第一章　*Maple* 基础

1 初识计算机代数系统 Maple

1.1 Maple 简说

1980 年 9 月, 加拿大 **Waterloo** 大学的符号计算机研究小组成立, 开始了符号计算在计算机上实现的研究项目, 数学软件 Maple 是这个项目的产品. 目前, 这仍是一个正在研究的项目.

Maple 的第一个商业版本是 1985 年出版的. 随后几经更新, 到 1992 年, Windows 系统下的 Maple 2 面世后, Maple 被广泛地使用, 得到越来越多的用户. 特别是 1994 年, Maple 3 出版后, 兴起了 Maple 热. 1996 年初, Maple 4 问世, 1998 年初, Maple 5 正式发行. 目前广泛流行的是 Maple 7 以及 2002 年 5 月面市的 Maple 8.

Maple 是一个具有强大符号运算能力、数值计算能力、图形处理能力的交互式计算机代数系统(Computer Algebra System). 它可以借助键盘和显示器代替原来的笔和纸进行各种科学计算、数学推理、猜想的证明以及智能化文字处理.

Maple 这个超强数学工具不仅适合数学家、物理学家、工程师, 还适合化学家、生物学家和社会学家, 总之, 它适合于所有需要科学计算的人.

1.2 Maple 结构

Maple 软件主要由三个部分组成: 用户界面(**Iris**)、代数运算器(**Kernel**)、外部函数库(**External library**). 用户界面和代数运算器是用 C 语言写成的, 只占整个软件的一小部分, 当系统启动时, 即被装入, 主要负责输入命令和算式的初步处理、显示结果、函数图象的显示等. 代数运算器负责输入的编译、基本的代数运算(如有理数运算、初等代数运算等)以及内存的管理. Maple 的大部分数学函数和过程是用 Maple 自身的语言写成的, 存于外部函数库中. 当一个函数被调用时, 在多数情况下, Maple 会自动将该函数的过程调入内存, 一些不常用的函数才需要用户自己调入, 如线性代数包、统计包等, 这使得 Maple 在资源的利用上具有很大的优势, 只有最有用的东西才留驻内存, 这保证了 Maple 可以在较小内存的计算机上正常运行. 用户可以查看 Maple 的非内存函数的源程序, 也可以将自己编的函数、过程加到 Maple 的程序库中, 或建立自己的函数库.

1.3 Maple 输入输出方式

为了满足不同用户的需要, Maple可以更换输入输出格式: 从菜单“Options | Input Display和Out Display下可以选择所需的输入输出格式.

Maple 7 有 2 种输入方式: Maple 语言(Maple Notation)和标准数学记法(Standard Math

Notation). Maple 语言是一种结构良好、方便实用的内建高级语言，它的语法和 Pascal 或 C 有一定程度的相似，但有很大差别. 它支持多种数据操作命令，如函数、序列、集合、列表、数组、表，还包含许多数据操作命令，如类型检验、选择、组合等. 标准数学记法就是我们常用的数学语言.

启动 Maple，会出现新建文档中的“**[>**”提示符，这是 Maple 中可执行块的标志，在“>”后即可输入命令，结束用“**;**”(显示输出结果)或者“**:**”(不显示输出结果). 但是，值得注意的是，并不是说 Maple 的每一行只能执行一句命令，而是在一个完整的可执行块中健入回车之后，Maple 会执行当前执行块中所有命令(可以是若干条命令或者是一段程序). 如果要输入的命令很长，不能在一行输完，可以换行输入，此时换行命令用“**shift+Enter**”组合键，而在最后一行加入结束标志“;”或“:”，也可在非末行尾加符号“\”完成.

Maple 7 有 4 种输出方式: Maple 语言、格式化文本(Character Notation)、固定格式记法(Typeset Notation)、标准数学记法(Standard Math Notation). 通常采用标准数学记法.

Maple 会认识一些输入的变量名称，如希腊字母等. 为了使用方便，现将希腊字母表罗列如下，输入时只需录入相应的英文，要输入大写希腊字母，只需把英文首字母大写:

α	β	γ	δ	ε	ζ	η	θ	ι	κ	λ	μ
alpha	beta	gamma	delta	epsilon	zeta	eta	theta	iota	kappa	lambda	mu
ν	ξ	o	π	ρ	σ	τ	υ	ϕ	χ	ψ	ω
nu	xi	omicron	pi	rho	sigma	tau	upsilon	phi	chi	psi	omega

有时候为了美观或特殊需要，可以采用 Maple 中的函数或程序设计方式控制其输出方式，如下例：

```
> for i to 10 do
printf("i=%+2d and i^(1/2)=%+6.3f", i, eval(sqrt(i)));
od;
```

```
i=+1 and i^(1/2)=+1.000i=+2 and i^(1/2)=+1.414i=+3 and i^(1/2)=+1.732i=+4 and
i^(1/2)=+2.000i=+5 and i^(1/2)=+2.236i=+6 and i^(1/2)=+2.449i=+7 and i^(1/2)=+2.646i=+8 and
i^(1/2)=+2.828i=+9 and i^(1/2)=+3.000i=+10 and i^(1/2)=+3.162
```

+2d 的含义是带符号的十进位整数，域宽为 2. 显然，这种输出方式不是我们想要的，为了得到更美观的输出效果，在语句中加入换行控制符“\n”即可：

```
> for i to 10 do
printf("i=%+2d and i^(1/2)=%+6.3f\n", i, eval(sqrt(i)));
od;
```

```
i=+1 and i^(1/2)=+1.000
i=+2 and i^(1/2)=+1.414
i=+3 and i^(1/2)=+1.732
```

i=+4 and i^(1/2)=+2.000
i=+5 and i^(1/2)=+2.236
i=+6 and i^(1/2)=+2.449
i=+7 and i^(1/2)=+2.646
i=+8 and i^(1/2)=+2.828
i=+9 and i^(1/2)=+3.000
i=+10 and i^(1/2)=+3.162

再看下例：将输入的两个数字用特殊形式打印：

```
> niceP:=proc(x,y)
printf("value of x=%6.4f, value of y=%6.4f",x,y);
end proc;
```

niceP := **proc** (*x*, *y*) printf("value of x=%6.4f, value of y=%6.4f" , *x*, *y*) **end proc**

```
> niceP(2.4,2002.204);
value of x=2.4000, value of y=2002.2040
```

1.4 Maple 联机帮助

学会寻求联机帮助是掌握一个软件的钥匙. Maple有一个非常好的联机帮助系统, 它包含了90%以上命令的使用说明. 要了解Maple的功能可用菜单帮助“Help”, 它给出Maple内容的浏览表, 这是一种树结构的目录表, 跟有…的词条说明其后还有子目录, 点击这样的词条后子目录就会出现(也可以用Tab键和up, down选定). 可以从底栏中看到函数命令全称, 例如, 我们选graphics…, 出现该条的子目录, 从中选 2D…, 再选plot就可得到作函数图象的命令plot的完整帮助信息. 一般帮助信息都有实例, 我们可以将实例中的命令部分拷贝到作业面进行计算、演示, 由此可了解该命令的作用.

在使用过程中, 如果对一个命令把握不准, 可用键盘命令对某个命令进行查询. 例如, 在命令区输入命令“?plot”(或help(plot);), 然后回车将给出plot命令的帮助信息, 或者将鼠标放在选定的要查询的命令的任何位置再点击菜单中的“Help”即可.

2 Maple 的基本运算

2.1 数值计算问题

算术是数学中最古老、最基础和最初等的一个分支, 它研究数的性质及其运算, 主要包括自然数、分数、小数的性质以及他们的加、减、乘、除四则运算. 在应用 Maple 做算术运算时, 只需将 Maple 当作一个“计算器”使用, 所不同的是命令结束时需加“;”或“:”.

在 Maple 中, 主要的算术运算符有“+”(加)、“–”(减)、“*”(乘)、“/”(除)以及“^”(乘方或幂，或记为**), 算术运算符与数字或字母一起组成任意表达式, 但其中“+”、“*”是最基本的运算, 其余运算均可归诸于求和或乘积形式. 算述表达式运算的次序为: 从左到右, 圆括号最先, 幂运算优先, 其次是乘除，最后是加减. 值得注意的是, “^”的表达式只能有两个操作数, 换言之, *a*^*b*^*c* 是错误的, 而“+”或“*”的任意表达式可以有两个或者两个以上的操作数.

Maple 有能力精确计算任意位的整数、有理数或者实数、复数的四则运算, 以及模算术、硬

件浮点数和任意精度的浮点数甚至于矩阵的计算等等. 总之, Maple 可以进行任意数值计算.

但是, 任何软件或程序毕竟只是人们进行科学研究的一种必要的辅助, 即便它有很多优点, 但也有它的局限性, 为了客观地认识数学软件、认识 Maple, 下面通过两个简单例子予以说明.

第一个简单的数值计算实例想说明 Maple 数值计算的答案的正确性:

```
> 3!!!;
```

2601218943565795100204903227081043611191521875016945785727541837850835631156947382240678577958130457082619920575892
2472595366415651620520158737919845877408325291052446903888118841237643411919510455053466586162432719401971139098455
3672727853709934562985558671936977407000370043078375899742067678401696720784628062922903210716166986726054898844551
4257193985499448939594496064045132362140265986193073249369770477606067680670176491669403034819961881455625195592566
9188308255149429475965372748456246288242345265977897377408964665539924359287862125159674832209760295056966999272846
7056374713753301924831358707612541268341586012944756601145542074958995256354306828863463108496565068277155299625679
0845235702552186222358130016700834523443236821935793184701956510729781804354173890560727428048583995919729021726612
2912984205160675790362323376994539641914751755675576953922338030568253085999774416757843528159134613403946049012695
4202883834710136373382448450666009334848444071193129253769465735433737572477223018153403264717753198453734147867432
7048457983786618703257405938924215709695994630557521063203263493209220738320923356309923267504401701760572026010829
2880423356066430898887102973807975780130560495763428386830571906622052911748225105366977566030295740433879834715185
5260280533386635713910104633641976909739743228599421983704697910995630338960467588986579571117656667003915674815311
5943980043625399399731203066490601325311304719028898491856203766669164468791125249193754425845895000311561682974304
6411425380748972817233759553806617198014046779356147936352662656833395097600000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000

上述运算结果在 IBM PC 机(1G, 128M)上计算只需要 0.01 秒, 得到如此复杂的结果(1747 位), 一个自然的问题是: 答案正确吗?

为了回答这个问题, 我们借助于数值分析方法, 由 Stiring 公式

$$n! \approx \sqrt{2\pi n} \cdot n^n \cdot \exp(-n)$$

可得: $720! \approx 2.60091 \times 10^{1746}$, 前三位数字与 Maple 输出结果相同, 且两者结果均为 1747 位. 另外, 在 720!的计算中, 5 的因子的个数为:

$$\left[\frac{720}{5}\right] + \left[\frac{720}{5^2}\right] + \left[\frac{720}{5^3}\right] + \left[\frac{720}{5^4}\right] = 178$$

这些 5 与足够多的 2 相乘将得到 178 个 0, 而 Maple 的输出结果中最后 178 位数为零. 由此, 可以相信 Maple 结果的正确性.

另一个例子则想说明 Maple 计算的局限性:

$$(-8)^{1/3} = ? \qquad (-8)^{2/6} = ?$$

Maple 在处理问题时, 为了避免失根, 从不求算术式的近似值, 分数则化简为既约分数. 因此, 在 Maple 中很容易得到:

$$(-8)^{1/3} = (-8)^{2/6}$$

显然这是错误的. 这一点可以从代数的角度予以分析.

不妨设 $(-8)^{1/3} = x$, 则 $x^3 + 8 = 0$, 即 $(x+2)(x^2 - 2x + 4) = 0$, 显然 $(-8)^{1/3}$ 有 3 个结果, -2 是其实数结果.

另一方面，设$(-8)^{2/6}=x$，则$x^6+(-8)^2=0$，即:

$$(x^3+8)(x^3-8)=(x+2)(x-2)(x^2-2x+4)(x^2+2x+4)=0$$

显然$(-8)^{2/6}$有 6 个结果, -2、2 是其实数结果.

这个简单的例子说明了Maple在数值计算方面绝对不是万能的，其计算结果也不是完全正确的，但是，通过更多的实验可以发现：Maple只可能丢失部分结果，而不会增加或很少给出完全错误的结果(如上例中Maple的浮点数结果皆为 $1.000000000+1.732050807I$). 这一点提醒我们，在利用Maple或其他任何数学软件或应用程序进行科学计算时，必须运用相关数学基础知识校验结果的正确性.

尽管Maple存在缺陷(实际上，任何一个数学软件或程序都存在缺陷)，但无数的事实说明Maple仍然不失为一个具有强大科学计算功能的计算机代数系统. 事实上，Maple同其他数学软件或程序一样只是科学计算的一个辅助工具，数学基础才是数学科学中最重要的.

2.1.1 有理数运算

作为一个符号代数系统, Maple 可以绝对避免算术运算的舍入误差. 与计算器不同, Maple 从来不自作主张把算术式近似成浮点数，而只是把两个有公因数的整数的商作化简处理. 如果要求出两个整数运算的近似值时，只需在任意一个整数后加“.”(或“.0”)，或者利用“**evalf**”命令把表达式转换成浮点形式，默认浮点数位是 10 (即: **Digits:**=10，据此可任意改变浮点数位，如 Digits:=20).

```
> 12!+(7*8^2)-12345/125;
```

$$\frac{11975048731}{25}$$

```
> 123456789/987654321;
```

$$\frac{13717421}{109739369}$$

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

$$.1249999989$$

```
> 10!; 100*100+1000+10+1; (100+100)*100-9;
```

$$3628800$$
$$11011$$
$$19991$$

```
> big_number:=3^(3^3);
```

$$big\_number := 7625597484987$$

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

$$13$$

上述实验中使用了一个变量“big_number”并用“**:=**”对其赋值，与 Pascal 语言一样为一个变量赋值用的是“**:=**”. 而另一个函数“**length**”作用在整数上时是整数的十进制位数即数字的长度. “%”是一个非常有用的简写形式，表示最后一次执行结果，在本例中是上一行输出结果. 再看下面数值计算例子:

1)整数的余(irem)/商(iquo)

命令格式:

irem(m,n);　　＃求 m 除以 n 的余数

irem(m,n,'q');　＃求m除以n的余数，并将商赋给q

iquo(m,n);　　＃求 m 除以 n 的商数

iquo(m,n,'r');　＃求m除以n的商数，并将余数赋给r

其中, m, n 是整数或整数函数，也可以是代数值，此时, irem 保留为未求值.

```
> irem(2002,101,'q'); # 求 2002 除以 101 的余数，将商赋给q
```

83

```
> q; #显示q
```

19

```
> iquo(2002,101,'r'); # 求 2002 除以 101 的商，将余数赋给r
```

19

```
> r; #显示r
```

83

```
> irem(x,3);
```

$$\mathrm{irem}(x, 3)$$

2)素数判别(isprime)

素数判别一直是初等数论的一个难点，也是整数分解问题的基础. Maple 提供的 isprime 命令可以判定一个整数 n 是否为素数. 命令格式: isprime(n);

如果判定 n 可分解，则返回 false，如果返回 true，则 n“很可能”是素数.

```
> isprime(2^(2^4)+1);
```

true

```
> isprime(2^(2^5)+1);
```

false

上述两个例子是一个有趣的数论难题。形如 $F^n = 2^{2^n} + 1$ 的数称为Fermat数，其中的素数称为Fermat素数，显然，F_0=3、F_1=5、F_2=17、F_3=257、F_4=65537 都是素数. Fermat曾经猜想所有的 F_n都是素数，但是Euler在 1732 年证明了F_5=641•6700417 不是素数. 目前，这仍是一个未解决的问题，人们不知道还有没有Fermat素数，更不知道这样的素数是否有无穷多.

3) 确定第 i 个素数(ithprime)

若记第 1 个素数为 2，判断第 i 个素数的命令格式: ithprime(i);

```
> ithprime(2002);
```

17401

```
> ithprime(10000);
```

104729

4) 确定下一个较大(nextprime)/较小(prevprime)素数

当 n 为整数时，判断比 n 稍大或稍小的素数的命令格式为:

nextprime(n);

prevprime(n);

```
> nextprime(2002);
```

2003

```
> prevprime(2002);
```

1999

5) 一组数的最大值(max)/最小值(min)

命令格式: max(x1,x2,…,xn); #求$x_1,x_2,\cdots,x_n$中的最大值

min(x1,x2,…,xn); #求$x_1,x_2,\cdots,x_n$中的最小值

```
> max(1/5,ln(3),9/17,-infinity);
```

$\ln(3)$

```
> min(x+1,x+2,y);
```

$\min(y, x+1)$

6)模运算(mod/modp/mods)

命令格式: e mod m; # 表达式 e 对 m 的整数的模运算

modp(e,m); # e 对正数 m 的模运算

mods(e,m); # e 对 m 负对称数(即 -m)的模运算

`mod`**(e,m);** # 表达式e对m的整数的模运算，与e mod m等价

值得注意的是，要计算 i^n mod m(其中 i 是一整数)，使用这种“明显的”语法是不必要的，因为在计算模 m 之前，指数要先在整数(可能导致一个非常大的整数)上计算. 更适合的是使用惰性运算符“&^”即: i &^n mod m，此时，指数运算将由 mod 运算符智能地处理. 另一方面, mod 运算符的左面优先比其他运算符低，而右面优先高于+和-，但低于*和/.

```
> 2002 mod 101;
```

83

```
> modp(2002,101);
```

83

```
> mods(49,100);
```

49

```
> mods(51,100);
```

-49

```
> 2^101 mod 2002;  # 同 2 &^101 mod 2002;
```

1124

7)随机数生成器(rand)

命令格式:

rand( ); # 随机返回一个 12 位数字的非负整数

rand(a..b); # 调用 rand(a..b)返回一个程序，它在调用时生成一个在范围[a, b]内的随机数

```
> rand();
```

427419669081

```
> myproc:=rand(1..2002):
> myproc();
```

1916

```
> myproc();
```

1204

注意, rand(n)是 rand(0..n-1)的简写形式.

2.1.2 复数运算

复数是 Maple 中的基本数据类型. 虚数单位 *i* 在 Maple 中用 **I** 表示. 在运算中, 数值类型转化成复数类型是自动的, 所有的算术运算符对复数类型均适用. 另外还可以用 **Re( )**、**Im( )**、**conjugate( )**和 **argument( )**等函数分别计算实数的实部、虚部、共轭复数和幅角主值等运算. 试作如下实验:

```
> complex_number:=(1+2*I)*(3+4*I);
```

$$complex\_number := -5 + 10\,I$$

```
> Re(%);Im(%%);conjugate(%%%);argument(complex_number);
```

$$-5$$

$$10$$

$$-5 - 10\,I$$

$$-\arctan(2) + \pi$$

值得注意的是上行命令中均以“;”结束, 因此不能将命令中的2个%或3个%(最多只能用3个%)改为1个%, 因为%表示上一次输出结果, 若上行命令改为“,”结束, 则均可用1个%.

为了在符号表达式中进行复数运算, 可以用函数 **evalc( )**, 函数 **evalc** 把表达式中所有的符号变量都当成实数, 也就是认为所有的复变量都写成 $a + bI$ 的形式, 其中 a、b 都是实变量. 另外还有一些实用命令, 分述如下:

1) 绝对值函数

命令格式: **abs(expr);**

当 expr 为实数时，返回其绝对值，当 expr 为复数时，返回复数的模.

```
> abs(-2002);      #常数的绝对值
```

$$2002$$

```
> abs(1+2*I);     # 复数的模
```

$$\sqrt{5}$$

```
> abs(sqrt(3)*I*u^2*v);    # 复数表达式的绝对值
```

$$\sqrt{3}\,|u^2\,v|$$

```
> abs(2*x-5);     # 函数表达式的绝对值
```

$$|2\,x - 5|$$

2)复数的幅角函数

命令格式: argument(x); # 返回复数 x 的幅角的主值

```
> argument(6+11*I);
```

$$\arctan\left(\frac{11}{6}\right)$$

```
> argument(exp(4*Pi/3*I));
```

$$-\frac{2}{3}\pi$$

3)共轭复数

命令格式:　conjugate(x);　＃返回 x 的共轭复数

```
> conjugate(6+8*I);
```

$$6 - 8\,I$$

```
> conjugate(exp(4*Pi/3*I));
```

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

2.1.3 数的进制转换

数的进制是数值运算中的一个重要问题. 而在 Maple 中数的进制转换非常容易, 使用 convert 命令即可.

命令格式:　convert(expr, form, arg3, ...);

其中, expr 为任意表达式, form 为一名称, arg3, ... 可选项.

下面对其中常用数的转换予以概述. 而 convert 的其它功能将在后叙章节详述.

1)基数之间的转换

命令格式:

convert(n, base, beta);　　#将基数为 10 的数 n 转换为基数为 beta 的数

convert(n, base, alpha, beta); ＃将基数为 alpha 的数字 n 转换为基数为 beta 的数

```
> convert(2003,base,7);  #将 10 进制数 2002 转换为 7 进制数, 结果为: (5561)7
```

$$[1, 6, 5, 5]$$

```
> convert([1,6,5,5],base,7,10);  ＃将 7 进制数 5561 转换为 10 进制数
```

$$[3, 0, 0, 2]$$

```
> convert(2002,base,60);          ＃将十进制数 2002 转换为 60 进制数, 得 33(分钟)22(秒)
```

$$[22, 33]$$

2)转换为二进制形式

命令格式: convert(n, binary);

其功能是将十进制数 n 转换为 2 进制数. 值得注意的是, 数可以是正的, 也可以是负的, 或者是整数, 或者是浮点数, 是浮点数时情况较为复杂.

```
> convert(2002,binary);
```

$$11111010010$$

```
> convert(-1999,binary);
```

$$-11111001111$$

```
> convert(1999.7,binary);
```

$$.1111100111\ \ 10^{11}$$

3)转换为十进制形式

其它数值转换为十进制的命令格式为:

convert(n, decimal, binary);　#将一个 2 进制数 n 转换为 10 进制数

convert(n, decimal, octal);　#将一个 8 进制数 n 转换为 10 进制数

convert(string, decimal, hex);　#将一个 16 进制字符串 string 转换为 10 进制数

```
> convert(11111010010, decimal, binary);
```

2002

> convert(-1234, decimal, octal);

-668

> **convert("2A.C", decimal, hex);**

42.75000000

4) 转换为 16 进制数

将自然数 n 转换为 16 进制数的命令格式为: convert(n, hex);

> convert(2002,hex); convert(1999,hex);

7D2

7CF

5)转换为浮点数

命令格式: convert(expr, float);

注意，convert/float 命令将任意表达式转换为精度为全局变量 Digits 的浮点数，且仅是对 evalf 的调用.

> convert(1999/2002,float);

.9985014985

> convert(Pi,float);

3.141592654

2.2 初等数学

初等数学是数学的基础之一，也是数学中最有魅力的一部分内容. 通过下面的内容我们可以领略 Maple 对初等数学的驾驭能力，也可以通过这些实验对 Maple 产生一些感性认识.

2.2.1 常用函数

作为一个数学工具，基本的数学函数是必不可少的，Maple 中的数学函数很多，现例举一二如下:

指数函数: exp

一般对数: log[a]

自然函数: ln

常用对数: log10

平方根: sqrt

绝对值: abs

三角函数: sin、cos、tan、sec、csc、cot

反三角函数: arcsin、arccos、arctan、arcsec、arccsc、arccot

双曲函数: sinh、cosh、tanh、sech、csch、coth

反双曲函数: arcsinh、arccosh、arctanh、arcsech、arccsch、arccoth

贝赛尔函数: BesselI、BesselJ、BesselK、BesselY

Gamma 函数: GAMMA

误差函数: erf

函数是数学研究与应用的基础之一，现通过一些实验说明 Maple 中的函数的用法及功能.

1) 确定乘积和不确定乘积

命令格式: product(f,k);

product(f,k=m..n);
product(f,k=alpha);
product(f,k=expr);

其中，f—任意表达式，k—乘积指数名称，m,n—整数或任意表达式，alpha—代数数 RootOf, expr—包含 k 的任意表达式.

> product(k^2,k=1..10);　#计算 k^2 关于 1..10 的连乘

$$13168189440000$$

> product(k^2,k);　　#计算 k^2 的不确定乘积

$$\Gamma(k)^2$$

> product(a[k],k=0..5);　　#计算a_i(i=0..5)的连乘

$$a_0\, a_1\, a_2\, a_3\, a_4\, a_5$$

> product(a[k],k=0..n);　　#计算a_i(i=0..n)的连乘

$$\prod_{k=0}^{n} a_k$$

> Product(n+k,k=0..m)=product(n+k,k=0..m);　#计算(n+k)的连乘，并写出其惰性表达式

$$\prod_{k=0}^{m} (n+k) = \frac{\Gamma(n+m+1)}{\Gamma(n)}$$

> product(k,k=RootOf(x^3-2));　　#计算 x^3-2 的三个根的乘积

$$2$$

product命令计算符号乘积，常常用来计算一个公式的确实或不确实的乘积. 如果这个公式不能求值计算, Maple返回Γ 函数. 典型的例子是:

> **product(x+k,k=0..n-1);**

$$\frac{\Gamma(x+n)}{\Gamma(x)}$$

如果求一个有限序列值的乘积而不是计算一个公式，则用 mul 命令. 如:

> **mul(x+k,k=0..3);**

$$x\,(x+1)\,(x+2)\,(x+3)$$

2)指数函数

计算指数函数 exp 关于 x 的表达式的命令格式为: exp(x);

> **exp(1);**

$$\mathbf{e}$$

> **evalf(%);**

$$2.718281828$$

> **exp(1.29+2*I);**

$$-1.511772633 + 3.303283467\,I$$

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

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

3)确定求和与不确定求和 sum

命令格式: sum(f,k);

sum(f,k=m..n);

sum(f,k=alpha);

sum(f,k=expr);

其中，f—任意表达式，k—乘积指数名称，m,n—整数或任意表达式，alpha—代数数 RootOf, expr—不含 k 的表达式.

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

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

```
> Sum(k^3,k=1..n)=sum(k^3,k=1..n);
```

$$\sum_{k=1}^{n} k^3 = \frac{1}{4}(n+1)^4 - \frac{1}{2}(n+1)^3 + \frac{1}{4}(n+1)^2$$

```
> Sum(k^4,k=1..n)=sum(k^4,k=1..n);
```

$$\sum_{k=1}^{n} k^4 = \frac{1}{5}(n+1)^5 - \frac{1}{2}(n+1)^4 + \frac{1}{3}(n+1)^3 - \frac{1}{30}n - \frac{1}{30}$$

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

$$\sum_{k=0}^{\infty} \frac{1}{k!} = \mathbf{e}$$

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

$$\sum_{k=0}^{n} a_k\,x_k$$

```
> Sum(k/(k+1),k)=sum(k/(k+1),k);
```

$$\sum_{k} \frac{k}{k+1} = k - \Psi(k+1)$$

```
> sum(k/(k+1),k=RootOf(x^2-3));
```

$$3$$

sum 函数可计算一个公式的确定和与不确定和，如果 Maple 无法计算封闭形式，则返回未求值的结果. 值得注意的是，在 sum 命令中将 f 和 k 用单引号括起来，可避免过早求值. 这一点在某些情况下是必需的.

```
> Sum('k','k'=0..n)=sum('k','k'=0..n);
```

$$\sum_{k=0}^{n} k = \frac{1}{2}(n+1)^2 - \frac{1}{2}n - \frac{1}{2}$$

如果计算一个有限序列的值，而不是计算一个公式，可用 add 命令．如:

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

$$5050$$

尽管 sum 命令常常用于计算显式求和，但在程序设计中计算一个显式和应该使用 add 命令．

另外，sum 知道各种求和方法，并会对各类发散的求和给出正确的结果，如果要将求和限制为收敛求和，就必须检查显式的收敛性．

3)三角函数/双曲函数

命令格式: sin(x); cos(x); tan(x); cot(x); sec(x); csc(x);

sinh(x); cosh(x); tanh(x); coth(x); sech(x); csch(x);

其中, x 为任意表达式．

值得注意的是三角函数/双曲函数的参数以弧度为单位．Maple 提供了利用常见三角函数/双曲函数恒等式进行化简和展开的程序，也有将其转化为其它函数的命令 convert.

```
> Sin(Pi)=sin(Pi);
```

$$\mathrm{Sin}(\pi) = 0$$

```
> coth(1.9+2.1*I);
```

$$.9775673582 + .03813995737\ I$$

```
> expand(sin(x+y));        #展开表达式
```

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

```
> combine(%);              #合并表达式
```

$$\sin(x+y)$$

```
> convert(sin(7*Pi/60),'radical');
```

$$\left(\frac{1}{8}\sqrt{3} + \frac{1}{8}\right)\sqrt{5-\sqrt{5}} - \frac{1}{16}\sqrt{2}\,(\sqrt{5}+1)\sqrt{3} + \frac{1}{16}\sqrt{2}\,(\sqrt{5}+1)$$

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

$$.3583679496$$

但有趣的是, combine 只对 sin, cos 有效，对 tan, cot 竟无能为力．

4)反三角函数/反双曲函数

命令格式: arcsin(x); arccos(x); arctan(x); arccot(x); arcsec(x); arccsc(x);

arcsinh(x); arccosh(x); arctanh(x); arccoth(x); arcsech(x); arccsch(x);

arctan(y,x);

其中, x, y 为表达式．反三角函数/反双曲函数的参数必须按弧度计算．

算子记法可用于对于反三角函数和反双曲函数．例如, sin@@(-1)求值为 arcsin.

```
> arcsinh(1);
```

$$\ln(1+\sqrt{2})$$

```
> cos(arcsin(x));
```

$$\sqrt{1-x^2}$$

> **arcsin(1.9+2.1*I);**

$$.7048051446 + 1.738617351\ I$$

5)对数函数

命令格式: ln(x);　　#自然对数

log[a](x);　　#一般对数

log10(x);　　#常用对数

一般地，在 ln(x)中要求 x>0. 但对于复数型表达式 x，有:

$$\ln(x) = \ln(\text{abs}(x)) + I * \text{argument}(x) \quad (\text{其中，} -\pi < \text{argument}(x) \le \pi)$$

> **ln(2002.0);**

$$7.601901960$$

> **ln(3+4*I);**

$$\ln(3 + 4\ I)$$

> **evalc(%);**　　# 求出上式的实部、虚部

$$\ln(5) + I \arctan\left(\frac{4}{3}\right)$$

> **log10(1000000);**

$$\frac{\ln(1000000)}{\ln(10)}$$

> **simplify(%);**　　#化简上式

$$6$$

2.2.2 函数的定义

Maple 是一个计算机代数系统，带未知或者已知字母变量的表达式是它的基本数据形式. 一个简单的问题是，既然表达式中可以包含未知变量，那么它是不是函数呢？试看下面一个例子:

> **f(x):=a*x^2+b*x+c;**

$$f(x) := a\ x^2 + b\ x + c$$

可以看出，Maple 接受了这样的赋值语句，但 f(x)是不是一个函数呢？要回答这个问题，一个简单的方法是求函数值:

> **f(x),f(0),f(1/a);**

$$a\ x^2 + b\ x + c,\ f(0),\ f\left(\frac{1}{a}\right)$$

由上述结果可以看出，用赋值方法定义的 f(x)是一个表达式而不是一个函数，因为 f(x)不能把所定义的“自变量”或者“参数”转换成别的变量或表达式. 但从赋值“过程”可以看出，*f*(*x*)虽然也算是一个“函数”，但却是一个没有具体定义的函数:

> **print(f);**

proc () **option** *remember* ; 'procname (args)' **end proc**

事实上，我们所做的赋值运算，只不过是在函数 *f* 的记忆表(**remember table**)中加入了 f(x)在 *x* 上的值，当我们把自变量换作 0 或 1/*a* 时，*f*(x)的记忆表中没有对应的表项，所以输出结果就是抽象的表达式.

在 Maple 中，要真正完成一个函数的定义，需要用算子(也称箭头操作符):

```
> f:=x->a*x^2+b*x+c;
```

$$f := x \to a\,x^2 + b\,x + c$$

```
> f(x),f(0),f(1/a);
```

$$a\,x^2 + b\,x + c,\ c,\ \frac{1}{a} + \frac{b}{a} + c$$

多变量的函数也可以用同样的方法予以定义，只不过要把所有的自变量定成一个序列，并用一个括号“()”将它们括起来(这个括号是必须的，因为括号运算优先于分隔符“,”).

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

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

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

$$5$$

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

$$f := (x, y) \to a\,x\,y\,\mathbf{e}^{(x^2+y^2)}$$

综上所述，箭头操作符定义函数的方式一般为:

一元函数: 参数->函数表达式

多多函数: (参数序列)->函数表达式

无参数函数也许不好理解，但可以用来定义常函数:

```
> E:=()->exp(1);
```

$$E := (\ ) \to \mathbf{e}$$

```
> E();
```

$$\mathbf{e}$$

```
> E(x);
```

$$\mathbf{e}$$

另一个定义函数的命令是unapply,其作用是从一个表达式建立一个算子或函数.

定义一个表达式为expr的关于x的函数f的命令格式为: f:=unapply(expr, x);

定义一个表达式为expr的关于x,y,…的多元函数f的命令格式为: f:=unapply(expr, x, y, …);

```
> f:=unapply(x^4+x^3+x^2+x+1,x);
```

$$f := x \to x^4 + x^3 + x^2 + x + 1$$

```
> f(4);
```

$$341$$

```
> f:=unapply(x*y/(x^2+y^2),x,y);
```

$$f := (x, y) \to \frac{x\,y}{x^2 + y^2}$$

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

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

借助函数 **piecewise** 可以生成简单分段函数：

```
> abs(x)=piecewise(x>0,x,x=0,0,x<0,-x);
```

$$|x| = \begin{cases} x & 0 < x \\ 0 & x = 0 \\ -x & x < 0 \end{cases}$$

清除函数的定义用命令 unassign.

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

$$f(1, 1)$$

除此之外，还可以通过程序设计方式定义函数(参见第 6 章).

定义了一个函数后，就可以使用 **op** 或 **nops** 指令查看有关函数中操作数的信息. nops(expr)返回操作数的个数，函数 **op** 的主要功能是获取表达式的操作数，其命令格式为：

op(expr);

op(i, expr);

op(i .. j, expr);

nops(expr);

如果函数op中的参数i是正整数，则op取出expr里第i个操作数，如果i是负整数，则其结果为 **op(nops(expr)+i+1, expr);** 而i=0 的情形较为复杂，当expr为函数时，op(0, expr)返回函数名，当 **expr**为级数时，返回级数的展开点($x-x_0$)，其它数据类型，op(0, expr)返回expr的类型.

命令 **op(i .. j, expr)**；执行的结果是 expr 的第 i 到第 j 个操作数，i..j 中含负整数时的情形同上.

命令 **op(expr);** 等价于 **op(1..nops(expr), expr)**;

特别地，当op函数中i为列表**[a1, a2, ..., an]**，则op([a1, a2, ..., an], expr); 等价于op(an, op(..., op(a2, op(a1, e))...))**;**

而当 expr 为一般表达式时，nops(expr)命令返回的是表达式的项数，当 expr 是级数时返回级数每一项的系数和指数的总和.

```
> expr:=6+cos(x)+sin(x)*cos(x)^2;
```

$$expr := 6 + \cos(x) + \sin(x)\cos(x)^2$$

```
> op(expr);
```

$$6, \cos(x), \sin(x)\cos(x)^2$$

```
> nops(expr);
```

$$3$$

```
> p:=x^2*y+3*x^3*z+2;
```

$$p := x^2 y + 3 x^3 z + 2$$

```
> op(1,p);
```

$$x^2 y$$

```
> op(1..nops(p),p);
```

$$x^2 y,\ 3\,x^3 z,\ 2$$

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

$$3,\ x^3,\ z$$

```
> u:=[1,4,9];
```

$$u := [1, 4, 9]$$

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

$$list$$

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

$$s := \sin(1) + \cos(1)\,(x-1) - \frac{1}{2}\sin(1)\,(x-1)^2 + O((x-1)^3)$$

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

$$x - 1$$

```
> nops(s);
```

$$8$$

下面一个有趣的例子说明了 Maple 在处理算术运算时的“个性”:

```
> op(x*y*z);
```

$$x,\ y,\ z$$

```
> op(x*y*z+1);
```

$$x\,y\,z,\ 1$$

2.2.3 Maple 中的常量与变量名

为了解决数学问题, 一些常用的数学常数是必要的. Maple 系统中已经存储了一些数学常数在表达式序列 **constants** 中:

```
> constants;
```

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

为了方便使用, 现将上述常数的具体含义列示如下:

常　　数	名 称	近似值
圆周率 π	Pi	3.1415926535
Catalan 常数 $C=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)^2}$	Catalan	0.9159655942
Euler-Mascheroni 常数 $\gamma=\lim_{n\to\infty}\left(\left(\sum_{k=1}^{n}\frac{1}{k}\right)-\ln n\right)$	gamma	0.5772156649

∞	infinity	

需要注意的是，自然对数的底数 e 未作为一个常数出现，但这个常数是存在的，可以通过exp(1)来获取.

在 Maple 中，最简单的变量名是字符串，变量名是由字母、数码或下划线组成的序列，其中第一个字符必须是字母或是下划线. 名字的长度限制是 499 个字符. 在定义变量名时常用连接符“.”将两个字符串连接成一个名. 主要有三种形式: “名.自然数”、“名.字符串”、“名.表达式”.

值得注意的是，在 Maple 中是区分字母大小写的. 在使用变量、常量和函数时应记住这一点. 数学常量 π 用 Pi 表示，而 pi 则仅为符号 π 无任何意义. 如 g, G, new_term, New_Team, x13a, x13A 都是不同的变量名.

在 Maple 中有一些保留字不可以被用作变量名:

by	do	done	elif	else	end	fi	for
from	if	in	local	od	option	options	proc
quit	read	save	stop	then	to	while	D

Maple 中的内部函数如 sin, cos, exp, sqrt, ……等也不可以作变量名.

另外一个值得注意的是在 Maple 中三种类型引号的不同作用:

` `: 界定一个包含特殊字符的符号，是为了输入特殊字符串用的;

' ': 界定一个暂时不求值的表达式;

" ": 界定一个字符串，它不能被赋值.

2.2.4 函数类型转换

函数类型转换是数学应用中一个重要问题，譬如，将三角函数转换成指数函数，双曲函数转换成指数函数，等等. 在 Maple 中，实现函数类型转换的命令是 **convert**. 命令格式:

convert(expr, form); #把数学式 expr 转换成 form 的形式

convert(expr, form, x); #指定变量 x，此时 form 只适于 exp、sin、cos

convert 指令所提供的三角函数、指数与函数的转换共有 exp 等 7 种:

(1) exp: 将三角函数转换成指数

(2) expln: 把数学式转换成指数与对数

(3) expsincos: 分别把三角函数与双曲函数转换成 sin、cos 与指数的形式

(4) ln: 将反三角函数转换成对数

(5) sincos: 将三角函数转换成 sin 与 cos 的形式，而把双曲函数转换成 sinh 与 cosh 的形式

(6) tan: 将三角函数转换成 tan 的形式

(7) trig: 将指数函数转换成三角函数与对数函数

```
> convert(sinh(x),exp);     #将 sinh(x)转换成 exp 类型
```

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

```
> convert(cos(x)*sinh(y),exp);
```

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

```
> convert(cos(x)*sinh(y),exp,y);
```

$$\cos(x)\left(\frac{1}{2}e^{y}-\frac{1}{2}\frac{1}{e^{y}}\right)$$

> **convert(exp(x)*exp(x^(-2)),trig);**

$$(\cosh(x)+\sinh(x))\left(\cosh\left(\frac{1}{x^2}\right)+\sinh\left(\frac{1}{x^2}\right)\right)$$

> **convert(arcsinh(x)*cos(x),expln);**

$$\ln(x+\sqrt{x^2+1})\left(\frac{1}{2}e^{(Ix)}+\frac{\frac{1}{2}}{e^{(Ix)}}\right)$$

> **convert(cot(x)+sinh(x),expsincos);**

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

> **convert(arctanh(x),ln);**

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

convert 在有理式的转换中也起着重要的作用. 在有关多项式运算的过程中, 利用秦九韶算法可以减少多项式求值的计算量. 在 Maple 中, 可以用函数 **convert** 将多项式转换为这种形式, 而 **cost** 则可以获取求值所需的计算量. 注意: cost 命令是一个库函数, 第一次调用时需要使用 **with(codegen)**加载. 例举如下:

> **with(codegen):**

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

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

> **cost(p);**

$$3\ additions\ +9\ multiplications$$

> **convert(p,'horner');** #将展开的表达式转换成嵌套形式

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

> **cost(%);**

$$4\ multiplications\ +3\ additions$$

同样, 把分式化成连分式(**continued fraction**)形式也可以降低求值所需的计算量.

> **(1+x+x^2+x^3)/p;**

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

> **cost(%);**

$$6\ additions\ +\ 12\ multiplications\ +\ divisions$$

> **convert(%%,'confrac',x);**

$$\frac{1}{4}\,\cfrac{1}{x-\frac{1}{4}-\frac{1}{4}\,\cfrac{1}{x-3+\cfrac{14}{x+\frac{29}{7}-\frac{20}{49}\,\cfrac{1}{x-\frac{1}{7}}}}}$$

> **cost(%);**

$$4\ divisions\ +\ 7\ additions$$

在某些场合下(比如求微分、积分时), 把分式化成部分分式(**partial fraction**)也就是几个最简分式的和式的形式也可以简化运算, 但简化程度不及连分数形式.

