

MATHEMATICA 用法简介

前 言

Mathematica 是美国 Wolfram Research 公司开发的数学软件。它的主要使用者是从事理论研究的数学工作者和其它科学工作者、从事实际工作的工程技术人员、学校里的学生和教师。Mathematica 可以用于解决各种领域的涉及复杂的符号计算和数值计算的问题。它可以完成许多复杂的工作，如求不定积分、做多项式的因式分解等等。它代替了许多以前仅仅只能靠纸和笔解决的工作，这种思维和解题工具的革新可能对各种研究领域和工程领域产生深远的影响。

Mathematica 可以做许多符号演算工作：它能做多项式的计算、因式分解、展开等，做各种有理式计算，求多项式、有理式方程和超越方程的精确解和近似解，做数值的或一般代数式的向量、矩阵的各种计算，求极限、导数、积分，做幂级数展开，求解某些微分方程等。Mathematica 还可以做任意位数的整数或分子分母为任意大整数的有理数的精确计算，做具有任意位精度的数值(实、复数值)的计算。所有 Mathematica 系统内部定义的整函数、实(复)函数也具有这样的性质。使用 Mathematica 可以很方便地画出用各种方式表示的一元和二元函数的图形。通过这样的图形，我们常可以立即形象地把握住函数的某些特性，而这些特征一般很难从函数的符号表达式中看清楚。

Mathematica 的能力不仅仅在于上面说的这些功能，更重要的在于它把这些功能有机地结合在一个系统里。在使用这个系统时，人们可以根据自己的需要，一会儿从符号演算转去做图形，一会又转去做数值计算。这种灵活性能带来极大的方便，常使一些看起来非常复杂的问题变得易如反掌。在学习和使用 Mathematica 的过程中读者会逐步体会这些。Mathematica 还是一个很容易扩充和修改的系统，它提供了一套描述方法，相当于一个编程语言，用这个语言可以写程序，解决各种特殊问题。

第一节 Mathematica 基本使用方法

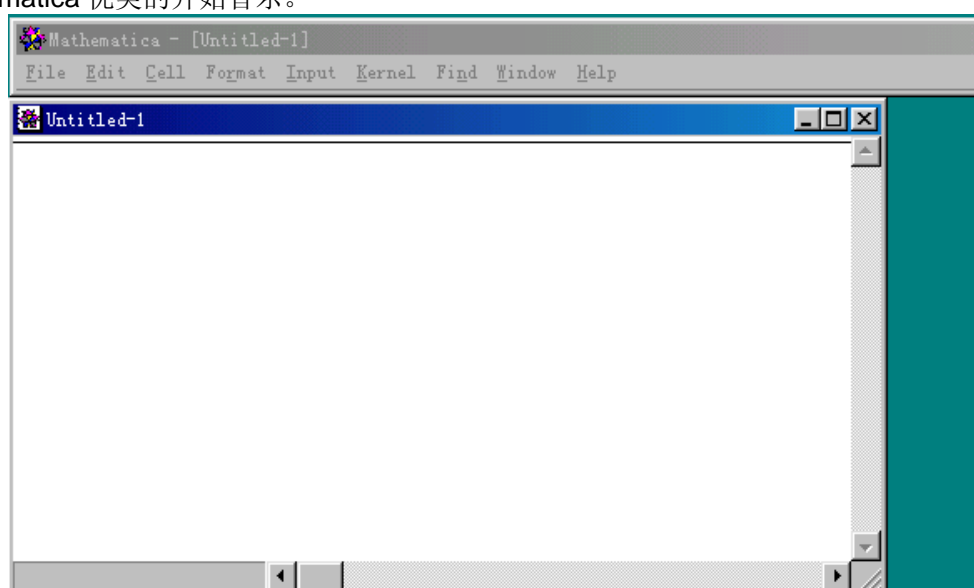
1.1 Mathematica 的使用方法

Mathematica 系统已经被移植到许多不同的计算机和运行环境上。在微型机上可以用的有 MS-DOS386 版本(不要求 387 协处理器)和 MS-DOS386 / 387 版本(要求 387 协处理器)。它们都可以在常见的 386、486 机器上运行，在 4 兆内存的机器上可以很好地工作，有更大的内存工作速度还会提高。另外还有在 Windows 上运行的版本，用户界面大大改善，但对系统的要求相对较高。在苹果公司的 Macintosh 机器以及许多工作站和大型机上也有可用的 Mathematica 版本。Mathematica 系统的开发者(美国 Wolfram 研究公司)于 1989 年推出系统的 1.2 版，于 1991 年推出 2.0 版。2.0 版比 1.2 版在功能上有了不小的扩充(增加了一大批函数等)，但基本结构变化不大。2.0 版对计算机的要求也有所提高。

