(x)=-\arctan\left(\frac{\sqrt{-(-1+\_C1x^2)\_C1}\,x}{-1+\_C1x^2}\right)x\right\}$$

函数得到两种形式的通解，并且用集合的形式表示，这与函数 separablesol 是一致的。当然，与上面一样，我们也可以用函数 dsolve 来直接求解：

```
> dsolve(eq,y(x),explicit);
```

$$y(x)=\arctan\left(\frac{\sqrt{-(-1+\_C1x^2)\_C1}\,x}{-1+\_C1x^2}\right)x,\quad y(x)=-\arctan\left(\frac{\sqrt{-(-1+\_C1x^2)\_C1}\,x}{-1+\_C1x^2}\right)x$$

很明显两种解法得到的结果是一样的。最后我们来看看利用 Maple 来实现我们平时求解一阶齐次微分方程的过程：

先来回顾一下一阶齐次方程的形式：dy/dx=f(y/x)。我们自然而然的想到用另外一个变量 u 去代替 y/x（即 y=ux），则原微分方程可化为：（udx+xdu）/dx=f(u)即 u+xdu/dx=f(u)。变成了一个可分离变量的常微分方程，接下来就可用分离变量法来求解了，看具体的例子：

首先我们将 y=u*x 代入原微分方程中，用 expand 展开得到关于自变量 x 和应变量 u 的微分方程：

```
> y:=u*x:
> one:=expand(eq);
```

$$one := \left(\frac{\partial}{\partial x}\mathrm{u}(x)\right)\mathrm{x}(x) + \mathrm{u}(x)\left(\frac{\partial}{\partial x}\mathrm{x}(x)\right) = \frac{\mathrm{u}(x)\mathrm{x}(x)}{x} + \tan\left(\frac{\mathrm{u}(x)\mathrm{x}(x)}{x}\right)$$

在上面，Maple 将 dx/dx 当成了一个微分式子，并没有当成 1，所以我们得用函数 subs 将上式中所有的 dx/dx 替换成 1，x(x)替换成 x：

```
> two:=subs(diff(x(x),x)=1,x(x)=x,one);
```

$$two := \left(\frac{\partial}{\partial x}\mathrm{u}(x)\right)x + \mathrm{u}(x) = \mathrm{u}(x) + \tan(\mathrm{u}(x))$$

这样就得到了真正的关于 x 和 u 的微分方程式，然后用函数 solve 解出 du/dx 的值，得到能够分离的变量：

```
> solve(tow,diff(u(x),x));
```

$$\frac{\tan(\mathrm{u}(x))}{x}$$

很明显这个方程式是能够分离变量的，我们可以写成：(1/tan(u))du=(1/x)dx 的形式，然后将两边的代数表达式进行积分：

```
> solve(tow,diff(u(x),x));
```

$$\frac{\tan(\mathrm{u}(x))}{x}$$

```
> left:=int(1/tan(u),u);
  rignt:=int(1/x,x);
```

$$left := \ln(\tan(u)) - \frac{1}{2}\ln(1+\tan(u)^2)$$

$$right := \ln(x)$$

left 中的 u 用 y/x 代入，并将 left 和 right 划上等号，再加上一个积分常数就得到了微分方程的通解：

```
> y:='y':x:='x':
  subs(u=y/x,left)=right+_C1;
```

$$\ln\left(\tan\left(\frac{y}{x}\right)\right)-\frac{1}{2}\ln\left(1+\tan\left(\frac{y}{x}\right)^2\right)=\ln(x)+\_C1$$

最后利用函数 solve 来求得微分方程的解：

```
> solve(%,y);
```

$$\arctan\left(\frac{\sqrt{-(-1+e^{(2\_C1)}x^2)e^{(2\_C1)}}x}{-1+e^{(2\_C1)}x^2}\right),$$

$$-\arctan\left(\frac{\sqrt{-(-1+e^{(2\_C1)}x^2)e^{(2\_C1)}}x}{-1+e^{(2\_C1)}x^2}\right)x$$

初看这个结果与上面的结果不太一样，其实只是积分常数的选取不太一致而已，很明显假如我们将 exp(2_C1)用另一个积分常数（比如_C2）来代替，结果就完全一样了。

（3）一阶线性常微分方程。假如一阶常微分方程能够写成：a(x)dy/dx+b(x)y+c(x)=0 的形式，其中 a，b，c 是在区间 I 上的连续函数，这方程就称为一阶线性常微分方程。a(x)≠0 的区间上，此方程可化为如下形式：dy/dx+p(x)=q(x)，其中 p，q 是在区间 I 上连续的函数。

对于一阶线性方程，Maple 里有一个专门的函数 linearsol 来求解，这个函数的具体表达式如下：

linearsol(lode,v);

参数 lode 是待求的一阶线性微分方程，v 是微分方程中的应变量。这个函数在求解过程中首先判断给定的微分方程是否是一阶线性常微分方程，如果是则返回微分方程的解，否则返回一个空的集合。下面我们通过求微分方程：dy/dx+y/x=sinx/x 来说明这个函数的用法，还是跟以前一样，我们首先定义微分方程：

```
> eq:='eq':
  eq:=diff(y(x),x)+y(x)/x=sin(x)/x;
```

$$eq:=\left(\frac{\partial}{\partial x}y(x)\right)+\frac{y(x)}{x}=\frac{\sin(x)}{x}$$

然后用 odeadvisor 函数来判断微分方程的形式：

```
> with(DEtools):
  odeadvisor(eq);
```

$$[\_linear]$$

函数返回参数 linear，说明 eq 是一个线性微分方程，我们可以用函数 linearsol 来求解：

```
> linearsol(eq,y(x));
```

$$\{y(x) = \frac{-\cos(x) + \_C1}{x}\}$$

最后用函数 odetest 检验解是否满足原方程：

```
> odetest(%[1],eq);
```

$$0$$

返回值为零，说明求得的解是正确的。下面我们用函数 dsolve 来求解：

```
> dsolve(eq,y(x),explicit);
```

$$y(x) = \frac{-\cos(x) + \_C1}{x}$$

很明显得到的解是相同的。在高等数学中解非齐次微分方程通常是用常数变异法来求解的，具体的做法就是先假设微分方程的解有如下的形式：

$$y = c(x)\,e^{\left(\int -p(x)\,dx\right)}$$

然后将此式代入方程 dy/dx+p(x)y=q(x)中，求出待定的函数 c(x):

$$c(x) = \int q(x)\,e^{\left(\int p(x)\,dx\right)}\,dx + c$$

最后得到要求的微分方程的解为：

$$y = \left(\int q(x)\,e^{\left(\int p(x)\,dx\right)}\,dx + c\right) e^{\left(\int -p(x)\,dx\right)}$$

现在我们就用 Maple 按照这一思路来求出上面那个微分方程的解。

首先我们定义 p(x)，q(x):

```
> p:='p':q:='q':
> p:=x->1/x;
  q:=x->sin(x)/x;
```

$$p := x \to \frac{1}{x}$$

$$q := x \to \frac{\sin(x)}{x}$$

然后代入上面的那个式子用 Maple 积分即能得到方程的解：

```
> y:=(int(q(x)*exp(int(p(x),x)),x)+_C1)*exp(-
  int(p(x),x));
```

$$y := \frac{-\cos(x) + \_C1}{x}$$

这个结果跟上面的那些是相同的。一阶线性微分方程是微分方程的基础，很多方程都要转换成这种方程来求解，在这里我们再举一个例子，在举这个例子之前我们先来介绍一下贝努里（Bernoulli）方程的定义，在高等数学中将形式为 $\frac{dy}{dx} + \mathrm{p}(x)\, y = \mathrm{q}(x)\, y^n$ (n≠0) 的微分方程称为贝努里方程，贝努里方程虽然不是一个线性方程，但可以转换为一阶线性微分方程来求解，转换的过程如下：

首先，将方程的两边都除以 y^n，等式变成：

$$y^{-n}\frac{dy}{dx} + p(x)y^{1-n} = q(x)$$

再令 $z = y^{1-n}$，则：

$$\frac{dz}{dx} = (1-n)y^{-n}\frac{dy}{dx} \qquad \text{即} \qquad y^{-n}\frac{dy}{dx} = \frac{1}{1-n}\frac{dz}{dx}$$

将其代入上面的式子则得到一阶线性微分方程：

$$\frac{dz}{dx} + (1-n)p(x)z = (1-n)q(x)$$

利用求解一阶线性微分方程的解法很容易就能得到 z，再还原为 y 就能得到原微分方程的解，下面我们来看一个具体的例子：求解方程 $\frac{dy}{dx} = 6\frac{y}{x} - xy^2$。

首先我们定义微分方程：

```
> eq:='eq':
  eq:=diff(y(x),x)=6*y(x)/x-x*y(x)^2;
```

$$eq := \frac{\partial}{\partial x}\mathrm{y}(x) = 6\frac{\mathrm{y}(x)}{x} - x\,\mathrm{y}(x)^2$$

这是一个 n=2 的贝努里方程，这个我们可以用函数 odeadvisor 来验证，如下：

```
> with(DEtools):
  odeadvisor(eq);
[[_homogeneous, class G],
  [_1st_order, _with_linear_symmetries], _rational, _Bernoulli]
```

最后一个返回的参数表明是一个贝努里方程，这样我们就可以用上面的方法来求解，先用 z 来代替 1/y，即令 z=1/y:

```
> z:='z':y:='y':x:='x':
  y:=1/z:
  eq1:=subs(y=1/z,eq);
```

$$eq1 := \frac{D(z)(x)}{z(x)^2} = 6\frac{1}{z(x)x} - \frac{1}{z(x)^2}$$

现在我们再来看看这个微分方程是什么形式:

```
> odeadvisor(eq1,z(x));
                        [_linear]
```

我们看到通过代换后，贝努里方程变成了一阶线性常微分方程，现在利用 solve 函数求出 D(z)(x)，这样就很容易看出 p(x)和 q(x):

```
> solve(eq1, D(2)(x));
```

$$\frac{-6z(x) + x^{2'}}{x}$$

```
> p:='p':q:='q':
  p:=x->6/x;
  q:=x->x;
```

$$p := x \to 6\frac{1}{x}$$

$$q := x \to x$$

再利用一阶线性微分方程的求解公式将 z 解出:

```
> z:=(int(q(x)*exp(int(p(x),x))+_C1)*exp
  (-int(p(x),x));
```

$$z := \frac{\frac{1}{8}x^8 + \_C1}{x^6}$$

最后将 z 用 1/y 代替，则得到最后的解：

```
> y:=1/z;
```

$$y := \frac{x^6}{\frac{1}{8}x^8 + \_C1}$$

这样我们就求得了贝努里方程的解，同时我们用函数 dsolve 来求解比较一下结果：

```
> y:='y':y:=x->y(x):
  dsolve(eq,y(x),explicit);
```

$$y(x) = 8\frac{x^6}{x^8 + 8\_C1}$$

最后我们来介绍 Maple 中一个专门求贝努里方程的函数 bernoullisol，它的具体函数表达式如下：

bernoullisol(1ode,v);

参数 1ode 是待求的一阶常微分方程，v 是微分方程中的应变量。这个函数首先判断给定的微分方程是否具有贝努里方程的形式，如果是则解出结果并返回求出的解，否则返回空集。下面我们就用此函数来求上面的贝努里方程：

```
> bernoullisol(eq, y(x));
```

$$y(x) = 8\frac{x^6}{x^8 + 8\_C1}$$

我们很高兴的看到所得到的解是完全相同的。

（4）全微分方程。有时我们将微分方程写成一般形式：

$$M(x,y)dx+N(x,y)dy=0$$

其中 M(x,y)和 N(x,y)是在平面上某个单连通区域内连续可微的函数，如果存在一个二元连续可微的函数 u(x,y)，使得：

$$\frac{\partial u}{\partial x} = M(x,y), \frac{\partial u}{\partial y} = N(x,y) \quad 即 \quad du = Mdx + Ndy$$

则称上面那个方程为全微分方程（exact differential equation）。全微分方程成立的充要条件是：

$$\frac{\partial M(x,y)}{\partial x} = \frac{\partial N(x,y)}{\partial y}$$

在 Maple 中我们可以利用这一点来判断微分方程是否是全微分方程，当然我们也可以用函数 odeadvisor 来判断，这两种方法我们在下面都将举例说明。在 Maple 中有一个专门用来求一阶全微分方程的函数 exactsol，它的具体函数表达式如下：

exactsol(lode,v);

参数 lode 是待求的微分方程，v 是微分方程中的待求的应变量。则这个函数首先判断给定的微分方程是否是全微分方程，如果是则进行求解运算并返回得到的微分方程的解.

下面我们来求一个方程：$\frac{dy}{dx}=\frac{2+ye^{xy}}{2y-xe^{xy}}$，通过例子来说明函数的应用。

首先我们得到 M(x,y)和 N(x,y)，同时定义微分方程：

```
> M:='M':N:='N':
  M:=(x,y)->2+y*exp(x*y);
  N:=(x,y)->-(2*y-x*exp)x*y));
```

$$M := (x, y) \to 2 + y\mathrm{e}^{(yx)}$$

$$N := (x, y) \to -2 + x\mathrm{e}^{(yx)}$$

```
> eq:='eq':
  eq:=M(x,y(x))+N(x,y(x))*diff(y(x),x)=0;
```

$$eq := 2 + \mathrm{y}(x)\mathrm{e}^{(\mathrm{y}(x)x)} + (-2\mathrm{y}(x) + x\mathrm{e}^{(\mathrm{y}(x)x)})\left(\frac{\partial}{\partial x}\mathrm{y}(x)\right) = 0$$

接下来我们分别利用全微分方程的充要条件和函数 odeadvisor 来判断方程 eq 是否是全微分方程：

```
> testeq(diff(M(x,y),y),diff(N(x,y),x));
```

true

```
> with(DEtools):
  odeadvisor(eq,y(x));
```

[_exact]

testeq 函数返回 true，而 odeadvisor 函数返回 exact，都表明微分方程 eq 是一个全微分方程，所以我们能够用函数 exactsol 来求解：

```
> exactsol(eq,y(x));
```

$$\{2x + \mathrm{e}^{(\mathrm{y}(x)x)} - \mathrm{y}(x)^2 + \_C1 = 0\}$$

返回的是以集合形式保存的通解，当然我们也能用函数 dsolve 来求解，从下面可以看出它们得到的结果是相同的：

```
> dsolve(eq,y(x),implicit);
```

$$\{2x+e^{(y(x)x)}-y(x)^2+\_C1=0\}$$

（5）高阶线性常系数齐次常微分方程。假如一个常微分方程可以写成如下形式：

$$\frac{d^n x}{dt^n}+a_1\frac{d^{n-1}x}{dt^{n-1}}+\cdots+a_{n-1}\frac{dx}{dt}+a_n x=0$$

其中 $a_1,\ldots,a_n$ 是常数，则称这种方程为高阶线性常系数齐次常微分方程。它的特征方程为：

$$\lambda_n+a_1\lambda^{n-1}+\cdots+a_0=0$$

特征方程的根称为特征根。在高等数学中有这样的定理：

1）设 λ 是特征方程的单重实根，则 $e^{\lambda t}$ 是常微分方程的一个实解；

2）设 α ±iβ 是特征方程的一对单重复根，则 $e^{\alpha t}\cos(\beta t)$, $e^{\alpha t}\sin(\beta t)$ 是微分方程的两个线性无关解；

3）设 λ 是特征方程的 k(1<k≤n)重实根，则 $e^{\lambda t}$，$te^{\lambda t}$，...，$t^{k-1}e^{\lambda t}$，是方程的 k 个线性无关的实解；

4）设 α ±iβ 是特征方程的一对 k(1<k≤n/2)重复根，则 $e^{\alpha t}\cos(\beta t)$, $e^{\alpha t}\sin(\beta t)$, $te^{\alpha t}\cos(\beta t)$, $te^{\alpha t}\sin(\beta t)$,...，$t^{k-1}e^{\alpha t}\cos(\beta t)$, $t^{k-1}e^{\alpha t}\sin(\beta t)$ 是微分方程的 2k 个线性无关解。

也就是说无论常系数线性常微分方程的特征函数的特征根是什么形式的，均能够得到 n 个线性无关解，也就能得到通解。下面我们举一个例子：

【例 5-3】 求 x(4)-x=0 的通解。

首先我们定义微分方程：

```
> eq:='eq':
  eq:=diff(x(t),t$4)-x(t)=0;
```

$$eq:=\left(\frac{\partial}{\partial t^4}x(t)\right)-x(t)=0$$

然后解出它的特征方程的特征根，利用上面的定理得到微分方程的解通：

```
> solve(m^4-1=0);
```

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

```
> x:='x':
  x:=t->x(t):
  sol1:=x(t)=_C1*exp(-1*t)+C2*exp(t)+_C3*cos(
  t)+_C4*sin(t);
```

$$sol1 := x(t) = \_C1\,e^{(-t)} + \_C2e^{t} + \_C3\cos(t) + \_C4\sin(t)$$

上面式子中我们首先将 x 自己的值传给自己，然后再定义它是关于 t 的函数，这是为了 x(t)是一个初始函数，并没有赋值。最后我们利用函数 odetest 来验证一下结果的正确性：

```
> with(DEtools):
  odetest(sol1,eq);
                          0
```

函数返回 0 说明我们求得的结果是正确的，下面我们再利用函数 dsolve 求解来比较一下：

```
> x:='x':
  x:=t->x(t):
  sol2:=dsolve(eq,x(t),explicit);
```

$$sol2 := x(t) = \_C1\,e^{t} + \_C2\sin(t) + \_C3\cos(t) + \_C4\,e^{-t}$$

得到的结果是一样的，在这里我们同样对 x(t)进行了初始化，如果读者有兴趣的话，可以试一试不经初始化直接使用会有什么后果。

（6）高阶常系数非齐次常微分方程。假如一个常微分方程可以写成如下形式：

$$\frac{d^n x}{dt^n} + a_1\frac{d^{n-1}x}{dt^{n-1}} + \cdots + a_{n-1}\frac{dx}{dt} + a_n x = f(t)$$

其中 $a_1,\ldots,a_n$ 是常数，$f(t)\neq 0$ 并且是已知的连续函数，则称这种方程为高阶线性常系数非齐次常微分方程。非齐次微分方程的具体解法非常复杂，这里只是简要的介绍一下比较系数法的一般过程，具体的介绍和公式推导请参考相关的高等数学书。

比较系数法的一般思路是先将该方程的非齐次项（即 f(t)）设为零，用求解常系数齐次常微分方程的方法先求出通解，然后假设一个特解 $x(t)=Q(t)e^{\lambda t}$,其中 Q(t)是一个待定的多项式。最后将这个特解代入原方程中，通过比较相应的系数就能得到特解，再加上原来得到的通解就完成了微分方程的通解。下面我们通过例子来说明这种解法的应用，求方程 $\frac{d^2x}{dx^2} - 3\frac{dx}{dt} = t^2 + 1 + te^{2t}$ 的解，首先我们定义这个微分方程以及它相应的常系数齐次常微分方程，并利用 solve 求出特征方程的特征根：

```
> neq:='eq':
  neq:=diff(x(t),t,t)-3*diff(x(t),t)=0
```

$$eq := \left(\frac{\partial^2}{\partial t^2}\mathrm{x}(t)\right) - 3\left(\frac{\partial}{\partial t}\mathrm{x}(t)\right) = 0$$

```
> nq:='eq':
  nq:=diff(x(t),t,t)-3*diff(x(t),t)=t^2+1+t*exp(2*t
  );
```

$$neq := \left(\frac{\partial^2}{\partial t^2}\mathrm{x}(t)\right) - 3\left(\frac{\partial}{\partial t}\mathrm{x}(t)\right) = t^2 + 1 + \mathrm{e}^{(2t)}t$$

```
> solve(m^2-3*m=0,m)=0;
```

$$(0,3) = 0$$

这是一个比较复杂的微分方程，直接比较系数比较困难，为了化简，我们将这个微分方程中的非齐次项拆成两个项，一项中不包含指数项，另一项中是指数项和多项式的乘积。这样一个微分方程就变成了两个常系数非齐次常微分方程：

```
> neq1:=diff(x(t),t,t)-3*diff(x(t),t)=t^2+1;
  neq2:=diff(x(t),t,t)-3*diff(x(t),t)=t*exp(2*t);
```

$$neq1 := \left(\frac{\partial^2}{\partial t^2}\mathrm{x}(t)\right) - 3\left(\frac{\partial}{\partial t}\mathrm{x}(t)\right) = t^2 + 1$$

$$neq2 := \left(\frac{\partial^2}{\partial t^2}\mathrm{x}(t)\right) - 3\left(\frac{\partial}{\partial t}\mathrm{x}(t)\right) = \mathrm{e}^{(2t)}t$$

下面就可以假设特解的形式了，由于第一个方程(neq1)中不含指数项，或者说指数项是 e^{0t}，而 0 是原方程的特征单根，所以第一个方程的特解应设为：

```
> x1:=t->t*(a1*t^2+b1*t+c1)*exp(0*t);
```

$$x1 := t \to t(a1\,t^2 + b1\,t + c1)\mathrm{e}^0$$

第二个方程中包含指数项 e^{2t}，而 2 不是特征方程的特征根，所以特解设为：

```
> x1:=t->t*(a1*t^2+b1*t+c1)*exp(0*t);
```

$$x1 := t \to t(a1\,t^2 + b1\,t + c1)\mathrm{e}^0$$

接下来我们先来求第一个方程的特解，即通过比较系数来确定特解中的系数，首先将特解的形式代入方程 neq1 中，并且用 expand 函数来展开，为了保证 x(t)是一个初始函数，

我们在开头加了定义：

```
> x:='x':
  x:=t->x(t):
  sol1:=expand(subs(x(t)=x1(t),neq1));
```

$$sol1 := 6a1\,t + 2b1 - 9a1\,t^2 - 6a1\,t - 3c1 = t^2 + 1$$

然后用 subs 函数的替代功能，分别设 t2=0，t=0；t2=0，t=1 和 t2=1,t=0 来得到项的系数，在求 t2 和 t 的项的系数时通过减去了零次项的系数来求得：

```
> sol11:=subs(t^2=0,t=0,sol1);
```

$$sol11 := 2b1 - 3c1 = 1$$

```
> sol12:=subs(t^2=0,t=1,sol1-sol11);
```

$$sol12 := 6a1 - 6b1 = 0$$

```
> sol13:=subs(t^2=0,t=0,sol1-sol11);
```

$$sol13 := -9a1 = 1$$

利用函数 solve 很轻松就能得到系数的解：

```
> solve({sol11,sol12,sol13});
```

$$\{a1 = \frac{-1}{9}, b1 = \frac{-1}{9}, c1 = \frac{-11}{27}\}$$

最后我们将这些系数代入到原来的假设中，就得到了第一个方程的特解：

```
> solve1:=subs(%,x1(t));
```

$$solve1 := t\left(-\frac{1}{9}t2 - \frac{1}{9}t - \frac{11}{27}\right)$$

通过同样的步骤我们能够求出第二个方程的特解：

```
> x:='x':
  x:=t->x(t):
  sol2:=simplify(expand(subs(x(t)=x2(t),neq2)));
```

$$sol12 := -2e^{(2t)}a2t^2 + 2e^{(2t)}a2t + 2e^{(2t)}a2 - 2e^{(2t)}b2t + e^{(2t)}b2 - 2e^{(2t)}c2 = e^{(2t)}$$

代入特解并展开后我们发现等式中多了一项我们用不着的项，不过这没关系，我们可以利用两边都除以一个指数项再来比较系数：

```
> sol21:=simplify(subs(t^2=0,t=0,sol2/exp(2*t)));
```

$$sol12 := 2a2 + b2 - 2c2 = 1$$

```
> sol22:=simplify(subs(t^2=0,t=1,sol2/exp(2*t)-so
  l21)));
```

$$sol22 := 2a2 - 2b2 = 1$$

```
> sol23:=simplify(subs(t^2=1,t=0,sol2/exp(2*t)-so
  l21)));
```

$$sol23 := -2a2 = 0$$

```
> solve({sol21,sol22,sol23})
```

$$\{a2 = 0, b2 = \frac{-1}{2}, c2 = \frac{-1}{4}\}$$

这样将系数代入第二假设的特解中就得到了第二个方程的特解：

```
> solve2:=subs(%,x2(t));
```

$$solve2 := \left(-\frac{1}{4} - \frac{1}{2}t\right)e^{(2t)}$$

按照高等数学中的定理，只要将两个特解相加就是原方程的一个特解，这个定理是很显然的，证明也很容易，这里不再证明了，我们由此得到了原方程的一个特解：

```
> x:='x':
  x:=t->x(t):
  cs:=x(t)=solve1+solve2;
```

$$cs := x(t) = t\left(-\frac{1}{9}t^2 - \frac{1}{9}t - \frac{11}{27}\right) + \left(-\frac{1}{4} - \frac{1}{2}t\right)e^{(2t)}$$

为了验证结果的正确性，我们调用函数 odeadvisor 来检验：

```
> with(DEtools):
  odetest(cs,neq);
```

$$0$$

返回值为零，说明我们求得的特解是正确的，下面我们来给出通解。由于上面已经求出了齐次方程的特征根，我们很容易就能求出非齐次微分方程对应的齐次方程的通解：

```
> es:= _C1*exp(0*t)+ _C2*exp(3*t);
```

$$es := \_C1 + \_C2\,e^{(3t)}$$

将齐次方程的通解和非齐次方程的特解相加就得到了非齐次方程的通解：

```
> x:='x':
> x:=t->x(t):
  ts:=x(t)=rhs(cs)+es;
```

$$ts := x(t) = t\left(-\frac{1}{9}t^2 - \frac{1}{9}t - \frac{11}{27}\right) + \left(-\frac{1}{4} - \frac{1}{2}t\right)e^{(2t)} + \_C1\frac{11}{27}t + \_C2e^{(3t)}$$

最后我们用函数 odetest 来验证一下结果的正确性，并用函数 dsolve 直接求解得到的结果来与之比较一下：

```
> odetest(ts,neq);
```

$$0$$

```
> dsolve(neq,x(t),explicit);
```

$$x(t) = \frac{1}{2}e^{(2t)}t - \frac{1}{4}e^{(2t)} - \frac{1}{9}t^3 - \frac{1}{9}t^2 + \frac{1}{3}e^{(3t)} - \_C1\frac{1}{9}t + \_C2$$

（7）常微分方程的级数解。微分方程的级数解是一个很有用的工具，当解用普通函数表述比较复杂，或者我们并不想知道解的高阶项情况时，我们就可以用 Maple 求级数解的功能来求解。它是由函数 dsolve 来实现的，具体的函数表达式如下：

```
dsolve(ODE, y(x), 'series');
dsolve({ODE, ICs}, y(x), 'series');
dsolve({sysODE, ICs}, {funcs}, 'series');
dsolve(ODE, y(x), 'type=series');
dsolve({ODE, ICs}, y(x), 'type=series');
dsolve({sysODE, ICs}, {funcs}, 'type=series');
```

从上面看函数表达式与先前用来求普通解时的表达式没什么区别，只是加了一个参数 'series' 或 'type=series'，这个参数是用来指定函数求得的解是级数形式的，两种参数表达式的效果是相同的，我们可以用任何一种。用这个函数时需要注意的是，如果不特别指定级数展开的阶数，则函数使用默认值（5 阶），如果在函数中没有给定微分方程的初始条件，则函数默认在 x=0 点处展开，如果我们给定了初始条件，则在给定的初始条件处展开。下面我们通过例子来熟悉一下：

【例 5-4】 求方程：$\frac{d^2y}{dx^2} + (\frac{dy}{dx})^2 = 0$ 的级数解。

我们首先看看在系统默认的情况下的解：

```
> eq:='eq':
  eq:=diff(y(x),x,x)+diff(y(x),x)^2=0;
```

$$eq := \left(\frac{\partial^2}{\partial x^2} y(x)\right) + \left(\frac{\partial}{\partial x} y(x)\right)^2 = 0$$

```
> ans:=dsolve({eq},y(x),type=series);
```

$$ans := y(x) = y(0) + D(y)(0)x - \frac{1}{2}D(y)(0)^2 x^2 + \frac{1}{3}D(y)(0)^3 x^3 - \frac{1}{4}D(y)(0)^4 x^4 + \frac{1}{5}D(y)(0)^5 x^5 + O(x)^6$$

下面我们将微分方程的初始条件（y(a)=A、dy(a)/dx=B）加入到函数 dsolve 的级数解形式中，看看得到的解有什么不同：

```
> ani:=dsolve({eq,y(a)=A,D(y)(a)=B},y(x),'series');
```

$$ani := y(x) = A + B(x-a) - \frac{1}{2}B^2(x-a)^2 + \frac{1}{3}B^3(x-a)^3 - \frac{1}{4}B^4 (x-a)^4 + \frac{1}{5}B^5(x-a)^5 + O((x-a)^6)$$

当给定待求的函数在 a 点处的初值后，级数就在点 a 处展开了。现在展开的级数还是 5 阶的，为了设定结果的阶数，必须在函数的外边用 Order 参数来设定，具体用法如下：

```
> Order:=8:
  anj:=dsolve(eq,y(x),series);
```

$$ani := y(x) = y(0) + D(y)(0)x - \frac{1}{2}D(y)(0)^2 x^2 + \frac{1}{3}D(y)(0)^3 x^3 - \frac{1}{4}D(y)(0)^4 x^4 + \frac{1}{5}D(y)(0)^5 x^5 - \frac{1}{6}D(y)(0)^6 x^6 + \frac{1}{7}D(y)(0)^7 x^7 + O(x^8)$$

在设定展开的阶数时有两点需要注意：一是展开的阶数并不是像我们要求的那样真正到 Order 阶，而是到 Order-1 阶，这从上面就可以看出来，如果读者有兴趣可以自己多试几次；二是这个参数的设定是有继承性的，也就是说这个参数不光对紧跟其后的函数求解起作用，它对所有在其后面的函数都起作用，看下面的情况：

```
> ank:=dsolve(diff(y(x),x,x)+diff(y(x),x)=0,y(x),se
  ries);
```

$$ank := y(x) = y(0) + D(y)(0)x - \frac{1}{2}D(y)(0)x^2 + \frac{1}{6}D(y)(0)x^3 - \frac{1}{24}D(y)(0)x^4 + \frac{1}{120}D(y)(0)x^5 - \frac{1}{720}D(y)(0)x^6 + \frac{1}{5040}D(y)(0)x^7 + O(x^8)$$

在求上面的微分方程时，我们并没有设阶数解的展开级数，系统也没按默认值展开，而是按照前面设定的值来展开，假如我们不知道原来设定了多少值，最好使用之前自己设定一次，即使是想用系统默认的值来展开。建议每次求微分方程的级数解的之前都设置展开级数的值。最后我们再来看一个例子：

【例 5-5】 求微分方程 $x^2\frac{d^2y}{dx^2}+x\frac{dy}{dx}+(x^2-v^2)y=0$（$v$ 不是整数）的级数解。这是一个柱函数方程，在数理方程中经常用到，不加限定地用 dsolve 函数我们能得到该方程的解是两个贝赛尔（Bessel）函数的线性组合：

```
> eq:='eq':
  eq:=x^2*diff(y(x),x,x)+x*diff(y(x),x)+(x^2-v^2)*y
  (x)=0;
```

$$eq := x^2\left(\frac{\partial^2}{\partial x^2}y(x)\right)+\left(\frac{\partial}{\partial x}y(x)\right)+(x^2-v^2)y(x)=0$$

```
> dsolve(eq,y(x),explicit);
```

$$y(x) = \_C1\,\mathrm{BesselJ}(v,x) + \_C2\,\mathrm{BesselY}(v,x)$$

BesselJ 和 BesselY 是两类贝赛尔函数，它的具体形式请参考相关书籍，这里就不再介绍了。下面我们用级数解法解出展开到5阶的解：

```
> Order:=5:
  dsolve(eq,y(x),type=series,'coeffs'=hypergeom');
```

$$y(x) = \_C1\,x^{v}\left(1+\frac{1}{-4v-4}x^2+\frac{1}{(-8v-16)(-4v-4)}x^4+O(x^5)\right) + \_C2x^{(-v)}\left(1+\frac{1}{4v-4}x^2+\frac{1}{(8v-16)(4v-4)}x^4+O(x^5)\right)$$

常微分方程的解析解就介绍到这儿，从本小节中，我们不但看到了如何利用 dsolve 求解各种各样的解，也看到了怎样利用 Maple 帮助我们用高等数学中介绍的方法来求某些特

殊的常微分方程。

5.1.2 常微分方程组的解析解

Maple 提供的函数 dsolve 不但能求微分方程，还能用来求微分方程组，它的用法跟用 solve 求方程组很相像，用来求微分方程组时的函数表达式如下：

dsolve({sysODE, Ics}, {funcs}, extra_args);

第一个参数是一个集合：sysODE 是待求的微分方程组中的各个微分方程的集合，Ics 是初始条件的集合；第二个参数也是一个集合，它用来设置微分方程中待求的函数；最后一个函数用来设置解的形式，比如设置解为隐式解。下面我们通过几个例子来学习常微分方程组的解法。

【例 5-6】求微分方程组：$\begin{cases}\dfrac{dx}{dt}=x+y+e^{-t}\\ \dfrac{dy}{dx}=y\end{cases}$ 的解析解。

还是跟原来相同，首先定义微分方程组，注意将方程组写成集合的形式：

```
> sysOde:='sysOde':
  sysOde:={D(x)(t)=x(t)+y(t)+exp(-t),D(y)(t)=y(t)};
```

$$sysOde := \{D(x)(t) = x(t) + y(t) + e^{(-t)}, D(y)(t) = y(t)\}$$

然后用函数 dsolve 直接求出方程组的解析解，并且解为显式解：

```
> dsolve(sxysode, {x(t), y(t)}, explicit);
```

$$\{y(t) = \_C2\, e^{t}, x(t) = e^{t}\, \_C2\, t - \frac{1}{2} e^{(-t)} + e^{t}\, \_C1\}$$

【例 5-7】求微分方程组：$\begin{cases}\dfrac{dx}{dt}=y-z\\ \dfrac{dy}{dt}=z\\ \dfrac{dz}{dt}=y\end{cases}$ 的解析解。

定义微分方程组，调用函数 dsolve 如下：

```
> sysOde:='sysOde':
  sysOde:={D(x)(t)=y(t)-z(t),D(y)(t)=z(t),D(z)(t)=y
  (t)};
```

$$sysOde := \{D(z)(t) = y(t), D(y)(t) = z(t), D(x)(t) = y(t) - z(t)\}$$

```
> dsolve(sysOde,{x(t),y(t),z(t)},implicit);
```

$$\{z(t) = \_C2\,e^{t} + \_C3\,e^{(-t)}, x(t) = 2\_C3\,e^{(-t)} + \_C1,$$
$$y(t) = \_C2\,e^{t} + \_C3\,e^{(-t)}\}$$

利用函数 dsolve，微分方程组的解析解很容易求出，有兴趣的读者可分析一下上面的解。常微分方程组的求解问题跟常微分方程的求解基本相同，没有什么新意，关于这方面的例子就介绍到这儿。

5.1.3　常微分方程定解问题的求解

一个微分方程在求解后往往是一个通解，有无穷多个解，但是这个方程所描述的物理过程的数量关系是唯一确定的，这就需要从微分方程的通解中找出所需要的解，为此，需要对微分方程附加某些条件，这就是所谓的定解条件，从微分方程中得到惟一确定的解的过程就是微分方程定解问题的求解。

定解问题的求解我们其实在上面已经碰到过，比如给定 a 点初始值求微分方程的级数解。在高等数学中，我们求微分方程的定解都是先解出方程的通解，然后再利用给定的初始条件确定通解中的积分常数，这样就得到了定解。现在用 Maple 也可以这么做，但是有时我们不必这么麻烦，我们可以利用 dsolve 函数的强大功能，直接求出微分方程的定解，函数求定解时的表达式如下：

dsolve({ODE, ICs}, y(x), extra_args):

上面的很多参数读者都已经很熟悉，可能就对参数 Ics 比较陌生，它是微分方程的初始条件，我们还是用一个例子来说明这个函数的用法：

【例 5-8】求微分方程 $\frac{1}{\sqrt{y}}y' + \frac{4x}{x^2-1}\sqrt{y} = x$，f(0)=4 的定解。

首先我们定义微分方程，然后利用 dsolve 的定解解法直接求出定解：

```
> eq:='eq':
  eq:=D(y)(x)/sqrt(y(x))+4*x*sqrt(y(x))/(x^2-1)=x;
```

$$eq := \frac{D(y)(x)}{\sqrt{y(x)}} + \frac{4x\sqrt{y(x)}}{x^2-1} = x$$

```
> sol1:=dsovle({eq, y(0)=4}, y(x), explicit);
```

$$sol1 := y(x) = \text{RootOf}(3\sqrt{\_Z}\,x^2 - 8\sqrt{\_Z} - x^4 + 2x^2 + 16)$$

这是一个用函数 RootOf 表示的解，关于它的具体表达的含义读者可以查看帮助，为了方便我们的阅读，我们用函数 allvalues 将其转换为我们熟悉的代数表达式：

```
> sol1:=allvalues(sol1);
```

$$sol1 := y(x) = \frac{1}{64}\frac{(-16 - 2x^2 + x^4)^2}{(x^2-1)^2}$$

下面我们利用先求微分方程的通解，再利用初始条件求出积分常数，最后得到微分方程的定解的方法来求微分方程的解。首先我们利用函数 dsolve 求出微分方程中的通解，由于我们不愿看到 RootOf 函数，所以这里采用隐式解：

```
> sol2:=dsolve(eq,y(x),implicit);
```

$$sol2 := \sqrt{y(x)} - \frac{\frac{1}{8}x^4 - \frac{1}{4}x^2 + \_C2}{x^2-1} = 0$$

然后我们用初始条件求出积分常数_C2:

```
> isolate(subs(x=0,y(0)=4,sol2),_C2);
```

$$\_C2 - \sqrt{4}$$

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

$$\_C2 = -2$$

最后将积分常数的值代入原方程的通解，得到微分方程的定解：

```
> isolate(subs(_C2=-2,sol2),y(x));
```

$$y(x) = \frac{\left(\frac{1}{8}x^4 - \frac{1}{4}x^2 - 2\right)^2}{(x^2-1)^2}$$

下面我们再来举一个例子来结束本小节。

【例 5-9】 求解微分方程 $\frac{d^2x}{dt^2} - 2\frac{dx}{dt} + x = 2te^t$，$x(0)=2$，$x'(0)=2$。

先来看看用直接求解的方法求出来的结果：

```
> eq:=diff(x(t),t,t)-2*diff(x(t),t)+x(t)=4*t*exp(t);
```

$$eq := \left(\frac{\partial^2}{\partial x^2}x(t)\right) - 2\left(\frac{\partial}{\partial x}x(t)\right) + x(t) = 4t\,e^t$$

```
> deslve({eq, x(0)=2, D(x)(0)=2}, x(t));;
```

$$x(t) = \frac{2}{3}t^3e^t + 2e^t$$

用这种方法来求解很简单，但我们还是希望先求出通解，具体过程如下：

用函数 dsolve 直接得到微分方程的通解：

```
> soll:=dsolve(eq,x(t),implicit);
```

$$soll := x(t) = \frac{2}{3}t^3e^t + \_C1e^t + C2te^t$$

代入 x(0)=2 得到一个关于积分常数的方程：

```
> eq1:=subs(t=0,x(0)=2,soll);
```

$$eq1 := 2 = \_C2e^0$$

我们顺理成章地想将一阶导数的初值代入得到另一个关于积分常数的方程，但是现在有一个问题，就是我们得到通解的时候，得到的只是一个等式，x(t)的值并不等于它的右边的形式，读者可以验证一下，因此在对 x(t)求导之前我们必须先将 x(t)设为一个解，这可以用函数 assign 来实现，如下：

```
> assign(soll);
```

```
> x(t);
```

$$\frac{2}{3}t^3e^t + \_C1e^t + C2te^t$$

执行完函数 assign 以后，我们执行 x(t);，可以看到现在 x(t)是一个解了，我们可以用求导函数对其求导：

```
> D(x)(t):=diff(x(t),t);
```

$$D(x)(t) := 2t^2e^t + \frac{2}{3}t^3e^t + \_C2e^t + \_C3e^t + \_C3e^t$$

我们以前说过函数 D 只是表示一个求导表达式，并不直接求出结果，在这里我们利用了这一点用来表达微分，下面将一阶导数初始值代入：

```
> eq2:=subs(t=0,2=%);
```

$$eq2 := 2 = \_C1e^0 + \_C2e^0$$

得到了两个关于积分常数的方程，我们就能用 solve 函数来求得两个积分常数的值，具体步骤如下：

```
> solve({eq1,eq2},{_C1,_C2});
```

$$\{\_C1 = 2, \_C2 = 0\}$$

最后将这两个值代入通解中得到微分方程的定解：

```
> x(t):=subs(_C1=2,_C2=0,rhs(sol1));
```

$$x(t) := \frac{2}{3}t^3e^t + 2e^t$$

5.1.4 常微分方程的数值解

高等数学中除了我们上面介绍过的几种特殊的常微分方程以外，大部分常微分方程都很难求出解析解，比如下面这个简单的一阶常微分方程：

$$dy / dx=\cos(xy)$$

假如我们用函数 dsolve 来求这个常微分方程的解析解的话，我们将很失望的看到，Maple 只是返回了一个空值：

```
> eq:='eq':
  eq:=diff(y(x),x)=cos(x*y(x));
```

$$eq := \frac{\partial}{\partial x}y(x) = \cos(x\,y(x))$$

```
> dsolve(eq,y(x));
```