> **convert(%%, 'parfrac',x);**

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

> **cost(%);**

$$2\ divisions\ +\ 6\ additions\ +\ 9\ multiplications$$

而把分数转换成连分数的方法为:

> **with(numtheory):**
> **cfrac(339/284);**

$$1+\cfrac{1}{5+\cfrac{1}{6+\frac{1}{9}}}$$

2.2.5 函数的映射—map 指令

在符号运算的世界里, 映射指令map可以说是相当重要的一个指令, 它可以把函数或指令映射到这些结构里的元素, 而不破坏整个结构的完整性. 命令格式为:

map(f, expr); ＃将函数f映射到expr的每个操作数

map(f, expr, a); ＃将函数f映射到expr的每个操作数, 并取出a为f的第2个自变量

map(f, expr, a1, a2,…, an); ＃将函数f映射到expr的每个操作数, 并取a1~an为f的第2~n+1个自变量

map2(f, a1, expr, a2, …, an); ＃以a1为第1个自变量, expr的操作数为第2个自变量, a2为第3个自变量…, an为第n+1个自变量来映射函数f

> **map(f,x1+x2+x3+x4,a1,a2,a3,a4);**

$$f(x1, a1, a2, a3, a4) + f(x2, a1, a2, a3, a4) + f(x3, a1, a2, a3, a4) + f(x4, a1, a2, a3, a4)$$

> **f:=x->sqrt(x)+x^2;**

$$f := x \to \sqrt{x} + x^2$$

> map(f,[a,b,c]);

$$[\sqrt{a} + a^2, \sqrt{b} + b^2, \sqrt{c} + c^2]$$

```
> map(h, [a,b,c],x,y);
```

$$[\mathrm{h}(a, x, y), \mathrm{h}(b, x, y), \mathrm{h}(c, x, y)]$$

```
> map(convert,[arcsinh(x/2),arccosh(x/2)],ln);
```

$$\left[\ln\left(\frac{1}{2}x+\frac{1}{2}\sqrt{x^2+4}\right), \ln\left(\frac{1}{2}x+\frac{1}{4}\sqrt{2\,x-4}\,\sqrt{2\,x+4}\right)\right]$$

```
> map(x->convert(x,exp),[sin(x),cos(x)]);
```

$$\left[\frac{-1}{2}I\left(\mathbf{e}^{(I\,x)}-\frac{1}{\mathbf{e}^{(I\,x)}}\right), \frac{1}{2}\mathbf{e}^{(I\,x)}+\frac{\frac{1}{2}}{\mathbf{e}^{(I\,x)}}\right]$$

上式的映射关系可通过下式理解：

```
> [convert(sin(x),exp),convert(cos(x),exp)];
```

$$\left[\frac{-1}{2}I\left(\mathbf{e}^{(I\,x)}-\frac{1}{\mathbf{e}^{(I\,x)}}\right), \frac{1}{2}\mathbf{e}^{(I\,x)}+\frac{\frac{1}{2}}{\mathbf{e}^{(I\,x)}}\right]$$

```
> restart:
  map2(f,a1,x1+x2+x3+x4,a2,a3,a4);
```

$$\mathrm{f}(a1, x1, a2, a3, a4)+\mathrm{f}(a1, x2, a2, a3, a4)+\mathrm{f}(a1, x3, a2, a3, a4)+\mathrm{f}(a1, x4, a2, a3, a4)$$

```
> map2(max,k,[a,b,c,d]);
```

$$[\max(a, k), \max(b, k), \max(c, k), \max(k, d)]$$

再看下面示例:

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

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

```
> nops(L);
```

$$10$$

```
> sqr:=(x)->x^2;
```

$$sqr := x \to x^2$$

```
> map(sqr,L);
```

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

```
> map((x)->x+1,L);
```

$$[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$$

```
> map(f,L);
```

$$[\mathrm{f}(1), \mathrm{f}(2), \mathrm{f}(3), \mathrm{f}(4), \mathrm{f}(5), \mathrm{f}(6), \mathrm{f}(7), \mathrm{f}(8), \mathrm{f}(9), \mathrm{f}(10)]$$

```
> map(f,{a,b,c});
```

$$\{\mathrm{f}(a), \mathrm{f}(b), \mathrm{f}(c)\}$$

```
> map(sqr,x+y*z);
```

$$x^2 + y^2\, z^2$$

```
> M:=linalg[matrix](3,3,(i,j)->i+j);
```

$$M := \begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{bmatrix}$$

```
> map((x)->1/x,M);
```

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \end{bmatrix}$$

3 求 值

3.1 赋值

在 Maple 中，不需要申明变量的类型，甚至在使用变量前不需要将它赋值，这是 Maple 与其它高级程序设计语言不同的一点，也正是 Maple 符号演算的魅力所在，这个特性是由 Maple 与众不同的赋值方法决定的. 为了理解其赋值机制，先看下面的例子.

```
> p:=9*x^3-37*x^2+47*x-19;
```

$$p := 9\, x^3 - 37\, x^2 + 47\, x - 19$$

```
> roots(p);
```

$$\left[[1, 2], \left[\frac{19}{9}, 1 \right] \right]$$

```
> subs(x=19/9,p);
```

$$0$$

在这个例子中，第一条语句是一个赋值语句，它的作用是把变量p和多项式$9x^3$-$37x^2$+$47x$-19相关联，在此之后，每当Maple遇到变量p时就取与之唯一关联的“值”，比如随后的语句roots(p)就被理解为roots($9x^3$-$37x^2$+$47x$-19)了，而变量x还没被赋值，只有符号“x”，通过subs语句得到将p所关联的多项式中的所有x都替换成19/9后的结果，这正是我们在数学中常用的方法—变量替换，但此时无论x还是p的值仍未改变，通过“> `x; p;`”这样的简单语句即可验证.

3.2 变量代换

在表达式化简中，变量代换是一个得力工具. 我们可以利用函数 subs 根据自己的意愿进行变量代换，最简单的调用这个函数的形式是这样的:

subs (*var = repacedment, expression*);

调用的结果是将表达式 expression 中所有变量 var 出现的地方替换成 replacement.

```
> f:=x^2+exp(x^3)-8;
```

$$f := x^2 + \mathbf{e}^{(x^3)} - 8$$

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

$$-7 + \mathbf{e}$$

```
> subs(x=0,cos(x)*(sin(x)+x^2+5));
```

$$\cos(0)\,(\sin(0) + 5)$$

由此可见，变量替换只得到替换后的结果，而不改变表达式的内容，而且 Maple 只对替换的结果进行化简而不求值计算，如果需要计算，必须调用求值函数 **evalf**. 如:

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

$$5.$$

变量替换函数 subs 也可以进行多重的变量替换，以两重代换为例:

subs (*var*1 = *repacedment*1, *var*2 = *repacedment*2, *expression*)

调用的结果和按从左到右的顺序连续两次调用是一样的，也就是先将 expression 中的 var1 替换成 replacement1，再将其结果中的 var2 替换成 replacement2，把这种替换称作顺序替换；与此相对，还可以进行同步替换，即同时将 expression 中的 var1 替换成 replacement1，而 var2 替换成 replacement2. 同步替换的调用形式为:

subs ({*var*1 = *repacedment*1, *var*2 = *repacedment*2 }, *expression*)

下面通过例子说明这几种形式的替换.

```
> subs(x=y,y=z,x^2*y);                    (顺序替换)
```

$$z^3$$

```
> subs({x=y,y=z},x^2*y);                  (同步替换)
```

$$y^2 z$$

```
> subs((a=b,b=c,c=a),a+2*b+3*c);          (顺序替换)
```

$$6\,a$$

```
> subs({a=b,b=c,c=a},a+2*b+3*c);          (轮　换)
```

$$b + 2\,c + 3\,a$$

```
> subs({p=q,q=p},f(p,q));                 (互　换)
```

$$\mathrm{f}(q, p)$$

3.3 假设机制

Maple 是一种计算机代数语言，显然，很多人会尝试用 Maple(或其他计算机代数语言)解决分析问题. 但由于分析问题与处理问题的考虑方法不同，使得问题的解决存在某些困难. 例如考虑方程 $(k^4+k^2+1)x = k^4+k^2+1$ 的解. 如果 k 是实数，结果显然是 x=1，但如果 k 是 $k^4+k^2+1 \neq 0$ 的复根，为了保证解 x=1 的正确性，必需添加附带条件：也就是当 $k^4+k^2+1 \neq 0$ 时 x=1. 这是一个对结果进行正确分析的例子. 然而从代数的角度考虑这个问题

时就会把 k 当作不定元，此时 k 没有值，从方程两端去除 k 的多项式是合法的，只要这个多项式不是零多项式即可(这一点是可以保证的，因为其所有系数不全为 0). 在此情况下 x=1 就不需要任何附加条件. 计算机代数系统经常采用这种分析的观点.

在 Maple 中，采用分析观点解决这类带有一定附加条件的实用工具是函数 assume，其命令格式为: assume(x, prop);

函数 assume 界定了变量与变量之间的关系式属性. assume 最普遍的用法是 assume(a>0)，该语句假设符号 a 为一个正实常数；若假定某一符号 c 为常数，使用命令 assume(c,constant)；另一方面，assume 可以带多对参数或多个关系式. 当给定多个参数时，所有假定均同时生效. 例如，要定义 a<b<c，可以用 assume(a<b, b<c)；同样地，要定义 0<x<1，可以用 assume(0<x,x<1). 当 assume 对 x 作出假定时，以前所有对 x 的假定都将被删除. 这就允许在 Maple 中先写“assume(x>0);”后再写“assume(x<0);”也不会产生矛盾.

```
> Int(exp(-s*t),t=0..infinity);
```

$$\int_0^{\infty} \mathbf{e}^{(-s\,t)}\,dt$$

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

```
Definite integration: Can't determine if the integral is convergent.
Need to know the sign of --> s
Will now try indefinite integration and then take limits.
```

$$\lim_{t \to \infty} -\frac{\mathbf{e}^{(-s\,t)} - 1}{s}$$

```
> assume(s>0);
> Int(exp(-s*t),t=0..infinity);
```

$$\int_0^{\infty} \mathbf{e}^{(-s\,t)}\,dt$$

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

$$\frac{1}{s}$$

3.4 求值规则

在多数情况下，Maple 的求值规则设计为做用户期望的事情，但要做到这一点很困难，因为不同的人在相同的情形下会有不同的期望. 在大多数情况下，全局变量被完全求值，局部变量被一层求值. 而由符号' '界定一个暂时不求值的表达式，单步求值仅去掉引号，不作计算，这也是允许取消指定名字或清除变量的原因. 如下例:

```
> x:=y;
```

$$x := y$$

```
> y:=z;
```

$$y := z$$

```
> z:=3;
```

$$z := 3$$

```
> x;
```

$$3$$

```
> y;
```

$$3$$

```
> x:='x';
```

$$x := x$$

```
> x;
```

$$x$$

```
> y;
```

$$3$$

对于不同的问题, Maple 设计了不同的求值命令. 现分述如下:

1) 对表达式求值

命令格式: eval(e, x=a); ＃求表达式 e 在 x=a 处的值

eval(e, eqns); ＃对方程或方程组 eqns 求值

eval(e); ＃表达式 e 求值到上面两层

eval(x,n); ＃给出求值名称的第 n 层求值

```
> p:=x^5+x^4+x^3+x^2+x+73;
```

$$p := x^5 + x^4 + x^3 + x^2 + x + 73$$

```
> eval(p,x=7);
```

$$19680$$

```
> P:=exp(y)+x*y+exp(x);
```

$$P := \mathbf{e}^y + x\,y + \mathbf{e}^x$$

```
> eval(P,[x=2,y=3]);
```

$$\mathbf{e}^3 + 6 + \mathbf{e}^2$$

当表达式在异常点处求值时, eval 会给一个错误消息. 如下:

```
> eval(sin(x)/x,x=0);
Error, numeric exception: division by zero
```

下面再看使用 eval 进行全层求值或者对名称几层求值的示例:

```
> a:=b: b:=c: c:=x+1:
> a;                  #默认的全层递归求值
```

$$x + 1$$

```
> eval(a);            ＃强制全层递归求值
```

$$x + 1$$

```
> eval(a,1);          ＃对 a 一层求值
```

$$b$$

```
> eval(a,2);          ＃对 a 二层求值
```

$$c$$

> **eval(a,3);** **＃对 a 三层求值**

$$x+1$$

> **eval(a,4);** **＃对 a 四层求值**

$$x+1$$

2) 在代数数(或者函数)域求值

命令格式: evala(expr); ＃ 对表达式或者未求值函数求值

evala(expr,opts); ＃求值时可加选项(opts)

所谓代数数(Algebraic number)就是整系数单变量多项式的根，其范围比有理数大，真包含于实数域，也就是说任意实数都是整系数多项式的根(如 π 就不是任何整系数多项式的根). 另一方面，代数数也不是都可以表示成为根式的，如多项式 x^5+x+1 的根就不能表示成为根式的形式.

代数数的计算，算法复杂，而且相当费时. 在 Maple 中，代数数用函数 **RootOf**()来表示. 如 $\sqrt{3}$ 作为一个代数数，可以表示为:

> **alpha:=RootOf(x^2-3,x);**

$$\alpha := \text{RootOf}\left(\_Z^2-3\right)$$

> **simplify(alpha^2);**

$$3$$

在 Maple 内部，代数数 α 不再表示为根式，而在化简时，仅仅利用到 $\alpha^2=3$ 这样的事实. 这里，Maple 用到一个内部变量_Z. 再看下面一个例子，其中 alias 是缩写的定义函数，而参数 lenstra 指 lenstra 椭圆曲线方法：

> **alias(alpha=RootOf(x^2-2)):**

> **evala(factor(x^2-2,alpha),lenstra);**

$$(x+\alpha)(x-\alpha)$$

> **evala(quo(x^2-x+3,x-alpha,x,'r'));**

$$-1+\alpha+x$$

> **r;**

$$3-\alpha+\alpha^2$$

> **simplify(%);**

$$5-\alpha$$

3) 在复数域上符号求值

操纵复数型表达式并将其分离给出expr的实部和虚部的函数为evalc，命令格式为:

evalc(expr);

evalc假定所有变量表示数值，且实数变量的函数是实数类型. 其输出规范形式为: expr1+I*expr2.

```
> evalc(sin(6+8*I));
```

$$\sin(6)\cosh(8)+I\cos(6)\sinh(8)$$

```
> evalc(f(exp(alpha+x*I)));
```

$$f(\mathbf{e}^{\alpha}\cos(x)+I\,\mathbf{e}^{\alpha}\sin(x))$$

```
> evalc(abs(x+y*I)=cos(u(x)+I*v(y)));
```

$$\sqrt{x^2+y^2}=\cos(u(x))\cosh(v(y))-I\sin(u(x))\sinh(v(y))$$

4) 使用浮点算法求值

浮点算法是数值计算的一种基本方法，在任何情况下均可以对表达式expr使用evalf命令计算精度为n的浮点数(n=Digits)，如果n缺省，则取系统默认值，命令格式为: **evalf(expr, n);**

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

3.1415926535897932384626433832795028841971693993751

```
> evalf(sin(3+4*I));
```

$$3.853738038-27.01681326\,I$$

```
> evalf(int(sin(x)/x,x=0..1),20);
```

.94608307036718301494

5) 对惰性函数求值

把只用表达式表示而暂不求值的函数称为惰性函数，除了第一个字母大写外，Maple中的惰性函数和活性函数的名字是相同的. 惰性函数调用的典型用法是预防对问题的符号求值，这样可以节省对输入进行符号处理的时间，而value函数强制对其求值. 对任意代数表达式f求值的命令格式为: **value(f);**

```
> F:=Int(exp(x),x);
```

$$F:=\int \mathbf{e}^{x}\,dx$$

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

$$\mathbf{e}^{x}$$

```
> f:=Limit(sin(x)/x,x=0);
```

$$f:=\lim_{x\to 0}\frac{\sin(x)}{x}$$

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

$$1$$

另外，将惰性函数的大写字母改为小写字母亦即可求值. 如下例:

```
> Limit(sin(x)/x,x=0)=limit(sin(x)/x,x=0);
```

$$\lim_{x\to 0}\frac{\sin(x)}{x}=1$$

4 数据结构

Maple 中有许多内建的与 FORTRAN、C 或 Pascal 不同的数据结构. 主要的数据结构有序列

(sequence)、列表(list)、集合(set)、代数数(algebraic number)、未求值或惰性函数调用、表(table)、级数(series)、串(string)、索引名(index)、关系(relation)、过程体(process)以及整数(integer)、分数(fraction)、浮点数(float)、复数(complex number)等数据结构，而矩阵(matrix)在 Maple 中表示为阵列，是一种特殊的表.

4.1 数据类型查询

在 Maple 中，用 **whattype** 指令来查询某个变量的数据类型或特定类型，命令格式为:

whattype(expr) # 查询 expr 的数据类型

type(expr, t) # 查询 expr 是否为 t 类型，若是则返回 true，否则返回 false

```
> whattype(12);
```

integer

```
> whattype(Pi);
```

symbol

```
> type(1.1,fraction);
```

false

```
> whattype(1.1);
```

float

4.2 序列，列表和集合

4.2.1 序列

所谓序列(Sequence)，就是一组用逗号隔开的表达式列. 如:

```
> s:=1,4,9,16,25;
```

$s := 1, 4, 9, 16, 25$

```
> t:=sin,com,tan,cot;
```

$t := \sin, com, \tan, \cot$

一个序列也可以由若干个序列复合而成，如:

```
> s:=1,(4,9,16),25;
```

$s := 1, 4, 9, 16, 25$

```
> s,s;
```

$1, 4, 9, 16, 25, 1, 4, 9, 16, 25$

而符号 NULL 表示一个空序列. 序列有很多用途，如构成列表、集合等. 事实上，有些函数命令也是由序列构成. 例如:

```
> max(s);
```

25

```
> min(s,0,s);
```

0

值得注意的是, op 和 nops 函数命令不适用于序列，如 op(s)或 nops(s)都是错误的，如果要使用 op(s)或 nops(s)前应先把序列 s 置于列表中.

```
> s:=1, 2, abc, x^2+1, `hi world`, Pi, x -> x^2, 1/2, 1;
```

$$s := 1, 2, abc, x^2 + 1, hi\ world, \pi, x \to x^2, \frac{1}{2}, 1$$

```
> op(s);
```
```
Error, wrong number (or type) of parameters in function op
```
```
> nops(s);
```
```
Error, wrong number (or type) of parameters in function nops
```
```
> op([s]);
```

$$1, 2, abc, x^2 + 1, hi\ world, \pi, x \to x^2, \frac{1}{2}, 1$$

```
> nops([stuff]);
```

$$9$$

函数 seq 是最有用的生成序列的命令，通常用于写出具有一定规律的序列的通项，命令格式为:

seq(f(i), i=m..n); # 生成序列 f(m), f(m+1), …, f(n) (m,n 为任意有理数)

seq(f(i), i=expr); # 生成一个 f 映射 expr 操作数的序列

seq(f(op(i,expr)), i=1..nops(expr)); # 生成 nops(expr)个元素组成的序列

```
> seq(i^2,i=1..10);
```

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

```
> seq(ithprime(i),i=1..20);
```

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71$$

```
> seq(i^3,i=x+y+z);
```

$$x^3, y^3, z^3$$

```
> seq(D(f),f=[sin,cos,tan,cot]);
```

$$\cos, -\sin, 1 + \tan^2, -1 - \cot^2$$

```
> seq(f(op(i,x1+x2+x3+x4)),i=1..nops(x1+x2+x3+x4));
```

$$f(x1), f(x2), f(x3), f(x4)$$

获得一个序列中的特定元素选用操作符[ ], 如:

```
> seq(ithprime(i),i=1..20);
```

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71$$

```
> %[6],%[17];
```

$$13, 59$$

4.2.2 列表

列表(list), 就是把对象(元素)放在一起的一种数据结构，一般地，用方括号[ ]表示列表. 如下例:

```
> l:=[x,1,1-z,x];
```

$$l := [x, 1, 1 - z, x]$$

```
> whattype(%);
```

$$list$$

空列表定义为[ ].

但下述两个列表是不一样的，因为对于列表而言，次序是重要的:

```
> L:=[1,2,3,4];
```

$$L := [1, 2, 3, 4]$$

```
> M:=[2,3,4,1];
```

$$M := [2, 3, 4, 1]$$

4.2.3 集合

集合(set)也是把对象(元素)放在一起的数据结构，与列表不同的是集合中不可以有相同的元素(如果有，Maple 也会自动将其当作同一个元素)，另外，集合中的元素不管次序. 一般地，用花括号表示集合.

```
> s:={x,1,1-z,x};
```

$$s := \{1, x, 1 - z\}$$

```
> whattype(%);
```

$$set$$

空集定义为{ }.

函数 nop 返回列表或集合的元素数，而 op 则可返回其第 I 个元素.

```
> op(1,s);
```

$$1$$

```
> s[1];
```

$$1$$

```
> op(1..3,s);
```

$$1, x, 1 - z$$

```
> s[1..3];
```

$$\{1, x, 1 - z\}$$

函数 member 可以判定元素是否属于一个列表或集合，如果属于，返回 true，否则返回 false.

```
> member(1+x,s);
```

$$false$$

可以通过下述方法在列表中增减元素:

```
> t:=[op(s),x];
```

$$t := [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, x]$$

```
> u:=[s[1..5],s[7..nops(s)]];
```

$$u := [[1, 4, 9, 16, 25], [49, 64, 81, 100]]$$

Maple 中集合的基本运算有交(intersect)、并(union)、差(minus):

```
> A:={seq(i^3,i=1..10)};B:={seq(i^2,i=1..10)};
```

$$A := \{1, 8, 27, 64, 125, 216, 343, 512, 729, 1000\}$$

$$B := \{1, 4, 9, 16, 25, 36, 49, 64, 81, 100\}$$

```
> A intersect B;
```

$$\{1, 64\}$$

```
> A union B;
```

$$\{1, 4, 8, 9, 16, 25, 27, 36, 49, 64, 81, 100, 125, 216, 343, 512, 729, 1000\}$$

```
> A minus B;
```

$$\{8, 27, 125, 216, 343, 512, 729, 1000\}$$

4.3 数组和表

在 Maple 中，数组(**array**)由命令 **array** 产生，其下标变量(**index**)可以自由指定．下标由 1 开始的一维数组称为向量(**vector**)，二维以上的数组称为矩阵(**matrix**)．数组的元素按顺序排列，任意存取一数组的元素要比列表或序列快的多．区分一个数据结构是数组还是列表要用“type”命令．

表(table)在建立时使用圆括号，变量能对一个表赋值，但一个在存取在算子中的未赋值变量会被自动地假定是表，表的索引可以成为任意 Maple 表达式．表中元素的次序不是固定的．

```
> A:=array(1..4);
```

$$A := \text{array}(1 .. 4, [\ ])$$

```
> for i from 1 to 4 do A[i]:=i: od:
> eval(A);
```

$$[1, 2, 3, 4]$$

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

$$true$$

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

$$false$$

```
> T:=table();
```

$$T := \text{table}([])$$

```
> T[1]:= 1;
```

$$T_1 := 1$$

```
> T[5]:= 5;
```

$$T_5 := 5$$

```
> T[3]:= 3;
```

$$T_3 := 3$$

```
> T[sam]:=sally;
```

$$T_{sam} := sally$$

```
> T[Pi]:=exp(1);
```

$$T_{\pi} := \mathbf{e}$$

```
> x:='x';
```

$$x := x$$

```
> T[(1+x+x^3)*sin(x)] := 0;
```

$$T_{(1+x+x^3)\sin(x)} := 0$$

```
> eval(T);
```

$$\text{table}([1 = 1, \pi = \mathbf{e}, 3 = 3, 5 = 5, (1 + x + x^3)\sin(x) = 0, sam = sally])$$

```
> T[3]:='T[3]';
```

$$T_3 := T_3$$

```
> eval(T);
```

$$\text{table}([1 = 1, \pi = \mathbf{e}, 5 = 5, (1 + x + x^3)\sin(x) = 0, sam = sally])$$

4.4 其他数据结构

串在Maple中是很重要的，他们主要用于取名字和显示信息. 一个Maple的串可以作为变量名，它们中的大多数是简单的、不需要加引号的串，但是如果变量名中包含/. 例如“diff/T”则必须把变量名用引号括起来.

索引名是像 Database[1,2,drawer]或 A[3]这样的对象，在使用索引前不需要直接建立表，如果不得不做，Maple 会自动建立表. 索引名通常被用于矩阵和向量. 为了保证 Maple 建立表的正确次序，建议在赋值前直接建立.

```
> x:=T[3];
```

$$x := T_3$$

```
> eval(T);
```

$$T$$

```
> T[5]:=y;
```

$$T_5 := y$$

```
> eval(T);
```

$$\text{table}([5 = y])$$

由此可见, Maple 并不直接建立 T 的表，直到给 T[5]赋了值.

数值数据结构(整数、分数、有理数、浮点数、硬件浮点数和复数等)在它们的使用中是大量

透明的. 浮点数是有传染性的, 这意味着如果数值结构中有一个是浮点数, 则整个结构自动转换为浮点数.

4.5 数据类型转换和合并

convert 是一个功能强大的类型转换函数, 它可以实现列表和数组的类型转换:

```
> L:=[1,2,3,4];
```

$$L := [1, 2, 3, 4]$$

```
> type(L,list);
```

$$true$$

```
> A:=convert(L,array);
```

$$A := [1, 2, 3, 4]$$

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

$$false$$

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

$$true$$

另一个有用的函数 zip 则可把两个列表或向量合并:

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

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

```
> Sqr:=(x)->x^2;
```

$$Sqr := x \to x^2$$

```
> M:=map(sqr,L);
```

$$M := [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]$$

```
> LM:=zip((x,y)->[x,y],L,M);
```

$$LM := [[1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81], [10, 100]]$$

```
> map(op,LM);
```

$$[1, 1, 2, 4, 3, 9, 4, 16, 5, 25, 6, 36, 7, 49, 8, 64, 9, 81, 10, 100]$$

5 Maple 高级输入与输出操作

Maple 提供了良好的接口来编辑与计算数学式. 许多时候, 我们可能需要把 **Maple** 的运算结果输出到一个文件中, 或者在一个文本编辑器里先编好一个较大的 **Maple** 程序, 再将它加载到 **Maple** 的环境里.

5.1 写入文件

5.1.1 将数值数据写入到一个文件

如果 Maple 的计算结果是一长串的数值串行或数组, 而想把它写到一个文件时, 用 writedata 命令.

若 Maple 的计算结果 data 为集合、矩阵、列表、向量等形式时，将其写入名为 filename 的文件时命令格式为: writedata("filename", data);

```
> with(linalg):
> M:=matrix(3,3,[1,2,3,4,5,6,7,8,9]);
```

$$M := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> writedata("e:\\filename.txt",M);
```

而将结果附加在一个已存在的文件后时，使用命令: writedata[APPEND]("filename", data);

```
> W:=matrix(2,2,[1,2,3,4]);
```

$$W := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> writedata[APPEND]("e:\\filename.txt",W);
```

需要注意的是，这里的 APPEND 是必需的，否则 W 结果将会覆盖 M 结果.

另外，若想将结果显示在屏幕上时，用命令: writedata('terminal', data);

```
> writedata[APPEND]("e:\\filename.txt",W);
> writedata('terminal',M);
1           2           3
4           5           6
7           8           9
```

5.1.2 将 Maple 语句写入一个文件

如果所要写入文件的是表达式、函数的定义或者是一个完整的程序，则使用命令 save，写入一个或多个语句的命令格式分别如下:

save name, "filename";

save name1, name2, …, "filename";

若 filename 的扩展名为.m，则 Maple 会以内定的格式储存，若扩展名为.txt，则以纯文本文件储存. 以内定的格式储存的文件作纯文本编辑器无法读取，但在大多数情况下，它会比纯文本文件的加载速度更快，且文件容量小.

```
> myfunc:=(k,n)->sum(x^k/k!,x=1..n);
```

$$myfunc := (k, n) \rightarrow \sum_{x=1}^{n} \frac{x^k}{k!}$$

```
> myresult:=myfunc(6,8);
```

$$myresult := \frac{37247}{60}$$

```
> save myfunc,myresult,"e:\\test.m";
```

调用已存 **m** 文件用命令 **read**. 试看下述实验:

```
> restart:
> myfunc(6,8);
```

$$myfunc(6, 8)$$

```
> read "e:\\test.m";
> myfunc(6,8);
```

$$\frac{37247}{60}$$

```
> myresult;
```

$$\frac{37247}{60}$$

而存为 txt 文件时则将整个语句存为一个文件:

```
> save myfunc,myresult,"e:\\test.txt";
> restart: read"e:\\test.txt";
```

$$myfunc := (k, n) \rightarrow \sum_{x=1}^{n} \frac{x^k}{k!}$$

$$myresult := \frac{37247}{60}$$

5.2 读取文件

在 **Maple** 里最常用的两个读取文件的命令，一个是读取数值数据，另一个是是读取 **Maple** 的指令.

5.2.1 读取数值数据

如果想把大量的数据导入 Maple 里进行进一步的运算或者要运用大量的实验数据在 Maple 环境绘图时，可以用 readdata( )命令完成.

从 filename 文件里读取 n 行数据时使用命令: readdata(**"filename",n**);

以指定的格式读取数据时使用命令: readdata(**"filename",[tyep1,type2,…]**);

```
> readdata("e:\\filename.txt",3);
```

[[1., 2., 3.], [4., 5., 6.], [7., 8., 9.]]

读取 filename 的前三列，第一列为整数形式，第二、三列为浮点数形式:

```
> readdata("e:\\filename.txt",[integer,float,float]);
```

[[1, 2., 3.], [4, 5., 6.], [7, 8., 9.]]

下面再看一个运用大量的实验数据在 Maple 环境绘图的实验:

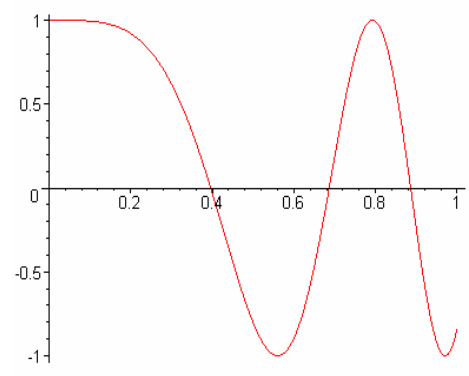
```
> mypts:=[seq([x/1000,cos(x^2/100000)],x=1..1000)]:
> writedata("e:\\data.txt",evalf(mypts));
> dots:=readdata("e:\\data.txt",100):
> nops(dots);
```

1000

```
> dots[1..4];
```

[[.001, 1.000000000], [.002, .9999999992], [.003, .999999996], [.004, .9999999872]]

```
> plot(dots,style=line);
```



5.2.2 读取 Maple 的指令

在编写程序时，在普通软件中先编好程序再将其读入 Maple 环境中常常比直接在 Maple 中编写更为方便. 如果要将程序代码或 Maple 指令加载用 read 命令:

read **"filename"**;

如下例:

```
> reatart:
> myfunc:=(a::list)->add(i,i=a);
```

$$myfunc := a::list \to \mathrm{add}(i, i = a)$$

```
> avg:=(a::list)->myfunc(a)/nops(a);
```

$$avg := a::list \to \frac{\mathrm{myfunc}(a)}{\mathrm{nops}(a)}$$

```
> save myfunc,avg,"e:\\function.m";
> restart:
> read "e:\\function.m";
> myfunc([1,2,3,4,5,6,7,8,9]);
```

$$45$$

```
> avg([1,2,3,4,5,6,7,8,9]);
```

$$5$$

5.3 与其它程序语言的连接

5.3.1 转换成 FORTRAN 或 C 语言

调用 **codegen** 程序包中的 **fortran** 命令可以把 Maple 的结果转换成 FORTRAN 语言:

```
> with(codegen,fortran):
  f:= 1-2*x+3*x^2-2*x^3+x^4;
```

$$f := 1 - 2\,x + 3\,x^2 - 2\,x^3 + x^4$$

```
> fortran(%);
      t0 = 1-2*x+3*x**2-2*x**3+x**4
 > fortran(f,optimized);
      t2 = x**2
      t6 = t2**2
```

```
      t7 = 1-2*x+3*t2-2*t2*x+t6
> fortran(convert(f,horner,x));
      t0 = 1+(-2+(3+(-2+x)*x)*x)*x
```

而 codegen 程序包中的 C 命令可以把 Maple 结果转换成 C 语言格式:

```
> with(codegen,C):
f:=1-x/2+3*x^2-x^3+x^4;
```

$$f := 1 - \frac{1}{2}x + 3\,x^2 - x^3 + x^4$$

```
> C(f);
      t0 = 1.0-x/2.0+3.0*x*x-x*x*x+x*x*x*x;
> C(f,optimized);
      t2 = x*x;
      t5 = t2*t2;
      t6 = 1.0-x/2.0+3.0*t2-t2*x+t5;
```

optimized 命令表示要对转换的表达式进行优化，如果不加此可选参数，则直接对表达式进行一一对应的转换.

5.3.2 生成 LATEX

Maple 可以把它的表达式转换成 LATEX，使用 latex 命令即可:

```
> latex(x^2+y^2=z^2);
{x}^{2}+{y}^{2}={z}^{2}
```

还可以将转换结果存为一个文件(LatexFile):

```
> latex(x^2 + y^2 = z^2, LatexFile);
```

再如下例:

```
> latex(Int(1/(x^2+1),x)=int(1/(x^2+1),x));
\int \! \left( {x}^{2}+1 \right) ^{-1}{dx}=\arctan\left( x \right)
```

第三章　线性代数

1 多项式

多项式是代数学中最基本的对象之一，它不但与高次方程的讨论有关，而且是进一步学习代数以及其它数学分支的基础.

1.1 多项式生成及类型测试

在 Maple 中，多项式由名称、整数和其他 Maple 值，通过+、-、*和^等组合而成. 例如：

```
> p1:=5*x^5+3*x^3+x+168;
```

$$p1 := 5\,x^5 + 3\,x^3 + x + 168$$

这是一个整系数单变量多项式. 多元多项式和定义在其他数域上的多项式可以类似构造：

```
> p2:=3*x*y^2*z^3+2*sqrt(-1)*x^2*y*z+2002;
```

$$p2 := 3\,x\,y^2\,z^3 + 2\,I\,x^2\,y\,z + 2002$$

由此可以看出, Maple 中多项式的生成与“赋值”命令相似.

另外，还可以通过函数 randpoly 生成随机多项式，生成一个关于 vars 的随机多项式的格式如下：

randpoly(vars, opts);

其中, vars 表示变量或者变量列表或集合, opts 为可选项方程或者指定属性的名称. 如：

```
> randpoly(x);      # 随机生成关于x的 5 次(默认)多项式
```

$$-42\,x^5 + 88\,x^4 - 76\,x^3 - 65\,x^2 + 25\,x + 28$$

```
> randpoly([x, y], terms=8);      # 随机生成关于[x, y]二元 8 项多项式
```

$$-78\,x\,y + 62\,x^3 + 11\,x^2\,y + 88\,x^3\,y + x\,y^3 + 30\,y^4 + 81\,x^4\,y - 5\,x^2\,y^3$$

```
> randpoly([x, sin(x), cos(x)]);
```

$$-73\sin(x)\cos(x) - 91\sin(x)\cos(x)^2 + x^3\cos(x) + 5\,x^4\sin(x) - 86\,x^2\cos(x)^3 + 43\sin(x)^4\cos(x)$$

而要随机生成关于[x, y, z]的密集的、均匀的、度为 2 的多项式的命令为：

```
> randpoly([x,y,z],dense,homogeneous,degree=2);
```

$$-85\,x^2 - 55\,z\,x - 37\,y\,x - 35\,z^2 + 97\,y\,z + 50\,y^2$$

用 type 命令可以测试多项式的类型：

```
> type(p1, polynom(integer, x));          #测试p1 是否是一个关于x的整系数多项式
```

true

```
> type(p2, polynom(complex, {x, y, z}));  #测试p2 是否是一个关于{x, y, z}的复系数多项式
```

true

1.2 提取多项式系数

coeff 函数用来提取一元多项式的系数，而多元多项式所有系数的提取用命令 coeffs，指定系数的提取用命令 coftayl.

(1) 提取多项式 p 中 x^n 的系数使用命令：coeff(p, x^n);或 coeff(p, x, n);

(2) 提取多项式 p 中变量 x 的所有系数并将相应的 x 幂存于变量 t 中：coeffs(p, x, 't');

(3) 返回 expr 在 x=a 处的 Taylor 展式中(x-a)^k 的系数: coeftayl(expr, x=a, k);

```
> p:=2*x^2+3*y^3*x-5*x+68;
```

$$p := 2\,x^2 + 3\,y^3\,x - 5\,x + 68$$

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

$$3\,y^3 - 5$$

```
> coeff(x^4-5*x^2-sin(a)*(x+1)^2, x^2);
```

$$-5 - \sin(a)$$

```
> s:=3*x^2*y^2+5*x*y;
```

$$s := 3\,x^2\,y^2 + 5\,x\,y$$

```
> coeffs(s);
```

$$5,\ 3$$

```
> coeffs(s, x, 't');
```

$$5\,y,\ 3\,y^2$$

```
> t;
```

$$x,\ x^2$$

```
> coeftayl(exp(x), x=0, 10);
```

$$\frac{1}{3628800}$$

```
> p:=3*(x+1)^3+sin(Pi/3)*x^2*y+x*y^3+x-6;
```

$$p := 3\,(x+1)^3 + \frac{1}{2}\sqrt{3}\,x^2 y + x\,y^3 + x - 6$$

> **coeftayl(p, x=-1, 1);**

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

> **coeftayl(p, [x, y]=[0, 0], [1, 0]);**

$$10$$

返回默认为降序排列的多元多项式的首项和末项系数分别使用命令 lcoeff、tcoeff：

> **lcoeff(p, x);**

$$3$$

> **tcoeff(p, x);**

$$-3$$

1.3 多项式的约数和根

1.3.1 多项式的最大公约因式(gcd)/最小公倍因式(lcm)

求多项式的最大公约因式/最小公倍因式的命令与求两个整数最大公约数/最小公倍数命令一样，都是 gcd/lcm. 命令格式分别为：

gcd(p1, p2, 't', 's');

lcm(p1, p2, 't', 's');

其中，第 3 个参数 t 赋值为余因子 p1/gcd(p1, p2)，第 4 个参数 s 赋值为余因子 p2/gcd(p1, p2).

> **p1:=x^4+x^3+2*x^2+x+1;**

$$p1 := x^4 + x^3 + 2\,x^2 + x + 1$$

> **p2:=x^2+x+1;**

$$p2 := x^2 + x + 1$$

> **gcd(p1, p2, 't', 's');**

$$x^2 + x + 1$$

> **t, s;**

$$x^2 + 1,\ 1$$

> **lcm(p1, p2);**

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

1.3.2 多项式的平方根(psqrt)和第 n 次方根(proot)

求多项式 p 的平方根，若不是完全平方，则返回_NOSQRT：psqrt(p);

求多项式 p 的 n 次方根，若不是完全 n 次方，则返回_NOROOT：proot(p, n);

> **p:=x^4+4*x^3+6*x^2+4*x+1;**

$$p := x^4 + 4\,x^3 + 6\,x^2 + 4\,x + 1$$

> **psqrt(p);**

$$x^2 + 2\,x + 1$$

> **proot(p, 4);**

$$x + 1$$

> **proot(p, 8);**

$$\_NOROOT$$

1.3.3 多项式相除的余式(rem)/商式(quo)

计算p1 除以p2 的余式，将商式赋值给q的命令格式为：rem(p1, p2, x, 'q');

计算p1 除以p2 的商式，将余式赋值给r的命令格式为：quo(p1, p2, x, 'r');

余式和商式满足：p1=p2*q+r，其中degree(r, x)<degree(p2, x)

> **rem(x^5+x^3+x, x+1, x, 'q');**

$$-3$$

> **q;**

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

> **quo(x^3+x^2+x+1, x-1, x, 'r');**

$$x^2 + 2\,x + 3$$

> **r;**

$$4$$

1.4 多项式转换及整理

1.4.1 将多项式转换成 Horner 形式

将多项式 poly 转换成关于变量 var 的 Horner 形式或者嵌套形式的命令格式如下：

convert(poly, horner, var);

> **convert(x^5+x^4+x^3+x^2+x+1, horner, x);**

$$1 + (1 + (1 + (1 + (x + 1)\,x)\,x)\,x)\,x$$

> **convert(x^3*y^3+x^2*y^2+x*y+1, horner, [x, y]);**

$$1 + (y + (y^2 + y^3\,x)\,x)\,x$$

1.4.2 将级数转换成多项式形式

将级数(series)转换成多项式(polynom)事实上就是忽略函数的级数展开式中的余项，

其命令格式为：

convert(series, polynom);

> **s:=series(sin(x), x, 10);**

$$s := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9 + O(x^{10})$$

> **type(s, polynom);**

$$false$$

> **p:=convert(s, polynom);**

$$p := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9$$

> **type(p, polynom);**

$$true$$

1.4.3 将级数转换成有理多项式(有理函数)

将级数 series(laurent 级数或 Chebyshev 类型级数)转换成有理多项式(有理函数)ratpoly 的命令格式为：

convert(series, ratpoly);

> **series(exp(x^2), x, 15);**

$$1 + x^2 + \frac{1}{2}x^4 + \frac{1}{6}x^6 + \frac{1}{24}x^8 + \frac{1}{120}x^{10} + \frac{1}{720}x^{12} + \frac{1}{5040}x^{14} + O(x^{15})$$

> **convert(%, ratpoly);**

$$\frac{\frac{1}{120}x^6 + \frac{1}{10}x^4 + \frac{1}{2}x^2 + 1}{-\frac{1}{120}x^6 + \frac{1}{10}x^4 - \frac{1}{2}x^2 + 1}$$

1.4.4 合并多项式系数(合并同类项)

将多项式具有相同次幂的项的系数合并在一起(包括正的、负的或者分数次幂)，即合并同类项(称为多项式的典范形式)，用命令 collect:

collect(p, x);

collect(p, x, form, func);

collect(p, x, func);

其中 x 是表示单变量 x 或多变量 x1, x2, …, xn 的一个列表或集合.

> **collect(a*ln(x)-ln(x)*x-x, ln(x));**

$$(a - x)\ln(x) - x$$

> **collect(x*(x+1)+y*(x+1), x);;**

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

> **collect(x*(x+1)+y*(x+1), y);**

$$x\,(x+1)+y\,(x+1)$$

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

collect(p, [x, y], recursive);

$$(1-a)\,y\,x^2+((1+a)\,y+1+a)\,x$$

> **collect(p, [y, x], recursive);**

$$((1-a)\,x^2+(1+a)\,x)\,y+(1+a)\,x$$

> **collect(p, {x, y}, distributed);**

$$(1+a)\,x+(1+a)\,x\,y+(1-a)\,y\,x^2$$

其中的参数recureive为递归式的，而distributed为分布式的。

1.4.5 **将多项式(或者值的列表)排序**

将多项式(或者值的列表)按升(或降)序次方排序时作命令sort. 命令格式：

sort(L);

sort(L, F);

sort(A);

sort(A, V);

其中, L—表示要排序的列表; F(可选项)—带两个参数的布尔函数; A—代数表达式; V(可选项)—变量

sort函数将列表L按升序次方，将代数表达式A中的多项式按降序次方排列.

如果给定了F，它将用来定义一个排序的列表. 如果F是符号“<”或者数值，那么L是一个列表类型，按数值型降序排列；如果F是符号“>”，L按数值型升序排列；如果F是一个词典编纂的符号，那么字符串和符号列表将按词典编纂的次序排列. 另外，F必须是一个有两个参数的布尔函数.

在Maple中，多项式并不自动按排序次序存储而是按建立的次序存储. 值得注意的是, sort函数对多项式的排序是破坏性的操作，因为输入的多项式将会按排序后的次序存储.

用来对列表排序的算法是一种带早期排序序列监测的合并排序的递归算法，而用于对多项式排序的算法是一个原位替换排序.

