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);
```

$$e$$

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

$$2.718281828$$

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

$$-1.511772633 + 3.303283467\ I$$

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

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

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);
```

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

```
> 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);
```

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

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

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

上式的映射关系可通过下式理解：

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

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

```
> 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;
```

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

```
> 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;
```

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

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

$$-7 + e$$

```
> 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);
```

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

```
> 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.
```

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

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

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

```
> 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);
```

$$\alpha := \mathrm{RootOf}(\_Z^2 - 3)$$

```
> 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)));
```

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

```
> 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);
```

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

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

$$e^{x}$$

```
> 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);
```

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

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

$$T_3 := T_3$$

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

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

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);
```

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

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);
```

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

```
> 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);
```

$$\text{myfunc}(6, 8)$$

```
> 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:

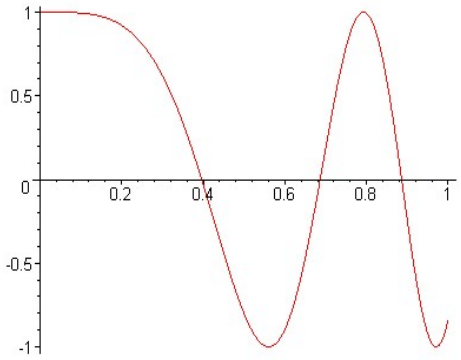
```
> mypts:=[seq([x/1000,cos(x^2/100000)],x=1..1000)]:
> writedata("e:\\data.txt",evalf(mypts));
> dots:=readdata("e:\\data.txt",100):
> nops(dots);
```

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;
```

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

```
> 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)
```