这个时候我们就得用数值方法得到微分方程的数值解，在 Maple 里求解常微分方程的数值解还是可以用函数 dsolve，它的具体函数表达式如下：

dsolve(deqns, vars, numeric)
dsolve(deqns, vars, numeric, options)
dsolve(deqns, vars, type=numeric)
dsolve(deqns, vars, type=numeric, options)

参数 deqns 是待求的微分方程组和初始值的集合，当然也可以只是一个微分方程，vars 是待求的微分方程中的待求函数，参数 numeric 和 type=numeric 设置函数求出方程的数值解而不是解析解，参数 options 是用来设置函数求解的各种设置，是一个可选参数，有关具体含义读者可以参考联机帮助。在本小节中，我们经常用到的参数是 output=listprocedure 和 value=[a_1,a_2,..,a_n]，a_n 表示待求的数值点的自变量的值，前者表示让 dsolve 函数以列表加过程的方式输出，而后者则让函数输出给定的几个点的函数值，如果还不太理解，可以从下面这个例子中得到启发，所用的微分方程还是上面用到的很简单的方程。

【例 5-10】 求方程 $\frac{dy}{dx}=\cos(xy)$，$y(0)=2$ 在点 x=1,2,3,4,5 处 y 的值。

解这个微分方程的数值解最常见的有两种方法，第一种就是将 dsolve 函数的 options 参数设成 output=listprocedure，让方程输出一个列表过程形式，如下：

```
> eq:='eq':
  eq:=D(y)(x)=cos(x*y(x));
```

$$eq := \mathrm{D}(y)(x) = \cos(x\,y(x))$$

```
> g:=dsolve({eq,y(0)=2},(y(x),numeric,output=listpr
  ocedure);
```

$$g := [x = (\mathbf{proc}(x) \ \ldots\ \mathbf{end\ proc}), y(x) = (\mathbf{proc}(x) \ \ldots\ \mathbf{end\ proc})]$$

第一步是先定义一个变量，这个大家都很熟悉；第二步是利用 dsolve 函数求数值解的功能求出微分方程的近似解，这个解以过程来表示，并且将自变量和应变量组成一个列表来表示。在数值解的时候用到的参数 numeric 也可以用 type=numeric 表示，两者没有区别。第三步就应该将数值代入，求出微分方程在待求点的数值，我们为了一次输出多个点的数值，先定义了一个列表，然后用 map 函数一次求出所有的点，关于 map 函数的作用可以参看上一章。

```
> s:=[1,2,3,4,5];
```

$$s := [1,2,3,4,5]$$

```
> mat(g,s);
```

$$[[x(1)=1, y(x)(1)=2.293156367492477 98],$$
$$[x(2)=2, y(x)(2)=1.393274134981450 80],$$
$$[x(3)=3, y(x)(3)=.671501592905471289],$$
$$[x(4)=4, y(x)(4)=.426707149866473068],$$
$$[x(5)=5, y(x)(5)=.328723945685689634]]$$

第二种方法就是利用参数 value=[]的设置，在函数中直接输入需要求解的点，dsolve 将直接输出所求得的数值：

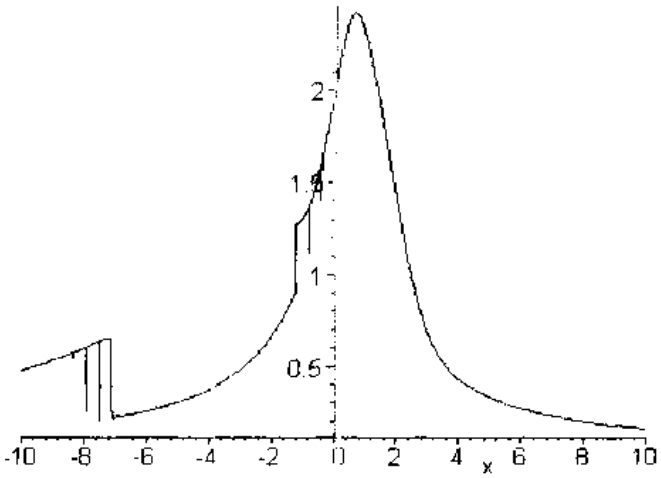
```
> dsolve({eq,y(0)=2},y(x),type=numeric,value=array
  ([1,2,3,4,5]));
```

	$[x, y(x)]$
1	2.293156550
2	1.393274240
3	.6715016131
4	.4267071505
5	.3287239457

如果只想知道随后的结果，显然第二种方法比较简单，是一个很好的选择。但假如我们还想观察微分方程的解的基本形状，则需要先解出过程，然后利用画图函数 plot 来画图，

这时第一种方法就是一个明智的选择，下面就让我们看看上面那个解的基本曲线。

```
> plot('rhs(g(x)[2])',x=-10..10);
```



我们之所以在 plot 函数的第一个参数 g(x)中用到[2]，是因为 g(x)表示的是一个列表，里面有自变量和其对应的应变量的值，我们在上面时已经注意到这一点了，而我们画图时只需要列表中 y 的值。

Dsolve 函数求数值解的功能不仅可以用于常微分方程的情况，还可以用于求解常微分方程组的数值解，它的用法跟求微分方程时完全相同，具体的过程我们还是来看一个实例：

【例 5-11】求常微分方程组 $\begin{cases} \dfrac{d^2y}{dt^2} = tx + y \\ \dfrac{dx}{dt} + \dfrac{dy}{dt} = tx\sin y \end{cases}$，$x(0)=2 \quad y(0)=2 \quad dy(0)/dx=3$ 在点 t=1，2，3，4，5 处 x 和 y 的值。

我们还是用先求过程再求数值的办法来求微分方程组的数值解，第一步还是先定义微分方程组和初始条件：

```
> eqs:='eqs':
  eqs:={diff(y(t),t,t)=x(t)*t+y(t),diff(x(t),t)+di
  ff(y(t),t)=t*x(t)*sin(y(t))};
```

$$eqs := \left\{ \left(\frac{\partial}{\partial t}\mathrm{x}(t)\right) + \left(\frac{\partial}{\partial t}\mathrm{y}(t)\right) = t\,\mathrm{x}(t)\sin(\mathrm{y}(t)), \frac{\partial^2}{\partial t^2}\mathrm{y}(t) = \mathrm{x}(t) + \mathrm{y}(t) \right\}$$

```
> inits:={x(0)=1,y(0)=2,D(y)(0)=3};
```

$$inits := \{\mathrm{x}(0)=1, \mathrm{y}(0)=2, \mathrm{D}(y)(0)=3\}$$

第二步将微分方程组和初始条件联合，用 dsolve 函数求出数值解：

```
> g:=dsolve(eqs union
  inits, {x(t),y(t)},numeric,output=listprocedure);
```

$$g := \left[t = (\mathbf{proc}(t) \ \ldots \ \mathbf{end\ proc}), x(t) = (\mathbf{proc}(t) \ \ldots \ \mathbf{end\ proc}), \right.$$

$$\left. y(t) = (\mathbf{proc}(t) \ \ldots \ \mathbf{end\ proc}), \frac{\partial}{\partial t} y(t) = (\mathbf{proc}(t) \ \ldots \ \mathbf{end\ proc}) \right]$$

第三步就是将所要求的点的自变量的值代入得到 x 和 y 的数值：

```
> s:='s':
  s:=[1,2,3,4,5]:
  evalf(map(g,s),5);
```

$$\left[\left[t(1) = 1., x(t)(1) = -3.0454, y(t)(1) = 6.4740, \left(\frac{\partial}{\partial t} y(t) \right)(1) = 6.2082 \right], \right.$$

$$\left[t(2) = 2., x(t)(2) = -7.4758, y(t)(2) = 12.902, \left(\frac{\partial}{\partial t} y(t) \right)(2) = 5.8919 \right],$$

$$\left[t(3) = 3., x(t)(3) = -22.487, y(t)(3) = 4.9812, \left(\frac{\partial}{\partial t} y(t) \right)(3) = -35.669 \right],$$

$$\left[t(4) = 4., x(t)(4) = 29.492, y(t)(4) = -44.924, \left(\frac{\partial}{\partial t} y(t) \right)(4) = -41.790 \right],$$

$$\left. \left[t(5) = 5., x(t)(5) = 34.157, y(t)(5) = -13.803, \left(\frac{\partial}{\partial t} y(t) \right)(5) = 138.15 \right] \right]$$

为了缩短输出的数值的长度，我们用函数 evalf 让其输出固定长度的，第二种方法的具体过程如下，很简单，我们只给出具体做法，不再进行说明：

```
> dsolve(eqs union
  inits,{x(t),y(t)},numeric,value=array(s));
```

$$\left[[t, x(t), y(t), \frac{\partial}{\partial t} y(t)] \right.$$

$$\left. \begin{bmatrix} 1 & -3.0491 & 6.4743 & 6.2083 \\ 2 & -7.4886 & 12.898 & 5.8728 \\ 3 & -22.151 & 4.9029 & -35.718 \\ 4 & 29.622 & -44.903 & -41.545 \\ 5 & 35.141 & -15.490 & 132.68 \end{bmatrix} \right]$$

5.2 偏微分方程

在实际情况中，我们不光要处理常微分方程，还要经常面对偏微分微分方程，比如弦的振动问题，热传导问题以及稳定问题都是典型的偏微分方程。偏微分方程在通常的情况下都得不到解析解，但对某些特殊的偏微分方程我们还是希望能够得到方程的解析解，或者方程解的具体形式。Maple 提供了一个函数 pdsolve 来处理微分方程解析解的求解，还提供了一个函数 PDEplot 用图形来描述微分方程的解析解的形状，在本节中，我们将重点来介绍这两个函数的使用，同时我们也将介绍对求解偏微分方程很有用的几个函数的具体用法。

5.2.1 偏微分方程的解析解

在开始本小节的开头我们先来看一个典型的偏微分方程，这是一个柔软的弦的横振动方程：$\frac{\partial^2 u}{\partial t^2} - a^2 \frac{\partial^2 u}{\partial t^2} = 0$。

这是一个多元二阶齐次偏微分方程，这种方程一般可用分离变量法来求得。用 Maple 可以直接用函数 pdsolve 来求解析解，我们下面先解出它的解析解，再来说明这个函数的具体用法：

```
> pde1:='pde1':
  assume(a,positive):
  pde1:=diff(u(x,t),t,t)-a^2*diff(u(x,t),x,x)=0;
```

$$pde1 := \left(\frac{\partial^2}{\partial t^2} u(x,t)\right) - a \sim^2 \left(\frac{\partial^2}{\partial t^2} u(x,t)\right) = 0$$

```
> pdsolve(pde1);
```

$$u(x,t) = \_F1(a \sim t + x) + \_F2(a \sim t - x)$$

上面我们使用了函数 pdsolve 多种用法中最简单的一种，下面我们来介绍一下此函数的具体用法，其表达式如下：

```
pdsolve(PDE);
pdsolve(PDE,f,HINT=...,INTEGRATE,build);
```

第一个参数 PDE 是偏微分方程表达式，它并没有像求常微分方程的函数 dsolve 那样，将第一个参数写成方程组的形式（复数形式，即 PDEs），主要是这个函数只能用来求偏微分方程，而对微分方程组无能为力；第二个参数 f 代表待求的函数形式，这跟函数 dsolve 中对应的参数是类似的；第三个参数 HINT=...用来设置函数输出结果的形式，比如可以将

输出结果设成两个独立的函数相乘的结果（如 f(x)*g(y)），或者两个函数相加的结果（f(x)+g(y)）以及 HINT=strip（这种参数只能在求一阶偏微分方程的解时有效，最后的结果将自变量和应变量用一个新的参数表示）等等，具体输出形式可根据具体方程和实际需要而定，这是一个可选参数；第四个参数 INTEGRATE 也是一个可选的参数，设置函数当待求的微分方程可以用分离变量来求时，自动将函数得到的常微分方程集合求解综合起来。第五个参数 build 设置函数将最后返回一个显式解。

函数 pdsolve 能够求解的方程形式是有限的，介入函数能够求出方程的解，则返回方程的解析解，否则返回为空。下面我们通过一个例子来熟悉这些参数的用法：

【例 5-12】求偏微分方程 $\frac{\partial u}{\partial x}-a\frac{\partial u}{\partial y}-bu=0$ 的解析解。

跟往常一样，我们先定义微分方程，为了保证参数 a 和 b 都是没有定义的，而只是一个常数，我们在定义开始处，将其本身赋值给自身：

```
> pde2:='pde2':
  a:='a':b:='b':
  pd2:=diff(u(x,y),x)-a*diff(u(x,y),y)-b*u(x,y)=0;
```

$$pde2 := \left(\frac{\partial}{\partial x}\mathrm{u}(x,y)\right) - a\left(\frac{\partial}{\partial x}\mathrm{u}(x,y)\right) - \mathrm{b}\,\mathrm{u}(x,y) = 0$$

然后我们先不设后面三个可选的参数，而是采用系统的默认设置来求偏微分方程的解析解，解的值赋给 sol1：

```
> sol1:='sol1':
  sol1:=pdsolve(pde2,u(x,y));
```

$$sol1 := \mathrm{u}(x,y) = \_\mathrm{F1}(y+ax)\,\mathrm{e}^{(bx)}$$

从结果中看，给出的解析解中包含一个函数，只是一个通解，只是偏微分方程的一般结果，假如偏微分方程没有确定的初始条件和边界条件或其他条件的限制，我们一般只能得到这种带任意函数的通解。为了验证这个解是否满足原方程，可以像求解常微分方程的解析解时一样，用 Maple 提供的函数来自动验证，验证偏微分方程的解的函数是 pdetest，它的用法跟 odetest 一模一样，具体表达式如下：

pdetest(sol,PDE):

第一个参数 sol 是待验证的微分方程的解析解，第二个参数 PDE 是待检验的偏微分方程。当 sol 是 PDE 的解是，函数返回 0，否则返回一个将解析解代入原偏微分方程后经化简的结果，下面我们用这个函数来验证解 sol1 是否是偏微分方程 pde2 的解析解：

```
> pdetest(sol1,pde2);
```

0

接下来我们来看看参数 HINT 的作用和用法，从上面已知，求出的解析解是多种多样的（因为有一个可以是任意的连续函数），假如我们想得到的函数解是一个关于自变量 x 的函数 f(x)和另一个关于自变量的函数 g(y)的乘积，即 u(x,y)=f(x)*g(y)的形式，则可以将参数 HINT 设成 HINT=f(x)*g(y)，如下：

```
> sol2:='sol2':
  sol2:=pdsolve(pde2,u(x,y),HINT=f(x)*g(y));
```

$$sol2 := (u(x,y) = f(x)g(y)) \text{ \& where } \left[\left\{\frac{\partial}{\partial y}g(y) = \frac{g(y) - c_1}{a} - \frac{g(y)b}{a}, \frac{\partial}{\partial x}f(x) = \_c_1 f(x)\right\}\right]$$

这个结果使用两个常微分方程来表示，没有给出明显的结果，为了得到我们熟悉的形式，我们可以用 Maple 提供的 build 函数将结果转换成显式解，这个函数是专门用来将偏微分方程的解转换成解析解的，它的函数表达式很简单，如下：

build(sol);

只有一个参数 sol，是偏微分方程的解。下面我们来看看用这个函数能将上面的解转换成何种形式：

```
> sol3:='sol3':
  with(PDEtools,build):
  sol3:=build(sol2);
```

$$sol3 := (u(x,y) = \frac{\_C1\, e^{(-c_1 x)} \_C2\, e^{\left(\frac{y c_1}{a}\right)}}{e^{\left(\frac{yb}{a}\right)}})$$

这个结果跟我们上面得到的解完全不同，不过这不用奇怪，因为我们对解的形式做了一定的限制，使原结果中那个可以是任意连续函数的部分变成了具体的形式，最终得到一个比较具体的解析解，如果对这个解不太放心，我们可以用 pdetest 来检验：

```
> pdetest(sol3,pde2);
                                    0
```

返回的结果明白无误的告诉我们 sol3 确实是偏微分方程 pde2 的解。其实要得到解 sol3 的形式，我们可以通过设置参数 build 来得到，它的作用跟函数 build 的作用是完全相同的：

```
> sol4:='sol4':
  sol4:=pdsolve(pde2,u(x,y),HINT=f(x)*g(y),build);
```

$$sol4 := u(x, y) = \frac{-C1\,e^{(-c(x))} - C2\,e^{\left(\frac{y-c_1}{a}\right)}}{e^{\left(\frac{yb}{a}\right)}}$$

显然解 sol4 和解 sol3 是完全一样的。假如我们想得到两个函数相加的结果，则可以将参数 HINT 设置成为 f(x)+g(y)的形式：

```
> sol5:=pdsolve(pde2,u(x,y),HINT=f(x)*g(y),build);
```

$$sol5 := u(x, y) = e^{(b\,x)} - C1 + \frac{-C2}{e^{\left(\frac{yb}{a}\right)}}$$

用 pdetest 检验结果：

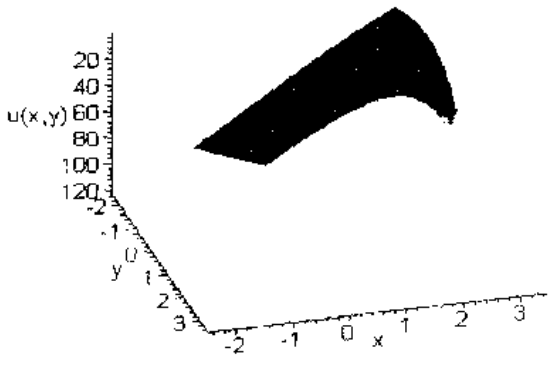
```
> pdetest(sol5,pde2);
                              0
```

我们已经用 pdsolve 函数和 build 函数求出来一个偏微分方程的解析解，并对所求得的解进行了验证。接下来我们来看看这个偏微分方程的解的形状是什么样的，我们用 Maple 提供的函数 PDEplot 来实现这个功能，其具体函数表达式如下：

PDEplot(PDE, inits, srange, options);

参数 PDE 是一个偏微分方程，inits 是偏微分方程的初始条件或者叫边界条件，srange 是初始条件中参数的取值范围，option 是一个可选参数，用来设置图像的形式，具体的设置可查看帮助，对方程 pde2 作图如下。

```
> with(PDEtools,PDEplot):
  a:=1:
  b:=1:
  PDEplot(pde2,u(x,y),[t,t,1/2*exp(3*t)],[t=0..1]);
```



为了画图方便，我们先给参数 a 和 b 分别赋值，然后假定微分方程的边界条件为 x=y=t 时，$u(x,y)=1/2e^{3t}$，（0<t<1）。为了使图形比较形象，这个图形是经过旋转的（在图上按住鼠标左键，然后移动鼠标就可以对图形进行旋转）。

使用 PDEplot 函数进行画图时要注意一点：这个函数的功能是有限的，它只能对一阶的偏微分方程进行作图，二阶或二阶以上的都不行，假如读者有兴趣，可以用这个函数对偏微分方程 pde1 进行作图试试。

下面再来看一个复杂一点的偏微分方程，并在求解过程中介绍一个转换偏微分方程形式的函数。

【例5-13】求偏微分方程 $a^2\frac{\partial^2 f}{\partial x^2}+2b\frac{\partial^2 f}{\partial y\partial x}+c\frac{\partial^2 f}{\partial y^3}=0$ 的解析解。

首先用函数 pdesolve 直接解出偏微分的解析解，如下：

```
> pde3:=a*diff(f(x,y),x,x)+2*b*diff(f(x,y),y,x)+c*di
  ff(f(x,y),y,)=0;
```

$$pde3:=a\left(\frac{\partial^2}{\partial x^2}f(x,y)\right)+2b\left(\frac{\partial^2}{\partial y\partial x}f(x,y)\right)+c\left(\frac{\partial^2}{\partial y^2}f(x,y)\right)=0$$

```
> pdsolve(pde3);
```

$$f(x,y):=\_F2\left(\frac{1}{2}\frac{(2\sqrt{b^2-ac}-2b)x}{a}+y\right)+\_F1\left(\frac{1}{2}\left(4\frac{\sqrt{b^2-ac}\,b)}{4b^2-4ac}+1\right)x-\frac{1}{2}\frac{ya}{\sqrt{b^2-ac}}\right)$$

要求出这种微分方程的解析解用 pdsolve 很方便，但有时我们不光想知道解的形式，还想知道怎样求解，这时可以利用 Maple 提供的函数 mapde，如果可能的话，这个函数能将给定的偏微分方程转换成我们所需要的比较简单的形式，下面我们就来介绍这个函数，并用它将上面那个偏微分方程进行化简，函数的具体表达式如下：

```
mapde(PDE,into);
mapde(PDE,into,f);
```

参数 PDE 是待转换的偏微分方程；参数 f 是一个可选的参数，设置将微分方程转换后的替代函数；into 是想要转换成的形式，具体的参数选择如下：

noF	将偏微分方程转换成不含待求函数的形式
ccoeff	将偏微分方程转换成常系数偏微分方程

canom　　将偏微分方程转换成只有一个混合导数的经典形式

canop　　将偏微分方程转换成不含混合导数的经典形式

函数 mapde 如果能够将给定的偏微分方程按要求的形式转换，则将返回一个转换后的偏微分方程，否则返回原方程，并给出不能转换的信息，我们下面就分别让函数转换成 4 种形式，显然偏微分方程已经是常系数且不含待求函数的形式了，我们不需要用前两个参数进行转换，如果强行转换将仍然返回原微分方程表达式，并附加错误信息，如下：

```
> with(PDEtools,mapde):
  mapde(pde3,noF);
```

$$\left\{a\left(\frac{\partial^2}{\partial x^2}f(x,y)\right)+2b\left(\frac{\partial^2}{\partial y\partial x}f(x,y)\right)+c\left(\frac{\partial^2}{\partial y^2}f(x,y)\right)=0\right\}\ \&\ \text{where}$$

the received pde has not the indeterminate function

下面我们将 pde3 转换成只包含一个混合导数和不包含混合导数的情形，并对新的偏微分方程进行求解：

```
> pde4:=mapde(pde3,canom);
```

$$pde4:=\sqrt{4b^2-4ac}\left(\frac{\partial^2}{\partial\_\xi2\,\partial\_\xi1}f(\_\xi1,\_\xi2)\right)\ \&\ \text{where}\left\{\_\xi1=\frac{1}{2}\left(4\frac{\sqrt{4b^2-4acb}}{4b^2-4ac}+1\right)x-\frac{1}{2}\frac{ya}{\sqrt{b^2-ac}},\ \_\xi1=\frac{1}{2}\frac{(-2b+2\sqrt{b^2-ac})x}{a}+y\right\}$$

用函数 mapde 进行化简以后，偏微分方程变成一个很简单的形式，下面的解也就是很显然的了：

```
> sol6:=pdsolve(op(1,pde4),build);
```

$$sol6:=f(\_\xi1,\_\xi2)=\_F2(\_\xi1)+\_F1(\_\xi2)$$

由于转换后的偏微分方程 pde4 的后面是用列表表示的，因此调用的时候得先用 op 函数函数转换一下。同样的道理，我们在下面求转换成不含混合导数的新的偏微分方程的时候也用 op 函数转换。

```
> pde5:=mapde(pde3,canop);
```

$$pde5 := \left(\left(-\frac{b^2}{a}+c\right)\left(\frac{\partial^2}{\partial \_\xi 1^2}f(\_\xi 1,\_\xi 2)\right)+a\left(\frac{\partial^2}{\partial \_\xi 2^2}f(\_\xi 1,\_\xi 2)\right)\right)$$

$$\&\ \text{where}\left\{\_\xi 2 = x, \_\xi 1 = y-\frac{bx}{a}\right\}$$

```
> sol7:=pdsove(op(1,pde5),build);
```

$$sol7 := f(\_\xi 1,\_\xi 2) =$$

$$\_F2\left(\frac{\sqrt{-a\left(-\frac{b^2}{a}+c\right)}\,\xi 1}{-\frac{b^2}{a}+c}+\_\xi 2\right)-\_F1\left(\frac{1}{2}\_\xi 1-\frac{1}{2}\frac{\_\xi 2\left(-\frac{b^2}{a}+c\right)}{\sqrt{-a\left(-\frac{b^2}{a}+c\right)}}\right)$$

到目前为止，我们已经基本熟悉了偏微分方程的解法，读者可能有一个感觉就是，用 pdsolve 函数来求解偏微分方程跟 dsolve 函数求解常微分方程是一模一样的，表面上的确如此，但实际上函数 pdsolve 是有很多缺陷的，它并不像函数 dsolve 那样功能强大，它只能处理最简单的几种偏微分方程，对于有些方程，我们必须进行转换用函数 dsolve 来求解。我们还是通过一个例子来说明这个问题。

【例 5-14】 求偏微分方程 $(\frac{d^2f}{dx^2})^2 - f\frac{df}{dy} = 0$ 的解析解。

我们先定义偏微分方程，并用函数 pdsolve 来求解：

```
> pde6:=diff(t(x,y), x, x)^2-f(x,y)*diff(f(x,y),y=0;
```

$$pde6 := \left(\frac{\partial^2}{\partial x^2}f(x,y)\right)^2 - f(x,y)\left(\frac{\partial}{\partial x}f(x,y)\right)=0$$

```
> pdsolve(pde6);
```

我们很失望的发现函数 pdsolve 只返回一个空值，观察函数可发现这是一个很简单的偏微分方程，在数理方程中我们好像就求解过这样的偏微分方程，所以我们应该能找到一种方法来求解这个偏微分方程。下面我们就来介绍一种求这种偏微分方程的方法——分离变量法。

分离变量法的第一步是将原待求函数分离成两个或两个以上的函数，每个函数的变量个数要比原函数少，并且一个变量只能在一个函数里，比如上面这个偏微分方程我们就能假定 f(x,y)=X(x)*Y(y)，将原待求函数 f(x,y)分解成两个各带一个变量的函数 X(x)和 Y(y)：

```
> X:='X':Y:='Y':
  X:=X->X(x):
  Y:=Y->Y(y):
  pde7:=expand(subs(f(x,y)=X(x)*Y(y),pde6));
```

$$pde7 := \left(\frac{\partial^2}{\partial x^2}\mathrm{X}(x)\right)^2 \mathrm{Y}(y)2 - \mathrm{X}(x)^2\,\mathrm{Y}(y)\left(\frac{\partial}{\partial x}\mathrm{Y}(y)\right) = 0$$

我们看到如果将方程移项并两边都各除以(X(x)*Y(y))^2 的话，方程两边将是各含 x 或 y 的函数：

```
> pde8:=lhs(pde7)+X(x)^2*Y(y)*diff(Y(y),Y)=rhs(pde7)
  +X(x)^2*Y(y)*diff(Y(y),y);
```

$$pde8 := \left(\frac{\partial^2}{\partial x^2}\mathrm{X}(x)\right)^2 \mathrm{Y}(y)2 = \mathrm{X}(x)^2\,\mathrm{Y}(y)\left(\frac{\partial}{\partial x}\mathrm{Y}(y)\right)$$

```
> pde8/(X(x)*Y(y))^2;
```

$$\frac{\left(\frac{\partial^2}{\partial x^2}\mathrm{X}(x)\right)^2}{\mathrm{X}(x)2} = \frac{\frac{\partial}{\partial x}\mathrm{Y}(y)}{\mathrm{Y}(y)}$$

因为方程的两边是关于不同变量的函数，它们的值相等，那么唯一的可能性就是它们都等于某一个常数，我们不妨设成是 a^2，则我们能得到下面两个常微分方程：

```
> eq1:=lhs(%)=a^2;
```

$$eq1 := \frac{\left(\frac{\partial^2}{\partial x^2}\mathrm{X}(x)\right)^2}{\mathrm{X}(x)^2} = a^2$$

```
> eq2:=rhs(%%)=a^2;
```

$$eq2 := \frac{\frac{\partial}{\partial y}\mathrm{Y}(y)}{\mathrm{Y}(y)} = a^2$$

这两个方程都是很简单的常微分方程，我们很容易就能用 dsolve 函数求出它们的解析解：

```
> sol9:=dsolve(eq1);
```

$$sol9 := \mathrm{X}(x) = \_C1\,e^{(\sqrt{a}x)} + \_C2\,e^{(-\sqrt{a}x)},\quad \mathrm{X}(x) = \_C1\sin(\sqrt{a}\,x) + \_C2\cos(\sqrt{a}\,x)$$

```
> sol10:=dsolve(eq2);
```

$$sol10 := \mathrm{Y}(y) = \_C1\, e^{(a^2 y)}$$

最后只需将这两个解相乘就能得到原方程的解析解，这一步很简单，读者可以自己完成，注意一点：这个方程有两个不同形式的解。

从上面这个例子可以看出，虽然 pdsolve 函数功能还算比较强大，但它并不是无所不能，当这个函数不能求解我们的偏微分方程时，建议观察一下原方程，看看能不能用别的方法来求解。

第6章　金融数学专题

随着市场经济的发展，股票、期货交易逐渐走进普通人的生活，分期付款的购房、购车方式，以及国库券、债券甚至一定程度的奖券投资，无处不是金融数学的领域。虽然只涉及其中最简单的问题，如复利计算、现值计算、分期付款额度等等，但所使用的公式却并不是所有的人都了解的。Maple 为帮助人们处理日常的金融数学问题，专门设计了一些函数并集中在 Finance 程序包中，方便用户的使用。

本章将通过一些常见实例来介绍 Finance 程序包中的函数，同时还会给出这些问题的基本计算公式以及它们的代数解法，方便读者了解 Maple 函数的实际计算过程。

6.1　复利计算

【例 6-1】假设存入银行￥12500，年利率 7%，按天记息，分别经过了 t=0，5，10，15，20，25，30 年后，获得的利息及总金各是多少？

复利的计算公式为：

假设总投资额为 P，年利率 r，每年记息 m 次，则投资 t 年后的总资金 A(t)为：

$$A(t)=P(1+\frac{r}{m})^{mt}$$

Maple 的计算过程为：

```
> actual:='acutual';interest:='interest';
```

$$actual := acutual$$

$$interest := interest$$

```
> actual:=t->12500*(1+0.07/365)^(365*t);
```

$$actual := t \rightarrow 12500\ 1.00019178^{(365t)}$$

```
> interest:=t->actual(t)-12500;
```

$$interest := t \rightarrow \text{actual}(t) - 12500$$

```
> Time:=seq(5*n,n=0..6);
```

$$Time := 0,5,10,15,20,25,30$$

```
> T_array:=array([seq([t,actual(t),
  interest(t)], t=Time)]):
```

```
> print(T_array);
```

$$\begin{bmatrix} 0 & 12500. & 0. \\ 5 & 17737.75488 & 5237.75488 \\ 10 & 25170.23585 & 12670.23585 \\ 15 & 35717.07790 & 23217.07790 \\ 20 & 50683.26182 & 38183.26182 \\ 25 & 71920.58199 & 59420.58199 \\ 30 & 102056.7723 & 89556.7723 \end{bmatrix}$$

如果用户需要处理大量的类似问题，重复输入公式无疑是很繁琐的。因此我们可以编写自己的函数来直接完成上述过程。下一章会详细介绍如何利用 Maple 编程，此处仅为用户提供一个感性的认识：

```
> actual:=(t,P,r,m)->P*(1+r/m)^(m*t):
> interest:=(t,P,r,m)->actual(t,P,r,m)-P:
> result:=proc(Time,P,r,m)
          array([seq([t,actual(t,P,r,m),
                    interest(t,P,r,m)],t=Time)])
> end:
> result([0,4,5,7],10000,0.12,365);
```

$$\begin{bmatrix} 0 & 10000. & 0. \\ 4 & 16158.98508 & 6158.98508 \\ 5 & 18218.70903 & 8218.70703 \\ 7 & 23159.25795 & 13159.25795 \end{bmatrix}$$

```
> result([1,3,5],23,0.6,12);
```

$$\begin{bmatrix} 1 & 41.30469550 & 18.30469550 \\ 3 & 133.2117711 & 110.2117711 \\ 5 & 429.6212755 & 406.6212755 \end{bmatrix}$$

这里提供的程序只完成了计算复利的基本功能，在阅读完第 7 章“Maple 6 程序设计”后，读者可以将此程序修改得更加完善，比如加入缺省的年利率按日、月、季度记息，每 n 年计算一次结果等等。实际上 Maple 各个程序包中的函数，基本都是以系统缺省函数为基础如此编写出来的，用户甚至可以编写自己的程序包，来整和有针对性的问题，相关内容详见第 7 章。

在金融程序包中，Maple 提供了函数 futurvalue 来计算此类问题。它的完全形式为：

futurevalue(amount,rate,nperiods)

amount 对应总的投资金额，rate 对应单次记息利率，nperiods 对应记息次数。由于这些参数在大部分 finance 程序包中的函数中都会涉及，所以在今后介绍其他函数时，我们不再重复这些参数的意义。利用 futurevalue 函数，我们可以完成上例中同样的功能：

```
> T_table:=array([seq([t,futurevalue(12500,
  0.07/365,365*t)], t=seq(5*i, i=0..8))]);
```

$$T\_table := \begin{bmatrix} 0 & 12500. \\ 5 & 17737.75488 \\ 10 & 25170.23585 \\ 15 & 35717.07790 \\ 20 & 50683.26182 \\ 25 & 71920.58199 \\ 30 & 102056.7723 \\ 35 & 144820.6409 \\ 40 & 205503.4424 \end{bmatrix}$$

读者可以看出，相对于用户的直接输入，使用 finance 程序库的惟一好处在于不需要用户键入原始公式，而可以直接输入参数计算结果。但用户享受这种别便捷的前提是要确定选择了正确的库函数，否则这种傻瓜型的函数反而会影响用户获得预期结果。

在 finance 程序库中，同计算复利相关的还有一个函数："effectiverate"。它可以计算复利所对应的实际单期利息。比如在例 6-1 中提到，年利率 7%，按日记息，则实际的日息并非简单的"年息/365"，利用 effectiverate 可以告诉我们正确的日息是多少：

```
> effectiverate(0.07,365);
                        .072501053
> (1+0.07/365)^365-1;
                        .072501053
> effectiverate(0.1,12);
                        .104713063
> (1+0.1/12)^12-1;
                        .104713063
```

第二条命令给出了实际日息的计算公式。

相对于 futurevalue 函数，读者会发现 finance 程序库中还有一个类似的 presentvalue 函数。它的功能与处理的问题正好同 futurevalue 函数相反。例如我们将例 6-1 的事例反过来：

【例 6-2】如果最终想从银行中提取¥12500，年利率 7%，按天记息，则如果分别提前 5，10，15，20，25，30 年存钱，各需要存入多少？

presentvalue 函数的完全形式同 futurevalue 完全一样，即：

presentvalue(amount,rate,nperiods)

请看如何利用它计算此例：

```
> array([seq([t,presentvalue(12500,0.07/365
  ,365*t)],t=seq(5*i,i=0..8))]);
```

$$\begin{bmatrix} 0 & 12500.00000 \\ 5 & 8808.393860 \\ 10 & 6207.728880 \\ 15 & 4374.657984 \\ 20 & 3082.871827 \\ 25 & 2172.535256 \\ 30 & 1531.010598 \\ 35 & 1078.920788 \\ 40 & 760.3278962 \end{bmatrix}$$

读者会发现，我们仅是将例 6-1 中使用的 futurevalue 函数简单替换为 presentvalue 就获得了相应的结果。

6.2 按年支付问题

【例 6-3】（1）计算在存入多少钱的情况下，才能在年利率 75%，按年记息时，每年提取￥45,000，连续支取 35 年；（2）计算在存入多少钱的情况下，才能在年利率 8%，按年记息时，每年提取￥20000+￥5000*k，连续提取 35 年。k=0，1，2，3，4，5。

现值问题的计算公式为：

假设现值为 P，年利率 j，年支付（投资）额为 R，总共支付（投资）n 年，则：

$$P = R\,\frac{1-(i+j)^{-n}}{j}$$

Maple 的计算过程为：

```
> r:='r';j:='j';n:='n';
```

$$r := r$$

$$j := j$$

$$n := n$$

```
> present:=(r,j,n)->r*((1-(1+j)
```

$$present := (r, j, n) \rightarrow \frac{r(1-(1+j)^{(-n)})}{j}$$

```
> present(45000,0.075,40);
```

566748.3899

计算第二问时，我们采取与例一相同的方式，即使用数组来表示最后的结果。数组的第一列代表每年所提取的金额，第二列对应所须存入的总金额。

```
> array([seq([20000+5000*k,present(20000+
  5000*k,0.08,35)],k=0..5)]);
```

20000	233091.3644
25000	291364.2054
30000	349637.0465
35000	407909.8876
40000	466182.7286
45000	524455.5698

从下例中，可以看出现值问题的另一种表达形式：分期付款问题。

【例 6-4】如果从今年年底，每年付款￥300，000 购买一幢别墅，15 年付清，其土地以年 10%的价值增值，不考虑银行的利息，则此幢别墅折合现价为多少钱？

这里我们使用 finance 程序库中的 annuity 函数来计算，它的完全形式为：

annuity(cash, rate, nperiods)

cash 代表每年所支付的金额，其余各个参数的意义同 presentvalue 函数。可以输入如下命令计算上例中的问题：

```
> annuity(300000,0.1,15);
```

.2281823852 10^7

即最后结果为 0.2281823852×10^7，将近 230 万元。如果我们使用 interface 函数，则可以查看 annuity 函数是如何定义的，这样可以让我们提前了解一些有关 Maple 系统库函数的生成方法。如：

```
> interface(verboseproc=2);
```

```
> readlib(annuity);
proc(Cash, Rate, Nperiods)
option
`Copyright(c) 1994 by Je' ro^ me M. Lang.All rights reserved.`;
description `Present value of an annuity paying Cash per period\
for Nperiods periods`;
    Cash * (1 / Rate - 1 / (Rate * (1 + Rate)^ Nperiods))
end proc
```

从显示的结果中我们会发现，其实这个函数的实际内容十分简单，只包含了一个计算公式。其余的均是对函数的说明及版权说明。因此读者完全可以依照此结构定义新的函数来方便自己的工作。

我们再将此方法使用在 presentvalue 函数上，会得到结果：

```
> readlib(presentvalue);
proc(Amount, Rate, Nperiods)
option`Copyright(c) 1994 by Je' ro^ me M. Lang.All rights reserved.`;
description `Present value of Amount given at Nperiods with Rate`;
    Amount / finance[' futurevalue'](1, Rate, Nperiods)
end proc
```

发现 presentvalue 函数是利用 futurevalue 函数间接计算而获得的结果。由这里获得的结果，我们可以得到计算 presentvalue 的公式：

$$P(t)=A(1+\frac{r}{m})^{mt}$$

利用此方法，还能获得所有 finance 程序库中函数所使用的公式，用户可以如此来验证这些函数是否满足具体问题的要求。

6.3 现金流动及相关问题

【例 6-5】如果从今年年底开始，会按照￥5000，￥15000，￥25000，￥30000 的顺序连续四年在每年年底获得收益，当折现率在这四年中保持 10%不变时，问这种收益相当于年初一次性获得多少现金？

这就是典型的现金流动问题。由于每年末的现金收益没有规律，我们无法把它化简为一个简单的公式表示出来，但我们可以利用列表这种数据结构按照固定标准统一处理每年的收益，从而编写出一个程序来化简计算过程。在使用了 6.1 节介绍的 presentvalue 函数后，可以编写出如下过程来处理此例：

```
> cash:=proc(list,rate)
>       local total, i;
>       total:=0;
>       for i to nops(list) do
>          total:=total+
                presentvalue(list[i],rate,i)
>       end do;
>       total
> end:
```

```
> cash([5000,15000,25000,30000],0.1);
                          56215.42244
```

计算结果为此种收益方式相当于年初一次获得收益约￥56215。

如果读者觉得不保险，可以利用 finance 程序库中的 cashflows 函数，完成同样的功能。函数的完全形式为：

cashflows(flow, rate)

flow 对应的是每年收益的列表对象。使用方法为：

```
> flow:=[5000,15000,25000,30000];
                flow := [5000,15000,25000,30000]
> cashflows(flow,0.1);
                          56215.42244
```

