利用 Maple

学习微积分

文中引用的 “教材” 指:

柴俊主编《高等数学》(科学出版社, 2007)

目　录

利用 Maple 学习微积分

赵书钦 编写

1 认识 Maple

Maple 是世界上享有盛誉的数学软件之一, 擅长做符号运算, 此外还可做任意精度的数值计算, 绘制二维和三维图形和动画, Maple 还提供建模、程序设计、仿真模拟、二次开发、与其他应用程序无缝连接等功能, 还可以编写技术文档、演示报告等. Maple 覆盖了所有的数学领域, 包括微积分、代数、微分方程、统计、线性代数、几何、变换等. Maple 操作简单, 数学元素的表示形式与高等数学中的习惯基本一致, 因而容易上手, 不易忘记, 是教师、学生、科研工作者和技术人员理想的数学工具. Maple 提供的 student 程序包 (也称函数包, 软件包), 是大学生学习微积分的好帮手.

与 Maple 齐名的还有 Mathematica, Matlab, MathCAD 等数学软件. Mathematica 与 Maple 一样是符号运算系统 (当然也可做数值计算), 功能强大, 但它用 f[x] 表示函数 f(x), 除非经常使用, 否则很容易写错, 给用户带来不便. Matlab 是著名的数值计算软件, 丰富的工具箱涵盖了方方面面, 较新的版本也可进行符号运算了, 这是通过调用 Maple 的符号运算软件包实现的, 因而在这方面不会超过 Maple. MathCAD 对工程技术人员很合适, 但若用于学习微积分, 则比 Maple 逊色了. 根据以上比较, 对大学生来说, 选用 Maple 是较为合适的.

值得一提的是现在已有一些免费的开源自由软件如 YACAS, Scilab 等, 在一定程度上可以取代上述几种商业软件. YACAS (Yet Another Computer Algebra System) 主要用于符号运算, Scilab 可以媲美 Matlab, 时机成熟时, 应在各个领域特别是在教育和学术研究机构大力推广应用这些开源软件.

考虑到 Scilab 缺少符号运算功能, 而 YACAS 还处在发展阶段, 所以我们目前仍用 Maple 做数学实验.

每过一两年, Maple 就会发布一个新版本, 本书交稿时已是 Maple 11 了, 本文使用的是 Maple 9 版本, 安装后会在桌面上生成两个图标, 红色枫叶的图标对应于可执行文件 maplew9.exe, 黄色的对应于 cwmaple9.exe, 后者适于内存较小的低档计算机. 点击 maplew9.exe, 出现窗口 (图 1), 这是 Maple 的 Worksheel Mode, 可称为工作页 (或工作面, 工作区, 工作表) 模式.

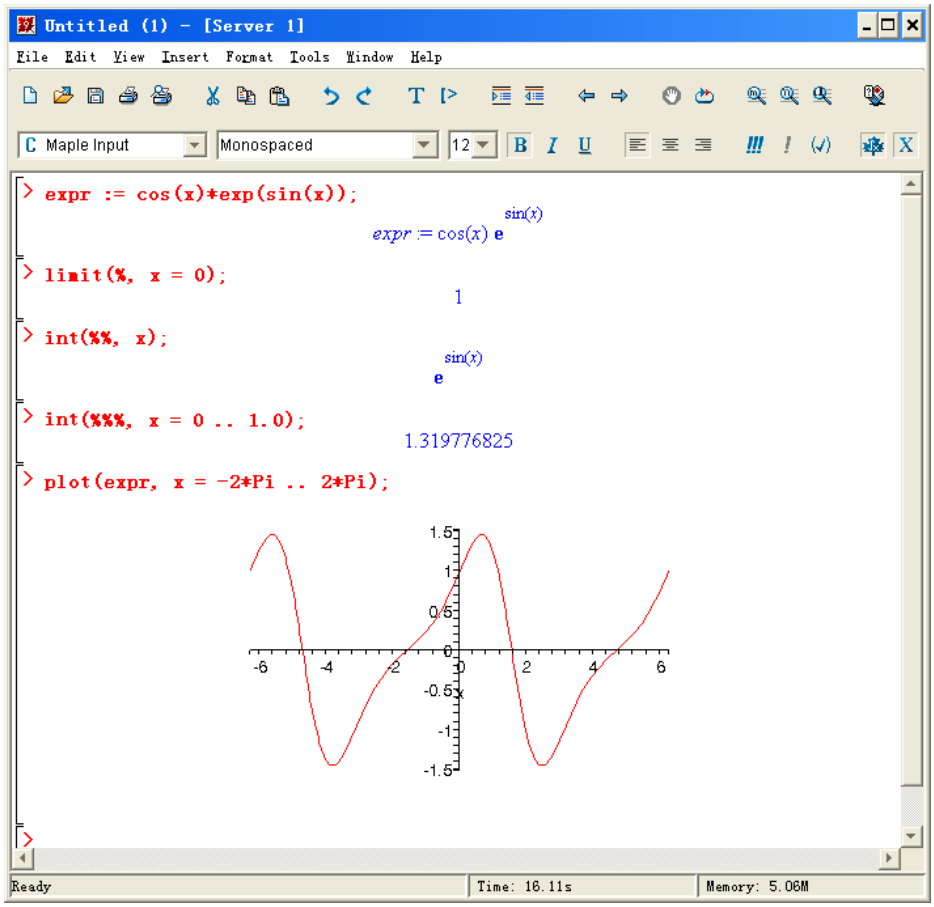


图 1: Maple 窗口

在窗口左上角的红色提示符 “>” 之后键入 Maple 语句 (默认变成红色), 并以分号 “;” 结尾, 回车后 Maple 将运行该语句, 并显示结果 (蓝色); 若输入的命令以冒号 “:” 结尾, 则 Maple 仍会运行该语句, 但不显示结果. 在窗口左侧有一个方括号, 它会把输入和输出括在一起, 构成一个组, 图 1 中显示了几个例子, 其中符号 “%” 是 “同上” 符号, 用于引用最近一次的运算结果, “%%” 表示倒数第二次运算结果, “%%%” 表示倒数第三次结果 (不能再多了).

“:=” 是命名或赋值符号. 对于 Maple 的各种对象都可命名一个名字, 简单的名字以字母开头, 后面可以跟随零个或多个字母、数字、下划线. 名字的最大长度取决于硬件系统, 在 32 位的系统上为 524 271 个字符, 在 64 位系统上是 34 359 738 335 个字符, 因此在命名时, 实际上是不必关心名字长短的.

以 _ENV 开头的名字是环境变量, 其它以下划线开头的名字被 Maple 用于全

局变量. 用倒引号 “`” 括起来的任意一串字符也可作为名字. 此外要特别提醒注意, Maple 是区分大小写的.

已有其他数学软件经验的人, 使用 Maple 时要特别注意 3 点:

- 赋值或命名使用 “:=” 号, 而不是 “=” 号;
- 表达式中用到乘法时, 乘号不可省略, 例如 2x 应写成 2*x;
- 表示范围时, 两个端点值之间用范围号 (双点“..”) 连接, 不是用逗号分隔.

1.1 学会求助

好的软件都有帮助系统, 遇到问题时要学会向软件本身求助. 点击 “Help” 菜单, 选择 “Quick Tour” 或 “Full Tour” (图 2), 就可以对 Maple 做快速或全面

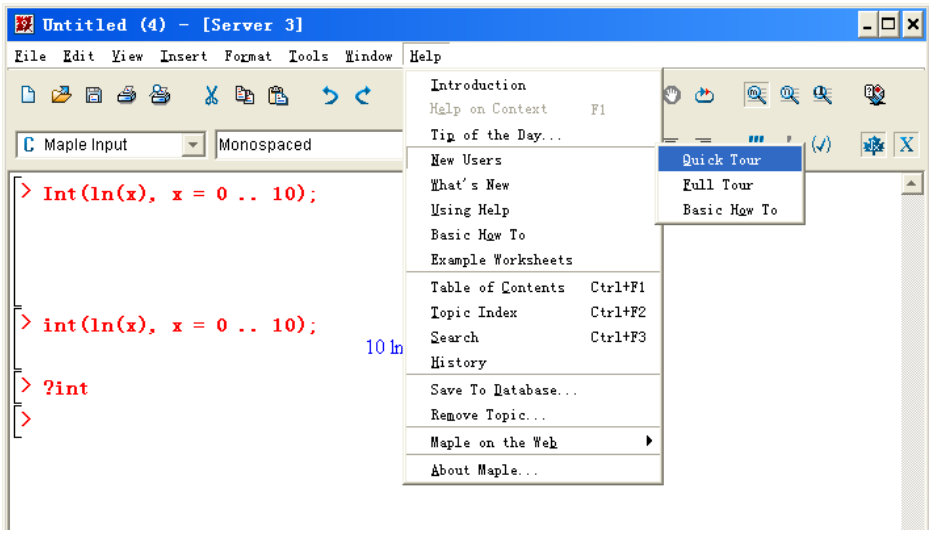


图 2: “帮助” 菜单

的漫游. 如果需要详细了解某个命令或函数, 则在提示符后输入问号和名字, 例如输入:

```
>?int
```

Maple 就会对积分函数 int 作详细的解释 (图 3).

1.2 各种符号

Maple 使用的字符集包括 26 个大写英文字母, 26 个小写英文字母, 10 个数字以及一些有特定用途的符号:

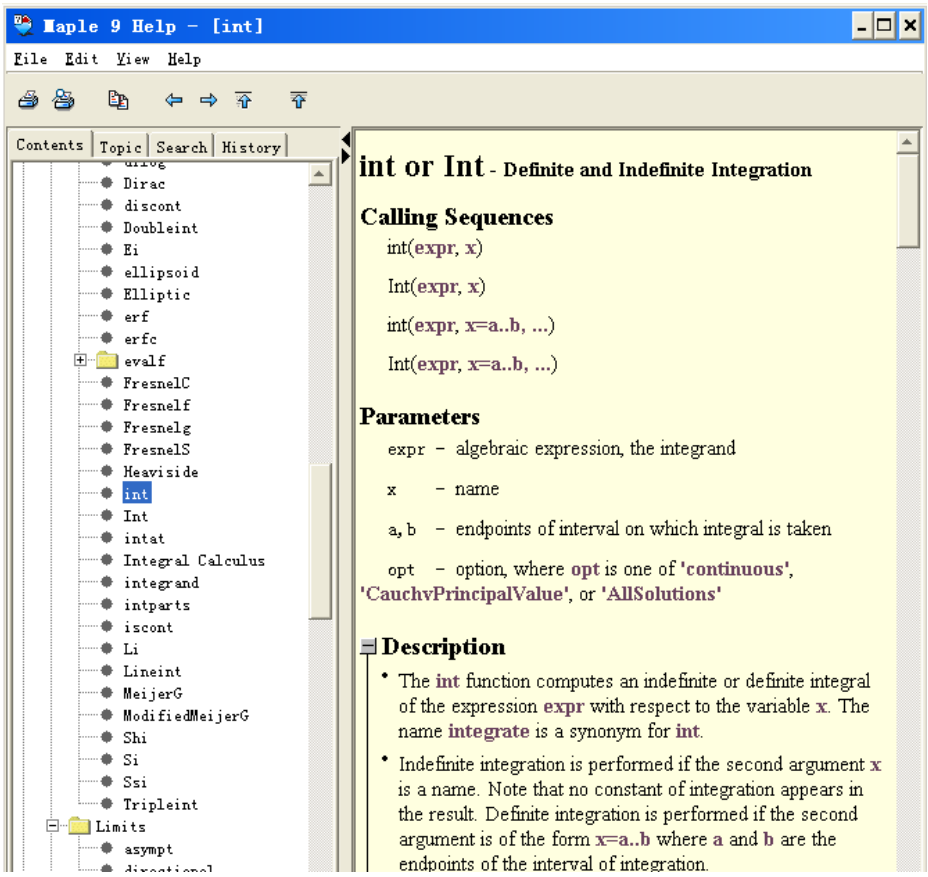


图 3: 命令 解释示例

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9

; : + − * / ^ ! = < > @ $. , ?

( ) [ ] { } ` ′ ″ | & _ % \ #

下面是 Maple 使用的一些运算符和几个常数.

表 1: 一元运算符

+	正号	not	逻辑否
−	负号	&	某些运算符前缀
!	阶乘号		

表 2: 二元运算符

算符	含义	算符	含义
+	加	<	小于
−	减	<=	小于或等于
*	乘	>	大于
/	除	>=	大于或等于
^	乘方	<>	不等于
:=	命名或赋值	=	等于
@	函数复合	−>	箭头 (用于定义函数)
@@	函数的重复复合	union	(集合的) 并
,	表达式的分隔符	minus	(集合的) 差
.	非交换乘法	intersect	(集合的) 交
..	范围	and	逻辑与
$	重复	or	逻辑或
::	过程的类型声明	xor	逻辑异或
||	名字或字符串的连接符	implies	蕴含
mod	模	subset	子集

表 3: 一些常数

常数	例子或名称 (在 Maple 中的名字)
整数	`2007`, `-1234`
有理数	$\frac{113}{355}$, $-\frac{2}{3}$
浮点数	`1.0`, `2.`, `3.45e-67`
	`Float(3.45,-67)`, `SFloat(3.45,-67)`
e (自然对数的底)	`exp(1)`
π (圆周率)	`Pi`
$\sqrt{-1}$ (虚数单位)	`I`
∞ (无穷大)	`infinity`
Catalan 常数 $c = \sum_{i=0}^{\infty} \frac{(-1)^i}{(2i+1)^2}$	`Catalan`
Euler 常数 $\gamma = \lim_{n\to\infty}(\sum_1^n \frac{1}{i} - \ln n)$	`gamma`

逻辑值: 真, 假, 未知	true, false, FAIL
空序列	NULL
未定义	undefined
符号常数序列 constants	false, γ, ∞, true, Catalan, FAIL, π

1.3 一些常用命令和函数

表 4: 一些常用命令和函数

命令或函数	作用
with	加载程序包
readlib	从程序库中读入过程
restart;	恢复初始状态
assume(表达式,性质)	设置 表达式 具有给定的 性质
is(表达式,性质)	测试 表达式 是否具有给定的 性质(返回逻辑值)
is(表达式)	测试 表达式 的真假(返回逻辑值)
Digits:=30	设置数值计算的有效数字位数(本示例为30位)
eval(表达式)	对 表达式 求值(表达式含浮点数时则自动作浮点运算)
eval(表达式,n)	对 表达式 (或名字) 求值, 计算到n层
eval(表达式,x=a)	计算 表达式 在x=a点的值
eval(表达式,方程组)	计算 表达式 在方程组确定的多个点上的值
evalf(表达式)	对表达式作浮点运算(默认10位精度)
evalf[n](表达式)	对表达式作n位精度的浮点运算
evalc(表达式)	计算表达式的实部和虚部
nops(表达式)	表达式的操作数(即运算分量)的个数
op(表达式)	表达式的各个操作数
op(i,表达式)	表达式的第i个操作数(i=0时的意义见帮助)
	(i>0时从左向右数, i<0时从右向左数)
op(i..j,表达式)	表达式的第i到第j个操作数
op([位置序号],表达式)	表达式中由[位置序号]指定的操作数
subs(旧=新,表达式)	在 表达式 中以 新表达式代换 旧表达式
subs(旧1=新1,...,	

旧n=新n，表达式)	在表达式中依次以新代换旧
subsop(i_1=新1,..., i_n=新n，表达式)	在表达式中将第 i_k 个操作数用新k代换
simplify(表达式)	化简表达式
simplify(表达式,选项)	根据选项化简表达式
map(过程,表达式)	将过程应用到表达式的各个操作数上
map2(过程,参数,表达式)	以参数作为第一参数,表达式的操作数 作为第二参数，应用过程
unassign('名字') 或 名字:='名字'	取消对指定名字的赋值或定义 例： unassign('f'); x:='x';

1.4 常用数学函数

在提示符后如下输入

```
>?inifcns
```

可看到 Maple 提供的数学函数清单, 若输入问号和函数名, 就可得到关于该函数的详细解释. 下面列出一些常用的数学函数.