> **sort([3, 2, 1, 5, 4]);**

$$[1, 2, 3, 4, 5]$$

> **sort(x^3+x^2+1+x^5, x);**

$$x^5+x^3+x^2+1$$

> **sort((y+x)^2/(y-x)^3*(y+2*x), x);**

$$\frac{(x+y)^2\,(2\,x+y)}{(-x+y)^3}$$

1.4.6 多项式的因式分解

一般情况下，计算带有整数、有理数、复数或代数数系数的多项式的因式分解使用命令 factor，命令格式为：

factor(expr, K);

其中, expr 为多项式, K 为因式分解所在的区域

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

$$(x+y)\,(x^2-x\,y+y^2)$$

```
> factor(t^6-1,  (-3)^(1/2));
```

$$\frac{1}{16}\,(2\,t+1-\sqrt{-3}\,)\,(2\,t+1+\sqrt{-3}\,)\,(2\,t-1-\sqrt{-3}\,)\,(2\,t-1+\sqrt{-3}\,)\,(t+1)\,(t-1)$$

```
> factor(x^3+5, {5^(1/3),  (-3)^(1/2)});
```

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

但是factor不分解整数，也不分解多项式中的整数系数，整数或整数系数的分解使用函数ifactor：

```
> ifactor(2^99+1);
```

$$(3)^3\ (19)\ (67)\ (683)\ (242099935645987\ )\ (20857)\ (5347)$$

1.5 多项式运算

1.5.1 多项式的判别式

多项式判别式在多项式求根中具有重要作用，例如二次多项式的判别式. 求多项式 p 关于变量 x 的判别式的命令格式为：

discrim(p, x);

```
> p1:=a*x^2+b*x+c;
```

$$p1 := a\,x^2+b\,x+c$$

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

$$-4\,a\,c+b^2$$

```
> p2:=a*x^3+b*x^2+c*x+d;
```

$$p2 := a\,x^3+b\,x^2+c\,x+d$$

```
> discrim(p2, x);
```

$$-27\,a^2\,d^2+18\,a\,d\,c\,b+b^2\,c^2-4\,b^3\,d-4\,a\,c^3$$

1.5.2 多项式的范数

计算关于变元 v 的多项式 p 的第 n 阶范数用命令 norm，格式如下：

norm(p, n, v);

其中 p 是展开的多项式，n 为实常数或无穷大. 对于 n≥1，范数 norm 定义为 norm(p, n, v)=sum(abs(c)^n), c=[coeffs(p, v)]^(1/n).

> **norm(x^3+x^2+x+1, 1);**

$$4$$

> **norm(x-3*y, 2);**

$$\sqrt{10}$$

> **norm(x-3*y, infinity);**

$$3$$

1.5.3 bernoulli 数及多项式

bernoulli 函数计算第 n 阶 bernoulli 数和关于表达式 x 的第 n 阶 bernoulli 多项式 bernoulli(n, x), bernoulli(n, x)通过指数生成函数定义为：

$$\frac{te^{xt}}{e^t-1}=\sum_{n=0}^{\infty}\frac{\mathrm{bernoulli}(n,x)}{n!}t^n$$

第 n 阶 bernoulli 数定义为：bernoulli(n)= bernoulli(n, 0)

> **bernoulli(6);**

$$\frac{1}{42}$$

> **bernoulli(10, x);**

$$\frac{5}{66}-\frac{3}{2}x^2+5\,x^4-7\,x^6+\frac{15}{2}x^8+x^{10}-5\,x^9$$

> **bernoulli(6, 1/2);**

$$\frac{-31}{1344}$$

1.5.4 用 Bernstein 多项式近似一个函数

bernstein 多项式起因于 stone-weierstrass 近似分析理论，该理论认为任何连续函数都可以在一个闭区间用多项式序列来近似，bernstein 多项式就是该理论的一个实例. 若给定 p:=(n, i, x)->binomial(n, i)*x^i*(1-x)^(n-i)时，Bernstein 多项式定义为：

$$\mathrm{Berstein}(n,f,x)=\sum_{i=0}^{n}p(n,i,x)\,f\left(\frac{i}{n}\right)$$

其中, binomial(n, r) = GAMMA(n+1)/GAMMA(r+1)/GAMMA(n-r+1)

在 Maple 中，在[0, 1]区间上近似等于函数 f(x)的关于 x 的 n 次方的 Bernstein 多项的操作命令是：

bernstein(n, f, x);

其中, f 必须是用一个程序或者操作符指定的单变量函数.

> **bernstein(3, x->1/(x+1), z);**

$$1-\frac{3}{4}z+\frac{3}{10}z^2-\frac{1}{20}z^3$$

> **f:= proc(t) if t < 1/2 then 4*t^2 else 2 - 4*t^2 end if end proc:**
bernstein(8, f, x);

$$\frac{1}{2}x-63\,x^5+\frac{7}{2}x^2+119\,x^6+20\,x^8-82\,x^7$$

1.5.5 获取多项式的最高/最低次方

获取多项式的最高/最低次方的 Maple 命令分别为 degree 和 ldegree，格式如下：

degree(p, x);

ldegree(p, x);

> **p:=3/x^3+x^3-(x-1)^4;**

$$p := 3\frac{1}{x^3}+x^3-(x-1)^4$$

> **degree(p, x); ldegree(p, x);**

4

-3

> **degree(x*sin(x), x);**

FAIL

> **degree(x*sin(x), sin(x));**

1

1.5.6 **Euler 数及多项式**

第 n 阶 Euler 多项式 E(n, x)由指数生成函数定义为：

$$\frac{2e^{xt}}{e^t H}=\sum_{n=0}^{\infty}\frac{\mathrm{E}(n,x)}{n!}t^n$$

据此，第 n 阶 Euler 数 E(n)由指数生成函数定义为：

$$\frac{2}{e^t+e^{-t}}=\sum_{n=0}^{\infty}\frac{\mathrm{E}(n)}{n!}t^n$$

由此可见，第 n 阶 Euler 多项式和 Euler 数的关系为：

$$E(n)=2^n E(n,\frac{1}{2})$$

在 Maple 中，获取第 n 阶 Euler 多项式和 Euler 数的命令分别为：

euler(n, x);　　euler(n);

其中, n 为非负整数, x 为表达式.

> **euler(6);**

$$-61$$

> **euler(6, x);**

$$x^6-3x^5+5x^3-3x$$

> **euler(6, 1/2);**

$$\frac{-61}{64}$$

1.5.7 多项式插值

函数 interp 计算次方小于或等于 n 的过插值点(x[1], y[1]), (x[2], y[2]), …, (x[n+1], y[n+1])关于变量 v 的多项式，命令格式如下：

interp(x, y, v);

值得注意的是，如果相同的 x 值输入了 2 次，不论是否输入了相同的 y 值都会导致错误，也就是说所有插值必须不一样. 如过点(1, 2), (2, 4), (3, 6), (4, 8), (5, 10), (6, 12), (7, 14), (8, 16), (9, 18), (10, 22)的关于 Z 的插值多项式的计算如下：

> **interp([1,2,3,4,5,6,7,8,9,10], [2,4,6,8,10,12,14,16,18,22], z);**

$$\frac{1}{181440}z^9-\frac{1}{4032}z^8+\frac{29}{6048}z^7-\frac{5}{96}z^6+\frac{3013}{8640}z^5-\frac{95}{64}z^4+\frac{4523}{1134}z^3-\frac{6515}{1008}z^2+\frac{9649}{1260}z-2$$

1.5.8 计算自然样条函数

计算一个关于变量 z 的次数是 n(默认为 3)的分段多项式来近似 X, Y 数据值. X 值必须唯一且按升序排列，而对 Y 值没有特别的约定. 命令格式为：

spline(X, Y, z, n);

> **spline([1, 2, 3, 4], [2, 3, 6, 5], x, linear);**#生成线性样条插值

$$\begin{cases} 1+x & x<2 \\ -3+3x & x<3 \\ 9-x & otherwise \end{cases}$$

> **spline([0, 1, 2, 3], [0, 1, 4, 3], x, cubic);**# 生成三次样条插值

$$\begin{cases} \frac{1}{5}x+\frac{4}{5}x^3 & x<1 \\ \frac{14}{5}-\frac{41}{5}x+\frac{42}{5}x^2-2x^3 & x<2 \\ -\frac{114}{5}+\frac{151}{5}x-\frac{54}{5}x^2+\frac{6}{5}x^3 & otherwise \end{cases}$$

1.6 有理式

所谓有理式，是指可以表示成 f/g 形式的式子，其中 f 与 g 均为多项式，且 $g \neq 0$.

1.6.1 获取表达式的分子/分母

对于一个有理式 x，可以用 **numer**(x)和 **denom**(x)来获得它的分子(numerator)和分母(denumirator):

```
> f:=x^2+3*x+2: g:=x^2+5*x+6: h:=f/g;
```

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

```
> numer(h);denom(h);
```

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

$$x^2+5x+6$$

1.6.2 有理表达式的标准化

和数字除法不同, Maple 并不自动化简分式，使分子和分母互不可约，除非分子和分母都表示成乘积的形式并且有公共部分. 如果要将有理式化简成最简形式即标准化，需要调用函数 **normal**( ) (也称正则化). 事实上，normal 函数提供了化简的一种基本形式，它首先识别在有理数主域上等于 0 的表达式包括求和、乘积、整数幂次和变量构成的任何表达式. Normal 的结果是使有理式转化为约化的标准形式，即：分子分母是带整数系数的素数多项式.

```
> normal(f/g);
```

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

```
> normal(x^2-(x+1)*(x-1)-1);
```

$$0$$

```
> normal((f(x)^2-1)/(f(x)-1));
```

$$f(x)+1$$

```
> normal(1/x+x/(x+1), expanded);
```

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

> **normal(2/x+y/3);**

$$\frac{1}{3}\,\frac{6+x\,y}{x}$$

1.6.3 将有理式转换为多项式与真分式和的形式

convert/parfrac可将有理函数 f 分解为多项式与真分式和的形式，这是有理函数积分的基础. 命令格式如下：

convert(f, parfrac, x, K);

其中, f为x的有理函数, x为主变量名称, K为可选参数(实数—real, 复数—complex, 扩展域—a field extended, 真—true, 假—false, 无平方项—sqrfree)等.

> **p:=(x^5+1)/(x^3+1);**

$$p := \frac{x^5+1}{x^3+1}$$

> **convert(p, parfrac, x);**

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

> **convert(x/(x-a)^2, parfrac, x);**

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

1.6.4 将浮点数转换为接近的有理数

convert/rational将一个浮点数转换成一个近似的有理数, 转换的精度取决于环境变量Digits的值. 命令格式为：

convert(float, rational, Digits);

> **convert(1.333333333, rational);**

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

> **convert(evalf(Pi), rational,6);**

$$\frac{355}{113}$$

```
> convert(evalf(Pi), rational, 3);
```

$$\frac{22}{7}$$

2 矩阵基本运算

2.1 数组和向量

在 Maple 中，数组(**array**)由命令 **array** 产生，其下标变量(**index**)可以自由指定．下标由 1 开始的一维数组称为向量(**vector**)，二维以上的数组称为矩阵(**matrix**)，因此，可以说向量与矩阵是数组的特殊形式．生成数组与向量的命令分别列示如下：

array (m, n); ＃产生一个下标变量为 m 到 n 的一维数组

array (m, n, [a1, a2 .. ak]); ＃产生一维数组，并依次填入初值

array ([a1, a2 ..ak]); ＃产生下标变量为 1 到 k 的一维数组，并依次填入初值

vector(n, [v1, v2, …, vn]); ＃产生 n 维向量(n 可省略)

```
> A:=array(-1..2);
```

$$A := \mathrm{array}(-1 \mathinner{..} 2, [\ ])$$

```
> A[0]:=12;
```

$$A_0 := 12$$

```
> A[1]:=2*x^2+x+4;
```

$$A_1 := 2x^2 + x + 4$$

```
> A;
```

$$A$$

```
> lprint(eval(A));
array(-1 .. 2, [(0)=12, (1)=2*x^2+x+4])
> B:=array(0..2, [x, x^2, x^3]);
```

$$B := \mathrm{array}(0 \mathinner{..} 2, [\ ])$$

$$(0) = x$$

$$(1) = x^2$$

$$(2) = x^3$$

```
> C:=array(1..3, [x, x^2, x^3]);
```

$$C := [x, x^2, x^3]$$

```
> print(C);
```

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

> **type(B, vector);**

false

> **type(B, array);**

true

> **type(C, vector);**

true

> **type(C, array);**

true

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

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

> **Vector(1..3, 5);**

$$\begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix}$$

注意 **vector** 命令使用时大小写的区别：矢量 vector(小写)指 linalg 包的基于数组的矢量，而 Vector(大写)指 LinearAlgebra 包的基于 rtable 的矢量.

当我们把一个向量赋给某个变量后，Maple 在显示这个向量时，只会显示出变量名称，而不会显示出向量的内容，这个设计可大大地简化向量的显示方式. 如果想要显示向量的内容，可用 **evalm** 指令来强迫对向量求值. 事实上，**evalm** 不仅可以显示向量，也可以显示矩阵.

> **A:=array([x, y, z]);**

$$A := [x, y, z]$$

> **B:=array([1, 2, 3]);**

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

> **2*A+B;**

$$2A + B$$

> **evalm(2*A+B);**

$$[2x + 1, 2y + 2, 2z + 3]$$

另一个有用的命令是将列表、数组或Vector矢量A转换成vector的函数convert：

convert(A, vector);

> **A:=array(1..2, 1..2, [[1, 2], [3, 4]]):**
convert(A, vector);

$$[1, 2, 3, 4]$$

```
> V:=Vector([1, 6, 8]);
```

$$V := \begin{bmatrix} 1 \\ 6 \\ 8 \end{bmatrix}$$

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

$$false$$

```
> convert(V, vector);
```

$$[1, 6, 8]$$

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

$$true$$

2.2 矩阵的建立

在使用 Maple 进行线性代数运算时，需要先载入线性代数工具包—**linalg**，绝大部分线性代数运算函数都存于该工具包中.

在 Maple 中，建立矩阵用 matrix 命令. 如建立一个 m 行 n 列的矩阵的命令格式为：

matrix(m,n);

matrix(m,n,init);

其中，init 可以有很多种选择，如程序、数组、列表、方程组、代数表达式或者矩阵的初值等。

```
> with(linalg):
> A:=array([[1,2,3],[4,5,6],[7,8,9]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> f:=(i,j)->i^2+j^3;
```

$$f := (i, j) \to i^2 + j^3$$

```
> B:=matrix(4,5,f);
```

$$B := \begin{bmatrix} 2 & 9 & 28 & 65 & 126 \\ 5 & 12 & 31 & 68 & 129 \\ 10 & 17 & 36 & 73 & 134 \\ 17 & 24 & 43 & 80 & 141 \end{bmatrix}$$

```
> C:=matrix(2,2,[1,2,3,4]);
```

$$C := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

而函数 diag 可以生成块对角阵, band 函数可以建立以常数对角元素的带状矩阵：

```
> diag(A,C);
```

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \\ 7 & 8 & 9 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 & 4 \end{bmatrix}$$

```
> M:=Matrix([[1,1,1],[2,2,2,2],[3,3,3,3]], shape=band[1,3], scan=band[1,3]);
```

$$M := \begin{bmatrix} 2 & 3 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix}$$

内建于 linalg 的另一个有用的命令是从线性方程组中提取系数矩阵或增广矩阵，实现这一功能的函数是 genmatrix，其命令格式为：

genmatrix(eqns, vars, flag);

```
> eqns:={x+2*z=a,3*x-5*y=6-z};
```

$$eqns := \{x + 2\,z = a, 3\,x - 5\,y = 6 - z\}$$

```
> A:=genmatrix(eqns,[x,y,z],'flag');
```

$$A := \begin{bmatrix} 1 & 0 & 2 & a \\ 3 & -5 & 1 & 6 \end{bmatrix}$$

```
> B:=genmatrix(eqns,[x,y,z]);
```

$$B := \begin{bmatrix} 1 & 0 & 2 \\ 3 & -5 & 1 \end{bmatrix}$$

由此例可以看出，若加参数**'flag'**则提取增广矩阵，否则提取系数矩阵，这一点对于求解线性方程组很重要.

另外, Maple中还有一些预先编好的矩阵，可以直接由函数(命令)获得：

```
> hilbert(3);
```

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

```
> hilbert(3,x-1);
```

$$\begin{bmatrix} \frac{1}{3-x} & \frac{1}{4-x} & \frac{1}{5-x} \\ \frac{1}{4-x} & \frac{1}{5-x} & \frac{1}{6-x} \\ \frac{1}{5-x} & \frac{1}{6-x} & \frac{1}{7-x} \end{bmatrix}$$

```
> toeplitz([1,2,3,4]);
```

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

```
> vandermonde([1,x-1,(x-1)^2,(x-1)^3]);
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x-1 & (x-1)^2 & (x-1)^3 \\ 1 & (x-1)^2 & (x-1)^4 & (x-1)^6 \\ 1 & (x-1)^3 & (x-1)^6 & (x-1)^9 \end{bmatrix}$$

```
> fibonacci(3);
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

还有一个有用的命令是随机生成矩阵的函数 randmatrix：

```
> randmatrix(3,3);
```

$$\begin{bmatrix} 63 & 57 & -59 \\ 45 & -8 & -93 \\ 92 & 43 & -62 \end{bmatrix}$$

同样，随机生成向量的函数为 randvector.

2.3 矩阵的基本运算

矩阵的基本运算命令列于下表，而矩阵的代数运算最直观的方法莫过于使用函数 **evalm** 了，只要把矩阵的代数计算表达式作为它的参数，就可以得到结果.

运算	函数调用	等效的运算符运算
加法	matadd(A, B)	evalm(A+B)
数乘	scalarmul(A, expr)	evalm(A*expr)

乘法	multiply(A, B, …)	evalm(A &*B &* …)
逆运算	inverse(A)	evalm(1/A)或 evalm(A^(-1))
转置	transpose(A)	无
行列式	det(A)	无
秩	rank(A)	无
迹	trace(A)	无

```
> A:=randmatrix(2,2);
```

$$A := \begin{bmatrix} 72 & 66 \\ -29 & -91 \end{bmatrix}$$

```
> B:=randmatrix(2,2);
```

$$B := \begin{bmatrix} -53 & -19 \\ -47 & 68 \end{bmatrix}$$

```
> matadd(A,B);   evalm(A+B);
```

$$\begin{bmatrix} 19 & 47 \\ -76 & -23 \end{bmatrix}$$

```
> multiply(A,B);   evalm(A*B);
```

$$\begin{bmatrix} -6918 & 3120 \\ 5814 & -5637 \end{bmatrix}$$

```
> inverse(%);       evalm(1/%);
```

$$\begin{bmatrix} \frac{-1879}{6952362} & \frac{-520}{3476181} \\ \frac{-323}{1158727} & \frac{-1153}{3476181} \end{bmatrix}$$

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

$$\begin{bmatrix} -.0002702678600 & -.0001495894489 \\ -.0002787541845 & -.0003316858357 \end{bmatrix}$$

```
> transpose(A);
```

$$\begin{bmatrix} 72 & -29 \\ 66 & -91 \end{bmatrix}$$

```
> rank(B); trace(B);
```

$$2$$

$$15$$

值得注意的是，矩阵乘法运算符“&*”有着和数乘“*”、除法“/”相同的优先级，在输入表达式时需要注意，必要时要使用括号，请看下面实验：

```
> evalm(A&*1/A);
```

Error, (in evalm/amperstar) &* is reserved for matrix multiplication

> **evalm(A&*(1/A));**

$$\&*(\ \ )$$

上面最后的输出结果表示单位阵.

2.4 矩阵的求值

数组和单个的表达式不同，具有数组类型的变量不会自动求值，而需要用 **eval** 等函数的显式调用来强制地求值. 作为数组的特例，矩阵和向量也是一样，但要让矩阵最终完成求值还需使用函数 **map**，**map** 函数的主要功能是，对于任意一元函数，不加任何说明，就可以作用在整个数组上，得到的结果是每一个元素分别调用函数所得结果所组成的数组.

> **with(linalg):**

> **Q:=matrix([[cos(alpha), sin(alpha)], [-sin(alpha), cos(alpha)]]);**

$$Q := \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

> **alpha:=0;**

$$\alpha := 0$$

> **eval(Q);**

$$\begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

> **map(eval, Q);**

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

另一个对矩阵中元素变量代换的命令是 subs 函数(但只能进行一层求值)，例如：

> **P:=matrix(2, 2, [1, 2, x, x^3]);**

$$P := \begin{bmatrix} 1 & 2 \\ x & x^3 \end{bmatrix}$$

> **subs(x=8, eval(P));**

$$\begin{bmatrix} 1 & 2 \\ 8 & 512 \end{bmatrix}$$

2.5 矩阵分解

矩阵分解在矩阵运算中的作用最明显的一点是可以大幅度提高运算效率. 矩阵分解算法较多，主要有LU分解、QR分解等。

2.5.1 **LU 分解**

LUdecomp(**A**, P='p', L='l', U='u' , U1='u1', R='r', rank='ran', det='d');

其中, A 为矩阵(长方形), P='p'—主元素因子, L='l'—单位下三角因子, U='u'—上三角因子, U1='u1'—修改的 U 因子, R='r'—行减少因子, rank='ran'—A 的秩, det='d'—U1 的行列式. 当然, 此命令的简写形式为：LUdecomp(**A**);

```
> A:=vandermonde([1,3,5,7]);
```

$$A := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 9 & 27 \\ 1 & 5 & 25 & 125 \\ 1 & 7 & 49 & 343 \end{bmatrix}$$

```
>LUdecomp(A, P='p', L='l', U='u' , U1='u1', R='r', rank='ran', det='d');
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 8 & 26 \\ 0 & 0 & 8 & 72 \\ 0 & 0 & 0 & 48 \end{bmatrix}$$

```
> evalm(l);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 3 & 1 \end{bmatrix}$$

```
> ran;
```

$$4$$

```
> d;
```

$$768$$

对于代数元素矩阵, 只有首项元素为 0 时才选主元素, 主元素矩阵返回为 P.

2.5.2 **QR 分解**

在QR分解中, 矩阵A被看作乘积Q•R, 此处, Q为正交或归一化矩阵, R为上三角阵. 该分解是对一系列线性无关向量作Gram-Schmidt正交化, 因此, Q包含了正交的向量, R的列记录了产生原向量的线性组合. 命令格式为：

QRdecomp(A, Q='q', rank='r', fullspan=value);

其中, A 为矩阵(长方形), Q='q'—A 的 Q 因子, rank='ran'—A 的秩, fullspan=value—在 Q 中包含空格(true 或 false).

```
> A:=matrix(3,3,[1,2,3,4,5,6,7,8,10]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

```
> QRdecomp(A,Q='q',rank='r');
```

$$\begin{bmatrix} \sqrt{66} & \frac{13}{11}\sqrt{66} & \frac{97}{66}\sqrt{66} \\ 0 & \frac{3}{11}\sqrt{11} & \frac{5}{11}\sqrt{11} \\ 0 & 0 & \frac{1}{6}\sqrt{6} \end{bmatrix}$$

```
> r;
```

$$3$$

```
> evalm(q);
```

$$\begin{bmatrix} \frac{1}{66}\sqrt{66} & \frac{3}{11}\sqrt{11} & \frac{1}{6}\sqrt{6} \\ \frac{2}{33}\sqrt{66} & \frac{1}{11}\sqrt{11} & -\frac{1}{3}\sqrt{6} \\ \frac{7}{66}\sqrt{66} & -\frac{1}{11}\sqrt{11} & \frac{1}{6}\sqrt{6} \end{bmatrix}$$

另外，对实正定矩阵可采用Cholesky分解，这是一种形式更对称的LU分解.

3 矩阵的初等变换和线性方程组求解

3.1 矩阵的初等变换

矩阵的初等变换是各种消去法的基础，是求解线性方程组的基础. 对于符号运算，有时候我们不仅要求最后的求解结果，而且要求中间的求解步骤，此时就需调用线性代数工具包中的初等变换函数. 如下表所示：

	函数调用	对应的初等变换
行变换	mulrow(A, r, expr)	用标量 expr 乘以矩阵 A 的第 r 行
	addrow(Ar1, r2, m)	将矩阵 A 的第 r1 行的 m 倍加到第 r2 行上
	swaprow(A, r1, r2)	互换矩阵 A 的第 r1 行和第 r2 行
列变换	mulcol(A, c, expr)	用标量 expr 乘以矩阵 A 的第 c 列
	addcol(A, c1, c2)	将矩阵 A 的第 c1 列的 m 倍加到第 c2 列上
	swapcol(A, c1, c2)	互换矩阵 A 的第 c1 列和第 c2 列

除了这些初等变换外，**Maple** 在 **linalg** 工具包中还提供了一些矩阵的形状变换，可以在利用初等变换解决问题时起到辅助作用，我们也将它们一并列出.

函数调用	所作的变换或操作
contat(A, B, …);	将矩阵 A, B, …在水平方向上合并成一个矩阵
stackmatrix(A, B, …)	将矩阵 A, B, …在竖直方向上合并成一个矩阵
row(A, i)	取矩阵 A 的第 i 行
col(A, i)	取矩阵 A 的第 i 列
row(A, i .. k)	取矩阵 A 的第 i 到 k 行
col(A, i .. k)	取矩阵 A 的第 i 到 k 列
delrows(A, i .. k)	删除矩阵 A 中 i 到 k 行剩下的子矩阵
delcols(A, i .. k)	删除矩阵 A 中 i 到 k 列剩下的子矩阵
extend(A, m, n, x)	扩展矩阵 m 行 n 列并用 x 填充
submatrix(A, m .. n, r .. s)	取矩阵 A 的 m 到 n 行、r 到 s 列组成的子矩阵

例：利用矩阵的初等变换判断矩阵A是否可逆，若可逆，求A^{-1}

$$A=\begin{pmatrix}-2 & 1 & 3\\ 0 & -1 & 1\\ 1 & 2 & 0\end{pmatrix}$$

```
> with(linalg):
A:=matrix([[-2, 1, 3], [0, -1, 1], [1, 2, 0]]);
```

$$A := \begin{bmatrix}-2 & 1 & 3\\ 0 & -1 & 1\\ 1 & 2 & 0\end{bmatrix}$$

```
> AI:=concat(A, array(identity, 1..3, 1..3));
```

$$AI := \begin{bmatrix}-2 & 1 & 3 & 1 & 0 & 0\\ 0 & -1 & 1 & 0 & 1 & 0\\ 1 & 2 & 0 & 0 & 0 & 1\end{bmatrix}$$

```
> mulrow(AI, 1, -1/2);
```

$$\begin{bmatrix} 1 & \frac{-1}{2} & \frac{-3}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 \\ 1 & 2 & 0 & 0 & 0 & 1 \end{bmatrix}$$

> **addrow(%,1, 3, -1);**

$$\begin{bmatrix} 1 & \frac{-1}{2} & \frac{-3}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & \frac{5}{2} & \frac{3}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

> **mulrow(%, 2, -1);**

$$\begin{bmatrix} 1 & \frac{-1}{2} & \frac{-3}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & \frac{5}{2} & \frac{3}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

> **addrow(%, 2, 1, 1/2);**

$$\begin{bmatrix} 1 & 0 & -2 & \frac{-1}{2} & \frac{-1}{2} & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & \frac{5}{2} & \frac{3}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

> **addrow(%, 2, 3, -5/2);**

$$\begin{bmatrix} 1 & 0 & -2 & \frac{-1}{2} & \frac{-1}{2} & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 4 & \frac{1}{2} & \frac{5}{2} & 1 \end{bmatrix}$$

> **mulrow(%, 3, 1/4);**

$$\begin{bmatrix} 1 & 0 & -2 & \frac{-1}{2} & \frac{-1}{2} & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

> **addrow(%, 3, 1, 2);**

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-1}{4} & \frac{3}{4} & \frac{1}{2} \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

> **addrow(%, 3, 2, 1);**

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-1}{4} & \frac{3}{4} & \frac{1}{2} \\ 0 & 1 & 0 & \frac{1}{8} & \frac{-3}{8} & \frac{1}{4} \\ 0 & 0 & 1 & \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

> **AA:=submatrix(%, 1..3, 4..6);**

$$AA := \begin{bmatrix} \frac{-1}{4} & \frac{3}{4} & \frac{1}{2} \\ \frac{1}{8} & \frac{-3}{8} & \frac{1}{4} \\ \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

上述判断矩阵可逆并求逆矩阵的方法即高斯—约当消去法，从例中可以看出，按高斯—约当消去法步骤逐一在 Maple 下操作即可实现．在这里，数学基础显然是最重要的．当然，上述例子可通过下述两句命令即可完成：

> **det(A);**

若 det(A)不为 0 执行下句即可求出 A 的逆矩阵：

> **inverse(A);**

3.2 线性方程组的解

线性方程组求解常常转化为与其等价的矩阵问题来解决的．Maple中可借助函数**genematrix**及高斯消去法函数**gausselim**联合完成：

> **eqns:={x+2*y+3*z=a,8*x+9*y+4*z=b,7*x+6*y+5*z=c};**

$$eqns := \{x + 2y + 3z = a, 8x + 9y + 4z = b, 7x + 6y + 5z = c\}$$

> **A:=genmatrix(eqns,[x,y,z],'flag');**

$$A := \begin{bmatrix} 1 & 2 & 3 & a \\ 8 & 9 & 4 & b \\ 7 & 6 & 5 & c \end{bmatrix}$$

> **gausselim(%);**

$$\begin{bmatrix} 1 & 2 & 3 & a \\ 0 & -7 & -20 & b-8a \\ 0 & 0 & \frac{48}{7} & c+\frac{15}{7}a-\frac{8}{7}b \end{bmatrix}$$

可以看出，在高斯消去法的结果矩阵中出现了许多分数，这使表达式看起来不美观，而且随着矩阵的增大，分数的分母会越来越大，如果矩阵中含有符号，结果矩阵中将出现分式．为了避免出现上述情况，可采用无分式高斯消去法(fraction-free Gaussian elimination)，相应的命令为 **ffgausselim**：

> **ffgausselim(%%);**

$$\begin{bmatrix} 1 & 2 & 3 & a \\ 0 & -7 & -20 & b-8a \\ 0 & 0 & -48 & -7c-15a+8b \end{bmatrix}$$

在把系数矩阵转化成相应的上三角阵后，就可以用回代(back substitution)的方法求出各个未知数的值—**backsub**：

> **backsub(%);**

$$\left[-\frac{7}{16}a-\frac{1}{6}b+\frac{19}{48}c, \frac{1}{3}b+\frac{1}{4}a-\frac{5}{12}c, \frac{7}{48}c+\frac{5}{16}a-\frac{1}{6}b\right]$$

和回代相对，也有前代(forward substitution)的概念，可用前代函数 **forwardsub** 法语解下三角矩阵．有了这一对函数，我们就可以用 LU 分解来线性方程组了，相应的命令函数为 **LUdecomp**．关于上一问题也可以利用高斯—约当消去法(Gauss-Jordan elimination)，把系数矩阵变换成单位阵直接得到结果：

> **gaussjord(A);**

$$\begin{bmatrix} 1 & 0 & 0 & -\frac{7}{16}a-\frac{1}{6}b+\frac{19}{48}c \\ 0 & 1 & 0 & \frac{1}{3}b+\frac{1}{4}a-\frac{5}{12}c \\ 0 & 0 & 1 & \frac{7}{48}c+\frac{5}{16}a-\frac{1}{6}b \end{bmatrix}$$

上面介绍的实际上都是线性方程求解的中间步骤，事实上我们可以一步到位，不管它到底用的是什么方法，而只要求得最终的结果．这方面，Maple 提供了一个非常好的函数 **linsolve**，这个函数不仅可以用来求解具有唯一解的线性方程组，而且可以求解有无穷多解的方程组并给出通解．试看下面的实验：

例：求解线性方程组：$\begin{cases} x_1+3x_2+3x_3+2x_4=-1 \\ 2x_1+6x_2+9x_3+5x_4=4 \\ -x_1-3x_2+3x_3=13 \end{cases}$

```
> A:=matrix([[1, 3, 3, 2], [2, 6, 9, 5], [-1, -3, 3, 0]]);
```

$$A := \begin{bmatrix} 1 & 3 & 3 & 2 \\ 2 & 6 & 9 & 5 \\ -1 & -3 & 3 & 0 \end{bmatrix}$$

```
> B:=vector([-1, 4, 13]);
```

$$B := [\,-1,\ 4,\ 13\,]$$

```
> linsolve(A, B);
```

$$[\,-13-3\,\_t_2+3\,\_t_1,\ \_t_2,\ \_t_1,\ 6-3\,\_t_1\,]$$

可以看到, Maple用辅助变量$_t_1$, $_t_2$给出了方程组的通解.

3.3 最小二乘法求解线性方程解

在实际问题中, 由于误差或者其他各方面的原因, 很容易出现无解的方程组. 这样问题的解决则变成求出一个“最优”的近似解. 在线性代数中, 通常采用最小二乘解(least-squares solution). linalg 中对应的函数是 **leastsqrs**, 它有两种调用格式：

leastsqrs(A, b);

leastsqrs(S, v);

其中，A 为矩阵, b 为向量，S 为线性方程组, v 为变量名。

```
> with(linalg):
> S:={c[0]+c[1]+c[2]-3, c[0]+2*c[1]+4*c[2]-10, c[0]-9/10, c[0]-c[1]+c[2]-3};
```

$$S := \{\,c_0-c_1+c_2-3,\ c_0+2\,c_1+4\,c_2-10,\ c_0+c_1+c_2-3,\ c_0-\frac{9}{10}\,\}$$

```
> leastsqrs(S,{c[0],c[1],c[2]});
```

$$\{\,c_1=\frac{7}{200},\ c_0=\frac{159}{200},\ c_2=\frac{91}{40}\,\}$$

再如：

```
> A:=matrix(3,2,[0,1,1,1,1,1]);
```

$$A := \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

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

$$b := [0, 1, -1]$$

```
> leastsqrs(A,b);
```

$$[0, 0]$$

对于相容方程组，leastsqrs命令与solve功能一样：

```
> leastsqrs({x+y=2,y+z=3,x+z=3},{x,y,z});
```

$$\{x = 1, y = 1, z = 2\}$$

```
> solve({x+y=2,y+z=3,x+z=3},{x,y,z});
```

$$\{x = 1, y = 1, z = 2\}$$

3.4 正定矩阵

在 Maple 中，判断一个矩阵是否是正(负)定矩阵非常简单，只需要使用命令 **definite** 即可，而且还可求出符号矩阵的正(负)定条件：

```
> with(linalg):
> A := matrix(2, 2, [2,1,1,3]);
```

$$A := \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

```
> definite(A, 'positive_def');
```

$$true$$

```
> B:=array(1..2, 1..2, 'symmetric'):
definite(B, 'negative_semidef');
```

$$B_{1,1} \le 0 \text{ and } -B_{1,1} B_{2,2} + B_{1,2}^{2} \le 0 \text{ and } B_{2,2} \le 0$$

由上述实验可以看出，**definite**的第 1 个参数是需要判定的矩阵，第 2 个参数是矩阵的正定性，共有 4 种情况：**'positive_def'**(正定)、**'positive_semidef'**(半正定)、**'negative_def'**(负定)、**'negative_semidef'**(半负定). 当判定数值矩阵时，返回布尔值true/false; 判定符号矩阵时，它返回一个布尔表达式，表示正负定的条件.

4 特征值、特征向量和矩阵的相似

4.1 矩阵的相似

在 Maple 中可以利用 **linalg** 中的 **issimilar** 判断两个矩阵是否相似. 命令格式如下：

issimilar(A, B, 'P');

其中 A, B 为方阵, P 为转换矩阵(可选). 如果 A, B 相似，则返回 true，否则返回 false.

```
> with(linalg):
> A:=matrix(2, 2);
```

$$A := \text{array}(1 .. 2, 1 .. 2, [\ ])$$

```
> B:=matrix(2, 2);
```

$$B := \text{array}(1 .. 2, 1 .. 2, [\ ])$$

```
> issimilar(A&*B, B&*A);
```

true

在这个例子中，Maple 作了一定的假设，由于 A 的行列式是一个符号表达式，Maple 在判定时假设它为非零常数. 不过更多情况下，这个函数是用在数值矩阵的判定上，同时也还可以求出转换矩阵 P, P 赋值为满足 A=inverse(P)*B*P 的转换矩阵 P.

```
> with(linalg,matrix,issimilar,eigenvalues,diag):
> A:=matrix(3,3,[1,2,3,4,5,6,7,8,9]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> B:=diag(eigenvalues(A));
```

$$B := \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{15}{2}+\frac{3}{2}\sqrt{33} & 0 \\ 0 & 0 & \frac{15}{2}-\frac{3}{2}\sqrt{33} \end{bmatrix}$$

```
> issimilar(A,B,P);
```

true

```
> print(P);
```

$$\begin{bmatrix} \frac{1}{6} & \frac{-1}{3} & \frac{1}{6} \\ -\frac{5}{12}+\frac{7}{132}\sqrt{33} & -\frac{1}{6}+\frac{1}{198}\sqrt{33} & \frac{1}{12}-\frac{17}{396}\sqrt{33} \\ \frac{5}{12}+\frac{7}{132}\sqrt{33} & \frac{1}{6}+\frac{1}{198}\sqrt{33} & -\frac{1}{12}-\frac{17}{396}\sqrt{33} \end{bmatrix}$$

```
> map(normal,evalm(P^(-1)&*B&*P));
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

4.2 特征值和特征向量

特征值问题可以说是最常见的线性代数问题了. 在 Maple 中求解特征矩阵(characteristic matrix)和特征多项式(characteristic polynomial)的函数分别是 **charmat** 和 **charpoly**, 求解相应的特征值和特征向量的函数分别是 **eigenvalues(eigenvals)** 和 **eigenvectors**. 命令格式如下:

charmat(A, lambda);

charpoly(A, lambda);

eigenvalues(A);

eigenvectors(A);

下面通过几个实例学习上述 4 个函数的用法:

```
> A:=randmatrix(2,2);
```

$$A := \begin{bmatrix} 17 & 72 \\ -99 & -85 \end{bmatrix}$$

```
> charmat(A,lambda);
```

$$\begin{bmatrix} \lambda - 17 & -72 \\ 99 & \lambda + 85 \end{bmatrix}$$

```
> charpoly(A,lambda);
```

$$\lambda^2 + 68\,\lambda + 5683$$

```
> eigenvalues(A);
```

$$-34 + 3\,I\sqrt{503}, -34 - 3\,I\sqrt{503}$$

```
> eigenvectors(A);
```

$$\left[-34 + 3\,I\sqrt{503}, 1, \left\{\left[1, -\frac{17}{24} + \frac{1}{24}\,I\sqrt{503}\right]\right\}\right], \left[-34 - 3\,I\sqrt{503}, 1, \left\{\left[1, -\frac{17}{24} - \frac{1}{24}\,I\sqrt{503}\right]\right\}\right]$$

```
> A:=matrix(3, 3, [1, 2, 2, 2, 1, 2, 2, 2, 1]);
```

$$A := \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

```
> charmat(A, lambda);
```

$$\begin{bmatrix} \lambda - 1 & -2 & -2 \\ -2 & \lambda - 1 & -2 \\ -2 & -2 & \lambda - 1 \end{bmatrix}$$

```
> charpoly(A, lambda);
```

$$\lambda^3 - 3\lambda^2 - 9\lambda - 5$$

> **eigenvals(A);**

$$5, -1, -1$$

> **eigenvectors(A);**

$$[-1, 2, \{[-1, 1, 0], [-1, 0, 1]\}], [5, 1, \{[1, 1, 1]\}]$$

可以看到，特征向量函数**eigenvectors**在给出特征向量的时候，还给出了特征值及特征值重数．一般情况下，如果同时需要得到特征值和特征向量用**eigenvectors**函数即可．另外，这个函数还可处理符号矩阵，此时，通常以根式的形式给出结果，如果加上可选参数**'implicit'**，结果就以**RootOf**的形式给出．

函数 **eigenvalues** 不仅可以求解普通特征值问题，还可求解广义特征值问题，即求解方程 $\det(\lambda C - A) = 0$ 相应的命令函数为 **eigenvals**(A, C).

由特征值理论，任何一个实对称阵，都可以用求特征值的方法将它对角化，但对于非对称阵或者是复矩阵，就没有这样的保证了．不过对于任意复矩阵 A，还是可以把它化成相似的 Jordan 标准型，相应的命令函数是 **jordan**，而且还可给出转换矩阵 P.

> **A:=matrix(3, 3, [2, -1, 1, 2, 2, -1, 1, 2, -1]);**

$$A := \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & -1 \\ 1 & 2 & -1 \end{bmatrix}$$

> **jordan(A, 'p');**

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

> **print(p);**

$$\begin{bmatrix} 0 & 1 & 1 \\ 3 & 2 & 0 \\ 3 & 1 & 0 \end{bmatrix}$$

在 Maple 中，可以调用函数 **JordanBlock**(λ, k) 构造约当块 $J(\lambda, k)$，利用该命令和 diag 函数连用，就可以生成具有约当标准型的矩阵:

> **JordanBlock(3, 4);**

$$\begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

> **linalg[JordanBlock](x, 5);**

$$\begin{bmatrix} x & 1 & 0 & 0 & 0 \\ 0 & x & 1 & 0 & 0 \\ 0 & 0 & x & 1 & 0 \\ 0 & 0 & 0 & x & 1 \\ 0 & 0 & 0 & 0 & x \end{bmatrix}$$

5 线性空间基本理论

5.1 基本子空间

与矩阵A相关的R^n的四个子空间分别是行空间、列空间、化零空间和左零空间. 化零空间, 实际上就是齐次线性方程组$AX=0$ 的解空间, 可利用linsolve求得其通解, 再将通解中的辅助变量依次替换成 1, 就可以获得解空间的一组基. 对于左零空间, 即方程组$A^TX=0$ 的解空间, 亦可用同样的方法得到. 行空间和列空间, 在linalg中也有专门的函数rowspace和colspace, 可以获得它们的基和维数. 例举如下：

> **with(linalg):**
> **A:=matrix(3, 2,[2, 0, 3, 4, 0, 5]);**

$$A := \begin{bmatrix} 2 & 0 \\ 3 & 4 \\ 0 & 5 \end{bmatrix}$$

> **rowspace(A, 'dim');**

$$\{[0, 1], [1, 0]\}$$

> **dim;**

$$2$$

> **colspace(A);**

$$\left\{\left[0, 1, \frac{5}{4}\right], \left[1, 0, \frac{-15}{8}\right]\right\}$$

函数rowspace和colspace的第二个参数是可选参数, 用来返回行空间或列空间的维数, 它必须是一个变量名—可以是一个未赋值的新变量, 也可以是已有值的变量, 但要

加上延迟求值符“'”.

而寻找向量空间的基的函数是 basis:

```
> v1:=vector([1,0,0]):
v2:=vector([0,1,0]):
v3:=vector([0,0,1]):
v4:=vector([1,1,1]):
basis({v1,v2,v3});
```

$$\{ v2, v1, v3 \}$$

```
> basis({vector([1,1,1]),vector([2,2,2]),vector([1,-1,1]),vector([2,-2,2]),vector([1, 0, 1]),
vector([0, 1, 1])});
```

$$\{ [1, 1, 1], [0, 1, 1], [2, -2, 2] \}$$

5.2 正交基和 Schmidt 正交化

在欧氏空间中，我们可以定义两个向量的内积(inner product)，在此基础上，我们还可以定义两个向量的夹角. 如向量α、β的夹角θ可以定义为：

$$\theta = \arccos \frac{(\alpha, \beta)}{\|\alpha\| \|\beta\|}$$

在Maple的linalg工具包中，有相应的函数innerprod、angel可以计算向量的内积和夹角. 如：

```
> alpha:=vector([1, 2, -1, 11]);
```

$$\alpha := [1, 2, -1, 11]$$

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

$$\beta := [2, 3, 1, -1]$$

```
> innerprod(alpha, beta);
```

$$-4$$

```
> angle(alpha, beta);
```

$$\pi - \arccos\left(\frac{4}{1905} \sqrt{127}\ \sqrt{15} \right)$$

求三维向量的向量积使用命令 crossprod:

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

$$\alpha := [1, 2, 1]$$

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

$$\beta := [2, 3, 1]$$

```
> crossprod(alpha,beta);
```

$$[\,-1,\ 1,\ -1\,]$$

定义内积为零的向量间正交，此时可以利用 Schmit 正交化方法，由欧氏空间中一组普通基得到两两正交的基. Maple 中的相应函数是 **GramSchmidt**，它的输入参数是由一组向量组成的集合(或有序表)，将给出 Schmidt 正交化后的向量集合(或有序表). 输入的向量必须是线性无关的，否则，结果向量之间也将线性相关. 函数并不对向量进行单位化. 如果需要得到一组正交标准基，还需要用 map 方法对这些向量使用单位化函数 **normalize**.