我们看一下 cashflows 函数的原始定义：

```
> readlib(cashflows);
proc(cashflows::list,Rate)
local ans,i;
option `Copyright(c) 1994 by Je' ro^ me M. Lang.All rights reserved.`;
description `Present value of a list of cash flow,one item per period`;
    ans := 0;
    for i to nops(cashflows) do
        ans := ans + finance[' presentvalue'](cashflows[i], Rate, i)
    end to;
    ans
end proc
```

可以发现它定义的过程同前边自己定义的过程几乎完全一样。说明之前我们使用的模型是正确的。

既然是这么简单的一个过程函数，我们还可以倒过来求解其中的参数，从而解决另一种相关的问题。比如下例：

【例 6-6】一项计划的初期一次性投资为￥100000，如果预期的其后 5 年收益顺序为￥10000，￥10000，￥25000，￥40000，￥50000，则只有折现率为多少时，才能保证五年后总收益等于总投资？

在设定折线率为 r 后，利用 Maple 可以按照如下步骤解决这个问题：

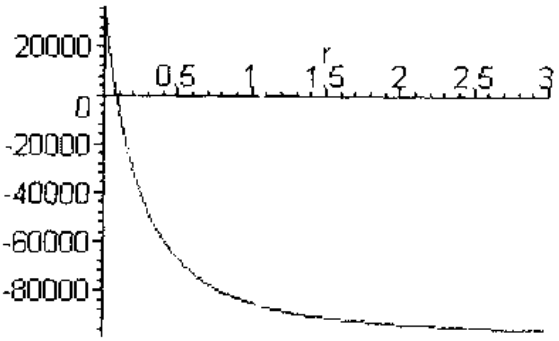
```
[> totals:='totals':r:='r':

> total:=-100000+cashflows([10000,10000,
  25000,40000,50000], r);
> fsolve(total=0, r);
```

$$total := -100000+\frac{10000}{1+r}+\frac{10000}{(1+r)^2}+\frac{25000}{(1+r)^3}+\frac{40000}{(1+r)^4}+\frac{50000}{(1+r)^5}$$

$$.08322913454$$

由此可以得知，预期折现率约为 8.32%。利用 plot 函数，我们可以观察折现率与总资金的关系：

```
> plot(total,r=0..3);
```



如果在现金流动问题中，每年的收益是有规律的，则可以转化为新的一种形式，如例 6-7 所示：

【例 6-7】一项投资，今年年末会获得收益￥25000，以后预期收益会以每年 5%的速度增长。如果年利率保持 11%不变，则如此连续获得收益 6 年，相当于今年一次性获得多少资金？

利用 growingannuity 函数，可以方便的计算此例。此函数的完全形式为：

growingannuity(cash, rate, growth, nperiods)

其中 growth 代表收益的年增长率。例 6-7 可以通过如下命令计算结果：

```
> growinganmuity(25000,0.11,0.05,6);
                    118137.5611
```

即如此六年的连续收益相当于一次性获得约￥11800。我们还可以利用前两节介绍的函数来计算由此相当的收益。如：

（1）计算六年中每年年末的实际收益：

```
> annualcash:=[25000,25000*1.05,25000*1.05^2,
  25000*1.05^3,25000*1.05^4,25000*1.05^5];
annualcash := [25000,26250.00,27562.5000,28940.62500,
  30387.65625,31907.03905
```

还可以用 futurevalues 来生成如上的序列：

```
> annualcash:=[seq(futurevalue(25000,0.05,i),
  i=0..5)];
annualcash := [25000,26250.00,27562.5000,28940.62500,
  30387.65625,31907.03905
```

注意使用 futurevalue 函数时，变量 i 的取值范围是 0~5，而不是一般认为的 1~6，这是因为第一年的收益是没有增长的，所以第一个 i 值必须取 0。

（2）利用 cashflows 计算相当的一次性收益：

```
> cashflows(annualcash,0.11);
                    118137.5609
```

由此获得了同使用 growingannuity 函数相同得结果。

让我们在例 6-7 的基础上，增加一个条件，使得问题变的相对更复杂：

【例 6-8】一项投资，今年年末会获得收益￥25000，以后预期收益会以每年 5%的速度增长。如果年利率保持 11%不变，则如此会连续获得收益 6 年。6 年后，收益的增加速度会变成 8%，如此会保持收益 5 年。求这 11 年的收益相当于今年年初一次性获得多少收益？

（1）按照例 6-7 计算同前六年收益相当的一次性收入：

```
> part1:=growinganmuity(25000,0.11,0.05,6);
                part1 := 118137.5611
```

（2）计算第六年末的收益：

```
> cash_6:=futurevalue(25000,0.05,5);
```

$$cash\_6 := 31907.03905$$

（3）计算改变收益增加率后第一次的收益：

```
> cash_7:=futurevalue(cash_6,0.08,1);
```

$$cash\_7 := 34459.60217$$

（4）计算由此时算起的收益，相当于第六年末时的价值：

```
> part2_6:=growingannuity(cash_7,0.11,0.08,5);
```

$$part2\_6 := 147056.6984$$

（5）计算这部分价值相当于现在的价值，同时将两部分收益对应的价值相加，得到最终结果：

```
> part2:=presentvalue(part2_6,0.12,6);
```

$$part2 := 74503.49999$$

```
> total:=part1+part2;
```

$$total2 := 192641.0611$$

实际上由于此例中投资的收益率小于现金的折现率，所以会出现每年收益与现价相比逐渐减小的情况。我们从 growingannuity 函数的原始定义函数中也会观察出同样的结果。如：

```
> readlib(growingannuity);
proc(Cash,Rate,GrowthRate,Nperiods)
option
 `Copyright(c)1994 by Je' ro^ me M. Lang.All rights reserved.`;
description `Present value of an annuity giving Cash per period\
for Nperiods period growing at GrowthRate`;
    if Rate = CrowthRate then  Cash * Nperiods / (1+GrowthRate)
     else  Cash * (1 - ((1+GrowthRate) / (1+ Rate))^ Nperiods) / (
         Rate - GrowthRate)
     end if
end proc
```

当收益增长率(growthrate)同折现率(rate)不相同时，growingannuity 的返回值为：

$$\text{total} = \text{Cash} * \left(\frac{1 - (\frac{1 + \text{GrowthRate}}{1 + \text{Rate}})^{\text{Nperiods}}}{\text{Rate} - \text{GrowthRate}} \right)$$

所以当收益增长率小于折现率时，返回值会有极值：

$$total = \frac{Cash}{Rate - GrowthRate}$$

在 Maple 中，定义了函数 growingperpetuity 来计算这类极值问题。它的完全形式为：

growingperpetuity(cash, rate, growth)

请看如下的例子：

【例 6-9】一个公司的股票，每年末会分红。第一年预期持股者每股收益为￥3.0，以后预期每股收益以年 5%的增长速度递增。如果折现率保持年 9%，问每股总收益相当于现价多少？

利用 Maple 的一条命令，我们就可以解决这个问题：

```
> growingperpetuity(3.0,0.09,0.05);
                          75.00000000
```

6.4　分期付款问题

分期付款这种形式的付款方式已经逐渐走进了平常人的生活。这种付款方式的最简单形式，可以归结为：一次性借贷 P，每次记息利率为 j，每次记息时付款，分 n 次，则每次需要支付金额 R 满足以下公式：

$$R = \frac{Pj}{1 - (1 + j)^{-n}}$$

请看如下的一个例子：

【例 6-10】如果一次性借贷￥75000，年利率 9.5%，按月记息，按月付款，20 年付清，则每月的支付金额为多少？

利用上述的公式，可以计算出此例的问题：

```
> amort:='amort';
```

$$amort := (p, j, n) \rightarrow \frac{Pj}{1 - (1 + j)^{(-n)}}$$

```
> amort:=(p,j,n)->n*amort(p,j,n)-p;
```

$$totintpaid := (p, j, n) \rightarrow n\,\text{amort}(p, j, n) - p$$

```
> amort(75000,0.095/12,20*12);
                        699.0983810
> 240*%;
                        167783.6114
> totintpaid(75000,0.095/12,20*12);
                        92783.6114
```

利用 amort 函数，我们求出了例 6-10 所问的问题。将它乘以 240，就得出了总共所需支付的金额。totintpaid 函数给出了相对于一次性支付，总共多付出的资金。

我们这里定义的函数 amort 只能针对这些简单的问题，Maple 提供的 amortization 函数可以给出此类问题更具体的解。amortization 函数的完全形式为：

amortization(amount, payments, rate, nperiods)

参数的意义同之前的函数类似，amount 对应总的借款金额，payments 对应每次付款金额，rate 对应利息，nperiods 为可选选项，如果用户没有输入，则系统自动计算到总还款金额等于借款金额后结束。比如我们看以下的例子：

【例 6-11】一次性借款￥75000，在年利率 9.5%，按月记息的情况下，每月还款￥2000，建立一个还款计划，计算需要多久才能全部付清？

此例是标准的使用 amortization 函数的例子，将其全部参数代入函数中后，会得到入下结果：

```
> time_table:=amortization(75000,1000,.095/12):
> op(time_table[1][2..9]);
[1,1000,593.7500000,406.2500000,74593.75000],
 [2,1000,590.5338542,409.4661458,74184.28385],
 [3,1000,587.2922472,412.7077528,73771.57610],
 [4,1000,584.0249775,415.9750225,73355.60108],
 [5,1000,580.7318419,419.2681581,72936.33292],
 [6,1000,577.4126356,422.5873644,72513.74556],
 [7,1000,574.0671524,425.9328476,72087.81271],
 [8,1000,570.6951840,429.3048160,71658.50789]
```

由于显示所有的列表所占篇幅过多，此处只显示时间表的头几项。amortization 函数返回的第一项，是一个类似时间表的列表类型数据，或者形象的说是一个二维数组。数组的行向量代表每次返还的信息，按顺序分别为："[返还次数，本次支付金额，截止到本次支付前本金增加的利息，刨去利息本次所支付的本金金额，支付后剩余未付金额]"，从这个时间表中，用户可以非常清楚的了解对每次付款产生的效果。

在 Maple 6 中，对 amortization 的返回值做了修改。在它的最后一项，增加了对总返还利息的显示。由于此列表的倒数第二项对应总支付次数，因此，显示最后两项数据，就可以对此种分期付款方案获得直观的了解，如：

```
> nops(time_table[1]);
                              116
> op(time_table[1][116]);
            115,234.0708976,1.838506430,232.2323912,0
> op(time_table[2]);
                          3923407092,-5
```

这里我们先用 nops 命令获得时间表的总项数，再利用 op 命令显示出最后一项的内容。从中就可以知道按这种分期付款方式，一共需要支付 115 个月，即 9 年零 7 个月。最后一次只需要支付金额约￥234。利用 op 命令显示时间表的第二组参数，得到的就是本方案共支付利息约￥39234（由于 Maple 的显示格式问题，它将实际数字表达成 3923407092 $\times 10^{-5}$）。

amortization 函数还会针对不同的分期付款方式，给出不同的计算结果，以方便用户选择适合的方法，请看对例 6-12 的变型：

【例 6-12】 一次性借款￥75000，在年利率 9.5%，按月记息的情况下，每月还款￥2000+当月所增长利息，按照此计划，需要多久才能全部付清？

我们在 amortization 函数中将“payments”一项的参数略做修改，就可以求出此问题的解。如：

```
> time_table:='time_table';
                        time_table := time_table
> time_table:= amortization(75000,(i,interest)
  ->1000+interest,.095/12):
> op(time_table[1][1..9]);
[0,0,0,-75000,75000],
  [1,1593.750000,593.7500000,1000.000000,74000.00000],
  [2,1585.833333,585.8333334,999.9999996,73000.00000],
  [3,1577.916667,577.9166667,1000.000000,72000.00000],
  [4,1570.000000,570.0000000,1000.000000,71000.00000],
  [5,1562.083333,562.0833334,999.9999996,70000.00000],
  [6,1554.166667,554.1666667,1000.000000,69000.00000],
  [7,1546.250000,546.2500000,1000.000000,68000.00000],
  [8,1538.333333,538.3333334,999.9999996,67000.00000]
```

```
> nops(time_table[1]);
                              76
> op(time_table[1][76]);
              75,1007.916667,7.916666667,1000.000000,0.
> op(time_table[2]);
                           2256250000,-5
```

我们可以看出，按照这种还款方式，只需要 75 个月就可以完全付清，而且总支付利息也减少至￥22562.5，计算起来相对比较简单，但每次还款金额相对较多。还有一种更复杂的付款方式如下例所示：

【例 6-13】 一次性借款￥75000，年利率 9.5%，按季度付款￥1000，且每年增加付款金额￥200，显示此种付款方式的时间表。

对付款额做简单公式化处理后，可以获得如下结果：

```
> time_table:='time_table';
                        time_table := time_table
> compute_payment:=(i,interest) ->
  1000+200*trunc((i-1)/4);

  compute_payment := (i, interest) → 1000 + 200 trunc(1/4 i - 1/4)
> time_table:=amortization(75000,
  compute_payment, effectiverate(0.095,4)/4):
> op(time_table[1][1..3]);
[0,0,0,-75000,75000],
   [1,1000,1845.717731,-845.717731,75845.71773],
   [2,1000,1866.530481,-866.530481,76712.24821]
> nops(time_table[1]);
                              75
> op(time_table[1][1..3]);
        74,1825.212937,43.83884984,1781.374087,0
> op(time_table[2]);
                        1258252129,-4
```

定义 compute_payment 函数时，使用了 trunc 函数，它舍去浮点数的小数部分，留下整数部分，这样可以保证 i 每增长 4 次后偿还金额才增加￥200。由于是按季度付款，所以这

里使用了 effectiverate 函数来计算相对于此种年利率，每季度的实际利率，并将计算结果代入 amortization 函数。因此最后得到的结果是需要 74 个季度才会完全付清。请读者不要将此结果同例 6-12 的结果相混淆。

如果只希望计算前几期的付款金额，可以加入第四个参数，如只显示上例中前 7 个季度的付款计划：

```
> amortization(75000, compute_payment,
   effectiverate(0.095,4)/4,7);
[[0,0,0,-75000,75000],
  [1,1000,1845.717731,-845.717731,75845.71773],
  [2,1000,1866.530481,-866.530481,76712.24821],
  [3,1000,1887.855423,-887.855423,77600.10363],
  [4,1000,1909.705163,-909.705163,78509.80879],
  [5,1200,1932.092615,-732.092615,79241.90140],
  [6,1200,1950.109100,-750.109100,79992.01050],
  [7,1200,1968.568962,-768.568962,80760.57946]],13360.57947
```

最后一个数字对应着支付了 7 个季度后总共支付的利息金额。

在本节的最后，我们将 amortization 函数的原始定义中的关键部分显示给读者，分析此段程序，会对读者继续下一章“Maple 6 程序设计”有不小的帮助：

```
if 3 < nargs then nperiods := Nperiods else nperiods := ∞ end if;
i := 0;
interest_t :=0;
balance := Amount;
tb[0] := [0,0,0,-Amount, Amount];
while 0 < balance do
    i := i+1;
    interest := balance*Rate;
    payment := Payments(i, interest);
    principal := payment - interest;
    balance := balance - principal;
    if balance < 0 then
        payment := payment + balance;
        principal := principal + balance;
        balance := 0
    end if;
    interest_t := interest_t + interest;
```

```
    tb[i] := [i,payment,interest,principal,balance];
    if nperiod ≤ i then break end if
  end do;
  [seq(tb[j],j=0 .. i)], interest_t
```

第 7 章　Maple 6 程序设计

作为 Maple 用户，我们可以在 Maple 交互界面下输入命令，调用 Maple 各种函数来解决各种问题。但有时我们希望解决一系列同一类型的问题，或者希望利用 Maple 编出我们需要的解决特定问题的函数和过程。幸运的是，Maple 提供了语法简单但却十分有用的编程语言，我们这里称它为 Maple 语言，实际上 Maple 语言就是由 Maple 各种命令以及一些简单的过程控制语句组成的。

由于用 Maple 编程时有很多 Maple 命令可供调用，它的编程功能十分强大并且简单易学。如果用户学习过面向过程的高级语言，如 C、Fortran 和 Pascal 等，就会发现，许多方面它们是十分相似的。用 Maple 语言编程可以很方便地解决许多复杂问题。

本章将介绍 Maple 编程的初步知识，重点介绍一般程序的编写和调试，Maple 高级编程用户可以参考有关这方面的著作。

7.1　简单的 Maple 6 程序设计

前面我们已经用过很多 Maple 的内部函数和程序包带的外部程序，对 Maple 函数的功能也有了一定的了解。但到目前为止，我们只知道 Maple 函数功能很强大，并且使用很方便，但一直不知道它们是怎么生成的。其实大部分函数都是用 Maple 语言写成的，先让我们来看一个关于 log 函数的源代码，这样能熟悉一下 Maple 编程的一般规则。

```
> interface( verboseproc = 2 ):
  readlib(log);
proc(x::algebraic)
local a;
option
 `Copyright(c) 1992 by the University of Waterloo.All rights reserved.`;
      if nargs ≠ 1 then ERROR (`expecting 1 argument, got `||nargs)
      elif type(procname,'indexed') then
      a := op(procname);
      if not type([a],['algebraic']) then error "invalid index"
      elif margs ≠ 1 then ERROR('expecting 1 argument, got'nargs)
      elif type(a,'complex(float)') or type(x,'complex(float)') then
```

```
            evalf(evalf(ln(x)/ln(a), Digits = +2))
        end if
    else ln(x)
    end if
end proc
```

为了得到 log 函数的原代码，我们首先调用了函数 interface，设置参数 verboseproc=2，然后用函数 readlib 就能直接查看函数，如果读者想查看别的函数的话，只需将上面的参数 log 改成想要查看的函数名即可。

虽然上面 log 函数的原代码并不是很复杂，但却包含了 Maple 编程的基本规则，下面我们从几个方面来进行介绍。

7.1.1 过程和函数

用过 Fortran 语言的人都知道，过程(procedure)和函数(function)是两个不同的概念，我们一般使用函数而不使用过程，比如在 C 语言里就取消了过程的概念，只保留了函数的概念。实际上过程和函数的功能差得不是很多，它们都可以实现对方的功能，为了跟 C 语言的习惯相同，虽然 Maple 只使用过程的概念，但有时我们也会使用函数这个概念，如果没有特别说明，我们对这两种概念将不做区分。

一个 Maple 语言的过程定义的格式如下：

```
proc (argseq)
    local lseq;
    global gseq;
    options oseq;
    description stringseq;
    statseq
end proc
```

一个过程一定是以 proc 开头，end proc（或 end）结束，这是必不可少的。过程可以带一个或几个形式参数 argseq，也可以不带；在过程的内部我们可以定义全局变量 gseq，局部变量 lseq(关于全局变量和局部变量的问题我们将在下一小节中做详细说明)，选项部分(oseq)和描述部分(stringseq)(这两个部分在后面也有详细的介绍)；当然在过程中我们也可以按照自己的要求写入语句或者调用 Maple 提供的过程。

下面我们动手自己编一个程序，如下：

```
> Area:=proc(r)
>                Pi*r^2;
>              end proc;
```

$$Area := \mathbf{proc}(r)\pi * r\hat{}2\ \mathbf{end\ proc}$$

这是一个很简单的过程，用来求给定半径的圆的面积。过程只带一个形参，并且里面既没有定义全局变量和局部变量，也没有选项部分和描述部分，只有一条计算圆的面积的式子。为了以后的方便调用，我们将这个过程赋值给了变量 Area，它就是一个过程名，就像 C 语言里的函数名一样，可以直接调用。下面我们调用一下刚刚编写的过程，比如我们计算半径为 10 的圆的面积：

```
> Area(10);
```

$$100\pi$$

从上面看到，调用的过程我们都已经很熟悉了，跟以前调用 Maple 的内部函数完全相同。当然我们也可以将这个过程返回的值赋给一个变量：

```
> s:=Area(10):
  s;
```

$$100\pi$$

多参数的过程编写跟单参数的编写是完全相同的，只是在各个形参间直接用逗号隔开，下面我们举一个三参数的例子——给定三角形的两边长度及其夹角，求其面积：

```
> TripleArea:=proc(a,b,q)
>                a*b*sin(q)/2;
>                end proc;
```

$$TripleArea := \mathbf{proc}(a,b,q)1/2 * a * b * \sin(q)\ \mathbf{end\ proc}$$

然后调用这个过程计算一个简单三角形的面积：

```
> s:=TripleArea(2,4,Pi/2):
  s;
```

$$4$$

上面两个函数都是用了最简单的过程来实现我们的目的，但其实它们都是很不完善的，它们都不能判断形参的形式，不能对过程异常给出有用的信息，而这是一个好的过程所必须具备的条件，在本章的后面我们将给出求三角形面积的改进过程。

7.1.2 全局变量和局部变量

学过 C 语言的人对全局变量和局部变量的概念想必都不陌生，在 C 语言里，在函数里面定义的变量一般叫局部变量，相对应的在函数外面的定义的变量就叫全局变量。而在 Maple 里全局变量和局部变量的区分跟 C 语言中有点出入：在 Maple 的过程里定义的也可能是全局变量，在上面讲到过程的定义的时候，我们曾经介绍过一个参数 gseq，它就是在过程中用 global 标示符声明的一个全局变量；但有一点可以肯定的是：在过程外面的肯定是全局变量，因此可以说局部变量肯定在过程里面，在过程里面的变量怎么判断其是全局变量还是局部变量呢？一般来说可以看其定义，如果是用 global 标示符标示的肯定是全局变量，而用 local 标示符标示的肯定是局部变量；假如变量没有经过标示符定义，以下两种情况 Maple 自动认为是局部变量：

（1）出现在赋值符左边的变量。

（2）在 for 循环或 seq、add、mul 命令中表示计数性质的变量。

除了这两种情况以外的其他变量，Maple 将认为是全局变量。为了对这些有一个较清晰的了解，我们来看下面这个过程：

```
> Temp:=proc()
>         global      nG
>         local       nL;
>         nG:=1;
>         nL:=2;
>         nTemp:=3;
>         for i from 1 to 10 do
>               nTemp:=0;
            od;
>         end;
Warning,`nTemp` is implicitly declared local to procedure `Temp`
Warning,`1` is implicitly declared local to procedure `Temp`

Temp := proc()
local nL, nTemp, i;
global nG;
    nG := 1; nL := 2; nTemp := 3; for i to 10 do nTemp := 0 end do
end proc
```

由于我们定义变量 i 和 nTemp，Maple 给出了警告信息，并提醒用户这两个变量将被作为局部变量，在生成过程的例子中我们能很清楚的看到这一点，Maple 将这两个变量用标

示符 local 定义了。

我们介绍了局部变量和全局变量的判别，下面来看看它们之间的区别。全局变量不属于过程，它可以存在于过程之外，而局部变量只能存在于过程之内，当过程创建时，局部变量也同时被创建，而当过程销毁时，局部变量也同时被销毁。我们来看看上面那些变量：

```
> Temp();
```

$$0$$

```
> nL;
```

$$nL$$

```
> nG;
```

$$1$$

```
> nTemp;
```

$$nTemp$$

```
> i;
```

$$i$$

变量 nG 是我们自己定义的一个全局变量，当过程 Temp 结束时，它的值仍然存在，说明变量没有销毁，而我们定义的局部变量 nL 在过程之外就不存在了，另外两个没有定义，但被 Maple 默认为局部变量的变量 nTemp 和 i 都不存在。虽然 Maple 能判断未定义的变量，但对我们来说有点模糊，为了使变量定义比较明了，建议对每一个变量都进行定义。

Maple 将不同过程的局部变量看成是单独的变量，即使变量名相同，它们之间也不会受影响，并且一个过程中的局部变量不会影响同名全局变量，在过程中使用这个变量时，过程内定义的变量优先，即这个变量的值将是局部变量的值，而与同名的全局变量无关，我们来看下面例子：

```
> Temp1:=proc()
>         local   nG;
          nG:=10;
          nG;
>       end;
```

$$Temp1 := \mathbf{proc}()\mathbf{local}\ nG; nG := \mathbf{10}; nG\ \mathbf{end\ proc}$$

```
> Temp1();
```

$$10$$

```
> nG;
```

$$1$$

我们可以看到我们在过程 Temp1 中将 nG 定义为局部变量，并在过程中赋值 10，但在过程外 nG 的值不变，还是等于 1，上面这些规则跟 C 语言都是相同的。但关于全局变量的

规则有些不同，在 Maple 中全局变量能够重复定义，后面的定义将覆盖前面的变量（其实是在赋值时覆盖，在定义时不覆盖），我们可以从下面看出：

```
> Temp2:=proc()
>         global nG,nG1;
          nG1:=nG;
          nG:=10;
>        end;
```

$$Temp2 := \mathbf{proc}()\mathbf{glocal}\ nG, nG1; nG2 := nG; nG := 10\ \mathbf{end\ proc}$$

```
> Temp2();
```

$$10$$

```
> nG1;
```

$$1$$

```
> nG;
```

$$10$$

为了查看全局变量在定义以后的数值我们另外定义了一个全局变量 nG1，从运行后的结果来看，重新定义后其值并没有改变，直到赋值后其值才改变了。

在程序设计时要注意一点：在过程中对全局变量进行赋值时要特别小心，因为别的过程也可能在使用这个全局变量，任何全局变量的改变将影响到所有使用这个变量的过程，而用户往往注意不到这一点，因此在选择变量的类型时要特别注意。

另外为了保持程序的模块化、增强程序的可读性，一般不在过程内部定义全局变量，假如过程内需要用到全局变量的值，一般都用参数来传递，这些原则跟其他高级语言的程序设计原则是一样的。

7.1.3 参数声明和标识符 args 的使用

在调用过程中我们往往要用到形参，Maple 允许不定义参数的类型而直接使用，但有时为了限制参数的类型，希望在输入的参数与我们预先定义的参数不匹配时，系统能够给出出错的信息，我们就应该进行参数类型的声明，参数声明的语法如下：

parameter::type;

parameter 是参数名，type 为参数的类型。先来看下面的这个过程：

```
> CircleArea:=proc(r::positive)
>                 Pi*r^2;
>                 end;
```

$$CircleArea := \mathbf{proc}(r :: positive)\pi * r\hat{}2\ \mathbf{end\ proc}$$

这个过程我们在前面已经出现过，是一个求圆面积的过程，但在前面定义时我们并没有设定参数的类型，即使将半径写成一个负数或者是一个字符，过程也会执行，这显然不是我们期望的，现在我们将半径的参数声明为一个正数后，当我们输入一个负数的半径时，系统将给出错误信息：

```
> CircleArea(-10);
Error, CircleArea expects its 1st argument, r, to be of type
positive, but received -10
```

只有我们输入满足要求的参数时，过程才会给出准确的计算结果：

```
> CircleArea(10);
                                100π
```

从上面可以看出，Maple 参数声明的过程是很简单的。Maple 参数的类型也是多种多样的，读者可以利用命令?type 来查看帮助。

跟 C 语言一样，参数传递对于用户来说是很方便的，在 C 语言中传递参数都是通过定义函数的形参来确定的，这就需要知道形参的具体个数和类型，但有时我们在定义的过程中并不知道参数的个数，或者参数的个数是可变的，碰到这种情况在 C 语言中一般都是很让人头疼的，但在 Maple 中事情就变得很简单，Maple 提供了一个全局的标识符 args 来传递参数，我们可以在定义过程中不定义形参，而直接使用 args，因为在调用过程时 Maple 会自动将参数传给 args，来看一个例子：

```
> ssum:=proc()
>       local i,total;
        total:=0;
>       for i from 1 to nargs do
>               total:=total+args[i];
>       end do;
>       total;
>       end;
SSum := proc()
local i, total;
    total := 0; for i to nargs do total := total + args[i] end do; total
end proc
```

这个过程没有任何形参的定义，除了用到标识符 args 以外，还用到一个参数 nargs，它是系统用来存储参数个数的一个变量，当我们调用过程时，系统自动将参数个数赋给它，这一点给我们编程带来了很大的方便，下面我们分别代入 1、2、3 个参数来看看结果：

```
> SSum(12);
                                  12
> SSum(10,11);
                                  21
> SSum(10,11,12);
                                  33
```

这个过程输入的参数除了可以是具体的数外，还可以是序列，如下：

```
> seq1:=seq(n,n=1..100);
seq1 := 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
   23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43
   44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,21,62,63,64,
   65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,
   86,87,89,90,91,92,93,94,95,96,97,98,99,100
```

利用标识符 args，再结合参数 nargs，我们就能很方便地处理形参个数不定的情形了。

7.1.4 注释和描述(description)

为了增强程序的可读性，我们经常要在程序的开头或各条语句的后面加上注释语句，注释语句对程序的运行没有影响，系统在编译时忽略注释语句。Maple 的注释语句是用“#”来实现的。我们来看一个例子：

```
> T:=proc()
>     local x;                    #x is a local viarable
>     x:=10;
>     end ;
                T := proc()local x; x := 10 end proc
> T()
                                  10
```

上面的过程中定义变量语句的后面加了注释语句来说明变量的属性，这是一种最常见的用法，能够提醒我们语句的用途，注释语句还有一种比较常用的用法就是在过程的开头或前面注释过程的用途：

```
> #************************************#
  #          proc       CircleArea      #
  #      Compute the Area of a circle    #
  #************************************#
  CircleArea:=proc(r::positive)
              pi*r^2
              end;
```

TCircleArea := **proc**(*r* :: *positive*)π * *r*;^2 **end proc**

我们用注释语句很明了地说明了过程的用途，大大增加了程序的可读性。

描述语句类似于注释语句，它对程序的功能也没影响，但系统编译的时候并不能忽略它，并且在程序的编译后能够给出一条注释语句，其具体形式如下：

description stringseq;

参数 stringseq 就是我们希望过程编译后提供注释的内容，描述语句在编译后的过程中存在，但在过程运行时并不显示：

```
> CircleArea:=proc(r::positive)
      description "Compute the Area of a circle";
                  Pi*r^2;
                  end;
```

CircleArea :=

proc(*r* :: *positive*)**description** "Compute tne Area of a circle"; π * *r*;^2 **end proc**

```
> CircleArea(10);
```

100 π

要查看一个过程可以用 print 函数，比如：

```
> print(CircleArea);
```

proc(*r* :: *positive*)**description** "Compute tne Area of a circle"; π * *r*;^2 **end proc**

7.1.5　过程返回值设置

一般我们编写一个过程都希望能得到一个结果，并且希望能够返回被我们利用，这就用到 Maple 的过程返回值的设置，在默认情况下，过程返回值是过程语句中最后一条语句的值，这种返回方式我们在上面已经见过，比如：

```
> CircleArea:=proc(r::positive)
                    Pi*r^2;
                    end;
```

$$CircleArea := \mathbf{proc}(r :: positive)\pi * r;\hat{}2\ \mathbf{end\ proc}$$

```
> f:=CircleArea(1);
```

$$f := \pi$$

求圆面积的过程的返回值就是一个求面积的语句计算得到的结果，这条语句就是最后一条语句，假如在求面积的语句后面再加上一条语句（显示半径的语句），返回的将是新加上去的语句的值，如下：

```
> CircleArea:=proc(r::positive)
                    Pi*r^2;
                    r;
                    end;
```

$$CircleArea := \mathbf{proc}(r :: positive)\pi * r;\hat{}2\ \mathbf{end\ proc}$$

```
> CircleArea(2);
```

$$2$$

上面我们都是返回一个参数，其实默认的返回方式也可以实现多参数的返回，只需将要返回的语句写在同一条语句里即可，看下面的例子：

```
> CircleArea:=proc(r::positive)
                    r,Pi*r^2;
                    end;
```

$$CircleArea := \mathbf{proc}(r :: positive)r, \pi * r\hat{}2\ \mathbf{end\ proc}$$

```
> CircleArea(10);
```

$$10, 100\pi$$

要将多个返回值赋给外面的全局变量，我们只需要像以前那样赋值即可：

```
> r:=%[1];
```

$$r := 10$$

```
> area:=%%[2];
```

$$area := 100\,\pi$$

参数的返回除了用系统默认的返回方式外，还可以用给形式参数赋值的方法来返回所需要的结果，这种方法跟 C 语言的参数赋值返回是一样的：

```
> CircleArea:=proc(r::positive,q::evaln,area::evaln)
                    q:="r";
                    area:=Pi*r^2;
                  end;
```

CircleArea :=

proc(*r* :: *positive*, *q* :: *evaln*, *area* :: *evaln*)*q* :="r"; *area* := π * *r*^2 **end proc**

```
> CircleArea(10m,q,area);
```

$$100\pi$$

```
> q;
```

"r"

```
> area;
```

$$100\pi$$

为了将第二、三个参数作为被赋值的参数，我们将它们声明为 evaln 类型，这个类型能将一个变量名传入。

除了以上两种方式能够实现过程值的返回以外，还有一种很常见的返回方式：利用 return（或 RETURN）来实现强制返回，函数的具体表达式如下：

return expr1, expr2, ...;
return (expr1,expr2,...);
RETURN (expr1, expr2, ...)

参数 expr1,expr2....是待返回的值（也可以是表达式或字符串），返回的值可以为空。使用时要注意没有 RETURN expr1,expr2.....这种格式：

```
> CircleArea:=proc(r::positive)
                  return r,Pi*r^2
                  end;
```

CircleArea := **proc**(*r* :: *positive*)**return** *r*, π * *r*^2 **end proc**

```
> CircleArea(10);
```

$$10, 100\pi$$

return(RETURN)返回语句将结束过程返回，其后面的语句将不再被执行，所以这条语句也可以作为中断过程进行强制返回的语句：

```
> CircleArea:=proc(r::positive,area::evaln)
  return r, Pi*r^2;
  area:=Pi*r^2;
  end;
```

CircleArea :=

proc(*r* :: *positive*, *area* :: *evaln*)**return** *r*, π * *r*^2; *area* := π * *r*^2**end proc**

```
> CircleArea(20,area);
```

$$20,400\,\pi$$

```
> area;
```

$$100\,\pi$$

上面这个过程中，我们在 return 语句后面的赋值语句并没有被执行，全局变量的值仍然是 100Pi 而不是我们希望的 400Pi。因此在设计程序时，假如要返回多个参数，都应该用一条 return 语句带多个返回值，避免用多条 return 语句来返回。

return 语句很简单，功能也很单一，但在结构化程序设计中，这条语句非常有用，我们在后面还将经常用到。

7.1.6 格式输出函数 printf

上一小节我们利用返回语句返回我们需要的值，用这种方法只能得到最后的结果，并且此时程序就被中断，假如我们需要在程序的运行过程中向屏幕中输出一些字符或数字而又不影响程序的运行，那就可以用格式输出函数 printf。

函数 printf 能输出若干个任意类型的数据，其一般的格式为：

printf(fmt, x_1, ..., x_n);

参数 fmt 表示“格式控制”，是用双引号括起来的字符串，也称“转换控制字符串”，它包括两种信息：

（1）格式说明，由“%”和格式字符组成，如%d、%f 等。它的作用是将输出的数据转换为指定的格式输出。格式说明总是由“%”字符开始的。

（2）普通字符，即需要原样输出的字符。参数 x_1,...,x_n 是函数将要输出的一些数据，也可以是表达式。例如：

```
printf(" %d %d",a,b);
printf("a+b= %d,a-b=%d",a+b,a-b);
```

在上面双引号中的字符除了“%d”以外，还有非格式说明的普通字符，它们按原样输出，我们将 a 和 b 分别设为 10 和 6，看看是什么结果。

```
> a:=10:b:=6:
  printf("%d,%d",a,b);
 10,6
```

```
> printf("a+b=%d,a-b=%d",a+b,a-b);
 a+b=16,a-b=4
```

printf 函数的格式字符有很多种，下面我们来介绍一下几种最常用的格式字符：

（1）d 格式符。用来输出十进制整数。有以下几种用法：

1）%d，按整型数据的实际长度输出（见上面）。

2）%md，m 为指定的输出值段的宽度。如果数据的位数小于 m，则左端补以空格，若大于 m，则按实际位数输出。如：

```
> a:=12345:b:=123:
  printf("%4d,%4d",a,b);
12345, 123
```

我们看到后一个数与前一个数相隔了 1 个字符。

（2）o 格式符。以八进制数形式输出整数。由于是将内存单元中的各位的值（0 或 1）按八进制形式输出，因此输出的数值不带符号，即将符号位也一起作为八进制的一部分输出，例如：

```
> a:=-1
  printf("%d,%O",a,a);
-1,37777777777
```

（3）x 格式符。以十六进制数形式输出整数，同样不会出现负的十六进制数。

（4）f 格式符。用来输出实数（包括单、双精度），以小数形式输出。有以下几种用法：

1）%f，不指定字段宽度，由系统自动指定，使整数部分全部如数输出，并输出 6 位小数。例如：

```
> f:=12.111
  printf("%f",f);
```

2）%m.nf，指定输出的数据共占 m 列，其中有 n 位小数，注意小数点也占一列。如果数值长度小于 m，则左端补空格，如：

```
> printf("%5.2f,%10.3f",f,f);
12.11,     12.111
```

3）%-m.nf，与%m.nf 基本相同，只是输出的数值向左端靠，右端补空格。

（5）e 和 E 格式符。以指数形式输出实数。可用以下形式：

1）%e，不指定输出数据所占的宽度和数值部分小数位数，由系统自动给出 6 位小数，指数部分占 4 位（如 e+02），其中 e 占 1 位，指数符号占 1 位，指数占 2 位，数值按标准化指数形式输出（即小数点前必须有而且只有一个非零数字），例如：

```
> e:=12.111:
  printf("%e",e);
1.211100e+01
```

2）%m.nf，m 指输出的数据共占 m 列（包括后面的指数部分），n 指数据的数字部分的小数部分，如：

```
> printf("%10.2e,%20.5e",e,e);
1.21e+01,      1.21110e+01
```

3）%-m.ne，与%m.ne 基本相同，只是输出的数值向左端靠，右端补空格。

（6）g 格式符。用来输出实数，它根据数值的大小，自动选 f 格式或 e 格式（选择输出时占宽度较小的一种），且不输出无意义的零，例如：

```
> printf("%g %g %g \n,123,123/456,123456789);
123          .269737  1.234568e+08
```

（7）c 格式符。用来输出一个字符，例如：

```
> c:='c':
  printf("%c%, c);
  c
```

（8）s 格式符。用来输出一个字符串，有几种用法：

1）%s，按实际长度输出字符串：

```
> printf("%s",'abcdefg');
abcdefg
```

2）%ms，输出字符串占 m 列，如字符串本身大于 m 列，则突破 m 的限制，将字符全部输出。若串长小于 m，则左补空格：

```
> s:="It 's me! ":
  printf("%4s,%12s",s,s);
It's me!,      It's me!
```

3）%-ms，跟%ms 相同，只是空格补在右边。

4）%m.ns，输出占 m 列，但只取字符串中左端 n 个字符。这 n 个字符输出在 m 列的右侧，左补空格：

```
> printf("%3.6s,%10.4s",s,s);
It's me,      It's
```

5）%-m.ns，其中 m、n 的含义同上，n 个字符输出在 m 列范围的左侧，右补空格，如果 n>m，则 m 自动取 n 值，即保证 n 个字符正常输出。

（9）a、A 格式符和 q、Q 格式符。a 和 A 用来准确地输出 Maple 中的表达式。而 q 和 Q 跟 a 的功能相似，只是这个格式符将后面所有的表达式序列都用 a 格式转换后输出：

```
> printf("head=%a tail=%q\n",a,b,c,d,e,f);
head=-1 tail=123, c, d, 12.111, 12.111
```

输出中既有数字又有字母，这是因为有些字母在上面我们已经赋值的缘故。

最后我们再来介绍一下一个很常用的控制字符“\n”的作用，这个控制字符是处理换行的，前面我们已经用过，只是我们没有看到它的作用，下面我们举个例子：

```
> printf("head=%a\ntail=%q\n",a,b,c,d,e,f);
head=-1
tail=123, c, d, 12.111, 12.111
```

7.2 选择结构

判断给定的条件是否满足，并根据判定的结果（真或假）决定执行给出的两种操作之一，这种程序结构就是选择结构。Maple 里的选择结构跟其他高级语言一样，也是用 if 语句来实现的，下面我们分三种形式来介绍 if 语句，并简略介绍一下 if 语句嵌套结构的使用。

普通的 if 语句可以分成三种形式：

1. if（表达式）then 语句 1 fi

这是最简单的一种选择语句，当表达式的值为真时，过程转入语句 1 执行，否则跳出选择语句，从下面的例子就很容易明白这一点：

```
> Max:=proc(x::numeric,y::numeric)
        if(x>=y) then return x;
        fi;
        end;
>
```

$$Max := \mathbf{proc}(x :: numeric, y :: numeric)\mathbf{if}\, y \le x\, \mathbf{then\ return}\, x\, \mathbf{end\ if\ end\ proc}$$

```
> Max(1,2);
> Max(3,2);
                                3
```