表 5： 常用数学函数

函数	在 Maple 中的名字
$\mathbf{e}^x$（指数函数）	exp(x)
$\ln x$（自然对数）	ln(x) 或 log(x)
$\lg x$ 即 $\log_{10} x$（常用对数）	log10(x) 或 log[10](x)
$\log_b x$（以 b 为底的对数）	log[b](x)
$\sqrt{x}$（平方根）	sqrt(x)
$\sqrt[n]{x}$（n次根）	surd(x,n) 或 root(x,n) 或 root[n](x) (surd 与 root 有区别，见帮助)
$\|x\|$（绝对值）	abs(x)
极小值，极大值	min(x1,x2,...), max(x1,x2,...)
$\max\limits_{x\in[a,b]}\{\|f(x)\|\}$	numapprox[infnorm](f(x),x=a..b,'xmax')
把 x 四舍五入到整数	round(x)

把 x 截尾成整数	`trunc(x)`
x 的分数部分	`frac(x)`
x 的上整数 $\lceil x \rceil$，下整数 $\lfloor x \rfloor$	`ceil(x), floor(x)`
最大公因式，最小公倍式	`gcd(a,b), lcm(a,b)`
a 与 b 模 p 的最大公因式	`Gcd(a,b) mod p`
符号函数	`signum(x), signum(1,x), signum(0,x,y)`
三角函数 (角的单位用弧度)	`sin(x), cos(x), tan(x),`
	`sec(x), csc(x), cot(x)`
双曲函数	`sinh(x), cosh(x), tanh(x),`
	`sech(x), csch(x), coth(x)`
反三角函数	`arcsin(x), arccos(x), arctan(x),`
	`arcsec(x), arccsc(x), arccot(x),`
	`arctan(y,x)`
反双曲函数	`arcsinh(x), arccosh(x), arctanh(x),`
	`arcsech(x), arccsch(x), arccoth(x)`
$\Gamma(x)$	`GAMMA(x)`
组合数 $\binom{n}{r}$ 或 C_n^r	`binomial(n,r)`
模余运算	`a mod b` 或 `` `mod`(a,b) ``
正余数 (0,...,b-1)	`modp(a,b)`
对称余数 $\left(-\left[\frac{\lvert b\rvert-1}{2}\right], \cdots, \left[\frac{\lvert b\rvert}{2}\right]\right)$	`mods(a,b)`
$a^b \bmod p$	`a &^ b mod p` 或 `Power(a,b) mod p`
第 n 个Bernoulli数和多项式	`bernoulli(n), bernoulli(n,x)`
第 n 个Euler数和多项式	`euler(n), euler(n,x)`

1.5 程序库

1.5.1 使用程序库中的程序

Maple 的程序库分为 3 组: 主库、附属库 (亦称杂库) 和程序包 (亦称函数包或软件包). 主库包含最常用的命令或函数 (通常定义为一个过程), 可直接使用. 附属库包含一些不太常用的命令, 使用其中的命令需先将其调入内存: `readlib(命令)`. 大量的程序分门别类地放在不同的程序包中, 使用程序包中的命令也要先将其调入内存, 有三种方式:

• with(包名) 将整个包调入内存, 包中命令在当前窗口中一直可用, 因而使用方便, 缺点是占用内存较多.

• with(包名, 命令名) 将包中指定命令调入内存, 在当前窗口中一直可用. 但若需要包中其他命令, 还要运行调入步骤.

• 包名[命令名](命令参数) 调入包中指定的命令完成一件任务, 仅当次有效, 优点是占有内存很少.

前两种方法加载程序包后, 直接使用命令名即可, 称为命令的"短格式", 第三种需要写上包名, 称为"长格式".

1.5.2 显示过程的编码内容

showstat(过程名)

showstat(过程名, 行号范围)

只有一个参数时显示过程的全部语句, 每行可能有多个语句, 行前带有编号; 第二个参数指定显示行号范围中的语句, 其余语句用省略号代替.

下面显示 exp 的编码, 由于太长, 这里指定显示 3 行:

```
>showstat(exp,1..3);
exp := proc(x::algebraic)
local i, t, q, n, f, r;
   1   try
   2     return _Remember(('procname')(args))
       catch :
   3     NULL
       end try;
       ...
end proc
```

使用 print 命令也可以显示过程的编码内容, 通常是简略显示:

```
>print(exp);
```

$$\mathbf{proc}(x :: algebraic) \ \ldots \ \mathbf{end\ proc}$$

加大交互界面显示信息的详细程度 (设置为 2 或 3), 就可看到完整程序了, 显示时不带行号, 显示形式与上一命令的显示略有不同:

```
>interface(verboseproc=2);
>print(exp);
proc (x::algebraic)
```

```
local i, t, q, n, f, r;
options system,
        'Copyright (c) 1992 by the University of Waterloo.
        All rights reserved.';
        try
                return _Remember('procname'(args))
        catch:
        end try;
```

(显示内容有一页多, 此处略去)

1.5.3 显示过程运行时的附加信息

加大信息显示水平的值 (整数, 最大为 5), 可以看到附加信息.

```
>infolevel[simplify] := 2;
```

$$infolevel_{simplify} := 2$$

```
>exp(sin(x)^2+cos(x)^2);
```

$$\mathbf{e}^{(sin(x)^2+cos(x)^2)}$$

```
>simplify(%);
simplify/do:, applying commonpow function to expression
simplify/do:, applying power function to expression
```

$$\mathbf{e}$$

当已运行了一个命令 (过程), 然后设置该命令的信息显示水平, 再次运行这个命令就不会得到附加信息, 此时可让系统忘记前面已运行过该命令, 例如:

```
>forget(simplify);
```

1.6 数据类型

简单的数据类型是整数类型 (integer), 浮点数类型 (float) 和字符串类型 (string, 用双引号括起来的一串字符). 由简单类型可以衍生复杂的类型: 表达式序列 (exprseq), 顺序表 (list, 也称列表), 集合 (set), 阵列 (array) 和表 (table).

注意, 把一串字符用双引号括起来就变成字符串, 若用单引号括起来则是禁止求值, 若用倒引号 (重音号, “`”) 括起来则是当作符号, 可用作对象的名字.

函数 `whattype(表达式)` 返回表达式的类型.

```
>whattype("abc");
```

$$string$$

```
>whattype(`abc`);
```

$$symbol$$

1.6.1 表达式序列 (exprseq)

表达式序列简称序列, 就是一组用逗号分隔的表达式, 例如

```
>3,1,2,x,sin(y*z);
```

$$3, 1, 2, x, \sin(yz)$$

表达式序列保持输入的顺序, 序列中可以有相同的表达式. 运算作用于序列上时, 一般是作用于序列的各个元素上:

```
>S := 4,3,2,1,2,3,4;
```

$$S := 4, 3, 2, 1, 2, 3, 4$$

```
>a||S;
```

$$a4, a3, a2, a1, a2, a3, a4$$

用表达式或函数来生成序列的工具是 seq.

```
>seq(i^2, i=0..6);
```

$$0, 1, 4, 9, 25, 36$$

```
>X := seq( i, i=0..6 );
```

$$X := 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6$$

```
>Y := seq( i^2, i=X );
```

$$Y := 0, 1, 4, 9, 16, 25, 36$$

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

$$"\mathrm{H}",\ "\mathrm{e}",\ "\mathrm{l}",\ "\mathrm{l}",\ "\mathrm{o}"$$

```
>seq( i, i="a".."f" );
```

$$"\mathrm{a}",\ "\mathrm{b}",\ "\mathrm{c}",\ "\mathrm{d}",\ "\mathrm{e}",\ "\mathrm{f}"$$

seq 参数中的计数器变量是局部变量, 不受外部同名变量的影响.

1.6.2 顺序表 (list)

将序列用方括号括起来就成为顺序表 (也称列表), 顺序表保持元素的顺序和重复的元素. 使用 nops 命令可以求出顺序表中元素的个数 (不能用于序列):

```
>L := [S];
```

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

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

$$7$$

op 命令析取表达式的各个操作数 (也称为表达式的成分或运算分量), 因此可以用 op 命令将顺序表转换成序列:

```
>op(L);
```

$$4, 3, 2, 1, 2, 3, 4$$

空顺序表为 [].

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

$$[]$$

1.6.3 集合 (set)

用花括号括起来的表达式序列就是集合, 与数学上的集合相同, 集合不保留重复的元素, 也不保持元素的输入顺序. 但 Maple 按系统内部规则为集合的元素设定了索引顺序.

```
>S;
```

$$4, 3, 2, 1, 2, 3, 4$$

```
>data_set := {S};
```

$$data\_set := \{1, 2, 3, 4\}$$

```
>nops(data_set);
```

$$4$$

使用索引序号可以从表达式序列、顺序表或集合中析取元素, 上述结构的第一个元素的索引序号为 1, 第二个元素的索引序号为 2, ..., 依此类推.

```
>S[3], L[3], data_set[3];
```

$$2, 2, 3$$

```
>L[2..5];
```

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

使用 op 也可以从顺序表或集合中析取元素, 但不能用于序列.

```
>op(3, S);
```

```
Error, wrong number (or type) of parameters in function op
```

```
>op(3, data_set);
```

$$3$$

```
>op(2..5, L);
```

$$3, 2, 1, 2$$

使用 `convert` 可以计算集合或顺序表中元素的和或积 (注意倒引号).

```
>convert(data_set, `*`), convert(L, `+`);
```

$$24, 19$$

1.6.4 阵列 (array)

顺序表是“一维”结构, 顺序表的每个元素对应一个正整数作为它的索引指标. 阵列结构是顺序表结构的扩展, 阵列可以是“多维”的, 元素的索引指标不再限于正整数, 可以是任意整数. 使用阵列需要事先声明维数.

定义一个有 10 个元素的一维空阵列:

```
>a1 := array(1..10);
```

$$a1 := array(1..10, [])$$

为元素赋值:

```
>a1[1] := 2; a1[3] := 2^3; a1[8] := 2^8;
```

$$a1_1 := 2$$

$$a1_3 := 8$$

$$a1_8 := 256$$

使用 `print` 命令显示阵列:

```
>print(a1);
```

$$[2, a1_2, 8, a1_4, a1_5, a1_6, a1_7, 256, a1_9, a1_{10}]$$

定义一个 3×3 二维阵列并同时赋初值:

```
>a2 := array(1..3,1..3,[[1,2,3],[2,3,4],[4,5,6]]);
```

$$a2 := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{bmatrix}$$

```
>a2[3,2];
```

$$5$$

定义一个 $2 \times 2 \times 2$ 三维阵列并同时赋初值:

```
>a3 := array(1..2,1..2,1..2, [[[1,2],[3,4]], [[5,6],[7,8]]]);
```

$$a3 := ARRAY([1..2, 1..2, 1..2],$$
$$[(1,1,1) = 1,\ (1,1,2) = 2,\ (1,2,1) = 3,\ (1,2,2) = 4,\ (2,1,1) = 5,$$
$$(2,1,2) = 6,\ (2,2,1) = 7,\ (2,2,2) = 8])$$

使用负整数作索引也是允许的:

```
>a4 := array(-2..1,[1,2,3]);
```

$$a4 := ARRAY([-2..1],\ [(-2) = 1,\ (-1) = 2,\ (0) = 3,\ (1) = a4_1])$$

```
>a4[-2], a4[1];
```

$$1,\ a4_1$$

```
>whattype(a4);
```

$$symbol$$

```
>whattype(eval(a4));
```

$$array$$

1.6.5 表 (table)

阵列可以看成是顺序表在维数上的扩展, 而表则是顺序表在索引指标类型上的扩展, 表的索引指标不再限于整数, 可以是其它类型, 表的元素值可以是各种数据结构. 生成表的命令是`table`, 用顺序表指定表的初值, 顺序表中可用`key=value`(索引指标=元素值)的形式指定表中的元素.

空表:

```
>table();
```

$$TABLE([\,])$$

只用列表的形式给出表的元素, 则自动用正整数作为表元素的索引:

```
>T := table([22,42]);
```

$$T := TABLE([1 = 22, 2 = 42])$$

下例中的人名含有空格, 使用倒引号括起来作为一个符号 (或用双引号括起来作为一个字符串), Maple 根据内部设定的顺序按索引指标显示表的元素:

```
>jingli := table([jingli=`Zhang San`, fujingli=[Zhao,Qian,Sun,Li]]);
```

$$jingli := TABLE([fujingli = [Zhao, Qian, Sun, Li], jingli = Zhang\ San])$$

```
>jingli[jingli];
```

$$Zhang\ San$$

```
>jingli[fujingli][2];
```

$$Qian$$

很多具有对应关系的事实都可用表来表示, 例如已知的求导关系 $\sin'(x) = \cos(x)$, $\cos'(x) = -\sin(x)$, 可写成下表:

```
>DS := table([sin=cos,cos=-sin]):
>DS[cos](Pi/2);
```

$$-1$$

可以添加或删除表的元素:

```
>T[8] := 64; T[2] := 'T[2]';
```

$$T_8 := 64$$

$$T_2 := T_2$$

一般情况下, 使用名字实际是对名字完全求值 (参见 1.7.1), 但对于 `array`, `table` 和 `proc`, 直接使用名字并不完全求值, 可用 `eval` 或 `op` 强制完全求值:

```
>T;
```

$$T$$

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

$$TABLE([1 = 22, 8 = 64])$$

```
>op(T);
```

$$TABLE([1 = 22, 8 = 64])$$

可用 `print` 显示一个表:

```
>print(T);
```

$$TABLE([1 = 22, 8 = 64])$$

1.7 求值

Maple 中对变量的求值实质上是对变量名字所指向的内存的搜寻过程, 并不包含计算. 对表达式的求值是计算, 在 Maple 中称为化简, 化简应由用户指出要求, 但基本的化简是自动执行的, 并根据内部次序对结果式子进行排序, 所以对表达式的求值总是伴随着自动化简和排序.

1.7.1 完全求值

对变量的完全求值就是到当前为止与它有关的赋值语句全部执行后的值.

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

$$x := y$$

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

$$y := z$$

```
>z := 5;
```

$$z := 5$$

```
>x;
```

$$5$$

1.7.2 求值的层次

只用一个参数的 eval 的命令通常默认是完全求值, 但可以增加第二个参数指定求值到第几个层次.

```
>eval(x);
```

$$5$$

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

$$y$$

```
>eval(x,2);
```

$$z$$

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

$$5$$

```
>eval(x,100);
```

$$5$$

一些复杂结构如阵列, 表和过程以及过程内部的局部变量, 不使用默认的完全求值规则, 可使用 eval 命令强制完全求值.

1.7.3 阻止求值

用单引号将名字或表达式括起来, Maple 就不对它求值, 而仅仅是剥去这对单引号.

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

$$x := y$$

```
>y := 5;
```

$$y := 5$$

```
>x;
```

$$5$$

```
>whattype(x);
```

$$integer$$

```
>'''x''';
```

$$''x''$$

```
>%;
```

$$'x'$$

```
>%;
```

$$x$$

```
>%;
```

$$5$$

对一个变量名加引号, 其结果就是这个变量名 (Maple 中的符号), 不是变量值. 如果需要撤销对一个变量的原有赋值, 只需对它重新赋值. 若将它重新赋值为它自己的名字 (不是名字代表的值), 则相当于该变量未赋值.

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

$$x := x$$

```
>x;
```

$$x$$

```
>whattype(x);
```

$$symbol$$

2 绘图命令

2.1 快速绘图命令 smartplot

`smartplot(f)`

`smartplot3d(f)`

其中 f 是一个或多个函数或方程. 该绘图命令没有任何其他参数, 只能绘制直角坐标系中的图形. 对于函数绘图, 二维绘图默认定义域是 `-10..10`, 三维绘图默认定义域是 `-5..5,-5..5`.

该绘图命令使用简单, 根据绘图结果可作进一步处理. 下面是几个例子:

```
>smartplot( cos(x) , sin(x) );
>smartplot( cos(x) , sin(y) );
>smartplot( x^2 + y^2 = 7.5^2 );
>smartplot( abs(x-3)+abs(y-2)=6 , (x-3)^2+(y-2)^2=6^2 );
>smartplot3d( sin(x^2 + y^2) );
```

对于单个函数的绘图, 例如 `smartplot(sin(x))`, 可以简单地写为 `plot(sin)`, 即在二维绘图命令 `plot` 中, 只写函数名, 不写变量和其他选项参数.

2.2 图形的交互式设置

对于作好的图形, 有几种交互手段可以对它作进一步的编辑和修饰, 所谓交互式, 就是人机对话: 人们发出命令, 机器就作出反应, 给出命令效果.

2.2.1 图形窗口菜单

窗口菜单包含了所用软件与当前窗口有关的所有命令. 点击已有的图形, 窗口菜单栏就出现了与图形绘制有关的几个菜单 (图 4 和图 5).

与图 1 比较, 可以看到多出来的菜单, 菜单中的一些常用命令也出现在图形工具栏和快捷菜单中. 对菜单不再详细介绍, 因为人们更多使用的是图形工具栏中的按钮或快捷菜单中的命令.

2.2.2 图形工具

图 4 中常用工具栏的下方就是二维图形工具栏, 这个工具栏左边是坐标显示窗口, 点击图形中任意一点, 该点坐标就显示出来, 这组坐标值可用鼠标拖黑, 并被复制粘帖到其他文本中.