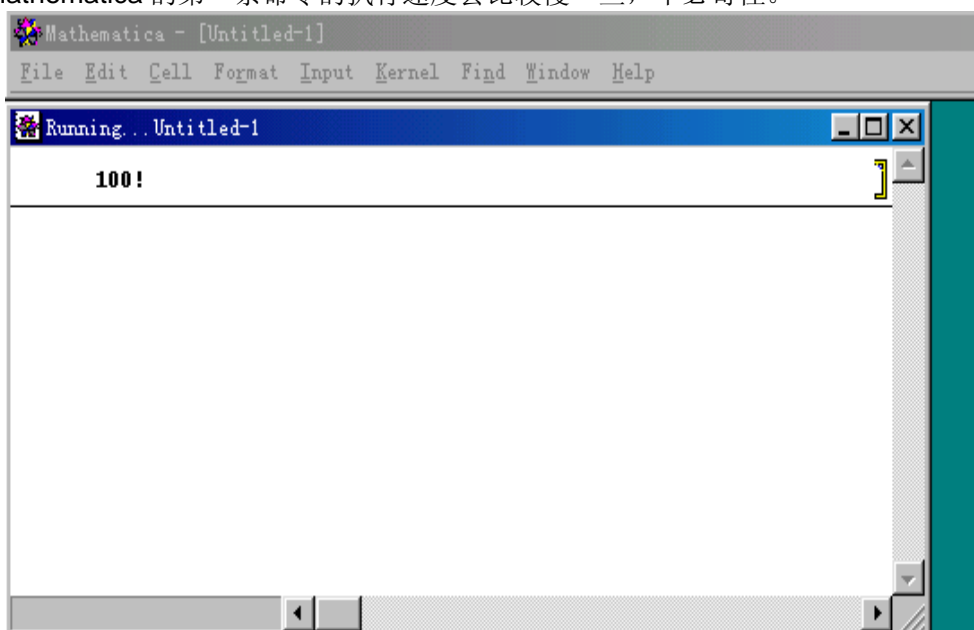
Mathematica 是个交互式的系统。它的用户界面依版本不同有两种。一种是行文形式的，一种是图形形式的。后一种使用起来方便些，看起来也漂亮些，但要求更大的内存，速度会有所降低，但它使用方便，而且可以很方便的与 Windows 下其他软件如 Word 等交换信息，图形处理也更方便。在有较好的微机的情况下，选择 Windows 版本是更好的选择。这里介绍的是目前使用较多的 Mathematica 3.0 版，目前最新的是 Mathematica 4.0 版。

1.2 启动与退出

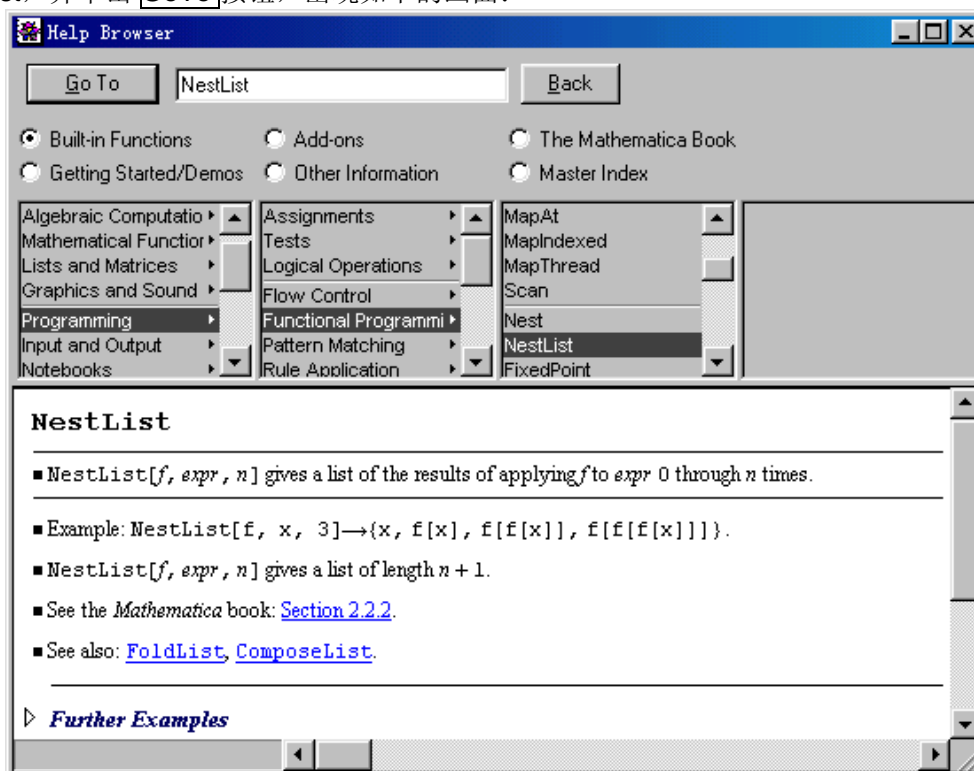
在 Windows 下，启动和结束 Mathematica 的方式和其他 Windows 应用程序没有什么两样，只需找到 Mathematica 图标，双击它即可。此时会出现 Mathematica 初始屏，显示版本信息和用户信息。等待约一秒即可进入 Mathematica 主窗口，并出现第一个 notebook 窗口(Untitled-1.nb)，可以开始在此窗口中输入命令进行计算工作。若有声卡，还可听到 Mathematica 优美的开始音乐。