上面过程是比较 x 和 y 的大小，当 x 不小于 y 时，返回 x 的值，否则退出选择。运行过程我们看到当 x<y 时，函数确实没有进入选择。结束选择语句的表达式 fi（是 if 的反写）也可以用 end if 来代替，下面我们碰到的语句中这两者也是可以互换的，对结果没有任何影响，读者如果有兴趣可以试一下。为了跟高级语言的编程习惯相同，我们在后面的程序中都将用 end if 来代替 fi。

2. if（表达式）then 语句 1 else 语句 2 end if

当表达式的值为真时，过程执行语句 1，否则执行语句 2，例如：

```
> Max:=proc(x::numeric,y::numeric)
        if(x>=y) then return x;
        else return y;
        end if;
        end;
```

$Max := \mathbf{proc}(x :: numeric, y :: numeric)\mathbf{if}\ y \le x\ \mathbf{then\ return}\ x\ \mathbf{else\ return}\ y\ \mathbf{end\ if\ end\ proc}$

```
> Max(1,2);
                                2
> Max(3,2);
                                3
```

3. if（表达式 1）then 语句 1

elif（表达式 2）then 语句 2
elif（表达式 3）then 语句 3
⋮ ⋮
elif （表达式 n）then 语句 n
else 语句 n+1
end if

这种选择语句都是从头开始判断，首先判断表达式 1，当表达式 1 为真时，执行语句 1，然后退出选择；否则判断表达式 2，若表达式 2 为真，则执行语句 2；否则判断表达式 3……依次类推，直到有一个表达式为真，如果表达式 1 到表达式 n 都为假，则执行语句 n+1。我们从下面这个例子很快就能明白这种选择语句的使用：

```
> Test:=proc(x::numeric)
   if(x>100) then return("x is more than 100");
   elif(x>=10) then return ("x is between 10 and
   100");
   elif(x>0) then return("x is beteewn 0 and 10");
   else return ("x is less than 0");
   end if;
   end:
> Test(101);
                          "x is more than 100"
> Test(50);
                        "x is between 10 and 100"
> Test(5);
                          "x is between 0 and 10"
> Test(-10);
                            "x is less than 0"
```

if 选择语句除了可以像上面那三种形式一样使用外，还可以嵌套组合使用，就是一个语句中包含多个 if 语句，它们的一般形式如下：

```
if（表达式 1）then
if（表达式 2）then 语句 1
    else 语句 2
end if
else
if（表达式 3）then 语句 3
    else 语句 3
end if
end if
```

当然各个语句中也可以没有 else 语句部分，且嵌套的层数可以不止一层，子 if 语句下面还可以嵌套 if 语句，我们看一下下面的程序就能明白嵌套的作用：

```
> Test:=proc(x::numeric)
        local y;
        if(x>100) then
                if (x>200) then
  y:=2;
                  return(y);
                else
                  y:=1;
                  return(y);
                end if;
          else
                y:=0;
                return(y);
          end if
          end:
> Test(400);
                                    2
> Test(200);
                                    1
> Test(100);
                                    0
```

7.3 循环结构

在许多问题中需要用到循环控制。例如输入全校学生成绩，求若干个数之和，迭代求根等。几乎所有使用的程序都包含循环。循环结构是 Maple 语言中三种最基本的结构之一，它和顺序结构、选择结构共同作为各种复杂程序的基本构造单元。因此熟练掌握选择结构和循环结构的概念及使用是 Maple 编程的最基本的要求。

在 Maple 中可以用以下语句实现循环：

（1）for...do 语句。

（2）while...do 语句。

（3）for...while...do 语句，这是上面两种语句的组合。

在下面各下节中我们分别介绍它们的用法。

7.3.1　for...do 循环

for...do 语句循环的一般形式如下：

for 变量名 i　[from 表达式 1] [by 表达式 2] to 表达式 3 do

　　语句

end do;

变量名 i 是用来控制循环的一个变量，既可以是全局变量也可以是局部变量；from 语句后放一个能计算出数值的代数表达式，表示变量 i 开始的值，这个语句放在方括号中表示这部分可以省略，当语句省略时，系统取默认值 1；by 语句用来设置变量变化的增量（步长），可以是正值也可以是负值，此语句也可以省略，省略时步长为 1；表达式 3 表示的数值是变量 i 的最后界限，当步长为正，变量 i 的值大于表达式 3 的值时，循环结束，而当步长为负，变量 i 小于表达式 3 的值时，循环也结束。

使用这种循环的时候要注意，这种循环语句每执行一次循环，变量 i 自动加上一个步长，不需要额外操作。我们还是来看一个例子，这个例子用来求 1 到给定一个整数 n 之间所有整数的和：

```
> SSum:=proc(n::positive)
        local i,s;
        s:=0;
        for i to n do
            s:=s+i;
            end do;
        return(x);
        end:
> SSum(100);
                                5050
```

假如我们想从 10 开始求和，并且步长不是为 1，而是为 2 的话，我们只需要加上 from 和 by 语句部分即可：

```
> SSum:=proc(n::positive)
        local i,s;
        s:=0;
        for i from 10 by 2 to n do
            s:=s+i;
            end do;
        return(s);
        end:
```

```
> SSum(100);
```

$$2530$$

我们也可以将步长设为负的，但要注意从起始值加上有限次的步长能够到达最后给定值，否则将不会进入循环：

```
> SSum:=proc(n::humeric)
        local i,s;
        s:=0;
        for i from 10 by -2 to n do
            s:=s+i;
            end do;
        return(s);
        end:
> SSum(-100);
```

$$-2520$$

```
> SSum(100);
```

$$0$$

7.3.2 while...do 循环

除了可以用 for...do 语句进行循环控制外，在 Maple 中还可以用 while...do 语句来进行控制，这一点与 C 语言中的情况非常相像，这种循环结构的形式如下：

while（表达式）do

　　语句

end do;

while 语句控制循环时首先判断“表达式”是否为真，如果是则进入循环执行“语句”，然后再判断“表达式”的结果是否为真，如果是就再进入循环语句，然后再判断，一直到表达式的结果为假，终止循环，比如：

```
> SSum:=proc(n::positive)
        local sum,i;
        sum:=0;i:=1;
        while(i<=n) do
            sum:=sum+i;
            i:=i+1;
         end do;
         return (sum);
         end:
> SSum(10);
```

55

while...do 语句与 for...do 语句比较，最大的不同点就是在判断是否进入循环的表达式的值的变化上，在 for...do 语句中，表达式的值是随着变量 i 而变化的，而 i 的值是每次循环后语句自动增减的，也就是说表达式的值可以是由语句自动变化引起的；而 while...do 语句则不同，语句每次进入循环，表达式的值不会发生变化，除非我们在语句中加入了改变表达式中的每个变量的值，所以在我们使用这种循环语句的时候，一定要在循环语句中设置改变表达式的值的语句，否则这个循环将是一个死循环，这将造成程序不能结束。

不过死循环也并不是绝对要避免的，有时为了编程的需要我们可能故意将 while...do 语句写成死循环的形式，而在循环内部来判断是否结束循环，比如上面程序我们就可以用这种形式来实现：

```
> SSum:=proc(n::positive)
        local sum,i;
        sum:=0;i:=1;
        while(2>1) do
            if(i>n) then
                return (sum);
            end if;
            sum:=sum+i;
            i:=i+1;
         end do;
         end:
> SSum(10);
```

55

我们在程序中用表达式（2>1）是否为真来判断是否进入 while...do 循环，显然这个表

达式的值不会改变，因此程序总会进入循环，是一个死循环的结构，但我们在循环内部设置了一条判断语句和强制返回语句（return 语句），可以在一定的条件下（i>n）跳出整个循环（在这里是跳出整个过程）。

提到强制返回语句我们在这儿顺便提一下另外两个跳出循环的语句：next 语句和 break 语句。这两条语句的功能跟 return 语句是不同的：next 语句是跳出本次循环，而 break 是结束这个循环，我们还是利用上面那个求和例子分别检验一下它们的作用：

我们首先来看一下 break 语句，为了与 return 语句的返回值有点区别，我们在求和程序循环结束后加上一条语句，给和 sum 加上 100。

```
> SSum:=proc(n::positive)
        local sum,i;
        sum:=0;i:=1;
        while(2>1) do
            if(i>n) then
                break;
            end if;
            sum:=sum+i;
            i:=i+1;
         end do;
         return(sum+100);
         end:
> SSum(10);
```

155

从运行的结果可以看出，break 语句在结束循环后并不立即结束整个过程，而是继续执行循环后面的语句，返回的结果就是 return 语句执行的结果。而 return 语句则不然，它是强制结束整个过程，并返回一个值，如下：

```
> SSum:=proc(n::positive)
        local sum,i;
        sum:=0;i:=1;
        while(2>1) do
            if(i>n) then
                return(sum);
            end if;
            sum:=sum+i;
            i:=i+1;
         end do;
         return(sum+100);
         end:
```

```
> SSum(10);
                                55
```

很明显上面这个程序并没有执行语句 return(sum+100)，而是在中间被强制中断。next 语句的作用我们先从下面这个程序入手来加以了解：

```
> SSum:=proc(n::positive)
        local sum,i;
        sum:=0;i:=1;
        while(2>1) do
            if(i>n) then
                next;
            end if;
            sum:=sum+i;
            i:=i+1;
         end do;
         return(sum+100);
         end:
```

这个程序跟上面两个程序相比惟一的不同点，就是上面的 return 语句或 break 语句被替换成了 next 语句，由于 next 语句只是结束一次循环，读者仅凭猜想就能知道这是一个死循环，下面我们来看看执行的结果：

```
> SSum(10);
Warning, computation interrupted
```

我们输入“SSum(10);”并按回车以后可以看到鼠标始终处于等待状态，整个过程不能自动结束，这是死循环的特征，利用 Maple 软件上的“stop”按钮，我们可以强制终止这个过程，这时系统返回一个计算被终止的警告信息。next 语句虽然不能终止死循环，但它也有特殊的功能，比如我们想求 1.. n 之间不能被 3 整除的所有整数的和，这时用 next 语句就很方便：

```
> SSum3:=proc(n::positive)
        local sum,i;
        sum:=0;i:=1;
        while(i>n) do
           i:=i+1;
           if((modp(i,3)=0)) then
                  next;
           end if;
           sum:=sum+i;
         end do;
         return(sum);
         end:
> SSum3(10);
```

37

7.3.3 for...while...do 循环

for...do 语句和 while...do 语句除了可以单独构成循环结构外，还可以组合起来使用，它们的组合形式如下：

for 变量名 i [from 表达式 1] [by 表达式 2] to [表达式 3] while(表达式 4) do
    语句
end do;

表达式 1 到表达式 3 以及变量 i 的含义跟 for...do 循环中的含义相同，这个循环中只是又多了一个判断是否进入循环语句的判断语句 while(表达式 4)，这种循环结构只有在变量小于等于（步长为正）或大于等于（步长为负）表达式 3 的值并且表达式 4 为真时才进入循环，否则结束循环结构，我们通过下面这个程序来理解这种结构：

```
> SSum:=proc(n::positive)
        local i,j,s;
        s:=0; j:=10;
        for i from 0 by 3 to n while(j>i) do
            s:=s+i*j;
            j:=j+1;
            end do;
         return(s);
         end:
```

这个函数执行循环的条件是变量 i 不大于外部给定的参数 n（因为本循环中步长为 3 是正的），并且变量 j 大于变量 i。我们输入不同的参数来检验这个程序，可以比较清晰地看出循环的结构：

```
> SSum(10);
                                268
> SSum(11);
                                268
> SSum(12);
                                268
> SSum(13);
                                450
> SSum(14);
                                450
> SSum(100);
                                450
```

前面三个数相同是因为变量 i 的步长是 3，这点我们能够比较容易想到，而当代入参数不小于 13 时是由于此后的循环都是 while 判断条件给中断的。其实上面程序也可以用一个 while...do 语句来完成：

```
> SSum:=proc(n::positive)
        local i,j,s;
        s:=0;
        i:=1; j:=10;
        while(j>i and i<=n) do
            s:=s+i*j;
            i:=i+3;
            j:=j+1;
            end do;
         return(s);
         end:
> SSum(10);
                                268
> SSum(13);
                                450
```

有兴趣的话，读者可以代入其他的参数进行进一步的验证。从以上的例子可以看出Maple的循环结构是比较灵活的，用什么样的循环结构，可以根据具体的情况或个人的习惯来确定，一般没有强制的规定，也很少有什么好坏之分。

与选择结构一样，循环结构也可以嵌套，它们的用法很简单，读者可以自己练习一下，这里就不再一一举例，最后我们来看一个综合的例子。

【例 7-1】求 100~200 间的全部素数。

```
> primenumber:=proc()
          local m,k,i;
          for m from 101 by 2 to 200 do
             k:=trunc(sqrt(m));
             i:=1;
             while(i<=k) do
             i:=1;
             while(i<=k) do
                 i:=i+1;
                 if(modp(m,i)=0) then
                       break;
                 end if;
                 if(i>=k+1) then
                        printf("%d,",m);
                 end if;
                 end do;
              end do;
            end:
> primenumber();
101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,
179,181,191,193,197,199,
```

7.3.4 递归调用

说到循环结构，我们在这儿顺便提到递归结构，严格意义上来说，递归调用跟我们上面讲的循环结构还是有很大的区别，但我们认为在内在的机制上它们还是与循环比较相似，因此在本节中介绍递归调用。

相对于普通的循环结构来说，递归调用要简单得多，递归只不过是一个调用自身的过程，我们来看下面这个例子就能明白如何使用递归调用：

```
> Fibonacci:=proc(n::nonnegint)
      if n<2 then
          n;
        else
          Fibonacci(n-1)+Fibonacci(n-2);
        end if;
    end:
> Fibonacci(10);
                              55
```

上面这个例子是一个斐波那契函数，它的第 n（n>1）项的值等于第 n-1 项和第 n-2 项的和，用递归的方法我们很容易得到任意项的值。假如我们想用循环来得到这个函数任意项的值可以用下面的方法来实现，但明显要比递归调用复杂：

```
> Fibonacci:=proc(n::nonnegint)
      local F,i;
      F[0]:=0; F[1]:=1;
      if (n<=1) then
            return (F[n]);
      end if;
      for i from 2 to n do
          F[i]:=F[i-1]+F[i-2];
       end do;
       return (F[n]);
       end:
> Fibonacci(10);
                              55
```

下面我们用一个例子来结束递归调用的介绍。

【例 7-2】求第 n 项车比雪夫多项式（Chebyshev Polynomials）的形式。车比雪夫多项式是一种比较典型的多项式。其定义为：

$$T_0(x)=1, \qquad T_1(x)=x$$
$$T_n(x)=2xT_{n-1}(x)+T_{n-2}(x) \qquad (n>1)$$

如果用公式推导来求这个多项式的话显然非常麻烦，用递归算法实现起来却相当轻松：

```
> T:=proc(n::nonnegint,x::name)
    option remember;
    if n=0 then
      return(1);
      end if;
      if n=1 then
        return(x);
      end if;
      return(2*x*T(n-1,x)+T(n-2,x));
      end:
> T(4,x);
```

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

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

$$8x^4+8x^2+1$$

7.4 程序调试器 debugger

我们编写程序的时候不可能一次成功，很多时候系统编译成功，但执行时却不是出现我们期望的结果。这时我们就需要对程序重新审视，第一步当然是进行人工源代码的检查，检查是否有明显的源代码错误，如果有错并对其进行修改后，程序执行正常，那么可以说人工调试结束，不需要用到调试器（debugger）了，但是现实情况往往是用人工方法找不出或者找出错误但进行修改后还是有错，在这种情况下再利用人工查错显然有点勉为其难了，在这个时候我们就应该用调试器。

进行过高级语言编程的读者对程序调试可能都比较熟悉，对本节的内容浏览一下就可以了，如果没有用过调试器的读者，希望通过学习本节对程序调试有一个基本的了解，并且能够自己进行简单的程序调试，对于比较复杂的程序的调试需要一定的经验，读者可以通过慢慢练习来加以掌握。

下面我们通过调试一个具体的 Maple 程序来介绍几个用于 Maple 程序调试的命令，并介绍程序调试的一般方法，最后我们再介绍几个也可用来程序调试的命令和方法。

首先我们来看看我们要调试的程序，这是一个求完数的程序，其题目如下：

【例 7-3】一个数如果恰好等于它的因子之和，这个数就称为“完数“。例如：6 的因子是 1、2、3，而 6=1+2+3，因此 6 是“完数”，编程找出 1000 之内的所有完数，并按下面格式输出其因子：

6 its factors are 1,2,3.

下面我们来看看我们编制的源程序，为了学习调试的方法，我们故意在源程序中加入了两个错误，其中一个是语法错误，另一个是设计错误，先来看源程序：

```
> restart;
  compute:=proc()
     local i,j,k,m,a,sum;
     for m from 1 to 1000 do
        k=trunc(sqrt(m));
        a[1]:=1;j:=1;sum:=1;
        for i from 2 to k do
            if(modp(m,i)=0) then
               j:=j+1;
               a[j]:=i;
               sum:=sum+i;
             end if;
         end do;
         if(sum=m) then
            printf("\n%d  its factors are ",m);
            for i from 1 to j do
                printf("%d,",a[i]);
            end do;
          end if;
      end do;
      end:
```

这个程序的思想为：利用判断一个正数是否为素数的方法，找出所有的因子，然后判断这些因子的和是否等于正数本身，由于 1 肯定是正数的因子，所以我们求因子都是从 2 开始判断。

有了源程序我们先来运行一下，看看结果是否正确：

```
> compute();
Error, (in compute) final value in for loop must be numeric or
character
```

我们没有得到想要的结果，说明程序中包含有错误，但直接看源程序又很难找出错误，没办法，只好进入程序调试，下面我们边介绍调试的方法边详细介绍各个调试函数的功能和具体用法。

7.4.1 显示程序(showstat)

我们调试程序时最常用的是在程序中设置断点，Maple 提供了一个利用语句号和逻辑表达式来设置断点的方法，但很不幸的是，Maple 中的语句号跟我们源程序中的行号并不是一一对应，为了得到语句号我们得首先用 Maple 提供的显示语句号的函数来显示整个源程序，这个函数就是 showstat，它的函数表达式如下：

showstat(procName);
showstat(procName, statRange);

第一个参数 procName 是我们待显示语句号的过程名，第二个参数用来设定需要显示语句号的范围，第二个参数是可选的，当第二个参数缺省时，Maple 显示出整个过程的语句号，下面我们来看看 compute 过程的语句号：

```
> showstat(compute);

compute := proc()
local i, j, k, m, a, sum;
    1   for m to 1000 do
    2     k = trunc(sqrt(m));
    3     a[1] := 1;
    4     j := 1;
    5     sum := 1;
    6     for i from 2 to k do
    7       if modp(m,i) = 0 then
    8         j := j+1;
    9         a[j] := i;
   10         sum := sum+i;
            end if
            end do;
   11       if sum = m then
   12         printf("\n%d its factors are ",m);
   13         for i to j do
   14           printf("%d ",a[i])
            end do
          end if
        end do
end proc
```

从上面可以看出，语句号和源程序的实际行号确实不一样，假如我们并不需要所有的语句号，也可以只显示一部分，比如语句 7 到语句 10：

```
> showstat(compute,7..10);

compute := proc()
local i, j, k, m, a, sum;
      ...
   7        if modp(m,i) = 0 then
   8           j := j+1;
   9           a[j] := i;
  10           sum := sum+1
            end if
        ...
end proc
```

7.4.2　设置断点(stopat)

有了语句号我们就能用来设置断点了，Maple 中设置断点的函数是 stopat，其具体的函数表达式如下：

stopat(procName);
stopat(procName, statNum);
stopat(procName, statNum, condition);

参数 procName 是待设置断点的过程的名称，参数 statNum 是需要设置断点的语句号，参数 condition 是一个逻辑表达式。当函数为第一种表达式时，函数在过程的第一条语句设置断点，而为第二种表达式时，函数在过程中指定语句号的语句上设置断点，第三种情况时，则是先判断是否满足逻辑表达式的条件，如果满足则在过程的指定语句号的语句上设置断点，否则将不设置断点。

与设置断点的函数相对应的函数是清除断点的函数 unstopat，它的具体函数表达式如下：

unstopat(procName);
unstopat(procName, statNum);

参数的含义跟上面的完全相同，这里就不再介绍了。当函数为第一种形式时，函数将清除掉指定的过程中的所有断点；第二种形式时将清除过程中指定语句号的断点。

下面我们来设置断点，为了保证过程中只有我们现在设置的这一个断点，我们先用

upstopat 函数来清除所有的断点。并在设置断点后执行过程，看看有什么变化：

```
> unstopat(compute):
  stopat(compute,12);
                                [compute]
> compute();
Error, (in compute) final value in for loop must be numeric or
character
```

我们在语句号为 12 的语句上设置断点，但执行过程后系统给出的错误信息仍然没有变，说明程序中的错误在我们设置的断点的前面，因此有必要重新设置断点，我们观察以后决定将断点设置在语句号 6 上，因为前面的语句都是一些赋值语句，跟系统的给出的错误信息没什么关系：

```
> unstopat(compute):
  stopat(compute,6);
                                [compute]
> compute();
1
compute:
    6*    for i from 2 to k do
            ...
          end do;
```

执行过程后，在语句号 6 的后面出现了一个星号，表明我们在这一条语句上设置了一个断点，过程在这条语句上停止运行，并进入 debug 状态（在执行这个过程后，系统给出 DBG>的提示符，说明我们现在处于调试 debug 状态中）。

在 debug 状态中我们可以查看任何一个变量的值，只需在提示符后面输入变量名即可：

```
DBG > a[1];
1
compute:
    6*    for i from 2 to k do
            ...
          end do;
```

```
DBG > sum;
1
compute:
    6*     for i from 2 to k do
             ...
           end do;
```

我们也可以一次查看多个变量，只需在各个变量之间用逗号隔开即可：

```
DBG > m, j;
1
1
compute:
    6*     for i from 2 to k do
             ...
           end do;
```

光进入调试状态我们还不能干什么有意义的事情，我们还必须能控制过程的运行，并在运行过程中监视各个变量的变化才能找出错误，下一小节我们就来介绍几个这样的函数。

7.4.3　单步执行 next，进入循环 into、跳出循环和子过程 outfrom

如果用过高级语言的编译器，对上面三个函数的功能肯定不会陌生，它们是进行程序调试的常用命令，只是很多编译器都是用菜单命令来实现，而这儿是用语句来实现。我们先来介绍一下这几个函数的具体功能：

next：单步执行，当碰到循环和子过程时将作为一条语句执行，不进入循环和子过程内部。

into：也是单步执行，进入循环但不进入子过程。

outfrom：从循环和子过程中跳出。

下面我们用这几条语句来控制过程的执行，并随时监视过程中我们感兴趣的变量的值，从中找出程序的错误：

首先我们要进入语句 6 的循环：

```
DBG > into;
Error, (in compute) final value in for loop must be numeric
or character
```

进入这个循环时出现错误，我们必须检查一下各个变量的值，我们上面已经查看了变

量 I、j、a 和 sum 的值，现在来看一下其他几个变量的值：

```
> compute();
k
compute:
    6*    for i from 2 to k do
            ...
          end do;
```

```
DBG > k;
k
compute:
    6*    for i from 2 to k do
            ...
          end do;
```

我们期望 k 的值是 sqrt(m)，应该是一个正整数，可是现在却没有赋值，显然前面的语句有问题，仔细观察我们发现 k 的赋值语句少了一个冒号，成了一条判断语句。这是用惯了 C 语言的人最容易犯的一个错误，提醒大家要注意。下面我们将这一错误改回来，重新运行，由于系统此时处于调试阶段，我们先用 quit 命令来退出调试状态：

```
DBG > quit
Warning, computation interrupted
```

```
> restart;
  compute1:=proc()
     local i,j,k,m,a,sum;
     for m from 1 to 1000 do
        k:=trunc(sqrt(m));
        a[1]:=1,j:=1;sum:=1;
        for i from 2 to k do
           if(modp(m,i)=0) then
              j:=j+1;
              a[j]:=i;
              sum:=sum+i;
           end if;
        end do;
        if(sum=m) then
```

```
                printf("\n%d  its factors are",m);
                for i from 1 to j do
                  pfintf("%d,",a[i]);
                end do;
              end if;
          end do;
          end:
> compute1();
1  its factors are1,
```

修改回来后程序不再出现错误信息，但运行的结果还是跟我们的期望值有很大的出入，因为我们知道至少 6 应该是一个“完数”，这种错误是调试程序时最不愿碰到的，我们从下面可以看出调试这种程序有多麻烦。

为了设置断点我们还是先来看看程序的语句号：

```
> showstat(compute1);

compute1 := proc()
local i, j, k, m, a, sum;
    1   for m to 1000 do
    2     k := trunc(sqrt(m));
    3     a[1] := 1;
    4     j := 1;
    5     sum := 1;
    6     for i from 2 to k do
    7       if modp(m,i) = 0 then
    8         j := j+1;
    9         a[j] := i;
   10         sum := sum+i;
            end if
          end do;
   11     if sum = m then
   12       printf("\n%d its factors are ",m);
   13       for i to j do
   14         printf('%d,",a[i])
```

```
>              end do
            end if
        end do
end proc
```

我们在语句号为 1 的语句上设置语句号，然后执行过程进入 debug 状态。

```
> stopat(compute1,1):

> compute1();
compute1:
   1*  for m to 1000 do
         ...
       end do
```

我们用 into 语句进入循环内部，由于 6 是一个“完数”，而 2 到 5 都不是，为了调试时比较明显，我们需要在 m=6 时进行调试．现在让我们看看 m 的值是多少：

```
> DBG > into
compute1:
    2     k := trunc(sqrt(m));

DBG > m;
1
compute1:
    2     k := trunc(sqrt(m));
```

现在 m 的值为 1，我们需要执行循环 5 次才能使 m 的值增加为 6，首先我们用 next 语句来调试程序，看看一条 next 语句能引起什么变化：

```
DBG > next;
1
compute1:
    3       a[1]:=1;
```

```
DBG > m;
1
compute1:
    3         a[1] := 1;
```

运行完以后 m 的值没有变，但执行了一条语句，为了到达 m=6 我们还有很长的路要走，我们需要反复运行 next 语句，使 m=6，我们的做法是将光标移到 next 语句上，回车执行一条语句，然后查看 m 的值，如果 m=6 时先暂停，否则再将光标移到 next 语句上，直到 m 值为 6，如下：

```
DBG > next;
1
compute1:
    2     k := trunc(sqrt(m));
```

```
DBG > m;
6
compute1:
    2     k := trunc(sqrt(m));
```

现在 m 的值已经为 6，达到我们预设的特殊值。接下来就是用 next 语句和 into 语句来一步步跟踪求完数的过程：

首先我们运行三次 next 语句使程序执行到语句 6 上：

```
DBG > next;
1
compute1:
    6     for 1 from 2 to k do
            ...
          end do;
```

接下去是一个循环，为了检查变量我们进入这个循环看看：

```
DBG> into;
1
compute1:
    7    if modp(m,i) = 0 then
           ...
         end if
```

接下去碰到一个选择语句，由于这是我们判断一个数是否为“完数”的核心，我们还是用 into 语句进入选择语句看看有没有错误，为了比较在执行选择语句前后各个变量的值，我们先来看看各个变量的值：

```
DBG> m,i,j,a[j],sum;
6
2
1
1
1
compute1:
                ...
              end if
```

在进入选择结构前 i=2，j=1，a[1]=1，sum=1，这些值都符合我们的要求，没有什么错误。我们接下来进入选择结构：

```
DBG> into;
1
compute1:
    8        j := j+1;
```

现在程序停留在选择语句的第一条赋值语句上，为了看看选择语句结束后，各个变量的值是否还满足要求，我们用 next 语句单步执行完成这层结构：

```
DBG> next;
2
compute1:
    9        a[j] := i;
```

```
DBG > next;
2
compute1:
    10          sum := sum + i
DBG > next;
3
compute1:
    11      if sum := sum + i
               ...
            end if
```

现在上一层选择结构已经结束，并且循环结构也结束了，下面来看看变量的值是否正确，程序是否是正常结束循环的：

```
DBG > m,i,j,a[j],sum;
6
3
2
2
3
compute1:
   11     if sum = m then
             ...
          end if
```

现在 i 的值为 3，大于 sqrt(6)，因此循环结束是正常的；而 sum 的值是 3，并不像我们想象中的那样是 6，因此肯定哪儿除了问题。我们再仔细观察一下其他的变量，按照我们的期望值，此时 j 的值应该是 3，因为 6 的因子有 3 个：1、2、3，我们在程序中肯定哪儿丢了一个因子，那到底丢的是哪个因子呢？这个可以从数组 a 的值入手，a[1]=1 是我们定义的，这没有错误，从上面我们有看到 a[2]=2，说明因子 2 也已经包含进去了，现在就看因子 3 是否已经包含进来，假如有因子 3 的话，它应该被保存在 a[3]里，下面我们来看看此时 a[3]是多少：

```
DBG > a[3];
a[3]
compute1:
    11      if sum := m then
               ...
            end if
```

我们发现因子 3 不在 a[3]里，因子 3 确实没有被包含进来，那是什么原因没有包含进 3 的呢？我们回过头看看源程序：本来我们是想利用判断一个正整数 m 是否为素数的方法来求一个数的因子，这样我们只需要判断 1 到 sqrt(m)之间有多少个因子即可，但我们还应该想到当 n(1<n≤sqrt(m))是一个因子时，m/n 也应该是 m 的一个因子，并且这个因子不在[1, sqrt(m)]中，我们在判别的时候应该加上这个因子，因此源程序就应该加上处理这个因子的语句，按照以上的思路我们将源程序修改为：

```
> restart;
  compute2:=proc()
     local i,j,k,m,a,sum;
     for m from 1 to 1000 do
        k:=trunc(sqrt(m));
        a[1]:=1,j:=1;sum:=1;
        for i from 2 to k do
           if(modp(m,i)=0) then
              j:=j+1;
              a[j]:=i;
              sum:=sum+i;
              if(m/i=i) then
                   break;
              else
                   j:=j+1;
                   a[j]:=m/i;
                   sum:=sum+m/i;
               end if;
           end if;
        end do;
        if (sum=m) then

>            printf("\n%d  its factors are ",m);
             for i from 1 to j do
                printf("%d,",a[i]);
             end do;
          end if;
       end do;
       end:
```

然后我们运行一下修改后的程序看结果是否正常：

```
> compute2();


1  its factors are 1,
6  its factors are 1,2,3
28  its factors are 1,2,14,4,7,
496  its factors are 1,2,248,4,124,8,62,16,31,
```

现在这个程序已经正常了。我们看到要调试一个程序是很麻烦的，特别是在观察各个变量的值寻找错误时。在上面调试程序的时候还碰到一件很单调的事情：一次次的运行 next 语句使 m=6，这主要是我们的程序设计结构有问题：我们将判断一个正整数是否为“完数”和输出所有的“完数”的语句都写在同一个过程内，这样造成了要调试“判断完数”的部分就必须经过一层层的“输出部分”。我们设计程序的时候要尽量避免这种情况的发生，而解决这种问题的方法就是利用子过程调用，将判断一个正整数是否为“完数”的部分用一个子过程来判断，然后整个过程再调用这个子过程即可，这样进行程序调试的时候就不需要对整个过程进行调试，只需要调试判断部分的子过程就行了。下面我们给出这种结构的源程序，读者可以在子过程中加入错误语句，练习一下单独调试的情况：

```
> Test:=proc(m::positive);
        local i,j,k,a,sum;
           k:=trunc(sqrt(m));
           a[1]:=1,j:=1;sum:=1;
           for i from 2 to k do
             if(modp(m,i)=0) then
               j:=j+1;
               a[j]:=i;
               sum:=sum+i;
               if(m/i=i) then
                    break;
               else
                    j:=j+1;
                    a[j]:=m/i;
                    sum:=sum+m/i;
               end if;
             end if;
         end do;
         if(sum=m) then
```

```
            if(sum=m) then
               return(true,j,a);
            else
               return(false);
            end if
         end:
> Compute:=proc();
          local m,a,i;
          for m from 1 to 1000 do
              a:=Test(m);
              if(a[1]=true) then
                printf("%d  its factors are ",m);
                for i from 1 to a[2] do
                   printf("%d,",a[3][i]);
                   end do;
                   printf("\n");
                end if;
            end do;
            end:
> Compute();
`1  its factors are 1,`
`6  its factors are 1,2,3`
`28  its factors are 1,2,14,4,7,`
`496  its factors are 1,2,248,4,124,8,62,16,31,
```

显然上面这种程序设计的方法比以前的要明了得多，程序调试起来也比较容易。

7.4.4 其他一些常用调试命令介绍

除了以上那些命令可以进行程序调试以外，我们还经常用到一些其他的命令，比如 DEBUG 和 cont。

DEBUG 函数用来在过程中设置断点，但它与 stopat 不同，DEBUG 函数要加入过程中才能起作用。当程序执行完 DEBUG 命令后，马上暂停程序的运行并调用程序调试器进入程序调试状态，它的函数表达式如下：

DEBUG ();
DEBUG(字符串);
DEBUG(逻辑表达式);

DEBUG 函数可以有三种形式，当为第一种情况时，过程只是进入调试状态；不显示其他信息，而当其参数是字符串时，过程显示字符串的内容并进入调试状态；当函数为第三种形式时，函数首先判断逻辑表达式的真假，当逻辑表达式为真时，过程进入调试状态，否则忽略 DEBUG 命令。

如果我们还记得上面用 next 命令来寻找 m=6 时的痛苦，现在就能很自然的想起用这个函数完成在 m=6 时自动进入调试状态，下面我们将看到用这种方法来调试将有多方便。首先我们在 for m from...和 k:=trunc(...)语句之间加入语句 DEBUG(m=6)：

```
for m from 1 to 1000 do
    DEBUG(m=6);
    k:=trunc(sqrt(m));
```

然后我们运行程序：

```
> compute1();
1  its factors are 1,
1
compute1:
    3    k := trunc(sqrt(m));
```

我们看到程序停留在 DEBUG 语句的后面，并且不用我们在外面设置断点自动进入了调试状态，接下来我们来看看现在 m 的值是多少：

```
DBG> m;
6
compute1:
```

我们看到用这种方法很容易就能到达我们期望的调试状态，比我们在上面用 next 语句一次次的执行并得随时监视 m 的值方便多了。并且用这种方法我们不必用 showstat 来显示语句号以及用 stopat 根据语句号来设置断点。

在源程序中设置显示断点 DEBUG 后，我们可以用 cont 命令来控制程序的执行，cont 命令是使程序运行到下一个用 DEBUG 设置的断点为止，当 DEBUG 命令在循环内时，cont 命令将使程序运行到下一次循环的 DEBUG 语句后，来看一下下面的例子。

```
> f := proc(x::numeric,y::numeric)
      local i,a;
      a := x^2;
      DEBUG();
      for i to 5 do
          a:= y^i;
          printf("%d\n",a);
          DEBUG(`Hello`);
      end do;
      a := (x+y)^2
  end:
```

当我们运行程序后，程序将停留在 DEBUG（）后面并进入调试状态：

```
> f(2,3);
`4`
`f:`
`    3     for i to 5 do`
`            ...`
`          end do:`
``
```

此时的 a 为 4，当输入 cont 命令后，语句将停留在第二个 DEBUG 命令后：

```
DBG> cont;
`3`
Hello
`f:`
`    4      a := y^i;`
``
```

现在再运行一次 cont 命令，语句还是停留在第二 DEBUG 命令后，但此时的 a 已经与上次不同，说明程序已经进行了一次循环，是在第二次进入循环后程序才暂停的：

```
DBG > cont;
`9`
Hello
`f:`
`    4       a := y^i;
``
```

一般我们在调试程序的时候 DEBUG 是一个非常有用的工具，它与上面介绍过的那些程序联合使用能够简化我们程序调试的工作量，使程序调试灵活多变。

7.5　文件输入/输出

很多时候我们希望能够从一个具有固定格式的文件中读入数据用来计算或进行字符处理，或将一些数据以各种格式写入文件中保存起来，这时我们就要进行文件的输入/输出操作，这一节我们将接触一些比较简单的文件读写操作。

7.5.1　文件的打开与关闭

和其他的高级语言不一样，Maple 中对文件的读和写之前我们可以不“打开”该文件，但由于我们习惯了在使用文件之前进行打开文件，然后再使用文件指针（在 Maple 中叫文件描述），在使用结束之后应关闭该文件。

Maple 中打开文件的函数是 fopen，这个跟 C 语言中是一样的，它的函数表达式如下：

fopen(name, mode);
fopen(name, mode, type)

第一个参数 name 是指待打开的文件的文件名，当文件不在默认目录下时，需要指定完整的路径名，与 C 语言的规则相同，路径名中的“\”需要用双反斜杆“\\”来代替，比如要打开 E 盘 temp 目录下的 a.dat 文件，其完整的路径文件名为：e:\\temp\\a.dat。

第二个参数是设置文件打开的模式，在 Maple 中共有三种模式可以打开：

（1）READ：文件以只读方式打开，当文件不存在时，函数返回一个文件不存在的错误的信息；而当文件已经被打开时，函数也将返回一个已经打开的错误信息；如果文件被成功打开，将返回一个文件的描述（是一个短整数）。

（2）WRITE：文件以只写的方式打开，当文件不存在时，函数将新建一个文件；当文件已经存在时，并且没有被打开时，文件将被清空，并返回一个文件的描述；而当文件已经存在并已经被打开时，函数将返回一个文件已经打开的错误信息。

（3）APPEND：文件的打开方式与 WRITE 参数类似，只是用这个参数后，函数不清

空已存在的文件而是在文件的末尾追加数据。

第三个参数 type 指定打开文件的类型，可以取以下两种形式：

（1）TEXT：文件以文本的形式打开。

（2）BINARY：文件以二进制的方式打开。

这个参数是可选的，当不设这个参数时，函数默认以文本方式打开。

在 Maple 中关闭文件的函数是 fclose，其具体函数表达式如下：

fclose(fps);

参数 fps 是已打开文件的描述，可以同时关闭多个文件。下面我们用一个简单的例子来练习一下文件的“打开”和“关闭”命令。

```
> fp1:=fopen("e:\\temp\\b.txt",APPEND,TEXT);
                              fp1 := 3
> fprintf(fp1,"1000");
                                4
> fclose(fp1);
```

现在我们到 e 盘 temp 目录下将看到一个文件名 b.txt 的文本文件，用记事本打开可以看到里面有一个“1000”的数据。下面我们用打开命令来打开它：

```
> fp2:=fopen("e:\\temp\\b.txt",REND,TEXT);
                              fp2 := 3
> i:=fscanf(fp2,"%d");
                              i := [1000]
> i;
                               [1000]
> fclose(fp2);
```

7.5.2 从文件中简单的输入和输出

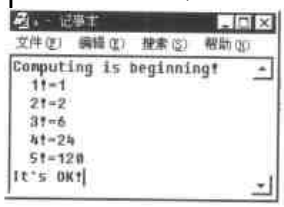
在本章第一小节我们曾经详细介绍了向屏幕或管道进行格式化输出的函数 printf，如果读者对其还没有忘的话，那现在学习起来就比较容易了，对文件进行输入操作的函数 fprintf 的结构和功能跟 printf 几乎完全一样的，所不同的是它们的输出对象不一样，还有就是函数 fprintf 比函数 fprintf 多一个参数 fps（输入文件对象的描述）。下面我们来看看它的具体函数表达式：

fprintf(fps, fmt, x_1, ..., x_n);

参数 fmt 和 x_1,...,x_n 的含义跟函数 printf 中的完全一样，这里就不再介绍了，读者可以

参看 7.1.6 节。参数 fps 是待写入的文件的描述，下面我们来看一个例子，顺便熟悉一下文件的打开和关闭：

```
> Test:=proc(n::positive)
        local i,j,k,fps;
        fps:=fopen("e:\\user\\a.txt",WRITE);
        fprintf(fps,"Computing is beginning!\n");
        for i to n do
            k:=1;
            j to i do
            k:=k*j;
            do;
            ntf(fps,"%3d!=%d\n",i,k);
        end do;
        fprintf(fps,"It's OK!");
        fclose(fps);
        end:
> Test(5);
```



这个过程是打开（或新建）一个文本文件，将求阶层的结果写入文件中，我们看到向文件中输入数据的函数 fprintf 非常的简单，使用也很灵活，下面我们运行这个过程，并可以用写字板打开我们输入数据的文件 a.txt（这个文件在 e:\user 下），如图 7-1 所示：

图 7-1　笔记本打开的数据文件

下面我们再来看一下从文件中读入数据的函数 fscanf，它的函数表达式如下：

fscanf(fps,fmt);