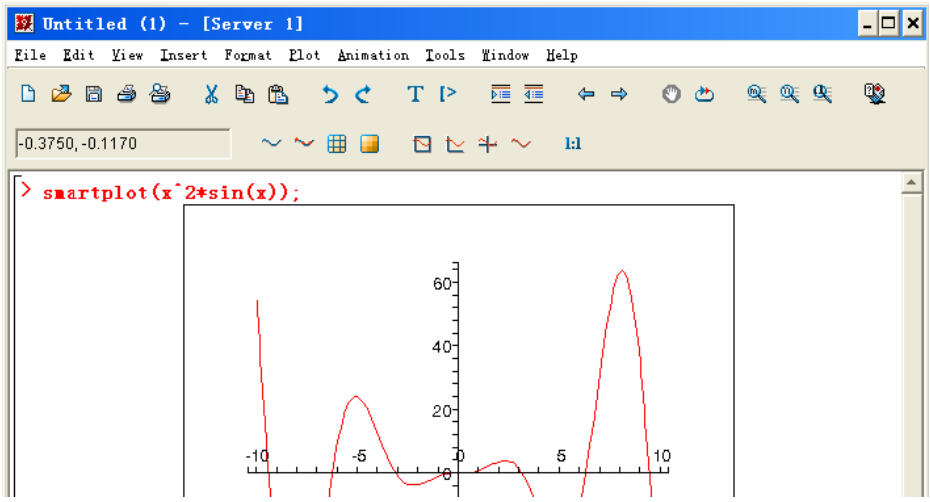


图 4: 图形窗口 (绘有二维图形)

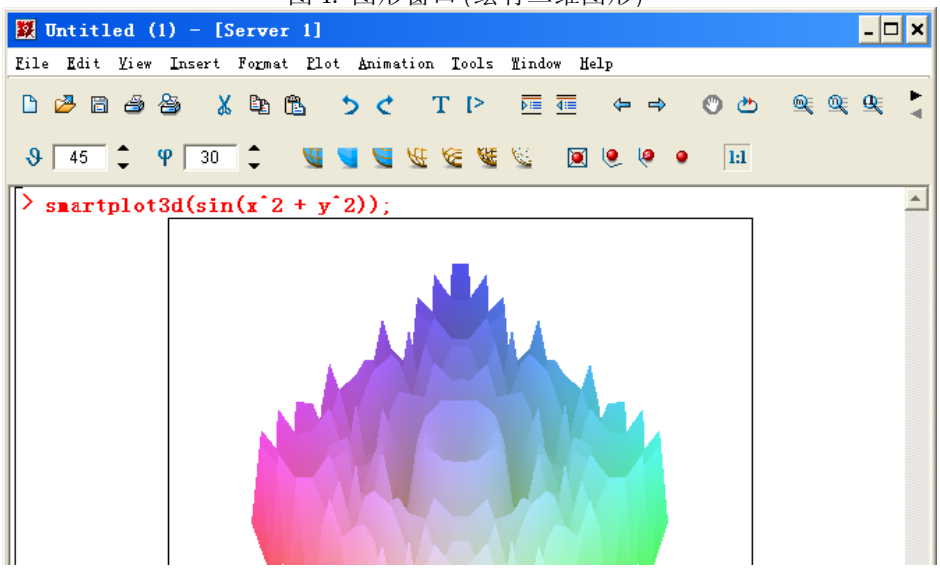


图 5: 图形窗口 (绘有三维图形)

其后 8 个按钮分成两组, 左边一组用于改变线型, 左起第一个按钮将数据点连成曲线或折线, 左起第二个按钮只画出数据点但不连线, 第三个和第四个按钮用于多边形区域图形, 很少用于曲线; 右边一组改变坐标轴的显示, 按钮的图案表示了坐标轴的显示面貌, 其中一个用于不显示坐标轴.

最右端的按钮是一个开关键, 用于控制横轴和纵轴上的单位长度是否相同, 当横竖两个方向上的单位长度等长, 即比例为 1:1 时. 图形虽然没有变形, 但可能并不好看.

图 5 中常用工具栏的下方是三维图形工具栏, 这个工具栏左边两框是视点角度显示窗口, 改变视点, 可以看到图形的转动. 其后是 7 个改变曲面风格的按钮和 4 个改变坐标轴显示的按钮, 最右边的按钮用于控制坐标轴单位是否等长. 图 5 中的图形已用过工具按钮改变了默认设置.

2.2.3 快捷菜单

右击图形区域, 就会弹出一个菜单 (参见右图), 称为图形的快捷菜单, 其中带有右向箭头的菜单命令表示还有下一级菜单. 快捷菜单也是用于对图形进行编辑、修饰和设置的. 快捷菜单与图形工具有些功能是重合的, 有些是相互补充的, 其中前 3 个命令用于剪切、复制和粘帖, 其他各项功能见表 6.

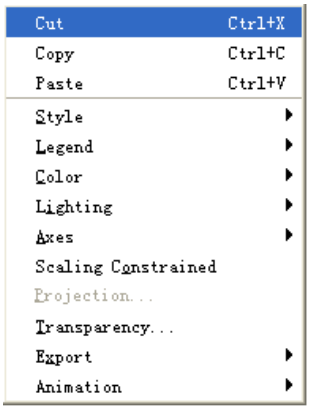


表 6: 图形的快捷菜单命令

命令	作用
Style	设置图形线型
Legend	控制是否显示图例
Color	设置图形颜色
Lighting	设置曲面照明方式
Axes	设置坐标轴
Scaling Constrained	控制横竖比例是否相同
Transparency	透明设置
Export	设置输出格式
Animation	动画设置
Projection	设置三维绘图的透视投影

2.3 二维绘图命令 plot

```
plot(f, h)
plot(f, h, v, opt1, opt2, ...)
```

参数的意义:

`f`	实函数
`h`	水平绘图范围
`v`	竖直绘图范围 (可不出现)
`opt1,opt2,...`	各种绘图选项

选项的格式是 *参数名* = *参数值*

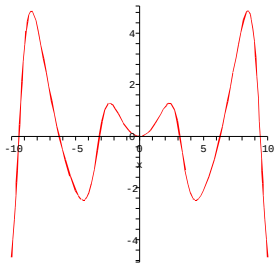
2.3.1 基本格式

绘制函数 $y = f(x)$ 的图象, 基本格式是给出函数 $f(x)$ 的表达式并指定自变量的范围: `plot(f(x), x = a .. b)`

```
>f := x->x*sin(x)/(2+cos(x));
>plot(f);
```

或

```
>plot(f(x), x=-10 ..  10);
```

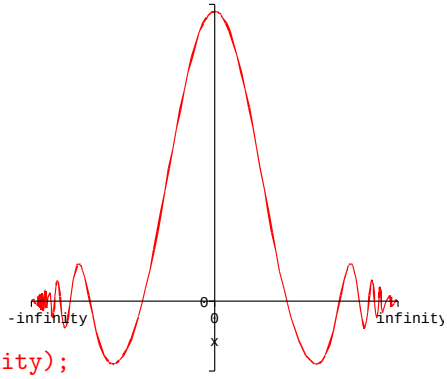


其中的单个函数也可换成函数集合或函数列表 (顺序表).

2.3.2 无穷区间

在有限的区域内绘制无穷区间上的图象, 这是 Maple 特有的功能. 用坐标轴上的有限点代表无穷远点, 实际上是做了变换, 因此图象会有变形, 这种图形主要用于定性研究, 对探讨无穷远处曲线的形态有所帮助. 这种图形不再显示坐标轴刻度, 如:

```
>plot(sin(x)/x, x=-infinity..infinity);
```



2.3.3 参数方程的图象

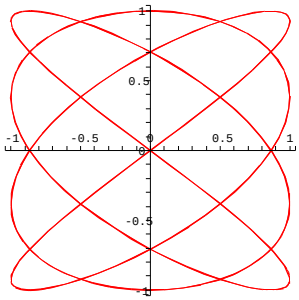
注意要把参数方程写成列表形式, 即放在方括号中:

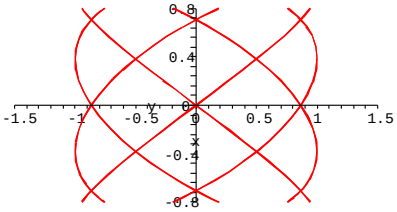
```
>plot([sin(4*t),sin(3*t),t=0..4*Pi]);
```

(输出见下面左图)

对于参数方程图像, 也可以指定坐标轴范围以控制显示范围, 当指定范围大于图形范围时, 图形四周会出现空白, 反之则有部分图形不能显示 (下面右图):

```
>plot([sin(4*t),sin(3*t),t=0..4*Pi], x=-1.5..1.5, y=-0.8..0.8);
```

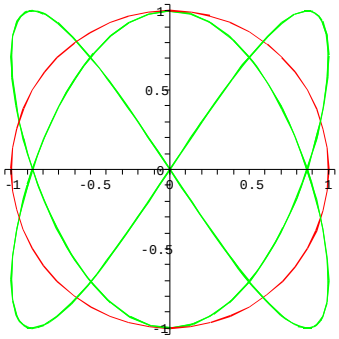




多个参数方程也可以画在同一图中.

```
>eq1:=[cos(t),sin(t),t=0..2*Pi]:
>eq2:=[sin(2*t),sin(3*t),t=0..4*Pi]:
>plot( [eq1, eq2] );
```

定义参数方程时用了方括号. 在同一个绘图命令中绘制多个图形, 也要用方括号构成列表, 或者用花括号构成集合.



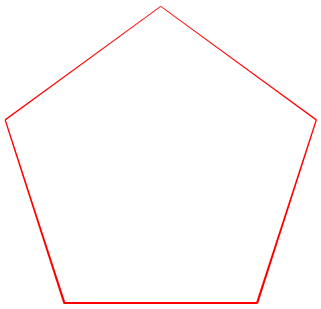
2.3.4 点列数据绘图

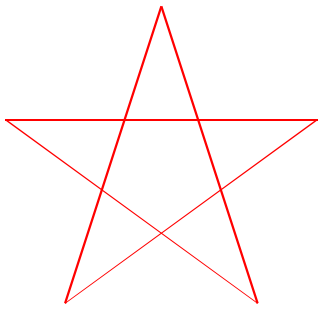
点列绘图命令格式是

plot($[[x_1, y_1], [x_2, y_2], \cdots, [x_n, y_n]]$)

外层方括号表示这是一个列表,不可省略. 绘图默认是用折线顺序连接各点, 若用上选项 `style=POINT`, 则只显示点而不连线.

```
>alpha := 2*Pi/5:
>beta := Pi/2:
>p:=array(1..5):
>for i to 5 do p[i]:=[cos(beta+(i-1)*alpha),sin(beta+(i-1)*alpha)] od:
>plot([ p[1],p[2],p[3],p[4],p[5],p[1] ],scaling=CONSTRAINED,axes=NONE);
>plot([ p[1],p[3],p[5],p[2],p[4],p[1] ],scaling=CONSTRAINED,axes=NONE);
```





2.3.5 极坐标

极坐标绘图命令是 `polarplot`, 该命令包含于 `plots` 程序包中, 加载整个包, 可看到包中含有的各个绘图命令:

```
>with(plots);
```

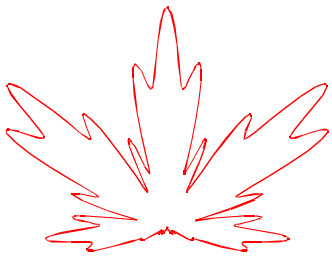
```
Warning, the name changecoords has been redefined
```

```
[animate, animate3d, animatecurve, arrow, changecoords, complexplot,
    complexplot3d, conformal, conformal3d, contourplot, contourplot3d, coordplot,
    coordplot3d, cylinderplot, densityplot, display, display3d, fieldplot, fieldplot3d,
    gradplot, gradplot3d, graphplot3d, implicitplot, implicitplot3d, inequal,
    interactive, listcontplot, listcontplot3d, listdensityplot, listplot, listplot3d,
    loglogplot, logplot, matrixplot, odeplot, pareto, plotcompare, pointplot,
    pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported,
    polyhedraplot, replot, rootlocus, semilogplot, setoptions, setoptions3d, spacecurve,
    sparsematrixplot, sphereplot, surfdata, textplot, textplot3d, tubeplot]
```

用极坐标方程绘制“枫叶”图形:

```
>S := t->100/(100+(t-Pi/2)^8):
>R := t->S(t)*(2-sin(7*t)-cos(30*t)/2):

>polarplot([R(t),t,t=-Pi/2..3/2*Pi],
  axes=none, scaling=constrained);
```



使用 `plot` 命令也可绘制极坐标图形, 只要加上选项 `coords=polar` 即可:

```
>plot([R(t),t,t=-Pi/2..3/2*Pi],
  axes=none, scaling=constrained, coords=polar);
```

2.3.6 隐函数绘图

plots 包中的 implicitplot 用于绘制隐函数图形.

```
implicitplot(expr, x=a..b, y=c..d, options)
implicitplot(f, a..b, c..d, options)
```

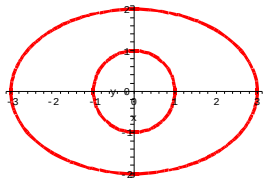
参数:

expr - 包含x和y的方程或方程组(方程列表或方程集合)

f - 由过程名或算子组成的方程

a,b,c,d - 实常数

options - 绘图可选项



若方程右端项是零, 可以省略等号和右端项.

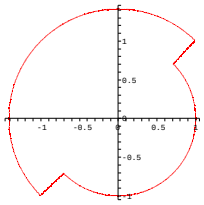
用命令的长格式作例 (结果见右图):

```
>plots[implicitplot]([x^2+y^2=1,x^2/9+y^2/4=1],
        x=-3..3,y=-2..2,scaling=constrained,thickness=3);
```

再看一例 (选项 grid 影响曲线光滑程度, 数值越大越光滑):

```
>p := proc(x,y) if x<y then x^2+y^2-2
      else x^2+y^2-1 end if end proc:

>plots[implicitplot](p, -1.5..1.5,
        -1.5..1.5, grid=[300,300]);
```



2.3.7 绘图选项

通过图形窗口菜单、图形工具栏和图形的快捷菜单可以对作好的图形进行各种修饰和设置, 这些效果也可通过在绘图命令中加上合适的选项而事先达到同样的目的. 如果将多个图象画在同一个图中, 使用选项参数可以对各个图象分别进行设置. 对于可用的绘图选项请看帮助内容, 参见图 6.

2.4 三维绘图命令 plot3D

```
plot3d(expr, x=a..b, y=c..d)
plot3d(f, a..b, c..d)
plot3d([exprf, exprg, exprh], s=a..b, t=c..d)
```

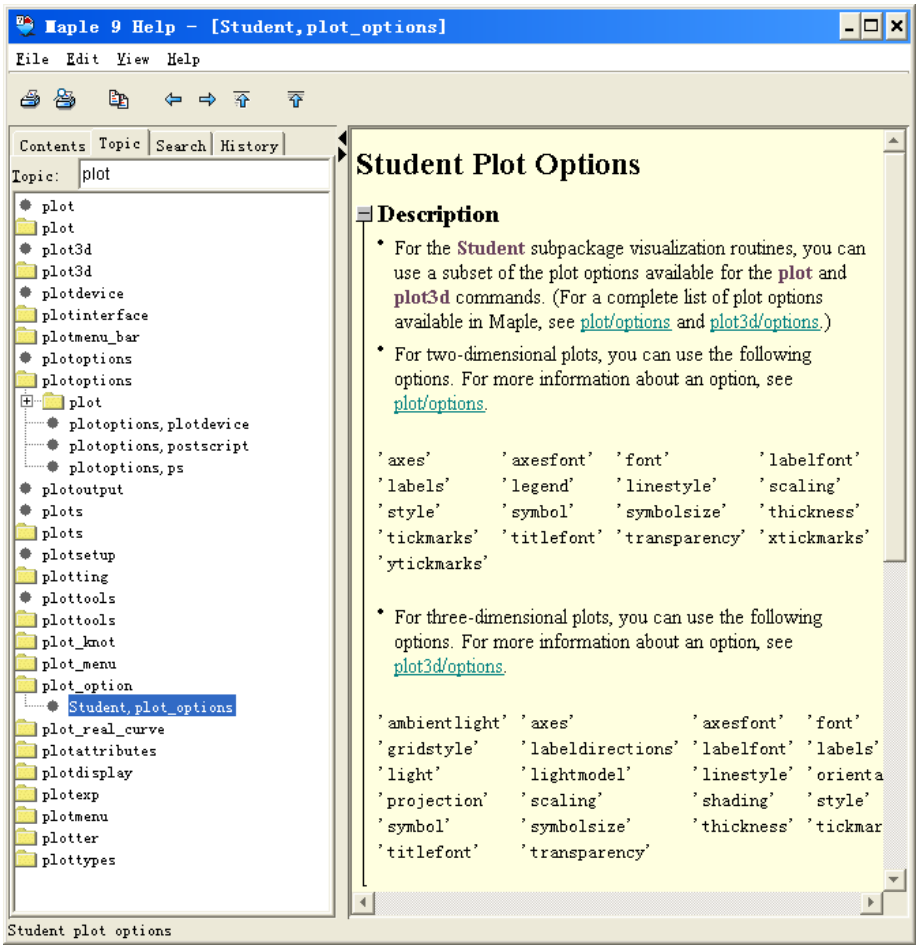


图 6: 绘图选项帮助信息

```
plot3d([f, g, h], a..b, c..d)
```

参数:

`expr`	-	关于 `x` 和 `y` 的表达式
`f, g, h`	-	过程或算子(函数名)
`exprf,exprg,exprh`	-	关于 `s` 和 `t` 的表达式
`a, b`	-	实常数、过程或关于 `y` 的表达式
`c, d`	-	实常数、过程或关于 `x` 的表达式
`x, y, s, t`	-	变量名

可用的三维绘图选项参见图 6.