例：用 Schmidt 正交化方法，由下列向量组构造出一组标准正交向量组：

$$(1,1,-1,-2)^T,\ (5,8,-2,-3)^T,\ (3,9,3,9)^T$$

```
> A:={vector([1, 1,-1, -2]), vector([5, 8, -2, -3]), vector([3, 9, 3, 9])};
```

$$A := \{\,[\,5,\ 8,\ -2,\ -3\,],\ [\,3,\ 9,\ 3,\ 9\,],\ [\,1,\ 1,\ -1,\ -2\,]\,\}$$

```
> simplify(map(normalize, GramSchmidt(A)));
```

$$\left\{\left[\frac{2}{91}\sqrt{2}\,\sqrt{91},\ -\frac{3}{182}\sqrt{2}\,\sqrt{91},\ -\frac{11}{182}\sqrt{2}\,\sqrt{91},\ \frac{3}{91}\sqrt{2}\,\sqrt{91}\right],\right.$$
$$\left.\left[\frac{1}{7}\sqrt{7},\ \frac{1}{7}\sqrt{7},\ -\frac{1}{7}\sqrt{7},\ -\frac{2}{7}\sqrt{7}\right],\left[\frac{2}{39}\sqrt{39},\ \frac{5}{39}\sqrt{39},\ \frac{1}{39}\sqrt{39},\ \frac{1}{13}\sqrt{39}\right]\right\}$$

事实上，向量的 **Schmidt** 正交化过程实际上给出了矩阵的 *QR* 分解.

第二章　微积分运算

微积分是数学学习的重点和难点之一，而微积分运算是 Maple 最为拿手的计算之一，任何解析函数，Maple 都可以求出它的导数来，任何理论上可以计算的积分，Maple 都可以毫不费力的将它计算出来. 随着作为数学符号计算平台的 Maple 的不断开发和研究，越来越多的应用程序也在不断地创设.

1 函数的极限和连续

1.1 函数和表达式的极限

在 Maple 中，利用函数 **limit** 计算函数和表达式的极限. 如果要写出数学表达式，则用惰性函数 **Limit**. 若 a 可为任意实数或无穷大时，求 $\lim\limits_{x\to a} f(x)$ 命令格式为: limit(f,x=a); 求 $\lim\limits_{x\to a^+} f(x)$ 时的命令格式为 limit(f, x=a, right); 求 $\lim\limits_{x\to a^-} f(x)$ 时的命令格式为 limit(f, x=a, left); 请看下述例子：

```
> Limit((1+1/x)^x,x=infinity)=limit((1+1/x)^x,x=infinity);
```

$$\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^{x}=\mathbf{e}$$

```
> Limit((x^n-1)/(x-1),x=1)=limit((x^n-1)/(x-1),x=1);
```

$$\lim_{x\to 1}\frac{x^{n}-1}{x-1}=n$$

```
> Limit(x^x,x=0,right)=limit(x^x,x=0,right);
```

$$\lim_{x\to 0+} x^{x}=1$$

```
> Limit(abs(x)/x,x=0,left)=limit(abs(x)/x,x=0,left);
```

$$\lim_{x\to 0-}\frac{|x|}{x}=-1$$

```
> Limit(abs(x)/x,x=0,right)=limit(abs(x)/x,x=0,right);
```

$$\lim_{x\to 0+}\frac{|x|}{x}=1$$

```
> limit(abs(x)/x,x=0);
```

$$undefined$$

对于多重极限计算，也用limit. 命令格式为: limit(f, points, dir); 其中, points是由一系列方程定义的极限点, dir(可选项)代表方向: left(左)、right(右)等. 例如:

```
> limit(a*x*y-b/(x*y),{x=1,y=1});
```

$$a - b$$

```
> limit(x^2*(1+x)-y^2*((1-y))/(x^2+y^2),{x=0,y=0});
```

$$undefined$$

由于多重极限的复杂性，很多情况下 limit 无法找到答案，此时，不应轻易得出极限不存在的结论，而是应该利用数学基础判定极限的存在性，然后再寻找别的可行的方法计算极限(如化 n 重根限为 n 次极限等)。如下例就是化二重极限为二次极限而得正确结果：

```
> limit((sin(x+y)/(sin(x)*sin(y)),{x=Pi/4,y=Pi/4}));
```

$$\mathrm{limit}\left(\frac{\sin(x+y)}{\sin(x)\sin(y)}, \{x=\frac{1}{4}\pi, y=\frac{1}{4}\pi\}\right)$$

```
> limit(limit(sin(x+y)/(sin(x)*sin(y)),x=Pi/4),y=Pi/4);
```

$$2$$

1.2 函数的连续性

1.2.1 连续

在 Maple 中可以用函数 **iscont** 来判断一个函数或者表达式在区间上的连续性. 命令格式为:

iscont(expr, x=a..b, 'colsed'/'opened');

其中, closed表示闭区间，而opened表示开区间(此为系统默认状态).

如果表达式在区间上连续，iscont返回true，否则返回false，当iscont无法确定连续性时返回FAIL. 另外，iscont函数假定表达式中的所有符号都是实数型. 颇为有趣的是，当给定区间[*a*,*b*] (*a*>*b*)时, iscont会自动按[*b*,*a*]处理.

```
> iscont(1/x,x=1..2);
```

$$true$$

```
> iscont(1/x,x=-1..1,closed);
```

$$false$$

```
> iscont(1/(x+a),x=0..1);
```

$$FAIL$$

```
> iscont(ln(x),x=10..1);
```

$$true$$

1.2.2 间断

函数 **discont** 可以寻找函数或表达式在实数域的间断点，当间断点周期或成对出现

时，Maple 会利用一些辅助变量予以表达，比如，**_Zn**~(任意整数)、**_NZn**~(任意自然数)和 **Bn**~(一个二进制数, 0 或者 1)，其中 n 是序号．判定 f(x)间断点的命令为:

discont(f, x);

```
> discont(ln(x^2-4),x);
```

$$\{-2, 2\}$$

```
> discont(arctan(1/2*tan(2*x))/(x^2-1),x);
```

$$\{-1, 1, \frac{1}{2}\pi \_Z1\sim + \frac{1}{4}\pi\}$$

```
> discont(round(3*x-1/2),x);
```

$$\{\frac{1}{3} + \frac{1}{3}\_Z1\}$$

函数round为“四舍五入”函数，上例并非一目了然，对其进一步理解可借助于函数plot或下面给出的fdiscont例子。

另一个寻找间断点的函数**fdiscont**是用数值法寻找在实数域上的间断点．命令格式为:

fdiscont(f, domain, res, ivar, eqns);

其中，f表示表达式或者，domain表示要求的区域，res表示要求的分辨率，ivar表示独立变量名称，eqns表示可选方程.

```
> fdiscont(GAMMA(x/2), x=-10..0, 0.0001);
```

[-10.0000086158831731 .. -9.99994148230377888 , -8.00006570776243642 .. -7.99994267738026022 , -6.00006150975869534 .. -5.99992641140166860 , -4.00004327130417892 .. -3.99994355004940650 , -2.00005917347370676 .. -1.99993152974711540 , -.000068747862775166 1410 .. .000012481225191394 1740]

```
> fdiscont(arctan(1/2*tan(2*x))/(x^2-1),x=-Pi..Pi);
```

[-1.00024845147850305 .. -.999333284093140484 , -.785872546183299604 .. -.785043522184549426 , .785028686514634 86 .. .785693822920970008 , .999744688307389716 .. 1.00074853662123409 , 2.35588567826642148 .. 2.35643817620602248]

```
> fdiscont(abs(x/10000),x=-1..1,0.000001);
```

[ ]

```
> fdiscont(tan(10*x),x=0..Pi,0.01,newton=true);
```

[.157079632679489656 , .471238898038468967 , .785398163397448280 , 1.09955742875642758 , 1.41371669411540690 , 1.72787595947438644 , 2.04203522483336552 , 2.35619449019234484 , 2.67035375555132414 , 2.98451302091030346]

```
> fdiscont(round(3*x-1/2),x=-1..1);
```

[-1.00003155195758886 .. -.999634596231187556 , -.667056666250248842 .. -.666327819216380068 , -.333731406929595187 .. -.333002495225188489 , -.000262904384890887231 .. .000298288004691702826 , .332976914927844203 .. .333778498831551751 , .666340020328179072 .. .667141797110034518 , .999646728223793524 .. 1.00008356747731275]

2 导数和微分

2.1 符号表达式求导

利用 Maple 中的求导函数 **diff** 可以计算任何一个表达式的导数或偏导数，其惰性形式 **Diff** 可以给出求导表达式，**$**表示多重导数. 求 expr 关于变量 x1, x2, …, xn 的(偏)导数的命令格式为:

diff(expr, x1, x2, …, xn);

diff(expr, [x1, x2, …, xn]);

其中, expr 为函数或表达式, x1, x2, …, xn 为变量名称.

有趣的是，当 n 大于 1 时, diff 是以递归方式调用的:

diff(f(x), x, y)=diff(diff(f(x), x), y)

```
> Diff(ln(ln(ln(x))),x)=diff(ln(ln(ln(x))),x);
```

$$\frac{\partial}{\partial x}\ln(\ln(\ln(x))) = \frac{1}{x\ln(x)\ln(\ln(x))}$$

```
> Diff(exp(x^2),x$3)=diff(exp(x^2),x$3);
```

$$\frac{\partial^3}{\partial x^3}\mathbf{e}^{(x^2)} = 12\,x\,\mathbf{e}^{(x^2)} + 8\,x^3\,\mathbf{e}^{(x^2)}$$

```
> diff(x^2*y+x*y^2,x,y);
```

$$2x + 2y$$

```
> f(x,y):=piecewise(x^2+y^2<>0,x*y/(x^2+y^2));
```

$$\mathrm{f}(x,y) := \begin{cases} \dfrac{x\,y}{x^2+y^2} & x^2+y^2 \neq 0 \\ 0 & \textit{otherwise} \end{cases}$$

```
> diff(f(x,y),x);
```

$$\begin{cases} \dfrac{y}{x^2+y^2} - \dfrac{2\,x^2\,y}{(x^2+y^2)^2} & x^2+y^2 \neq 0 \\ 0 & \textit{otherwise} \end{cases}$$

```
> diff(f(x,y),x,y);
```

$$\begin{cases} \dfrac{1}{x^2+y^2} - \dfrac{2\,y^2}{(x^2+y^2)^2} - \dfrac{2\,x^2}{(x^2+y^2)^2} + \dfrac{8\,x^2\,y^2}{(x^2+y^2)^3} & x^2+y^2 \neq 0 \\ 0 & \textit{otherwise} \end{cases}$$

```
> normal(%);
```

$$\begin{cases} -\dfrac{x^4-6x^2y^2+y^4}{(x^2+y^2)^3} & x^2+y^2\neq 0 \\ 0 & otherwise \end{cases}$$

函数 diff 求得的结果总是一个表达式，如果要得到一个函数形式的结果，也就是求导函数，可以用 D 算子. D 算子作用于一个函数上，得到的结果也是一个函数. 求 f 的导数的命令格式为: D(f);

值得注意的是, f 必须是一个可以处理为函数的代数表达式，它可以包含常数、已知函数名称、未知函数名称、箭头操作符、算术和函数运算符.

复合函数表示为f@g，而不是f(g)，因此D(sin(y))是错误的，正确的应该是D(sin@y).

D 运算符也可以求高阶导数，但此时不用$，而用两个@@.

D 运算符并不局限于单变量函数，一个带指标的 **D** 运算符 D[i](f)可以用来求偏导函数, D[i](f)表示函数 f 对第 i 个变量的导函数，而高阶导数 D[i,j](f)等价于 **D[i](D[j](f))**.

```
> g:=x->x^n*exp(sin(x));
```

$$g := x \to x^n\, \mathbf{e}^{\sin(x)}$$

```
> D(g);
```

$$x \to \frac{x^n\, n\, \mathbf{e}^{\sin(x)}}{x} + x^n \cos(x)\, \mathbf{e}^{\sin(x)}$$

```
> Diff(g,x)(Pi/6)=D(g)(Pi/6);
```

$$\left(\frac{\partial}{\partial x} g\right)\left(\frac{1}{6}\pi\right) = 6\,\frac{\left(\frac{1}{6}\pi\right)^n n\, \mathbf{e}^{(1/2)}}{\pi} + \frac{1}{2}\left(\frac{1}{6}\pi\right)^n \sqrt{3}\, \mathbf{e}^{(1/2)}$$

```
> D(D(sin));
```

$$-\sin$$

```
> (D@@2)(sin);
```

$$-\sin$$

```
> f:=(x,y,z)->(x/y)^(1/z);
```

$$f := (x, y, z) \to \left(\frac{x}{y}\right)^{\left(\frac{1}{z}\right)}$$

```
> Diff(f,y)(1,1,1)=D[2](f)(1,1,1);
```

$$\left(\frac{\partial}{\partial y} f\right)(1, 1, 1) = -1$$

D 运算符和函数 diff 的差别:

1） D 运算符计算运算符的导数，而 diff 计算表达式的导数;

2） D 的参数和结果是函数型运算符，而 diff 的参数和结果是表达式;

3） 将含有导数的表达式转换为 D 运算符表达式的函数为: convert(expr,D);

```
> f:=diff(y(x),x$2):
```

$$f := \frac{\partial^2}{\partial x^2} y(x)$$

```
> convert(f,D);
```

$$(D^{(2)})(y)(x)$$

4） 将 D(f)(x)表达式转换为 diff(f(x),x)形式的命令: convert(expr,diff,x);

```
> f:=D(y)(x)-a*D(z)(x);
```

$$f := D(y)(x) - a\,D(z)(x)$$

```
> convert(f,diff,x);
```

$$\left(\frac{\partial}{\partial x} y(x)\right) - a\left(\frac{\partial}{\partial x} z(x)\right)$$

D 运算符可以计算定义为程序的偏导数，此即 Maple 自动求导功能(详细内容参看第 6 章).

下面我们讨论在积分学当中的一个微妙的漏洞，在大多数计算机代数系统中都会出现这个问题，甚至于在许多教科书和积分表中这种情况也是长期存在。

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

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

```
> F:=int(f,x);
```

$$F := \frac{2}{3}\sqrt{3}\,\arctan\left(\frac{1}{3}\left(2\tan\left(\frac{1}{2}x\right)+1\right)\sqrt{3}\right)$$

```
> limit(F,x=Pi,right), limit(F,x=Pi,left);
```

$$-\frac{1}{3}\pi\sqrt{3}, \frac{1}{3}\pi\sqrt{3}$$

关于函数 f(x)的积分仅在一些区间上是正确的，因为 F 是不连续的，虽然由微积分的基本定理可知当 f 连续时 F 应该是连续的。进一步的讨论 F 的不连续点：

```
> discont(F,x);
```

$$\{2\,\pi\,\_Z2 + \pi\}$$

因此，F 在 $x = n\pi$ 处有跳跃间断点。

在对多元函数 f(x,y)求混合偏导数时，Maple 总自以为是 $\frac{\partial f}{\partial x \partial y} = \frac{\partial f}{\partial y \partial x}$，这一点在 f(x,y)连续的情况下当然正确，但不连续时不正确。一个典型的例子是：

```
> f(x,y):=piecewise(x^2+y^2<>0,x*y*(x^2-y^2)/(x^2+y^2));
```

$$f(x,y):=\begin{cases}\dfrac{x\,y\,(x^2-y^2)}{x^2+y^2} & x^2+y^2\neq 0\\ 0 & otherwise\end{cases}$$

****

```
> normal(diff(f(x,y),x,y));
```

$$\begin{cases}\dfrac{x^6+9\,x^4\,y^2-9\,x^2\,y^4-y^6}{\left(x^2+y^2\right)^3} & x^2+y^2\neq 0\\ 0 & otherwise\end{cases}$$

```
> normal(diff(f(x,y),y,x));
```

$$\begin{cases}\dfrac{x^6+9\,x^4\,y^2-9\,x^2\,y^4-y^6}{\left(x^2+y^2\right)^3} & x^2+y^2\neq 0\\ 0 & otherwise\end{cases}$$

因此，使用 Maple 进行科学计算时，一定要先运用数学理论和方法对问题进行简单推导，然后再利用 Maple 辅助计算，切不可把所有的事情都交给 Maple，如果那样的话会出现错误甚至是低级的错误。

2.2 隐函数求导

隐函数或由方程(组)确定的函数的求导，使用命令 **implicitdiff**. 假定 f, f1,⋯,fm 为代数表达式或者方程组, y, y1,⋯,yn 为变量名称或者独立变量的函数，且 m 个方程 f1,⋯,fm 隐式地定义了 n 个函数 y1,⋯,yn，而 u, u1,⋯,ur 为独立变量的名称, x, x1,⋯,xk 为导数变量的名称. 则：

(1) 求由 f 确定的 y 对 x 的导数:

implicitdiff(f,y,x);

(2) 求由 f 确定的 y 对 x1,⋯,xk 的偏导数:

implicitdiff(f,y,x1,⋯,xk);

(3) 计算 u 对 x 的导数，其中 u 必须是给定的 y 函数中的某一个

implicitdiff({f1,⋯,fm},{y1,⋯,yn},u,x);

(4) 计算 u 对 x1,⋯,xk 的偏导数

implicitdiff({f1,⋯,fm},{y1,⋯,yn},u,{x1,⋯xk});

(5) 计算 u 的高阶导数

implicitdiff({f1,⋯,fm},{y1,⋯,yn},{u1,⋯,ur}, x1,⋯,xk);

implicitdiff(f,y,x)命令的主要功能是求隐函数方程 f 确定的 y 对 x 的导数，因此，输入的 f 必须是 x 和 y 或者代数表达式的方程(其中代数表达式为 0). 第二个参数 y 指定了非独立变量、独立变量或常数，如果 y 是名称，就意味着非独立变量，而所有其他出现在输入的 f 和求导变量 x 中名称以及不看作是常数类型的变量，统统视作独立变量处理.

如果方程 f1,…,fm 是超定的, implicitdiff 返回 FAIL. 例如:

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

$$f := \mathbf{e}^{y} - x\,y^{2} = x$$

```
> implicitdiff(f,y,x);
```

$$-\frac{y^{2}+1}{-\mathbf{e}^{y}+2\,x\,y}$$

```
> g:=x^2+y^3=1;
```

$$g := x^{2} + y^{3} = 1$$

```
> implicitdiff(g,z,x);
```

$$FAIL$$

如果是对多元函数求多个偏导数, 结果将用偏微分形式给出. 可以给定最后一个可选参数来确定结果的表达形式, 默认情况下或者给定 **notation=D**, 这时结果中的微分用 D 运算符表示, 否则可以给定 **notation=Diff**, 这样给出的结果中的微分运算符和使用 Diff 时相同, 即用 ∂ 来表示. 试作以下实验:

$$\begin{cases} x = \cos u \cos v \\ y = \cos u \sin v \\ z = \sin u \end{cases} \quad 求 \frac{\partial^{2} z}{\partial x^{2}}$$

```
> f:=x=cos(u)*cos(v);
```

$$f := x = \cos(u)\cos(v)$$

```
> g:=y=cos(u)*sin(v);
```

$$g := y = \cos(u)\sin(v)$$

```
> h:=z=sin(u);
```

$$h := z = \sin(u)$$

```
> implicitdiff ({f,g,h}, {z(x,y), u(x,y), v(x,y)}, {z}, x, x, notation=Diff);
```

$$\left\{ \left(\frac{\partial^{2}}{\partial x^{2}} z \right)_{y} = -\frac{\sin(v)^{2}\sin(u)^{2} + 1 - \sin(v)^{2}}{\sin(u)^{3}} \right\}$$

2.3 函数的极值

2.3.1 函数的极值

极值包含两种情形：极大值和极小值。在 **Maple** 中, 有两个求函数极值的命令: minimize, maximize, 命令格式如下:

minimize (expr, vars, range);

maximize (expr, vars, range);

```
> expr1:=x^3-6*x+3:
minimize(expr1,x=-3..3);
```

$$-6$$

```
  maximize(expr1,x=-3..3);
```

$$12$$

```
> minimize(tanh(x),x,`infinite`);
```

$$-1$$

```
  maximize(tanh(x),x,`infinite`);
```

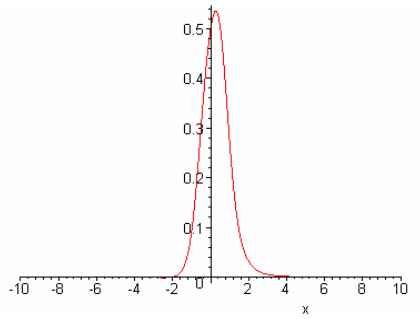
$$1$$

虽然，minimize 和 maximize 这两个命令很好用，但对于一些特殊的函数而言，这两个指令不但有可能无法求得极值，还有可能给我们错误的解. 因此，在 Maple 下求极值最好的方法是先作图(鼠标右键点击函数解析式选择 plots 命令即可)，由图上找出极值的大概位置，然后再由 Maple 提供的各种指令来求解. 下面一个例子是关于函数极大极小值的求解问题，此处，图形提供了做题的部分思路，尤其是求驻点时：

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

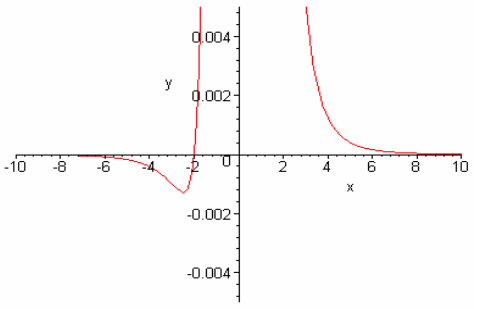
$$f := \frac{x+2}{3+(x^2+1)^3}$$

```
> plot(f,x=-10..10);
```



从上图可以看出，f(x)函数从区域-2 到 4 之间有极值出现，且极大值小于 1. 为了更清楚的了解函数图像性质，我们在 plot 命令中加入因变量 y 的变化范围。由此可看出 f(x)与 x 轴有交点，且在-2 附近有一极小值：

```
> plot(f,x=-10..10,y=-0.005..0.005);
```



进一步应用导数性质求解该问题：

```
> d:=diff(f,x);
```

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

```
> simplify(d);
```

$$-\frac{-4+5\,x^6+9\,x^4+3\,x^2+12\,x^5+24\,x^3+12\,x}{(4+x^6+3\,x^4+3\,x^2)^2}$$

由图形可见，极值“可能”出现在 x=-2 和 x=0 附近，可用下述语句求出确切极点，然后使用 eval 命令求出相应的极值：

```
> xmin:=fsolve(d=0,{x=-2});
```

$$xmin := \{x = -2.485165927\}$$

```
> xmax:=fsolve(d=0,{x=0});
```

$$xmax := \{x = .2700964050\}$$

```
> Digits:=4:
> Xmin:=eval(f,xmin);
```

$$Xmin := -.001302$$

```
> Xmax:=eval(f,xmax);
```

$$Xmax := .5360$$

2.3.2 条件极值

有时候，我们还会遇到在条件 $q(x,y)=0$ 下计算函数 $f(x,y)$ 的极大值和极小值，这就是条件极值. 在求解时需要先构造一个函数 $g(x,y)=f(x,y)+\mu q(x,y)$（μ 称为拉格朗日乘子），然后将 $g(x,y)$分别对 x 和 y 求导，得到联立方程组，求解方程组即可得到函数 $f(x,y)$ 的极大值和极小值.

下面求解 $f(x,y)=x^2+y^2$ 在条件 $q(x,y)=x^2+y^2+2x-2y+1$ 下的极大值和极小值.

```
> f:=x^2+y^2:
> q:=x^2+y^2+2*x-2*y+1:
> g:=f+mu*q;
```

$$g := x^2+y^2+\mu\,(x^2+y^2+2\,x-2\,y+1)$$

```
> exp1:=diff(g,x); exp2:=diff(g,y);
```

$$exp1 := 2x + \mu(2x + 2)$$

$$exp2 := 2y + \mu(2y - 2)$$

```
> exp3:=solve({q=0,exp1,exp2},{x,y,mu});
```

$$exp3 := \{y = \text{RootOf}(2\_Z^2 - 4\_Z + 1),\ x = -\text{RootOf}(2\_Z^2 - 4\_Z + 1),\ \mu = 1 - 2\,\text{RootOf}(2\_Z^2 - 4\_Z + 1)\}$$

```
> allvalues(exp3);
```

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

```
> subs({x=-1-1/2*2^(1/2),y=1+1/2*2^(1/2)},f):
> fmax:=evalf(%);
```

$$fmax := 5.828427124$$

```
> subs({x=-1+1/2*2^(1/2),y=1-1/2*2^(1/2)},f):
> fmin:=evalf(%);
```

$$fmin := .1715728755$$

3 积分运算

3.1 不定积分

Maple 有许多内建的积分算法，一般地，用 **int** 求不定积分．命令格式为:

int(expr,x);

```
> Int(x^2*arctan(x)/(1+x^2),x)=int(x^2*arctan(x)/(1+x^2),x);
```

$$\int \frac{x^2 \arctan(x)}{1+x^2}\,dx = \frac{1}{8}\ln(1+Ix)^2 + \frac{1}{2}I\left(-x + \frac{1}{2}I\ln(1-Ix)\right)\ln(1+Ix) + \frac{1}{8}\ln(1-Ix)^2 + \frac{1}{2}Ix\ln(1-Ix) - \frac{1}{2}\ln(1+x^2)$$

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

$$\frac{1}{3}\ln(x-1) - \frac{1}{6}\ln(1+x+x^2) + \frac{1}{3}\sqrt{3}\arctan\left(\frac{1}{3}(2x+1)\sqrt{3}\right)$$

```
> int(exp(-x^2),x);
```

$$\frac{1}{2}\sqrt{\pi}\,\text{erf}(x)$$

```
> Int(ln(x+sqrt(1+x^2)),x);
```

$$\int \ln(x + \sqrt{1+x^2})\,dx$$

```
> value(%)+c;
```

$$\ln(x + \sqrt{1+x^2})\,x - \sqrt{1+x^2} + c$$

```
> int(exp(-x^2)*ln(x),x);
```

$$\int \mathrm{e}^{(-x^2)} \ln(x)\, dx$$

可以看出，Maple 求不定积分的结果中没有积分常数，这一点需要注意. 但是，这有一定好处的，尤其当对结果作进一步处理时，由于 Maple 符号计算的特点，引入积分常数相当于引入一个变量，对于计算极为不便. Maple 中不定积分的计算过程为:

(i) 首先，Maple 用传统方法处理一些特殊的形式，如多项式、有理式、形如 $\left(\sqrt{a+bx+cx^2}\right)^n$ 和 $Q(x)\left(\sqrt{a+bx+cx^2}\right)^n$ 的根式，以及形如 $\mathrm{Pn}(x)\cdot\ln x$ 或 $\frac{P_1(x)}{Q_1(x)}\cdot\ln\frac{P_2(x)}{Q_2(x)}$ 的表达式;

(ii) 如果传统方法难以奏效, Maple 将应用 **Risch-Norman** 算法，以避免在包含三角函数和双曲函数的积分中引入复指数和对数;

(iii) 如果仍然无法得到答案, Maple 将采用 **Risch** 算法，这将无法避免地在结果表达式中引入有关积分变量的 RootOf 的表达式;

(iv) 如果最终还是没有找到解析表达式, Maple 会把积分式作为结果返回.

3.2 定积分

定积分与不定积分的计算几乎一样，只是多了一个表示积分区域的参数. 在[a,b]上求 f 的定积分的命令格式为:

int(f, x=a..b);

```
> Int(1/(1+x^2),x=-1..1)=int(1/(1+x^2),x=-1..1);
```

$$\int_{-1}^{1} \frac{1}{1+x^2}\, dx = \frac{1}{2}\pi$$

抛物线 $y^2 = 2px$ 与 $x^2 = 2py$ 所围图形的面积计算过程如下:

```
> assume(p>0);
> int(sqrt(2*p*x)-x^2/(2*p),x=0..2*p);
```

$$\frac{4}{3}p^2$$

Maple 中的定积分是怎样完成的呢？最简单的想法是: 按照 Newton-Leibnize 定理，先求出被积函数的任一个原函数，再求其在积分限上的增量即可得定积分. 试看下例(其中的函数 **rhs** 用来获取等式右边部分—**right hand side**，相应地，左边部分为 **lhs**):

```
> Int(1/x^2,x)=int(1/x^2,x);
```

$$\int \frac{1}{x^2}\,dx = -\frac{1}{x}$$

```
> subs(x=1,rhs(%))-subs(x=-1,rhs(%));
```

-2

显然，这是错误的，在上述积分中含有一个暇点 x=0，积分是发散的，对此，Maple 是知道的:

```
> Int(1/x^2,x=-1..1)=int(1/x^2,x=-1..1);
```

$$\int_{-1}^{1} \frac{1}{x^2}\,dx = \infty$$

在大多数情况下，Maple 通过查表和形式匹配，或者利用特殊函数的导数来求定积分.

```
> Int( exp(-x^2)*ln(x)^2, x=0..infinity )=int( exp(-x^2)*ln(x)^2, x=0..infinity );
```

$$\int_0^{\infty} \mathbf{e}^{(-x^2)} \ln(x)^2\,dx = \frac{1}{16}\pi^{(5/2)} + \frac{1}{8}\sqrt{\pi}\,\gamma^2 + \frac{1}{2}\sqrt{\pi}\,\gamma\ln(2) + \frac{1}{2}\sqrt{\pi}\,\ln(2)^2$$

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

1.947522181

```
>Int(sin(x)/x,x=-1..1)=int(sin(x)/x,x=-1..1);
```

$$\int_{-1}^{1} \frac{\sin(x)}{x}\,dx = 2\,\mathrm{Si}(1)$$

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

1.892166141

```
> Int(sin(x^2),x=-infinity..infinity)=int(sin(x^2),x=-infinity..infinity);
```

$$\int_{-\infty}^{\infty} \sin(x^2)\,dx = \frac{1}{2}\sqrt{2}\,\sqrt{\pi}$$

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

1.253314137

在返回一个未求值的定积分的情况下，可以对积分式调用 evalf 来获得数值积分. 命令格式为:

evalf(int(f, x=a..b));

evalf(Int(f, x=a..b, Digits, flag);

其中, f 为被积函数, x 积分变量, a..b 积分区间, Digits 表示需要精度的位数, flag 指定要使

用的数值方法的名称. 值得注意的是, 上述命令格式第一式 int 中的 i 可以大写也可以小写(输出结果略有形式上的不同), 第二式的 I 必须大写.

Maple 中默认的数值积分方法是 **Clenshaw-Curitis** 4 阶方法; 当收敛很慢(由于存在奇点)时, 系统将试着用广义的级数展开和变量代换取消积分的奇异性; 如果存在不可去奇点, 则改而采用自适应双指数方法. 在数值精度不高的情况下(比如 Digits≤15), 采用自适应的牛顿—柯特斯方法就够了. 通过指定 **evalf/int** 语句的第 4 个参数, 可以选择积分方法. 可供选择的有 3 种方法:

_Ccquad -- Clenshaw-Curitis 4 阶方法

_Dexp -- 自适应双指数方法

_Ncrule -- 牛顿—柯特斯方法

```
> evalf(Int(1/sqrt(x),x=0..2,15,_Dexp));
```

2.82842712474619

```
> evalf(Int(sin(x)/x,x=0..1,20,_NCrule));
```

.94608307036718301494

```
> evalf(Int(sin(x)*ln(x+1),x=0..1));
```

.2265353653

```
> evalf(Int(sin(x)*ln(x+1),x=0..1));
```

.2265353653

前面述及函数 value 的主要功能是对惰性函数求值, 但它与 evalf 是有区别的, 试通过下例体会 value 与 evalf 功能的不同之处:

```
> P:=Int(x^2*sin(sin(x)),x=0..Pi);
```

$$P := \int_0^{\pi} x^2 \sin(\sin(x))\, dx$$

```
> value(P);
```

$$\int_0^{\pi} x^2 \sin(\sin(x))\, dx$$

```
> evalf(P);
```

5.289745102

3.3 其它积分方法

Maple有丰富的内建积分算法, 除了上述**int**命令外, 另外一些算法也非常有用. 本节简述其中几种较为有用的算法.

3.3.1 三角和双曲积分

三角和双曲积分主要有下述几种:

$$S_i(x)=\int_0^x \frac{\sin t}{t}dt$$

$$C_i(x)=gamma+\ln(x)+\int_0^x \frac{\cos t-1}{t}dt$$

$$Ssi(x)=S_i-\frac{\pi}{2}$$

$$S_{hi}(x)=\int_0^x \frac{\sinh t}{t}$$

$$C_{hi}(x)=gamma+\ln(x)+\int_0^x \frac{\cosh t-1}{t}dt$$

上述函数在Maple中的调用格式分别为: Si(x); Ci(x); Ssi(x); Shi(x); Chi(x); 其中x为表达式(复数).

函数Si, Ssi和Shi是完整的, 函数Ci和Chi在原点处有一个对数极点, 在负实半轴上有一个分支截断点.

```
> int(sin(x)/x,x=0..1);
```

$$\mathrm{Si}(1)$$

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

$$.9460830704$$

```
> Ci(3.14159+1.23*I);
```

$$-.02624028922 - .4706897380\ I$$

```
> Ssi(2002.118);
```

$$.0003014039122$$

```
> evalf(Shi(Pi));
```

$$5.469640347$$

```
> evalf(Chi(3+4*I));
```

$$-2.077477803 + 2.142816206\ I$$

```
> convert(Ci(x),Ei);
```

$$-\frac{1}{2}\mathrm{Ei}(1, I\,x)-\frac{1}{2}\mathrm{Ei}(1, -I\,x)+\frac{1}{2}I\,(\mathrm{csgn}(x)-1)\,\mathrm{csgn}(I\,x)\,\pi$$

3.3.2 Dirac 函数和 Heaviside 阶梯函数

Dirac 函数和 Heaviside 函数主要应用于积分变换或者求解微分方程, 也可以用来表示分段连续函数. 其定义分别如下:

$$\mathrm{Dirac}(t)=\begin{cases}0 & t\neq 0\\ \infty & t=0\end{cases}\qquad \mathrm{Heaviside}(t)=\begin{cases}0 & t<0\\ 1 & t>0\end{cases}$$

命令格式为:

```
Dirac(t);        # Dirac函数(t=0时为无穷大, 其余处处为0)
Dirac(n,t);      # Dirac函数的n阶导数
```

Heaviside(t);　　＃Heaviside函数(t<0时为0, t>0时为1, t=0时无意义)

```
> Int(Dirac(t),t=-infinity..infinity)=int(Dirac(t),t=-infinity..infinity);
```

$$\int_{-\infty}^{\infty} \mathrm{Dirac}(t)\,dt = 1$$

```
> Int(Dirac(t),t)=int(Dirac(t),t);
```

$$\int \mathrm{Dirac}(t)\,dt = \mathrm{Heaviside}(t)$$

```
> Diff(Heaviside(t),t)=diff(Heaviside(t),t);
```

$$\frac{\partial}{\partial t}\mathrm{Heaviside}(t) = \mathrm{Dirac}(t)$$

3.3.3 指数积分

对于非负整数n，指数积分Ei(n,x)在实部Re(x)>0上定义为:

$$E_i(n,x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

单参数的指数积分是一个Cauchy主值积分，只对实参数x有如下定义:

$$E_i(x) = PV - \int_{-\infty}^{x} \frac{e^t}{t} dt$$

特别地，当x<0时，有: $E_i(x) = -E_i(1,-x)$

Ei(1,x)可以解析延拓到除了0点之外的整个复平面. 对于所有的这些函数，0是一个分支点，负实半轴是分支截断. 分支截断上的值要满足函数在增加参数的方向上是连续的条件.

指数函数和不完全GAMMA函数有如下关系:

$$E_i(n,x) = x^{n-1}GAMMA(1-n,x)$$

```
> Ei(1,1.); #=evalf(Ei(1,1));
```

$$.2193839344$$

```
> simplify(Ei(1,I*x)+Ei(1,-I*x));
```

$$-2\,\mathrm{Ci}(x) - I\,\pi + I\,\pi\,\mathrm{csgn}(x)$$

```
> expand(Ei(5,x));
```

$$\frac{1}{4}\mathbf{e}^{(-x)} - \frac{1}{12}x\,\mathbf{e}^{(-x)} + \frac{1}{24}x^2\,\mathbf{e}^{(-x)} - \frac{1}{24}x^3\,\mathbf{e}^{(-x)} + \frac{1}{24}x^4\,\mathrm{Ei}(1,x)$$

```
> evalf(Ei(1));
```

$$1.895117816$$

```
> int(exp(-3*t)/t,t=-x..infinity);
```

$$\mathrm{Ei}(1, -3x)$$

```
> int(exp(-3*t)/t,t=-x..infinity,CauchyPrincipalValue);
```

$$-\mathrm{Ei}(3x)$$

上述最后两例的结果大相径庭，原因是在最后一例中出现了“CauchyPrincipal Value”一选项，这一命令的主要功能是通知int将间断点的左右极限作为极限来处理，此时，独立变量按相同的速度接近间断点.

3.3.4 对数积分

对数积分Li(x)的定义为:

$$L_i(x) = PV - \int_0^x (1/\ln t)\mathrm{d}t = Ei(\ln x) \qquad (x \geq 0)$$

其中，PV-int表示Cauchy主值积分. 该函数只对实数参数 $x \geq 0$ 有定义，它给出了小于或等于x的素数的一个近似值.

```
> Li(2002.);   #对数积分在 2002.0 处的值
```

$$315.0723560$$

```
> nops(select(isprime, [$1..2002]));   # 小于或等于 2002 的实数中素数个数
```

$$303$$

```
> convert(Li(x),Ei);        #对数积分转换为指数积分
```

$$\mathrm{Ei}(\ln(x))$$

3.3.5 椭圆积分

所谓椭圆积分是形如 $\int_a^b R(x, y^{1/2})dx$ 的积分，其中R是一个有理数，y是3次或4次多项式，这是椭圆积分的代数形式. 除此之外还有三角形式、双曲三角等形式.

椭圆积分可以用初等函数项和椭圆函数项，如EllipticF，EllipticE和EllipticPi表示成它们的Legendre标准形式.

```
> ans:=int(sqrt(1+x^4)/(1-x^4),x=0..1/3);
```

$$ans := -\frac{1}{8}\sqrt{2}\,(\ln(2) + \ln(\sqrt{41} - 3)) + \frac{1}{8}\sqrt{2}\,(\ln(2) + \ln(3 + \sqrt{41}))$$
$$-\frac{1}{4}\sqrt{2}\,\arctan\left(\frac{1}{6}\sqrt{82}\,\sqrt{2}\right) + \frac{1}{8}\pi\sqrt{2}$$

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

$$.33457573315002445140$$

```
> assume(0<k,k<1);
> int(x^2/sqrt((1-x^2)*(1-k^2*x^2)),x=0..k);
```

$$\frac{\mathrm{EllipticF}(k, k)}{k^2} - \frac{\mathrm{EllipticE}(k, k)}{k^2}$$

```
> int(1/sqrt(-(x-1)*(x-2)*(x-3)),x=0..1/2);
```

$$\sqrt{2}\ \text{EllipticF}\left(\frac{2}{5}\sqrt{5},\frac{1}{2}\sqrt{2}\right)-\sqrt{2}\ \text{EllipticF}\left(\frac{1}{3}\sqrt{2}\ \sqrt{3},\frac{1}{2}\sqrt{2}\right)$$

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

$$.270099742$$

3.3.5 换元积分法和分部积分法

换元积分法是积分计算中一种重要而实用的方法. 在Maple中, 对被积函数施行变量代换的命令是changevar, 该命令在工具包student中, 须先调用student工具包. 命令格式为:

changevar(s, f);

changevar(s, f, u);

changevar(t, g ,v);

其中, s是形式为h(x)=g(u)的一个将x定义为u的函数的表达式, f为积分表达式(如Int(F(x),x=a..b);), u为新的积分变量名称, t为定义的多元变量代换的方程组, g为二重或者三重积分, v为新变量的列表.

Changevar函数对积分、求和或者极限实现变量代换. 第1个参数s是用旧变量定义新变量的一个方程, 如果包含了两个以上的变量, 新变量必须放置在第3个参数位置, 而第2个参数是一个要被替换的表达式, 一般包含Int, Sum或者Limit等非求值形式(应尽量使用这种形式以便最后用value求值).

当问题为二重或三重积分时, 定义多元变量代换的方程由一个集合给出, 而新变量由一个列表给出.

```
> with(student):
> changevar(cos(x)+1=u,Int((cos(x)+1)^3*sin(x),x),u);
```

$$\int -u^3\,du$$

```
> changevar(x=sin(u),Int(sqrt(1-x^2),x=a..b),u);
```

$$\int_{\arcsin(a)}^{\arcsin(b)} \sqrt{1-\sin(u)^2}\,\cos(u)\,du$$

```
> changevar({x=r*cos(t),y=r*sin(t)},Doubleint(1,x,y),[t,r]);
```

$$\int\int |r|\,dt\,dr$$

分部积分法(integration by parts)通过调用**student**工具包中的**intparts**来完成:

```
> with(student):
> int(x*exp(-a^2*x^2)*erf(b*x),x);
```

$$\int x\,\mathbf{e}^{(-a^2x^2)}\,\text{erf}(b\,x)\,dx$$

```
> intparts(%,erf(b*x));
```

$$-\frac{1}{2}\frac{\operatorname{erf}(b\,x)\,\mathbf{e}^{(-a^2x^2)}}{a^2}-\int-\frac{\mathbf{e}^{(-b^2x^2)}\,b\,\mathbf{e}^{(-a^2x^2)}}{\sqrt{\pi}\,a^2}\,dx$$

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

$$-\frac{1}{2}\frac{\operatorname{erf}(b\,x)\,\mathbf{e}^{(-a^2x^2)}}{a^2}+\frac{\frac{1}{2}\,b\,\operatorname{erf}(\sqrt{b^2+a^2}\,x)}{a^2\sqrt{b^2+a^2}}$$

3.3 重积分和线积分

在Maple中，重积分的形式函数有**Doubleint**(二重)和**Trippleint**(三重)，均在**student**工具包中，应用前需调用 student 工具包，它们适用于定积分和不定积分，可用 **value** 来获得积分结果的解析表达式. 命令格式为:

Doubleint(g, x, y);

Doubleint(g, x, y, Domain);

Doubleint(g, x = a..b , y = c..d);

Tripleint(g, x, y, z)

Tripleint(g, x, y, z, Domain)

Tripleint(g, x = a..b, z = e..f, y = c..d)

其中, g 为积分表达式, x, y, z 为积分变量, Domain 为积分区域.

```
> with(student):
> Doubleint(f(x,y),x,y);
```

$$\int\int \mathrm{f}(x,y)\,dx\,dy$$

比较以下两个实验:

```
> Doubleint(x+y,x=0..1,y=1..exp(x)):
  %=value(%);
```

$$\int_1^{\mathbf{e}^x}\int_0^1 x+y\,dx\,dy=\frac{1}{2}\,\mathbf{e}^x-1+\frac{1}{2}\,(\mathbf{e}^x)^2$$

```
> Doubleint(x+y,y=1..exp(x),x=0..1):
  %=value(%);
```

$$\int_0^1\int_1^{\mathbf{e}^x} x+y\,dy\,dx=-\frac{1}{4}+\frac{1}{4}\,(\mathbf{e})^2$$

在这两个形式函数中，我们还可以加入一个可选的参数，用来表示积分区域(通常用 S 表示二维区域，用Ω表示三维区域). 注意: 在 Maple 中，这个参数仅仅用来做形式上的表示，不可以用来求值.

```
> Tripleint(x^2*y^2*z^2,x,y,z,Omega);
```

$$\iiint_{\Omega} x^2\, y^2\, z^2\, dx\, dy\, dz$$

在 Maple 中还有一个计算用参数方程形式表示的第一型曲线积分的函数—Lineint, 它也在 student 工具包中. 下面通过一个实例说明这一函数的用法.