参数的形式和含义几乎和函数 fprintf 的相同，这里就不再介绍了，我们还是从例子中得到知识吧，下面这个例子是将上面生成的文件 a.txt 打开并输出到屏幕上显示：

```
> TestScanf:=proc()
      local s,fps;
      fps:=fopen("e:\\user\\a.txt",READ);
      while(feof(fps)=false) do
          s:=fscanf(fps,%s);
          printf("%s\n",op(s));
      end do;
      fclose(fps);
      end proc:
```

其中用到了一个函数 feof，这个函数用来判断文件描述指针是否已经到达文件的末尾，假如到文件末尾则返回 ture，否则返回 false。在对文件进行操作时要注意文件的指针会自动在文件中移动，比如我们在上面用 fscanf 从文件指针 fps 所指的文件 a.txt 中读取一个字符串时，指针 fps 会自动移动到这个字符串的后面，而无须我们控制。

下面我们来看看运行的结果：

```
> TestScanf();
Computing
is
beginning!
1!=1
2!=2
3!=6
4!=24
5!=120
It's
OK!
```

我们确实是将文件中的内容都读了出来，并显示在计算机屏幕上，但有一点遗憾的是输出的格式并不与它们在文件中的格式吻合，这是很头疼的问题，主要是我们用 fcanf 读取字符串，当系统碰到空格时，函数将认为字符串已经结束，而不是将我们原来输入的整个字符串当一个字符串。这种难题用 fscanf 函数很难解决，我们需要一种功能跟强大的输入输出函数，下面一小节就将介绍这一种函数。

7.5.3 writeline 和 readline

上面我们看到 fscanf 函数读入一行数据时会比较麻烦，而 writelline 和 readline 函数则能很容易的实现对文件或管道进行整行输入输出的操作，下面我们来看一下这两个函数的

具体表达式：

writeline(file, str ...):

参数 file 是待写入的文件的描述或文件名，参数 str 是待写入的字符串，可以为空，也可以是多个字符串。当参数 file 为默认时，字符串被写入当前的流中（通常是显示器的屏幕），当 file 是一个文件名时，函数将先判断文件是否已经被打开，当未被打开时，函数将自动打开该文件，如果文件是以只读方式打开时，函数将以只写并且是文本的方式打开。假如参数 str 为空的话，函数将只打印一新行。

我们来看一个具体的例子：

```
> write:=proc()
        local fps;
        fps:=fopen("f:\\xx\\a.txt",WRITE,TEXT);
        writeline(fps);
        writeline(fps,"It's OK!","The File is
  end!");
        writeline(default);
        writeline(default,"It's OK!","The File is
  end!");
        fclose(fps);
    end proc:
> write();


It's OK!
The File is end!
```

我们在屏幕上看到当待输入的字符串为空时，函数只是打印一个空行，我们打开文件 a.txt 也将看到同样的效果。下面我们用 readline 函数将我们刚刚输入到文件中的字符串给读出来，并输入到屏幕上，在使用这个函数之前我们先来看看它的具体表达式：

readline(file):

readline():

参数 file 跟上面的相同，是一个文件名或文件的描述。Readline()和 readline(-1)以及 readline(default)的作用是一样的，都是从默认流（一般都是屏幕）中读取一行字符串，字符串的位置由当前的流的指针而定；当函数从文件中读取数据时，函数将从当前指针的下一行读入一行字符串，假如指针不是在文件的末尾，函数将返回一个字符串，否则将返回 0，利用这一点我们能够判断指针是否已经到了文件末尾，假如参数 file 是文件名时，函数将首先判断文件是否已经打开，假如没有打开，函数将自动以只读方式文本模式打开，读者

看完下面的例子就能对这个函数有比较深的理解：

```
> read1:=proc()
          local string;
          string:=readline();
          writeline(default,string);
          string:=readline("f: \\xx\\a.txt");
          while(string<>0) do
             writeline(default,string);
             string:=readline("f:\\xx\\a.txt");
          end do
      end proc:
> read1();
       string:=readline();

It's OK!
The File is end!
                                        0
```

我们看到从屏幕上读到的字符串刚好是自己那条语句，这是比较凑巧，假如我们再运行一下这个过程，我们将看到从屏幕中的读入的数据已经改变，已经是下面一条语句的整行字符串（包括空格），从这个运行结果我们很清晰的看到函数 readline 读取数据后指针移动的结果：

```
> read1();
       writeline(default,string);

It's OK!
The File is end!
                                        0
```

7.5.4 writebytes 和 readbytes

Maple 除了提供对单个字符串（中间没有空格）和一行字符串进行文件的输入输出外，还提供了一个功能很强大的可以输入输出任意个字符的函数 writebytes 和 readbytes，利用这两个函数以及上面介绍的几个函数，我们就能很方便地处理文本和数值型方式的输入输出。

下面让我们先来看一下函数 writebytes 的具体表达形式：

writebytes(file, bytes):

参数 file 是文件名或文件的描述；参数 bytes 是待写入的字符串，或整数列表。假如参数 file 是文件名且文件并未打开，或者打开方式是只读方式，函数将重新以只写的方式打开；而当参数 file 是 dafault 时，函数将向屏幕（一般的默认流）上输出。当参数 bytes 为整数列表时，函数将自动将整数转换成这个整数对应的字符，假如整数大于 255，函数将不能执行这条命令，返回一个错误信息。如果函数成功执行，将返回写入的字符数。我们来看一个简单的例子：

```
> writeb:=proc()
      writebytes("c:\\a.txt","This is a
  test!\nThe file is over!\n");
      writebytes("c:\\a.txt",[112,225,66]);
      fclose("c:\\a.txt);
      writebytes(default,"This is a text!\nThe
  file is over!\n");
      writebytes(default,[112,225,66]);
  end proc:
```

```
> writeb();
`This is a text!`
`The file is over!`
`pyB`

                    3
```

我们看到函数最后返回一个 3，这是由最后一条 writebytes 返回的，其他几个语句返回的字符数只是没有显示出来而已。下面我们来看看当我们写入一个大于 255 的整数的时候将发生什么事情：

```
> writebytes(default,[300,666]);
Error, (in writebytes) byte value is out of range
```

下面我们来看一下函数 readbytes 的具体表达式：

readbytes(file,len,TEXT);

参数 file 是保存数据的文件名或文件的描述；参数 len 则是需要读入的字节数，当这个参数省略时，函数将只读入一个字节（文件指针所指的下一个字节），当文件中的字节数小于所给的长度时，函数将只读入实际的字节数；参数 TEXT 用来指定读入的数据以字符的

形式存储。函数执行完以后将返回实际读入的字节数，当文件指针在文件的末尾时，函数将返回 0，利用这一点我们能够判断文件是否已经结束。下面我们来看一个例子：

```
> fclose("c:\\a.txt");
  readbytes("c:\\a.txt",infinity,TEXT);
                "This is a text!\nThe file is over!\npyB"
```

为了保证读入的数据是从文件开头开始的，我们先将文件关闭，这样函数 readbytes 必须重新打开文件，文件指针指定在文件的开头。在这个例子中，我们指定文件以文本方式读入，且读入的字节数是正无穷，由于文件中的字节不可能是无穷个，所以能保证我们将整个文件中的数据读出来。

假如我们现在想要的数据不是字符型的，而是整型的，我们只要不指定参数 TEXT 即可，如下：

```
> fclose("c:\\a.txt");
  readbytes("c:\\a.txt",infinity
[84,104,105,115,32,105,115,32,97,32,116,101,115,116,33,13,10,
  84,104,101,32,102,105,108,101,32,105,115,32,111,118,101,114,
  33,13,10,112,225,66]
```

对照 ASCII 码表可以发现这些值其实就是上面那些字符对应的值。下面我们来看看在不指定读入字符长度的情况下读入数据的情况：

```
> fclose("c:\\a.txt");
  reaadbytes("c:\\a.txt");
                              [84]
```

函数现在返回的是第一个字符，假如我们再次执行 readbytes 函数，将发现读入的是第二个字符：

```
> reaadbytes("c:\\a.txt");
                              [104]
```

依此类推，我们用这种方法也能很容易将一个文件中所有的数据都读出来。

7.5.6 writedata 和 readdata

我们看到上面几个函数都是以字符形式来保存各种输入输出的，当我们想输入某个数字并且希望保存的时候仍然是数值形式存在，这时我们就可以利用函数 writedata 和 readdata。它们的具体函数表达式如下：

writedata(fileID, data, format, default);

readdata(fileID, format, n)

参数 fileID 是待写入或读入的文件名或文件的描述，当 fileID 参数设为 terminal 时，操作就是对默认终端（一般是指屏幕）进行的。参数 data 是待写入的数据，参数 foramt 设定读写的格式（整型、浮点型还是字符串），这是一个可选的参数，当这个参数被省略时，默认为浮点型。参数 n 用来指定读入的数据的列数。下面我们来举一个例子，通过这个例子我们可以对这两个函数有一个比较清晰的了解。

首先我们定义一个数组：

```
> A:=array([[1.5,2.2.,3.4],[2.7,3.4,5.6],[1.8,3.1,6
  .7]] );
```

$$A := \begin{bmatrix} .5 & 2.2 & 3.4 \\ 2.7 & 3.4 & 5.6 \\ .8 & 3.1 & 6.7 \end{bmatrix}$$

然后我们将这个数组写入文件 a.dat 和屏幕上，从中能够看出 writedata 函数与其他几个函数的区别：

```
> fclose("c:\\a.dat");
  writedata("c:\\a.dat",A);
  fclose("c"\\a.dat");
  writedata(terminal,A,integer);
1 2 3
2 3 5
1 3 6
```

我们往文件中写的是浮点型数据，而往屏幕上写的是整型数据（它是将浮点数取整而不是四舍五入），读者可以比较这两者的区别，下面我们来看看函数 readdata 的使用，首先我们不指定读入的列数：

```
> readdata("c:\\a.dat",integer);
                          [1,2,1]
```

可以看到当不指定读入的列数时，函数只读入一列数据，假如想读入多列数据我们必须指定，当指定列数大于实际列数时，函数将返回实际的列数：

```
> readdata("c:\\a.dat",5);
          [[1.5,2.2,3.4],[2.7,3.4,5.6],[1.8,3.1,6.7]]
```

7.6 代码转换

通过前面几节的学习，我们可以看到用 Maple 进行编程比用 Fortran 和 C 语言直接编程要简单得多，这主要是 Maple 中包含有丰富的常用函数，而在 C 和 Fortran 中数学类的库函数不可能这么多，比如矩阵求逆公式在 Maple 中是可以直接使用的，而在其他两种语言中都要自己编写，或到网上、书上找到源程序才能实现，这样用起来有时很不方便。但是有时我们确实需要在 C 中或 Fortran 中能实现这种功能，这时最好的办法就是用 Maple 先实现数学模型的处理，然后再用 Maple 产生相应的 C 或 Fortran 程序代码。

本节我们来看看如何利用 Maple 函数，将 Maple 语言转化为 Fortran 和 C 语言以及 LaTeX 语言。

7.6.1 生成 Fortran 代码

能将 Maple 程序转换成 Fortran 程序代码的函数是 fortran，其具体函数表达式为：

```
fortran(s);
fortran(s,option);
```

其中参数 s 是 Maple 的表达式，可以是普通的代数表达式，也可以是代数表达式的矩阵，还可以是等式列表，当然也可以是一个过程（或叫子程序）。参数 option 是一个可选的参数，它可以是一些等式的集合，用来设置转换以后的格式，如 precision=double（转换后的数值常数为双精度类型），mode=single（转换后的 Fortran 函数取值为单精度类型）等等。参数的选择可以多种多样，读者有兴趣可以查看帮助。

下面我们来考虑 2 阶矩阵求逆的例子。在 Maple 中，我们可以输入：

```
> A:=matrix(2,2);
```

$$A := \mathrm{array}(1..2,1..2,[])$$

```
> IA:=linalg[inverse](A);
```

$$LA := \begin{bmatrix} \dfrac{A_{2,2}}{A_{1,1}A_{2,2} - A_{1,2}A_{2,1}} & -\dfrac{A_{1,2}}{A_{1,1}A_{2,2} - A_{1,2}A_{2,1}} \\ -\dfrac{A_{2,1}}{A_{1,1}A_{2,2} - A_{1,2}A_{2,1}} & \dfrac{A_{1,1}}{A_{1,1}A_{2,2} - A_{1,2}A_{2,1}} \end{bmatrix}$$

对定义的矩阵 A，我们调用 linalg 程序包函数 inverse 很快就能得到逆矩阵的一般形式。我们可以用 fortran 函数将逆矩阵 IA 用 Fortran 代码表示出来。这样，我们就可以直接在 Fortran 程序中应用上面的代码。

```
> with(codegen, fortran):
  fortran(IA);
      IA(1,1) = A(2,2)/(A(1,1)*A(2,2)-A(1,2)*A(2,2))
      IA(1,2) = -A(1,2)/ A(1,1)*A(2,2)-A(1,2)*A(2,1))
      IA(2,1) = -A(2,1)/ A(1,1)*A(2,2)-A(1,2)*A(2,1))
      IA(2,2) = A(1,1)/(A(1,1)*A(2,2)-A(1,2)*A(2,1))
```

这只是一个简单的例子，如果要求更高阶的矩阵的逆矩阵，用同样的步骤即可，由于矩阵阶数上升后，运算量将成指数增长，转换成的代码长度也将成几何级数增长，我们在这里就不介绍了。

下面我们来看看我们如何用 fortran 将自己编写的 Maple 过程转换成 Fortran 源代码：

```
> test:=proc(x::numeric)
      local y;
      y:=x^2*exp(x)+Pi;
      end proc:
> with(codegen,fortran):
  Fortran(test,precision=double);
          doubleprecision function test(x)
          doubleprecision x

          doubleprecision y

            test = x**2*exp(x)+0.3141592653589793D1
            return
          end
```

7.6.2　生成 C 代码

同样，我们调用函数 C 可以产生相应的优化的 C 代码。它的具体函数表达式如下：

C(s):
C(s,options);

这个函数中参数的含义与 fortran 的完全相同。由于 C 这个符号常常被使用，因此 Maple 需要将该函数加载进内存才能使用。如下所示。

```
> with(codegen, c):
  C(IA);
      IA[0][0] = A[1][1]/(A[0][0]*A[1][1]-A[0][1]*A[1][0]);
      IA[0][1] = -A[0][1]/(A[0][0]*A[1][1]-A[0][1]*A[1][0]);
      IA[1][0] = -A[1][0]/(A[0][0]*A[1][1]-A[0][1]*A[1][0]);
      IA[1][1] = A[0][0]/(A[0][0]*A[1][1]-A[0][1]*A[1][0]);
```

在本小节结束之前，我们再来转化一个例子。这个例子中定义了一个过程 f，它是一个曲面的表达式。然后我们用 Maple 计算函数 f 对 x，y 的梯度：

```
> F:=proc(x,y)
      local t;
      t:=x*y;
      x+t-y*t
      end proc:
```

```
> F(x,y);
```

$$x + xy - y^2 x$$

```
> G:=GRADIENT(F);
G := proc(x, y)
local df, t;
    t := x * y;
    df := array(1..1);
    df[1] := 1 - y;
    RETURN(1 + df[1] * y, -t + df[1] * x)
end proc
```

```
> G(x,y);
```

$$1 + (1 - y)y, -xy + (1 - y)x$$

然后我们打开程序 codengen 程序包，将这个函数在转换成 C 代码：

```
> with(codegen):
  C(G);
void G(x,y,crea_par)
double x;
double y;
double crea_par[2];
{
  double df[1];
  double t;
  {
    t = x*y;
    df[0] = 1.0-y;
    crea_par[0] = 1.0+df[0]*y;
    crea_par[1] = -t+df[0]*x;
    return;
  }
}
```

7.6.3　生成 Latex 代码

能将 Maple 代码转换成 Latex 格式的函数是 latex，它的具体函数表达式如下：

latex(expr)
latex(expr, filename)
latex(expr, filename, 'append')

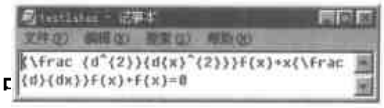
参数 expr 是待转换的表达式，从它的函数表达式我们就可以看出能够转换成 latex 形式的代码不如转换成 C 和 Fortran 的多，它只能转换表达式；参数 filename 是转换后的代码写入的文件名；而参数 append 则指定了代码将不覆盖原文件中的所有内容而是将代码加入到文件的尾部。我们先来看一个例子：

```
> x:='x':f:='f':
  f:=x->f(x);
  eq:=diff(f(x), x$2)+x*diff(f(x),x)+f(x)=0;
```

$$f := f$$

$$eq := \left(\frac{\partial^2}{\partial x^2} f(x)\right) + x\left(\frac{\partial^2}{\partial x^2} f(x)\right) + f(x) = 0$$

```
> latex(eq);
{\frac {d^{2}}{d{x}^{2}}}f(x)+x{\frac {d}{dx}}f(x)+f(x)=0
```



，所以代码在屏幕上输出，假如我们需要将代码输入到文件中，则需要给出一个文件名（注意：当文件名带后缀时，必须用双引号包含进去）：

```
> latex(eq,"testlatex.txt");
```

这时我们可以在 Maple 的安装目录下找到 testlatex.txt 文件，用写字板打开可以看到转换后的 latex 代码，如图 7-2 所示：

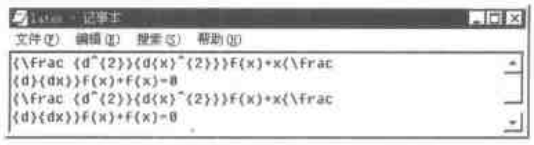


图 7-2 记事本打开的 latex 代码

假如我们想把输出文件放在 Maple 目录下的 users 子目录中的话，可以用下面这种方法实现：

```
> filename:="users\\latex.txt":
  latex(eq,filename);
```

现在打开 Maple 安装目录底下的 users 目录我们就能发现 latex.txt 文件，里面内容跟上面是一样的。假如我们想在文件的后面再加入 latex 代码，只需使用 append 参数就行了，如下所示：

```
> latex(eq,filename,append);
```

现在打开文件 latex.txt，可以发现文件中又多了一行代码，见图 7-3：

图 7-3 使用 append 后的代码

7.7　Maple 与其他软件的接口

Maple 除了能够自己单独处理各种问题以外，它还能利用别的软件的程序包处理问题，别的软件也能利用 Maple 进行某些问题的处理，本节我们介绍一下 Maple 和 Matlab 以及 Excel2000 的接口。

7.7.1　Maple 调用 Matlab 的函数

Maple 中包含一个 matlab 程序包，利用这个程序包我们就能在 Maple 中像在 Matlab 中一样使用 Matlab 函数，它的使用方法如下：

```
with(matlab): function(M, ...);
Matlab[function](M, ...);
```

我们可以看到调用函数使用起来特别的方便，不过需要注意的是在 Maple 中不能使用所有的 Matlab 函数，只能使用其中的一部分，它们是：

chol　loselink　defined　det　dimensions　eig　evalM　fft　getvar　inv　lu　ode45　openlink　qr　setup　setvar　size　square　transpose

这些函数的功能和具体使用方法请参看有关 Matlab 的书籍或 Maple 中的相关帮助，这里就不再介绍了。下面我们来举一个例子来看看如何调用 Matlab 中的函数：

```
> maplematrix_a:=matrix(2,3,[1,2,3,4,5,6]);
```

$$maplematrix\_a := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
> maplematrix_b:=matrix(3,3,[9,8,7,6,5,4,3,2,1]]);
```

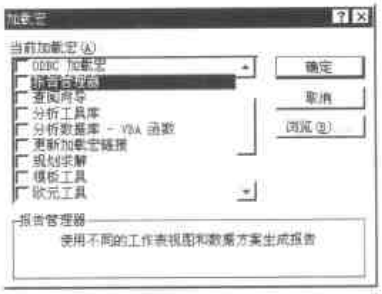
$$maplematrix\_b := \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

```
> Matlab[dimensions](maplematrix_a);
                        [2,3]
> Matlab[dimensions](maplematrix_b);
                        [3,3]
```

注意在调用 Matlab 程序包中的某些函数时需要安装动态连接库 libeng.dll，否则将不能进行正常调用，在安装的时候应该尽量完全安装。

7.7.2 在 Excel2000 中调用 Maple 函数

Maple 6 比以前的版本增加了一些新的功能，其中一项就是增加了与 Excel 2000 的接口，使我们能够在 Excel 2000 和 Maple 6 之间互相调用。不过如果读者使用的是简体中文



能直接使用它们之间的接口，必须先在 Excel2000 中先加载宏，下面介绍如何在 Excel2000 加载宏，然后再介绍它们之间的互相

Maple 6 宏

ple 6 宏很简单，读者只需按照下面的步骤操作即可：

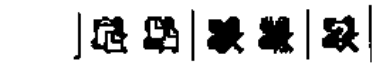
（1）单击 Excel 2000 菜单的“工具”→“加载宏”，弹出一个加载宏的对话框，如图 7-4 所示。

图 7-4 加载 Maple 6 宏的初始对话框

（2）单击“浏览”按钮，在 Maple 6 的安装目录中找到具有 Excel 图标的名为 WMIMPLEX.xla 的文件，双击就能将 Maple 6 宏加入到 Excell2000 中。

（3）这时我们在加载宏的对话框中将发现多了一项“Maple 6 Excel Add-in”，确认前面的复选框已经被选中，然后单击“确定”按钮，如图 7-5 所示。

（4）这样我们在 Excel2000 的工具条上就多了几个如下图标：



这说明我们已经将 Maple 6 宏加入到 Excel2000 中了，下面我们就在 Maple 6 和 Excel2000 中互相调用了。

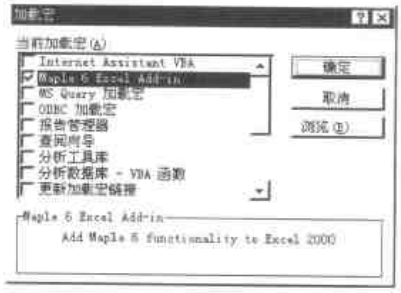


图 7-5　加载完 Maple 6 宏后的对话框

2. 将 Excel2000 中的数据拷贝到 Maple 6 中

在上面几个工具栏中的第一个图标就是将 Excel 2000 中的数据拷贝到 Maple 6 中，它使我们能很方便的将 Excel 2000 中的一些结果输入到 Maple 6 中，我们来看下面的例子，是将 Excel 2000 中的一些 3×3 的数据导入到 Maple 6 中成为 3×3 矩阵，首先我们在 Excel 2000 中输入下列数据：

	A	B	C
1	1	4	7
2	2	5	8
3	3	6	9

然后选中需要拷贝的数据，再用鼠标单击“Copy From Excel to Maple”图标，在 Maple 6 中单击“粘贴”图标或使用键盘“Ctrl+V”，就能将这个矩阵拷贝到 Maple 6 中，得到的结果如下：

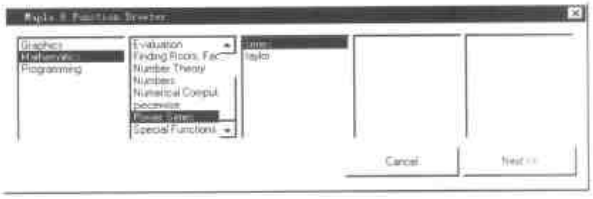
```
> Matrix(1..3,1..3,{(1,1)=1,(1,2)=4,(1,3)=7,(2,1
  )=2,(2,2)=5,(2,3)=8,(3,1)=3,(3,2)=6,(3,3)=9});
```

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

3. 在 Excel2000 中使用 Maple 6 函数

在 Excel 2000 中加载 Maple 6 宏就可以很方便的调用一些 Maple 6 函数，单击“View

Maple 6 Function Wizard”，可以看到我们能够调用的函数。下面我们通过三个例子来简单



级数展开，展开到 6 阶。

程，首先来看看第一种方法：

Wizard”弹出一个“Maple 6 Fuction Browse”对话框，选择“Mathematics”→“Power Series”→“series”，如图 7-6 所示：

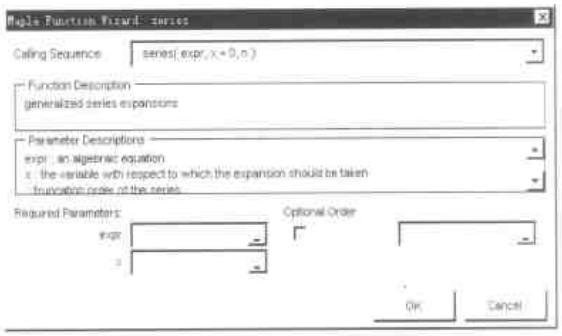
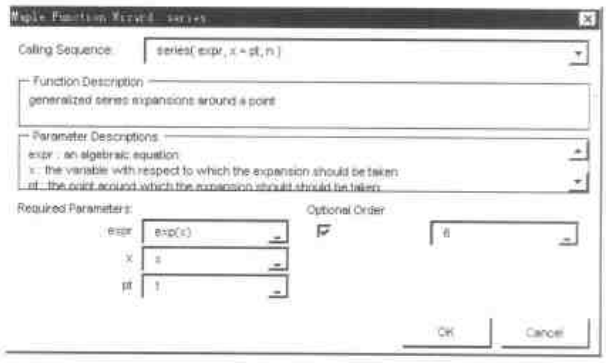


图 7-6　选择 Mapple 函数对话框

然后我们单击“Next”按钮，弹出“Maple Function Wizard: series”对话框，如图 7-7 所示：

图 7-7　调用 Mapple 函数对话框

我们看到在对话框的最上方给出了我们需要调用的函数的形式，由于我们需要在 x=1 处展开而不是 x=0 处，所以应该将上面的表达式选为 series(expr，x=pt，n)形式，并在 expr 文本框中输入待展开的表达式 exp(x)，在 x 文本框中输入展开的变量 x，在 pt 文本框中输入



先框，并在后面的文本框中输入展开的阶数 6，使对

图 7-8　修改后的 Maple 函数对话框

单击“OK”，我们就得到了表达式的展开形式，在 Excel2000 中看到的形式如下：

```
series(exp(1)+exp(1)*(x-1)+1/2*exp(1)*(x-
1)^2+1/6*exp(1)*(x-1)^3+1/24*exp(1)*(x-
1)^4+1/120*exp(1)*(x-1)^5+O((x-1)^6), x=-(-1), 6)
```

下面我们来看看第二种解题过程，这种过程实际上就是直接用函数调用而不是使用 Excel 2000 宏的导航功能，细心的读者可能发现在 Excel2000 的输入框中产生上面整个过程的其实就是一条语句：

```
=Maple("series( exp(x), x = 1    ,6 );")
```

这条语句的使用方法跟以前的用法是相同的，所不同的只是在最外层加入了一个 Maple 的函数声明，告诉 Excel 调用的是 Maple 的函数，利用这种方法调用 Maple 函数要快得多，但不如利用导航功能来得直观。

【例 7-5】求微分方程 dy/dx=cos(x*y)（y(0)=2）的数值解。

为了简便，我们使用直接进行语句输入的方式，在 A1 中直接输入如下公式：

```
=Maple("g:=dsolve({D(y)(x)=cos(x*y(x)),y(0)=2},y(x),type=`numeric`);")
```

对于里面函数的含义，想必读者都已经很熟悉，我们就不再介绍了，输入完毕后，直

的过程（见图 7-9）：

图 7-9　在 Excel 中使用 Mapple 得到的结果

为了得到某一点（比如 x=1）上的解我们只需在某个格内（比如 B2）内输入如下语句：

=Maple("g(1)")

即可，得到的结果如下：

```
[x = 1, y(x) = 2.29315655054552492]
```

在 Excel2000 中调用 Maple 中函数的过程和在 Maple 中使用相差不大，使用起来也很方便。下面我们再来看一个关于画图的例子：

【例 7-6】画出函数 f(x,y)=x^2+y^2 的三维图形。

在某一个输入格内输入如下语句：

=Maple("plot3d(x^2+y^2 ,x = -10..10,y=0..10);")

然后回车就可得到我们想要的三维图像（见图 7-10）：

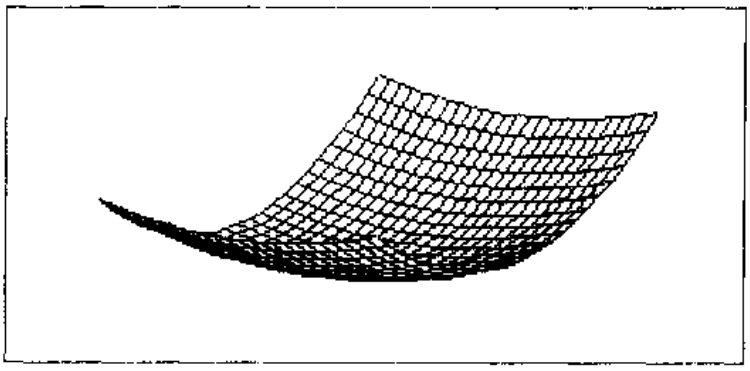


图 7-10　在 Excel 中用 Maple 函数得到的一个三维图形

Maple 和 Excel2000 新的接口为我们提供了灵活使用 Maple 和 Excel 的工具，从上面的例子可以看出，这是一个很有用的工具，学会使用这些特性往往能使我们事半功倍。

第 8 章 Maple 的绘图功能

经过前七章的学习，读者已经见到了很多由 Maple 生成的二、三维图形。将抽象的函数、公式表示成形象的图形，从而方便对其性质的理解；利用动画模拟现实过程，生动的反映实际过程，这都是绘图功能给用户带来的便利。Maple 之所以在欧美流行，一个重要原因就是它简单但功能强大的图形绘制函数，以及它内建的大量特殊函数，可以灵活、方便的实现用户所预期的功能。

本章将先对 Maple 的绘图功能做简要概述，再按照系统自带的不同程序库对 Maple 的绘图功能进行详细的分类介绍。

8.1 绘图功能概述

可以将 Maple 的绘图函数简单表示为：

函数名（"公式、数据"，"变量范围"，"函数参数"）

用户根据实际问题确定"公式、数据"的形式，再根据所希望生成图形的特点选择适当的函数来完成此功能，最后调整用到的"变量范围"、"函数参数"来获得好的显示效果。这个过程几乎存在于所有绘图函数的使用过程中。Maple 系统可以利用它的 GUI——图形用户界面帮助用户选择不同的函数参数，来获得不同的观察效果，但确定选择哪一个程序库的函数来显示图形，选择什么样的的参数才能获得最佳的显示效果，还是需要用户的判断。在这一节，我们将对 Maple 的 GUI 图形界面以及自带的各类程序库做简要介绍，希望能为用户今后的选择提供一定的参考。

8.1.1 二维图形工作环境设置

本书的第 2 章曾对 Maple 的工作簿略做介绍，第 3 章则简要介绍了同线性代数相关的图形界面，Maple 也有专门的公式输入面板。读者可以发现，针对不同问题，Maple 系统都会提供相应的图形用户界面——GUI 来方便它自带功能的实现。绘图功能也是如此，当用户选择的对象为图形对象时，不仅在菜单栏（MENU）中会出现新的选项，工具栏（TOOLBAR）中也会多出新的按钮。下面就让我们一项项的来介绍这些功能。

在普通的编辑状态，用户是看不见 Maple 中同绘图相关的菜单与工具栏的，如果用户在工作簿中利用函数生成了一个二维图形，并使用鼠标选中系统生成的图形（图形周围出现黑框，并且四个顶点及各边的中心都有黑色小方块），则 Maple 的菜单及工具条上就会出

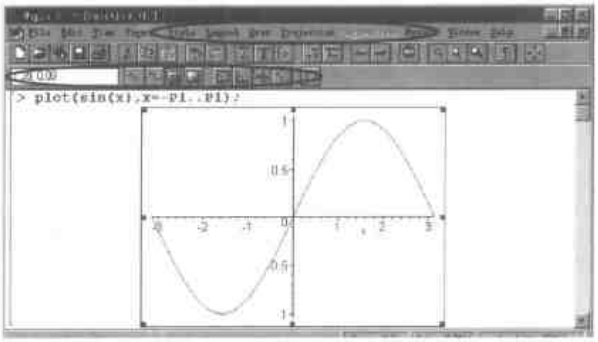


图 8-1　选中图形时四周将出现黑框和新的菜单与工具栏

这里生成的是二维图形，所以新增的菜单与工具栏均为同二维图形相关的选项。拖动黑框各边的中点可在水平或垂直方向上对图形进行拉伸或收缩，拖动四个角则可以同时控制图形在水平或垂直方向上的大小。当然此方法适用于所有被黑框包围的图形对象，包括将要介绍的三维图形和动画图形。

下面简要介绍一下新增菜单选项与对应的工具栏图标：

1．Style 菜单

图 8-2 所示为 Style 菜单，主要提供对曲线的线型、点、线宽等属性的设置选项。

图 8-2　Style 菜单

包括的选项有：

Line: 使用“线条”方式显示图形，对应工具栏上的按钮，效果如图 8-3（a）所示。

Point: 使用“点”的方式显示图形，对应工具栏上的按钮，效果如图 8-3（b）所示。

Patch: 使用带网格的多边形填充图形，对应按钮，效果如图 8-3（c）所示。

真充图形，对应按钮，效果如图 8-3（d）所示。

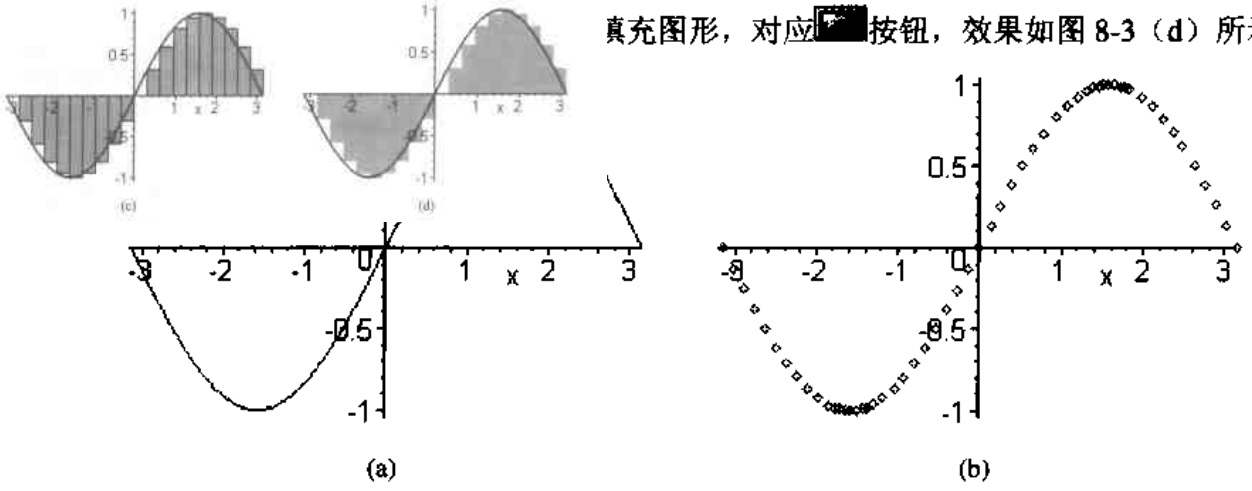


图 8-3 Style 菜单线型选择对应效果

（a）Line 效果；（b）Point 效果；（c）Patch 效果；（d）Patch o/w grid 效果

Symbol：设置在使用“点”方式显示图形时点的形状。缺省为“Diamond:菱形”（◇），如图 8-3（b）所示。其他选项还包括“Cross:十字”（+），“Point:点”（·），“Circle:圆”（○），“Box:正方形”（□）。多种显示方式的目的是为了在同一幅图中显示不同函数时区分方便。由于其效果同图 8-3（b）类似，此处就不再举例。

Symbol Size: 设置以“点”方式显示图形时点的大小。缺省值为 10。

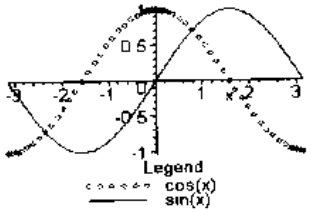
Line Style: 设置线形。缺省为“Solid:实线”，供选择的还有“Dot:点”，效果同选择“Symbol”中的“Point”选项类似；“Dash:虚线”（— —）；“DashDot:点划线”（·—）。

Line Width: 设置线宽。缺省为 Medium，供选择的还有“Thin:细线”与“Thick:粗线”。

gend 菜单

(a)所示是 Maple 6 新增设的菜单，包括“Show Legend:显示图例”和“Edit Legend:编辑图例”两个选项。作用是为图形对象加上图例说明，效果如图 8-4（b）所示。

(a)



(b)

图 8-4　Legend 菜单及其效果

（a）菜单；　（b）效果

3. Axes 菜单

如图 8-5 所示，它提供对图形坐标显示方式的选择。

图 8-5　Axes 菜单

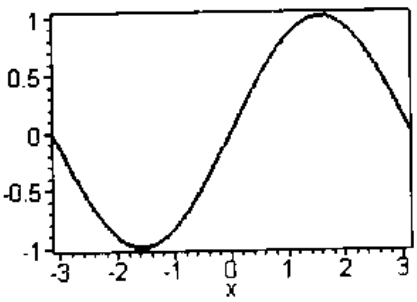
包括的选项有：

Boxed: 设置以矩形框的形式显示坐标，对应工具栏上的按钮，效果如图 8-6（a）所示。

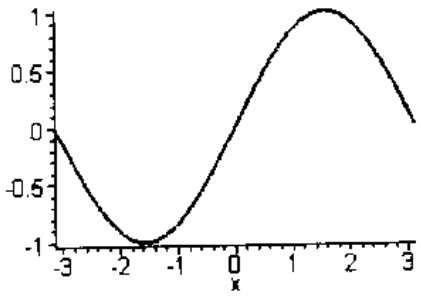
Framed: 设置以左下坐标轴形式显示坐标，对应按钮，效果如图 8-6（b）所示。

Normal：缺省（中心十字）坐标形式，对应按钮，效果如图 8-6（c）所示。

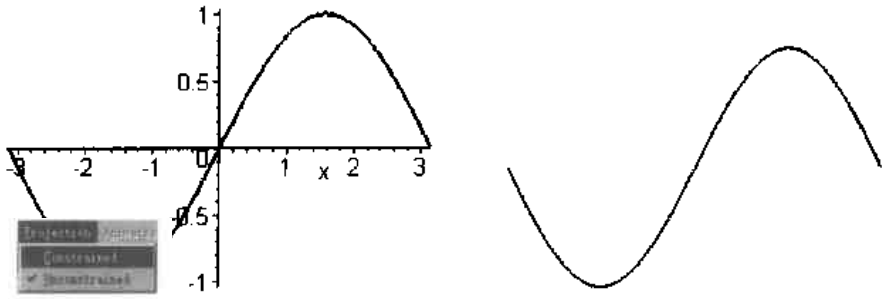
None：隐藏坐标形式，对应按钮，效果如图 8-6（d）所示。



(a)

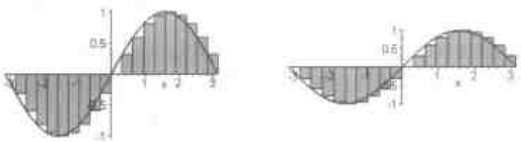


(b)



(c)　　(d)

图 8-6　Axes 菜单坐标选择对应效果



ned 效果；（c）Normal 效果；（d）None 效果

Projection 菜单图 8-7 所示

图 8-7　Projection 菜单

默认情况下，Maple 显示图像不按坐标轴等比例显示，而是选择 Maple 认为最“合理”的比例显示，即默认选中 Unconstrained 菜单。当选中 Constrained 时 Maple 将按等比例显示图像（对应工具栏中按下[ ]按钮的效果），如图 8-8 所示。

(a)　　(b)

图 8-8　对坐标限制带来的影响

（a）Unconstrained；（b）Constrained

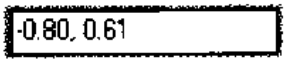
5. Export 菜单

如图 8-9 所示，是 Maple 6 的新增功能。其功能为将所选择的图形对象另存为其他图形

对　　、“GIF”、“BMP”等 Windows 中的常见格式。选中所希望　　保存对话框。具体过程请读者自行实践。

图 8-9　Export 菜单

工具条的最左方还有一项没有介绍的功能：



以当前坐标轴为基准显示坐标，注意只显示鼠标的点击位置。

8.1.2　三维图形工作环境设置

如果选择的对象是利用 Plot3D 等三维图形生成函数创建的对象，菜单栏除会出现同选中二维图形相同的几个选项外（但选项的内容多数都发生了变化），还新增了“color”选项；二维图形工具条会被三维图形工具条替换，如图 8-10 所示：

图 8-10　三维工具栏

1. Style 菜单

除了原有的“Patch”，“Patch o/w grid”，“Point”选项(分别对应图标[redacted]，[redacted]，[redacted])外，三维图形显示还增加了：

Patch and contour: 无网格多边形填充加等高线，对应按钮[redacted]，显示效果如图 8-11（a）所示。

Hiden line: 无填充，多边形显示，对应按钮[redacted]，显示效果如图 8-11（b）所示。

Counter: 等高线显示，对应按钮[redacted]，显示效果如图 8-11（c）所示。

Wireframe: 无填充，多边形透射显示，对应按钮[redacted]，显示效果如图 8-11（d）所示。

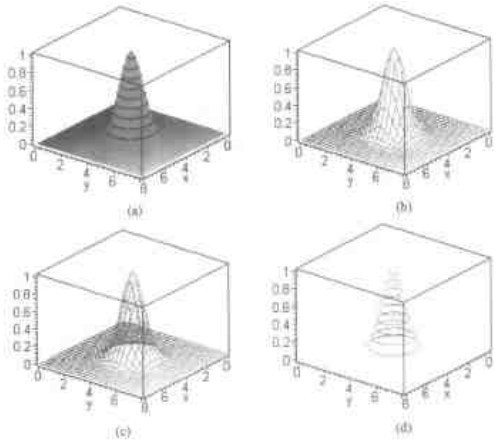


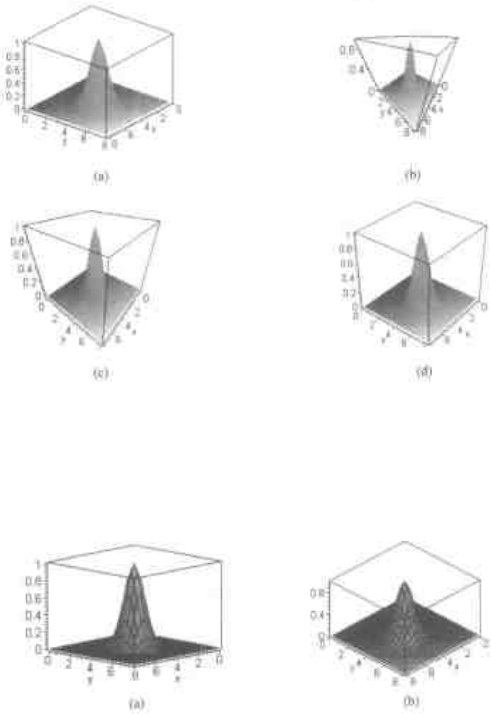
图 8-11　Style 菜单线型选择对应效果

（a）Patch and contour 效果；（b）Hiden line 效果；（c）Counter 效果；（d）Wireframe 效果

2．Color 菜单

如图 8-12 所示，主要提供对三维图形着色方案的选择，以及额外光源的使用方案。由于其效果主要由颜色的改变而提供，灰度显示无法如实表示其效果，这里就不举例说明。

图 8-12　Color 菜单



Constrained”和“Unconstrained”选项外，新增了对视角ive”，以及“Near Perspective”、“Medium Perspective”和选择不同视角位置而产生的效果。

图 8-13 变化观察位置带来的影响

（a）no perspective；（b）near perspective；（c）medium perspective；（d）far perspective

二维工具栏中显示鼠标点击位置的地方在三维工具栏中变成了显示三维物体的角度：

其中 θ 代表绕垂直轴旋转的角度；φ代表绕水平轴旋转的角度。

如图 8-14 所示：

图 8-14　改变图形旋转角度带来的变化

（a）θ=45°，φ=45°；　（b）θ=45°，φ=75°

在图形上拖动鼠标同样会产生旋转物体的效果，具体方法还须用户自己实践总结。

8.1.3 动画工作环境设定

利用 plots 程序库提供的 animate 或 animate3d 函数，用户可以轻松地建立一组动态画面。如果创建了一个二维图形的动画，当点击所创建的动画对象时，会出现如图 8-15 的工具栏：

图 8-15 动画工具栏


停止、播放动画按钮，用来选择动画播放状态。


以一帧一帧的方式显示动画。


选择动画的播放方向，向前或向后。会影响上两个按钮带来的播放效果。


调整播放速度按钮，最低速度为 1fps（帧/秒），最高速度为 75fps。每次点击都会小部分的改变播放速度。


播放方式选择，设置单向播放或循环播放。


切换工具栏上的按钮。由于动画本身是由多幅二维图形构成，所以 Maple 可以让用户单独处理每幅图形。通过点击上下箭头，用户可以在二维图形工具栏与动画工具栏中切换。

三维动画的工具栏同二维类似，只是用户可以利用  按钮切换到三维工具栏，利用上一小节介绍的工具处理动画的每一帧。

选择了动画对象后，菜单栏中的“animation”选项会被激活，它包含选项“Play”,“Next”，“Backward”，“Faster”，“Slower”以及“Continuous”。它们都在工具栏中有对应的按钮，这里就不再单独介绍。

需要提醒读者的是，Maple 在菜单、工具栏中提供的处理图形的方法，都集成在以鼠标右键点击图形对象后生成的菜单中，而且绝大部分都可以在生成图形时在函数参数中设定。但可以通过函数设定的参数中的绝大部分却没有集成在工具栏或菜单中，Maple 只是将最常用的参数集成起来，因此有很多效果是无法通过鼠标来实现的。所以读者如果想系统的了解 Maple 的图形处理功能，就还需要继续阅读后续章节。

8.1.4 绘图程序库

除了系统自带的函数 Plot，Plot3d 外，Maple 与画图有关的程序包有：plots、plottools、geometry、geom3d 以及 starplots。每个程序库内的函数都有自己专门的处理对象。它们的特点简要总结如下：

1. plots 程序库

是最常加载的程序库。其中 animate 函数可以生成动画。如果需要观察某一函数随时间或位置变化的效果，使用它是十分方便的；contourplot 可以画等高线，使得对一些不容易选择观察角度的三维物体可以比较不同部位的高低差别；display 函数可以同时将几组图形显示在一个图形对象上，非常适用于比较函数间的差别（本书的前几章已有多个例子使用了此函数）；implicitplot 函数可以对方程作图。许多情况下，我们并不能得到显式的函数间的关系，这时可以通过此函数对方程作图而观察出函数特性。这些函数都将在后面详细介绍。plot 程序库中的函数列表如表 8-1 所示：

表 8-1　plot 程序库函数列表

animate	animate3d	animatecurve	changecoords	complexplot
complexplot3d	conformal	contourplot	contourplot3d	coordpolt
coordplot3d	cylinderplot	densityplot	display	display3d
fieldplot	fieldplot3d	gradplot	gradplot3d	implicitplot
implicitplot3d	inequal	listcontplot	listcontplot3d	listdensityplot
listplot	listplot3d	loglogplot	loglogplot	matrixplot
odeplot	pareto	**pointplot**	pointplot3d	polarplot
polygonplot	polygonplot3d	polyhedraplot	replot	rootlocus
semilogplot	setoptions	setoptions3d	spacecurvee	sparsematrixplot
sphereplot	surfdata	testplot	testplot3d	tubeplot

如果读者仔细观察这里所列出的 50 个函数，会发现其中有 14 对函数仅差别在结尾是否有“3d”上。显而易见，这 14 对函数是处理同一对象在不同维度上的问题的函数。因此掌握了其中一个在某一维度上的用法，另一个函数也可以依法处理。本书会对其中的少部分函数做简要介绍，如果读者还有其他方面的需要，可以通过输入“? plots”的方式获得联机帮助。

2. Plottools

程序包提供了许多生成基本几何体的函数，如画弧函数 arc，画圆函数 circle，画多边形函数 polygon，旋转函数 rotate，比例放大函数 scale，画矩形函数 rectangle，画球体的 sphere 函数，画半球函数 hemisphere 以及平移函数 transform、变换函数 transform 和反射函数 reflect 等等。plottools 程序库中的函数列表如表 8-2 所示：

表 8-2 plottools 程序库函数列表

arc	arrow	circle	cone	cuboid
curve	cutin	cutout	cylinder	disk
dodecahedron	ellipse	ellipticArc	hemisphere	hexadedron
homothety	hyperbola	icosahedron	line	octahedron
pieslice	point	polygon	project	rectangle
reflect	rotate	scale	semitorus	sphere
stellate	tetrahedron	torus	transform	vrml

3. geometry 及 geom3d 程序库

geometry 程序包是另外一个画图函数程序包。该程序包主要提供了为解决二维欧基里德空间内解析几何问题的各种专用函数，如判断点共线（IsOnLine）、判断平行（AreParallel）、判断是否垂直（ArePerpendicular），以及是否相似（AreSimilar）、相切（AreTangent），求交点等等。由于此程序库内包含的函数十分庞大，这里仅列出其中的一部分（如表 8-3 所示），读者可以通过“with(geometry);”命令显示其中的全部函数。

表 8-3 geometry 程序库函数列表

AreCollinear	AreConjugate	AreConcurrent	AreConcyclic	AreOrthogonal
CrossProduct	ExteriorAngle	SpiralRotation	GlideReflection	IsOnCircle
MakeSquare	FindAngle	SimsonLine	MajorAxis	MinorAxis
Appolonius	OnSegment	DefinedAs	EulerCircle	NagelPoint
circumcircle	detail,	ellipse	fexcircle	foci
form	hyperbola	inradius	incircle	inversion
intersection	line	median	midpoint	stretch
sides	square	segment	triangle	vertex
powerpc	slope	radius	center	circle
reflection	rotation	altitude	diameter	projection

对应的，geom3d 程序库的针对目标是三维欧基里德空间内的问题，它也包含有同 geometry 程序库功能类似的函数。这里就不再一一列出。

4. statplots 子程序库

此函数库是 stats 程序库的子库，通过“with(stats)”或“statplots()”命令来调用。它的功能主要集中在对统计学数据的图形显示上。包括函数有“boxplot：柱状图”、“histogram：统计图”、“scatterplot：散点图”以及改变坐标的 xscale、yscale、zscale、xshift、yshift、zshift、xyexchange、yzexchange、yzexchange 等。

8.2　二维图形绘制—PLOT 及相关函数的应用

二维图形是最简单明了、数学中最早接触的图形，同时也是最容易实现的图形。Maple 中可以绘制的二维图形种类十分广泛，涉及的函数也很多。本书当然不可能将它们完全剖析并展示给读者。为了能让读者系统的了解 Maple 的二维作图系统，我们将先介绍有关的参数设定，再介绍几种基本函数的画法，希望能起到抛砖引玉的效果。

8.2.1　二维绘图参数设置

二维图形绘制的最基本函数是 plot。它是 Maple 系统自带函数，不需要加载任何程序包就可以直接调用。plot 函数的功能十分强大，相应的，它可以附加的参数也异常繁多。利用“?plot”命令可以系统地看到同它相关的参数。其实上一节所介绍的 Maple 菜单及工具栏中针对图形对象的功能，也都可以通过改变 plot 函数的参数来实现。实际上我们从 ?plot 命令中看到的图形设置参数，并不只是针对 plot 一个函数而设计的。几乎 Maple 中所有的函数中都可以附带这些参数，从而改变它生成的图形对象的显示效果。因此彻底掌握同图形对象相关的参数对我们深入了解 Maple 的图形系统有很大的帮助。因此在这里我们将相关参数系统的列出来，同时也方便读者今后的查询。

1．adaptive

系统默认为 TRUE。如果设为 false 将不能使用自适应作图功能（adaptive plotting）。

2．axes

设置坐标轴的类型，可以是 FRAME、BOXED、NORMAL 和 NONE 之一，默认为 NORMAL。

3．axesfont=l

设置坐标轴标注文字的字体，类似于 font 选项。

4．color=n

让用户自己指定曲线的颜色，也可以用 colour=n 来设置。“n”为用户希望的颜色名称，可供选择的颜色如表 8-4 所示：

表 8-4　系统自带颜色一览

aquamarine	碧绿色	black	黑色
blue	天蓝色	navy	海蓝色
coral	珊瑚色	cyan	青色（蓝绿色）

续表

brown	棕色	gold	金色
green	绿色	gray(grey)	灰色
khaki	黄褐色	magenta	红紫色（品红）
maroon	栗色	orange	橘黄色
pink	粉红色	plum	深紫色
red	红色	sienna	土黄色，赭色
tan	棕褐色，茶色	turquoise	青绿色
violet	紫罗兰色	wheat	淡黄色
white	白色	yellow	黄色

如果用户对这些颜色还不满意，可以使用 RGB，或 HUE 配色方案。方法如下所示：