使用表达式画曲面的例子 (图 7):

```
>plot3d(sin(x+y), x=-2..2, y=-2..2);
```

使用函数名 (算子) 画曲面的例子, 指定了网线数量 (图 8):

```
>plot3d(binomial, 0..5, 0..5, grid=[10,10]);
```

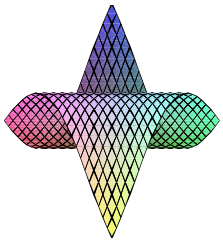


图 7:

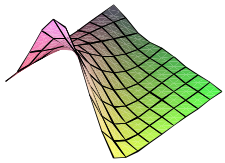


图 8:

同一表达式 `(1.3)^x*sin(y)` 在直角坐标系, 球面坐标系和柱面坐标系中的图形 (图 9):

```
>plot3d((1.3)^x*sin(y),x=-1..2*Pi,y=0..Pi,style=patch);
>plot3d((1.3)^x*sin(y),x=-1..2*Pi,y=0..Pi,coords=spherical,style=patch);
>plot3d((1.3)^x*sin(y),x=-1..2*Pi,y=0..Pi,coords=cylindrical,style=patch);
```

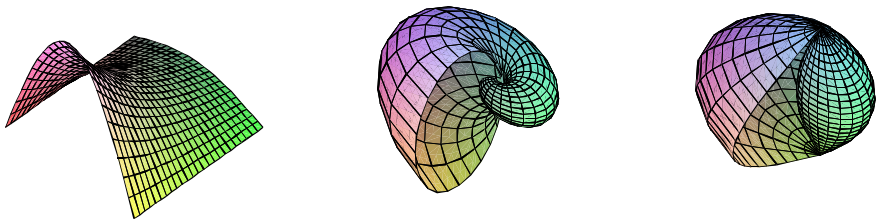
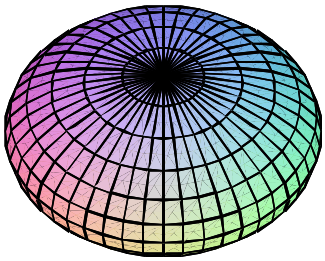


图 9: 表达式 $(1.3)^x * \sin(y)$ 在直角系, 球面系和柱面系中的图形

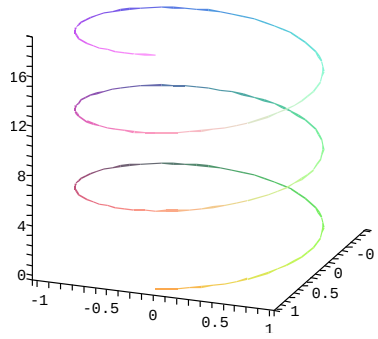
参数方程绘图命令, 三维与二维的格式有所不同, 在三维情况, 参数的变化范围要写在方程列表的外面:

```
>plot3d( [ sin(u)*cos(v),
           cos(u)*cos(v),
           .5*sin(v) ],
           u=0..Pi, v=0..2*Pi,
           scaling=constrained);
```



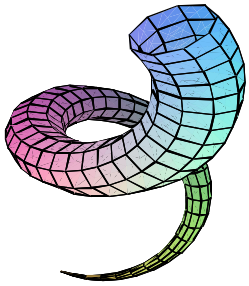
加载plots后, 可以绘制空间曲线.

```
>with(plots):
>spacecurve([cos(t),sin(t),t],
    t=0..6*Pi, numpoints=100,
    axes=framed,
    orientation=[20.7,70.5]);
```



命令tubeplot将空间曲线画成了空心管道, 选项radius指定管道半径(可以是变量).

```
>tubeplot([cos(t),sin(t),t],
            t=0..3*Pi,
            radius=t/15);
```



2.5 动画

加载 plots 程序包后, 有两方法可以产生动画, 一个是 animate 命令, 另一个是带有选项 insequence=true 的 display 命令.

2.5.1 使用 animate 命令产生动画

```
animate(plotcommand, plotargs, t=a..b, ...)
animate(plotcommand, plotargs, t=L, ...)
```

参数:

plotcommand	-	产生二维或三维图形的绘图命令(绘图函数名或过程名)
plotargs	-	绘图命令的各个参数
t	-	动画参数名(该参数的每个值对应于动画的一帧图形)
a,b	-	动画参数的变化范围(实常数)
L	-	动画参数取值列表(由实常数或复常数构成)

使用第一个调用语句时, 默认动画有25帧, 即把动画参数的变化范围 a..b 等分成25个值, 对应于每个分点值画一帧图形. 若不想使用默认帧数, 可以使用选项 frames=n 指定动画帧数为n(正整数). 使用第二个调用语句时, 按照L给出的数值制作动画的各帧图形.

如果使绘图参数中的图形范围端值依赖于动画参数, 就可制作出轨迹动画.

绘制动画后, 看到的是静止的第一帧图, 点击图形, 菜单和工具栏都有变化. 通过菜单栏的 Animation 菜单, 或右键快捷菜单的 Animation 命令的下级菜单, 或动画工具栏的按钮 (图 10), 就可播放动画了. 注意类似于播放器的快进与快退按钮, 其作用是增快或减退播放速度, FPS 后面的数字表示播放速度 — 每秒帧数, 数字越大, 速度越快, 数字小一点看得清楚.

图 10: 动画工具栏

在 Maple 9 中, 低版本的 animate, animate3d, animatecurve 命令仍然可用, 不再介绍.

下面是一些动画例子, 在 Maple 中运行给出的语句, 利用刚刚介绍的方法, 实际看看动画效果.

例 2.1: : 画一个圆周:

```
>with(plots):
>animate( plot, [[cos(theta),sin(theta),theta=0..t]],
            t=0..2*Pi, scaling=constrained, frames=50 );
```

例 2.2: : 参数方程定义的空间曲线 (3 圈多一点的弹簧) 的震动:

```
>opts := thickness=5, numpoints=100, color=black:
>animate( spacecurve, [[cos(theta),sin(theta),(2+sin(t))*theta],
                theta=0..3.1*(2*Pi), opts], t=0..2*Pi );
```

例 2.3: : 一个衰减的正弦曲线作为固定背景, 一个小球沿着这条曲线运动:

```
>sinewave := plot( sin(x)*exp(-x/5),x=0..20 ):
>animate( pointplot, [ [[t,sin(t)*exp(-t/5)]],
            symbol=circle, symbolsize=10],
            t=0..20, frames=60, background=sinewave );
```

2.5.2 使用 display 命令显示动画

一个 display 命令可以显示一个或一系列图形, 带选项 insequence=true 顺次显示这些图形就得到一个动画.

如果不写 `insequence` 选项, 则 `display` 将这一系列图形都叠加显示在同一个画面上, 成为一个静止画面而不是动画了.

参见第 37 页 §3.3.4 和第 51 页 §3.6 中的动画.

3 微积分

3.1 定义函数

3.1.1 用映射定义函数

用户定义一个函数的标准方法是使用“映射”运算符 (–>, 输入时键入一个减号和一个大于号), 例如定义一个函数, 用于将华氏温度转换成摄氏温度, 函数名为 f2c:

```
>f2c := f -> evalf[4]((f-32)*5/9);
```

$$f2c := f \rightarrow evalf_4\left(\frac{5}{9}f - \frac{160}{9}\right)$$

```
>f2c(100);
```

$$37.78$$

再看一些例子.

```
>g := (x,y) -> sin(x)*cos(y) + x*y;
```

$$g := (x, y) \rightarrow sin(x)cos(y) + xy$$

```
>g(Pi/2, Pi);
```

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

```
>h := x -> (2*x, x^3);
```

$$h := x \rightarrow 2x, x^3$$

```
>h(3);
```

$$6,\ 27$$

3.1.2 由表达式得到函数

定义函数的另一种方法是用 unapply 将含有未知量的表达式变成函数关系:

```
>c := (f-32)*5/9;
```

$$c := \frac{5}{9}f - \frac{160}{9}$$

```
>fc1 := unapply(c,f);
```

$$fc1 := f \rightarrow \frac{5}{9}f - \frac{160}{9}$$

```
>fc1(100.0);
```

$$37.77777778$$

```
>fc1(x);
```

$$\frac{5}{9}x - \frac{160}{9}$$

加选项, 指定自变量类型是数值型:

```
>fc2 := unapply(c, f::numeric);
```

$$fc2 := f :: numeric \rightarrow \frac{5}{9}f - \frac{160}{9}$$

```
>fc2(x);
```

Error, invalid input: fc2 expects its 1st argument, f, to be of type numeric, but received x

`codegen[makeproc]` 和 `student[makeproc]` 也可将表达式转换成函数或过程, 参见 Maple 的帮助信息.

3.1.3 定义分段函数

`piecewise(条件1,式1, 条件2,式2, ..., 条件n,式n, 其它情况表达式)` 用于定义分段函数, 这相当于编程语言中的 `if-elseif` 语句: 如果条件1成立, 则为式1, 否则, 如果条件2成立, 则为式2, 以此类推; 若所有条件都不成立, 则为其他情况表达式, 其缺省值是 0.

例子:

```
>piecewise(x>0,x);
```

$$\begin{cases} x & 0 < x \\ 0 & otherwise \end{cases}$$

```
>piecewise(x*x>4 and x<8,f(x));
```

$$\begin{cases} f(x) & -x^2 < 4 \textbf{ and } x < 8 \\ 0 & otherwise \end{cases}$$

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

$$\begin{cases} f(x) & x < -2 \\ 0 & x <= 2 \\ f(x) & x < 8 \\ 0 & 8 <= x \end{cases}$$

3.1.4 用过程定义函数

参见第 61 页 §5.1.

3.2 极限、连续、间断点

3.2.1 极限

```
limit(f, x=a, dir)
Limit(f, x=a, dir)
```

参数:

```
f   - 表达式
x   - 变量名
a   - 极限点，可以是有限值或无穷远(infinity, -infinity)
dir - 方向(可选)，其值为下列值之一: left, right, real, complex
```

注意大小写的不同, 一般而言, 同样字母组成的命令, 首字母大写时是对应的全小写命令的"惰性"版本, 即不求值. 对惰性版本的结果施行 value 命令就可得到值了, 结果当然与小写字母命令的结果是相同的.

```
>Limit( (1+tan(x))^cot(x), x=0 );
```

$$\lim_{x \to 0} (1 + \tan(x))^{\cot(x)}$$

```
>limit( (1+tan(x))^cot(x), x=0 );
```

$$\mathbf{e}$$

还要注意, infinity 相当于高等数学中的 $+\infty$.

```
>limit(exp(x), x=infinity);
```

$$\infty$$

```
>limit(exp(x), x=-infinity);
```

$$0$$

级数极限的例子: $\lim_{n \to \infty} \sum_{k=1}^{n} \left(\sqrt[3]{1 + \frac{k^2}{n^3}} - 1 \right)$,

```
>Limit(Sum(root[3](1+k^2/n^3) - 1, k=1..n), n=infinity);
```

$$\lim_{n \to \infty} \left(\sum_{k=1}^{n} \left(\left(1 + \frac{k^2}{n^3} \right)^{(1/3)} - 1 \right) \right)$$

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

$$\frac{1}{9}$$

```
>limit(sum(root[3](1+k^2/n^3) - 1, k=1..n), n=infinity);
```

$$\frac{1}{9}$$

分段函数也可求极限:

```
>p := piecewise(x<0, x, exp(x));
```

$$p := \begin{cases} x, & x < 0 \\ \mathbf{e}^{x}, & otherwise \end{cases}$$

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

$$undefined$$

```
>limit(p,x=0,left);
```

$$0$$

```
>limit(p,x=0,right);
```

$$1$$

此例在 $x = 0$ 点不能确定极限, 但可求出左、右极限分别为 0 和 1. 所以在 $x = 0$ 点极限不存在.

Maple 还可以求多重极限:

```
>limit(x+1/y, x=0,y=infinity);
```

$$0$$

3.2.2 连续

使用命令 `iscont` 可以判断函数在指定区间上是否连续, 默认区间是开区间, 即不检查端点的连续性, 若指定闭区间, 需加上选项 `'closed'`. 以上面定义的分段函数 `p` 为例:

```
>iscont(p,x=-infinity..infinity);
```

$$false$$

```
>iscont(p,x=0..infinity);
```

$$true$$

```
>iscont(p,x=-infinity..0);
```

$$true$$

```
>iscont(p,x=0..1,'closed');
```

$$false$$

3.2.3 间断点

命令 `discont` 用于求出函数的间断点集合:

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

$$\{0\}$$

```
>discont(tan(x),x);
```

$$\left\{\pi\_Z1\sim+\frac{\pi}{2}\right\}$$

其中 $\_Z1\sim$ 代表整数, 这就是集合 $\left\{n\pi+\frac{\pi}{2},\ n\in\mathbf{Z}\right\}$.

3.3 导数、偏导数

计算导数和偏倒数的命令有 3 个: `diff`, `D`, `implicitdiff`.

`diff` 用于对表达式求导, 结果也是表达式; `D` 用于函数名 (算子) 求导, 结果仍是算子; `implicitdiff` 用于隐函数求导.

3.3.1 求导命令 diff

```
>Diff(tan(x),x);
```

$$\frac{d}{dx}\tan(x)$$

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

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

```
>Diff(tan(x),x) = diff(tan(x),x);
```

$$\frac{d}{dx}\tan(x)=1+\tan(x)^2$$

当求导结果不能用具体的表达式给出时, 就直接给出导数的数学表示:

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

$$\frac{d}{dx}f(x)$$

```
>diff(f(x,y),x,y);
```

$$\frac{\partial^2}{\partial x\partial y}f(x,y)$$

高阶导数如 `diff(f(x),x,x,x)` 可以写成 `diff(f(x),x$3)`, 高阶偏导数如 `diff(f(x,y),x,x,x,y,y)` 可以写成 `diff(f(x),x$3,y$2)`. 即对变量 x 进行 k 次求导时, 可缩写成 `x$k` 的形式.

求导变量可以放在方括号中, 看起来更清楚. 当不写求导变量时, 就必须用一个空的方括号, 此时返回表达式本身.

```
>diff( f(x,y,z), [x,z$3] );
```

$$\frac{\partial^4}{\partial x\partial z^3}f(x,y,z)$$

```
>diff( f(x,y,z), [] );
```

$$f(x,y,z)$$

3.3.2 求导命令 D

D 只用于对函数名 (算子) 求导数, 结果仍是函数名 (算子).

```
>D(tan);
```

$$1+\tan^2$$

```
>D(ln);
```

$$x->\frac{1}{x}$$

```
>D(ln)(x); D(ln)(2)      #导函数在一点处的值
```

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

对于多元函数f, 用 D[i](f) 指定对第i个变量求导, 用 D[i,j](f) 指定对第i, j变量求导. 常用的求导法则都成立, 如 D(f+g) = D(f) + D(g), D(f*g) = D(g*f) = D(f)*g + D(g)*f, D(f@g) = D(f)@g * D(g), 等等. 此外, 总是假定混合偏导数可交换: D[i,j](f) = D[j,i](f). 注意, 函数 f 和 g 的复合函数用 f@g 而不是用 f(g) 表示.