例: 求曲线积分 $\int_C y^2 ds$, 其中 C 为摆线 $x = a(t-\sin t), y = a(1-\cos t), 0 \le t \le 2\pi$ 的一拱.

```
> with(student):
Lineint(y^2,x=a*(t-sin(t)),y=a*(1-cos(t)),t=0..2*Pi);
value(%);
```

$$\int_0^{2\pi} a^2\,(1-\cos(t))^2 \sqrt{\left(\frac{\partial}{\partial t} a\,(1-\cos(t))\right)^2 + \left(\frac{\partial}{\partial t} a\,(t-\sin(t))\right)^2}\, dt$$

$$\frac{256}{15}\,\frac{a^4}{\sqrt{a^2}}$$

3.4 利用辅助手段积分

机器终归是机器, 再聪明的机器也无法彻底代替人脑, Maple 也一样, 它只能作为我们数学计算、推证的助手. 下面通过例子来体会 Maple 的真正用处.

例: 求广义积分 $\int_0^{\infty} e^{-cx^2} dx$, 其中 c>0.

按照常规, 我们会通过下面的语句进行计算:

```
> Int(exp(-c*x^2), x=0..infinity) = int(exp(-c*x^2), x=0..infinity);
```

但 Maple 告诉我们, 由于无法确定常数 *c* 的正负号, 因而无法确定积分是否收敛.

解决这一问题的办法是, 通过 assume 设定 c 的取值范围:

```
> assume(c>0);
> Int(exp(-c*x^2),x=0..infinity)=int(exp(-c*x^2),x=0..infinity);
```

$$\int_0^{\infty} \mathbf{e}^{(-c\sim x^2)}\, dx = \frac{1}{2}\,\frac{\sqrt{\pi}}{\sqrt{c\sim}}$$

解决这一问题的另一方法是假设 c 是另一个参数 p 的绝对值(**c:=abs(p)**), 这样 c 就自然是一个非负的参数了.

```
> with(student):
> int(x*exp(-a^2*x^2)*erf(b*x),x);
```

$$\int x\, e^{(-a^2 x^2)}\, \mathrm{erf}(b\,x)\, dx$$

```
> intparts(%,erf(b*x));
```

$$-\frac{1}{2}\frac{\mathrm{erf}(b\,x)\, e^{(-a^2 x^2)}}{a^2} - \int -\frac{e^{(-b^2 x^2)}\, b\, e^{(-a^2 x^2)}}{\sqrt{\pi}\, a^2}\, dx$$

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

$$-\frac{1}{2}\frac{\mathrm{erf}(b\,x)\, e^{(-a^2 x^2)}}{a^2} + \frac{\frac{1}{2}\, b\, \mathrm{erf}(\sqrt{b^2+a^2}\, x)}{a^2\sqrt{b^2+a^2}}$$

其中, erf(x)为误差函数(**error function**), 定义为: $erf(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$.

例: 求证 $\int_0^{2\pi}\frac{1}{1+3\sin t^2}dt=\pi$

在 Maple7 中，该积分直接利用 Maple 计算也可完成证明：

```
> Int(1/(1+3*sin(t)^2),t=0..2*Pi)=int(1/(1+3*sin(t)^2),t=0..2*Pi);
```

$$\int_0^{2\pi}\frac{1}{1+3\sin(t)^2}\,dt=\pi$$

这里，我们将其作为一个例子，试图说明应用有关数学理论知识转化问题，然后再利用 Maple 进行辅助计算的方法和技巧。

由复变函数理论知道，此积分可用围道积分(contour integration)来求解. 首先，我们把被积函数写成复变量 $z=e^{it}$ 的形式，然后把原问题转化成围道积分，再求得结果. 过程为:

```
> p:=1/(1+3*sin(t)^2);
```

$$p := \frac{1}{1+3\sin(t)^2}$$

```
> convert(p,exp);
```

$$\frac{1}{1-\frac{3}{4}\left(e^{(I\,t)}-\frac{1}{e^{(I\,t)}}\right)^2}$$

```
> factor(%)/diff(exp(I*t),t);
```

$$\frac{4\,I\,\mathbf{e}^{(I\,t)}}{(3\,(\mathbf{e}^{(I\,t)})^{2}-1)\,((\mathbf{e}^{(I\,t)})^{2}-3)}$$

```
> g:=subs(exp(I*t)=Z,%);
```

$$g := \frac{4\,I\,Z}{(3\,Z^{2}-1)\,(Z^{2}-3)}$$

```
> solve(denom(g)=0,Z);
```

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

```
> readlib(residue):
> residue(g,Z=1/3*sqrt(3));
```

$$\frac{-1}{4}\,I$$

```
> residue(g,Z=-1/3*sqrt(3));
```

$$\frac{-1}{4}\,I$$

```
> 2*pi*I*(%+%%);
```

$$\pi$$

其中，函数residue(f, x=a) 计算表达式f对变量x在a点附近的代数残差(algebraic residue)，残差定义为f的Laurent级数中(x-a)^(-1)的系数.

4 级　数

4.1 数值级数和函数项级数求和以及审敛法

我们可以用 Maple 中的函数 **sum** 方便地求得级数的和，无论是有限项还是无穷项，常数项级数还是函数项级数．相应地，和式的形式函数是 Sum. 求连乘积使用命令 **product**.

```
> Sum(1/(4*k^2-1), k=1..infinity)= sum(1/(4*k^2-1), k=1..infinity);
```

$$\sum_{k=1}^{\infty}\frac{1}{4\,k^{2}-1}=\frac{1}{2}$$

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

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

```
> Product(1/k^2,k=1..n)=product(1/k^2,k=1..n);
```

$$\prod_{k=1}^{n}\frac{1}{k^2}=\frac{1}{\Gamma(n+1)^2}$$

Maple 对级数求和的方法如下:

(i) 多项式级数求和用贝努利级数公式: $\sum_{k=0}^{n-1}k^m=\frac{1}{m+1}C_{m+1}^{k}B_k n^{m+1-k}$，其中，贝努利数$B_k$由以下隐式递推公式定义: $B_0=1,\sum_{k=0}^{m}C_{m+1}^{k}B_k=0$

(ii) 有理函数级数求和是用 Moenck 方法，得到的结果是一个有理函数加伽玛函数(Polygamma function)ψ 及其导数的项;

(iii) Gosper 算法是 Risch 算法的离散形式，它被用到求包含级乘和乘幂的级数和上;

(iv) 计算无穷项级数的和有时会用到超比级数.

收敛或发散是级数的重要性质，在这里主要以绝对收敛的比值审敛法(ratio test for absolute convergence)为例说明 Maple 的使用，其余类推.

绝对收敛的比值审敛法的数学原理是:

设$\sum a_k$ 为不含 0 的交错级数，并令$\rho=\lim_{k\to\infty}\frac{|a_{k+1}|}{|a_k|}$，则:

1） 若ρ <0，级数绝对收敛;

2） 若ρ >0 或$\rho=\infty$，级数发散;

3） 若ρ =1，待定

例: 判定级数$\sum_{k=1}^{\infty}\frac{(-1)^k 2^k}{k!}$是否绝对收敛.

```
> f:=k->(-1)^k*12^k/(k!);
```

$$f := k \to \frac{(-1)^k\, 12^k}{k!}$$

```
> r:=simplify(abs(f(k+1))/abs(f(k)));
```

$$r := 12\,\frac{1}{|k+1|}$$

```
> Limit(r,k=infinity);
```

$$\lim_{k\to\infty}\ 12\,\frac{1}{|k+1|}$$

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

$$0$$

由此可见，r=0<1，级数绝对收敛. 事实上，还可以用 sum 对级数求和，确定级数收敛于一个定值:

```
> sum(f(k),k=1..infinity);
```

$$e^{(-12)}\,(1 - e^{12})$$

4.2 幂级数

幂级数的有关计算在专门的工具包**powseries**中，这个工具包含有生成和处理幂级数的各种常用工具. 如果我们已知一个幂级数的系数，就可以用函数**powcreate**来生成它，其参数是系数所满足的方程或方程组，格式为:

```
> with(powseries);
> powcreate(t(n)=3^sqrt(n));
```

没有任何结果显示出来，事实上，此时, Maple 已经按照要求把该幂级数的系数函数赋给了 t(n)，用下述命令即可看出:

```
> t(2);
```

$$3^{(\sqrt{2})}$$

显然，这样的级数很不直观，更多的时候我们需要幂级数的截断表达式(truncated power series form)，此时可以通过该工具包中的 **tpsform** 命令完成，这是一个很有用的 Maple 函数:

```
> tpsform(t,x,8);
```

$$1 + 3\,x + 3^{(\sqrt{2})}\,x^2 + 3^{(\sqrt{3})}\,x^3 + 3^{(\sqrt{4})}\,x^4 + 3^{(\sqrt{5})}\,x^5 + 3^{(\sqrt{6})}\,x^6 + 3^{(\sqrt{7})}\,x^7 + O(x^8)$$

但是，大多数情况下，我们并不知道幂级数的系数，而只知道幂级数的和的解析表达式，需要把它展开成和式. 对于一些常用的函数，**powseries** 工具包中有一些函数可以生成对应的幂级数，比如 sin(p)、cos(p)、exp(p)、ln(p)、sqrt(p)的幂级数可以分别通过 powsin(p)、powcos(p)、powexp(p)、powlog(p)、powsqrt(p)来得到，其中, p 可以是单变量函数或表达式甚至级数.

```
> t:=powlog(1+x+x^2):  tpsform(t,x,6);
```

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

对于多项式，可以用 **powpoly(expr, x)** 得到对应的幂级数，其中 **expr** 是多项式, x 是变量. 而任意函数的表达式可以通过函数 **evalpow** 来获得其幂级数形式.

```
> t:=evalpow(1/(1-3*x+2*x^2)):
> tpsform(t,x,6);
```

$$1 + 3\,x + 7\,x^2 + 15\,x^3 + 31\,x^4 + 63\,x^5 + O(x^6)$$

掌握了级数的有关生成后，可以对这些级数进行运算了，**powseries** 工具包中具有对幂级数的各种运算：加(**powadd**)、减(**subtract**)、乘(**multiply**)、除(**quotient**)、求负

(**negative**)、求倒数(**inverse**)、复合(**compose**)、求逆(**reversion**)、求导(**powdiff**)、积分(**powint**)等. 下面通过实例学习这些运算.

```
> restart:with(powseries):
powcreate(t(n)=t(n-1)/n,t(0)=1):
powcreate(v(n)=v(n-1)/2,v(0)=1):
s:= powadd(t, v):  tpsform(s, x, 7);
```

$$2+\frac{3}{2}x+\frac{3}{4}x^2+\frac{7}{24}x^3+\frac{5}{48}x^4+\frac{19}{480}x^5+\frac{49}{2880}x^6+O(x^7)$$

```
> p:=multiply(t,v): tpsform(p,x,7);
```

$$1+\frac{3}{2}x+\frac{5}{4}x^2+\frac{19}{24}x^3+\frac{7}{16}x^4+\frac{109}{480}x^5+\frac{331}{2880}x^6+O(x^7)$$

```
> q:=quotient(t,v):  tpsform(q,x,7);
```

$$1+\frac{1}{2}x-\frac{1}{12}x^3-\frac{1}{24}x^4-\frac{1}{80}x^5-\frac{1}{360}x^6+O(x^7)$$

```
> w:=powdiff(t):  tpsform(u,x,6);
```

$$1+x-2x^2+x^3+x^4-2x^5+O(x^6)$$

```
> u:=powint(v):  tpsform(u,x,6);
```

$$x+\frac{1}{4}x^2+\frac{1}{12}x^3+\frac{1}{32}x^4+\frac{1}{80}x^5+O(x^6)$$

4.3 泰勒级数和劳朗级数

在 Maple 中，可以用命令 **taylor** 方便快捷地得到一个函数或表达式在一点的任意阶 Tayloe 展开式，而一般级数展开命令为 series. 命令格式为:

taylor(expr,eqn/nm,n);

series(expr, eqn, n);

其中, expr 表示表达式, eqn/nm 表示方程(如 x=a)或名称(如 x), n(非负整数)表示展开阶数.

在调用 taylor 或 **series** 级数时，只需要指定有待展开的表达式、展开点、展开的阶数就可以了. 如果不给定展开点，默认为 0 点，如果不指定展开阶数时默认为 6 阶(命令 **order** 可获取截断级数的展开阶数). 另外，series 函数可以展开更一般的截断函数，比如 laurent 级数等，它会根据情况决定展开成什么类型级数.

```
> taylor(sin(tan(x))-tan(sin(x)),x=0,19);
```

$$-\frac{1}{30}x^7-\frac{29}{756}x^9-\frac{1913}{75600}x^{11}-\frac{95}{7392}x^{13}-\frac{311148869}{54486432000}x^{15}+O(x^{17})$$

```
> series(GAMMA(x),x=0,3);
```

$$x^{-1}-\gamma+\left(\frac{1}{12}\pi^2+\frac{1}{2}\gamma^2\right)x+\left(-\frac{1}{3}\zeta(3)-\frac{1}{12}\pi^2\gamma-\frac{1}{6}\gamma^3\right)x^2+O(x^3)$$

```
> order(%);
```

3

如果需要限制展开为 lauren 级数，可以使用 numapprox 工具包中的 laurent 函数，否则, series 有可能会得出一些更为广义的级数，比如 Puisseux 级数:

```
> with(numapprox);
> laurent(exp(x), x=0,10);
```

$$1+x+\frac{1}{2}x^2+\frac{1}{6}x^3+\frac{1}{24}x^4+\frac{1}{120}x^5+\frac{1}{720}x^6+\frac{1}{5040}x^7+\frac{1}{40320}x^8+\frac{1}{362880}x^9+O(x^{10})$$

```
> series(1/(x*(1+sqrt(x))),x=0,3);
```

$$\frac{1}{x}-\frac{1}{\sqrt{x}}+1-\sqrt{x}+x-x^{(3/2)}+x^2-x^{(5/2)}+O(x^3)$$

Maple 在计算截断级数上还具有一个有用的特性，即待展开的表达式或函数不一定要有解析表达式，而只要可以求导就能计算.

```
> Int(exp(x^3),x)=int(exp(x^3),x);
> series(rhs(%),x=0);
```

$$\int \mathbf{e}^{(x^3)}\,dx=-\frac{1}{3}(-1)^{(2/3)}\left(\frac{2}{3}\frac{x\,(-1)^{(1/3)}\pi\sqrt{3}}{\Gamma\left(\frac{2}{3}\right)(-x^3)^{(1/3)}}-\frac{x\,(-1)^{(1/3)}\Gamma\left(\frac{1}{3},-x^3\right)}{(-x^3)^{(1/3)}}\right)$$

$$x+\frac{1}{4}x^4+O(x^7)$$

与前面介绍的幂级数不同，这里可以直接对截断形式的级数进行求导和积分运算:

```
> sin_series:=series(sin(x),x);
```

$$sin\_series := x-\frac{1}{6}x^3+\frac{1}{120}x^5+O(x^6)$$

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

$$1-\frac{1}{2}x^2+\frac{1}{24}x^4+O(x^5)$$

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

$$x-\frac{1}{6}x^3+\frac{1}{120}x^5+O(x^6)$$

关于多元级数需调用函数 **mtaylor**，结果是一般的多项式而非截断函数，自然就可以进行多项式的运算了. 命令格式为:

mtaylor(f,v);

mtaylor(f,v,n);

mtaylor(f,v,n,w);

其中, f 表示代数表达式, v 表示方程的列表或集合, n(可选，默认值为 6)表示截断阶数的非负整数, w(可选)表示自然数列表，权变量列表.

```
> mtaylor(sin(x^2+y^2),[x,y],8);
```

$$x^2+y^2-\frac{1}{6}x^6-\frac{1}{2}y^2x^4-\frac{1}{2}y^4x^2-\frac{1}{6}y^6$$

```
> whattype(%);
```

$$+$$

更进一步，可以用**normal**函数对展开式进行正则化(其更主要的功能是对有理函数化简)，用**coeff**获得展开式的系数，与多项式不同的是，对于级数，**coeff**只能获得主变量的幂系数．实际上，更为方便的是，可以利用库函数**coeftayl**，不展开就能获得泰勒级数的系数，计算***f***在**x=x$_0$**点的泰勒展开式中$(x-x_0)^k$的系数的命令格式为：

coeftayl (f, x=x$_0$, k);

```
> normal((f(x)^2-1)/(f(x)-1));
```

$$f(x)+1$$

```
> normal(sin(x*(x+1)-x));
```

$$\sin(x^2)$$

```
> normal(1/x+x/(x+1),expanded);
```

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

```
> coeftayl(series(sin(x),x=0,20),x=0,11);
```

$$\frac{-1}{39916800}$$

```
> coeftayl(series(exp(x),x=0,30),x=1,13);
```

$$\frac{56874039553217}{130286647826605670400000}$$

5 积分变换

无论在数学理论研究还是在数学应用中，积分变换都是一种非常有用的工具．积分变换就是将一个函数通过参变量积分变为另一个函数．常用的积分变换包括拉普拉斯变换(**Laplace transforms**)、傅里叶变换(**Fourier transforms**)、梅林变换(**Melin transforms**)以及汉克尔变换(Hankel transforms)等．函数 f 的积分变换的定义如下:

$$T(f)(s)=\int_a^b f(t)K(s,t)dt$$

其中 ***K*** 为变换核．具体如下:

变换名称	定　义	变换命令	逆变换命令
拉普拉斯	$\int_0^\infty f(t)e^{-st}dt$	**laplace(f(t), t, s)**	**invlaplace(f(t), t, s)**

傅 里 叶	$\int_{-\infty}^{+\infty} f(t)e^{-ist}dt$	**fourier(f(t), t,s)**	**invfourier(f(t), t,s)**
梅 林	$\int_0^{\infty} f(t)e^{s-1}dt$	**melin(f(t), t,s)**	**invmelin(f(t), t,s)**

这些函数都在 **inttrans** 工具包中，使用时用 **with** 引入.

5.1 拉普拉斯变换

形如多项式或者一些特定函数(如 Diract function、Heaviside function、Bessel function 等)组成的有理表达式或它们的和，都可以用 Maple 积分变换工具包 **inttrans** 中的函数 **laplace** 求得其拉普拉斯变换，相应的逆变换用 **invlaplace**.

```
> t^2-exp(t)+sin(a*t);
```

$$t^2 - \mathbf{e}^t + \sin(a\,t)$$

```
> with(inttrans):
> laplace(%%,t,s);
```

$$2\,\frac{1}{s^3} - \frac{1}{s-1} + \frac{a}{s^2+a^2}$$

```
> invlaplace(%,s,t);
```

$$t^2 - \mathbf{e}^t + \sin(a\,t)$$

所有的积分变换都主要应用于微分和积分方程领域。由于拉普拉斯变换的导数和积分性质，它被大量地应用到微分方程和积分方程的求解上. 作为拉普拉斯变换的一个应用，下面求解 Volterra 积分方程:

```
> with(inttrans):
> e:=u(t)+int((t-x)*u(x),x=0..t)-cos(t);
```

$$e := \mathrm{u}(t) + \int_0^t (t-x)\,\mathrm{u}(x)\,dx - \cos(t)$$

```
> laplace(e,t,s);
```

$$\mathrm{laplace}(\mathrm{u}(t), t, s) + \frac{\mathrm{laplace}(\mathrm{u}(t), t, s)}{s^2} - \frac{s}{s^2+1}$$

下面要从这个简单的代数方程中解出 **laplace(e,t,s)**来，值得注意的是要解出来的不是一个变量，而是一个子式。当然，我们可以先用 **subs** 将这个子式代换为一个变量，再用 **solve** 求解。但在 Maple 中有一个更方便的命令：库函数 **isolate** 可直接从方程是解出这个子式来：

```
> readlib(isolate)(%,laplace(e,t,s));
```

$$\text{laplace}(\text{u}(t), t, s) + \frac{\text{laplace}(\text{u}(t), t, s)}{s^2} - \frac{s}{s^2+1} = 0$$

然后求解变换后的结果 laplace(u(t),t,s)：

```
> solve(%,laplace(u(t),t,s));
```

$$\frac{s^3}{s^4 + 2\,s^2 + 1}$$

作反变换即可得到 u(t)，也就是原方程的解：

```
> u(t):=invlaplace(%,s,t);
```

$$\text{u}(t) := \left(-\frac{1}{2}\,t\,\sin(t) + \cos(t)\right)\text{Dirac}(t)$$

再看另外一类积分方程的 Laplace 变换求解：

```
> int_eqn:=int(exp(a*x)*f(t-x),x=0..t)+b*f(t)=t;
```

$$int\_eqn := \int_0^t \mathbf{e}^{(a\,x)}\,\text{f}(t-x)\,dx + b\,\text{f}(t) = t$$

对其进行拉普拉斯变换，得到 f(t)的变换 L(f)(s)的方程：

```
> laplace(%,t,s);
```

$$\frac{\text{laplace}(\text{f}(t), t, s)}{s-a} + b\,\text{laplace}(\text{f}(t), t, s) = \frac{1}{s^2}$$

```
> readlib(isolate)(%,laplace(f(t),t,s));
```

$$\text{laplace}(\text{f}(t), t, s) = \frac{1}{s^2\left(\frac{1}{s-a} + b\right)}$$

再对上式做逆变换，即可得所求：

```
> invlaplace(%,s,t);
```

$$\text{f}(t) = \frac{a\,t}{-1+b\,a} - \frac{2\,\sinh\left(\frac{1}{2}\,\frac{(-1+b\,a)\,t}{b}\right)\mathbf{e}^{\left(1/2\,\frac{(-1+b\,a)\,t}{b}\right)}}{(-1+b\,a)^2}$$

5.2 傅里叶变换

Maple 工具包 **inttrans** 中的函数 **fourier** 可以求表达式的傅里叶变换:

```
> with(inttrans):
> 1/(1+t^2);
```

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

```
> fourier(%,t,omega);
```

$$e^{\omega}\,\pi\,\text{Heaviside}(-\omega)+e^{(-\omega)}\,\pi\,\text{Heaviside}(\omega)$$

上式的结果中出现了 **Heaviside** 函数，它的导数是 **Diract** 函数，它在负半轴上取 0，在正半轴上取 1. 一般地，可对其做逆变换:

```
> invfourier(%,omega,t);
```

$$-\frac{1}{(1+I\,t)\,(-1+I\,t)}$$

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

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

对于一些函数，比如三角函数、Diract 函数、Heaviside 函数和 Bessel 函数等, Maple 会利用卷积定理、查表、定积分等方法求它们的傅里叶变换:

```
> fourier(BesselJ(0,t),t,omega);
```

$$2\,\frac{-\text{Heaviside}(\omega-1)+\text{Heaviside}(\omega+1)}{\sqrt{1-\omega^2}}$$

我们甚至可以扩充Maple里的傅里叶变换表，尽管Maple内部的公式表可能比任何一本教科书都要完全. 例如，定义函数f(t)的傅里叶变换为F(s)/(1+s^2)，然后利用Maple中的函数**addtable**将其加到变换表中去:

```
> addtable(fourier,f(t),F(s)/(1+s^2),t,s);
> fourier(exp(3*I*t)*f(2*t),t,omega);
```

$$\frac{1}{2}\,\frac{F\left(\frac{1}{2}\omega-\frac{3}{2}\right)}{1+\frac{1}{4}(\omega-3)^2}$$

addtable 是 **inttrans** 中的函数，它不仅可以用来扩充傅里叶变换的变换表，对其他的积分也适用. 其第一个参数是积分变换名，第二个是函数，第三个是该函数经过变换后的函数，后两个参数是变换前后函数的自变量.

函数 **fourier** 和 **invfourier** 是用来计算符号表达式的连续傅里叶变换的，对于离散域上的数值傅里叶变换，可以调用快速傅里叶变换(Fast Fourier Transformation)函数 **FFT**，相应的逆变换函数是 **iFFT**.

对于一个长度为N的离散数值向量x=[x_0，x_1，……，x_{N-1}]，其傅里叶变换X=[X_0，X_1，……，X_{N-1}]的定义如下:

$$X_k = \sum_{j=1}^{N-1} x_j e^{\frac{2\pi i j k}{N}}$$

其中，$0 \le k \le N-1$. 快速傅里叶变换的算法决定了 N 必须为 2 的整数次幂. 作为例子，我们取 $N = 2^3$ 来计算实数序列[1, 1, 1, 1, -1, -1, -1, -1]的傅里叶变换. 和 **fourier** 不同, **FFT** 与 **iFFT** 都是函数 (Maple 7 以前的版本是库函数)，而不是 **inttrans** 工具包中的函数.

FFT 是用来对复数序列进行傅里叶变换的，需要分别提供数据的实部和虚部.

```
> x:=array([1,1,1,1,-1,-1,-1,-1]);
```

$$x := [1, 1, 1, 1, -1, -1, -1, -1]$$

```
> y:=array([0,0,0,0,0,0,0,0]);
```

$$y := [0, 0, 0, 0, 0, 0, 0, 0]$$

```
> FFT(3,x,y);
```

8

```
> print(x);
```

[0, 2.000000001, 0., 1.999999999, 0, 1.999999999, 0., 2.000000001]

```
> print(y);
```

[0, -4.828427122, 0., -.828427124, 0, .828427124, 0., 4.828427122]

FFT的第一个参数表明了序列的元素个数，上面的 3 表示要换的数是 2^3个，后两上参数分别是变换的实部和虚部，它们的数据类型都是数组(**array**)，可以用函数**array**从数据列表生成. 有关数组的显示用**print**命令. 从上面的结果可以看出，FFT把变换的结果直接赋给了输入的数组，而不是把结果作为返回值返回.

再用 **iFFT** 检验变换的正确性:

```
> iFFT(3,x,y);
```

8

```
> print(x);
```

[1.000000000, .9999999990, .9999999995, .9999999985, -1.000000000, -.9999999990, -.9999999995, -.9999999985]

```
> print(y);
```

[0., -.2500000000 10^{-9}, 0., .2500000000 10^{-9}, 0., .2500000000 10^{-9}, 0., -.2500000000 10^{-9}]

这里的结果是将实部与虚部分开的，我们可以用 zip 函数把他们合成为复数形式:

```
> zip((a,b)->a+b*I,x,y);
```

[1.000000000 + 0. *I*, .9999999990 − .2500000000 10^{-9} *I*, .9999999995 + 0. *I*, .9999999985 + .2500000000 10^{-9} *I*,
-1.000000000 + 0. *I*, -.9999999990 + .2500000000 10^{-9} *I*, -.9999999995 + 0. *I*, -.9999999985 − .2500000000 10^{-9} *I*]

zip 的第一个参数是一个二元函数，它将后两个向量中的每一个元素按此函数结合成一个新的向量.

5.4 其他积分变换

在 Maple 中还有一些其他的积分变换，它们的调用格式和前面的基本一样.

变　换	定　义	Maple 中的函数
傅里叶余弦变换	$\sqrt{\frac{2}{\pi}}\int_0^{\infty} f(t)\cos(st)dt$	**fourierscos(f(t),t,s)**
傅里叶正弦变换	$\sqrt{\frac{2}{\pi}}\int_0^{\infty} f(t)\sin(st)dt$	**fourierssin(f(t),t,s)**
汉克尔变换	$\int_0^{\infty} f(t)\sqrt{st}BesslJ(v,st)dt$	**hankel(f(t), t, s, nu)**
希尔伯特变换	$\frac{1}{\pi}\int_{-\infty}^{+\infty}\frac{f(t)}{t-s}dt$	**hilbert(f(t), t, s)**

下面通过几个例子说明这些变换的应用：

① Hankel 变换：

```
> with(inttrans):
> assume(k,integer,k>0):
> hankel(sqrt(t^2),t,s,k);
```

$$2\,\frac{\sqrt{2}\;\Gamma\left(\frac{1}{2}k+\frac{5}{4}\right)}{s^2\,\Gamma\left(\frac{1}{2}k-\frac{1}{4}\right)}$$

② Fourier正余弦变换

```
> assume(a>0):
> fouriercos(Heaviside(a-t),t,s);
```

$$\frac{\sqrt{2}\;\sin(a\,s)}{\sqrt{\pi}\;s}$$

```
> fouriercos(%,s,t);
```

$$\text{Heaviside}(a-t)$$

```
> fouriersin(Heaviside(a-t),t,s);
```

$$2\,\frac{\left(\sqrt{2}\,\sin\left(\frac{1}{2}\,a\,s\right)\right)^2}{\sqrt{\pi}\,s}$$

③ Hilbert 变换

```
> assume(k,integer,k>0):
> hilbert(Dirac(x)+sin(k*x)/x,x,y);
```

$$\frac{-1+\pi\cos(y\,k)-\pi}{y\,\pi}$$

④ Mellin 变换

```
> mellin(1/(1+t),t,s);
```

$$\pi\csc(\pi\,s)$$

```
> mellin(ln(1+t),t,s);
```

$$\frac{\pi\csc(\pi\,s)}{s}$$

第四章 方程求解

1 代数方程(组)求解

1.1 常用求解工具—solve

求解代数方程或代数方程组，使用 Maple 中的 **solve** 函数. 求解关于 x 的方程 eqn=0 的命令格式为：

solve(eqn, x);

求解关于变量组 vars 的方程组 eqns 的命令为：

solve(eqns, vars);

```
> eqn:=(x^2+x+2)*(x-1);
```

$$eqn := (x^2 + x + 2)(x - 1)$$

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

$$1, -\frac{1}{2} + \frac{1}{2} I\sqrt{7}, -\frac{1}{2} - \frac{1}{2} I\sqrt{7}$$

当然, solve 也可以求解含有未知参数的方程：

```
> eqn:=2*x^2-5*a*x=1;
```

$$eqn := 2x^2 - 5ax = 1$$

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

$$\frac{5}{4}a + \frac{1}{4}\sqrt{25a^2 + 8}, \frac{5}{4}a - \frac{1}{4}\sqrt{25a^2 + 8}$$

solve 函数的第一个参数是有待求解的方程或方程的集合，当然也可以是单个表达式或者表达式的集合，如下例：

```
> solve(a+ln(x-3)-ln(x),x);
```

$$3\frac{\mathbf{e}^a}{-1 + \mathbf{e}^a}$$

对于第二个参数, Maple 的标准形式是未知变量或者变量集合，当其被省略时，函数 **indets** 自动获取未知变量. 但当方程中含有参数时，则会出现一些意想不到的情况：

```
> solve(a+ln(x-3)-ln(x));
```

$$\{x = x,\ a = -\ln(x-3) + \ln(x)\}$$

很多情况下，我们知道一类方程或方程组有解，但却没有解决这类方程的一般解法，或者说没有解析解．比如，一般的五次或五次以上的多项式，其解不能写成解析表达式．Maple 具备用所有一般算法尝试所遇到的问题，在找不到解的时候，Maple 会用 RootOf 给出形式解．

```
> x^7-2*x^6-4*x^5-x^3+x^2+6*x+4;
```

$$x^7 - 2x^6 - 4x^5 - x^3 + x^2 + 6x + 4$$

```
> solve(%);
```

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

```
> solve(cos(x)=x,x);
```

$$\text{RootOf}(\_Z - \cos(\_Z))$$

对于方程组解的个数可用 **nops** 命令获得，如：

```
> eqns:={seq(x[i]^2=x[i],i=1..7)};
```

$$eqns := \{x_1^{\,2} = x_1,\ x_2^{\,2} = x_2,\ x_3^{\,2} = x_3,\ x_4^{\,2} = x_4,\ x_5^{\,2} = x_5,\ x_6^{\,2} = x_6,\ x_7^{\,2} = x_7\}$$

```
> nops({solve(eqns)});
```

$$128$$

但是，有时候，Maple 甚至对一些“显而易见”的结果置之不理，如：

```
> solve(sin(x)=3*x/Pi,x);
```

$$\text{RootOf}(3\,\_Z - \sin(\_Z)\,\pi)$$

此方程的解为 $\pm\frac{\pi}{6}$，0，但 Maple 却对这个超越方程无能为力，即便使用 **allvalues** 求解也只有下述结果：

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

$$\text{RootOf}(3\,\_Z - \sin(\_Z)\,\pi,\ 0.)$$

另外一个问题是，Maple 在求解方程之前,会对所有的方程或表达式进行化简，而不管表达式的类型，由此而产生一些低级的错误：

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

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

```
> solve(%);
```

$$1$$

但是，大量实验表明，**solve** 的确是一个实用的方程求解工具，但是也不可盲目相信它给出的一切结果，特别是对于非线性方程而言，对于给出的结果需要加以验证.

下面通过几个例子说明在 Maple 中非线性方程组的求解问题.

例：求解方程组：$\begin{cases} x^2+y^2=25 \\ x^2-9=y \end{cases}$

```
> eqns:={x^2+y^2=25,y=x^2-5};
```

$$eqns := \{ y = x^2 - 5, x^2 + y^2 = 25 \}$$

```
> vars:={x,y};
```

$$vars := \{ x, y \}$$

```
> solve(eqns,vars);
```

$$\{ x = 0, y = -5 \}, \{ x = 0, y = -5 \}, \{ y = 4, x = 3 \}, \{ y = 4, x = -3 \}$$

也可用下面的语句一步求出：

```
> solve({x^2+y^2=25,y=x^2-5},{x,y});
```

$$\{ x = 0, y = -5 \}, \{ x = 0, y = -5 \}, \{ y = 4, x = 3 \}, \{ y = 4, x = -3 \}$$

这个问题非常简单，但通常遇到的非线性问题却不是这么简单，例如要求解方程组：$x^2+y^2=1, \sqrt{x+y}=x-y$

```
> eqns:={x^2+y^2=1,sqrt(x+y)=x-y};
vars:={x,y};
```

$$eqns := \{ x^2 + y^2 = 1, \sqrt{x+y} = x - y \}$$

$$vars := \{ x, y \}$$

```
> sols:=solve(eqns,vars);
```

$$sols := \{y = \text{RootOf}(2\_Z^2 + 4\_Z + 3, -1.000000000 - .7071067812\ I),$$
$$x = -\text{RootOf}(2\_Z^2 + 4\_Z + 3, -1.000000000 - .7071067812\ I) - 2\}, \{x = 1, y = 0\}$$

可以看出，方程解的形式是以集合的序列给出的，序列中的每一个集合是方程的一组解，这样就很利于我们用 **subs** 把解代入原方程组进行检验：

```
> subs(sols[2],eqns);
```

$$\{ 1 = 1 \}$$

```
> sols2:=allvalues(sols[1]);
```

$$sols2 := \{ x = -1 + \frac{1}{2} I \sqrt{2}, y = -1 - \frac{1}{2} I \sqrt{2} \}$$

```
> simplify(subs(sols2,eqns));
```

$$\{ I\sqrt{2} = I\sqrt{2}, 1 = 1 \}$$

1.2 其他求解工具

1.2.1 数值求解

对于求代数方程的数值解问题, Maple 提供了函数 **fsolve**, **fsolve** 的使用方法和 **solve** 很相似:

fsolve(eqns, vars, options);

其中，eqns 表示一个方程、方程组或者一个程序，vars 表示一个未知量或者未知量集合，options 控制解的参数(诸如：complex: 复根; maxsols=n：只找到 n 阶最小根; intervals：在给定闭区间内求根，等).

```
> fsolve(x^5-x+1,x);
```

-1.167303978

```
> fsolve(x^5-x+1,x,complex);
```

-1.167303978 , -.1812324445 − 1.083954101 *I*, -.1812324445 + 1.083954101 *I*, .7648844336 − .3524715460 *I*, .7648844336 + .3524715460 *I*

```
> fsolve(x^3-3*x+1,x,0..1);
```

.3472963553

对于多项式方程, **fsolve** 在默认情况下以给出所有的实数解，如果附加参数 **complex**, 就可以给出所有的解. 但对于更一般的其他形式的方程，**fsolve** 却往往只满足于得到一个解：

```
> eqn:=sin(x)=x/2;
```

$$eqn := \sin(x) = \frac{1}{2}x$$

```
> fsolve(eqn);
```

0.

```
> fsolve(eqn,x,0.1..infinity);
```

1.895494267

```
> fsolve(eqn,x,-infinity..-0.1);
```

-1.895494267

函数 **fsolve** 主要基于两个算法，通常使用牛顿法，如果牛顿法无效，它就改而使用切线法. 为了使 **fsolve** 可以求得所有的实根，我们通常需要确定这些根所在的区间. 对于单变量多项式，函数 **realroot** 可以获得多项式的所有实根所在的区间.

```
> 4+6*x+x^2-x^3-4*x^5-2*x^6+x^7;
```

$$4 + 6x + x^2 - x^3 - 4x^5 - 2x^6 + x^7$$

```
> realroot(%);
```

$$[[0, 2], [2, 4], [-2, -1]]$$

函数 **realroot** 还有一个可选参数，它是用来限制区间的最大长度的，为了保证使用数值求解方法时收敛，我们可以用它限制区间的最大长度：

```
> realroot(%%,1/1000);
```

$$\left[\left[\frac{1195}{1024}, \frac{299}{256}\right], \left[\frac{3313}{1024}, \frac{1657}{512}\right], \left[\frac{-633}{512}, \frac{-1265}{1024}\right]\right]$$

求解方程或方程组的整数解时使用函数**isolve**，它常常被用来求解不定方程．例如著名的“百钱买百鸡”问题♣的求解过程为：

```
> isolve({x+y+z=100,5*x+3*y+z/3=100});
```

$$\{z = 75 + 3\ \_Z1, x = 4\ \_Z1, y = 25 - 7\ \_Z1\}$$

据此可得满足该问题的三组解为：

$$\{x, y, z\}=\{4, 18, 78\}, \{x, y, z\}=\{8, 11, 81\}, \{x, y, z\}=\{12, 4, 84\}$$

1.2.2 整数环中的方程(组)求解

利用 Maple 中的函数 **msolve(eqns, vars, n)**，可以在模 ***n*** 的整数环中求解方程(组)eqns.

例：在Z_7中求解**Pell**方程 $y^7 = x^3 - 28$

```
> msolve(y^7=x^3-28,7);
```

$$\{x = 3, y = 6\}, \{x = 4, y = 1\}, \{y = 0, x = 0\}, \{x = 1, y = 1\}, \{y = 6, x = 6\},$$
$$\{x = 2, y = 1\}, \{y = 6, x = 5\}$$

再如下例：

```
> msolve(y^4=x^3+32,5);
```

$$\{x = 2, y = 0\}, \{x = 4, y = 1\}, \{x = 4, y = 2\}, \{x = 4, y = 3\}, \{x = 4, y = 4\}$$

1.2.3 递归方程的求解

在 Maple 中，可以求解有限差分方程(也称递归方程)，所需调用的函数是 **rsolve**，该函数使用的是一些比较通用的方法，例如产生函数法、z 变换法以及一些基于变量替换和特征方程的方法．作为例子，求解 Fibonacci 多项式:

```
> eq:=f(n)=f(n-1)+2*f(n-2);
```

$$eq := f(n) = f(n-1) + 2\,f(n-2)$$

```
> rsolve({eq,f(0)=1,f(1)=1},f(n));
```

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

当然，并不是所有的递归形式的函数方程的解可以写成解析形式，如果不能，Maple 将保留原来的调用形式．此时，可用 **asympt** 函数获得它的渐进表达式，也就是 1/n 的级数解．例如，对于一个具有超越形式的递归函数方程，仍然可以得到解的渐进形式：

♣ **百钱买百鸡问题**：用 100 元钱买 100 只鸡，大公鸡 5 元钱 1 只，大母鸡 3 元钱 1 只，小鸡 1 元钱 3 只，问如何买法？

```
> rsolve(u(n+1)=ln(u(n)+1),u(n));
```

$$\text{rsolve}(u(n+1)=\ln(u(n)+1),u(n))$$

```
> asympt(%,n,5);
```

$$2\,\frac{1}{n}+\frac{\_C+\frac{2}{3}\ln(n)}{n^2}+\frac{\frac{1}{9}-\frac{1}{3}\_C+\frac{1}{2}\_C^2-\left(-\frac{2}{3}\_C+\frac{2}{9}\right)\ln(n)+\frac{2}{9}\ln(n)^2}{n^3}+O\left(\frac{1}{n^4}\right)$$

1.2.4 不等式(组)求解

求解一元不等式方程(组)使用命令**solve**：

```
> solve((x-1)*(x-2)*(x-3)<0,x);
```

$$\text{RealRange}(-\infty,\text{Open}(1)),\text{RealRange}(\text{Open}(2),\text{Open}(3))$$

```
> solve((x-1+a)*(x-2+a)*(x-3+a) < 0, {x});
```

$$\{x<1-a\},\{2-a<x,x<3-a\}$$

```
> solve(exp(x)>x+1);
```

$$\text{RealRange}(-\infty,\text{Open}(0)),\text{RealRange}(\text{Open}(0),\infty)$$

```
> solve({x^2*y^2=0,x-y=1,x<>0});
```

$$\{y=0,x=1\},\{y=0,x=1\}$$

对于由不等式方程组约束的最优问题的求解使用“线性规则”工具包**simplex**:

```
> with(simplex):
> cnsts:={3*x+4*y-3*z<=23, 5*x-4*y-3*z<=10,7*x+4*y+11*z<=30};
```

$$cnsts := \{3x+4y-3z\le 23,5x-4y-3z\le 10,7x+4y+11z\le 30\}$$

```
> obj:=-x+y+2*z;
```

$$obj := -x+y+2z$$

```
> maximize(obj,cnsts union {x>=0,y>=0,z>=0});
```

$$\{z=\frac{1}{2},y=\frac{49}{8},x=0\}$$

2 常微分方程求解

微分方程求解是数学研究与应用的一个重点和难点. Maple 能够显式或隐式地解析地求解许多微分方程求解. 在常微分方程求解器 dsolve 中使用了一些传统的技术例如 laplace 变换和积分因子法等, 函数 pdesolve 则使用诸如特征根法等经典方法求解偏微分方程. 此外, Maple 还提供了可作摄动解的所有工具, 例如 Poincare-Lindstedt 法和高阶多重尺度法.