```
> macro(skyblue = COLOR(RGB, 0.1960, 0.6000, 0.8000));
  plot(tan(x), x = -1..1, color=skyblue);
```

如果用户想用不同的颜色标记一组曲线的化，可以使用“color=[n1,n2]”的形式表示，“n1”，“n2”按输入顺序对应函数中的曲线名称。

5. coords=<name>

指定参数作图时所用的坐标系，坐标系名称由 name 指定。默认的坐标系为 Cartesian(笛卡儿)坐标系，在二维曲线范围内，Maple 还提供如表 8-5 所示的其他坐标系供用户选择。如果想获得 Maple 所支持的所有坐标系，可以使用“?coords”命令来查询。

表 8-5 Maple 中可供选择的二维坐标系名称

bipolar	双极坐标	cardioid	心形坐标
cassinian	卡斯尼亚	elliptic	椭圆
hyperbolic	双曲线	invcassinian	反卡斯尼亚
invelliptic	反椭圆	logarithmic	对数坐标
logcosh	对数余割	maxwell	麦克斯韦
parabolic	抛物线	polar	极坐标
rose	玫瑰	tangent	切线

6. discont=s

当 s 设为 true 时 plot 将首先调用 discont 函数确认输入是否连续，然后将横轴自动分为几个区间来表示原来并不连续的曲线。

7. font=list

该选项是图形和图像中文本对象的字体，list 是列表[family, style, size]。其中 family 是以下之一：

TIMES（对应 Windows 标准字库中的 Times New Roman 字体）；

COURIER（对应 Windows 标准字库中的 Courier 字体）；

HELVETICA 或 SYMBOL（对应 Windows 的 Symbol 字体）。

当 family 设置为 TIMES 时，style 的参数可以是 ROMAN, BOLD, ITALIC 或 BOLDITALIC 之一。当 family 设置为 HELVETICA 或 COURIER 时，style 可以省略或是 BOLD, OBLIQUE. BOLDOBLIQUE 之一。family 是 SYMBOL 时，将不接受 style 选项。size 对应字体大小。

8. labels=[x,y]

该选项指定坐标轴的标注，x 和 y 的值必须是字符串。默认值是 plot 函数中的变量名。

9. labelfont=list

指定坐标轴的标注文字的字体，同 font 设置相同。

10. linestyle=n

画图中选择的线型，各数值对应的线型为：

n=1：	实线；	n=2：	曲线由点构成；
n=3：	虚线；	n=4：	点划线

11. numpoints=n

画图时最小点数，默认值 n = 50。如果选择 plot 自适应画图，有些时候在函数的拐点处 plot 函数无法平滑的显示，这时就需要设定 n 来更好的显示曲线。

12. resolution=n

设定水平方向显示分辨率。默认值为 n = 200。自适应绘图中止时 n 值才起作用。对不光滑的函数曲线，使采样点将增多。

13. sample

为=以参数列表的形式提供函数的初始值。

14. scaling

控制绘图的两坐标轴刻度比例,CONSTRAINED 或 UNCONSTRAINED，该参数的默认

值为 UNCONSTRAINED，即 Maple 自动调整 X，Y 坐标的比例，以达到最佳的显示效果，会影响真实的横、纵坐标比例。

15. style=s

指定绘图的线型。s 可以是 LINE, POINT, PATCH 或 PATCHNOGRID。默认值是 LINE。POINT 即由散点组成曲线，PATCH 则以片状颜色填充图形，PATCHNOGRID 同 PATCH，不过没有网格。

16. symbol=s

指定点的形状，s 可以是 BOX(□), CROSS(+), CIRCLE(○), POINT(·)或 DIAMOND(◇)。

17. thickness=n

设定线型的宽窄。n 可以是 0, 1, 2 或 3。默认值为 0，随数字增加，线型变粗。

18. tickmarks=[m,n]

m，n 分别 X 轴、Y 轴的标记点数。如果仅希望改变其中的某一个轴，则需要使用参数 xtickmarks 或 ytickmarks。

19. title=t

图形对象的标题，t 必须为字符串。默认为无标题。

20. titlefont=list

设定标题的字体，参数与 font 相同。

21. view=[xmin..xmax, ymin..ymax]

指定图形对象所显示的曲线最大和最小坐标。默认是显示整个曲线。

22. xtickmarks=n

ytickmarks=n

指定横轴或纵轴上合理的标记数目，不少于 n。n 必须是一个正整数或一个列表。如果 n 是一个列表，列表的元素值将标记坐标轴。

读者可能会观察出其中很多参数都对应着菜单或工具栏中的选项，改变这些参数的效果这里就不再逐一列出。但需要提醒的是如果改变了菜单或工具栏中有关图形对象的参数设定，则会变成缺省设定，从而影响到其后生成的所有图形，而在函数中指定的参数则只会对此函数生成的图形对象起作用，而不会改变缺省设定。

利用 plots 程序库中的“setoptions”，或“setoptions3d”则可以通过命令改变系统的缺省设定。如图 8-16 所示，使用 setoptions 后，工具栏上相应的按钮也会被按下：

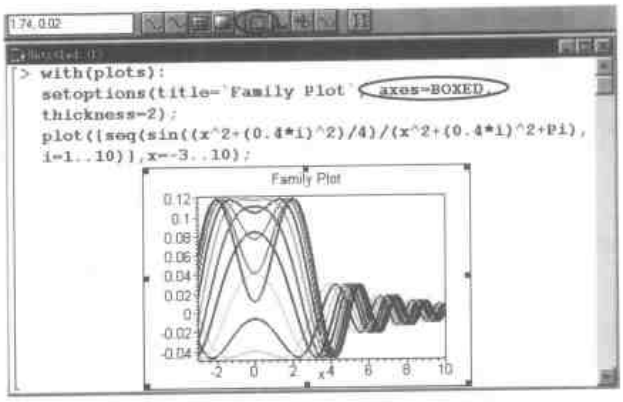


图 8-16 使用 setoptions 后的效果。

8.2.2 简单函数的二维图形绘制

这里所说的简单函数是值可以表示成 f(x)=expression(x)的形式的函数。即某种单变量的表达式。在前几章，我们已经多次使用过 plot 函数来生成这种函数对应的图形。plot 函数的完全表达形式为：

plot(f,h,v.....)

f 对应用户希望绘制的目标函数，h 为用户指定的横、纵轴坐标范围，以“变量名=a..b”的形式表示。v 为指定的图形显示参数，可以是上一节中介绍过的任何一个参数，或它们的组合。参数间以“,”分割。由于读者可以在本书的前几章找到大量的使用 plot 函数的例子，本节就不再列举其简单的应用，而将重点放在如何利用“过程”来定义自己的函数而使 plot 完成一些特殊的功能。

让我们先重温一个在 3.5 节使用过，但未做详细解释的自定义函数 loopplot：

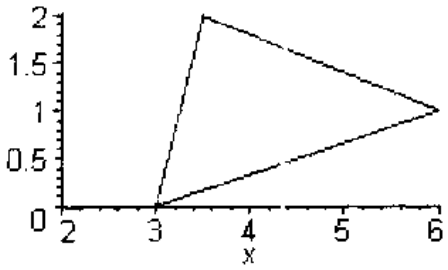
```
> with(plots):
Warning, the name changecords has been redefined
```

```
> loopplot:=proc(L)
>     plot([op[L],L[1]],args[2..nargs]);
> end:
```

```
> L1:=[[3,0],[6,1],[7/2,2]];
```

$$L := \left[[3,0],[6,1],\left[\frac{7}{2},2\right]\right]$$

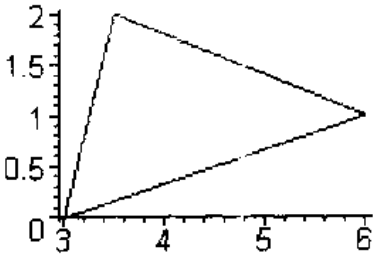
```
> loopplot(L1,x=2..6,scaling=constrained);
```



顾名思义，loopplot 函数的作用就是生成一组循环的数据点，即将第一个数据点插到数据点列表的最后一位，再利用 plot 对散点绘图的功能形成一个封闭图形。而为了生成一个完善的过程，可以模仿使用 plot 函数的参数，我们在定义过程中使用了“args[2..nargs]”，这样可以将用户在 loopplot 函数中声明的参数传给 plot 函数（nargs 表示输入参数的个数），从而获得所有只在使用 plot 函数时才会有的便利。

实际上，Maple 提供的函数 polygonplot 可以直接完成这项任务，如继续使用上例中的数据：

```
> polygonplot(L1);
```



然而由于 polygonplot 以及 polygonplot3d 的原理同我们自定义的 loopplot 过程类似，使得如果我们想生成一个四棱柱或是类似的三维物体，将无法简单的通过输入四棱柱的 8 个顶点坐标来获得正确的图形。希望读者注意。

再介绍一个分段函数的例子。比如绘制如下的一个函数：

$$\begin{cases} x2+11 & x>3 \\ x3-7 & 0<=x<3 \\ x & -3<=x<0 \\ 5 & x<-3 \end{cases}$$

我们利用过程来建立此分段函数：

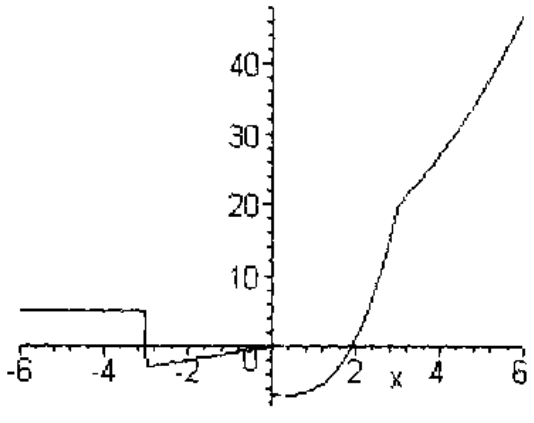
```
> f:='f':
> f:=proc(x)
>     if x>3 then x^2+11
>     elif x<=3 and x>0 then x^3-7
>     elif x<0 and x>=-3 then x
>     elif x<-e then 5
>     fi
> end:
```

以后，可以直接使用 f（x）来调用此分段函数。对分段函数的画图同普通函数一样，但我们却不能直接在 plot 函数中使用 f（x），否则就会出现如下提示：

```
> polt(f(x),x=-6..6);
Error, (in f) cannot evaluate boolean: -x < -3
```

由于 plot 函数是边对函数赋值边提取坐标绘制，无法在执行完过程中的全部判断语句后为函数赋值，因此系统会因无法确定纵坐标的数值而报错。这时，只需用单引号将 f（x）括起来，Maple 就会将其当作一个整体，正确执行画图命令。如：

```
> plot('f(x)',x=-6..6);
```



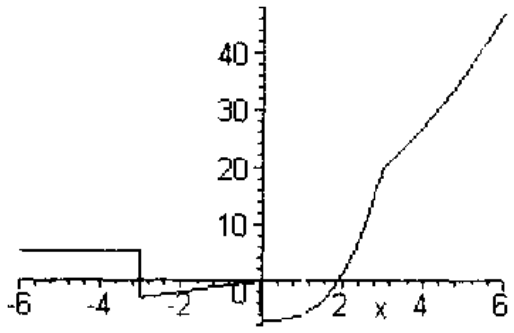
当然，在 Maple 中，已经为用户设计了函数“piecewise”来定义分段函数。piecewise

函数的完全表达形式为：

piecewise(cond_1,f_1, cond_2,f_2, ..., cond_n,f_n, f_otherwise)

cond_1 参数一般为一个不等式或用“and”“or”“not”连接的不等式组，表示分段函数的第一个区间，f_1 参数表示在此区间内函数的取值，一般为一多项式，以后的参数以此类推，最后的 f_otherwise 表示在定义区间外其他位置函数的取值，也一般用多项式表示。利用 piecewise 函数不仅可以简单的定义分段函数，还会避免使用过程定义分段函数后会出现的画图错误。如：

```
> f:=x->piecewise(x>=3,x^2+11,x>=0 and x<3,
  x^3-7, x<0 and x>=-3, x, x<-3, 5):
> plot(f(x),x=-6..6);
```



8.2.3 多元函数方程作图

有时候我们所获得的方程并不是一个简单的单变量函数，而是一个多变量的方程，这时候 plot 函数处理这些问题就有些力不从心了。先看一个简单的解析几何中的圆方程：

【例 8-1】作图：

$$(x-4)^2+(y+4)^2=25$$

先让我们看如何利用 plot 函数完成这项任务：

```
> eq:=(x-4)^2+(y+4)^2=25;
```

$$eq := (x-4)^2 + (y+4)^2 = 25$$

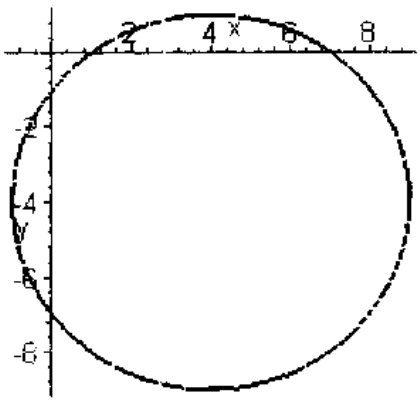
```
> plot(eq,x=-1..9, y=-9..1);
plotting error, empty plot
```

由于直接将方程代入 plot 函数无法获得正确结果，因此我们要将其变形后再绘制：

```
> sols:=[solve(eq,y)];
```

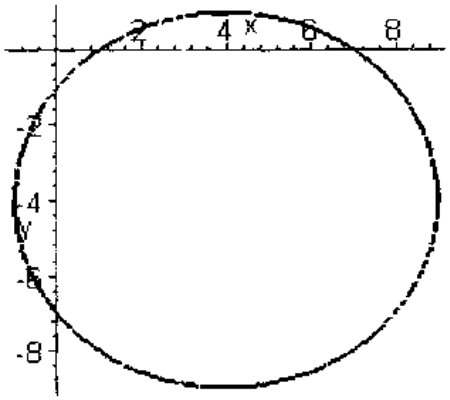
$$sols := [-4+\sqrt{9-x^2+8x}, -4-\sqrt{9-x^2+8x}]$$

```
> p1:=plot(sols[1],x=-1..9):
> p2:=plot(sols[2],x=-1..9):
> display(p1,p2,scaling=constrained,thickness=2);
```

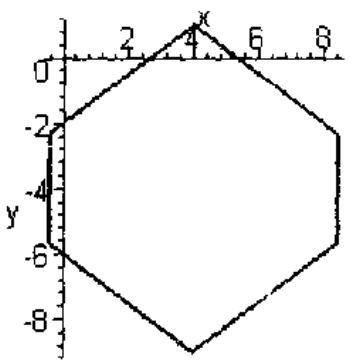


然而这只是简单的情况，很多时候二元方程是无法如此分离变量的。这时候，就需要使用 plots 程序库的附带函数 implicitplot 了。implicitplot 一般用来处理二元函数的方程或表达式问题，它会以一定间隔取点来近似原始函数的形状，如同样利用它绘制上例的圆形：

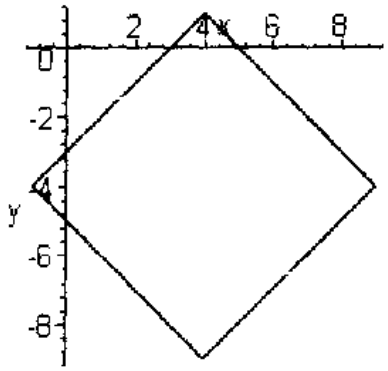
```
> implicitplot(eq,x=-1..9,y=-9..1,scaling=
  constrained,thickness=2);
```



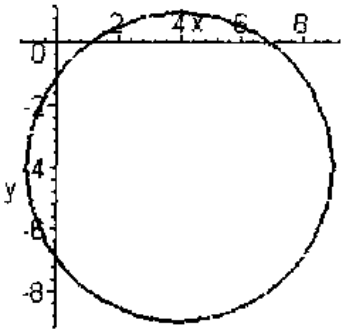
如果读者仔细观察这时生成的圆，并将它同 plot 函数生成的圆对比，会发现 implicitplot 生成的圆有很多间隙，这是由于它默认的取点密度不够所致，这时就需要设定参数 grid=[m，n]。m，n 分别对应函数在 x，y 轴所取的采样点。如果选择不同的采样点，有时会获得意想不到的结果。如图 8-17 所示：



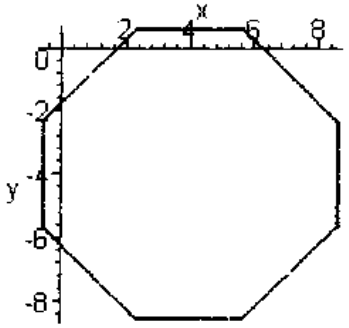
[m,n]=[3,3]



[m,n]=[3,4]



[m,n]=[4,4]



[m,n]=[9,9]

图 8-17　不同的采样点获得的不同效果

implicitplot 函数默认取样点数是[25　25]，但我们看出绘制圆形并不是采样点越多效果越好。所以如果用户不满意此函数生成的图形，可以尝试改变取样点来获取满意的效果。

然而，如同其函数名，“implicitplot->盲从的画图”一样，由于 implictiplot 只利用采样点来绘制图形，而不考虑函数间的断点或拐点，利用它绘制的图形可能会导致对用户的错误引导。 如用它与 plot 同样绘制 y=tan（x）的曲线：

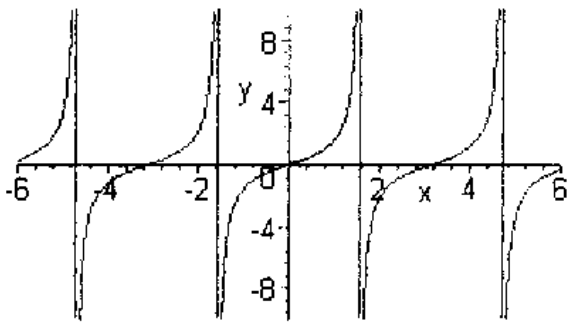
```
> eq1:=tan(x)-y=0;
```

$$eq1 := \tan(x) - y = 0$$

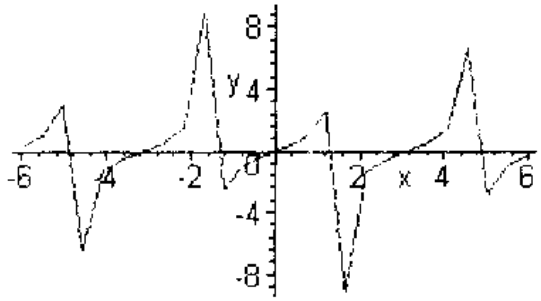
```
> eq1:=tan(x);
```

$$eq2 := \tan(x)$$

```
> plot(eq2,x=-6..6,y=-10..10);
```

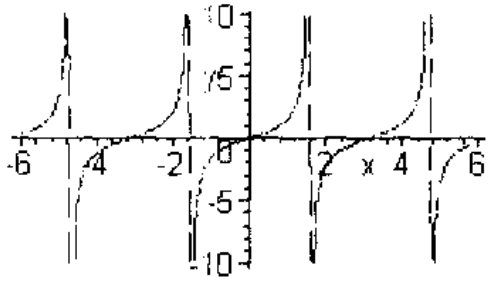


```
> implicitplot(eq1,x=-6..6,y=-10..10);
```



如果我们设置 implicitplot 的参数“grid=[100，100]”，在等待很长一段时间后，会出现如下结果：

```
> implicitplot(eq1,x=-6..6,y=-10..10,grid=
  [100, 100]);
```



虽然我们知道如果再增加采样点数，会获得更清晰的图形，但毕竟此类函数并不适合利用 implicitplot 来绘制。所以用户需要选择适当的函数来处理不同的问题。

8.2.4　不等式的绘制

在本书的 3.5.3 节，我们在介绍线性代数在运筹学中的应用时曾经使用过 Maple 中绘制不等式的函数“inequal”，但没有具体地介绍它的使用方法。不等式也是初等代数中的一个重要组成部分，这一节将详细地把 inequal 函数介绍给读者。

inequeal 函数的调用格式为：

inequal(ineqs, xspec, yspec, options)

其中参数 ineqs 表示不等式或不等式组，xspec 表示显示图像的 x 轴区间范围，yspec 表示显示图像的 y 轴范围，option 为如下的表达式：

optionname = (图形选项)

inequal 函数将整个平面点集分成四个区域：(1)满足所有不等式的合理区域；(2)至少不

满足一个不等式的区域；（3）不等式的边界区域，但不满足不等式；（4）满足不等式的边界区域。分别对应着 optionname 中的“optionsfeasible”，“optionsexcluded”，“optionsopen”和“optionsclosed”。用户可以通过这个选项分别对这四个区域进行参数设定。等号后的选项可以是 8.2.1 节所列出的所有图形参数。请看以下的一个例子：

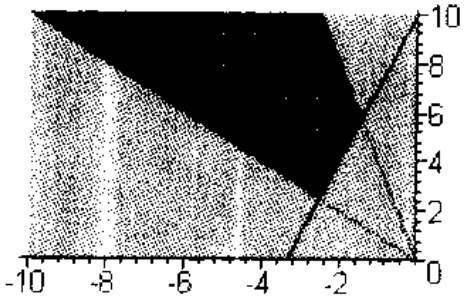
【例 8-2】 作图表示满足如下不等式组的解空间：

$$\begin{cases} x+y>0 \\ 3x-y\le 10 \\ 4x+y<9 \end{cases}$$

```
> with(plots):
> ineqs:={x+y>0,3*x-y<=-10,4*x+y<9};
```

$$ineqs := \{3x - y \le -10, 4x + y < 9, 0 < x + y\}$$

```
> inequal(ineqs,x=-10..0,y=0..10,
  optionsexcluded=(color=gray,thickness=2),
  optionsfeasible=(color=black,thickness=1));
```



8.2.5 极坐标及其他坐标系下的图形绘制

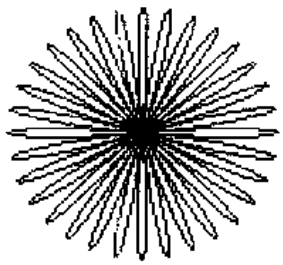
在此之前，所有介绍过的 plot 函数都是在直角坐标（或称笛卡儿坐标）下绘制图形的。然而有很多复杂曲线却可以在其他坐标系下以非常简单的形式表示出来。这一节，将主要介绍如何利用 plot 函数绘制基于其他坐标的函数。

先介绍另一种较常用的坐标系：极坐标。这时的 plot 函数的调用形式为：

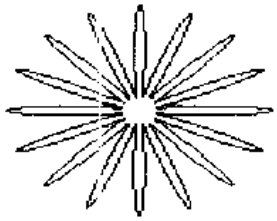
```
plot(f(theta),theta=a..b,coords=polar);
```

其中，参数 theta 是极角，f(theta)构成极半径，它是极角 theta 的函数。第二个参数 theta 给出作图范围。第三个参数 coords=polar 是选项，表示是在极坐标下作图。来看几个例子：

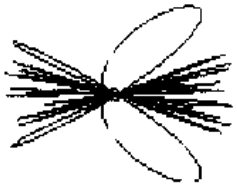
```
> plot(cos(16*theta), theta=-Pi..Pi, coords=polar,
  scaling=constrained);
```



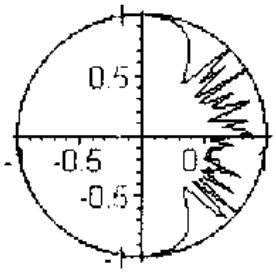
```
> plot(exp(cos(16*theta), theta=-Pi..Pi, coords=p
  olar,scaling=constrained);
```



```
> plot(sin(theat^(-2)), theta=-5*Pi..5*Pi, coords=符号
  polar);
```



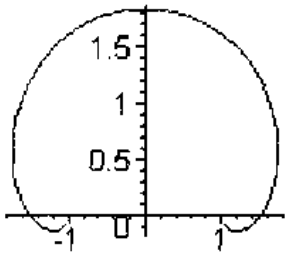
```
> plot(cos(sin(theat^(-5))), theta=-10*Pi..10*Pi,
  coords=polar);
```



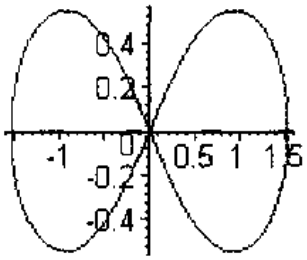
利用极坐标可以很容易地产生一些在普通直角坐标中很复杂的曲线，有时仅仅是改变很小的一个参数就会获得迥异的图形。读者有兴趣的话可以自己设计一些函数来观察效果。

在表 8-5 中，我们曾列出了 Maple 中所有固化的坐标系。它们都可以在 plot 函数中表示出来。例如我们生成一个双极坐标曲线，一个椭圆坐标曲线：

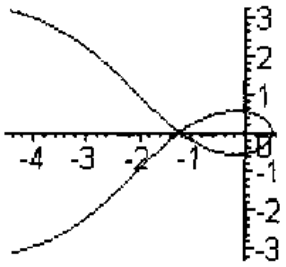
```
> plot(cos-Pi..Pi, coords=bipolar);
```



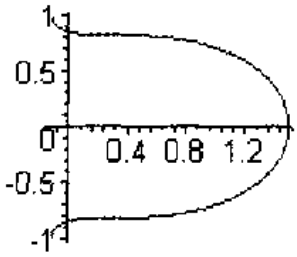

```
> plot(cos-Pi..Pi, coords=elliptic);
```




```
> plot(cos-Pi..Pi, coords=parabolic);
```




```
> plot(cos-Pi..Pi, coords=maxwell);
```



双极坐标的定义是：

$$x = \sinh(v)/(\cosh(v)-\cos(u))$$

$$y = \sin(u)/(\cosh(v)-\cos(u))$$

椭圆坐标的定义为：

$$x = \cosh(u)*\cos(v)$$

$$y = \sinh(u)*\sin(v)$$

抛物线坐标的定义为：

```
x = (u^2-v^2)/2
y = u*v
```

麦克斯韦坐标的定义为：

```
x = a/Pi*(u+1+exp(u)*cos(v))
y = a/Pi*(v+exp(u)*sin(v))
```

可能一般读者对这两个坐标系都不太了解，Maple 提供了一个函数 coordplot 可以帮助用户形象的理解各种坐标系的特点。利用“coordplot（坐标系名）”的形式，Maple 可以显示出它所支持的所有坐标系的横、纵坐标轴的关系。在此，我们挑选了几种有特点的坐标系，如图 8-18 所示：

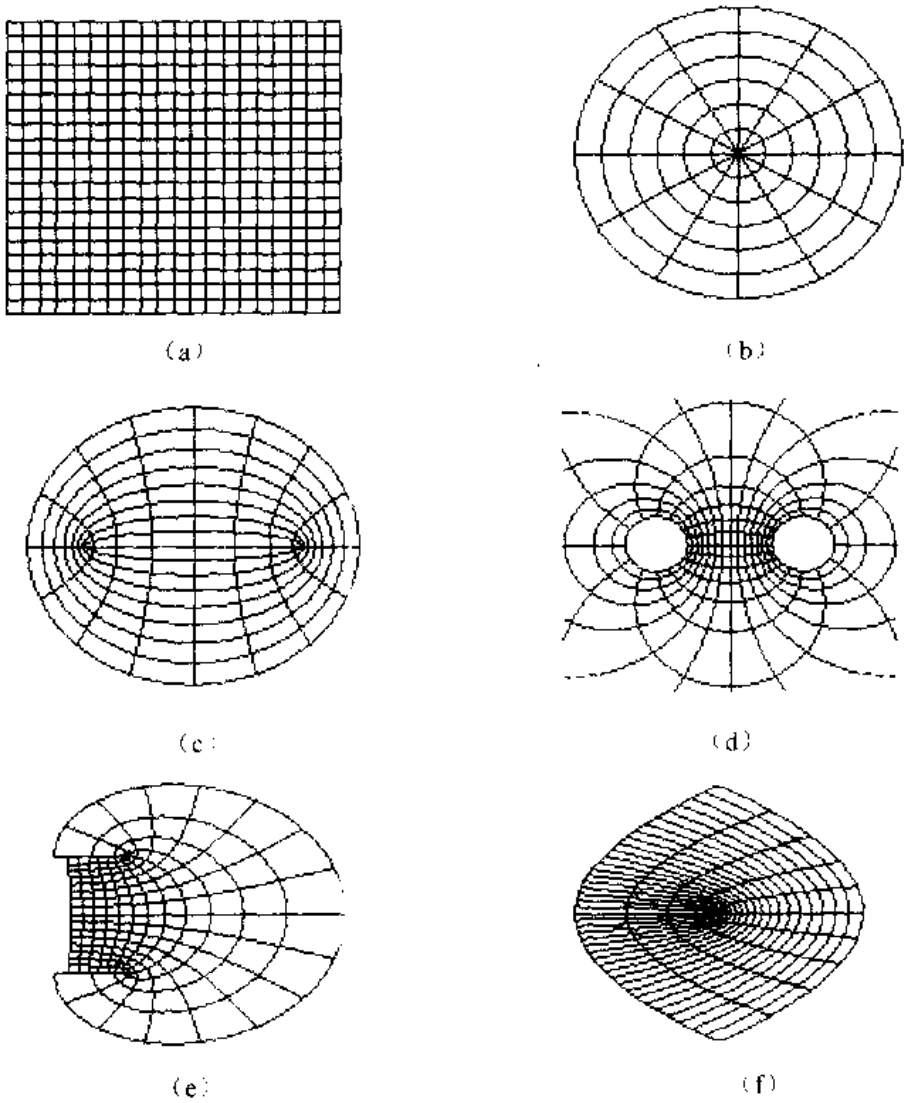


图 8-18　几种典型坐标系的结构

（a）笛卡儿坐标；（b）极坐标；（c）双极坐标；

（d）椭圆坐标；（e）抛物线坐标；（f）麦克斯韦坐标

这里所画出的，以及表 8-5 所列出的，仅是一些二维坐标系的结构，Maple 还内建了很多三维坐标系，由于显示效果问题，这里就不再举例了。有兴趣的读者可以用“? coords”来了解它们的详细内容。同时，在联机帮助中还有这些坐标系的定义公式，方便用户在使用时查询。

8.2.6 参数曲线的绘制

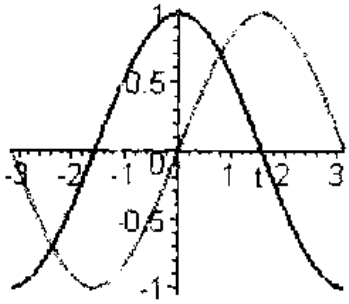
在很多情况下函数是以参数方程的形式表示出来的。plot 函数同样可以对这类函数进行绘制。这时的 plot 函数的调用形式为：

plot([x(t),y(t),t=a..b], opts) 或

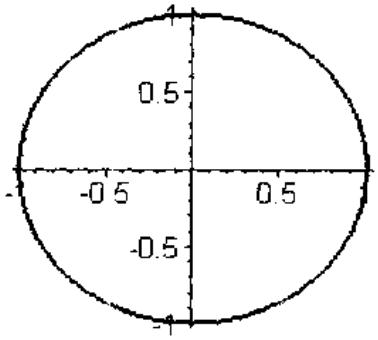
plot([r(t),theta(t),t=a..b],coords=polar,opts)

第一种表达式为直角坐标下参数方程的形式，后一种为极坐标下参数方程的形式。函数最后的 opts 参数同普通的 plot 函数参数一样。注意此时参数方程被“[]”括住，而不同于平常利用“{}”括住的方程组的表示形式。请看如下的例子：