例 3.1: : 验证函数 $z=\ln\sqrt{x^2+y^2}$ 满足拉普拉斯方程 $\dfrac{\partial^2 z}{\partial x^2}+\dfrac{\partial^2 z}{\partial y^2}=0$.

```
>z := (x,y)->ln(sqrt(x^2+y^2));
```

$$z:=(x,y)->\ln\left(\sqrt{x^2+y^2}\right)$$

```
>D[1$2](z):  # 为页面紧凑, 不再显示中间结果
>D[2$2](z):
>(%% + %)(x,y):
>simplify(%);
```

$$0$$

高阶导函数:

```
>D(D(D(f)));
```

$$D^{(3)}(f)$$

```
>(D@@3)(f);
```

$$D^{(3)}(f)$$

函数一词有时指函数关系 f, 有时指函数表达式 $f(x)$, 当需要明确区分时, 可将前者称为函数名或算子, 后者称为表达式.

函数名, 函数表达式, 求导命令D和diff之间的关系及转换见图 11.

此外使用 convert 命令也可在 D 和 diff 两种形式之间进行转换.

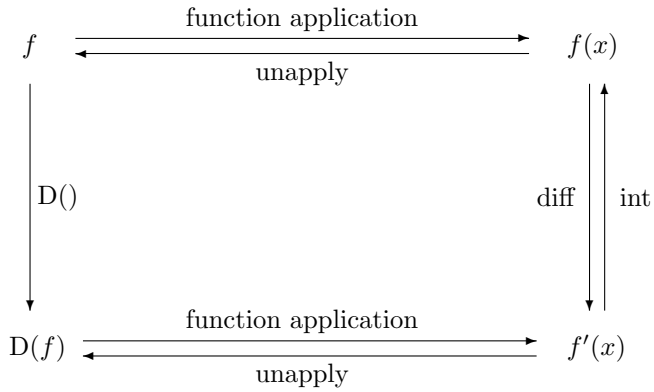


图 11: 函数与求导命令相互之间的转换关系

```
>diff(f(x,y,z),[x,x,y,z]);
```

$$\frac{\partial^4}{\partial x^2 \partial y \partial z} f(x,y,z)$$

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

$$\mathrm{D}_{1,1,2,3}(f)(x,y,z)$$

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

$$\frac{\partial^4}{\partial x^2 \partial y \partial z} f(x,y,z)$$

```
>lprint(%);
```

$$\mathrm{diff}(\mathrm{diff}(\mathrm{diff}(\mathrm{diff}(f(x,y,z),x),x),y),z)$$

命令 `lprint` 用于显示表达式的内部结构.

3.3.3 隐函数求导命令 `implicitdiff`

```
implicitdiff(f, y, x)
implicitdiff(f, y, x1,...,xk)
implicitdiff({f1,...,fm}, {y1,...,yn}, u, x)
implicitdiff({f1,...,fm}, {y1,...,yn}, u, x1,...,xk)
implicitdiff({f1,...,fm}, {y1,...,yn}, {u1,...,ur}, x)
implicitdiff({f1,...,fm}, {y1,...,yn}, {u1,...,ur}, x1,...,xk)
```

参数:

- `f,f1,...,fm` - 确定隐函数的代数表达式或方程
- `y,y1,...,yn` - 因变量或函数(表明某个因变量依赖哪些自变量)
- `u,u1,...,ur` - 因变量(指明要求导的对象)

`x,x1,...,xk` - 求导变量名

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

$$f := x^2 + y^3 = 1$$

```
>implicitdiff(f,y,x);
```

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

```
>implicitdiff(f,x,y);
```

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

```
>implicitdiff(f,y,z);
```

$$0$$

```
>implicitdiff(f,y(x),x);
```

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

```
>f := R=P*V/T;
```

$$f := R = \frac{PV}{T}$$

```
>implicitdiff(f, P, T);
```

$$\frac{P}{T}$$

```
>implicitdiff(f, V, P);
```

$$-\frac{V}{P}$$

```
>implicitdiff(f, T, V);
```

$$\frac{T}{V}$$

```
>% * %% * %%%;
```

$$-1$$

3.3.4 导数的几何意义

从教材 §3.1 可知, 函数 $f(x)$ 在点 x_0 的导数 $f'(x)$ 是曲线 $y = f(x)$ 在点 $p_0(x_0, f(x_0))$ 处切线的斜率, 而切线是割线的极限位置. 我们可用 `student` 程序包中的一些命令来演示从这种几何问题引出导数概念的过程.

由于要用到 `student` 程序包, 所以要加载它:

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

[D, *Diff, Doubleint, Int, Limit, Lineint, Product, Sum, Tripleint, changevar, completesquare, distance, equate, integrand, intercept, intparts, leftbox, leftsum,*

makeproc, middlebox, middlesum, midpoint, powsubs, rightbox, rightsum, showtangent, simpson, slope, summand, trapezoid]

除了单个字母命令 D 之外, 其他以大写字母开头的命令是对应的全小写命令 (如果有的话) 的"惰性"版本, 即不进行求值.

为了画出曲线和割线, 需要给出具体的函数 f 和 x_0 的值, 例如

```
>f := x->x^2/4;
```

$$f := x -> \frac{1}{4}x^2$$

```
>x0 := 1;
```

$$x0 := 1$$

为作过 p_0 的割线, 在曲线上取定点 p_0 和 动点 $p_1(x_0+h, f(x_0+h))$:

```
>p0 := [x0,f(x0)];
```

$$p0 := \left[1, \frac{1}{4}\right]$$

```
>p1 := [x0+h,f(x0+h)];
```

$$p1 := \left[1+h, \frac{1}{4}(1+h)^2\right]$$

student 包中的 slope 命令可以求出过 p_0, p_1 点的割线的斜率:

```
>m := slope(p0,p1);
```

$$m := -\frac{\frac{1}{4} - \frac{1}{4}(1-h)^2}{h}$$

割线斜率的极限是

```
>limit(m,h=0);
```

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

这当然等于 $f(x)$ 在 x_0 处的导数 $f'(x_0)$:

```
>f1 := D(f);
```

$$f1 := x -> \frac{1}{2}x$$

```
>f1(x0);
```

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

下面用几何图形来演示割线趋向于切线的过程.

割线的点斜式方程是

```
>y - p0[2] = m*(x-p0[1]);
```

$$y - \frac{1}{4} = -\frac{\left(\frac{1}{4} - \frac{1}{4}(1-h)^2\right)(x-1)}{h}$$

使用命令 `isolate` 把方程中的 y 离析出来 (结果变成了斜截式):

```
>isolate(%,y);
```

$$y = -\frac{\left(\frac{1}{4} - \frac{1}{4}(1-h)^2\right)(x-1)}{h} + \frac{1}{4}$$

由这个方程很容易得到关于 x 的函数, 即把方程右端项 (rhs) 转换成函数, 函数名可命名为 `secant`:

```
>secant := unapply(rhs(%),x);
```

$$secant := x -> -\frac{\left(\frac{1}{4} - \frac{1}{4}(1-h)^2\right)(x-1)}{h} + \frac{1}{4}$$

现在就可以对给定的 h 作出割线的图, 当 $h \to 0$ 时, 割线应当趋向于切线. 为了能清楚看出割线逼近切线的过程, 我们让 h 从一列趋向于零的等差数列中取值, 例如数列为:

```
>h_list := [seq(3-0.1*i,i=0..29)];
```

$$h\_list := [3., 2.9, 2.8, 2.7, 2.6, 2.5, 2.4, 2.3, 2.2, 2.1, 2.0, 1.9, 1.8, 1.7, 1.6, 1.5, 1.4, 1.3, 1.2, 1.1, 1.0, 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]$$

对应于每一个 h 值, 都可作出函数和割线的图形. 首先作一列作图命令, 用割线的斜率作为图形标题:

```
>S := seq(plot([f(x),secant(x)],x=0..4,view=[0..4,0..4],
        title=convert(evalf[4](m),string)),h=h_list):
```

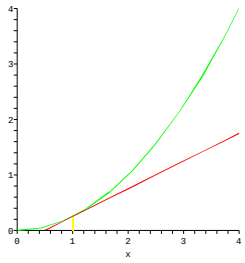
然后用 `plots` 程序包中的 `display` 命令带选项 `insequence=true` 顺次显示这些已作好的图形就得到一个动画:

```
>display(S,insequence=true,view=[0..4,0..4]);
```

在纸面上无法演示动画, 我们从中取出了 6 帧画面排成 2 行 3 列的阵列显示成连环画, 以便看出割线的变化过程 (图 12):

```
>aa := array(1..2, 1..3):
>for i to 2 do  for j to 3 do aa[i,j] := S[(i-1)*15+j*5] od:  od:
>display(aa);
```

从动画或连环画上可以看出割线是趋向于切线的.



如果需要直接显示出 $f(x)$ 在 $x = x_0$ 处的切线, 可以使用 `showtangent` 命令:

```
>showtangent(f(x),x=x0,view=[0..4,0..4]);
```

3.3.5 曲线形态与切线的关系

从教材 §4.4 和 §4.5 可知, 根据切线的斜率, 可以判断曲线的升降和凹凸, 当

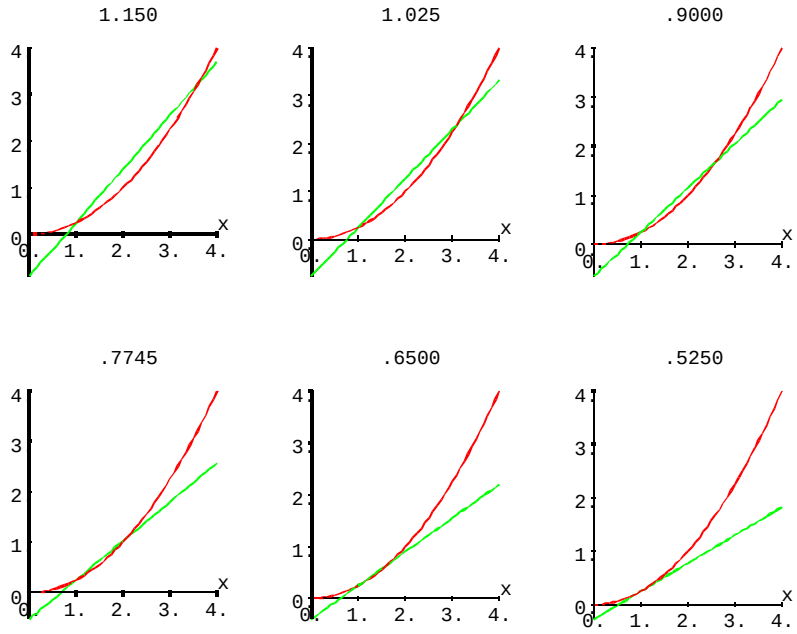


图 12: 纸面上表示动画的“连环画”

切线斜率为零时, 可找到曲线的驻点. 下面用一个动画演示上述关系:

```
>restart:  with(student):  with(plots):

>f := x->3*(x+1)^(1/10)+x*sin(x)/3:
>animate(showtangent, [f(x),x0,x=0..2*Pi], x0=0..2*Pi,
            view=[0..2*Pi,1..5], frames=16);
```

使用 `display(animate(...))` 可以自动将动画显示成阵列形式的连环画 (图 13), 用起来很方便:

```
>display(animate(showtangent, [f(x),x0,x=0..2*Pi], x0=0..2*Pi,
            view=[0..2*Pi,1..5], frames=16));
```

但各帧的显示范围和坐标框架会有变化, 与动画的实际效果差别较大, 不如上一节显示连环画的方法好:

3.4 级数展开

3.4.1 一般的级数展开命令

```
series(expr, eqn)
series(expr, eqn, n)
```

参数:

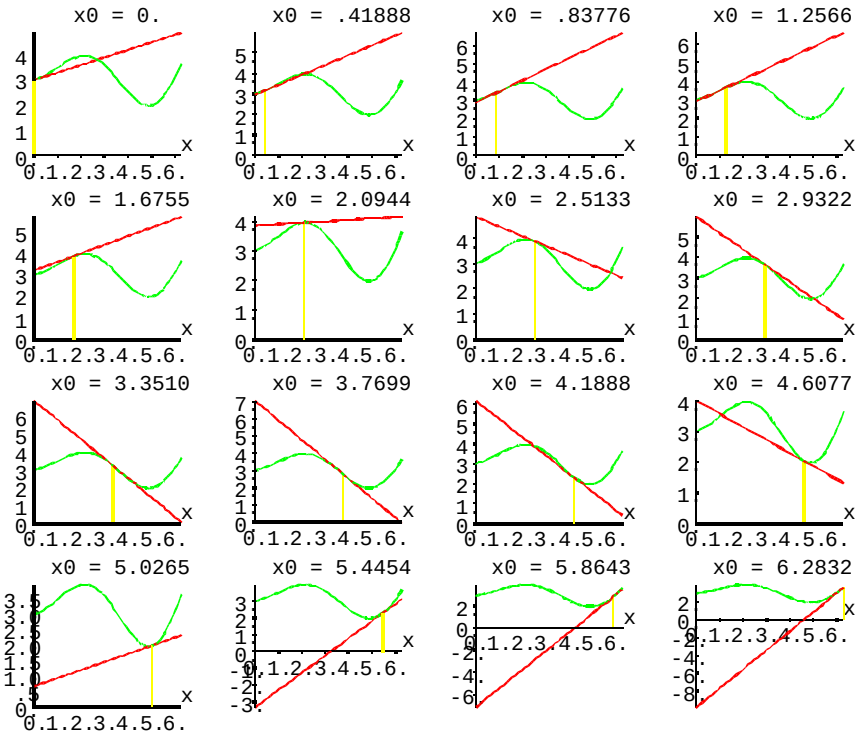


图 13: 曲线形态与切线的关系

`expr` - 表达式

`eqn` - 等式(如`x=a`)或名字(如`x`,等价于`x=0`)

`n` - (可选项) 非负整数

`series` 将表达式在指定点展开成级数, 当指定第 3 个参数 `n` 时, 展开时从第 `n` 阶截断, 不指定第 3 个参数时, 截断阶数由环境变量 `Order` 决定, 其默认值是 6. 注意, 中间计算时是从第 `n` 阶截断的, 但合并到最终结果, 有可能小于这个阶数.

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

$$x + x^2 + 2x^3 + 3x^4 + 5x^5 + O(x^6)$$

```
>series(x/(1-x-x^2), x=0, 6);
```

$$x + x^2 + 2x^3 + 3x^4 + 5x^5 + O(x^6)$$

```
>series(x+1/x, x=1, 3 );
```

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

```
>series(x^3/(x^4+4*x-5),x=infinity);
```

$$\frac{1}{x}-\frac{4}{x^4}+\frac{5}{x^5}+O(\frac{1}{x^7})$$

测试展开结果的类型:

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

$$series$$

x趋于无穷时的渐近展开命令是 `asympt(expr,x)` 或 `asympt(expr,x,n)`, 所以上式可写成:

```
>asympt(x^3/(x^4+4*x-5),x);
```

$$\frac{1}{x}-\frac{4}{x^4}+\frac{5}{x^5}+O(\frac{1}{x^7})$$

实际上, `asympt(expr,x,n)` 是利用 `series` 定义的:

```
subs( x=1/x, series( subs(x=1/x, expr), x=0, n ) );
```

在特殊情况下, `series` 展开的结果不一定是 Maple 的级数类型:

```
>s := series(sqrt(sin(x)), x=0, 4);
```

$$s:=\sqrt{x}-\frac{1}{12}x^{(5/2)}+O(x^{(9/2)})$$

```
>type(s, series);
```

$$false$$

```
>whattype(s);
```

$$+$$

3.4.2 数学函数的级数展开

当对一个已知的数学函数作级数展开时, 将 `series` 首字母大写即可, 但这个首字母大写命令并不是全小写命令的"惰性"版本, 两者有很大区别: `Series` 只能展开已知的数学函数, 但结果给出更多信息, 此外要加载数学函数程序包 `MathematicalFunctions`.

```
>with(MathematicalFunctions, Series);
```

$$[Series]$$

比较下列结果:

```
>series( arcsin(z), z );
```

$$z+\frac{1}{6}z^3+\frac{3}{40}z^5+O(z^6)$$

```
>Series( arcsin(z), z );
```

$$z+\frac{1}{6}z^3+\frac{3}{40}z^5+O(z^7),\ And(|z|^2<1)$$

3.4.3 首项

只求级数展开的首项 (leadterm), 命令是:

```
series('leadterm'(expr), eqn)
series('leadterm'(expr), eqn, n)
```

参数意义同 series 的参数.

```
>series(leadterm(x^3/(x^4+4*x-5)),x=infinity);
```

$$\frac{1}{x}$$

当首项的阶大于指定 (或默认) 的截断阶数时, 得不到真正的首项:

```
>series(leadterm(cos(x^50)-1),x);
```

$$O(x^{100})$$

只要将截断阶数指定得充分大即可:

```
>series(leadterm(cos(x^50)-1), x, 10^9);
```

$$-\frac{1}{2}x^{100}$$

3.4.4 Taylor 展开

将一般的级数展开命令 series 换成 taylor, 就是 Taylor 展开, 但若展开结果不是 Taylor 级数, 则返回一个出错信息.

```
>taylor( 1/x, x=1, 3 );
```

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

```
>taylor( 1/x + y + x^3, x=0 );
Error, does not have a taylor expansion, try series()

>series( 1/x + y + x^3, x=0 );
```

$$x^{(-1)}+y+x^3$$

3.4.5 转换级数成多项式

级数是一种特殊的数据结构, 由于余项的存在, 有些命令不能直接用于级数, 例如不能直接画级数的图. 把级数转换成多项式就可作各种处理了. 转换命令是 convert(s,polynom), 其中 s 是一个级数.