需要注意的是，Mathematica 的计算核心并不是马上启动，只有在您给出了确实的计算指令后才开启，比如：100!，这时窗口的标题栏中会出现“Running...Untitled-1”等信息。所以 Mathematica 的第一条命令的执行速度会比较慢一些，不必奇怪。

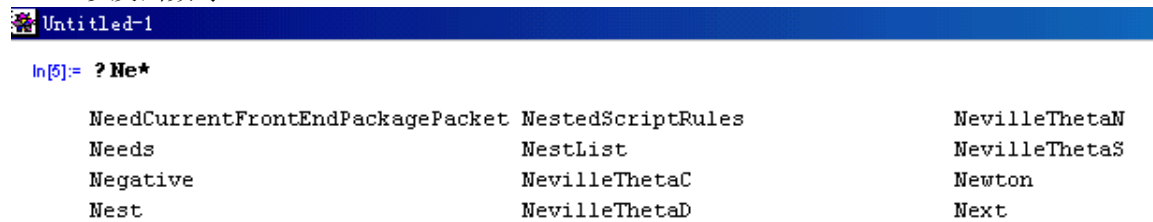


当我们要查找关于“NestList”的用途与用法，可在上方 **GoTo** 后面的白色区域输入：NestList，并单击 **GoTo** 按钮，出现如下的画面：



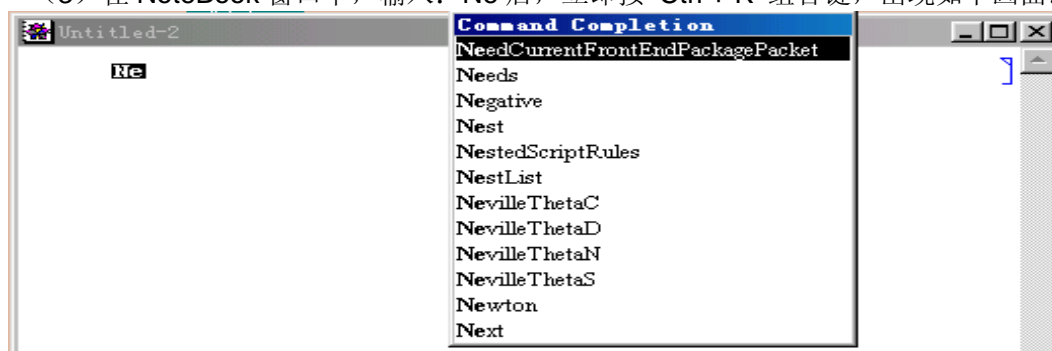
从中我们可以得到关于“NestList”的语法说明、基本例子、可参考的内容，以及进一步的较全面的例子。你可以修改例子中的一些参数，并可当场执行看到结果。这是学习 Mathematica 的较为有效的方法。