```
> x(t):=sin(t):y(t):=cos(t):
> plot({x(t), y(t)},t=-Pi..Pi, thickness=2);
```




```
> plot([x(t), y(t)],t=-Pi..Pi], thickness=2);
```



利用参数方程画出的参数曲线有时是十分有意思的。这些曲线甚至可以模拟出实际中的许多事物，如枫叶、云彩、手掌、小动物等。有兴趣的读者可以尝试用以下的函数绘图：

```
s := t->100/(100+(t-Pi/2)^8): r := t -> s(t)*(2-sin(7*t)-cos(30*t)/2):
plot([r(t),t,t=-Pi/2..3/2*Pi],coords=polar);
```

会获得很漂亮的图形☺。

8.3　三维图形绘制—— PLOT3D 及相关函数应用

在介绍完用 Maple 绘制基本二维图形后，我们开始利用它绘制三维图形。在绘制原理上，三维图形与二维图形没有本质的差别。但由于涉及到如何在二维的显示设备上表示的问题，在图形学中，三维图形增加了投影方式的选择。然而在 Maple 系统中，用户却不需要考虑投影方面的问题，因为几乎所有的三维图形都是以斜投影的方式表示的。用户可以改变图形本的方向以及视点的远近。由于三维图形绘制时可以选择的参数同二维图形基本相同，在对 plot3d 函数做基本介绍后，本节的重点将转移到一些 plots 程序库附带的其他三维图形绘制函数上，比如等高线绘制、密度图绘制等等，希望能为读者使用这些函数时提供一定的参考。

8.3.1　基本三维函数图形的生成

由于现实中人们所熟知的三维图形一般并非简单函数可以生成的，即使可以表示，也涉及到很多参数的设定，所以在实际三维绘图中，plot3d 函数并不像 plot 函数那样常用。一般只是用它对三维函数曲线进行绘制。它的使用方法同 plot 函数几乎完全一样，比如利用简化输入绘制一个三维曲面：

```
> restart;
> f:=(x,y)->cos(x*y):
> plot3d(f,-3..3,-2..2,orientation=[160,50]);
```

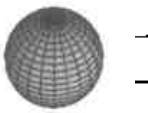
之所以称这种输入方法为简化输入，因为我们在输入坐标范围的时候，并不是按照“x=a..b,y=c..d”这种规则形式书写的，而是直接写成“a..b,c..d”的形式。注意这样的输入方法只对函数变量定义为“x，y”的形式有效，系统会自动将坐标 x 轴的范围定义为“a..b”，而将 y 轴定义为“c..d”。同样，在 plot 函数中也可以对变量被定义为 x 的函数使用这种简化输入法。相对于二维的 plot 函数，plot3d 在坐标轴的形式，曲线、曲面的式样等一些方

面有新的参数。在上例中使用的 orientation 参数就是其中之一，它可以确定用户的观察角度。“[]”中的数值分别对应三维工具栏中 θ 角与 φ 角。由于这些参数同 plot 函数中的参数大同小异，而且我们在 8.1.2 节已经对它们进行了说明，本节就不再赘述。

plot3d 函数同样可以在其他坐标系下绘图，注意这时所对应的坐标名称同二维图形时的坐标系名称几乎完全不同，它所支持的坐标系如表 8-6 所示：

表 8-6 Maple 中可供选择的三维坐标系名称

bipol	bispherical,	cardioidal	cardioidcylindrical
cassc	confocalellip	confocalparab	conical
cylin	ellcylindrical	ellipsoidal	hypercylindrical
invcasscylindrical	invellcylindrical	invoblspheroidal	invprospheroidal
logco	logcylindrical	maxwellcylindrical	oblatespheroidal
parab	paracylindrical	prolatespheroidal	rosecylindrical
sixsphere	spherical	tangentcylindrical	tangentsphere
toroic			



我们举一个简单的例子：

（1）在球坐标下绘图：

```
> plot3d(1,t=0..2*Pi,p=0..Pi,coords=spherical),
```

```
> plot3d(x*sin(y),x=-1..3*Pi,y=0..2*Pi,coords=
  spherical);
```

```
> plot3d(cos(x)*cos(y), x=-1..3*Pi, y=0..2*Pi,
coords= spherical);
```

坐标下绘图：

```
> (1,angle=0..2*Pi,height=-5..5,
coords=cylindrical);
```

```
(theta,theta=0..6*Pi,z=-1..1,
=cylindrical);
```

（3）toroidal 坐标系：

```
> plot3d(theta,theta=0..8*Pi, phi=0..Pi,
  coords=toroidal(1));
```

```
> plot3d(theta/phi,theta=0..8*Pi,phi=0..Pi,
  coords=toroidal(1));
```

8.3.2 多面体的生成

上一节主要介绍的是如何利用函数生成三维图形，这一小节将介绍如何生成一些基本的三维图形。虽然一些多面体表面上看起来很简单，如正四棱柱，正四面体等，但它们并不容易

函数来表示。这时，就需要 Maple 系统为我们提供一些缺省的函数来生成它们。这项任务的是 polyhedraplot 函数。它的基本形式是：

polyhedraplot(L,option)

L 为多面体顶点坐标列表，option 为与 plot3d 相同的三维图形参数设置。如果希望系统自动生成一个多面体，则可以将 L 指定为一个点：[0，0，0]。比如生成一个正六面体：

```
> p
```

```
aplot([0,0,0],polytype=hexahedron);
```

同样利用“hexahedron”参数，我们还可以生成一个“魔方”，如：

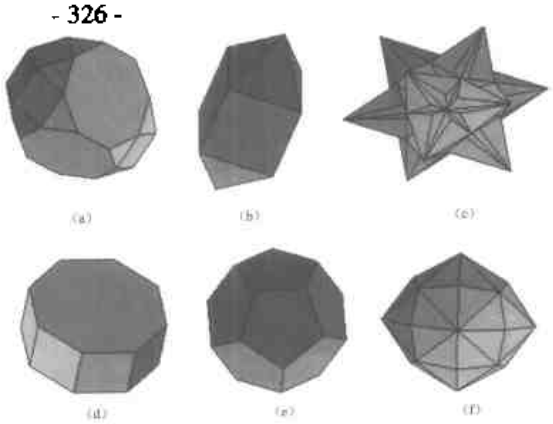
```
> polyhedraplot([0,0,0],[0,0,1],[0,1,0],[0,1
  ,1],[1,0,0],[1,0,1],[1,1,1],[1,1,0],polyty
  pe=hexahedron);
```

在使用 polyhedraplot 函数时，最重要的参数就是 polytype，利用系统内建的多种多边形结构，用户可以方便的生成几乎所有的规则多边形。利用函数“polyhedra_supported（）”，可以获得所有 Maple 支持的多边形列表。

出乎一般人的意料，输入“polyhedra_supported（）:”后，系统显示的内建多面体名称有两屏之多，图 8-19 为排序后的名称列表：

[*AugmentedDodecahedron*, *AugmentedHexagonalPrism*, *AugmentedPentagonalPrism*, *AugmentedSphenocorona*, *AugmentedTriangularPrism*, *AugmentedTruncatedCube*, *AugmentedTruncatedDodecahedron*, *AugmentedTruncatedTetrahedron*, *BiaugmentedPentagonalPrism*, *BiaugmentedTriangularPrism*, *BiaugmentedTruncatedCube*, *BigyrateDiminishedRhombicosidodecahedron*, *DecagonalAntiprism*, *DecagonalPrism*, *DiminishedRhombicosidodecahedron*, *ElongatedPentagonalCupola*, *ElongatedPentagonalDipyramid*, *ElongatedPentagonalGyrobicupola*, *ElongatedPentagonalGyrobirotunda*, *ElongatedPentagonalOrthobicupola*, *ElongatedPentagonalOrthobirotunda*, *ElongatedPentagonalOrthocupolarotunda*, *ElongatedPentagonalPyramid*, *ElongatedPentagonalRotunda*, *ElongatedSquareCupola*, *ElongatedSquareDipyramid*, *ElongatedSquareGyrobicupola*, *ElongatedSquarePyramid*, *ElongatedTriangularCupola*, *ElongatedTriangularDipyramid*, *ElongatedTriangularGyrobicupola*, *ElongatedTriangularOrthobicupola*, *ElongatedTriangularPyramid*, *GreatDodecahedron*, *GreatIcosahedron*, *GreatStellatedDodecahedron*, *GyrateBidiminishedRhombicosidodecahedron*, *GyrateRhombicosidodecahedron*, *GyroelongatedPentagonalBicupola*, *GyroelongatedPentagonalBirotunda*, *GyroelongatedPentagonalCupola*, *GyroelongatedPentagonalCupolarotunda*, *GyroelongatedPentagonalPyramid*, *GyroelongatedPentagonalRotunda*, *GyroelongatedSquareBicupola*, *GyroelongatedSquareCupola*, *GyroelongatedSquareDipyramid*, *GyroelongatedSquarePyramid*, *GyroelongatedTriangularBicupola*, *GyroelongatedTriangularCupola*, *HexagonalAntiprism*, *HexagonalPrism*, *HexakisIcosahedron*, *HexakisOctahedron*, *MetabiaugmentedDodecahedron*, *MetabiaugmentedHexagonalPrism*, *MetabiaugmentedTruncatedDodecahedron*, *MetabidiminishedIcosahedron*, *MetabidiminishedRhombicosidodecahedron*, *MetabigyrateRhombicosidodecahedron*, *MetagyrateDiminishedRhombicosidodecahedron*, *OctagonalAntiprism*, *OctagonalPrism*, *ParabiaugmentedDodecahedron*, *ParabiaugmentedHexagonalPrism*, *ParabiaugmentedTruncatedDodecahedron*, *PentagonalPyramid*, *PentagonalRotunda*, *PentakisDodecahedron*, *RhombicDodecahedron*, *RhombicTriacontahedron*, *SmallStellatedDodecahedron*, *SnubDisphenoid*, *SnubSquareAntiprism*, *SquareAntiprism*, *SquareCupola*, *SquareGyrobicupola*, *SquareOrthobicupola*, *SquarePyramid*, *TetrakisHexahedron*, *TrapezoidalHexecontahedron*, *TrapezoidalIcositetrahedron*, *TriakisIcosahedron*, *TriakisOctahedron*, *TriangularCupola*, *TridiminishedIcosahedron*, *TridiminishedRhombicosidodecahedron*, *TrigyrateRhombicosidodecahedron*, *bilunabirotunda*, *disphenocingulum*, *dodecahedron*, *echidnahedron*, *hebesphenomegacorona*, *hexahedron*, *icosahedron*, *octahedron*, *octahemioctahedron*, *sphenocorona*, *sphenomegacorona*, *tetrahedron*, *tetrahemihexahedron*]

图 8-19 Maple 内建的多面体名称



E此书中一一给出实例，如果读者仔细观察，回发现
·为读者做举例说明，如图 8-20 所示：

图 8-20　系统内建多面体举例

（a）BiaugmentedTruncatedCube；（b）ElongatedSquareDipyramid；（c）GreatIcosahedron；（d）OctagonalPrism；（f）HexakisOctahedron；（e）dodecahedron

8.3.3　其他三维作图函数

在介绍完基本三维函数作图以及基本多面体的绘制后，再向读者介绍一些 Maple 的 plots 程序库中附带的其他可以绘制三维图形的函数。对于那些以“3d”为结尾的函数，由于它们大部分的参数设置同它的“母函数”没有区别，而且绘图功能类似，这里就不再详细介绍此类函数了。下面列出的一些，都是有特殊功能的函数：

1．cylinderplot

函数的完全形式为：

cylinderplot(L,r1,r2,options)

这里的 L 为有两个变量的函数或表达式，r1，r2 则分别对应这两个变量的取值范围，而 options 则可以是任意系统内建的三维函数绘图参数。此函数的使用方法如下：

```
:s):
>lot(z+exp(cos(4*theta)),
.Pi,z=0..3);
```

实际上由函数名读者就可以看出，cylinderplot 是在柱坐标下的作图，只不过 Maple 将原来柱坐标的手续稍微简化了。我们可以利用 plot3d 函数绘制出同样的图形：

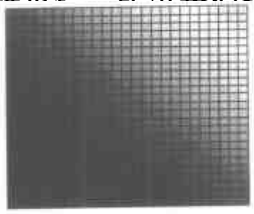
```
>xp(cos(4*theta)),theta=0..Pi,z=0..3
coords=cylindrical);
```

cylinderplot 内参数 L 还可以表示成包含总共两个变量的三个函数的集合，这时实际上就是对一个参数方程绘图，需要为两个参数分别指定取值范围：r1,r2。如：

```
> cylinderplot([z*theta,theta,cos(z^2)],theta=
  0..Pi,z=-2..2);
```

2. densityplot

顾名思义，densityplot 的作用是绘制密度曲线。虽然利用此函数绘制出的曲线是在二维坐标上显示的，但由于它将坐标上的每个点都以灰度显示，相当于在其 Z 方向上又多了一维坐标，所以我们将它归类在三维作图函数中。densityplot 函数的完全形式为：



densityplot(expr, x=a..b,y=c..d)或

densityplot(f,a..b,c..d)

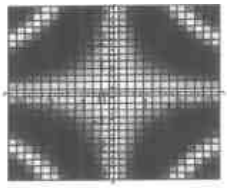
中，参数“expr”代表一个包含有两个变量的表达式，后边利用“x=a..b,y=c..d”的形式确定这两个变量的取值范围。第二种形式中的“f”为一个以 x，y 为变量的方程，因此后边可以直接以“a..b,c..d”的形式指定变量的取值范围。例如：

```
t(sin(x+y),x=-1..1,y=-1..1);
```

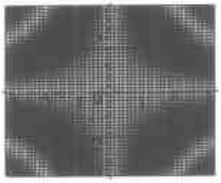
读者可以将 densityplot 理解成对三维图形的压缩形式。即将一个三维投影到二维平面上，再根据其 Z 轴高度对应成每个点的亮度。例如用 plot3d 重新绘制上一个函数：

```
> plot3d(sin(x+y),x=-1..1,y=-1..1);
```

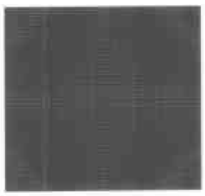
在使用 densityplot 函数时，系统会缺省绘制 25*25 个数据点，即总共 625 个数据点。但有时，在使用缺省参数时获得的效果非常不好，这时就需要改变取样点数来获得更平滑、更准确的曲线。如：



```
:(cos(x*y),x=-Pi..Pi,y=-Pi..Pi);
```



```
> (cos(x*y),x=-Pi..Pi,y=-Pi..Pi,gri
;
```



当然采样点数目并不是可以无限制提高的。更精细的图形的代价是消耗更多的时间以及大量的系统资源。而且有时候更多的采样点并不能带来更好的显示效果。如图 8-21 所示，当我们将采样点扩充 4 倍后，获得的图形并不十分理想：

图 8-21　高采样点绘图后的结果

3. matrixplot（A, option）

A 对应一个矩阵，option 为一般三维作图参数。这是 Maple 中很奇怪的一个函数。它将一个矩阵的余子式表示成三维形式。即三维坐标中的 x 轴对应矩阵的列向量位置，y 轴

对应横向量位置，而将相应位置上矩阵元所对应的余子式的大小对应图像中此点的 z 轴坐标。由此而构成一个三维曲面。比如：

```
> with(linalg):
> B:=toeplitz([1,2,3,4,-4,-3,-2,-1]);
```

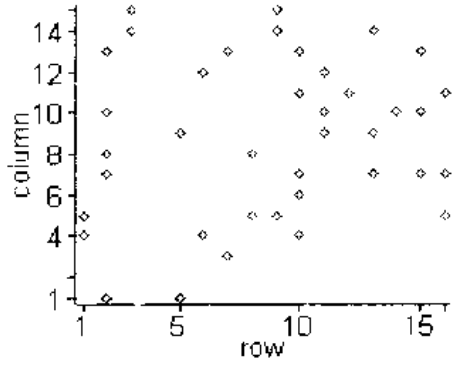
$$B := \begin{bmatrix} 1 & 2 & 3 & 4 & -4 & -3 & -2 & -1 \\ 2 & 1 & 2 & 3 & 4 & -4 & -3 & -2 \\ 3 & 2 & 1 & 2 & 3 & 4 & -4 & -3 \\ 4 & 3 & 2 & 1 & 2 & 3 & 4 & -4 \\ -4 & 4 & 3 & 2 & 1 & 2 & 3 & 4 \\ -3 & -4 & 4 & 3 & 2 & 1 & 2 & 3 \\ -2 & -3 & -4 & 4 & 3 & 2 & 1 & 2 \\ -1 & -2 & -3 & -4 & 4 & 3 & 2 &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

```
> F := (x,y) -> sin(x*y):
> matrixplot(B,color=F);
```

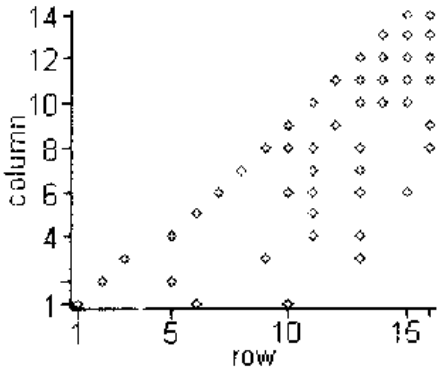
有关矩阵的余子式以及 toeplitz 函数，详见第 3 章“线性代数”。一般使用 matrixplot 函数对特殊矩阵或特殊函数矩阵的余子式作图，可以让用户对这些特殊矩阵的特性做进一步了解。在 plots 程序库中，还有同矩阵相关的一个函数：sparsematrixplot，它同 matrixplot 函数在使用形式上相同。但 sparsematrixplot 仅是将矩阵中的元素投影在二维平面上，并不表示出元素的大小。因此正像此函数的名字一样，它针对的对象是“散矩阵”，即大多数元素为零的矩阵。它可以显示出矩阵元素的空间分布，为用户提供一种感性的认识，例如我们随机生成一个“散矩阵”，并观察几种消元法化简后的效果：

```
> A:=randmatrix(15,16,sparse):
> B:=evalf(gausselim(A),4):
> C:=evalf(gausselim(A),4):
```

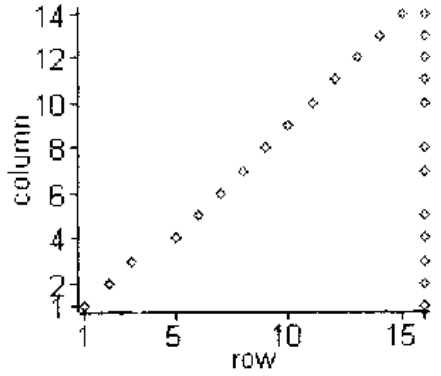
```
> sparsematrixplot(A);
```




```
> sparsematrixplot(B);
```




```
> sparsematrixplot(C);
```



如果读者对第 3 章中的 gauss、gauss-jord 消元法还没有透彻理解的话，看完以上的例子，应该能获得一个感性的认识。这也就是 sparsematrixplot 函数的目的所在。

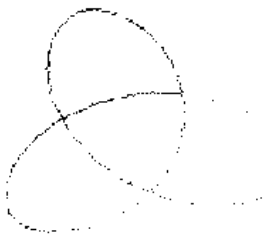
4. spacecurve（L,options）

spacecurve 函数使用参数方程的方法绘制空间曲线。参数 L 为一个含三个表达式的函数列表，系统默认将列表中的三个表达式分别对应 X，Y，Z 坐标。三个表达式一般只有一个变量。变量的取值范围可以同时在函数列表中定义，也可以放到列表外同 options 一起定

义。如：

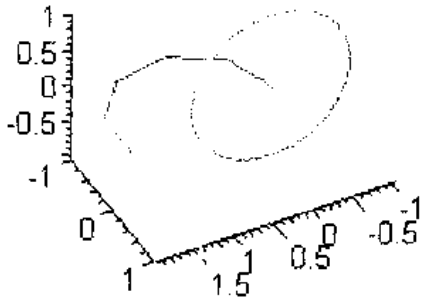
```
> spacecurve([cos(t),sin(t),t/5,t=0..4*Pi]);
```

```
> knot:= [-10*cos(t)- 2*cos(5*t)+15*sin(2*t),
  -15*cos(2*t)+10*sin(t)-2*sin(5*t),10*cos(3*t)]:
> spacecurve(knot,t=0..2*Pi);
```



也可以在 spacecurve 函数中添加多个参数曲线，而在一幅图中，显示出多条空间曲线间的位置关系，如：

```
> spacecurve({[sin(t),0,cos(t),t=0..2*Pi],
  [cos(t)+1,sin(t),0,numpoints=10]},
  t=-Pi..Pi,axes=FRAME);
```



spacecurve 的作用是弥补 plot3d 在参数曲线绘制方面的不足。因为在前文已经提过，plot3d 绘制参数曲线必须是有两个变量的参数平面，如果希望生成空间曲线类型的只包含单一参数的参数曲线，使用 plot3d 函数会产生系统错误。然而 spacecurve 函数只能对单变量参数方程绘图，请读者注意。

5. tubeplot（C，options）

同 类似，tubeplot 函数也是以参数方程的形式绘制空间的管状图形。它参数中“C 间曲线参数方程，一般也对应同一变量的三个表达式，而且系统也同样会按顺序将三个表达式指定为图像对应的在 X，Y，Z 轴的坐标。比如我们用 tubeplot 函数对上节中使用过的参数方程再绘制一遍：

```
> tubeplot([cos(t),sin(t),t/5,t=0..4*Pi]);
```

```
> (knot,t=0..2*Pi);
```

可以看出，虽然图形的结构同使用 spacecurve 函数绘制的图形类似，但在效果上已有了很大的差别。

相对于 spacecurve 函数，由于 tubeplot 生成的是三维形态的图形，所以在 options 选项中，它有自己独特的两个参数：tubepoints 与 numpoints。这两个参数同普通三维参数中的 grid=[m，n]的效果一样，但系统的默认值却发生了改变。其中 numpoint 对应[m,n]中的 m，系统默认值为 50，tubepoints 对应 n，默认值为 10。通过调整采样点的多少，可以获得不同的显示效果。如：

```
> tubeplot([sin(t),cos(t),1],t=-Pi..Pi,
  grid=[30,6]);
```

```
> t([sin(t),cos(t),1],t=-Pi..Pi,
grid=[6,30]);
```

```
> ([sin(t),cos(t),1],t=-Pi..Pi,
0,30]);
```

```
> tubeplot([sin(t),cos(t),1],t=-Pi..Pi,
  grid=[60,60]);
```

tubeplot 函数的另一个特别参数是“radius”，它可以定义所绘制的管状图形半径的大小。同时，用户也可以将它表示成参数参数方程变量的某种函数形式，从而可以获得一个半径随时变化的图形。例如：

```
([cos(t),sin(t),0,t=-Pi..Pi,radius
      =0.25*(t-Pi)]);
```

tubeplot 函数还可以同时显示几组三维图形，从而获得更有趣的结果。例如：

```
> tubeplot([cos(t),sin(t),0,[0,sin(t)-1,
  cos(t)]}, t=0..2*Pi, radius=1/2);
```

在 tubeplot 函数，以及其他的三维作图函数中，用户还可以将图形的渲染颜色定义成参数方程的形式，从而获得更加绚丽的三维物体。由于印刷原因，本书无法将改变颜色参数后获得的效果显示出来，有兴趣的用户可以使用如下函数来观察效果。

```
> F := (x,y)->sin(x*y):
> tubeplot({[cos(t),sin(t),0],[0,sin(t)-1,
  cos(t)]},t=0..2*Pi,radius=1/4,color=F):
```

8.3.4　三维图形参数设置

不同于第二节二维图形绘制中在开始就介绍相关的参数设定，我们在三维图形绘制的最后才将其涉及的参数介绍给读者。由于毕竟不是专业绘图软件，Maple 可以让用户设定的参数并不多，而且某些参数也无法细致调整。所以三维函数的绘制在 Maple 中还是很粗糙的，一般只为用户提供一个感性的认识。如果用户想获得真正细致的三维图像，笔者还是建议使用如 3DS MAX 之类的专业软件。这里只将三维函数绘制所涉及的参数简单列出，以供读者参考。

1. ambientlight=[r,g,b]

设置照明光源的颜色，[r，g，b]分别对应 RGB 颜色中的三原色的强度。

2. axes=f

设置坐标轴的式样，f 是“BOXED,NORMAL,FRAME,NONE”四个名字之一。

3. axesfont=1

设置坐标轴的字体。所设参数同二维设定中 FONT 参数。

4. color=c 或 colour=c

设置三维物体的渲染颜色，颜色设定与二维颜色相同。

5. contours=n

设定等高线作图时等高线的数目，默认 n=10。

6. coords=c

设置绘制三维图形所在的坐标系。详细设定可以参看 plot3d[coords]。

7. filled=true(false)

当设定为 true 时，函数绘制的三维平面与坐标平面间会被填充起来。默认值为 false。此参数只对 plot3d,contourplot3d,listcontplot3d 起作用。

8. font=1

可设定的参数同二维函数中的 FONT 设定。

9. grid=[m,n]

设定绘制三维图形时使用的四方格子点数。不同三维函数都有默认的格点数目。

10. gridstyle=x

设置使用何种方式拼接三维图形，x 默认为“rectangular”，即 4 边形，用户还可以将它设定为“triangular”，利用三角形拼接三维图形。

11. labeldirections=[x,y,z]

x,y,z 分别对应三个坐标轴标签的书写方向，默认为水平书写“HORIZONTAL”，用户还可以将它设定为“VERTICAL”。即垂直绘制。

12. labelfont=l

设定坐标轴标签的字体，所使用参数同 FONT。

13. labels=[x,y,z]

设定坐标轴标签的内容，x，y，z 只能为字符串格式，分别对应三个坐标轴的标签。

14. light=[phi,theta,r,g,b]

设定光源的方向及颜色，phi,theta 分别对应球坐标中的两个角度，r,g,b 分别对应 RGB 着色方案中三原色的数值。为 0 到 1 间的数值。

15. lightmodel=x

对应光源的类型。用户可以将 x 设定为“light1”“light2”“light3”“light4”4 种系统定义类型之一。默认值为“none”。

16. linestyle=n

设定构架三维图形的曲线的线型，可选择的 n 的数值同二维图形参数中的 linestyle。

17. numpoints=n

设定三维图形的最小绘制点数。同二维图形参数。

18. orientation=[theta,phi]

设定生成三维图形的朝向，theta,phi 分别对应球坐标的角度，默认值均为 45 度。

19. projection=r

设定透视投影的类型。r 可以是 0 到 1 间的任意数值。还可以是系统自定义的三个名字之一：“FISHEYE”，“NORMAL”和“ORTHOGONAL”，分别对应着 0，0.5，1 三个数值。默认为“ORTHOGONAL”。

20. scaling=s

同二维参数中的 scaling，设定三维图形是否严格按照比例绘制。

21. shading=s

设定三维物体表面的颜色，s 可以是 XYZ,XY,Z,Z-GREYSCALE,Z-HUE,NONE 之一。

22. style=s

设定三维物体表面帖图的类型。s 可以是 POINT,HIDDEN,PATCH,WIREFRAME,CONTOUR，PATCHNOGRID，PATCHCONTOUR 或 LINE。具体对应内容见 8.1.2 节。默认为 PATCH。

23. symbol=s

设置三维图形的线型，s 可以是 BOX,CROSS,CIRCLE,POINT 或 DIAMOND。

24. view=zmin..zmax

设置显示图形时 z 轴的显示范围，同样可以设定为 xmin..xmax,ymin..ymax。默认为显示整个三维物体。

8.4 特殊图形绘制——PLOTTOOLS 程序库简介

plottools 程序库中汇集了各种常用的作图函数。它的作用就是简化用户的输入，方便图形的生成。在这里，我们简要介绍其中的一些函数。由于此程序库的目的就是方便用户的输入，所以它的参数配置十分简单，相信读者可以有举一反三的能力。

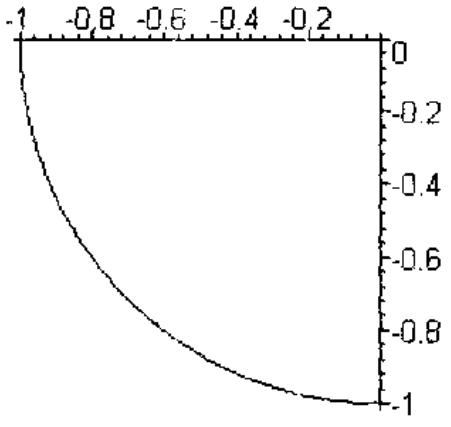
需要提醒读者的是，plottools 中的函数生成的是 Maple 的图形数据结构数据。用户无法直接利用它获得图形式的结果，必须要调用 plots 程序库中的 display 函数来显示这些数据。因此 plottools 程序库更像是 plots 库的扩展函数库。必须联合调用才可以获得图形方式的显示结果。以下，让我们来具体看几个例子。

8.4.1 绘制圆弧—arc

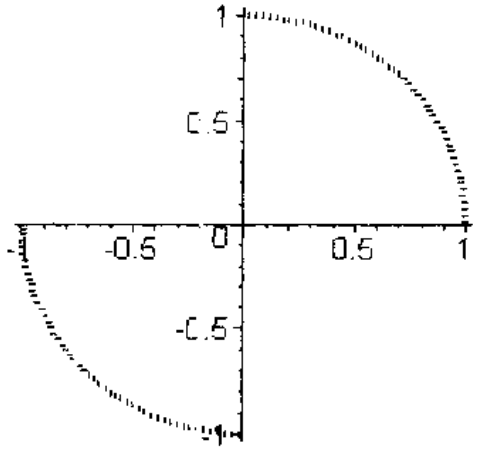
arc(c,r,a..b)

参数中的 c 代表圆弧的中心，r 为半径，a..b 为第一段原弧的起、止位置，对应极坐标中的角度，以弧度表示。arc 函数的使用如下所示：

```
> f:=arc([0,0],1-Pi..-Pi/2):
> display(f);
```

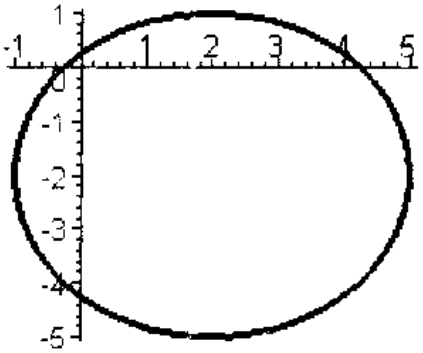


```
> display(arc([0,0],1,-Pi..-Pi/2),
  arc([0,0],1,0..Pi/2),linestyle=2, thickness=3);
```

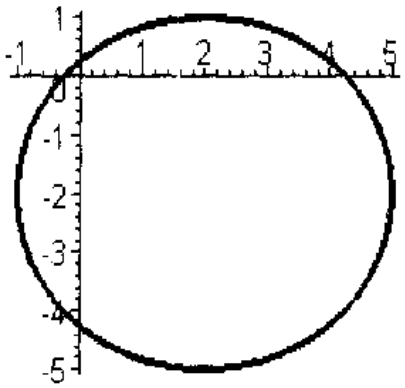


注意在 arc 函数中，有很多 plot 函数可以使用的参数并不起作用，要在使用 display 函数时添加才可以生效。一般同线型、线宽相关的参数可以在使用 arc 函数时同时定义，而在 display 函数中则可以定义任何参数，比如：

```
> display(arc([2,-2],3,0..2*Pi,thickness=3,
  scaling=constrained));
```




```
> display(arc([2,-2],3,0..2*Pi,thickness=3,
  scaling=constrained));
```



实际上，arc 函数的这种性质是在 plottools 程序库中普遍存在的。因为 Maple 图形数据结构中没有专门的对象储存有关图形显示方式的数据，所以即使在函数生成时定义了类似 scaling 的参数，也不可能在使用 display 函数时表现出来。有关 Maple 图形的数据，由于同用户并没有太大关系，所以在这里并不准备对它做详细介绍。有兴趣的读者，可以观察 lprint 函数显示出的具体内容，自己分析。例如：

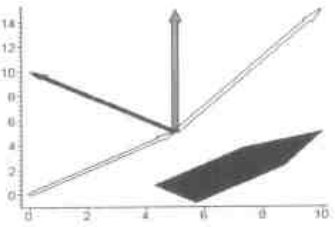
```
> lprint(plot(x,x=0..4,numpoints=10,adaptive=false));
PLOT(CURVES([[0., 0.], [.4650(5955555555516, .465005955555555516],
[.86960515l111111140, .869605151111111140], [1.3246192622222215, 1
.3246192622222215], [1.78264950222222218, 1.78264950222222218], [2
.2385028044444439, 2.2385028044444439], [2.66113716888888874, 2.6
6113716888888874], [3.09875057333333314, 3.09875057333333314], [3.5
5133098222222188, 3.55133098222222188], [4., 4.]], COLOUR(RGB, 1.0, 0.
,0.)), AXESLABELS("x",""),VIEW(0...4., DEFAULT))
```

8.4.2 绘制箭头--arrow

arrow(base, dir, wb, wh, hh)

base：箭头的起始点，为一个二维、三维点或一个向量；dir：箭头的方向，类型同 base 一样；wb：箭头“杆”的宽度，为一数值；wh：箭头“尖”的粗细，为一数值；hh：箭头“尖”相对于“杆”的长度，为一比例数值。在 hh 参数后，用户还可以添加其他同线型相关的函数。arrow 函数的效果如下：

```
> l1 := arrow([0,0], [5,5], .2, .4, .1,
  color=white):
> l2 := arrow([5,5], vector([0,10]), .2, .4, .1,
  color=gray):
> l3 := arrow([5,5], [0,10], .2, .4, .1,
  color=black):
> l4 := arrow(vector([5,5]), vector([5,10]), .2,
  .4, .1,  color=white):
> l5 := arrow([5,0], vector([5,5]), 2, 2, .4,
  color=black):
```



```
,12,13,14,15,axes=framed);
```

从这个例子中，读者可以观察出使用向量方式绘制箭头与使用坐标点方式的不同。简单的说，[illegible]row 中的参数是向量，则绘制的就是由起始点开始的对应向量，如果参数是坐标，[illegible]row 函数就绘制由起始点连接到终点的箭头。定义三维坐标或是三维向量同二维[illegible]司，这里就不再举例说明。

8.4.3　绘制圆锥、圆球—cone，sphere

cone(c, r, h, ...)

sphere(c, r, ...)

这两个函数的参数几乎一样。c 对应着圆锥或圆球的中心点坐标；r 对应着圆锥或圆球的半径；cone 函数中的 h 对应着圆锥的高度；其后的参数同普通三维绘图参数类似。除了参数 c 必须指定外，r，h 都是可选的，如果用户没有指定，系统会默认设定为 1。例如我们联合使用这两个函数绘制一个“冰激凌”：

```
> icecream := cone([0,0,-2],0.8, 2, color=gray),
  sphere([0,0,0.1], 0.6, color=white):
> plots[display](icecream, scaling=constrained);
```

注意这里我们将由两个不同函数生成的三维数据顺序保存在了变量 icecream 函数中，中间使用了“,”作为分割符。当然我们可以将两个函数分别定义成两个变量。三维图形的数据包含的信息量一般十分巨大，有兴趣研究的读者可以在设置了小网格点后仔细研究其结构。例如：

```
> lprint(sphere([0,0,0],1,color=white,grid=[3,3]));
POLYGONS([[0., 0., 1.],[1., 0., .1615544574e-14],[-1., .323108
9149e-14, .1615544574e-14],[0., 0., 1.]],[[1., 0., .1615544574e
-14],[.3231089149e-14, 0., -1.],[-.3231089149e-14, .1043993709
e-28, -1.],[-1., .3231089149e-14, .1615544574e-14]],[[0., 0., 1
.],[-1., .3231089149e-14, .1615544574e-14],[1., -.6462178298e-
14, .1615544574e-14],[0., 0., 1.]],[[-1., .3231089149e-14, .161
5544574e-14],[-.3231089148e-14, .1043993709e-28, -1.],[.323108
91                087987418e-28, -1.],[1., -.6462178298e-14, .161554
45          COLOR(RGB,1.,1.,1.))
```

虽然 phere 函数不会绘制出有形状的图形，但读者可以由此发现 Maple 系统是用 polygons 函数，即多边形函数来近似表示球形的。这实际也是计算机图形学的基本原理。

8.4.4 切割多边形的绘制—cutin 与 cutout

这两个函数不仅在名称上类似，而且实现的功能也很相似。以 cutin 函数为例，它的完全形式为：

cutin(p,r)

p 为系统定义的多面体名称，其详细列表如图 8-19 所示。r 为取值范围从 0 到 1 的浮点数或分数。cutin 函数在原来显示多面体的地方生成一个同原始图形大小相同的多面体，但其各个表面是由原始多面体表面由外缘按比例“r”切割后生成的表面。也许这样解释并不是很清除，请读者看一下的例子：

```
> display(cutin(octahedron(), 2/3));
```

```
> (octahedron());
```

```
> (cut in (dodecahedron(), 2/3));
```

```
ay(dodecahedron());
```

为了对比 cutin 函数的效果，我们显示了一个完成的正十二面体。注意我们在这里并没有使用 8.3.2 节介绍的 plots 程序库中的 polyhedraplot 函数（如果要使用 polyhedraplot 函数，用户则需要输入“polyhedraplot ([0,0,0], plottype=dodecahedron)”），而使用了一个简单的命令 dodecahedron()，就生成了这个正十二面体。dodecahedron()函数是 plottools 库函数，类似的，plottools 库中还设定了“hexahedron：正六面体，icosahedron：正二十面体，octahedron：正八面体，tetrahedron：正四面体”四个正多面体绘制函数。如此，就可以利用这些简单的函数绘制这五种等边多面体。

cutout 函数同 cutin 函数正好相反，它将所有 cutin 函数砍掉的部分显示出来，将两个函数显示的相同比例的图形拼接起来，还可以形成一个完整的多面体。例如：

```
> display(cutout(dodecahedron(), 2/3));
```

```
> {%,%%%};
```

利用 cutout 函数的极限形式，我们还可以生成多边形的骨架，其效果同在生成多面体时使用参数“style=wireframe”类似。如：