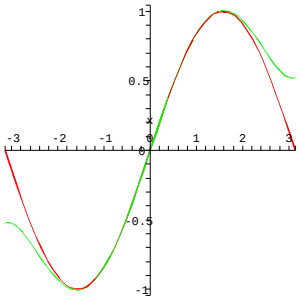
```
>s := series(sin(x),x);
```

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

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

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

```
>plot({sin(x),p}, x = -Pi ..  Pi);
```



可以看出, 在区间 $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ 上, 用一个 5 阶多项式就可以很好地逼近 $\sin x$ 了. 下面计算最大误差值和达到最大误差值的点:

```
>numapprox[infnorm](sin(x)-p, x=-Pi/2..Pi/2, 'xmax');
```

$$0.004524855539$$

```
>xmax;
```

$$-1.570796327$$

3.4.6 傅里叶级数

教材 §11.6 中讨论了周期为 $2l$ 的函数 $f(x)$ 的傅里叶级数

$$f(x) \sim \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{l} + b_n \sin\frac{n\pi x}{l}\right),$$

其中系数

$$a_n = \frac{1}{l}\int_{-l}^{l} f(x)\cos\frac{n\pi x}{l}\,\mathrm{d}x \quad (n = 0, 1, 2, \cdots),$$

$$b_n = \frac{1}{l}\int_{-l}^{l} f(x)\sin\frac{n\pi x}{l}\,\mathrm{d}x \quad (n = 1, 2, \cdots).$$

对于定义于区间 $[a, b]$ 上的函数 $f(x)$, 如果满足收敛定理的条件, 只需在 $(-\infty, +\infty)$ 上拓展成周期为 $2l = b - a$ 的周期函数, 利用上述结论也可以在 $[a, b]$ 上展成傅里叶级数, 此时 $l = (b - a)/2$,

$$f(x) \sim \frac{c_0}{2} + \sum_{n=1}^{\infty}\left(c_n \cos\frac{2n\pi x}{b-a} + s_n \sin\frac{2n\pi x}{b-a}\right),$$

其中系数

$$c_n = \frac{2}{b-a}\int_{a}^{b} f(x)\cos\frac{2n\pi x}{b-a}\,\mathrm{d}x \quad (n = 0, 1, 2, \cdots),$$

$$s_n = \frac{2}{b-a}\int_{a}^{b} f(x)\sin\frac{2n\pi x}{b-a}\,\mathrm{d}x \quad (n = 1, 2, \cdots).$$

为了观察在区间 $[a, b]$ 上傅里叶级数前 n 项逼近函数 $f(x)$ 的的效果, 需要同时画出函数和不同项数的傅里叶级数的图形, 可以编写如下一个过程 (编程知识参见第 61 页 §5), 对于每个正整数 $i \leqslant n$, 求出傅里叶级数前 i 项, 画出对应的图形和 $f(x)$ 的图形, 即可动态地看到逼近过程.

```
FourierApprox := proc( f, xrange::name=range, n::posint )
   local x, a, b, l1, i, p, q, fourier;
   a := lhs( rhs(xrange) );
   b := rhs( rhs(xrange) );
   l1 := 2/(b-a);
   x := l1 * Pi * lhs(xrange);
   fourier := l1 * evalf( Int(f, xrange) ) / 2;
   for i to n do
      fourier := fourier
                 + l1 * eval( Int(f*cos(i*x), xrange) ) * cos(i*x)
                 + l1 * eval( Int(f*sin(i*x), xrange) ) * sin(i*x);
      q[i] := plot( fourier, xrange, color=blue,
                    title=convert(i,string), args[4..nargs] );
   od;
   p := plot( f, xrange, color=red, args[4..nargs] );
   q := plots[display]( [ seq( q[i], i=1..n ) ], insequence=true);
   plots[display]( [q, p] );
end proc;
```

教材 §11.6 例1: $f(x)$ 是周期为 2π 的函数, 其在 $[-\pi, \pi]$ 上的表达式为

$$f(x) = \begin{cases} -\dfrac{\pi}{4}, & -\pi \leqslant x \leqslant 0, \\ \dfrac{\pi}{4}, & 0 < x < \pi. \end{cases}$$

我们来看看用它的傅里叶级数逼近的效果, 由于计算较慢, 取 $n = 6$. 如果直接使用教材上推导出的结果而不是用这里的一般方法, 可以节省大量时间.

```
>f := piecewise(-Pi<=x and x<=0,-Pi/4, 0<x and x<Pi, Pi/4):
>FourierApprox(f, x=-Pi..Pi, 6, discont=true);
```

产生的动画已存为文件 fouries.gif, 其中两幅静态画面参见图 14.

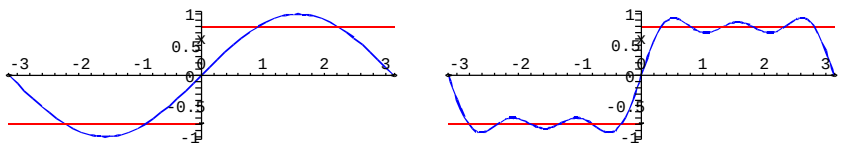


图 14: 傅里叶逼近

3.4.7 多变量 Taylor 展开

多变量 Taylor 展开的命令是`mtaylor`, 结果是一个正常多项式, 不带余项. 下面以例说明用法.

将函数 $f(x,y)$ 在 $(x,y)=(0,0)$ 处展开, 从 3 阶处截断:

```
>mtaylor(f(x,y), [x,y], 3);
```

$$f(0,0)+D_1(f)(0,0)x+D_2(f)(0,0)y+\frac{1}{2}D_{1,1}(f)(0,0)x^2+xD_{1,2}(f)(0,0)y+\frac{1}{2}D_{2,2}(f)(0,0)y^2$$

将 $\sin(x^2+y^2)$ 在 $(x,y)=(1,0)$ 处展开, 从 3 阶处截断:

```
>mtaylor(sin(x^2+y^2), [x=1,y], 3);
```

$$sin(1)+2cos(1)(x-1)+(-2sin(1)+cos(1))(x-1)^2+cos(1)y^2$$

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

$$false$$

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

$$true$$

3.5 积分

3.5.1 不定积分、定积分

```
int(expr, x)
Int(expr, x)
int(expr, x=a..b)
Int(expr, x=a..b)
int(expr, x=a..b, opt)
Int(expr, x=a..b, opt)
```

参数:

```
expr - 被积分的表达式
x    - 积分变量
a,b  - 积分区间端点(下限与上限)
opt  - 选项, 其值可为下列3个值之一:
       'continuous', 'CauchvPrincipalValue', 'AllSolutions'
```

- 积分命令 `int` 可以写成 `integrate`. 首字母大写是“惰性”版本, 即不求值.
- `int(expr, x)` 计算不定积分, 结果不带任意常数.

• `int(expr, x=a..b)` 和 `int(expr, x=a..b, opt)` 计算定积分.

– 当积分区间内有被积函数的间断点时, 若不带选项, 则定积分自动变成由间断点分隔成的几个子区间上定积分的和, 在每个子区间内部被积函数连续.

– 选项 `'continuous'` 指示积分不寻找间断点, 可设置环境变量 `_EnvContinuous` 为 `true` 代替这个选项.

– 选项 `'CauchyPrincipalValue'` 指示求定积分的 Cauchy 主值[1].

– 选项 `'AllSolutions'` 强制积分返回完整解集. 代替这个选项, 可设置环境变量 `_EnvAllSolutions` 为 `true`.

• 当 `a` 和 `b` 是有限复数时, 则在从 `a` 到 `b` 的直线上计算定积分.

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

$$\int \frac{x}{x^3-1}dx$$

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

$$-\frac{1}{6}\ln\left(x^2+x+1\right)+\frac{1}{3}\sqrt{3}\arctan\left(\frac{1}{3}(2x+1)\sqrt{3}\right)+\frac{1}{3}\ln(x-1)$$

对于某些特殊的表达式, 可能积分出特殊函数:

```
>int( exp(-x^2), x=0..1 );
```

$$\frac{1}{2}\mathrm{erf}(1)\sqrt{\pi}$$

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

$$0.7468241330$$

无穷区间上的积分 (本例结果中出现了 Euler 常数 γ):

```
>int( exp(-x^2)*ln(x), x=0..infinity );
```

$$\frac{1}{4}\sqrt{\pi}\gamma-\frac{1}{2}\sqrt{\pi}\ln(2)$$

当不定积分不能用有限表达式表示时, 就显示不定积分本身:

```
>int( exp(-x^2)*ln(x), x );
```

$$\int \mathrm{e}^{(-x^2)}\ln(x)dx$$

对此可先求出不定积分的 Taylor 展开, 然后对展开式进行积分:

```
>series( %, x=0, 4 );
```

$$(\ln(x)-1)x+\left(-\frac{1}{3}\ln(x)+\frac{1}{9}\right)x^3+\mathrm{O}(x^5)$$

[1] 设积分为 $\int_a^b f(x)\,\mathrm{d}x$, $c\in[a,b]$ 是 $f(x)$ 的间断点 (为简单假设只有这一个间断点且是内点), 如果 $\lim\limits_{\eta\to 0}\left(\int_a^{c-\eta} f(x)\,\mathrm{d}x+\int_{c+\eta}^b f(x)\,\mathrm{d}x\right)$ 存在, 则极限值就称为是积分的柯西主值.

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

$$\frac{1}{2}\ln(x)x^2 - \frac{3}{4}x^2 - \frac{1}{12}x^4\ln(x) + \frac{7}{144}x^4 + \mathrm{O}(x^6)$$

带选项的积分:

```
>int( 1/(x+a)^2, x=0..2, 'continuous');
```

$$\frac{2}{a(2+a)}$$

```
>int(1/x^3, x=-1..2, 'CauchyPrincipalValue');
```

$$\frac{3}{8}$$

椭圆积分的例子:

```
>Int(1/sqrt(2*t^4 - 3*t^2 - 2), t = 2..3);
```

$$\int_2^3 \frac{1}{\sqrt{2t^4 - 3t^2 - 2}}dx$$

```
>int(1/sqrt(2*t^4 - 3*t^2 - 2), t = 2..3);
```

$$\frac{1}{5}\sqrt{5}EllipticF\left(\frac{1}{3}\sqrt{7}, \frac{1}{5}\sqrt{5}\right) - \frac{1}{5}\sqrt{5}EllipticF\left(\frac{1}{2}\sqrt{2}, \frac{1}{5}\sqrt{5}\right)$$

带有 'AllSolutions' 选项的积分:

```
>int(1/x, x=a..2);
```

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

```
>r := int(1/x, x=a..2, 'AllSolutions');
```

$$r := \begin{cases} undefined & a < 0 \\ \infty & a = 0 \\ -\ln(a) + \ln(2) & 0 < a \end{cases}$$

```
>r assuming a > 0;
```

$$-\ln(a) + \ln(2)$$

```
>r assuming a < 0;
```

$$undefined$$

```
>int(abs(sin(x)), x=0..2*Pi*m) assuming m::integer;
```

$$\int_0^{2\pi m} |\sin(x)|dx$$

```
>int(abs(sin(x)),x=0..2*Pi*m,'AllSolutions') assumingm::integer;
```

$$4m$$

3.5.2 重积分

在 student 程序包中, 有一个二重积分命令 Doubleint 和一个三重积分命令 Tripleint, 它们分别产生二重积分和三重积分的符号, 但并不进行计算, 而且没有对应的全小写命令.

```
>with (student):
>Doubleint(g(x),x,y);
```

$$\iint g(x)dxdy$$

```
>Doubleint(h*g,x,y,D);
```

$$\iint_D hgdxdy$$

```
>Doubleint(h*g, x=1..n, y=2..4);
```

$$\int_2^4 \int_1^n hgdxdy$$

```
>Tripleint(g(x,y,z),x,y,z);
```

$$\iiint g(x,y,z)dxdydz$$

```
>Tripleint(h*g,x,y,z,V);
```

$$\iiint_V hgdxdydz$$

```
>Tripleint(h*g, x=1..n, y=2..4, z=a..b);
```

$$\int_a^b \int_2^4 \int_1^n hgdxdydz$$

在 Maple 中计算重积分, 可以对上述结果使用 value 命令, 或者直接写成累次积分的形式进行重积分计算. 以教材的例 9.2.10 为例:

```
>Doubleint( exp((y-x)/(y+x)), x=0..2-y, y=0..2 );
```

$$\int_0^2 \int_0^{2-y} \mathbf{e}^{(\frac{y-x}{y+x})} dxdy$$

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

$$\int_0^2 \int_0^{2-y} \mathbf{e}^{(\frac{y-x}{y+x})} dxdy = -\mathbf{e}^{(-1)} + \mathbf{e}$$

```
>int(int(exp((y-x)/(y+x)),x=0..2-y), y=0..2);
```

$$-\mathbf{e}^{(-1)} + \mathbf{e}$$

3.5.3 数值积分

在 Maple 中求数值积分, 是将 evalf 命令用到积分函数 int 或 Int 上. 前

者 evalf(int(...)) 实际上是先求出积分的符号解, 然会再进行数值计算; 后者 evalf(Int(...)) 不求出积分, 而是运用各种方法直接进行数值计算, 这才是真正意义上的数值积分.

求数值积分时, 有一些附加的选项, 选项格式是 选项名=选项值:

method = <方法名>	指定数值积分所用方法
digits = <正整数>	指定计算结果精度(中间计算自动使用更高精度)
epsilon = <数值>	指定计算结果的相对误差限.