帮助处理常微分方程(组)的各类函数存于 Detools 软件包中, 函数种类主要有：可视化类的函数, 处理宠加莱动态系统的函数, 调整微分方程的函数, 处理积分因子、李对称

法和常微分方程分类的函数，微分算子的函数，利用可积性与微分消去的方法简化微分方程的函数，以及构造封闭解的函数等．更重要的是其提供的强大的图形绘制命令 Deplot 能够帮助我们解决一些较为复杂的问题．

2.1 常微分方程的解析解

求解常微分方程最简单的方法是利用求解函数 **dsolve**．命令格式为：

dsolve(ODE);

dsolve(ODE, y(x), extra_args);

dsolve({ODE, ICs}, y(x), extra_args);

dsolve({sysODE, ICs}, {funcs}, extra_args);

其中，ODE—常微分方程，y(x)—单变量的任意变量函数，Ics—初始条件，{sysODE}—ODE 方程组的集合，{funcs}—变量函数的集合，extra_args—依赖于要求解的问题类型．

例如，对于一阶常微分方程 $xy' = y\ln(xy) - y$ 可用 **dsolve** 直接求得解析解：

```
> ODE:=x*diff(y(x),x)=y(x)*ln(x*y(x))-y(x);
```

$$ODE := x\left(\frac{\partial}{\partial x}\mathrm{y}(x)\right) = \mathrm{y}(x)\ln(x\,\mathrm{y}(x)) - \mathrm{y}(x)$$

```
> dsolve(ODE,y(x));
```

$$\mathrm{y}(x) = \frac{\mathbf{e}^{\left(\frac{x}{\_C1}\right)}}{x}$$

可以看出，**dsolve** 的第一个参数是待求的微分方程，第二个参数是未知函数．需要注意的是，无论在方程中还是作为第二个参数，未知函数必须用函数的形式给出(即：必须加括号，并在其中明确自变量)，这一规定是必须的，否则 Maple 将无法区分方程中的函数、自变量和参变量，这一点和我们平时的书写习惯不一致．为了使其与我们的习惯一致，可用 **alias** 将函数用别称表示：

```
> alias(y=y(x));
> ODE:=x*diff(y,x)=y*ln(x*y)-y;
```

$$ODE := x\left(\frac{\partial}{\partial x}y\right) = y\ln(x\,y) - y$$

```
> dsolve(ODE,y);
```

$$y = \frac{\mathbf{e}^{\left(\frac{x}{\_C1}\right)}}{x}$$

函数 **dsolve** 给出的是微分方程的通解，其中的任意常数是用下划线起始的内部变量表示的．

在 Maple 中，微分方程的解是很容易验证的，只需要将解代入到原方程并化简就可以了．

```
> subs(%,ODE);
```

$$x\left(\frac{\partial}{\partial x}\frac{\mathbf{e}^{\left(\frac{x}{\_C1}\right)}}{x}\right)=\frac{\mathbf{e}^{\left(\frac{x}{\_C1}\right)}\ln\left(\mathbf{e}^{\left(\frac{x}{\_C1}\right)}\right)}{x}-\frac{\mathbf{e}^{\left(\frac{x}{\_C1}\right)}}{x}$$

```
> assume(x,real): assume(_C1,real):
> simplify(%);
```

$$-\frac{\mathbf{e}^{\left(\frac{x\sim}{\_C1\sim}\right)}(-x\sim+\_C1\sim)}{x\sim\ \_C1\sim}=-\frac{\mathbf{e}^{\left(\frac{x\sim}{\_C1\sim}\right)}(-x\sim+\_C1\sim)}{x\sim\ \_C1\sim}$$

```
> evalb(%);
```

$$true$$

evalb 函数的目的是对一个包含关系型运算符的表达式使用三值逻辑系统求值，返回的值是 true, false 和 FAIL. 如果无法求值，则返回一个未求值的表达式. 通常包含关系型运算符“=, <>, <, <=, >, >=”的表达式在 Maple 中看作是代数方程或者不等式. 然而，作为参数传递给 evalb 或者出现在 if 或 while 语句的逻辑表达式中时，它们会被求值为 true 或 false. 值得注意的是, evalb 不化简表达式，因此在使用 evalb 之前应将表达式化简，否则可能会出错.

再看下面常微分方程的求解： $y'=\sqrt{y^2+1}$

```
> alias(y=y(x)):
> ODE:=diff(y,x)=sqrt(y^2+1);
```

$$ODE:=\frac{\partial}{\partial x}y=\sqrt{y^2+1}$$

```
> dsolve(ODE,y);
```

$$y=\sinh(x+\_C1)$$

函数 **dsolve** 对于求解含有未知参变量的常微分方程也完全可以胜任:

```
> alias(y=y(x)):
> ODE:=diff(y,x)=-y/sqrt(a^2-y^2);
```

$$ODE:=\frac{\partial}{\partial x}y=-\frac{y}{\sqrt{a^2-y^2}}$$

```
> sol:=dsolve(ODE,y);
```

$$sol:=x+\sqrt{a^2-y^2}-\frac{a^2\ln\left(\frac{2\,a^2+2\sqrt{a^2}\sqrt{a^2-y^2}}{y}\right)}{\sqrt{a^2}}+\_C1=0$$

由此可见，对于不能表示成显式结果的微分方程解，Maple 尽可能将结果表示成隐

式解. 另外, 对于平凡解 y=0 常常忽略, 这一点应该引起注意.

dsolve 对于求解微分方程初值问题也十分方便的:

```
> ODE:=diff(u(t),t$2)+omega^2*u(t)=0;
```

$$ODE := \left(\frac{\partial^2}{\partial t^2}\,u(t)\right) + \omega^2\,u(t) = 0$$

```
> dsolve({ODE,u(0)=u0,D(u)(0)=v0},u(t));
```

$$u(t) = \frac{v0\,\sin(\omega\,t)}{\omega} + u0\,\cos(\omega\,t)$$

2.2 利用积分变换求解微分方程

对于特殊的微分方程, 我们还可以指定 **dsolve** 利用积分变换方法求解, 只需要在 **dsolve** 中加入可选参数 **method=transform** 即可. 其中 **transform** 是积分变换, 可以是 **laplace**、**fourier**、**fouriercos** 或者 **fouriersin** 变换.

作为例子, 我们来看一个具有阻尼的振子在阶跃冲击(Heaviside 函数)下的响应:

```
> ODE:=diff(u(t),t$2)+2*d*omega*diff(u(t),t)+omega^2*u(t)=Heaviside(t);
```

$$ODE := \left(\frac{\partial^2}{\partial t^2}\,u(t)\right) + 2\,d\,\omega\left(\frac{\partial}{\partial t}\,u(t)\right) + \omega^2\,u(t) = \text{Heaviside}(t)$$

```
> initvals:=(u(0)=u[0],D(u)(0)=v[0]);
```

$$initvals := u(0) = u_0,\ D(u)(0) = v_0$$

```
> solution:=dsolve({ODE,initvals},u(t),method=laplace);
```

$$solution := u(t) = \frac{\frac{1}{\omega} + e^{(-t\,d\,\omega)}\left(\frac{(\omega^2 u_0 - 1)\cosh(t\sqrt{d^2\omega^2-\omega^2})}{\omega} + \frac{(\omega\,v_0 + d\,\omega^2 u_0 - d)\sinh(t\sqrt{d^2\omega^2-\omega^2})}{\sqrt{d^2\omega^2-\omega^2}}\right)}{\omega}$$

Maple 给出了问题的通解, 但没有区分自由振动(d=0)、欠阻尼(0<d<1)、临界阻尼(d=1)和过阻尼(d>1)的情况. 下面加以区分求解:

```
> assume(omega>0):
> simplify(subs(d=0,solution));
```

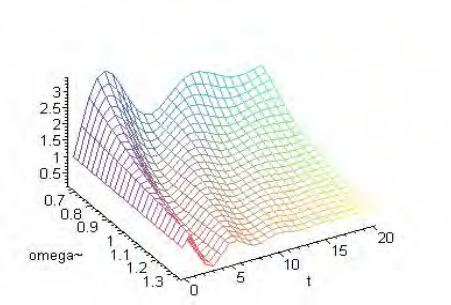
$$u(t) = \frac{1 + \cos(t\,\omega)\,\omega^2\,u_0 - \cos(t\,\omega) + v_0\,\sin(t\,\omega)\,\omega}{\omega^2}$$

```
> K:=subs(d=1/5,u[0]=1,v[0]=1,solution);
```

$$K := u(t) = \frac{\frac{1}{\omega} + e^{(-1/5\,t\,\omega)}\left(\frac{(\omega^2-1)\cosh\left(t\sqrt{-\frac{24}{25}\omega^2}\right)}{\omega} + \frac{\left(\omega + \frac{1}{5}\omega^2 - \frac{1}{5}\right)\sinh\left(t\sqrt{-\frac{24}{25}\omega^2}\right)}{\sqrt{-\frac{24}{25}\omega^2}}\right)}{\omega}$$

```
> with(plots):
```

> plot3d(rhs(%%),omega=2/3..4/3,t=0..20,style=hidden,orientation=[-30,45],axes=framed);



对于 d=1 的情况，可可用下式获得结果：

> **limit(rhs(solution),d=1);**

$$\frac{(\omega^2 u_0 + \omega^2 v_0 t - 1 + \omega^3 u_0 t - t\omega + e^{(t\omega)})\, e^{(-t\omega)}}{\omega^2}$$

再如下例：

> **diff(u(t),t$2)+3*diff(u(t),t)+2*u(t)=exp(-abs(t));**

$$\left(\frac{\partial^2}{\partial t^2}u(t)\right) + 3\left(\frac{\partial}{\partial t}u(t)\right) + 2\,u(t) = e^{(-|t|)}$$

> **dsolve(%,u(t),method=fourier);**

$$u(t) = \frac{2}{3}e^{(-2t)}\,\text{Heaviside}(t) + \frac{1}{6}e^{t}\,\text{Heaviside}(-t) + e^{(-t)}\,t\,\text{Heaviside}(t) - \frac{1}{2}e^{(-t)}\,\text{Heaviside}(t)$$

2.3 常微分方程组的求解

函数 dsolve 不仅可以用来求解单个常微分方程，也可以求解联立的常微分方程组. 特别是对于线性微分方程组，由于数学上具有成熟的理论，Maple 的求解也是得心应手. 其命令格式为：

dsolve({eqn1, eqn2, …, ini_conds}, {vars});

其中, **ini_conds** 是初始条件.

> **eqn1:={diff(x(t),t)=x(t)+y(t),diff(y(t),t)=y(t)-x(t)};**

$$eqn1 := \{\frac{\partial}{\partial t}x(t) = x(t) + y(t), \frac{\partial}{\partial t}y(t) = y(t) - x(t)\}$$

> **dsolve(eqn1,{x(t),y(t)});**

$$\{x(t) = e^{t}(\_C1\,\sin(t) + \_C2\,\cos(t)), y(t) = e^{t}(\_C1\,\cos(t) - \_C2\,\sin(t))\}$$

> **eqn2:=2*diff(x(t),t$2)+2*x(t)+y(t)=2*t;**

$$eqn2 := 2\left(\frac{\partial^2}{\partial t^2}x(t)\right) + 2\,x(t) + y(t) = 2\,t$$

> **eqn3:=diff(y(t),t$2)+2*x(t)+y(t)=t^2+1;**

$$eqn3 := \left(\frac{\partial^2}{\partial t^2} y(t)\right) + 2\,x(t) + y(t) = t^2 + 1$$

> **dsolve({eqn2, qn3, x(0)=0, D(x)(0)=1, y(0)=0, D(y)(0)=0}, {x(t),y(t)});**

$$\{x(t) = \frac{1}{8}\sin(\sqrt{2}\,t)\sqrt{2} + \frac{1}{12}t^3 - \frac{1}{48}t^4 + \frac{3}{4}t,$$

$$y(t) = \frac{1}{4}\sin(\sqrt{2}\,t)\sqrt{2} - \frac{1}{2}t + \frac{1}{2}t^2 - \frac{1}{6}t^3 + \frac{1}{24}t^4\}$$

2.4 常微分方程的级数解法

1) 泰勒级数解法

当一个常微分方程的解析解难以求得时，可以用 Maple 求得方程解的级数近似，这在大多数情况下是一种非常好的方法. 级数解法是一种半解析半数值的方法.

泰勒级数法的使用命令为：

dsolve({ODE,Ics}, y(x), **'series');** 或dsolve({ODE,Ics}, y(x), **'type=series');**

下面求解物理摆的大幅振动方程：$l\theta = -g\sin\theta$，其中 l 是摆长，θ 是摆角，g 是重力加速度.

> **ODE:=l*diff(theta(t),t$2)=-g*sin(theta(t));**

$$ODE := l\left(\frac{\partial^2}{\partial t^2}\theta(t)\right) = -g\sin(\theta(t))$$

> **initvals:=theta(0)=0,D(theta)(0)=v[0]/l;**

$$initvals := \theta(0) = 0, D(\theta)(0) = \frac{v_0}{l}$$

> **sol:=dsolve({ODE,initvals},theta(t),type=series);**

$$sol := \theta(t) = \frac{v_0}{l}t - \frac{1}{6}\frac{g\,v_0}{l^2}t^3 + \frac{1}{120}\frac{g\,v_0\,(v_0^2 + g\,l)}{l^4}t^5 + O(t^6)$$

> **Order:=11:**
> **sol:=dsolve({ODE,initvals},theta(t),type=series);**

$$sol := \theta(t) = \frac{v_0}{l}t - \frac{1}{6}\frac{g\,v_0}{l^2}t^3 + \frac{1}{120}\frac{g\,v_0\,(v_0^2 + g\,l)}{l^4}t^5 - \frac{1}{5040}\frac{g\,v_0\,(11\,g\,l\,v_0^2 + g^2\,l^2 + v_0^4)}{l^6}t^7 + \frac{1}{362880}\frac{g\,v_0\,(57\,g\,v_0^4\,l + 102\,g^2\,v_0^2\,l^2 + g^3\,l^3 + v_0^6)}{l^8}t^9 + O(t^{11})$$

2) 幂级数解法

对于一个符号代数系统来说，幂级数是必不可少的微分方程求解工具. 幂级数求解函数 **powsolve** 存于工具包 **powseries** 中. 但是，这一求解函数的使用范围很有限，它只可以用来求解多项式系数的线性常微分方程或方程组,其求解命令为：powseries[function](prep)或直接载入软件包后用 function(prep), prep 为求解的线性微分方程及其初值.

例：求解： $xy' + y'' + 4x^2 y = 0$

```
> ODE:=x*diff(y(x),x$2)+diff(y(x),x)+4*x^2*y(x)=0;
```

$$ODE := x\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right) + \left(\frac{\partial}{\partial x}\mathrm{y}(x)\right) + 4\,x^2\,\mathrm{y}(x) = 0$$

```
> dsolve(ODE,y(x));
```

$$\mathrm{y}(x) = \_C1\ \mathrm{BesselJ}\left(0, \frac{4}{3}x^{(3/2)}\right) + \_C2\ \mathrm{BesselY}\left(0, \frac{4}{3}x^{(3/2)}\right)$$

```
> initvals:=y(0)=y0,D(y)(0)=0;
```

$$initvals := \mathrm{y}(0) = y0,\ \mathrm{D}(y)(0) = 0$$

```
> with(powseries):
> sol:=powsolve({ODE,initvals});
```

$$sol := \mathbf{proc}\ (powparm)\ \ldots\ \mathbf{end\ proc}$$

```
> tpsform(sol,x,16);
```

$$y0 - \frac{4}{9}y0\,x^3 + \frac{4}{81}y0\,x^6 - \frac{16}{6561}y0\,x^9 + \frac{4}{59049}y0\,x^{12} - \frac{16}{13286025}y0\,x^{15} + \mathrm{O}(x^{16})$$

也可以用 **powsolve** 给出的函数直接获得用递归形式定义的幂级数系数，不过参数必须用***_k***，这是 **powsolve** 使用的临时变量.

```
> sol(_k);
```

$$-4\,\frac{\mathrm{a}(\_k - 3)}{\_k^2}$$

例：求解一维谐振子的解： $y'' + (\varepsilon - x^2)y = 0$

```
> alias(y=y(x)):
> ODE:=diff(y,x$2)+(epsilon-x^2)*y=0;
```

$$ODE := \left(\frac{\partial^2}{\partial x^2}y\right) + (\varepsilon - x^2)\,y = 0$$

```
> H:=powsolve(ODE);
```

$$H := \mathbf{proc}\ (powparm)\ \ldots\ \mathbf{end\ proc}$$

```
> tpsform(H,x,8);
```

$$C0 + C1\,x - \frac{1}{2}\varepsilon\,C0\,x^2 - \frac{1}{6}\varepsilon\,C1\,x^3 + \left(\frac{1}{24}\varepsilon^2\,C0 + \frac{1}{12}C0\right)x^4 + \left(\frac{1}{120}\varepsilon^2\,C1 + \frac{1}{20}C1\right)x^5 + \left(-\frac{1}{30}\varepsilon\left(\frac{1}{24}\varepsilon^2\,C0 + \frac{1}{12}C0\right) - \frac{1}{60}\varepsilon\,C0\right)x^6 + \left(-\frac{1}{42}\varepsilon\left(\frac{1}{120}\varepsilon^2\,C1 + \frac{1}{20}C1\right) - \frac{1}{252}\varepsilon\,C1\right)x^7 + \mathrm{O}(x^8)$$

```
> H(_k);
```

$$\_\frac{\varepsilon\, a(\_k-2)-a(\_k-4)}{\_k\,(\_k-1)}$$

2.5 常微分方程的数值解法

在对微分方程的解析解失效后，可以求助于数值方法求解微分方程. 数值求解的好处是只要微分方程的条件足够多时一般都可求得结果，然而所得结果是否正确则必须依赖相关数学基础加以判断. 调用函数 **dsolve** 求常微分方程初值问题的数值解时需加入参数 **type=numeric**.

另一方面，常微分方程初值问题数值求解还可以选择算法，加入参数“method=方法参数”即可，方法参数主要有：

rkf45：4~5 阶变步长 Runge-Kutta-Fehlberg 法

dverk78：7~8 阶变步长 Runge-Kutta-Fehlberg 法

classical：经典方法，包括向前欧拉法，改进欧拉法，2、3、4 阶龙格库塔法，Sdams-Bashford 方法等

gear：吉尔单步法

mgear：吉尔多步法

2.5.1 变步长龙格库塔法

下面用 4~5 阶 Runge-Kutta-Fehlberg 法求解 van der Pol 方程：

$$\begin{cases} y''-(1-y^2)y'+y=0 \\ y(0)=0, y'(0)=-0.1 \end{cases}$$

```
> ODE:=diff(y(t),t$2)-(1-y(t)^2)*diff(y(t),t)+y(t)=0;
```

$$ODE := \left(\frac{\partial^2}{\partial t^2} y(t)\right) - (1 - y(t)^2)\left(\frac{\partial}{\partial t} y(t)\right) + y(t) = 0$$

```
> initvals:=y(0)=0,D(y)(0)=-0.1;
```

$$initvals := y(0) = 0, D(y)(0) = -.1$$

```
> F:=dsolve({ODE,initvals},y(t),type=numeric);
```

$$F := \mathbf{proc}\ (rkf45\_x)\ \ldots\ \mathbf{end\ proc}$$

此时，函数返回的是一个函数，可以在给定的数值点上对它求值：

```
> F(0);
```

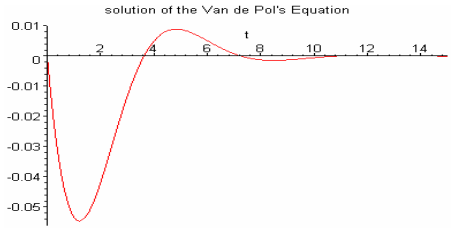
$$\left[t = 0., y(t) = 0., \frac{\partial}{\partial t} y(t) = -.1\right]$$

```
> F(1);
```

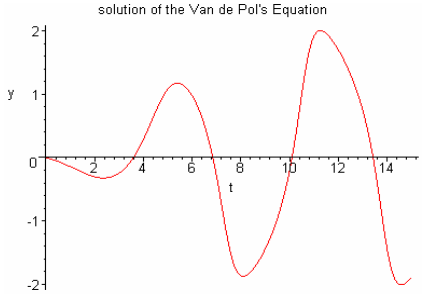
$$\left[t = 1., y(t) = -.144768589749425608\ , \frac{\partial}{\partial t} y(t) = -.178104066128215944\right]$$

可以看到, F 给出的是一个包括 t、y(t)、D(y)(t)在内的有序表，它对于每一个时间点可以给出一组数值表达式. 有序表的每一项是一个等式，可对其作图描述.

```
> plot('rhs(F(t)[2])', t=0..15, title="solution of the Van de Pol's Equation");
```



> **plots[odeplot](F,[t,y(t)],0..15,title="solution of the Van de Pol's Equation");**



2.5.2 吉尔法求解刚性方程

在科学和工程计算中，常常会遇到这样一类常微分方程问题，它可以表示成方程组：$y' = f(t, y), y(t_0) = y_0$，称其为刚性方程，其解的分量数量相差很大，分量的变化速度也相差很大. 如果用常规方法求解，为了使变量有足够高的精度，必须取很小的步长，而为了使慢变分量达到近似的稳态解，则需要很长的时间，这样用小步长大时间跨度的计算，必定造成庞大的计算量，而且会使误差不断积累. 吉尔法是专门用来求解刚性方程的一种数值方法.

> **ODE:=diff(u(t),t)=-2000*u(t)+999.75*v(t)+1000.25,diff(v(t),t)=u(t)-v(t);**

$$ODE := \frac{\partial}{\partial t}\mathrm{u}(t) = -2000\ \mathrm{u}(t) + 999.75\ \mathrm{v}(t) + 1000.25,\ \frac{\partial}{\partial t}\mathrm{v}(t) = \mathrm{u}(t) - \mathrm{v}(t)$$

> **initvals:=u(0)=0,v(0)=-2;**

$$initvals := \mathrm{u}(0) = 0,\ \mathrm{v}(0) = -2$$

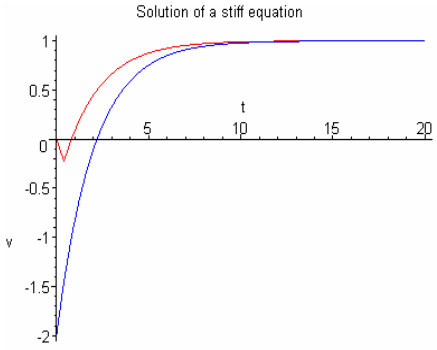
> **ansl:=dsolve({ODE,initvals},{u(t),v(t)},type=numeric,method=gear);**

$$ansl := \mathbf{proc}\ (x\_gear)\ \ldots\ \mathbf{end\ proc}$$

> **ansl(10,0);**

$$[\,t = 10.,\ \mathrm{u}(t) = .989893921726687442,\ \mathrm{v}(t) = .979787842765888594\ ]$$

> **p1:=plots[odeplot] (ansl,[t,u(t)],0..20,color=red):**
p2:=plots[odeplot] (ansl,[t,v(t)],0..20,color=blue):
plots[display] ({p1,p2}, title="Solution of a stiff equation");



2.5.3 经典数值方法

Maple 中常微分方程数值解法中有一类被称作是“经典”(**classical**)方法. 当然, 称其为经典方法不是因为它们常用或是精度高, 而是因为它们的形式简单, 经常被用于计算方法课上的教学内容. 它们是一些常见的固定步长方法, 在 **dsolve** 中用参数 **method=classical**[方法名称], 如果不特别指出, 将默认采用向前欧拉法. 主要有:

foreuler: 向前欧拉法(默认)

hunform: Heun 公式法(梯形方法, 改进欧拉法)

imply: 改进多项式法

rk2: 二阶龙格库塔法

rk3: 三阶龙格库塔法

rk4: 四阶龙格库塔法

adambash: Adams-Bashford 方法(预测法)

abmoulton: Adams-Bashford-Moulton 方法(预测法)

下面给出微分方程数值方法的参数表:

参数名	参数类型	参数用途	参数用法
initial	浮点数的一维数组	指定初值向量	
number	正整数	指定向量个数	
output	'procedurelist'(默认)或'listprocedure'	指定生成单个函数或多个函数的有序表	Procedurelis: 单个函数, 返回有序表 Listprocedure: 函数的有序表
procedure	子程序名	用子程序形式指定第一尖常微分方程组的右边部分	参数 1: 未知函数的个数 参数 2: 自变量 参数 3: 函数向量 参数 4: 导函数向量
start	浮点数	自变量起始值	
startinit	布尔量(默认 FALSE)	指定数值积分是否总是从起始值开始	对 dverk78 不适用
value	浮点数向量(一维数组)	指定需要输出函数值的自变量数值点	如果给定, 结果是一个 2×2 的矩阵. 元素[1,1]是一个向量, 含自变量名和函数名称; 元素[2,1]是一个数值矩阵, 其中第一列 value 的输入相同, 其他列中是相应的函数值

另外，还有一些特殊的附加参数：

maxfun：整数类型，用于最大的函数值数量，默认值 50000，为负数时表示无限制

corrections：正整数类型，指定每步修正值数量，在 **abmoulton** 中使用，建议值≤4

stepsize：浮点数值，指定步长

下面看一个简单的例子：

```
> ODE:=diff(y(x),x)=y(x)-2*x/y(x);
```

$$ODE := \frac{\partial}{\partial x} y(x) = y(x) - \frac{2x}{y(x)}$$

```
> initvals:=y(0)=1;
```

$$initvals := y(0) = 1$$

```
> sol1:=dsolve({ODE,initvals},y(x),numeric,method=classical,stepsize=0.1,start=0);
```

$$sol1 := \mathbf{proc}(x\_classical) \ \ldots \ \mathbf{end\ proc}$$

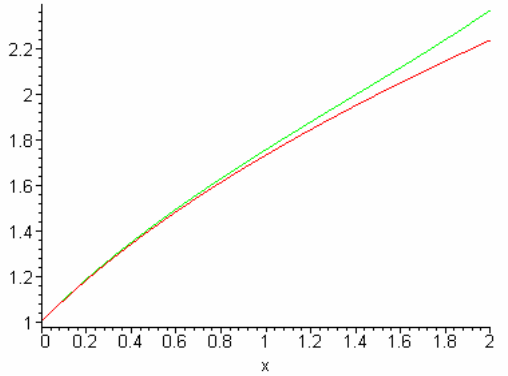
而其解析解为：

```
> sol2:=dsolve({diff(y(x),x)=y(x)-2*x/y(x), y(0)=1}, y(x));
```

$$sol2 := y(x) = \sqrt{2x+1}$$

将两者图形同时绘制在同一坐标系中比较，可以发现，在经过一段时间后，欧拉法的数值结果会产生较大误差.

```
> plot({rhs(sol2),'rhs(sol1(x)[2])'},x=0..2);
```



求解微方程，无论使用什么方法或者加入什么选项，求解完成后必须利用相关数学知识进行逻辑判断，绝对不对简单迷信 Maple 给出的结果，否则很有可能得到一个对于方程本身也许还看得过去，但在数学或者物理意义上不合理的解.

2.6 摄动法求解常微分方程

由于微分方程求解的复杂性，一般微分方程常常不能求得精确解析解，需要借助其它方法求得近似解或数值解，或者两种方法兼而有之. 摄动法是重要的近似求解方法.

摄动法又称小参数法，它处理含小参数 ε 的系统，一般当 ε=0 时可求得解 x_0. 于是可把原系统的解展成 ε 的幂级数 $x = x_0 + x_1\varepsilon + x_2\varepsilon^2 + \cdots$，若这个级数当 $\varepsilon \to 0$ 时一

致收敛，则称正则摄动，否则称奇异摄动．摄动法的种类繁多，最有代表性的是庞加莱—林斯泰特(**Poicare-Lindstedt**)法，在此，我们以该方法求解**van der Pol**方程：

$$y'' - \varepsilon(1-y^2)y' + y = 0$$

当ε=0 时该方程退化为数学单摆的常微分方程，当ε=1 时为 3.5 讨论的情况，对任意ε，该微分方程拥有一个渐进稳定的周期解，称为极限环．

由于 van der Pol 方程中没有显式的时间依赖项，不失一般性，设初值为 y(0)=0．在庞加莱—林斯泰特法中，时间通过变换拉伸：

$\tau = \omega t$，其中 $\omega = \sum_{i=0}^{\infty} \omega_i \varepsilon^i$

对于 $y(\tau)$，van der Pol 方程变为：

$$\omega^2 y'' - \omega\varepsilon(1-y^2)y' + y = 0$$

```
restart:
diff(y(t),t$2)-epsilon*(1-y(t)^2)*diff(y(t),t)+y(t)=0;
```

$$\left(\frac{\partial^2}{\partial t^2}\,y(t)\right) - \varepsilon\,(1-y(t)^2)\left(\frac{\partial}{\partial t}\,y(t)\right) + y(t) = 0$$

```
> ODE:=DEtools[Dchangevar]({t=tau/omega,y(t)=y(tau)},%,t,tau);
```

$$ODE := \omega^2\left(\frac{\partial^2}{\partial \tau^2}\,y(\tau)\right) - \varepsilon\,(1-y(\tau)^2)\,\omega\left(\frac{\partial}{\partial \tau}\,y(\tau)\right) + y(\tau) = 0$$

```
> e_order:=6:
> macro(e=epsilon,t=tau):
> alias(seq(y[i]=eta[i](tau),i=0..e_order)):
> e:=()->e:
> for i from 0 to e_order do
eta[i]:=t->eta[i](t)
od:
> omega:=1+sum('w[i]*e^i','i'=1..e_order);
```

$$\omega := 1 + w_1\,\varepsilon + w_2\,\varepsilon^2 + w_3\,\varepsilon^3 + w_4\,\varepsilon^4 + w_5\,\varepsilon^5 + w_6\,\varepsilon^6$$

```
> y:=sum('eta[i]*e^i','i'=0..e_order);
```

$$y := \eta_0 + \eta_1\,\varepsilon + \eta_2\,\varepsilon^2 + \eta_3\,\varepsilon^3 + \eta_4\,\varepsilon^4 + \eta_5\,\varepsilon^5 + \eta_6\,\varepsilon^6$$

```
> deqn:=simplify(collect(ODE,e),{e^(e_order+1)=0}):
```

> **for i from 0 to e_order do**
ode[i]:=coeff(lhs(deqn),e,i)=0
od:
> **ode[0];**

$$y_0 + \left(\frac{\partial^2}{\partial \tau^2} y_0\right) = 0$$

> **ode[1];**

$$\left(\frac{\partial^2}{\partial \tau^2} y_1\right) - \left(\frac{\partial}{\partial \tau} y_0\right) + y_1 + 2\, w_1 \left(\frac{\partial^2}{\partial \tau^2} y_0\right) + \left(\frac{\partial}{\partial \tau} y_0\right) {y_0}^2 = 0$$

> **ode[2];**

$$\left(\frac{\partial}{\partial \tau} y_0\right) w_1 {y_0}^2 + 2 \left(\frac{\partial}{\partial \tau} y_0\right) y_0 y_1 - \left(\frac{\partial}{\partial \tau} y_1\right) + \left(\frac{\partial^2}{\partial \tau^2} y_2\right) + y_2 + \left(\frac{\partial}{\partial \tau} y_1\right) {y_0}^2 - \left(\frac{\partial}{\partial \tau} y_0\right) w_1 + 2\, w_1 \left(\frac{\partial^2}{\partial \tau^2} y_1\right) + 2 \left(\frac{\partial^2}{\partial \tau^2} y_0\right) w_2 + \left(\frac{\partial^2}{\partial \tau^2} y_0\right) {w_1}^2 = 0$$

> **dsolve({ode[0],eta[0](0)=0,D(eta[0])(0)=C[1]},eta[0](t));**

$$y_0 = C_1 \sin(\tau)$$

> **eta[0]:=unapply(rhs(%),t);**

$$\eta_0 := \tau \to C_1 \sin(\tau)$$

> **ode[1];**

$$\left(\frac{\partial^2}{\partial \tau^2} y_1\right) - C_1 \cos(\tau) + y_1 - 2\, w_1\, C_1 \sin(\tau) + {C_1}^3 \cos(\tau) \sin(\tau)^2 = 0$$

> **map(combine,ode[1],'trig');**

$$\left(\frac{\partial^2}{\partial \tau^2} y_1\right) - C_1 \cos(\tau) + y_1 - 2\, w_1\, C_1 \sin(\tau) + \frac{1}{4} {C_1}^3 \cos(\tau) - \frac{1}{4} {C_1}^3 \cos(3\,\tau) = 0$$

> **ode[1]:=map(collect,%,[sin(t),cos(t)]);**

$$ode_1 := -2\, w_1\, C_1 \sin(\tau) + \left(-C_1 + \frac{1}{4} {C_1}^3\right) \cos(\tau) + \left(\frac{\partial^2}{\partial \tau^2} y_1\right) + y_1 - \frac{1}{4} {C_1}^3 \cos(3\,\tau) = 0$$

> **dsolve({ode[1],eta[1](0)=0,D(eta[1])(0)=C[2]},eta[1](t),method=laplace);**

$$y_1 = \left(-\frac{1}{8} C_1 (C_1 - 2) (C_1 + 2)\, \tau + w_1\, C_1 + C_2\right) \sin(\tau) + \left(\frac{1}{32} {C_1}^3 - C_1\, \tau\, w_1\right) \cos(\tau) - \frac{1}{32} {C_1}^3 \cos(3\,\tau)$$

> **map(collect,%,[sin(t),cos(t),t]);**

$$y_1 = \left(-\frac{1}{8} C_1 (C_1 - 2) (C_1 + 2)\, \tau + w_1\, C_1 + C_2\right) \sin(\tau) + \left(\frac{1}{32} {C_1}^3 - C_1\, \tau\, w_1\right) \cos(\tau) - \frac{1}{32} {C_1}^3 \cos(3\,\tau)$$

```
> solve({coeff(lhs(ode[1]),sin(t))=0,coeff(lhs(ode[1]),cos(t))=0});
```

$$\{ C_1 = 0, w_1 = w_1 \}, \{ C_1 = 2, w_1 = 0 \}, \{ C_1 = -2, w_1 = 0 \}$$

```
> w[1]:=0:C[1]:=-2:
> ode[1];
```

$$\left(\frac{\partial^2}{\partial \tau^2} y_1 \right) + y_1 + 2 \cos(3 \tau) = 0$$

```
> dsolve({ode[1],eta[1](0)=0,D(eta[1])(0)=C[2]},eta[1](t),method=laplace);
```

$$y_1 = \frac{1}{4} \cos(3 \tau) - \frac{1}{4} \cos(\tau) + C_2 \sin(\tau)$$

```
> eta[1]:=unapply(rhs(%),tau);
```

$$\eta_1 := \tau \rightarrow \frac{1}{4} \cos(3 \tau) - \frac{1}{4} \cos(\tau) + C_2 \sin(\tau)$$

```
> map(combine,ode[2],'trig'):
> ode[2]:=map(collect,%,[sin(t),sin(3*t),cos(t),cos(3*t)]);
```

$$ode_2 := \left(\frac{1}{4} + 4 w_2 \right) \sin(\tau) + \frac{5}{4} \sin(5 \tau) + \left(\frac{\partial^2}{\partial \tau^2} y_2 \right) - \frac{3}{2} \sin(3 \tau) + 2 C_2 \cos(\tau) - 3 C_2 \cos(3 \tau) + y_2 = 0$$

```
> solve({coeff(lhs(ode[2]),sin(t))=0,coeff(lhs(ode[2]),cos(t))=0});
```

$$\{ C_2 = 0, w_2 = \frac{-1}{16} \}$$

```
> assign(%):
> dsolve({ode[2],eta[2](0)=0,D(eta[2])(0)=C[3]},eta[2](t),method=laplace):
> eta[2]:=unapply(rhs(%),t);
```

$$\eta_2 := \tau \rightarrow -\frac{3}{16} \sin(3 \tau) + \left(\frac{29}{96} + C_3 \right) \sin(\tau) + \frac{5}{96} \sin(5 \tau)$$

```
> for i from 0 to e_order do
    map(combine,ode[i],'trig'):
    ode[i]:=map(collect,%,[seq(sin((2*j+1)*t),j=0..i),seq(cos((2*j+1)*t),j=0..i)]):
    solve({coeff(lhs(ode[i]),sin(t))=0,coeff(lhs(ode[i]),cos(t))=0}):
    assign(%):
    dsolve({ode[i],eta[i](0)=0,D(eta[i])(0)=C[i+1]},eta[i](t),method=laplace):
    collect(%,[seq(sin((2*j+1)*t),j=0..i),seq(cos((2*j+1)*t),j=0..i)]):
    eta[i]:=unapply(rhs(%),t);
od:
> omega;
```

$$1 - \frac{1}{16} \varepsilon^2 + \frac{17}{3072} \varepsilon^4 + \frac{35}{884736} \varepsilon^6$$

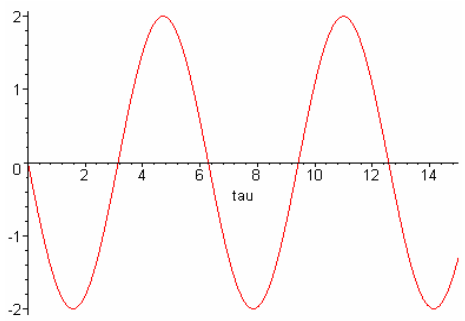
> **y(t):**

> **y:=unapply(simplify(y(t),{e^e_order=0}),t):**

> **e:=1:y(t);**

$$-\frac{1037927}{552960}\sin(\tau)+\frac{1519}{540}\sin(\tau)\cos(\tau)^6-\frac{81}{32}\sin(\tau)\cos(\tau)^4-\frac{61}{80}\sin(\tau)\cos(\tau)^8+\frac{13}{256}\sin(\tau)\cos(\tau)^2+\frac{5257957}{3317760}\cos(\tau)^3+\frac{5533}{7200}\cos(\tau)^{11}+\frac{912187}{129600}\cos(\tau)^7-\frac{799991}{1105920}\cos(\tau)-\frac{1212373}{259200}\cos(\tau)^5-\frac{114941}{28800}\cos(\tau)^9$$

> **plot(y(t),t=0..15);**



在该问题的求解过程中，前半部分我们按照交互式命令方式输入，也就是把数学逻辑推理的过程“翻译”成 Maple 函数，而在后半部分，则采用程序设计方式书写了数学推导过程，这是应用 Maple 解决实际问题的两种方式. 前一种方法只需了解 Maple 函数即可应用，而后一种程序设计方式则需掌握 Maple 程序设计语言. 但是，不论是那一种方式，数学基础总是最重要的.

3 偏微分方程求解初步

Maple 中偏微分方程求解器为 **pdsolve**，该函数及其它偏微分方程求解工具存于软件包 **PDEtools** 中. 函数 **pdsolve** 能够很快的辨认出偏微分方程是否为可以用标准方法求解的类型，如果无法判别，则 **pdsolve** 采用一种启发式的算法尝试偏微分方程按特征结构分离出来. **pdsolve** 的策略就是寻找给定偏微分方程的通解，如不能成功则寻找可以完全分离的变量，因此，该函数返回的结果可能为：

(i) 通解；

(ii) 近似的通解(即包含任意函数但又不足以得到通解的解)；

(iii) 变量分离的非耦合的常微分方程.

如不能完全分离变量，则函数再次调用自身，如仍失败则返回未完全分离的变量并给出一个警告信息. 其命令格式为：

pdsolve(PDE, f);

其中, PDE 为偏微分方程, f 为被求解函数.

下面通过几个例子说明 pdsolve 的用法：

> **PDE1:=diff(u(x,t),[t$2])-1/v^2*diff(u(x,t),[x$2]);**

$$PDE1 := \left(\frac{\partial^2}{\partial t^2}\,u(x,t)\right) - \frac{\frac{\partial^2}{\partial x^2}\,u(x,t)}{v^2}$$

> **pdsolve(PDE1,u(x,t));**

$$u(x,t) = \_F1(-t - v\,x) + \_F2(t - v\,x)$$

> **restart:**

> **PDE2:=a*x*diff(u(x,t),x)+b*t*diff(u(x,t),t)=0;**

$$PDE2 := a\,x\left(\frac{\partial}{\partial x}\,u(x,t)\right) + b\,t\left(\frac{\partial}{\partial t}\,u(x,t)\right) = 0$$

> **pdsolve(PDE2,u(x,t));**

$$u(x,t) = \_F1\left(\frac{t}{x^{\left(\frac{b}{a}\right)}}\right)$$

> **PDE3:=diff(u(x,t),x$2)-t^2*diff(u(x,t),t$2)-t*diff(u(x,t),t)=0;**

$$PDE3 := \left(\frac{\partial^2}{\partial x^2}\,u(x,t)\right) - t^2\left(\frac{\partial^2}{\partial t^2}\,u(x,t)\right) - t\left(\frac{\partial}{\partial t}\,u(x,t)\right) = 0$$

> **pdsolve(PDE3,u(x,t));**

$$u(x,t) = \_F1(t\,\mathbf{e}^{x}) + \_F2(t\,\mathbf{e}^{(-x)})$$

> **restart:**

> **heatPDE:=diff(u(x,t),t)=diff(u(x,t),[x$2]);**

$$heatPDE := \frac{\partial}{\partial t}\,u(x,t) = \frac{\partial^2}{\partial x^2}\,u(x,t)$$

> **pdsolve(heatPDE,u(x,t));**

$$(u(x,t) = \_F1(x)\,\_F2(t))\ \&\text{where}\left[\left\{\frac{\partial^2}{\partial x^2}\,\_F1(x) = \_c_1\,\_F1(x), \frac{\partial}{\partial t}\,\_F2(t) = \_c_1\,\_F2(t)\right\}\right]$$

> **dsolve(diff(F2(t),t)=c[1]*F2(t),F2(t));**

$$F2(t) = \_C1\,\mathbf{e}^{(c_1\,t)}$$

> **dsolve(diff(F1(x),[x$2])=c[1]*F1(x));**

$$F1(x) = \_C1\,\mathbf{e}^{\left(\sqrt{c_1}\,x\right)} + \_C2\,\mathbf{e}^{\left(-\sqrt{c_1}\,x\right)}$$

> **result:=rhs(%)*rhs(%%);**

$$result := \left(\_C1\,\mathbf{e}^{\left(\sqrt{c_1}\,x\right)} + \_C2\,\mathbf{e}^{\left(-\sqrt{c_1}\,x\right)}\right)\,\_C1\,\mathbf{e}^{(c_1\,t)}$$

> **subs(u(x,t)=result,heatPDE):expand(%);**

$$\_C1^2\, c_1\, e^{(c_1 t)}\, e^{(\sqrt{c_1}\, x)} + \frac{\_C1\, c_1\, e^{(c_1 t)}\, \_C2}{e^{(\sqrt{c_1}\, x)}} =$$

$$\_C1^2\, c_1\, e^{(c_1 t)}\, e^{(\sqrt{c_1}\, x)} + \frac{\_C1\, c_1\, e^{(c_1 t)}\, \_C2}{e^{(\sqrt{c_1}\, x)}}$$

> **lhs(%)-rhs(%);**

$$0$$

第五章　*Maple* 图形绘制

图形无疑是数学中最令人着迷的部分，一些枯燥的公式可以从图形看出其美. 历史上有许多学者利用函数图形解决了学科中的许多难题.

客观地说, Maple 不是一种可视化的语言—它不会产生出版品质的图形. 然后，它的图形功能非常强大，足以提供更多的关于函数的信息. 当然，如果需要，它的图形作适当改进即可满足出版要求.