```
> display(cutout(dodecahedron(), 20/21));
```

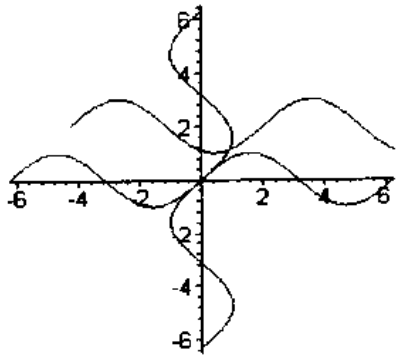
8.4.5 多边形的反射—reflect

reflect 可以生成二维或三维图像的反射图。它的完全形式为：

reflect(p,[points])

这里 p 为一个二维或三维图形的数据点，“[]”中的点为用户希望作为反射点的坐标。对二维图形，可以是一个二维坐标或两个二维坐标，单点反射的效果如同平移曲线，两点反射的效果为以这两个点的连线作为反射中心的镜面反射，如：

```
> p := plot(sin, -2*Pi..2*Pi):
> display(p, reflect(p, [1,1])
  reflect(p, [[0,0], [1,1]]));
```



对三维图形，可以是一个点，两个点确定的直线，或三个点确定的平面。比如以平面 z=0 为中心对三维函数 sin(x*y)反射：

```
3d(sin(x*y)+3,x=-Pi..Pi, y=-Pi..Pi):
ect(p,[[0,0,0],[1,0,0],[0,1,0]]):
isplay]([p,q]);
```

8.5　其他绘图相关函数

作为对前几节的补充，同时也为了让读者能更全面的了解 MAPLE 的绘图系统，我们在这一节中对一些没有被包括在前几节的绘图函数或程序库进行举例说明，其中包括函数“smartplot/smartplot3d”，“animate”，“showtangent”，以及程序库“geometry”。如果读者对其中的某些函数感兴趣，可以通过阅读联机帮助来获得更详细的信息。

8.5.1　智能绘图函数：smartplot 及 smartplot3d

在使用 plot、plot3d 或 display 函数绘图时，用户会发现当图形对象生成后，留给用户可以继续修改的余地非常小，甚至只想修改坐标的取值范围，也需要在对原始函数进行修改后重新绘图才行。如果在一幅图上同时显示多个函数曲线，就很难对不同的曲线分别修改。为此，MAPLE 提供了系统内置函数“smartplot”和“smartplot3d”来帮助用户实现这些功能。smartplot/smartplot3d 的函数形式十分简单，用户只需要输入：smartplot(函数表达式)。它生成不同于 plot 函数的图像格式，同时，用户可以对图像进行后期处理。

smartplot 函数的主要特点体现在用户可以在鼠标的帮助下，对一幅图中显示的多条曲线进行分别处理，同时可以实现拷贝、移动等功能。当用户选择了由 smartplot 函数生成的图形对象后，点击鼠标右键，会出现如图 8-22 所示的选项：

图 8-22　smartplot 图形对象的控制菜单

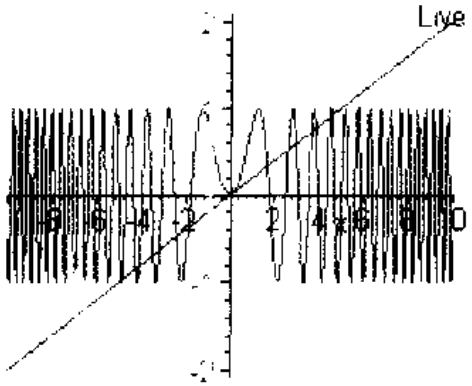
的图形上的鼠标右键菜单，这里多出了 Color 选项以及 Axe 中的 Ranges。注意只有当鼠标位于某条绘制的曲线或曲面上方时，才会出现 Color 选项，这时，改变 Color 选项的内容，可以单独改变这条曲线的颜色。类似的，如果此时改变右键菜单中的 Style 选项的内容，也会单独改变这条曲线的式样，如果鼠标所处的位置没有任何曲线，这时改变 Style 选项将同时改变图中所有曲线的式样。当用户选择了 Ranges 选项，并在弹出的窗口中修改相应坐标的取值范围，就会得到预期的显示效果。如图 8-23 所示：

图 8-23　Axis Ranges 窗口

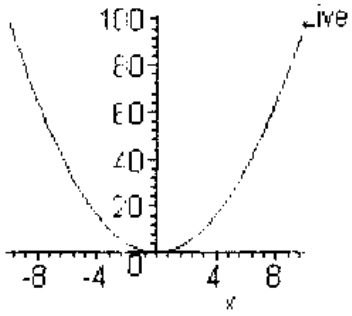
用户还可以用类似编辑文字的方式通过拖动鼠标来移动单条曲线，或以 Ctrl 键加鼠标拖动的方式来复制单条曲线。在下例中，我们将一步步显示如何对 smartplot 函数生成的对象进行修改：

（1）分两次生成 $\sin(x^2)$,x/5 和 x^2 曲线。

```
> smartplot(sin(x^2),x/5);
```




```
> smartplot(x^2);
```



（2）利用鼠标将第二幅图复制到第一幅图，如图 8-24 所示：

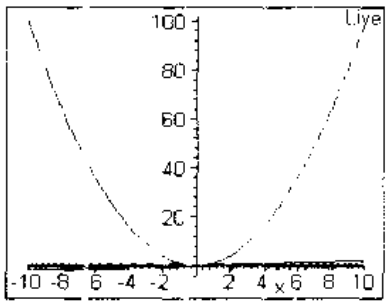


图 8-24　利用鼠标拷贝曲线

（3）改变坐标轴范围，获得更好的显示效果。如图 8-25 所示：

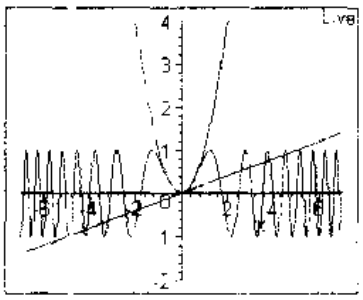


图 8-25　改变坐标轴范围后的效果

（4）改变各条曲线的线型与线宽，获得区别效果。如图 8-26 所示：

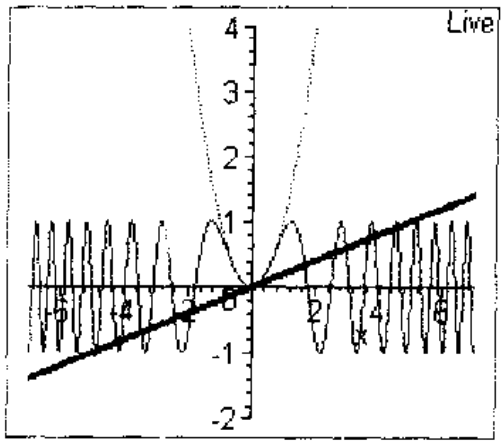


图 8-26 分别改变各条曲线线型后的效果

8.5.2 动画图形显示——animate 函数

利用 plots 程序库中的函数 animate/animate3d，Maple 可以将一系列的静态图形连续显示，从而生成动画式的结果。由于相对于 matlab，Maple 的动画效果不是十分理想，而且本书中也无法演示不同参数所对应的不同动画效果，所以在这里我们仅对 animate 函数做简单介绍，用户可以通过运行系统联机帮助中的例子来观察实际的动画效果。

animate 函数的完全表达形式为：

animate(F,x,t,...)

F 为用户希望绘制的动画曲线，是变量 x 与时间 t 的函数。x,t 两个参数对应创建函数时横坐标的取值范围以及时间变化的取值范围。其后可以附加同 plot 函数类似的其他二维图形参数。比如用户输入如下命令：

plots[animate](x*t,x=-1..1,t=1..3,numpoints=5,frames=100);

就可以获得一组斜率随时间变化的直线的动画图形。注意当系统生成动画对象时，并不会自动播放，用户须在点击了动画对象后，利用动画工具条（见图 8-15），或 animate 菜单来控制动画的播放过程，有关它们的具体介绍，参看 8.1.3 节。

读者可以尝试利用如下程序建立一个稍微复杂的动画对象，显示出一个“鸡心”的形成过程：

```
animate([(1+cos(n*t/180*Pi))*cos(n*t/180*Pi),
(1+cos(n*t/180*Pi))*sin(n*t/180*Pi),t=0..1],
n=1..360);
```

最后的结果如图 8-27 所示：

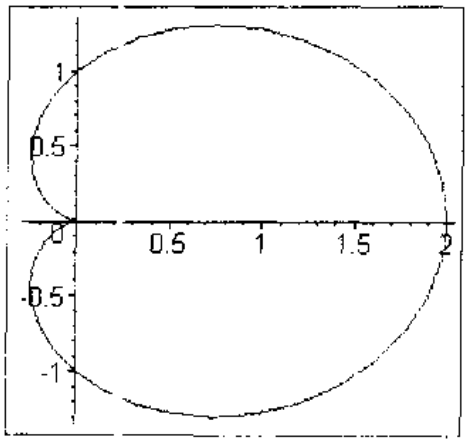


图 8-27　animate 函数生成动画的结果

8.5.3　显示曲线中某点的切线——showtangent

在第 2 章，我们我们曾经对 student 程序包做过简要介绍。这个程序包主要集中了初、高等数学中的常用函数。在绘制图形方面，它只包含了一个函数："showtangent"，作用是显示一条曲线中某个点的切线。它的完全形式为：

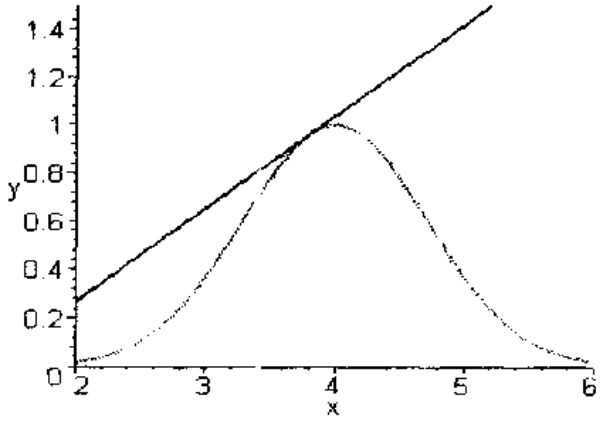
showtangent(f(x),x=a,...)

其中 f（x）为用户所希望寻找切线的曲线方程，"x=a" 指定了曲线上切点的位置。比如我们显示高斯曲线 f(x)=exp(-(x-4)^2)在点 x=3.8 上的切线：

```
> restart;
> f:=x->exp(-(x-4)^2);
```

$$f := x \to e^{(-(x-4)^2)}$$

```
> with(student):
> showtangent(f(x), x=3.8, x=2..6, y=0..1.5,
  thickness=2);
```



系统自动以一条浅黄色线显示切点对应的 x 轴坐标。如果用户希望找到此时的切线方程，除了用对原始函数微分、求出切线斜率后利用切点确定切线方程外，还可以使用 geometry 程序库中的函数来寻找圆形的切点。在下一节中，我们会对 geometry 程序库做简单介绍。

8.5.4 几何作图程序包：geometry

geometry 程序库包含了大量的几何图形分析函数，用于解决各种几何问题。它有自己的数据结构，因此由它的函数生成的图形结构无法直接被 plot 函数或 plots 程序库中的 display 函数调用。而需要使用其自带的函数 draw 来绘制图形。我们先利用 geometry 来求经过某点的一个圆的切线：

```
> restart;
> withf (geomentry):
> circle(c,(x-1)^2 + (y+2)^2 = 9,[x,y]);
```

$$c$$

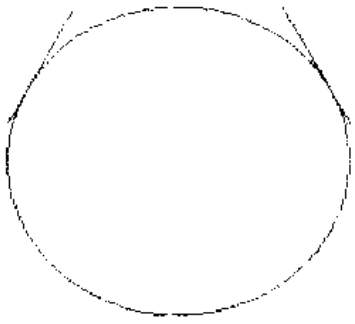
```
> point(P,1,4.5);
```

$$P$$

```
> TangentLine(obj, P, c, [l1, l2]);
```

$$[l1, l2]$$

```
> draw([c, l1, l2], axes=none);
```



```
> Equation(l1);Equation(l2); Equation(c);
```

$$-17.09150732 + 5.115384615x + 2.661360599y = 0$$

$$6.860738081 + 5.115384615x - 2.661360599y = 0$$

$$x^2 - 2x - 4 + y^2 + 4y = 0$$

利用简单的微分运算，可以得到在此点处切线的斜率：

```
> D(x->exp(-(x-4)^2))(x);
```

$$(-2x+8)e^{(-(x-4)^2)}$$

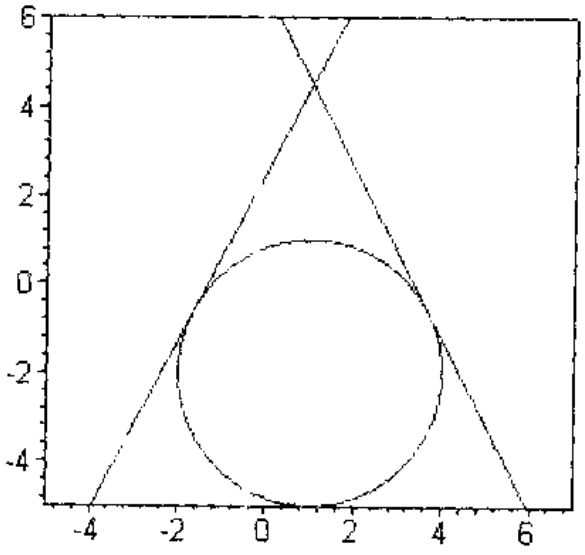
```
> evalf(subs(x=3.8,%));
```

$$.3843157757$$

系统自动在以一条浅黄色线显示切点对应的 x 轴坐标。如果用户希望找到此时的切线方程，除了用对原始函数微分、求出切线斜率后利用切点确定切线方程外，还可以使用 geometry 函数库中的函数来寻找圆形的切点。在下一节中，我们会对 geometry 函数库做简单介绍。

在实际使用 draw 函数绘图时，用户会发现可以在其中使用的参数非常少，甚至连显示坐标的取值范围都无法设定，因此并不能获得很好的显示效果。如果用户想改进利用 geometry 程序库获得的函数图形，可以将使用 draw 函数绘制的图形利用 plots 程序库中的 display 函数重新显示，这样就可以随意改变图形参数来获得最佳的显示效果。例如继续对上例进行操作：

```
> p1:=draw(11):
> p2:=draw(12):
> p3:=draw(c):
> with(plots):
> display([p1,p2,p3],view=[-5..7,-5..6]);
```



读者可以看出 geometry 的主要功能并不是绘制图形，它提供的丰富函数是用来帮助用户计算或证明涉及平面几何或立体几何的问题。下面，我们使用 geometry 程序库中提供的函数，来证明：

三角形 A1A2A3 的垂心 H、外接圆圆心 O 和重心 G 三点共线，同时 HG=2GO。

限于篇幅限制，在证明过程中不对所涉及的函数做详细说明，读者应该可以从例子中使用的命令掌握它基本的使用方法，如果要详细理解，还需要阅读联机帮助。

（1）通过三个顶点，定义一个三角形及其外接圆：

```
> triangle(T, [point(A1, 2, 4), point(A2, 0, 0),
  point(A3, 7, 0)]):
> circumcircle(C, T, 'centernae'=0):
```

（2）利用函数 altitude 找出三角形 T 的三条垂线，其中 A11，A22 和 A33 分别是垂线在对边的垂足，altitude 函数实际上得到了上述三个垂足。

```
> altitude(A2A22, A2, T, A22):
  altitude(A3A33, A3, T, A33):
  altitude(A1A11, A1, T, A11):
```

（3）利用 orthocenter 和 centroid 函数，分别求出三角形 T 的垂心 H 和重心 G：

```
> orthocenter(H, T): centroid(G, T):
```

（4）再利用 median 函数，求出三角形各边的中点及中线：

```
> median(A1M1, A1, T, M1):
  median(A2M2, A2, T, M2):
  median(A3M3, A3, T, M3):
```

（5）调用函数 AreCollinear 检验 H,O,G 三点是否共线：

```
> Arecollinear(O, H, G);
                              true
```

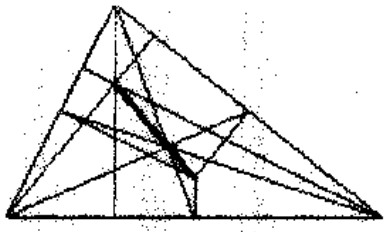
（6）利用 distance 函数确定 HG,GO 的长度，并利用 testeq 函数来判断等式是否成立：

```
> testeq(simplify(distance(H, G))=
  2*simplify(distance(G, O)));
                              true
```

注意在这里我们对 distance 函数计算的结果做了化简。由于 MAPLE 系统不能直接对根式化简，因此我们直接使用 testeq 函数并不能获得正确结果。

（7）利用 draw 函数将以上的结果绘制出来：

```
> draw([C(color=yellow, filled=true),
  T(color=blue), OM1, OM2, OM3, A3M3, A2M2, A1M1, A2A22,
  A3A33, A1A11, dsg1(color=black, thickness=3), dsg2
  (thickness=3,color=black)], axes=NONE);
```



附录 1 Maple 函数库列表

函数库名称	对应英文全称	函数库内容
DEtools	differential equations tools	微积分工具
Domains	create domains of computation	创建计算域
GF	Galois Fields	伽罗瓦迪场
GaussInt	Gaussian Integers	高斯整数相关函数
Groebner	Groebner basis calculations n skew algebras	Groebner 基
LREtools	manipulate linear recurrence relation	线性递归相关函数
Linear Algebra	linear algebra package based on rtable data structures	基于 rtable 数据格式的线性代数相关程序包
Matlab	Matlab Link	与 Matlab 的接口函数
Ore_algebra	basic calculations in algebras of linear operators	线性算子的基本代数运算
PDEtools	tools for solveing partial differential equations	偏微分方程相关函数
Spread	Spreadsheets	扩展工作簿生成函数
algcurves	Algebraic Curves	代数曲线
codegen	Code Generation	程序代码生成器
combinat	combinatorial functions	复合函数
combstruct	combinatorial structures	复合结构
context	context sensitive menus	上下文敏感菜单
diffalg	differential algebra	偏微分代数
difforms	differential forms	微分形式
finance	financial mathematics	金融数学
genfunc	rational generating functions	有理数产生函数
geom3d	Euclidean three-dimensional geometry	欧基里德三维几何
geometry	Euclidean geometry	欧基里德几何
group	permutation and finitely-presented group	排列与有限群（群论相关函数）
inttrans	integral transforms	积分变换
liesymm	Lie symmetries	李对称
linalg	linear algebra package based on array data structures	基本线性代数包

续表

函数库名称	对应英文全称	函数库内容
networks	graph networks	图形化的网络计算函数
numapprox	numerical approximation	数值逼近
numtheory	number theory	数论
orthopoly	orthogonal polynomials	正交多项式
padic	p-adic numbers	P 进制数转换包
plots	graphics package	绘图程序库
plottools	basic graphical objects	基本图形绘制函数
polytools	polynomial tools	多项式相关函数
powseries	formal power series	幂级数
process	(Unix)-multi-processing	Unix 下的多线程计算函数
simplex	linear optimization	线性优化
stats	statistics	统计函数
student	student calculus	学生综合函数库
sumtools	indefinite and definite sums	无限与有限求和
tensor	tensor computations and General Relation	张量操作与广义相对论

附录 2 Maple 基本函数及其功能

此附录中收录的，是几乎所有 Maple 系统的自带函数，即不须添加任何函数库可直接执行的函数。

函数名称	基本功能
Afactor	绝对因式分解
Afactors	另一种返回形式的绝对因式分解
AiriAiZeros	返回 AiryAi 函数的实数根
AiriBiZeros	返回 AiryBi 函数的实数根
AiryAi	“爱里”A 型函数(The Airy wave functions)
AiryBi	“爱里”B 型函数(The Airy wave functions)
AngerJ	Anger 函数
Berlekamp	因数分解
BesselI, BesselJ	第一类贝塞耳函数
BesselK, BesselY	第二类贝塞耳函数
BesselJZeros	第一类贝塞耳函数 n 阶正实数解
BesselYZeros	第二类贝塞耳函数 n 阶正实数解
Beta	Beta 函数
C	生成 C 语言代码(codegen)
Chi	双曲余弦积分
Ci	余弦积分
CompSeq	描述计算序列
Content	content 函数的简单形式
D	微分算子
DESol	描述微分等式解的数据结构
Det	det 函数（行列式）的简化形式
Diff	同 diff
Dirac	狄拉克函数

续表

函数名称	基本功能
DistDeg	高阶因式分解
Divide	divide 函数的简化形式
Ei	指数积分
Eigenvals	数值矩阵的特征值或特征向量
函数名称	基本功能
EllipticCE	第二类互余完全椭圆积分
EllipticCK	第一类互余完全椭圆积分
EllipticCPi	第三类互余完全椭圆积分
EllipticE	第二类完全或不完全椭圆积分
EllipticF	第一类不完全椭圆积分
EllipticK	第一类完全椭圆积分
EllipticNome	q(k)= exp(-Pi*EllipticCK(k)/EllipticK(k))
EllipticPi	第三类完全或不完全椭圆积分
Eval	计算表达式
Expand	简化 expand 函数
FFT	快速傅立叶变换
Factor	因式分解
Factors	因式分解
FresnelC	菲涅耳余弦积分
FresnelS	菲涅耳正弦积分
Fresnelf/Frensnelg	菲涅耳辅助函数
GAMMA	Γ函数与不完全Γ函数
GaussAGM	高斯算术、几何平均值
Gaussejord	高斯约当消元法
Gausselim	高斯消元法
Gcd	最大公约数
Gcdex	扩展多项式欧几里得代数函数
HankelH1	第三类贝塞耳函数
HankelH2	第三类贝塞耳函数
Heaviside	亥维塞阶跃函数
Im	去复数的虚部

续表

函数名称	基本功能
Interp	多项式插值函数(interp 函数的简化形式)
Inverse	逆矩阵（inverse 函数的简化形式）
Irreduc	即约性函数（irreduc 函数的简化形式）
JacobiAM	雅可比幅值函数 am
JacobiSN JacobiCN JacobiDN JacobiNS JacobiNC JacobiND	雅可比椭圆函数
JacobiSC JacobiCS JacobiSD JacobiDS JacobiCD JacobiDC	雅可比椭圆函数
JacobiTheta1 JacobiTheta2 JacobiTheta3 JacobiTheta4	雅可比 θ 函数
JacobiZeta	雅可比 ζ 函数
KelvinBei	开尔文 Bei 函数
KelvinBer	开尔文 Ber 函数
KelvinHei	开尔文 Hei 函数
KelvinHer	开尔文 Her 函数
KelvinKei	开尔文 Kei 函数
KelvinKer	开尔文 Ker 函数
KummerM	Kummer M_mu 函数
KummerU	Kummer U_mu 函数
LambertW	Lambert W 函数
Lcm	最小公倍数
LegendreP	第一类勒让德函数或连带函数
LegendreQ	第二类勒让德函数或连带函数
LerchPhi	一般 LerchPhi 函数
Li	对数积分
Linsolve	简化线性方程求解
LommelS1	Lommel s 函数
LommelS2	Lommel S 函数
MOLS	正交的拉丁平方函数
Maple_floats	计算软件可以实现的不同浮点数极限
MatlabMatrix	连接 Matlab 的矩阵
MeijerG	修正的 MeiJer G 函数

续表

函数名称	基本功能
Normal	normal（通分、化简）函数的简化函数
Nullspace	计算零空间的基
Power	power（幂函数）的简化形式
Powmod	余数的幂函数
Prem	prem(伪余数)函数的简化形式
Primitive	判断多项式是否有能取模
Primpart	primpart 函数的简化形式
函数名称	基本功能
ProbSplit	多项式可分解为同阶因式的可能性
Product	product(乘)函数的简化形式
Psi	双 γ 函数与多 γ 函数
Quo	quo(求多项式的商)函数的简化形式
RESol	一种递归方程解的数据结构
Randpoly	有限域的随机多项式
Randprime	有限域的随机 monic prime 多项式
Ratrecon	ratrecon（重组有理函数）的简化形式
Re	取复数的实部
Rem	rem(求多项式相除余项)函数的简化形式
Resultant	resultant(合成多项式)函数的简化形式
RootOf	求方程的根
Roots	多项式求模后的根
SPrem	sprem(奇异伪余式)函数的简化形式
Searchtext	查找文本
Shi	双曲正弦函数
Si	正弦积分
Smith	矩阵的 Sm th 正交化
Sqrfree	非平方因式分解
Ssi	转移正弦积分
StruveH	Struve H 函数
StruveL	Struve L 函数
Sum	sum(求和)函数的简化形式

续表

函数名称	基本功能
Svd	计算矩阵的奇异数/向量
TEXT	显示文本
WeberE	韦伯函数
WeierstrassP	Weierstrass ℘ 函数
WeierstrassPPrime	Weierstrass ℘ 函数的导数
WeierstrassSigma	Weierstrass Σ 函数
WeierstrassZeta	Weierstrass ζ 函数
WhittakerM	Whittaker M 函数
WhittakerW	Whittaker W 函数
Zeta	黎曼 ζ 函数
abs	绝对值
add	加法
addcoords	添加坐标系
addressof	得到所指表达式的内存地址
algsubs	对多项式子式的替换
alias	定义缩写
allvalues	求出等式的全部可能解
anames	顺序显示已定义的变量名
antisymm	反对称参数
applyrule	应用运算规则
arccos	反余弦函数
arccosh	反双曲余弦函数
arccot	反余切函数
arccoth	反双曲余切函数
arcsec	反正割函数
arcsech	反双曲正割函数
arcsin	反正弦函数
arcsinh	反双曲正弦函数
arctan	反正切函数
arctanh	反双曲正切函数
argument	求复数坐标下的向量角度

续表

函数名称	基本功能
array	定义数组
assign	赋值
assigned	已赋值变量
assume	假设条件
asympt	渐进线展开
attribute	返回变量属性
bernstein	函数的近似 Bernstein 多项式
branches	绘制多值函数的分值图
bspline	计算多项式的 B 样条曲线
cat	串联表达式
ceil	向上取整
charfcn	集合或表达式的特征函数
chrem	中国求余函数
close	关闭非缓冲文件与关闭管道
coeff	求多项式系数
compiletable	设定模式匹配表
compoly	判断一个多项式的可能变量组合
conjugate	求共轭复数
content	判断多变量多项式的内容
convergs	显示递推关系的后项
convert	类型转换
copy	对表或数组元素赋值
cos	余弦函数
cosh	双曲余弦函数
cot	余切函数
coth	双曲余切函数
csc	余割函数
csch	双曲余割函数
cdgn	判断实数或复数的符号
currentdir	显示或设定当前的工作目录
dawson	Dawson 积分

续表

函数名称	基本功能
define	定义变量或过程名
degree	显示多项式为阶
denom	求表达式的分母
depends	判断属性的依赖关系
diff	微分或偏微分
diffop	线性微分算子
dilog	二重对数函数
dinterp	计算可能的插值次数
disassemble	把对象分成非连续地址
discont	计算函数的断点
discrim	计算一次方程的判别式
dismantle	显示 Maple 表达式的数据结构
divide	多项式相除
dsolve	求解一般微分方程
eliminate	对方程组中的特定变量进行消元
ellipsoid	求椭圆体的表面积
elliptic_int	椭圆积分
entries	取表或序列的元素值
erf	误差函数
erfc	同 erf 互补的误差函数及其迭代积分
erfi	erf 的虚部函数（-i*erf）
eulermac	euler-Maclaurin 求和函数
eval	计算
evala	在代数域计算
evalapply	用户可定义的函数计算过程
evalb	计算逻辑表达式
evalc	将复数表达式分解为实部与虚部
evalf	将对象转化为浮点数
evalfint	数值积分
evalhf	利用硬件计算浮点数
evalm	矩阵计算

续表

函 数 名 称	基 本 功 能
evaln	标识符计算
evalr	利用“域”算法来计算表达式的值
evalrC	计算有界区间
exp	指数
expand	展开多项式
expandoff	暂时禁止对多项式的展开
expandon	解除对多项式展开的禁止
extract	提取公因式
factor	因式分解
factors	另一种返回形式的因式分解
fclose	关闭缓冲区的文件
fdiscont	在实数域内查找函数的断点
feof	判断是否已到达文件的末尾
fflush	加速缓冲区文件输出
filepos	设置或返回当前文件指针位置
fixdiv	计算多项式的最大因式
floor	向下取整
fnormal	对浮点数的归一
fopen	打开文件
forget	从记忆表（remember talbe）中删除单元
fprintf	管道/文件格式输出
frac	取小数部分
freeze	固定某表达式中的变量
fremove	删除文件
frontend	将一般表达式转换为有理表达式
fscanf	从管道/文件中读入
fsolve	对等式进行浮点数近似求解
galois	计算不可约多项式的 galois 群
gc	收集系统垃圾
gcd	多项式最大公约数（公因子）
gcdex	多项式的扩展欧几里得代数

续表

函数名称	基本功能
genpoly	利用 Z-adic 展开从整数 n 中获得多项式
getenv	取得某个环境变量的内容
harmonic	调波函数
has	判断是否包含某个子式
hasfun	判断是否包含某个函数
hasoption	从一个列表/集合中提取某个参数
hastype	判断表达式中是否包含某指定类型
heap	优先队列式数据结构
hfarray	硬件浮点数序列
history	保存所有计算过的数值
hypergeom	超集合分布函数
iFFT	快速傅立叶逆变换
icontent	多项式的整数部分
igcd	计算整数的最大公约数
igcdex	整数欧几里得展开
ilcm	计算整数的最小公倍数
ilog	计算以某整数为底的对数并取整
ilog10	计算以 10 为底的对数，并取整
implicitdiff	隐函数求导
indets	寻找某表达式的不确定因子
index	Maple 帮助索引
indices	取表或矩阵的下标
inifcn	检索 Maple 内置数学函数
ininame	检索 Maple 内置变量
initialcondition	设定初始条件
int	定积分或不定积分
intat	某特定点的积分
interface	查询或设置用户界面参数
interp	多项式插值
invfunc	反函数
invztrans	Z 变换的逆变换

续表

函数名称	基本功能
iostatus	检查所有打开文件的状态
iperfpow	判断某整数是否为另一数的整数幂
iquo	整数相除的商
iratrecon	有理式重组
irem	整数相除的余数
iroot	整数的整数阶根式
irreduc	多项式不可约判定
iscont	连续性判定
isdifferentiable	分段函数某点连续性判定
isolate	分离多项式或等式中的变量
ispoly	判断多项式是否含有某一特定阶
isqrfree	门由因式分解
isqrt	判断是否为某整数的平方根
issqr	判断是否为某整数的平方
latex	生成 LaTeX 2e 的格式输出
lattice	简化基矢量
lcm	最小公倍数
lcoeff	求多变量多项式的第一个系数
leadterm	求展开多项式的第一项
length	求出对象所占内存的字节数
lexorder	判断两字符串是否符合字典顺序
lhs	提取等式的左项
limit	求极限
ln	自然对数
lnGAMMA	log-GAMMA 函数
log	一般对数
log10	以 10 为底的对数
lprint	按行打印表达式
map	将某指定过程作用于表达式中的所有项
map2	将指定第一个参数的过程作用于表达式
match	判断两个同形式的多项式是否可能相等

续表

函数名称	基本功能
matrix	生成矩阵
max	求最大值
maximize	计算多项式的极大值
maxnorm	提取多项式的最大系数
member	判断集合或列表中的元素的包含关系
min	求最小值
minimize	计算多项式的极小值
modp	模运算
modp1	单变量多项式模运算
modp2	双变量多项式模运算
modpol	表达式在商域的换算
mods	模运算
msolve	利用 Z mod 方式求解等式
mtaylor	多变量泰勒级数展开
mul	求序列的积
nextprime	求下一个比指定整数大的质数
nops	显示多项式的项数
norm	多项式的范数
normal	标准化（化简）有理式
nprintf	按标准格式显示某一名称
numboccur	计算某多项式中某特定项的出现次数
numer	显示表达式的分子
odetest	检验常微分方程的隐式或显式解
op	从表达式中提取因子
open	打开文件
optimize	优化函数
order	显示序列截断时的阶数
parse	分析一句表达式对应的 MAPLE 处理结果
patmatch	模式匹配
pclose	关闭管道(pipe)
popen	开启管道

续表

函数名称	基本功能
pdesolve	求解偏微分方程
pdetest	验证偏微分方程解
pdsolve	求偏微分方程的解析解
piecewise	定义分段函数
plot	二维绘图函数
plot3d	三维绘图函数
plotsetup	设置绘图参数
pochhammer	一般 pochhammer 函数
pointto	表达式指向的内存地址
poisson	泊松级数展开
polar	转换成极坐标形式
polylog	普通多对数函数
polynom	多项式
powmod	对多项式先乘方，再取模
prem	多项式的伪余数
prevprime	求下一个比指定整数小的质数
primpart	多项式关于某变量的最简系数
print	打印
printf	格式输出
procbody	创建一个过程的特殊表达形式
procmake	创建一个 Maple 过程
product	乘积
proot	多项式的 n 阶根
property	显示设定的属性
protect	设置保护变量
psqrt	多项式的平方根
queue	队列式数据结构
quo	两多项式相除的商
radnormal	标准化含根式的表达式
radsimp	化简含根式的表达式
rand	产生随机数

续表

函数名称	基本功能
randomize	重置随机数发生因子
randpoly	产生随机多项式
rationalize	有理化分母
ratrecon	分式的合并
readbytes	按字节从文件或管道中读取数据
readdata	从文件或管道中读取数值
readline	按行从文件或管道中读取数据
readstat	从输入流中读取一个语句
realroot	分离多项式的各个实根
recipoly	判断一个多项式是否为自反的
rem	两多项式相除的余项
remove	从序列、集合、列表或函数中删除某项
residue	计算某表达式的代数残差
resultant	合并两个多项式的同类项（消元）
rhs	提取等式的右项
root	代数表达式的 n 阶根式
roots	求多项式的解
round	四舍五入
rsolve	求解递推方程
savelib	保存特定参数到 Maple 储存区
scanf	格式输入
searchtext	文本搜索
sec	正割函数
sech	双曲正割函数
select	从集合、序列、列表、表中提取元素
seq	序列
series	泰勒展开
setattribute	设置属性
shake	估算数值区间
showprofile	显示 Maple 对某过程的处理信息
showtime	显示某命令运行占用的系统时间与空间

续表

函数名称	基本功能
sign	判断符号
signum	判断实数或复数的符号
simplify	表达式化简
sin	正弦函数
singular	寻找某表达式的奇点
sinh	双曲正弦函数
smartplot	高级二维绘图函数
smartplot3d	高级三维绘图函数
solve	解方程
solvefor	对方程组中的某特定变量求解
sort	多项式或数值排序
spline	计算自然样条曲线
split	多项式因式分解
sprem	多项式的稀疏伪余数
sprintf	按字符格式输出
sqrfree	多项式因式分解
sqrt	平方根
sscanf	按字符格式输入
ssystem	调用本地系统命令
stack	堆栈数据结构
sturm	某区间内多项式解的个数
sturmseq	多项式的 sturm 序列
subs	替换表达式
subsop	替换表达式中的某一特定项
substring	替换字符串
sum	求和
surd	根式计算
symmdiff	计算两集合的对称差
system	简化的 ssystem 函数
table	建立表
tan	余切函数

续表

函数名称	基本功能
tanh	双曲余切函数
testfloat	比较代浮点数的表达式
thaw	取消 freeze 保护的变量
time	计算某部分的 CPU 占用时间
timelimit	设置某步计算所需最多 CPU 时间
traperror	设定错误终止位置
trigsubs	处理三角恒等式
trunc	向原点靠近取整
type	显示类型
typematch	判断类型是否相同
unames	未指定变量名列表
unapply	向某多项式指定变量
unassign	取消对某变量名的指定
unprotect	取消 protect 设定的保护
userinfo	显示用户信息
value	递推
vector	定义向量
verify	验证结果
whattype	判断表达式类型
with	调用 MAPLE 函数库
writebytes	按字节写文件或管道
writedata	写数据
writeline	按行写文件或管道
writestat	写字符串或表达式到文件或管道中
writeto	将输出结果写到文件
zip	按指定方式合并两列表或矩阵
ztrans	Z 变换

附录 3　Maple 的网络资源

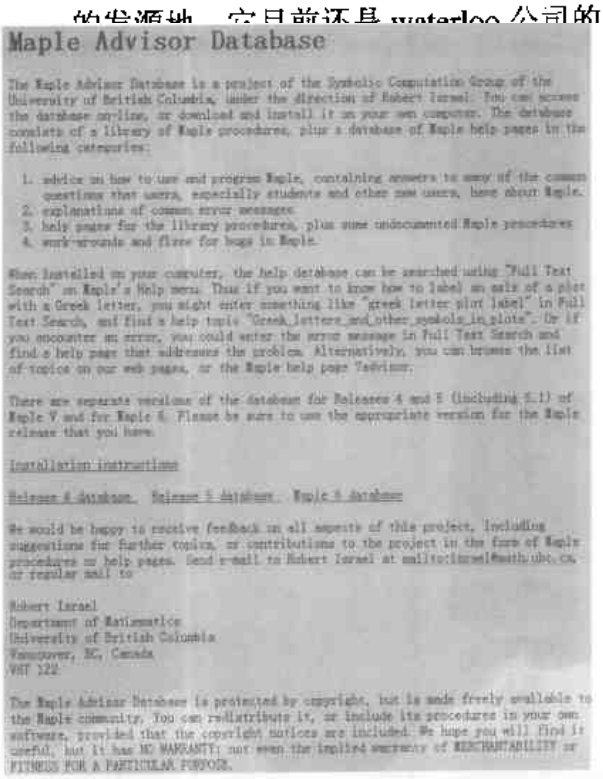
的主页。由此可以连接到其他的同 Maple 产品相关
伴（http://channels.maplesoft.com/），Maple 的应用中
e 的注册站（http://register.maplesoft.com/）等等。

terloo Maple Corporate Site"。网址为：http://www.
laple 最新版本的说明，新函数库下载，以及常见问
，可以让注册用户利用远端服务器处理大数据量的

software.html

Symbolic Computation Group，符号计算组），Maple

的发源地，它目前还是 waterloo 公司的 R&D（研发基地）。本网站提供了各地汇集的 Maple



Maple Advisor Database

The Maple Advisor Database is a project of the Symbolic Computation Group of the University of British Columbia, under the direction of Robert Israel. You can access the database on-line, or download and install it on your own computer. The database consists of a library of Maple procedures, plus a database of Maple help pages in the following categories:

1. advice on how to use and program Maple, containing answers to many of the common questions that users, especially students and other new users, have about Maple.
2. explanations of common error messages
3. help pages for the library procedures, plus some undocumented Maple procedures
4. work-arounds and fixes for bugs in Maple.

When installed on your computer, the help database can be searched using "Full Text Search" on Maple's Help menu. Thus if you want to know how to label an axis of a plot with a Greek letter, you might enter something like "greek letter plot label" in Full Text Search, and find a help topic "Greek_letters_and_other_symbols_in_plots". Or if you encounter an error, you could enter the error message in Full Text Search and find a help page that addresses the problem. Alternatively, you can browse the list of topics on our web pages, or the Maple help page ?advisor.

There are separate versions of the database for Releases 4 and 5 (including 5.1) of Maple V and for Maple 6. Please be sure to use the appropriate version for the Maple release that you have.

Installation instructions

Release 4 database　Release 5 database　Maple 6 database

We would be happy to receive feedback on all aspects of this project, including suggestions for further topics, or contributions to the project in the form of Maple procedures or help pages. Send e-mail to Robert Israel at israel@math.ubc.ca, or regular mail to

Robert Israel
Department of Mathematics
University of British Columbia
Vancouver, BC, Canada
V6T 1Z2

The Maple Advisor Database is protected by copyright, but is made freely available to the Maple community. You can redistribute it, or include its procedures in your own software, provided that the copyright notices are included. We hope you will find it useful, but it has NO WARRANTY; not even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

的一些资料。

advisor/

olumbia：不列颠哥伦比亚大学）提供的 Maple 扩展

信息库。用户可下载并安装至本地计算机，此信息库会同已安装好的 Maple 帮助程序整合。它包括针对初级用户的常见问题解答；对常见错误信息的解释；扩展的帮助信息，包括一些未公开的 Maple 操作步骤；Maple 主程序的补丁；针对 Maple 的不同版本，还有不同的下载程序。

附录 4 参考文献

Maple V by example. Martha L. Abell, James P. Braselton, San Diego, Calif. : Academic Press, 1999

Maple V : programming guide. M.B. Monagan, et al.,New York : Springer, 1998.

Solving problems in scientific computing using Maple and MATLAB. Walter Gander, Jiri Hrebek. Berlin : New York : Springer, 1997.

A guide to Maple. Ernie Kamerich. New York : Springer, 1999.

马春庭等著. 掌握和精通 Maple. 北京：机械工业出版社，2000.

归行洗等著. 线性代数的应用. 上海：上海科学普及出版社，1994.