数值积分的方法很多, 键入 "?evalfint" 看帮助信息, 参见图 15. 对于不熟悉计算数学的人, 可不指定方法, 让 Maple 去自动选择方法. 不指定数值积分精度, 则使用默认精度 (由变量 Digits 决定的精度):

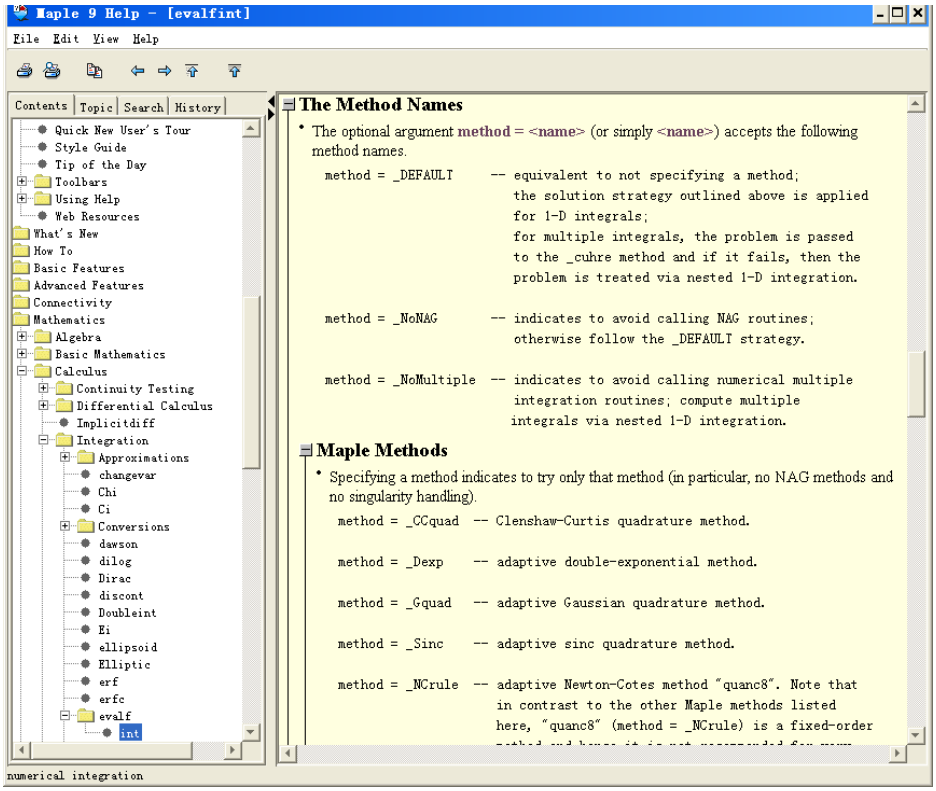


图 15: 数值积分的一些方法

```
>Digits;
```

$$10$$

```
>evalf(Int( sin(x)*ln(x)*exp(-x^3), x = 0..infinity ));
```

$$-.1957885158$$

指定精度和方法:

```
>e1 := 1/GAMMA(x):
>Int(e1,x=0..2) = evalf(Int(e1,x=0..2,digits=20,method=_Dexp));
```

$$\int_0^2 \frac{1}{\Gamma(x)}dx = 1.6263783986861406145$$

指定精度的另一种方式 `evalf[n](Int(...)`:

```
>e2 := 1/(1+ln(1+x)):
>Int(e2, x=0..1) = evalf[32]( Int(e2, x=0..1, method=_Gquad) );
```

$$\int_0^1 \frac{1}{1+\ln(1+x)}dx = 0.73716070962368003213791626905536$$

重积分的数值积分有两种写法, 一种是先写成累次积分形式 (并不求出积分), 再用数值方法求数值积分:

```
>e3 := exp(x+y+z)/((5*x+1)*(10*y+2)*(15*z+3)):
>Int(Int(Int(e3, x=0..4), y=0..3), z=0..sqrt(2));
```

$$\int_0^{\sqrt{2}}\int_0^3\int_0^4 \frac{\mathbf{e}^{(x+y+z)}}{(5x+1)(10y+2)(15z+3)}dxdydz$$

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

$$0.9331611325$$

另一种是简化写法, 只写一个 `Int`, 将多个变量的积分限写成一个顺序表, 上例可写成:

```
>evalf(Int(e3,[x=0..4, y=0..3, z=0..sqrt(2)]));
```

$$0.9331611325$$

有更多变量的数值积分:

```
>h := 1/(2 + sin(Pi*sqrt(87)*(x1+x2+x3+x4+x5+x6)));
```

$$h := \frac{1}{2+\sin(\pi\sqrt{87}(x1+x2+x3+x4+x5+x6))}$$

```
>evalf(Int(h,[x1=-1..1,x2=-1..1,x3=-1..1,x4=-1..1,x5=-1..1,x6=-1..1],
            method=_MonteCarlo, epsilon=0.5e-2));
```

$$36.91495229$$

当相对误差限设为 `0.5e-2` 即 0.5×10^{-2} 时, 大约只有 3 位有效数字, 例如上例结果取为 36.9 即可. 当设置了精度选项 `digits=k`, 则结果的相对误差大约是 $0.5\times10^{1-k}$, 若同时还设置 `epsilon` 选项, 则选项值不应小于这个式子.

3.6 用 Maple 演示定积分的几何意义

定积分的几何意义是曲边梯形的面积, 为了计算曲边梯形面积, 将曲边梯形

分割成若干个小曲边梯形, 用小矩形代替小曲边梯形, 小矩形与它所代替的小曲边梯形同宽, 小矩形的高度既可与小梯形左边同高, 也可与小梯形的右边同高, 或者与小梯形中线同高 (分别称为左矩形、右矩形和中矩形), 这些小矩形的面积之和就是大曲边梯形面积的近似值, 分割的越细近似程度就越高.

`student` 程序包中的 `leftbox`, `rightbox` 和 `middlebox` 命令用于画出函数曲线和一些小矩形 (分别是左矩形、右矩形和中矩形). 当分割越来越细时, 可以动态看出小矩形面积之和趋近于曲边梯形的过程.

命令用法:

```
leftbox  (f(x), x=a..b, <plot options>)
rightbox (f(x), x=a..b, <plot options>)
middlebox(f(x), x=a..b, <plot options>)
leftbox  (f(x), x=a..b, n, 'shading'=<color>, <plot options>)
rightbox (f(x), x=a..b, n, 'shading'=<color>, <plot options>)
middlebox(f(x), x=a..b, n, 'shading'=<color>, <plot options>)
```

参数:

`f(x)`	- 关于x的表达式
`x`	- 积分变量
`a`	- 积分下限
`b`	- 积分上限
`n`	- (可选项) 分割个数(默认值是分割成4个小矩形)
`color`	- 涂布矩形的颜色
`plot options`	- (可选项) 绘图选项

将上述命令名中的 `box` 改成 `sum`, 则是求小矩形面积之和.

假设函数为 $f(x)=3(x+1)^{\frac{1}{10}}+x\sin x/3$, 区间是 $[0,2\pi]$. 容易求出 $\int_a^b f(x)\,\mathrm{d}x=\frac{30}{11}(2\pi+1)^{(11/10)}-\frac{2}{3}\pi-\frac{30}{11}\approx 19.40435184$:

```
>f := x->3*(x+1)^(1/10)+x*sin(x)/3; a := 0; b := 2*Pi;
```

$$f := x->3(x+1)^{(1/10)}+\frac{1}{3}x\sin(x)$$

$$a := 0$$

$$b := 2\pi$$

```
>int(f(x),x=a..b);
```

$$\frac{30}{11}(2\pi+1)^{(11/10)}-\frac{2}{3}\pi-\frac{30}{11}$$

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

$$19.40435184$$

下面用定积分的几何意义来求其值. 首先画出左矩形和右矩形 (中矩形放在后面再画), 不指定小矩形个数 (因此只有 4 个), 左矩形不指定颜色 (默认使用浅绿色), 右矩形指定用黄色. 横竖轴用等长单位, 参见图 16.

```
>with(student):
>leftbox(f(x),x=a..b, scaling=constrained);
>rightbox(f(x),x=a..b, shading=YELLOW, scaling=constrained);
```

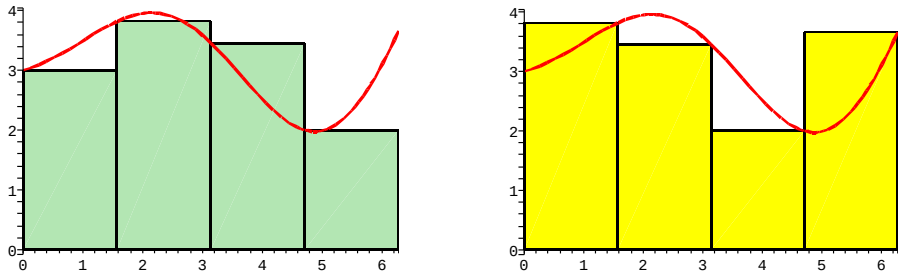


图 16: 左矩形与右矩形

左矩形面积之和:

```
>leftsum(f(x),x=a..b);
```

$$\frac{1}{2}\pi\left(\sum_{i=0}^{3}\left(3\left(\frac{1}{2}i\pi+1\right)^{(1/10)}+\frac{1}{6}i\pi\sin\left(\frac{1}{2}i\pi\right)\right)\right)$$

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

$$19.28793890$$

右矩形面积之和:

```
>rightsum(f(x),x=a..b);
```

$$\frac{1}{2}\pi\left(\sum_{i=1}^{4}\left(3\left(\frac{1}{2}i\pi+1\right)^{(1/10)}+\frac{1}{6}i\pi\sin\left(\frac{1}{2}i\pi\right)\right)\right)$$

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

$$20.32297433$$

由于小矩形个数较少, 所以面积误差较大, 增加小矩形个数, 面积的近似值应该更好, 取一系列值作实验:

```
>boxes := [seq(i^3, i=3..8)];
```

$$boxes := [27, 64, 125, 216, 343, 512]$$

对列表中的每个数, 分别计算左矩形、右矩形和中矩形的面积值:

```
>seq(evalf[5](leftsum(f(x),x=a..b,n)),n=boxes);
```

$$19.336, 19.374, 19.389, 19.395, 19.399, 19.401$$

```
>seq(evalf[5](rightsum(f(x),x=a..b,n)),n=boxes);
```

$$19.490, 19.439, 19.421, 19.414, 19.410, 19.408$$

```
>seq(evalf[5](middlesum(f(x),x=a..b,n)),n=boxes);
```

$$19.400, 19.403, 19.405, 19.405, 19.405, 19.404$$

可见随着小矩形个数的增加, 面积的误差越来越小. 还可看出, 中矩形逼近的更好些.

为了从几何上观察用中矩形逼近曲边梯形的过程, 我们画出一系列的中矩形图形, 为了看得清楚, 分割的个数取等差数列 2, 3, 4, 5, . . . , 50:

```
>boxes := [seq(i, i=2..50)];
```

对于每个分割值都作一个 `middlebox` 图形, 用对应的 `middlesum` 值作标题, 生成一列图形:

```
>S:=seq(middlebox(f(x),x=a..b,n,title=convert(evalf[5](middlesum(f(x),
          x=a..b,n)),string)),n=boxes):
```

使用 `plots` 包中的 `display` 命令把这一列图形显示成一个动画:

```
>with(plots):
>display(S, insequence=true, scaling=constrained);
```

为了在纸面上看出动态效果, 我们抽取了9帧图形, 排成连环画, 见图 17.

```
>s := [1,2,4,7,11,17,25,35,49]:
>aa := array(1..3, 1..3):
>for i to 3 do for j to 3 do aa[i,j] := S[s[(i-1)*3+j]] od:od:
>display(aa,font=[TIMES,ROMAN,10],axesfont=[TIMES,ROMAN,6]);
```

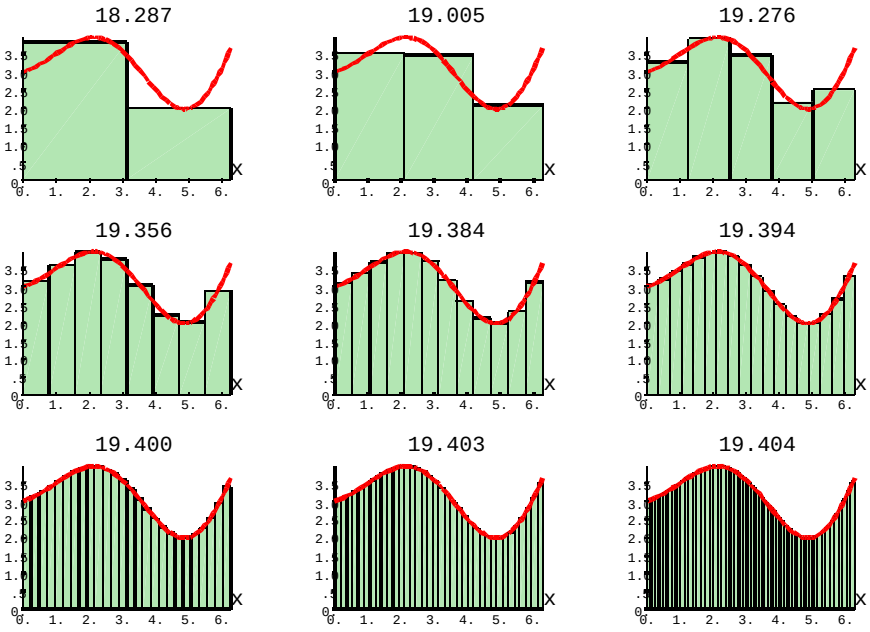


图 17: 定积分的几何意义

4 微分方程

4.1 常微分方程符号解

解常微分方程的命令是 dsolve:

```
dsolve(ODE)
dsolve(ODE, y(x), extra_args)
dsolve({ODE, ICs}, y(x), extra_args)
dsolve({sysODE, ICs}, {funcs}, extra_args)
```

参数:

- ODE - 常微分方程
- y(x) - 待求的单变量函数
- ICs - 初始条件
- {sysODE} - 一组微分方程
- {funcs} - 一组待求的函数
- extra_args - (可选项) 依赖于待解问题的类型

此处 {x, y(x)} 表示任何一对自变量和因变量.

下面例题选自《高等数学》下册第 12 章.

例 4.1: 求微分方程 $y' = \dfrac{y}{1+x^2}$ 通解和满足初始条件 $y(0) = 1$ 的特解.

```
>ode1 := diff(y(x),x)=y(x)/(1+x^2);
```

$$ode1 := \frac{d}{dx}y(x) = \frac{y(x)}{1+x^2}$$

```
>dsolve(ode1);
```

$$y(x) = \_C1\,\mathrm{e}^{\arctan(x)}$$

其中 $\_C1$ 表示一个任意常数.

用 `odetest(解,微分方程)` 验证微分方程的解, 若返回 0 (或经过化简, 变换得到 0), 表明微分方程的解是正确的. 例如上例的解是对的:

```
>odetest(%,ode1);
```

$$0$$

带上初始条件求解微分方程 (微分方程与初始条件放在花括号中), 就得到了特解:

```
>ans1 := dsolve({ode1, y(0)=1});
```

$$ans1 := y(x) = \mathrm{e}^{\arctan(x)}$$

若想了解 (咨询) 所解微分方程的类型或解法, 可使用 `DEtools` 包中的 `odeadvisor` 命令, 先加载包:

```
>with(DEtools);
```

[*DEnormal, DEplot, DEplot3d, DEplot_polygon, DFactor, DFactorLCLM, DFactorsols, Dchangevar, GCRD, LCLM, MeijerGsols, PDEchangecoords, RiemannPsols, Xchange, Xcommutator, Xgauge, abelsol, adjoint, autonomous, bernoullisol, buildsol, buildsym, canoni, caseplot, casesplit, checkrank, chinisol, clairautsol, constcoeffsols, convertAlg, convertsys, dalembertsol, dcoeffs, de2diffop, dfieldplot, diffop2de, dperiodic_sols, dpolyform, dsubs, eigenring, endomorphism_charpoly, equinv, eta_k, eulersols, exactsol, expsols, exterior_power, firint, firtest, formal_sol, gen_exp, generate_ic, genhomosol, gensys, hamilton_eqs, hypergeomsols, hyperode, indicialeq, infgen, initialdata, integrate_sols, intfactor, invariants, kovacicsols, leftdivision, liesol, line_int, linearsol, matrixDE, matrix_riccati, maxdimsystems, moser_reduce, muchange, mult, mutest, newton_polygon, normalG2, odeadvisor, odepde, parametricsol, phaseportrait, poincare, polysols, power_equivalent, ratsols, redode, reduceOrder, reduce_order, regular_parts, regularsp, remove_RootOf, riccati_system, riccatisol, rifread, rifsimp, rightdivision, rtaylor, separablesol, solve_group, super_reduce,*

symgen, symmetric_power, symmetric_product, symtest, transinv, translate, untranslate, varparam, zoom]

再使用命令:

```
>odeadvisor(ode1);
```

$$[\_separable]$$

这表示可分离变量型.

可以画出这个特解的图形 (图18):

```
>plot(rhs(ans1),x=-5..5,scaling=constrained,tickmarks=[4,2]);
```

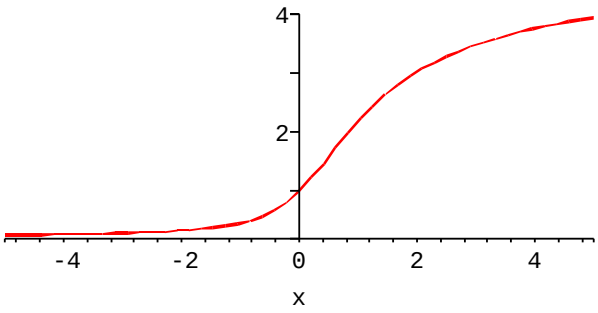


图 18: 微分方程特解的图形示例

若想了解 `dsolve` 求解过程的一些细节, 可提高相应的信息显示水平, 将 `infolevel[dsolve]` 的值由默认的 1 改为从 2 到 5 之间的一个整数, 例如:

```
>infolevel[dsolve] := 3;
```

$$infolevel_{dsolve} := 3$$

```
>dsolve(ode1);
```

Methods for first order ODEs:

— Trying classification methods —

trying a quadrature

trying 1st order linear

< − 1st order linear successful

$$y(x) = \_C1\,\mathrm{e}^{\arctan(x)}$$

求微分方程组 $\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = -x - 5y + 1, \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = x + y + t \end{cases}$ 满足初始条件 $x(0) = y(0) = 1$ 的特解 (下册例 12.4.2). 为了输入方便, 设置了几个别名.

```
>alias(x=x(t), x0=x(0), y=y(t), y0=y(0)):
>dsolve(diff(x,t)=-x-5*y+1, diff(y,t)=x+y+t, x0=1, y0=1,x,y);
```

$$\left\{x = -\frac{15}{8}\sin(2\,t) + \frac{5}{4}\cos(2\,t) - \frac{1}{4} - \frac{5}{4}t,\ y = \frac{1}{2}\cos(2\,t) + \frac{7}{8}\sin(2\,t) + \frac{1}{2} + \frac{1}{4}t\right\}$$

4.2 常微分方程的幂级数解

教材上第 §12.5 节有一个定理: 对于二阶线性变系数齐次微分方程

$$y''(x) + a(x)y'(x) + b(x)y(x) = 0,$$

如果系数 $a(x)$, $b(x)$ 都能展开成 x 的幂级数, 且收敛区间为 $|x| < R$, 那么该微分方程有形如

$$y(x) = \sum_{n=0}^{\infty} a_n x^n$$

的解, 且也以 $|x| < R$ 为收敛区间. 所以对二阶变系数齐次方程可以求其幂级数形式的解. 对于其他当微分方程的解不能用初等函数或其积分表示时, 也可以试求幂级数解.