限于篇幅，本章所有图形未作打印，读者只需在计算机上按照书中语句操作即可观其效果，更多图形功能可通过 Maple 帮助获得.

1　二维图形制作

Maple 所提供的二维绘图指令 **plot** 可以绘制二维的函数图、参数图、极坐标图、等高线图、不等式图，等等. 这些绘图指令有些已经内嵌在其核心程序里, Maple 启动时即被装入，直接调用函数命令即可，有些则需要使用 **with(plots)**调用 plots 函数库才能完成.

1.1 基本二维绘图指令

plot (f(x), x=xmin .. xmax);

plot (f(x), x=xmin .. xmax, y=ymin .. ymax);

plot ([f1(x), f2(x), …], x=xmin .. xmax);

plot (f(x), x=xmin .. xmax, option);

其中，xmin..xmax 为 x 的变化范围，ymin..ymax 为 y(即 f(x))的变化范围. **option** 选项参数主要有：

axes：设定坐标轴的显示方式，一般有 FRAME(坐标轴在图形的左边与下面)、BOXED(坐标轴围绕图形)、NORMAL(一般方式显示)或 NONE(无)

color：设定图形所要涂的颜色(可选用也可自设)

coords：指定绘图时所用的坐标系(笛卡尔坐标系(**cartesian**，默认)、极坐标系(**polar**)、

双极坐标系(bipolar)、**logarthmic**(对数坐标系)等

discont：设定函数在不是否用线段连接起来(**discont=true** 则不连接，默认是 **discont=false**)

labels：设定坐标轴的名称(**labels=[x, y]**, x 与 y 分别为 x 与 y 坐标轴的名称)

linestyle：设定所绘线条的线型(**linestyle=n**, n 为 1 是实线, 2 为点, 3 为虚线, 4 为虚线与点交错)

numpoints：设定产生一个函数图形所需的最少样点

scaling：设置 x 与 y 轴的比例(unconstrained 非约束，constrained 约束，比例为 1:1)

style：设定图形的显示样式(LINE(线形)、POINT(点)、PATCH(显示多边形与边线)、PATCHNOGRID(只显示色彩而无边界)

symbol：设定点的格式(主要有 BOX(方块)、CROSS(十字)、CIRCLE(圆形)、POINT(点)、DIAMOND(菱形)等几项)

thickness：设定线条的粗细(0、1、2、3 几种参数, 数值越大线条越粗)

tickmarks：设定坐标轴刻度的数目(设定 **tickmarks=[m, n]**, 则 x 轴刻度为 m, y 轴为 n)

title：定义图形的标题(要用" "把标题引起来)

view：设定屏幕上图形显示的最大坐标和最小坐标，缺省是整个曲线

下面通过一些实例学习：

```
> plot(sin(1/x),x=-0.1..0.1,title="y=sin(1/x)",axes=normal);

> plot(1/(2*sin(x)),x=-10..10,y=-30..30);
```

试比较下述三图的效果:

```
> plot(tan(x),x=-2*Pi..2*Pi);

> plot(tan(x),x=-2*Pi..2*Pi, y=-5..5);

> plot(tan(x),x=-2*Pi..2*Pi, y=-5..5,discont=true);
```

(此处命令 **discont=true** 的作用是去除垂直渐近线)

```
> plot(sin(cos(6*x))/x, x=0..15*Pi, y=-0.6..0.5, axes=NONE);
```

```
> plot(Zeta(x),x=-3..3,y=-3..3,discont=true);
```

除了绘制基本的函数图之外，**plot** 还可绘制自定义函数的图形，也可以同时绘制多个函数图.

```
> f:=x->sin(x)+cos(x)^2;
plot(f(x),x=0..16);

> plot([sin(x),sin(x^2),sin(x^3/10)],x=-2*Pi..2*Pi);
```

利用 **seq** 指令产生一个由函数所组成的序列，并将此函数的序列赋给变量，然后将函数序列绘于同一张图上.

```
> f:=x->sin(x)+cos(x);
fs:=seq(f(x)^(n-1)+f(x)^n,n=1..4):
plot([fs],x=0..20);

> f:=x->x*ln(x^2):g:=x->ln(x):
plot({f,g},0..2,-1.5..1.5);
```

也可以直接把 **seq** 指令放在 **plot** 里来绘出一系列的函数图.

```
> plot([seq(f(x)^(2/n),n=1..3)],x=0..10);
```

1.2 二维参数绘图

更多情况下，我们无法把隐函数化成显函数的形式，因而 plot 指令无法在二维的平面里直接绘图. 但是，在某些情况下，我们可以把平面上的曲线 f(x, y)化成 x=x(t), y=y(t)的形式，其中 t 为参数(**parameter**). 据此即可绘图，其命令格式如下：

plot ([x(t), y(t), t=tmin .. tmax]);

plot ([x(t), y(t), t=tmin .. tmax], xmin .. xmax, y=ymin .. ymax);

plot ([x(t), y(t), t=tmin .. tmax], scaling=CONSTRAINED);

plot ([[x1(t), y1(t), t1=t1min .. t1max], [x2(t), y2(t), t2=t2min .. t2max],…]);

```
> plot([t*exp(t),t,t=-4..1],x=-0.5..1.5,y=-4..1);

> plot([sin(t),cos(t),t=0..2*Pi]);

> plot([sin(t),cos(t),t=0..2*Pi],scaling=CONSTRAINED);
```

上述两上语句都是绘制圆的命令，但由于后者指定的 x、y 坐标的比例为 1:1，所以才得到了一个真正的圆，而前者由于比例不同，则像个椭圆．下面则是内摆线的图形：

```
> x:=(a,b)->(a-b)*cos(t)+b*cos((a-b)*t/b);
```

$$x := (a, b) \to (a - b)\cos(t) + b\cos\left(\frac{(a - b)\,t}{b}\right)$$

```
> y:=(a,b)->(a-b)*sin(t)-b*sin((a-b)*t/b);
```

$$y := (a, b) \to (a - b)\sin(t) - b\sin\left(\frac{(a - b)\,t}{b}\right)$$

当 a=1, b=0.58 时，(x(a,b), y(a,b))图形绘制命令为：

```
> plot ([x(1,0.58), y(1,0.58), t=0..60*Pi], scaling=CONSTRAINED);
```

再作 a, b 取其它值时的情形：

```
> plot([x(2,1.2),y(2,1.2),t=0..6*Pi],scaling=CONSTRAINED);

> plot([x(2,8),y(2,8),t=0..16*Pi],scaling=CONSTRAINED);

> plot([x(2,12),y(2,12),t=0..16*Pi],scaling=CONSTRAINED);
```

下面再看同时绘制多个图形的情形.

```
> plot([[cos(3*t),sin(2*t),t=0..2*Pi],[sin(t),cos(3*t),t=0..2*Pi]]);
```

1.3 数据点绘图

如果所绘的图形是间断性的数据，而不是一个连续的函数，那么我们可以把数据点绘在 x-y 坐标系中，这就是所谓的数据点绘图．其命令格式如下：

plot([[x1, y1], [x2, y2], …], style=point);

plot([[x1, y1], [x2, y2], …]);

```
> data1:=seq([2*n,n^3+1],n=1..10):
plot([data1],style=point);

> data2:=seq([n,1+(-1)^n/n],n=1..15):
plot([data2],style=point,view=[0..20,0..2]);

> data3:=seq([t*cos(t/3),t*sin(t/3)],t=1..30):
plot([data3],style=point);
```

1.4 其它坐标系作图

由于所研究的问题的特殊性，常常需要选用不同的坐标系，在 Maple 中除笛卡尔坐标系（**cartesian**，也称平面直角坐标系，默认)外，还提供了 **polar**(极坐标系)、elliptic(椭圆坐标系)、bipolar(双极坐标系)、maxwell(麦克斯韦坐标系)、logarithmic(双数坐标系)等 14 种二维坐标系，其中最常用的是极坐标系。设定坐标系的命令是 **coords.**

```
> plot(ln(x+1)^2,x=0..8*Pi, coords=polar, scaling=CONSTRAINED,thickness=2);

> plot(sin(6*x),x=0..68*Pi, coords=polar, scaling=CONSTRAINED, tickmarks=[3,3]);

> plot([sin(20*x),cos(sin(2*x))],x=0..2*Pi,coords=elliptic, scaling=CONSTRAINED,
color=[red,blue]);

> plot(exp(sin(68*t)+cos(68*t)), t=0..2*Pi, coords=polar, scaling=CONSTRAINED);

> plot([seq(sin(t)+n*cos(t), n=-5..5)], t=0..Pi, coords=polar, scaling=CONSTRAINED);
```

试比较y=sin(x)在不同坐标系中的图形显示：

```
> plot(sin(x),x=0..2*Pi,coords=polar,scaling=CONSTRAINED);

> plot(sin(x),x=0..2*Pi,coords=bipolar,scaling=CONSTRAINED);

> plot(sin(x),x=0..2*Pi,coords=elliptic,scaling=CONSTRAINED);

> plot(sin(x),x=0..2*Pi,coords=maxwell,scaling=CONSTRAINED);

> restart:
> with(plots,polarplot):
> r:=(n,theta)->cos(5*theta)+n*cos(theta);
```

$$r := (n, \theta) \rightarrow \cos(5\,\theta) + n\cos(\theta)$$

```
> plot({seq([r(n,t)*cos(t),r(n,t)*sin(t),t=0..Pi],n=-5..5)});

> polarplot((exp(cos(theta))-2*cos(4*theta)+sin(theta/12)^5),theta=0..24*Pi);
```

2 三维绘图

2.1 基本三维绘图指令

三维空间的绘图比二维空间更有变化性和趣味性，其命令函数为 plot3d，可直接调用. 命令格式如下：

plot3d(f(x,y), x=xmin .. xmax, y=ymin .. ymax);

plot3d({f(x,y), g(x,y), …}, x=xmin .. xmax, y=ymin .. ymax);

plot3d(f(x,y), x=xmin .. xmax, y=ymin .. ymax, options);

其中，xmin..xmax 为 x 的变化范围，ymin..ymax 为 y(即 f(x))的变化范围. Option 选项参数与二维时的情形相似，这里只列示新增指令的意义：

cotours：设定等高线的数目或者等高线的值

grid：设定组成曲面的样点数或方形网格的数量

gridstyle：设定网格的形状(rectangular—矩形，triangular—三角形)

orientation：设定观看图形的视角(但设定视角的最佳方式是用鼠标拖动图形)

projection：设定投影的模式

shading：设定曲面着色的方式

与二维情形相同，在Maple中三维绘图坐标系的选定使用命令coords，缺省坐标系为笛卡尔坐标系(**cartesian**)，此外还有：bipolarcylindrical(双极坐标), bispherical(双球面坐标), cardioidal(心脏线坐标), cardioidcylindrical(心形柱坐标), casscylindrical( ), confocalellip(共焦椭球坐标), confocalparab(共焦抛物线坐标), conical(锥形坐标), cylindrical(柱坐标), ellcylindrical(椭柱坐标), ellipsoidal(椭球坐标), hypercylindrical (超圆柱坐标), invcasscylindrical, invellcylindrical(逆椭球坐标), invoblspheroidal( ), invprospheroidal( ), logcoshcylindrical(双数双曲余弦柱坐标), logcylindrical(对数柱坐标), maxwellcylindrical(麦克斯韦柱坐标), oblatespheroidal( ), paraboloidal(抛物面坐标), paracylindrical(参数柱坐标), prolatespheroidal(扁类球坐标), rosecylindrical(玫瑰形柱坐标), sixsphere(六球坐标), spherical(球坐标), tangentcylindrical(正切柱坐标), tangentsphere(正切球坐标)和toroidal(圆环面坐标).

```
> plot3d(x*y^2/(x^2+y^4),x=-1..1,y=-1..1,axes=boxed);

> plot3d(x*y/(x^2+y^2+2*x*y),x=-4..4,y=-4..4, axes=BOXED);

> plot3d(sin(x*y),x=-Pi..Pi,y=-Pi..Pi);

> plot3d({2*sin(x)*cos(y),-6*x/(x^2+y^2+1)},x=-4..4,y=-4..4);
```

> **plot3d(sin(z/2), t=0..3*Pi/2, z=-4..4, coords=spherical);**

> **plot3d(1,t=0..2*Pi,p=0..Pi, coords=spherical, scaling=constrained);**

> **plot3d(sin(t)*sin(p^2), t=0..Pi, p=0..Pi, coords=spherical, grid=[35,35]);**

> **plot3d(theta,theta=0..8*Pi,phi=0..Pi, coords=spherical, style=wireframe);**

> **plot3d(theta,theta=0..8*Pi,phi=0..Pi, coords=toroidal(2), style=wireframe);**

> **plot3d(theta,theta=0..8*Pi,z=-1..1, coords=cylindrical, style=patch):**

2.2 三维参数绘图

当二元函数无法表示成 $z = f(x, y)$ 时，有时可以用一组参数方程表示，关于这类参数方程的 Maple 作图，指令如下：

plot3d([fx, fy, fz], t=tmin .. tmax, u=umin .. umax);

plot3d([fx, fy, fz], t=tmin .. tmax, u=umin .. umax, options);

```
> plot3d([sin((x+10)/2),cos(y^3/3),x],x=-4..4,y=1..4);
```

> **plot3d([cosh(u)*cos(v),cosh(u)*sin(v),u],u=-2..2,v=0..2*Pi);**

> **plot3d([cos(u)*cos(v),cos(u)*sin(v),u^2],u=-2..2,v=0..2*Pi,axes=FRAME);**

> **plot3d([cos(u)*cos(v),cos(u)*sin(v), sin(u)], u=-1..1, v=0..2*Pi, orientation=[146,21], scaling=CONSTRAINED);**

3 特殊作图

3.1 图形的显示与合并

```
> with(plots):
g1:=plot(cos(x),x=-2*Pi..2*Pi):
g2:=plot(sin(x),x=-2*Pi..2*Pi,thickness=5):
display(g1,g2,axes=BOXED);
```

```
> g3:=plot3d(2*exp(-sqrt(x^2+y^2)),x=-6..6,y=-6..6):
g4:=plot3d(sin(sqrt(x^2+y^2)),x=-6..6,y=-6..6):
```

```
display(g3,g4);
```

3.2 不等式作图

不等式作图基本上有 4 部分：

① 解区间(feasible region)：此区域完全满足所有的不等式;

② 非解区间(excluded region)：此区域不完全满足所有不等式;

③ 开线(open lines)：不等式的边界，但不包含此边界;

④ 闭线(closed lines)：不等式的边界(包含此边界)

```
> with(plots):
inequal(2*x-5*y<6,x=-3..3,y=-3..3);
```

```
> ineqns:={x-y+2>0,2*x+3*y+9>0,8*x+3*y-27<0};
sol:=solve(ineqns,{x,y});
ans:=map(convert,sol,equality);
implicitplot(ans,x=-6..8,y=-10..10);
```

```
> inequal(ineqns,x=-6..8,y=-10..10,optionsexcluded=
(color=wheat),optionsopen=(color=red));
```

```
> neweqs:=ineqns union{x>=0,y>=0}:
> inequal(neweqs,x=-6..8,y=-10..10,optionsexcluded=
(color=wheat),optionsopen=(color=red));
```

3.3 空间曲线绘图

```
> with(plots):
spacecurve([cos(t/2),sin(t/2),t,t=0..68*Pi],numpoints=500);
```

```
> spacecurve({[3*cos(t), 3*sin(t), t, t=0..12*Pi], [2+t*cos(t), 2+t*sin(t), t, t=0..
10*Pi]}, numpoints=200);
```

```
> spacecurve({[t*cos(2*Pi*t),t*sin(2*Pi*t),2+t],[2+t,t*
cos(2*Pi*t),t*sin(2*Pi*t)],[t*cos(2*Pi*t),2+t,t*sin
(2*Pi*t)]},t=0..10,shading=none,numpoints=500,style=
line,axes=boxed);
```

3.4 隐函数作图

```
> with(plots):
eqn:=x^2+y^2=1;
sol:=solve(eqn,x);
plot([sol],y=-1..1,scaling=constrained);
```

```
> implicitplot(eqn,x=-1..1, y=-1..1, scaling=constrained);
```

```
> implicitplot((x^2+y)^2=x^2-y^2-1/60, x=-3..3, y=-3..3, grid=[100,100]);
```

```
> implicitplot3d(x^3+y^3+z^3+1=(x+y+z+ 1)^3,x=-2..2,y=-2..2,z=-2..2);
```

```
>implicitplot3d(r=(1.3)^x*sin(y),x=-1..2*Pi,y=0..Pi,r=0.1..5, coords=spherical);
```

```
> p:= proc(x,y,z) if x^2<y then x^2 + y^2 else x - y end if end proc:
implicitplot3d(p,-2..2,-1..3,0..3);
```

3.5 等高线与密度图

```
> with(plots):
expr:=6*x/(x^2+y^2+1);
plot3d(expr,x=-6..6,y=-6..6,orientation=[-119,37]);
```

上面是**expr**的三维图，试看其密度图(**contourplot**)、等高线图(**densityplot**)：

```
> densityplot(expr,x=-6..6,y=-6..6,grid=[60,60],style=patchnogrid,axes=boxed);
```

```
> contourplot(expr,x=-6..6,y=-6..6,contours=[-2.7,-2,-1,1,2,2.7],grid=[60,60],thickness=2);
```

还可以用 **display** 将等高线图与密度图绘制在同一张图上：

```
> display(%,%%);
```

进一步，还可以为等高线图着色(用 **filled=true**)，并以 **coloring** 来指定着色的方向.

```
> contourplot(expr,x=-10..10,y=-6..6,filled=true,grid=[50,50],coloring=
  [white,red],axes=boxed);
```

```
> contourplot3d(expr, x=-6..6, y=-4..4, axes=boxed, orientation=[-124,67],
```

```
filled=true,coloring=[navy,pink]);
```

3.6 对数作图

对数作图主要有三种情形：logplot（线性-对数）、loglogplot（对数-对数）、semilogplot (对数-线性).

```
> with(plots):
  logplot(x^2-x+4,x=1..12);

> loglogplot(x^2-x+4,x=1..12);

> semilogplot(x^2-x+4,x=1..12);

> loglogplot([cos(2*t)^2+3,sin(t^2)^2+1,t=0..3]);
```

3.7 高级作图指令

3.7.1 在图形上加上文字

textplot 和 **textplot3d** 指令可以分别在二维与三维图形上加上文字，其默认方式是文字居中对齐，如果想要改变对齐方式，可以利用 **align=direction** 来设定，**direction** 选项可以是 BELOW、ROGHT、ABOVE、LEFT 中的任一种，或是其中几种的组合.

```
> with(plots):
  g1:=textplot([3,0.2,"sin(2*x)/(x^2+1)"],align={right,above}):
  g2:=plot(sin(2*x)/(x^2+1),x=-6..6):
  display(g1,g2);

> textplot3d([[1,2,3,"My plot3d"],[1,-1.1,1,"z=sin(2*x+y)"]],color=blue,axes=frame):
  plot3d(sin(2*x+y),x=-1..2,y=-1..2):
  display(%,%%,orientation=[159,47]);
```

3.7.2 根轨迹作图

根轨迹图(**root locus plot**)是控制学上相当重要的一个部分，许多系统的特性(如稳定度(**stability**)均可从根轨迹图上显示出来．所谓根轨迹图，也就是调整转换函数(**transfer function**)的特性方程式的某项系数，在复数平面上画出特性方程式的根变化情形(可能有实数根或共轭复数根).

```
> with(plots):
  rootlocus((s^5-s^3+2)/(s^2+1),s,-6..12,style=point);

> rootlocus((s^6+s^3+2)/(s^2+1),s,-6..12);

> rootlocus((s^2+2*s+2)/(s-1),s,-10..10);
```

3.7.3 向量场与梯度向量场的作图

向量场(**vector field**)与梯度向量场(**gradient vector field**)的概念常用来描述电磁学中的电磁场，或者是流体力学中的流场.

```
> with(plots):
  fieldplot([sin(2*x*y),cos(2*x-y)],x=-2..2,y=-2..2,arrows=SLIM,axes=boxed,
  grid=[30,30]);

> fieldplot3d([sin(2*x*y),cos(2*x-y),sin(z)],x=-2..2,
  y=-2..2,z=0..2,arrows=SLIM,axes=frame,grid=[12,12,6]);

> fieldplot3d([(x,y,z)->2*x,(x,y,z)->2*y,(x,y,z)->1],-1..1,-1..1,-1..1,axes=boxed);

> gradplot(sin(x)*cos(y),x=-2..2,y=-2..2,arrows=SLIM,axes=boxed);

> gradplot3d(z*sin(x)+cos(y),x=-Pi..Pi,y=-Pi..Pi,z=0..2,
  arrows=SLIM,axes=boxed,grid=[6,6,6]);
```

4)复数作图

二维的复数作图 **complexplot** 是以 x 轴为实轴，以 y 轴为虚数轴来作图，而三维的复数作图 **complexplot3d** 则是以 x、y 轴所组成的平面为复数平面, z 轴为虚数轴来作图.

```
> with(plots):
  complexplot(x+x*I,x=0..8);

> complexplot(sinh(3+x*I),x=-Pi..Pi,scaling=constrained);

> complexplot3d(sech(z),z=-2-3*I..2+3*I,axes=frame);

> complexplot3d(GAMMA(z),z=-2.5-2*I..4+2*I,view=0..6,
```

```
grid=[35,33], linestyle=2, orientation=[-132,76], axes=frame);

> complexplot([1+2*I, 3-4*I, 5+6*I, 7-8*I], x=0..12,style=point);
```

5)复数映射绘图

复数映射作图命令 conformal(f(z), range)是以 f(z)为映射函数，按 range 所指定的范围映射到另一个复数平面.

```
> with(plots):
  conformal(sin(z),z=-Pi/2-1.5*I..Pi/2+1.5*I);

> conformal(tan(z),z=-Pi/4-I..Pi/4+I);

> conformal(1/z,z=-1-I..1+I,-6-6*I..6+6*I, color=magenta);

> conformal((z-I)/(z+I),z=-3-3*I..3+3*I,-4-4*I..4+4*I,grid=[30,30],style=LINE);

> conformal3d(sin(z),z=0..2*Pi+I*Pi);
```

6)圆管作图

```
> with(plots):
> tubeplot([2+t*cos(t),2+t*sin(t),t],t=0..5.6*Pi,radius=4,grid=[124,16]);

> tubeplot([3*sin(t),t,3*cos(t)],t=-3*Pi..4*Pi,radius=1.2+sin(t),numpoints=80);

> tubeplot({[cos(t),sin(t),0],[0,sin(t)-1,cos(t)]},t=0..2*Pi,radius=1/4);

> tubeplot({[cos(t),sin(t),0],[0,sin(t)-1,cos(t)]},t=0..2*Pi,radius=1/10*t);
```

在 Maple 的三维绘图中，我们甚至于可以使用一个程序或一个二元算子定义艳丽的色彩：

```
> F:=(x,y)->sin(x):
  tubeplot({[cos(t),sin(t),0],[0,sin(t)-1,cos(t)]},t=0..2*Pi,radius=1/4,color=F,style=patch);
```

7)曲面数据作图

```
> with(plots):
  pts:=[[[0,0,3],[0,1,3],[0,2,4]],[[1,0,4],[1,1,5],[1,2,5]],[[2,0,4],[2,1,5],[2,2,6]]]:
  surfdata (pts, labels=["x","y","z"], orientation=[-123,45], axes=boxed,
  tickmarks=[3,3,3]);

> pts:=seq([seq([x/2, y/2, -x*y/(x^2+y^2+1)], y=-8..8)], x=-8..8):
  surfdata([pts],axes=frame,orientation=[-60,-100]);

> cosdata :=[seq([ seq([i,j,evalf(cos((i+j)/2))], i=-5..5)], j=-5..5)]:
  sindata :=[seq([ seq([i,j,evalf(sin((i+j)/2))], i=-5..5)], j=-5..5)]:
  surfdata( {sindata,cosdata}, axes=frame, labels=[x,y,z],orientation=[-35,80] );
```

8) 多边形和多面体绘制

```
> with(plots):
> ngon:=n->[seq([ cos(2*Pi*i/n), sin(2*Pi*i/n) ], i = 1..n)]:
  display([polygonplot(ngon(8)), textplot([0,0,`Octagon`]) ], color=pink);

> head:=[0,0],[-10,0],[-18,6],[-18,14],[-14,17],[-14,24],[-10,20],[0,20],[10,20],[14,24],
  [14,17], [18,14],[18,6],[10,0]:
  leye:=[-10,14],[-7,12],[-10,10],[-13,12]:
  reye:=[10,14],[7,12],[10,10],[13,12]:
  koko:=[-0.5,7.5],[0.5,7.5],[0,8.5]:
  polygonplot({[head],[leye],[reye],[koko]},axes=NONE);

> polyhedraplot([0,0,0],polyscale=0.6,polytype=hexahedron,scaling=CONSTRAINED,
  orientation=[-30,70]);

> polyhedraplot([0,0,0],polytype=octahedron);

>polyhedraplot([0,0,0],polytype=dodecahedron,style=PATCH, scaling=CONSTRAINED,
  orientation=[-60,60],axes=boxed );

> polyhedraplot([0,0,0],polyscale=0.6,polytype=icosahedron);

> polyhedraplot([0,0,0],polytype=TriakisIcosahedron,style=PATCH,scaling=CONSTRAINED,orientation=[71,66]);
```

4 动 画

Maple 具有动画功能，存于 plots 库中的动画函数分别为 animate 和 animate3d. 要创建一个动画，必须在所需做动画的函数中加入附加参数(时间参数)并简单地告诉 animate 或 animate3d 函数需要多少次以及在那个时间内计算曲面，动画函数就可以足够快地播放图形的时间序列，以创建运动的现象. 其命令格式分别如下：

animate (F, x, t);

animate3d (F,x,y,t);

其中，F—要绘图的函数，x, y—横轴、纵轴的变化范围，t—结构参数的变化范围

```
> with(plots):
> animate(cos(3*t)*sin(3*x),x=0..2*Pi,t=0..2*Pi,frames=100,color=red,scaling=constrained);

> animate( [u*sin(t),u*cos(t),t=-Pi..Pi],u=1..8,view=[-8..8,-8..8]);

> s:=t->100/(100+(t-Pi/2)^8): r:=t->s(t)*(2-sin(7*t)-cos(30*t)/2):
   animate([u*r(t)/2,t,t=-Pi/2..3/2*Pi],u=1..2,numpoints=200,coords=polar,axes=none,
   color=green);

> animate3d(x*cos(t*u),x=1..3,t=1..4,u=2..4,coords=spherical);

> animate3d(sin(x)*cos(t*u),x=1..3,t=1..4,u=1/4..7/2,coords=cylindrical);

> animate3d([x*u,u*t,x*cos(t*u)],x=1..3,t=1..4,u=2..4,coords=cylindrical);

> animate3d(cos(3*t)*sin(3*x)*cos(3*y),x=0..Pi,y=0..Pi,t=0..2*Pi,frames=100,
   color=cos(x*y), scaling=constrained);

> p:=seq(plot([[0,0],[sin(2*Pi*t/100),cos(2*Pi*t/100)]]),t=0..100):
     display([p],insequence=true,scaling=constrained,axes=none);
```

再看一个更复杂的动画例子—摆线的运动动画：

```
> restart; with(plots):
> revolutions:=omega*t/(2*Pi):
> translation:=[2*Pi*r*revolutions,0]:
> rotation:=[r*sin(omega*t),r*cos(omega*t)]:
> cycloid:=translation+rotation+[0,r]:
```

```
> omega:=1:r:=1:rollTime:=4*Pi:
> cycloidTrace:=animatecurve([cycloid[1],cycloid[2],t=0..rollTime],
      view=[0..r*omega*rollTime,0..4*r],scaling=constrained,color=blue,frames=100):
> disk:=animate([translation[1]+r*cos(s),r+r*sin(s),s=0..2*Pi],t=0..rollTime,
      scaling=constrained,frames=100,view=[0..r*omega*rollTime,0..4*r]):
> chord:=animate([translation[1]+(s/r)*rotation[1],(s/r)*rotation[2]+r,s=-r..r],t=0..
      rollTime,scaling=constrained,color=blue,frames=100,view=[0..r*omega*rollTime,0..4*r]):
> display([cycloidTrace,disk,chord],title=`Animation of a Cycloid`);
```

第六章　*Maple* 程序设计

前面，我们使用的是 Maple 的交互式命令环境. 所谓交互式命令环境，就是一次输入一条或几条命令，然后按回车，这些命令就被执行了，执行的结果显示在同一个可执行块中. 对于大多数用户来说，利用交互式命令环境解决问题已经足够了，但如果要解决一系列同一类型的问题或者希望利用 Maple 编写需要的解决特定问题的函数和程序，以期更加充分地利用 Maple 的强大功能，提高大规模问题的计算效率，掌握一定的程序设计是必要的. 幸运的是, Maple 自身提供了一套编程工具，即 Maple 语言. Maple 语言实际上是由 Maple 各种命令以及一些简单的过程控制语句组成的.

1 编程基础

1.1 算子

所谓算子，是从一个抽象空间到另一个抽象空间的函数. 在数学上算子的含义通常是函数到函数的映射. 在 **Maple** 中，算子常用“箭头”记号定义(也称箭头操作符)：

```
> f:=x->a*x*exp(x);
```

$$f := x \to a\, x\, \mathbf{e}^{x}$$

```
> g:=(x,y)->a*x*y*exp(x^2+y^2);
```

$$g := (x, y) \to a\, x\, y\, \mathbf{e}^{(x^2+y^2)}$$

另外，函数 **unapply** 也可以从表达式建立算子：

```
> unapply(x^2+1,x);
```

$$x \to x^2 + 1$$

```
> unapply(x^2+y^2,x,y);
```

$$(x, y) \to x^2 + y^2$$

当我们依次把算子 f 作用到参数 0, a, x^2+a 时即可得平常意义上的函数值：

```
> f:=t->t*sin(t);
```

$$f := t \to t\sin(t)$$

```
> f(0);
```

$$0$$

```
> f(a);
```

$$a\sin(a)$$

```
> f(x^2+a);
```

$$(x^2+a)\sin(x^2+a)$$

上述结果是函数作用的例子. 而最后一个结果 $(x^2+a)\sin(x^2+a)$ 实际上是算子 f 与算子 g:=t->t^2+a 复合后再作用到参数 x 的结果.

从数学上讲, 作用与复合是不同的，它们产生的结果是有区别的, 但在使用它们时, 两者还是有些重叠的. 在 Maple 中, 可以依赖于语法把它们区分开:

(1) 当复合两个算子时, 结果仍是算子, 两个算子的定义域必须是相容的;

(2) 当把一个算子作用于一个参数(参数必须在算子的定义域中)时, 结果是一个表达式;

(3) 在 Maple 中，函数作用的语法是使用括号( )，如函数 f 作用到参数 u 写作 f(u). 而复合算子的符号是@，多重复合时使用符号@@.

通过进一步的例子可以清楚区分作用与复合的功能: f 和 g 复合的结果是算子 $f\circ g := t \mapsto f(g(t))$，而把这个算子作用到参数 x 得到表达式 $f(g(x))$. 例如, $f = t \mapsto \sin(t+\varphi), g = u \mapsto \exp(u)$，则 $f\circ g := z \mapsto \sin(\exp(z)+\varphi)$ 是一个算子，而 $f(g(x)) = \sin(\exp(x)+\varphi)$ 是一个表达式, 因为 x 是一个实数. 试比较下述两例：

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

$$(\mathrm{D}(g))@f\ \mathrm{D}(f)$$

```
> D(g*h);
```

$$\mathrm{D}(g)\,h + g\,\mathrm{D}(h)$$

另外一个应引起注意的问题是算子(函数)与表达式的异同，在第一章 2.2.2 中曾探讨过函数与表达式的区别，这里再通过几个例子说明其中的微妙差异：

```
> f1:=x^2+1;
> f2:=y^2+1;
```

$$f1 := x^2+1$$

$$f2 := y^2 + 1$$

```
> f3:=f1+f2;
```

$$f3 := x^2 + 2 + y^2$$

再看下面的例子：

```
> g1:=x->x^2+1;
> g2:=y->y^2+1;
```

$$g1 := x \to x^2 + 1$$

$$g2 := y \to y^2 + 1$$

```
> g3:=g1+g2;
```

$$g3 := g1 + g2$$

与前面例子不同的是，两个算子(函数) **g1**, **g2**相加的结果依然是函数名**g3**，出现这个问题的主要原因是g1 与g2 分别为x，y的函数，Maple认为它们的定义域不相容. 要得到与前例的结果，只需稍作改动：

```
> g3:=g1(x)+g2(y);
```

$$g3 := x^2 + 2 + y^2$$

下面的例子想说明生成 Maple 函数的两种方式“箭头操作符”及“unapply”之间微妙的差异：

```
> x:='x': a:=1: b:=2: c:=3:
> a*x^2+b*x+c;
```

$$x^2 + 2\,x + 3$$

```
> f:=unapply(a*x^2+b*x+c,x);
```

$$f := x \to x^2 + 2\,x + 3$$

```
> g:=x->a*x^2+b*x+c;
```

$$g := x \to a\,x^2 + b\,x + c$$

由此可见，f 中的 a,b,c 已经作了代换，而 g 中则显含 a,b,c。再看下面实验：

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

$$x^2 + 2\,x + 3$$

$$x^2 + 2\,x + 3$$

f 与 g 两者相同，再对其微分：

```
> D(f); D(g);
```

$$x \to 2\,x + 2$$

$$x \to 2\,a\,x + b$$

再改变常数 c 的值，观察 f 与 g 的变化：

```
> c := 15;
```

$$c := 15$$

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

$$x^2 + 2\,x + 3$$

$$x^2 + 2\,x + 15$$

由此可见，在利用 Maple 进行函数研究时，对同一问题应该用不同方法加以校验，而这一切的支撑是数学基础！

1.2 编程初体验

利用算子可以生成最简单的函数—单个语句的函数，但严格意义上讲它并非程序设计，它所生成的数据对象是子程序. 所谓子程序，简单地说，就是一组预先编好的函数命令，我们由下面的简单程序来看看 **Maple** 程序的结构：

```
> plus:=proc(x,y)
    x+y;
end;
```

这个程序只有 2 个参数，在程序内部它的名称是 *x*, y，这是 Maple 最简单的程序结构，仅仅在 **proc( )**和 **end** 中间加上在计算中需要的一条或者多条命令即可，**Maple** 会把最后一个语句的结果作为整个子程序的返回结果，这一点需要引起注意. 再看下例：

```
> P:=proc(x,y)
    x-y;
    x*y;
    x+y;
 end:
> P(3,4);
```

7

显然，尽管程序**P**有三条计算命令，但返回的只是最后一个语句**x+y;**的结果. 要想输出所有的计算结果，需要在程序中增加**print**语句:

```
> P:=proc(x,y)
```

```
   print(x-y);
   print(x*y);
   print(x+y);
 end:
> P(3,4);
```

$$-1$$

$$12$$

$$7$$

再看下面几个例子：

```
> for i from 2 to 6 do
expand((x+y)^i );
od;
```

$$x^2 + 2\,x\,y + y^2$$

$$x^3 + 3\,x^2\,y + 3\,x\,y^2 + y^3$$

$$x^4 + 4\,x^3\,y + 6\,x^2\,y^2 + 4\,x\,y^3 + y^4$$

$$x^5 + 5\,x^4\,y + 10\,x^3\,y^2 + 10\,x^2\,y^3 + 5\,x\,y^4 + y^5$$

$$x^6 + 6\,x^5\,y + 15\,x^4\,y^2 + 20\,x^3\,y^3 + 15\,x^2\,y^4 + 6\,x\,y^5 + y^6$$

```
> F:=proc(n::integer)
if n mod 12=0 then true
else false
fi
end:
> F(123^123), F(1234567890^9);
```

$$false, true$$

从上面几个简单的例子可以看出 **Maple** 子程序主要包含以下一些内容：

(i) 把定义的子程序赋值给程序名 **procname**，以后就可以用子程序名 **procname** 来调用程序；

(ii) 子程序一律以 **proc(**)开头，括号里是程序的输入参数，如果括号中什么都没有，表示这个子程序没有任何输入参数；

(iii) 子程序中的每一个语句都用分号(或冒号)分开(这一点不是主要的，程序设计时，

在可能的时候—过程当中的最后一个语句、for-循环、if 语句中的最后一个语句省略终结标点也是允许的，这并不是为了懒惰，而是因为在终结语句后面插入一个语句产生的影响要比仅仅执行一个新语句产生的影响大)；

(iv) 在定义完子程序之后, Maple 会显示它对该子程序的解释 (除非在 end 后用冒号结束)，它的解释和你的定义是等价的，但形式上不一定完全相同；

(v) Maple 会自动地把除了参数以外的变量都作为局部变量(local variable)，这就是说，它们仅仅在这个子程序的定义中有效，和子程序以外的任何同名变量无关.

在定义了一个子程序以后，执行它的方法和执行任何 Maple 系统子程序一样—程序名再加上一对圆括号( )，括号中包含要调用的参数，如果子程序没有参数，括号也是不能省略的.

除了上面给出的程序设计方法外，在 Maple 中还可以直接由“->” (箭头)生成程序，如下例：

```
> f:=x->if x>0 then x else -x fi;
```

$f := \mathbf{proc}\ (x)\ \mathbf{option}\ \ operator, arrow;\ \mathbf{if}\ 0 < x\ \mathbf{then}\ \ x\ \mathbf{else}\ \ -x\ \mathbf{end\ if}\ \ \mathbf{end\ proc}$

```
> f(-5),f(5);
```

5, 5

甚至于程序名也可以省略，这种情况通常会在使用函数 map 时遇到：

```
> map(x->if x>0 then x else -x fi,[-4,-3,-2,0,1]);
```

[4, 3, 2, 0, 1]

如果需要察看一个已经定义好的子程序的过程，用 **eval** 命令，查看 Maple 中源程序(如 factor 函数)使用下述组合命令：

interface(verboseproc=2);

print(factor);

再看一个更为有用的简单程序：代数方程的参数解。该程序在代数方程 f(x,y)=0 求解中使用了一个巧妙的代换 y=tx 得到了方程的参数解，它的主要用途是用来画图、求积分、求微分和求级数. 程序如下：

```
> parsolve:=proc(f,xy::{list(name),set(name)},t::name)
local p,x,y;
x:=xy[1];
y:=xy[2];
p:={solve(subs(y=t*x,f),x)}minus{0};
map((xi,u,xx,yy)->{xx=xi,yy=u*xi},p,t,x,y)
end:
```

调用该程序可以方便求解：

```
> parsolve(u^2+v^2=a^2,[u,v],t);
```

$$\{\{u=-\frac{a}{\sqrt{1+t^2}},v=-\frac{t\,a}{\sqrt{1+t^2}}\},\{u=\frac{a}{\sqrt{1+t^2}},v=\frac{t\,a}{\sqrt{1+t^2}}\}\}$$

```
> f:=randpoly([x,y],degree=3,sparse);
```

$$f := -53\,x + 85\,x\,y + 49\,y^2 + 78\,x^3 + 17\,x\,y^2 + 72\,y^3$$

```
> parsolve(f,[x,y],t);
```

$$\{\{y=\frac{1}{2}\frac{t\,(-49\,t^2-85\,t+\sqrt{2401\,t^4+23594\,t^3+10829\,t^2+16536}\,)}{72\,t^3+78+17\,t^2},$$
$$x=\frac{1}{2}\frac{-49\,t^2-85\,t+\sqrt{2401\,t^4+23594\,t^3+10829\,t^2+16536}}{72\,t^3+78+17\,t^2}\},\{$$
$$y=\frac{1}{2}\frac{t\,(-49\,t^2-85\,t-\sqrt{2401\,t^4+23594\,t^3+10829\,t^2+16536}\,)}{72\,t^3+78+17\,t^2},$$
$$x=\frac{1}{2}\frac{-49\,t^2-85\,t-\sqrt{2401\,t^4+23594\,t^3+10829\,t^2+16536}}{72\,t^3+78+17\,t^2}\}\}$$

1.3 局部变量和全局变量

Maple 中的全局变量，是指那些在交互式命令环境中定义和使用的变量，前面所使用的变量几乎都属于全局变量. 而在编写子程序时，需要定义一些只在子程序内部使用的变量，称其为局部变量. 当 **Maple** 执行子程序时，所有和局部变量同名的全局变量都保持不变，而不管在子程序中给局部变量赋予了何值. 如果要把局部变量定义为全局变量，需要用关键词 **global** 在程序最开始加以声明，而局部变量则用 **local** 声明，虽然这是不必要的，但在程序设计时，声明变量是有一定好处的.

下面通过实例演示局部变量与全局变量的不同. 为了更清楚地观察子程序对全局变量的影响，在子程序外先设定一个变量 a 的值：

```
> a:=1;
```

$$a := 1$$

```
> f:=proc( )
   local a;
   a:=12345678/4321;
   evalf(a/2);
end;
> f();
```

1428.567230

```
> a;
```

1

```
> g:=proc( )
   global a;
   a:=12345678/4321;
   evalf(a/2);
end;

> g();
```

$$1428.567230$$

```
> a;
```

$$\frac{12345678}{4321}$$

显然，在前一个程序中，由于在子程序外已经赋值给 a，a 是全局变量，它的值不受子程序中同名局部变量的影响；而在后一个子程序中，由于重新把 a 定义为全局变量，所以子程序外的 a 随着子程序中的 a 值的变化而变化.

子程序中的输入参数，它既不是全局的，也不是局部的. 在子程序内部，它是形式参数，也就是说，它的具体取值尚未被确定，它在程序调用时会被替换成真正的参数值. 而在子程序外部，它们仅仅表示子程序接受的参数的多少，而对于具体的参数值没有关系.

1.4 变量 nargs, args 与 procname

在所有程序中都有三个有用的变量：**nargs**, **args** 与 **procname**. 前两个给出关于调用参量的信息: **nargs** 变量是调用的实际参量的个数，**args** 变量是包含参量的表达式序列，**args** 的子序列通过范围或数字的参量选取. 例如，第 i 个参量被调用的格式为: **args[i]**. **nargs, args** 变量通常在含有可选择参量的程序中使用. 下面看一个例子：

```
> p:=proc( )
local i;
RETURN(nargs,[seq(i^3,i=args)])
end proc:
> p(1,2,3,4,5,6,7,8,9,10);
```

$$10, [1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]$$

该程序利用 **Maple** 函数 **RETURN** 返回了输入参量的个数以及参量序列的立方列表，**RETURN** 函数使用时必须在其后加圆括号，即使无结果返回时也得如此。下面是一个关于求任意序列最大值的复杂程序：

```
> maximum:=proc( )
  local r, i;
```

```
  r:=args[1];
  for i from 2 to nargs do
    if args[i]>r then r:=args[i]
    end if
  end do;
     r;
end proc:
> maximum(12^4,5^7,69^3,10^4+4.9*1000);
```

328509

如果变量 **procname** (程序被调用的名字)存在的话，它可以用来直接访问该程序，通常用 **procname(args)** 完成调用:

```
> f:=proc(a)
if a>0 then RETURN(a^(1/2))
else RETURN('procname(args)')
end if
end proc:
> f(100-3^6),f(10^2+90-9*4);
```

$$f(-629),\sqrt{154}$$

2 基本程序结构

所有的高级程序设计语言都具有程序结构，因为为了完成一项复杂的任务，仅仅按照语句顺序依次执行是远远不够的，更需要程序在特定的地方能够跳转、分叉、循环，……，与此同时，程序结构就产生了.