使用 `dsolve` 时, 加上选项 `series` 或 `type=series` 就得到幂级数解.

求微分方程 $y'' + xy = 0$ 满足初始条件 $y(0) = 1$, $y'(0) = 0$ 的特解 (下册例 12.5.2).

```
>ode2 := D(D(y))(x)+x*y(x);
```

$$ode2 := \mathrm{D}^2(y)(x) + xy(x)$$

```
>dsolve({ode2,y(0)=1,D(y)(0)=0},y(x),type=series);
```

$$y(x) = 1 - \frac{1}{6}x^3 + O(x^6)$$

为了得到更精确的解, 可以将截断阶数由默认的 6 提高到更大的值,例如 16:

```
>Order := 16;
```

$$Order := 16$$

```
>dsolve({ode2,y(0)=1,D(y)(0)=0},y(x),type=series);
```

$$y(x) = 1 - \frac{1}{6}x^3 + \frac{1}{180}x^6 - \frac{1}{12960}x^9 + \frac{1}{1710720}x^{12} - \frac{1}{359251200}x^{15} + O(x^{16})$$

若不求级数解, 则解为

```
>dsolve(ode2,y(0)=1,D(y)(0)=0,y(x));
```

$$y(x) = \frac{1}{2}\Gamma\left(\frac{2}{3}\right)3^{(2/3)}\,AiryAi(-x) + \frac{1}{2}\Gamma\left(\frac{2}{3}\right)3^{(1/6)}\,AiryBi(-x)$$

将这个解展开, 也得到同样的级数解:

```
>series(rhs(%), x, 16);
```

$$y(x) = 1 - \frac{1}{6}x^3 + \frac{1}{180}x^6 - \frac{1}{12960}x^9 + \frac{1}{1710720}x^{12} - \frac{1}{359251200}x^{15} + O(x^{16})$$

设置微分方程求解显示信息水平为 2, 可以看到两种解法 (有无选项 `type=series`) 的求解过程是不同的.

4.3 常微分方程初值问题的数值解

随着 Maple 版本的提高, 求解微分方程的能力越来越强, 但仍有大部分的微分方程求不出符号解. 对于常微分方程初值问题, 使用 `dsolve` 求解时, 加上选项 `numeric` 或 `type=numeric` 就可得到数值解.

考虑初值问题:

```
>ode3 := {diff(u(t),t)=sin(t*u(t)), u(0)=2};
```

$$ode3 := \left\{ \frac{d}{dt}u(t) = \sin(t\,u(t)),\ u(0) = 2 \right\}$$

求不出它的符号解, 但可求得数值解:

```
>U := dsolve(ode3,u(t),type=numeric);
```

$$U := \mathbf{proc}(x\_rkf45)\ \ldots\ \mathbf{end\ proc}$$

得到的解是一个过程, 只要给出 t 的值, 由这个过程就可得到对应的 $u(t)$ 的近似值 (称为数值解):

```
>U(0); U(1); U(2.5);
```

$$[t = 0.,\ u(t) = 2.]$$

$$[t = 1.,\ u(t) = 2.6697149099118098 2]$$

$$[t = 2.5,\ u(t) = 1.6983989662254455 0]$$

求微分方程初值问题数值解的默认方法是 rkf45 方法, 即 Fehlberg 4~5阶 Runge–Kutta 法. 其他的数值方法请看有关 dsolve 的帮助信息.

4.4 常微分方程的图形表示

如果求出了微分方程初值问题的符号解, 那么可以参见前面介绍过的绘图命令画出它的图形, 不再讨论.

4.4.1 微分方程数值解的图形

plots 程序包中的 odeplots 用于绘制微分方程数值解的图形, 最简单的用法是将求数值解时的输出结果作为 odeplots 的命令参数. 以上节的数值解为例, 使用了命令的长格式, 见右图:

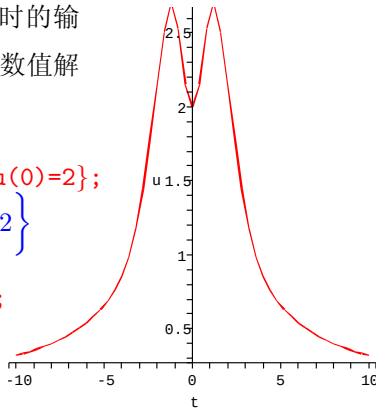
```
>ode3 := {diff(u(t),t)=sin(t*u(t)), u(0)=2};
```

$$ode3 := \left\{ \frac{d}{dt}u(t) = \sin(t\,u(t)),\ u(0) = 2 \right\}$$

```
>U := dsolve(ode3,u(t),type=numeric);
```

$$U := \mathbf{proc}(x\_rkf45)\ \ldots\ \mathbf{end\ proc}$$

```
>plots[odeplot](U);
```



还可以指定一些可选项, 例如 范围, 颜色, 线条粗细, 等等:

```
>plots[odeplot](U, 0..15, color=blue, thickness=2);
```

如果加上选项 frames=<n> (n 为正整数), 则结果是一个动画:

```
>plots[odeplot](U, 0..15, frames=20);
```

4.4.2 DEtools 程序包中的绘图命令

在微分方程工具包 DEtools 中含有几个绘图命令:

DEplot	绘制微分方程(组)解的图形
DEplot3d	绘制微分方程(组)解的三维图形
dfieldplot	绘制微分方程(组)的方向场
PDEplot	绘制一阶偏微分方程的图形
phaseportrait	绘制单个方程的解曲线或方程组的方向场

5 Maple 编程简介

程序设计语言都包含四种控制结构: 顺序结构, 分支结构, 循环结构, 过程.

5.1 过程

过程的基本结构:

```
名字 := proc(形式参数表)
    local 局部变量表;
    globai 全局变量表;
    option 选项;
    description 过程的描述域;
    过程体
end proc;
```

定义过程时, 除了 proc 和 end proc 不可省略外, 其他语句根据实际需要都可省略.

过程体中若有语句

```
RETURN(表达式);
```

则过程返回 表达式 的值, 否则, 在正常情况下会返回过程体中最后一个可执行语句的运行结果.

定义一个函数实际上就是定义一个过程, 例如 $g(x) = \mathbf{e}^{\cos x} - \ln(2 + \sin x)$ 可定义为过程:

```
g := proc(x)
   exp(cos(x))-ln(2+sin(x))
end proc;
```

这样定义的函数与用映射符号定义的函数是一样的. 画出这个函数的图形, 可以看出在 1.5 附近有一个根, 可如下求出这个根:

```
>fsolve(g(x)=0, x, x=1..2);
```

1.477924405

有关过程的局部变量、全局变量、变量的作用域等概念与典型的编程语言是类似的, 但要注意在 Maple 中有分层求值的概念, 在过程体的执行过程中, 对于全局变量进行完全求值, 对于局部变量则只进行一层求值. 如果需要在过程体中对局部变量完全求值, 只需对该变量使用 eval 命令.

对于需要实际参数的过程, Maple 先计算实参的值, 然后用实参的值代换对应的形参. 实参个数可以不同于形参的个数, 如果实参个数少于形参个数, 则只有当缺少的实参在过程体的运行中用到时才会报错; 如果实参个数过多, 未用到的实参则被忽略.

过程的形参表中不必写出所有形参, 在过程体中可以使用 `args` 接收实参序列, 第 44 页 §3.4.6 的例子中用到了 `args`, 其中 `nargs` 的值是实参的个数.

5.2 分支结构, if 算子

分支结构的用途是按着不同的情况选择不同的结果, 常用的有 3 种形式, 其中的布尔表达式可以是条件表达式或逻辑表达式, 语句 `fi` 可写成 `end if`:

```
if 布尔表达式 then
    语句序列
fi
```

Maple 首先求布尔表达式的值, 当值为 `true` 时运行 `then` 和 `fi` 之间的语句序列, 当值为 `false` 或 `FAIL` 时, 跳过(不运行)上述的语句序列而结束分支结构.

```
if 布尔表达式 then
    语句序列1
else
    语句行列2
fi
```

Maple 首先求布尔表达式的值, 当值为 `true` 就运行 `then` 和 `else` 之间的语句序列1, 然后结束分支结构; 当值为 `false` 或 `FAIL` 时, 则跳过语句序列1, 运行 `else` 与 `fi` 之间的语句序列2, 然后结束分支结构.

```
if 布尔表达式1 then
   语句序列1
elif 布尔表达式2 then
   语句序列2
elif 布尔表达式3 then
   语句序列3
  ......
elif 布尔表达式n then
```

```
   语句序列n
else
   语句序列f
fi
```

Maple 按顺序求各布尔表达式的值:

若 布尔表达式1 的值为 true, 则运行 语句序列1, 然后结束分支结构;

若 布尔表达式2 的值为 true, 则运行 语句序列2, 然后结束分支结构;

.....

若 布尔表达式n 的值为 true, 则运行 语句序列n, 然后结束分支结构;

若所有布尔表达式的值都是 false, 则运行 语句序列f, 然后结束分支结构;

可以不存在 else 分支, 此时若所有布尔表达式的值都是 false, 则分支结构中的所有语句序列都不运行.

例 5.1: 斐波那契 (Fibonacci) 数列: 1, 1, 2, 3, 5, 8, 13, 21, 34, $\cdots$

$$\text{递推公式 } f(n)=\begin{cases}1, & \text{当 } n=1 \text{ 或 } n=2,\\ f(n-1)+f(n-2), & \text{当 } n>=3.\end{cases}$$

求第 30 项的大小. 首先编写如下一个过程, 然后对给定的正整数 n, 就可求出对应的斐波那契数了.

```
fib:=proc(n)
    if not( is(n,integer) and is(n>0) ) then
       lprint(‘argument must be positive integer!‘)
    elif n=1 or n=2 then
       1
    else
       fib(n-1) + fib(n-2)
    fi
end proc;
```

```
>fib(30);
```

832040

所用时间为:

```
>time(fib(30));
```

4.309

如果在形参表中指定形参的类型, 上述过程可以简化为

```
fib:=proc(n::posint)
    if n <= 2 then
       1
    else
       fib(n-1) + fib(n-2)
    fi
end proc;
```

关于整数的类型有 integer (整数), posint (正整数), negint (负整数), nonposint (非正整数), nonnegint (非负整数).

输入

```
>?type
```

可以看到 Maple 中的各种类型, 注意类型 `anything` 可以代表除表达式序列外的任何类型.

在递归过程中, 每一步都需要前两步的结果, 使用过程选项 `remember`, Maple 就将每步计算结果都存储在记录表中, 后面用到前面结果时就不必重新计算了, 可以显著提高效率, 例如上例改为

```
fib:=proc(n::posint)
    option remember;
    if n <= 2 then
       1
    else
       fib(n-1) + fib(n-2)
    fi
end proc;
```

即使求第一万项的值, 也只是一眨眼的功夫:

```
>time(fib(10000));
```

0.210

另一种控制分支结构的语句是 `if` 算子:

```
`if`(布尔表达式, 表达式1, 表达式2)
```

注意不要忘了倒引号 (重音号) “`” (位于键盘左上角第二排). 该语句当布尔表达式的值为 `true` 时, 返回表达式1的值, 否则返回表达式2的值.

```
>a := 3:  b := 5:
>5*(Pi + `if`(a > b, a, b));
```

$$5\pi + 25$$

5.3 循环结构

循环就是翻来覆去地重复运行一些语句, 最基本的循环语句是:

```
do 语句序列 od;
```

其中 od 可写成 end do, 语句序列可称为循环体. 这种循环结构会无限重复运行循环体, 退出(结束)这种循环的方法是在循环体中适当位置写上语句break, 或者在循环体中的过程中有语句return, 并且在循环时能运行到break或return语句.

例 5.2: 求最小的n, 使得 $3*6*9*12*\cdots*n >= 10^{10}$.

```
>n:=0:  p:=1:
>do n:=n+3:  p:=p*n:  if p>=10^10 then print(n); break fi od:
```

$$30$$

通常并不需要无限循环, 实际需要的是有一定重复次数的循环, 或者是在一定条件下的有限循环. 循环结构的一般形式是:

```
[for <name>] [from <expr>] [by <expr>] [to <expr>] [while <expr>]
        do <statement sequence> od;
```

或

```
[for <name>] [in <expr>] [while <expr>]
        do <statement sequence> od;
```

方括号中的内容是可选项, 根据实际需要选用, 下面是几种常用的形式.

5.3.1 while 循环

```
while 布尔表达式 do
    语句序列
od;
```

当布尔表达式的值为true时, 执行语句序列, 然后又检查布尔表达式, 若仍为true, 则再次执行语句序列, 然后又去检查布尔表达式..., 直到布尔表达式的值为false时结束循环.

例 5.3: 以指定精度p显示调和级数前n项的和 $s = \sum_{k=1}^{n} \frac{1}{k}$.

```
ex3 := proc(n::posint, p::posint)
   local s, k:
   description ‘以指定精度 p 显示调和级数前 n 项的和‘;
   s:=0: k:=1:
   while k<=n do
      s := s + 1/k:
      k := k + 1:
   od:
   printf("sum = %s", convert(evalf(s,p),string) );
end proc;
```

```
>ex3(10000,50);
```

sum = 9.7876060360443822641784779048516053348592629455777

本例计算过程都是有理数的四则运算, 计算结果是精确的有理数, 仅最后显示时指定了显示精度. 下面是另一种常用的循环控制语句:

5.3.2 for 循环

```
for 循环变量 from 初始值 by 步长 to 终止值 while 条件 do
        语句序列
od;
```

执行过程是:

1. 当循环变量的初始值没有超出终止值并且条件为 true 时就运行语句序列, 然后转到第 2 步; 否则结束循环.

2. 将循环变量当前值加上步长作为新的当前值, 转到第 3 步.

3. 若循环变量的当前值没有超出终止值并且条件为 true 时就运行语句序列, 然后转到第 2 步; 否则结束循环.

几点说明:

1. 当步长是正数时, 超出终止值是指大于终止值; 当步长是负数时, 超出终止值是指小于终止值.

2. 将 “for 循环变量” 称为 for 子句, 将 “from 初始值” 称为 from 子句, 以此类推, 除了 for 子句必须放在首位外, 其它子句可按任意顺序放置.

3. 任何子句都可省略, 省略的子句实际上被它的缺省值 (默认值) 代替了, 缺省值见表 7.

例 5.4: 找出不小于 10^{10} 的最小素数.

表 7: for 循环中控制子句的缺省值

子句	for	from	by	to	while
缺省值	虚拟变元	1	1	infinity	true

```
>for i from 10^10 while not is(i,prime) do od;
>i;
```

10000000019

例 5.5: 求一个 6 位数, 将该数的首位数移到最后得到的数是该数的 3 倍.

```
>for n from 10^5 to (10^6-1)/3 do
    if irem(n,10^5) * 10 + iquo(n,10^5) = 3 * n then
        print(n);
    fi;
 od;
```

142857

285714

5.3.3 for-in 循环

```
for 循环变量 in 表达式 while 条件 do
      语句序列
od;
```

循环过程是循环变量逐次取表达式的各个分量的值, 每次取值后当条件为 `true` 时运行循环体, 当条件为 `false` 时不运行循环体而直接取下一个值, 直至遍历结束. 这里的表达式可以是表达式序列, 顺序表, 集合, 和式或乘积.

```
>tot:=1:  for z in 1, x, y, q^2, 3 do tot := tot*z od:  tot;
```

$3xyq^2$

5.3.4 break 和 next

`break` 和 `next` 是循环结构的另外两个控制语句: 在循环时运行到 `break` 就退出包含它的最内层的循环结构, 继续运行该层循环结构后面的第一个语句. 运行到 `next` 时则是结束本轮循环, 即跳过循环体中下面尚未运行的语句, 开始下一轮的循环. 这两个语句与在 C 语言中的用法相同, 不再举例.