2.1 for 循环

在程序设计中，常常需要把相同或者类似的语句连续执行多次，此时，通过 **for** 循环结构可以更便捷地编写程序. 试看下面几个例子：

① 求 1 至 5 自然数的和：

```
> total:=0:
for i from 1 to 5 do
   total:=total+i:
od;
```

$$total := 1$$
$$total := 3$$
$$total := 6$$
$$total := 10$$

$$total := 15$$

② 列示 2，4，6，8，10 及其平方数、立方数：

```
>for i from 2 to 10 by 2 do
'i'=i, 'i^2'=2^i, 'i^3'=i^3;
od;
```

$$i = 2,\ i^2 = 4,\ i^3 = 8$$

$$i = 4,\ i^2 = 16,\ i^3 = 64$$

$$i = 6,\ i^2 = 64,\ i^3 = 216$$

$$i = 8,\ i^2 = 256,\ i^3 = 512$$

$$i = 10,\ i^2 = 1024,\ i^3 = 1000$$

③ 列示第 100 到第 108 个素数(第 1 个素数是 2)：

```
>for j from 100 to 108 do
  prime[j]=ithprime(j);
 od;
```

$$prime_{100} = 541$$
$$prime_{101} = 547$$
$$prime_{102} = 557$$
$$prime_{103} = 563$$
$$prime_{104} = 569$$
$$prime_{105} = 571$$
$$prime_{106} = 577$$
$$prime_{107} = 587$$
$$prime_{108} = 593$$

④ 列示 5 到 10 的阶乘及其因数分解：

```
>for n in seq(i!,i=5..10) do
n=ifactor(n);
od;
```

$$120 = (2)^3\ (3)\ (5)$$

$$720 = (2)^4\ (3)^2\ (5)$$

$$5040 = (2)^4\ (3)^2\ (5)\ (7)$$

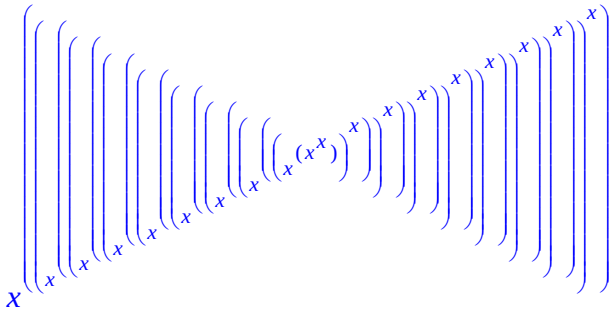
$$40320 = (2)^7\ (3)^2\ (5)\ (7)$$

$$362880 = (2)^7\ (3)^4\ (5)\ (7)$$

$$3628800 = (2)^8\ (3)^4\ (5)^2\ (7)$$

⑤ 一个复杂的幂指函数生成：

```
>f:=proc(x,n)
 local i, t;
 t:=1;
 for i from 1 to n do
    t:=x^(t^x);
 od;
 t;
 end:
> f(x,10);
```



通过上述例子可以看出，Maple 的 **for** 循环结构很接近于英语语法. 在上面例子中, *i* 称为循环变量, 用于循环计数, **do** 后面的部分, 称为循环体, 就是需要反复执行的语句, 可以是一条语句, 也可以是多条语句. 在循环体中, 也可以引用循环变量. 循环体的结束标志, 也就是整个 for 循环结构的结束标志, 是字母“od”. for 循环结构的语法可总结为：

for 循环变量 **from** 初始值 **to** 终止值 (**by** 步长) **do**

循环体

od

在 for 循环结构中，初始值和终了值必须是数值型的, 而且必须是实数, 如果初始值是 1, 可以省略不写, 而步长用 by 引出.

对于上面程序中的重复循环似乎可有可无, 但实际科学计算中, 重复次数往往成千上万, 或者重复的次数无法确定, 循环就必不可少了. 下面再看一个实用程序—从 1 到 n

的自然数的求和程序：

```
> SUM:=proc(n)
    local i, total;
    total:=0;
    for i from 1 to n do
       total:=total+i;
    od;
end;
> SUM(1000);
```

$$500500$$

该程序仅仅具有练习例举的意义，因为在 Maple 中像这样一个问题只需要一个简单语句就可以完成：

```
> Sum(n,n=1..1000)=sum(n,n=1..1000);
```

$$\sum_{n=1}^{1000} n = 500500$$

更一般地，对于确定的求和可用 **add** 命令(该命令无惰性函数形式)：

```
> add(n,n=1..1000);
```

$$500500$$

再看下面一个动画制作例子(图略去)：

```
> for i from -20 to 30 do
   p||i:=plots[implicitplot](x^3+y^3-5*x*y=1-i/8,x=-3..3,y=-3..3,numpoints=800,
 tickmarks=[2,2] )
 od:
> plots[display]( p||(-20..30), insequence=true );
```

2.2 分支结构(条件语句)

所谓分支结构，是指程序在执行时，依据不同的条件，分别执行两个或多个不同的程序块，所以常常称为条件语句. if 条件结构的语法为：

if 逻辑表达式 1(条件 1) **then** 程序块 1
elif 逻辑表达式 2(条件 2) **then** 程序块 2
　　else 程序块 3
fi

下面是一个判断两个数中较大者的例子：

```
> bigger:= proc(a,b)
```

```
if a>=b then a
else b
fi
end;
```

再如下例：

```
> ABS:=proc(x)
    if x<0 then -x
 else x;
    fi;
end;
```

显然，这个绝对值函数的简单程序，对于任意实数范围内的数值型变量计算都是没有问题的，但对于非数值型的参数则无能为力．为此，我们需要在程序中添加一条判断参数是否为数值型变量的语句：

```
> ABS:=proc(x)
    if type(x,numeric) then
        if x<0 then -x else x;
        fi;
      else
        'ABS'(x);
    fi;
end:
```

这里，我们用到一个 **if** 语句的嵌套，也就是一个 **if** 语句处于另一个 **if** 语句当中．这样的结构可以用来判断复杂的条件．我们还可以用多重嵌套的条件结构来构造更为复杂的函数，例如下面的分段函数：

```
> HAT:=proc(x)
    if type(x,numeric) then
        if x<=0 then 0;
        else
            if x<=1 then x;
                else 0;
            fi;
        fi;
    else
        'HAT'(x);
    fi;
end;
```

尽管在上面的程序中我们用了不同的缩进来表示不同的层次关系，这段程序还是很难懂．对于这种多分支结构，可以用另一种方式书写，只需在第二层的 if 语句中使用 **elif**，这样就可以把多个分支形式上写在同一个层次中，以便于阅读．

```
> HAT:=proc(x)
    if type(x,numeric) then
       if x<=0 then 0;
       elif x<=1 then x;
       else 0;
       fi;
     else
       'HAT'(x);
    fi;
end;
```

和许多高级语言一样，这种多重分支结构理论上可以多到任意多重.

再来看看前面的绝对值函数的程序，似乎是完美的，但是，对于乘积的绝对值则没有办法，下面用 map 命令来修正，注意其中的 type(…, `*`)是用来判断表达式是否为乘积，进而对其进行化简.

```
> ABS:=proc(x)
    if type(x,numeric) then
       if x<0 then -x else x fi;
    elif type(x,`*`) then map(ABS,x);
    else
       'ABS'(x);
    fi;
end:
> ABS(-1/2*b);
```

$$\frac{1}{2}\,\mathrm{ABS}(b)$$

分段函数是一类重要的函数，条件语句在定义分段函数时具有明显的优越性，下面学习几个实例：

(1) $f(x)=\begin{cases} x^2+1 & x<0 \\ \sin(\pi x) & 0\le x \end{cases}$

```
> f:=proc( x )
  if x<0 then x^2+1 else sin(Pi*x) fi;
 end;
```

(2) $g(x)=\begin{cases} x^2+x & x\le 0 \\ \sin(x) & x<3\,\pi \\ x^2-6\,\pi\,x+9\,\pi^2-x+3\,\pi & 3\,\pi\le x \end{cases}$

```
>g:=proc(x)
   if x<=0 then
     x^2+x
   else
```

```
    if x<3*Pi then
      sin(x)
    else
      x^2-6*x*Pi+9*Pi^2-x+3*Pi
    fi
  fi
end;
```

2.3 **while** 循环

for 循环在那些已知循环次数，或者循环次数可以用简单表达式计算的情况下比较适用. 但有时循环次数并不能简单地给出，我们要通过计算，判断一个条件来决定是否继续循环，这时，可以使用 while 循环. while 循环标准结构为：

while 条件表达式 **do**

循环体

od

Maple 首先判断条件是否成立，如果成立，就一遍遍地执行循环体，直到条件不成立为止.

下面看一个简单的程序，是用辗转相除法计算两个自然数的最大公约数(**Euclidean** 算法).

```
> GCD:=proc(a::posint, b::posint)
    local p,q,r;
    p:=max(a,b);
    q:=min(a,b);
    r:=irem(p,q);
    while r<>0 do
      p:=q;
      q:=r;
      r:=irem(p,q);
    od;
    q;
end:
> GCD(123456789,987654321);
```


在上面程序中的参数a、b后面的双冒号“::”指定了输入的参数类型. 若类型不匹配时输出错误信息. 再看下面一个扩展**Euclidean**算法的例子：

```
>  mygcdex:= proc(a::nonnegint,b::nonnegint,x::name,y::name)
```

```
  local a1,a2,a3,x1,x2,x3,y1,y2,y3,q;
  a1:=a;  a2:=b;
  x1:=1;  y1:=0;
  x2:=0;  y2:=1;
  while (a2<>0) do
    a3:= a1 mod a2;
    q:= floor(a1/a2);
    x3:=x1-q*x2;
    y3:=y1-q*y2;
    a1:=a2;  a2:= a3;
    x1:=x2;  x2:= x3;
    y1:=y2;  y2:= y3;
  od;
  x:=x1;  y:=y1;
  RETURN(a1)
 end:
> mygcdex(2^10,6^50,'x','y');
```

1024

2.4 递归子程序

正如在一个子程序中我们可以调用其他的子程序一样(比如系统内部函数，或者已经定义好的子程序)，一个子程序也可以在它的内部调用它自己，这样的子程序我们称为递归子程序. 在数学中，用递归方式定义的例子很多，如 Fibonacci 数列：

$$f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2} (n \geq 2)$$

下面我们在 Maple 中利用递归子程序求 Fibonacci 数列的第 n 项：

```
> Fibonacci:=proc(n::nonnegint)::list
    if n<=1 then n;
      else
      Fibonacci(n-1)+Fibonacci(n-2);
    fi;
end:
> seq(Fibonacci(i),i=0..19);
```

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181

```
> Fibonacci(20);
```

6765

但是，应用上述程序计算 Fibonacci(2000)时则不能进行，原因是要计算 F2000，就先得计算 F1999、F1998、……，无限制地递归会耗尽系统的堆栈空间而导致错误.

由数据结构有关理论知道，递归程序理论上都可以用循环实现，比如我们重写上面的程序如下：

```
> Fibonacci:=proc(n::nonnegint)
    local temp,fnew,fold,i;
    if n<=1 then n;
    else
       fold:=0;
       fnew:=1;
       for i from 2 to n do
          temp:=fnew+fold;
          fold:=fnew;
          fnew:=temp;
       od;
    fi;
end:
> Fibonacci(2000):

> time(Fibonacci(2000));
                                .019
```

利用循环，程序不像前面那么易懂了，但同时带来的好处也是不可忽视的，循环结构不仅不会受到堆栈的限制，而且计算速度得到了很大的提高.

我们知道，Maple 子程序默认情况下把最后一条语句的结果作为子程序的结果返回. 我们可以用 RETURN 命令来显式地返回结果，比如前面的递归结果可等价地写成：

```
> Fibonacci:=proc(n::nonnegint)
    option remember;
    if n<=1 then RETURN(n);
    fi;
    Fibonacci(n-1)+Fibonacci(n-2);
end:
```

程序进入 n<=1 的分支中，执行到 RETURN 命令就跳出程序，而不会接着执行后面的语句了. 另外，使用 RETURN 命令在一定程度上可以增加程序的可读性.

在第二章中曾提到 Maple 的自动求导功能，下面通过实例说明。第一个例子是关于分段函数求导问题：

```
> F:=x->if x>0 then sin(x) else arctan(x) fi;
```

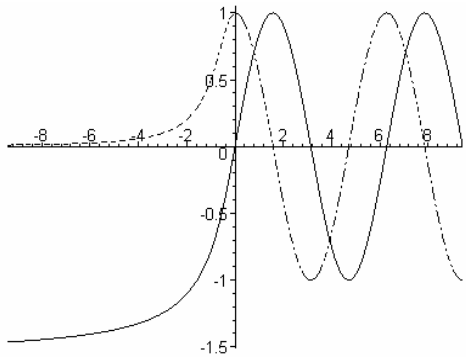
$$F := \mathbf{proc}(x)\ \mathbf{option}\ operator, arrow;\ \mathbf{if}\ 0 < x\ \mathbf{then}\ \sin(x)\ \mathbf{else}\ \arctan(x)\ \mathbf{end\ if}\ \mathbf{end\ proc}$$

```
> Fp:=D(F);
```

$$Fp := \mathbf{proc}(x)\ \mathbf{option}\ operator, arrow;\ \mathbf{if}\ 0 < x\ \mathbf{then}\ \cos(x)\ \mathbf{else}\ 1/(1 + x^2)\ \mathbf{end\ if}\ \mathbf{end\ proc}$$

将分段函数及其导数的曲线绘制在同一个坐标系中，实线为 F，虚线为 Fp：

```
> plot({F,Fp},-3*Pi..3*Pi,linestyle=[1,4],color=black);
```



第二个例子试图说明 Maple 能够完成对多个函数的自动求导：

```
> f:=proc(x)
  local s,t;
  s:=ln(x);
  t:=x^2;
  s*t+3*t;
end;
```

$$f := \mathbf{proc}(x)\ \mathbf{local}\ s, t;\ s := \ln(x);\ t := x^2;\ s \times t + 3 \times t\ \mathbf{end\ proc}$$

```
> fp:=D(f);
```

$$fp := \mathbf{proc}(x)\ \mathbf{local}\ tx, s, t, sx;\ sx := 1/x;\ s := \ln(x);\ tx := 2 \times x;\ t := x^2;\ sx \times t + tx \times s + 3 \times tx\ \mathbf{end\ proc}$$

程序 fp 是怎样构造的呢？对比 f 和 fp 的源代码可以帮助我们了解 Maple 自动求导

机理：

在 fp 的源代码中，每一条赋值语句 V:=f(v1,v2, … vn)之前是 Vx:= fp(v1,v2, … vn)，此处 vi 是局部变量或形式参数，fp(v1,v2, … vn)是对 f(v1,v2, … vn)在形式上的微分，用该赋值语句的的导数取代最后一条语句或返回值。这种方法被称为向前自动微分法。在 Maple 内部对这种方法有一些限制：不能有递归程序，不能有记忆选项，不能对全局变量赋值，只能有常数的循环变量，等。

下面一个例子说明自动求导在计算时间和存储空间上的优越性，这是一个利用递归定义的 $f_1 = x, f_n = x^{f_{n-1}} (n > 1)$ 的幂函数的自动求导问题：

```
> f:=proc(x,n)
   local i,t;
   t:=1;
   for i to n do
      t:=x^t
   od;
   t
end;
```

f := proc (*x*, *n*) local *i*, *t*; *t* := 1; for *i* to *n* do *t* := *x*^*t* end do ; *t* end proc

```
> f1_3:=D[1$3](f):        # 求f对x的三阶导数
> setbytes:=kernelopts(bytesused):     # 使用寄存器内存量
> settime:=time():                  # 开始计时
> f1_3(1.1,22);
```

18.23670379

```
> cpu_time:=(time()-settime)*seconds;     # 计算时间
```

cpu_time := .230 *seconds*

```
> memory_used:=evalf((kernelopts(bytesused)-setbytes)/1024*kbytes,5); # 存储空间大小
```

memory_used := 671.86 *kbytes*

再利用符号微分重复上述步骤：

```
> f22:=unapply(f(x,22),x):
> setbytes:=kernelopts(bytesused):
> settime():
> (D@@3)(f22)(1.1);
```

18.23670379

```
> cpu_time:=(time()-settime)*seconds;
```

$$cpu\_time := 119.160\ seconds$$

```
> memory_used:=evalf((kernelopts(bytesused)-setbytes)/1024*kbytes,5);
```

$$memory\_used := 63423.\ kbytes$$

显然Maple自动求导在计算时间和存储空间上具有明显的优越性。巧妙利用自动微分解决较复杂的函数求导问题有十分现实的意义。

3 子程序求值

在 Maple 子程序中的语句，求值的方式和交互环境中的求值有所不同，这在某种程度上是为了提高子程序的执行效率而考虑的. 在交互式环境中，Maple 对于一般的变量和表达式都进行完全求值(除了数组、映射表等数据结构外). 比如先将 b 赋给 a，再将 c 赋给 b，则最终 a 的值也指向 c.

```
> a:=b:
b:=c:
a+1;
```

$$c + 1$$

但在子程序内部情况就不一样了，试看下述实验：

```
> f:=proc()
   local a,b;
   a:=b;
   b:=c;
   a+1;
end:
```

结果居然是：

```
> f();
```

$$b + 1$$

这是因为 a 和 b 都是局部变量，Maple 对于子程序中的局部变量只进行一层的求值. 下面，我们针对子程序中的不同的变量，系统介绍其求值机制.

3.1 参数

子程序中的参数，就是那些在 proc( )的括号中的变量. 参数是一类特殊的变量，在调用子程序时，会把它们替换成实际的参数.

```
> sqrt1:=proc(x::anything,y::name)
     y:=x^2;
```

```
end:
> sqrt1(d,ans);
```

$$d^2$$

```
> ans;
```

$$d^2$$

我们再来试试别的参数，比如前面赋值过的 a 作为第一个参数，第二个参数仍用 ans，只不过加了单引号：

```
> sqrt1(a,'ans');
```

$$c^2$$

```
> ans;
```

$$c^2$$

事实上，Maple 在进入子程序以前就已经对参数进行了求值，因为是在子程序外进行求值，所以求值规则服从调用时所在的环境. 上面是在交互式环境下调用 sqrt1 得到的结果，作为对照看看在子程序内部调用上面的 sqrt1 子程序会有什么不同.

```
> g:=proc()
   local a,b,ans;
   a:=b;
   b:=c;
   sqrt1(a,ans);
end:
> g();
```

$$b^2$$

因为这次调用是在程序内部，所以只进行第一层求值，进入 sqrt1 的参数值为 b.

Maple 对于子程序的参数，都只在调用之前进行一次求值，而在子程序内部出现的地方都用这次求值的结果替换，而不再重新进行求值. 如前所述之 sqrt1：

```
> sqrt1:=proc(x::anything,y::name)
      y:=x^2;
      y+1;
end:
> sqrt1(d,'ans');
```

$$ans + 1$$

可见，对参数赋值的作用只有一个—返回一定的信息．因为在子程序内部，永远不会对参数求值．我们可以认为，参数是一个 0 层求值的变量．

3.2 局部变量和全局变量

Maple 对于局部变量只进行一层求值，也就相当于用 eval(a, 1)得到的结果．这种求值机制不仅可以提高效率，而且还有着更重要的作用．比如在程序中，我们往往需要把两个变量的值交换，一般地，我们使用一个中间变量来完成这样的交换．但如果求值机制和交互式环境中一样，将达不到我们的目的．试看，在交互式环境下，我们企图用这样的方法交换两个未被赋值的变量会有什么后果：

```
> temp:=p:
p:=q:
q:=temp;
```

$$q := q$$

不管在交互式环境下还是在程序中，Maple 对于每一个全局变量都进行完全求值，除非它是数组、映射表或者子程序．对于数组、映射表和子程序，Maple 采用赋值链中的上一名称来求值．

除了上面这些特殊数据对象及下面一个特例外，总结起来，Maple 对于子程序的参数进行 0 层求值，对于局部变量进行 1 层求值，而对于全局变量，则进行完全求值．

对于上面说的求值规则，在 Maple 中还有个特例，就是“同上”操作符“%”需要引起注意．就其作用来说，它应该属于局部变量．在进入一个子程序时，Maple 会自动地把该子程序中的%设置成 NULL(空)．但是，对于这个“局部变量”，Maple 不遵循上面的规则，无论在那儿都会将其完全求值．我们下面通过一个例子说明它在子程序中的求值机制：

```
> f:=proc()
   local a,b,c,d;
   print("At the beginning, [%] is", %);
   a:=b;
   b:=c;
   c:=d;
   a+1;
   print("Now [%] is",%);
end:
> f();
```

"At the beginning, [%] is"

"Now [%] is", $d + 1$

可以看出，尽管在子程序内部，局部变量a仅进行一层求值，但指代a+1的同上操作符却进行了完全求值，得到d+1的结果.

3.3 环境变量

所谓环境变量就是从一个程序退出时系统自动设置的全局变量。Maple中有几个内建的环境变量，如Digits，Normalizer，Testzero, mod, printlevel等. 从作用域来看，它们应该算作是全局变量. 在求值上，它们也像其他全局变量一样，始终都是完全求值. 但是如果在子程序中间对环境变量进行了设置，那么，在退出该子程序时，系统会自动地将其恢复成进入程序时的状态，以保证当前行时的环境. 正因如此，我们称其为环境变量. 试看下面程序：

```
> f:=proc()
   print("Entering f. Digits is", Digits);
   Digits:=20;
   print("Now Digits has become", Digits);
end:
> g:=proc()
   print("Entering g. Digits is", Digits);
   Digits:=100;
   print("Now Digits is", Digits);
   f();
   print("Back in g from f, Digits is", Digits);
end:
> f();
```

"Entering f. Digits is", 10

"Now Digits has become" , 20

```
> g();
```

"Entering g. Digits is", 10

"Now Digits is", 100

"Entering f. Digits is", 100

"Now Digits has become" , 20

"Back in g from f, Digits is" , 100

```
> Digits;
```

10

可以看出，从子程序 g 中返回时，Digits 又被自动地设成了原来的值(10). 如果你需要自己定义一些具有这样的特性的环境变量，也十分简单—Maple 会把一切以_Env 开始的变量认作是环境变量.

4 嵌套子程序和记忆表

4.1 嵌套子程序

很多情况下，可以在一个子程序的内部定义另一个子程序. 实际上，我们在写这样的程序时常常没有意识到它是嵌套子程序. 用于对每一个元素操作的 map 命令在嵌套程序设计中很有用. 比如，编写一个子程序，它返回有序表中的每一个元素都将被第一个元素除的结果.

```
> nest:=proc(x::list)
      local v;
      v:=x[1];
      map(y->y/v,x);
end:
> lst:=[2,4,8,16,32]:
> nest(lst);
```

[1, 2, 4, 8, 16]

在上面的程序中，我们在子程序中定义了另一个子程序: y->y/v，这个子程序中有一个变量 v, Maple 根据有效域的范围，认为它就是外面的子程序 nest 中的同名变量 v. 那么这是一个全局变量还是一个局部变量？显然两者都不是，因为把上面的例子中的内部子程序的 v 声明成 local 或者 global，都无法达到我们的目的.

那么, Maple 究竟是怎么样来判断一个变量的作用域呢？首先，它自里向外，一层一层地寻找显式地用 local、global 声明的同名变量，或者是子程序的参数. 如果找到了，它就将其绑定在外层的同名变量之上，实际上就是将两个变量视为同一，就像上面例子中的情况. 如果没有找到，就遵循下面的原则：如果变量位于赋值运算符“:=”的左边，就视其为局部变量，否则均认为它是全局变量. 再看下面一个实例：

```
> uniform:= proc(r::constant..constant)
    proc()
    local intrange, f;
    intrange:=map(x->round(x*10^Digits),evalf(r));
```

```
    f:= rand(intrange);
    evalf(f()/10^Digits);
  end:
end:
> U:=uniform(cos(2)..sin(1)): Digits:=4:
> seq(U(),i=1..10);
```

.4210, .7449, -.06440, .7386, .7994, -.2166, -.1174, .02740, .2570, .3538

4.2 记忆表

记忆表的目的是为了提高计算效率，它把每一个计算过的结果存储在一个映射表中，所以在下一次用相同的参数直接调用以避免重复计算.

(1) remember 选项

在程序中可以加入 remember 选项来建立该子程序的记忆表. remember 选项可以把对时间和空间的指数增长的需求约简到线性增长。下面再建立一个 Fibonacci 子程序：

```
> Fibonacci:=proc(n::nonnegint)
    option remember;
    if n<=1 then RETURN(n) fi;
    Fibonacci(n-1)+Fibonacci(n-2);
end:
```

在我们计算了 F3 之后，它的记忆表中就有了 4 项(记忆表中的子程序的第 4 个元素—用 op (4, …)来获得)，可以用 op 命令来检查一个子程序的记忆表：

```
> Fibonacci(3);
```

2

```
> op(4,eval(Fibonacci));
```

table([0 = 0, 1 = 1, 2 = 1, 3 = 2])

但是，过分地使用 remember 选项也并非全是好事，一方面，使用 remember 选项的程序对内存的需求比正确写出的不使用 remember 选项的程序要高，另一方面，remember 选项不重视全局变量和环境变量的值(除了在 evalf 中对 Digits 特别重视外)，从而可能导致错误。还有一点，remember 选项与表、阵列、向量不能混用，因为 remember 无法注意到表中元素的变化。

(2) 在记忆表中加入项

通常调用具有记忆表的子程序并且得到计算结果，Maple 会自动地把结果加到记忆表中去，但也可以手动在记忆表中添加内容. 试看下面的程序：

```
> F:=proc(n::nonnegint)
```

```
    option remember;
    F(n-1)+F(n-2);
end:
```

现在对记忆表赋值，注意这样赋值不会使程序被调用：

```
> F(0):=0;F(1):=1;
```

$$F(0) := 0$$

$$F(1) := 1$$

试调用程序得到想要的结果：

```
> F(100);
```

$$354224848179261915075$$

由于程序F的代码很短，很容易忘记对初始条件赋初值，这样会导致永不终止的递归，从而产生死循环. 因此，程序**Fibonacci**较F要好一些.

(3) **在记忆表中删除项**

记忆表是映射表的一种，可以方便地在其中添加或者删除特定的项，和一般变量一样，删除一个表项只需要将其名称用 evaln 赋给它本身就行了. 假设，因为输入错误，我们在前面一个程序中加入了一个错误的项：

```
> Fibonacci(2):=2;
```

$$Fibonacci(2) := 2$$

查看记忆表：

```
> T:=op(4,eval(Fibonacci));
```

$$T := table([2 = 2])$$

再将相应的项删除掉，则又回复原来的记忆表：

```
> T[2]:=evaln(T[2]);
```

$$T_2 := T_2$$

```
> op(4,eval(Fibonacci));
```

$$table([0 = 0, 1 = 1, 2 = 1, 3 = 2])$$

Maple 也可以自动删除子程序的记忆表，如果我们子程序定义时声明 system 选项，系统将会在回收无用内存时将记忆表删除，就如同大多数系统内部子程序一样.

对于记忆表的使用范围，一般地，它只适用于对于确定的参数具有确定的结果的程序，而对于那些结果不确定的程序，比如用到环境变量、全局变量，或者时间函数等的程序，就不能使用，否则将会导致错误的结果.

5 返回子程序的子程序

在所有的子程序中，编写返回值为一个子程序的程序所遇到的困难也许是最多的了．编写这样的程序，需要对 Maple 的各种变量的求值规则、有效语句有透彻的理解．Maple 一些内部子程序就有这样的机制，比如随机函数 rand，返回的就是一个子程序，它会在给定的范围内随机地取值．下面以牛顿迭代法和函数的平移为例说明这种程序设计技巧。

5.1 牛顿迭代法

牛顿迭代法是用来求解非线性方程数值解的常用方法之一．首先，选择一个接近于精确解的初值点，接着，在曲线上该初值点处作切线，求出切线与 x 轴的交点．对于大部分函数，这个交点比初值点更要接近精确解．依次，重复迭代，就可以得到更精确的点．牛顿迭代法的数学过程为：

对于方程 $f(x)=0$，选定初值 x_0，然后利用递推公式 $x_{k+1}=x_k-\dfrac{f(x_k)}{f'(x_k)}$ 来逐步得到精确解．试看下面求解方程 $x-2\sqrt{x}=0$：

```
> MakeInteration:=proc(expr::algebraic,x::name)
    local interation;
    interation:=x-expr/diff(expr,x);
    unapply(interation,x);
  end:
> expr:=x-2*sqrt(x);
```

$$expr := x - 2\sqrt{x}$$

```
> Newton:=MakeInteration(expr,x);
```

$$Newton := x \to x - \frac{x - 2\sqrt{x}}{1 - \frac{1}{\sqrt{x}}}$$

然后我们利用已经得到的迭代递推函数 Newton，可以用来求解这个超越方程：

```
> x0:=3.0;
```

$$x0 := 3.0$$

```
> to 4 do x0:=Newton(x0);od;
```

$$x0 := 4.098076213$$
$$x0 := 4.000579795$$
$$x0 := 4.000000019$$
$$x0 := 4.000000001$$

可以看到，经过 4 次迭代结果已经相当满意了.

在上面的子程序 MakeInteration 的输入参数类型是代数表达式，我们可以编制用函数作为参数的子程序. 由于函数的求值具有特殊性，在程序中需要用 eval 对其进行完全求值.

```
> MakeInteration:=proc(f::procedure)
    (x->x)-eval(f)/D(eval(f));
end:
```

由于输入参数是一个程序，就不需要再显式地指出自变量了，我们用 D 运算符求它的导函数，并且作运算. 这里，**(x->x)**表示恒等函数 $f(x)=x$. 从这里，我们也可以知道，在 Maple 中，不仅代数表达式可以进行运算，具有相同自变量的函数也可以相互进行运算. 利用上面的程序求解超越方程 $x^2-\cos x=0$：

```
> g:=x->x^2-cos(x);
```

$$g := x \to x^2 - \cos(x)$$

```
> Newton:=MakeInteration(g);
```

$$Newton := (x \to x) - \frac{x \to x^2 - \cos(x)}{x \to 2\,x + \sin(x)}$$

```
> x0:=1.0;
```

$$x0 := 1.0$$

```
> to 4 do x0:=Newton(x0);od;
```

$$x0 := .8382184099$$
$$x0 := .8242418682$$
$$x0 := .8241323190$$
$$x0 := .8241323123$$

5.2 函数的平移

考虑这样一个简单的问题，已知一个函数 f，我们需要得到另一个函数 g，满足

$g(x)=f(x+1)$. 在数学中, 这样的操作称为平移. 在 Maple 中, 可以用简单的程序实现:

```
> shift:=(f::procedure)->(x->f(x+1));
```

$$shift := f::procedure \to x \to f(x+1)$$

```
> shift(sin);
```

$$x \to \sin(x+1)$$

```
> shift((x)->x);
```

$$x \to (x \to x)(x+1)$$

```
> %(1);
```

$$2$$

```
> shift:=proc(f::procedure)
    local x;
    unapply(f(x+1),x);
end;
> shift(sin);
```

$$x \to \sin(x+1)$$

```
> h:=(x,y)->x*y;
```

$$h := (x, y) \to x\,y$$

```
> shift(h);
Error, (in h) h uses a 2nd argument, y, which is missing
> shift:=(f::procedure)->(x->f(x+1,args[2..-1]));
```

$$shift := f::procedure \to x \to f(x+1, \mathrm{args}_{2\,..\,-1})$$

```
> shift(sin);
```

$$x \to \sin(x+1, \mathrm{args}_{2\,..\,-1})$$

```
> hh:=shift(h);
```

$$hh := x \to h(x+1, \mathrm{args}_{2\,..\,-1})$$

```
> hh(x,y);
```

$$(x+1)\,y$$

```
> h:=(x,y,z)-> y*z^2/x;
```

$$h := (x, y, z) \to \frac{y\, z^2}{x}$$

```
> hh(x,y,z);
```

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

```
> shift := proc(f::procedure)
    loca l F;
    subs('F'=eval(f),x->F(x+1,args[2..-1]));
end;
> H:=shift(h);
```

$$H := x \to \left((x, y, z) \to \frac{y\, z^2}{x} \right)(x + 1, \mathrm{args}_{2\,..\,-1})$$

```
> h:=45;
```

$$h := 45$$

```
> H(x,y,z);
```

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

```
> f:=proc(x) x^2 end;
```

$$f := \mathbf{proc}\ (x)\ x\text{^}2\ \mathbf{end\ proc}$$

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

$$\mathbf{proc}\ (x)\ x\text{^}2\ \mathbf{end\ proc}$$

6 局部变量的进一步探讨

局部变量的作用域是当前的子程序，而且，对于该子程序的每一次运行，局部变量都是不同的变量. 简单来说，每一次调用子程序，都产生一些新的变量；如果你两次调用相同的子程序，那么，第二次所用的局部变量和第一次的局部变量是不同的.

有时候，在退出一个子程序时，局部变量并不消失. 比如在子程序中将局部变量作为结果返回了，那么在子程序结束后，这些局部变量仍然存在. 对于这些变量，常常难以捉摸，因为它们可以和全局变量同名，但却都是不同的变量—它们在系统内存中占有不同的位置，修改其中一个变量的值不会影响到其他值.

为了对这种情况有一定的了解，我们首先定义一个返回局部变量的子程序：

```
> make_a:=proc()
    local a;
    a;
end;
```

检查所产生的变量是否相同，可以根据中间元素的唯一性(自动删除相同的元素).

```
> test:={a,a,a};
```

$$test := \{a\}$$

```
> test:=test union {make_a()};
```

$$test := \{a, a\}$$

每次调用程序 make_a 生成的同名变量，是互不相同的，而且，它们和全局变量 a 也不相同. 注意：Maple 并不是只依据变量名来区别变量的. 在交互式环境中键入变量 a 时, Maple 认为它是全局变量. 所以，可以用上面的同名变量集合 test 中很容易地找到全局变量.

```
> member(a,test,num);
```

$$true$$

```
> num;
```

$$1$$

同名变量的最大用途在于可以用来写出一串通常用 Maple 很难得到的表达式. 举例来说, Maple 会自动地把表达式 a+a 代简成 2a，要得到表达式 a+a=2a 是一件不容易的事. 现在，我们可以利用上面的子程序 make_a 来轻松地写出这样的式子.

```
> a+make_a()=2*a;
```

$$a + a = 2\,a$$

对于 Maple 来说，虽然它们具有相同的变量名，但等号左边的两个 a 是不同的，所以，它没有自动地把它化简成 2a. 对于全局变量，可以用变量名直接引用，于是对于另一个用 make_a 得到的变量，它的引用就需要费一番周折. 我们可以通过 remove 命令去掉等式左边的全局变量 a 来得到另一个 a.

```
> eqn:=%;
```

$$eqn := a + a = 2\,a$$

```
> another_a:=remove(x->eval(x=a),lhs(eqn));
```

$$another\_a := a$$

现在全局变量 another_a 指向了那个难以捉摸的 a，我们要做的是把那个 a 赋成全局变量 a，用赋值语句显然是不行的，我们用 assign 命令将其赋值. assign 命令和赋值语句的不同在于被赋值的变量是作为参数传给 assign 的，所以 Maple 会自动地将它求值为它所指的 a，这样，就可以为 a 赋值了.

```
> assign(another_a);
> eqn;
```

$$a + (\ ) = 2\,a$$

这时，等式两边都是作为全局变量的 a 了．我们用 evalb 可以检验等式的正确性(虽然这是显然的，但对于复杂的等式，这却是有必要的).

```
> evalb(%);
```

$$false$$

在这一小节中，我们引入了令人费解的“越界”的局部变量．而实际上，可能在以前的学习中就已经碰到过这样的问题．已经学习过的assume命令，它将一个变量赋予另一个有确定范围的变量—新的变量只是在原来的变量后面加上一个“~”．这个具有波浪线的变量就是一个“越界”的局部变量—如果在交互式环境中输入它(注意：需要用一对反向撇号“`' '`”括起来), Maple将不能识别，因为编译程序认为输入的是全局变量.

```
> assume(b>0);
> x:=b+1;
```

$$x := b\sim + 1$$

```
> subs('b'=c,x);
```

$$b\sim + 1$$

Maple所做的是把局部变量b~赋给全局变量b．如果我们将b另外赋值，b~仍然存在，则由它所生成的表达式也仍然存在.

```
> b:='b';
```

$$b := b$$

```
> x;
```

$$b\sim + 1$$

7 扩展 Maple 命令

虽然编写程序可以满足各种不同的需要，但是有时候，根据需要扩展 Maple 原有命令可以事半功倍，事实上，Maple 更大的特点提供了一种“平台”.

在大多数现代编程语言中，都支持自定义的数据结构和数据类型，Maple 也具有这样的功能，只需将一个结构类型赋值给`type/TypeName`， TypeName 就可以作为一个类型使用了．这样，对于结构本身只需要写一次，减少了出错的可能性，也减少了工作量.

```
> `type/Variables`:={name,list(name),set(name)};
```

$$type/Variables := \{name, \mathrm{list}(name), \mathrm{set}(name)\}$$

```
> type(x,Variables);
```

$$true$$

```
> type({x[1],x[2]},Variables);
```

true

在这个例子中，我们把几个类型的集合赋给了`type/Variables`，也就是说我们定义的数据类型 Variables 是这几个类型的集合. 所以，不管单个变量还是变量的集合，都是 Variables 类型的.

另外，还可以将一个子程序赋给`type/TypeName`. 在测试一个数据对象是否具有 TypeName 类型时，Maple 会自动调用该子程序. 但是子程序必须返回布尔值(true 或者 false).

作为例子，下面定义一个全排列(permutation)的数据类型，也就是检测一个有序表是否含有从 1 到 n 的所有自然数(且每一个只出现一次). 提取有序表中所有元素的集合，再与 1 到 n 的自然数集比较，就可以实现这一要求.

```
> `type/permutation`:=proc(p)
    local i;
    type(p,list) and {op(p)}={seq(i,i=1..nops(p))};
end;
> type([2,3,1,4],permutation);
```

true

```
> type([1,2,3,1],permutation);
```

false

自定义的类型检测函数可以具有多于一个的参数. 比如，要检测一个表达式 expr 是否具有类型 TypeName(parameters)，Maple 就会用如下形式调用检测函数 TypeName(expr,parameters). 例如，定义一元线性表达式类型 LINEAR，它是一个未知变量的一次多项式.

```
> `type/LINEAR`:=proc(f,v::name)
    type(f,polynom(anything,v)) and degree(f,v) =1;
end;
```

type/LINEAR :=

proc (*f*, *v*::*name*) type(*f*, polynom(*anything*, *v*)) **and** degree(*f*, *v*) = 1 **end proc**

```
> type (x^2,LINEAR(x));
```

false

```
> type (a*x+b,LINEAR(x));
```

true

8 程序调试

任何语言编写的程序都不能保证没有错误，Maple 也一样。有时候错误非常隐蔽，

仅仅通过检查源程序或者数值试验可能无法排队程序中的错误。Maple 中提供了一些实用的调试工具。

Maple 的主要调试工具之一是 **printlevel**, 该工具对程序的执行过程产生详细的计算，包括程序调用的参量、子程序计算步骤和结果. 可以合理地设置 **printlevel** 来控制调试的深度(其默认值为 1), 然后直接调用所需调试的程序即可看到程序的执行过程. 此时, 将打印出程序内的代码. 当 **printlevel** 的值增加, 附加的信息将会被打印出来, 这些信息当然对程序调试是重要的.

另一个可选择的调试工具是使用函数 **trace**, 该函数以被跟踪的函数作为参量. 除了指定程序的参量和结果, 跟踪过程会提供每一语句的执行结果. 除此之外在 trace 的输出上没有别的控制. 函数 untrace 被用作去掉 trace 的影响.

命令 **showstat** 可以显示源程序、语句对应的编号、断点的设置、以及当前的中断位置。它的调用格式为 **showstat**(*procdure,number*)，其中 procedure 是需要显示的子程序名称。对于无条件断点，showstat 将在其语句编号后显示星号“*”；对于条件断点，则显示问号“？”。可选参数 number，指定语句号或者语句号的范围。这时，将只显示所指定的语句，其余的语句都用“…”表示。

而函数 **stopat** 设置断点，它的调用格式为：**stopat**(*procedure, number ,condition*)，其中 procedure 是子程序名称；number 是需要设置中断的语句编号，它可以省略，默认情况下将在该程序的第一条语句之前设置断点；condition 是在该断点处中断的条件，它是一个布尔表达式，可以包含任意全局变量，该子程序中的局部变量和参数，在省略这一参数时，stopat 将设置无条件断点。无条件断点的语句号后面将用星号标记，而条件断点则吉林省记以问号。

命令 **unstopat** 用来清除程序中的断点，它的调用格式为：unstopat(procedure , number)，其中 number 是需要删除的断点的语句编号，也是可选参数，如果省略，将删除该程序中所有的断点。

在用户自己编写的源程序中，也可以用调用函数 DEBUG 的方法加入一个显式的断点：**DEBUG**(*condition*)，其中 *condition* 是一个布尔表达式，表示中断的条件，在它的值为 true 时引起中断，如果为 false 或者 FAIL 时将忽略这一中断语句；它是可先参数，默认情况下交导致无条件中断。DUBEG 的参数也可以是非布尔类型的表达式，这使用程序将无条件中断，并且显示这个表达式的值。

显式断点的调用结果和调试断点的结果相同，都是使程序中断，并显示上一执行语句的结果以及下一条将要执行的语句。

监视断点可以对变量进行监视，用命令 stopwhen 加入，它有两种调用格式：**stopwhen**(*globalVariableName*)或 **stopwhen**([*procedure* , *VariableName*]). 第一种调用设置了对全局变量 *globalVariableName* 的监视，还可以用它来设置对环境变量的监视；第二种调用则在程序 procedure 内部监视变量 *VariableName*（可以为全局变量或者局部变量）。

在 Maple 对所监视的变量进行赋值以后，监视断点就中断程序的运行，并显示对应的赋值语句（对于所赋的值已进行化简），而不是显示结果。然后，依照惯例显示下一

条语句。需要注意的是，监视断点在赋值之后引起中断，而不是赋值以前。

要清除监视断点，可以调用命令 unstopwhen，它的参数和 stopwhen 完全一样，也可以在调试环境中调用。如果不提供任何参数，unstopwhen 将清除所有的监视断点。

出错断点可以截获 Maple 的错误住处并引起程序中断，进入调试状态。命令 stoperror 可以设置出错中断，它的调用格式为 **stoperror** (“*error message* “)，其中 *errormessage* 指定了需要截获的 Maple 出错信息，也可以使用 all 作为参数，以截获所有的错误。stoperror 的返回值是反有设置出错断点的错误有序表.

Maple 中还有一个实用工具 **mint**，它可以检查 Maple 句法错误, 而且比运行调试工具速度要快得多. 另外，该函数还可以检查程序中是否有全局变量以及所有的局部变量是否都使用了.

虽然 Maple 提供了功能强大的程序调试工具，利用它可以监视程序的内部执行过程，可以更快地找到程序的错误并加以修改. 但是，由于程序设计本身的复杂性，因此，在编写程序时应严格按照相关算法设计程序，万勿把全部希望寄托在程序的调试上.