(2) 在 Notebook 窗口中，输入：? Ne*，并执行它，会出现所有以 Ne 开头的变量、常量以及函数等。



再完整地输入：? NestList 并执行它，会得到较完整的语法解释，但没有方法（1）来得完整和具体。

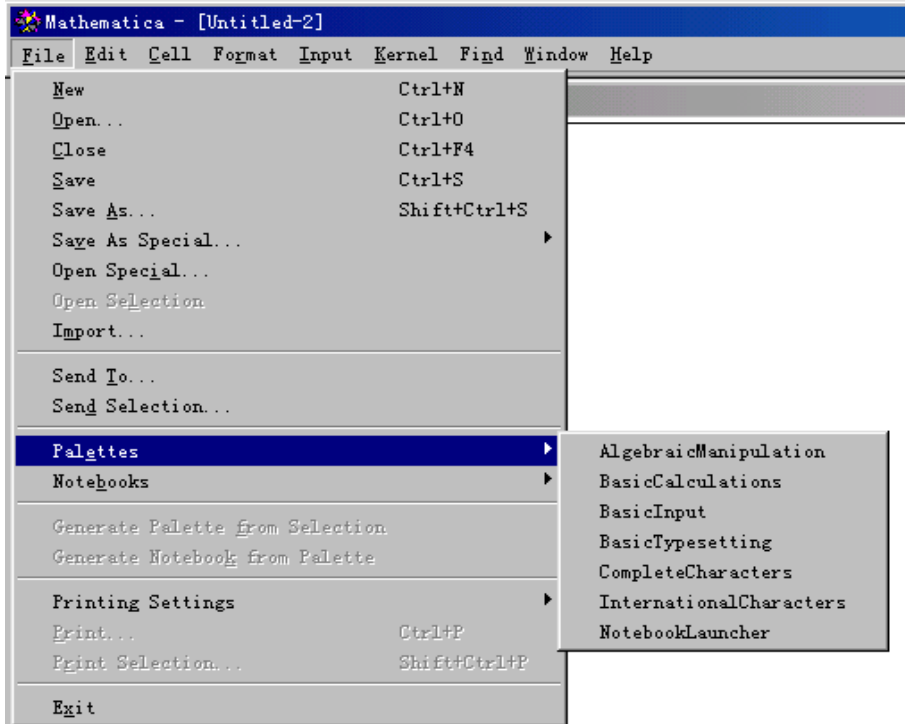
(3) 在 Notebook 窗口中，输入：Ne 后，立即按 **Ctrl + K** 组合键，出现如下画面：



此时用鼠标单击 **NestList** 行，便可将完整的命令或函数转录到光标所在位置处（我们称之

为：自动补全法），你只要补充必要的参数等，就可执行它。

(4) 选“File”菜单，再选“Palettes”菜单项，出现如下画面：



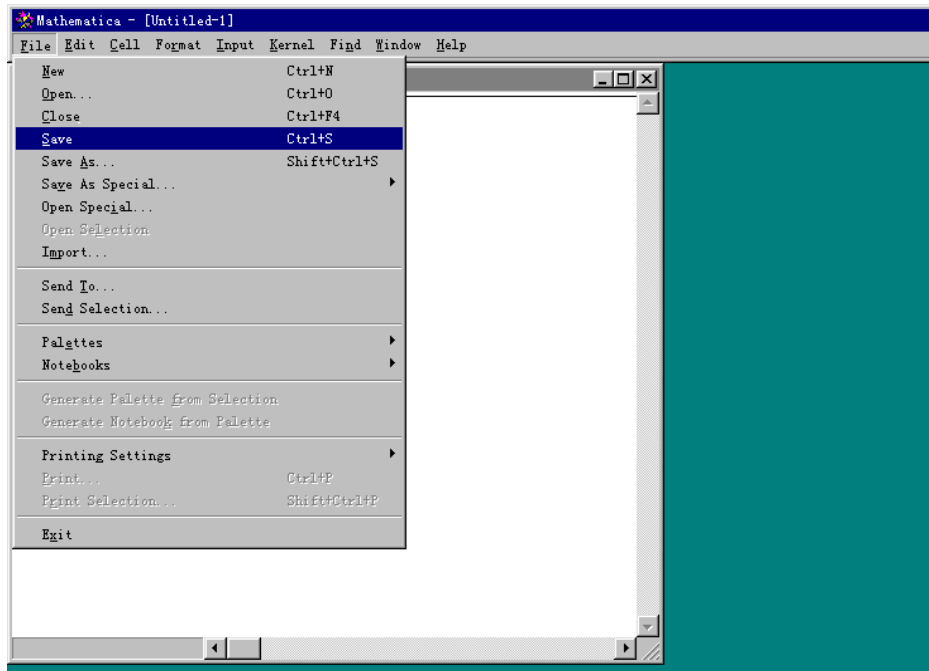
这时，你可以只利用选项板（Palettes）而不用键盘就可以完成必要的输入，这一点在 Mathematica4.0 中已有更好的改进。限于篇幅，本项从略。

1.3 文件的存储与读取

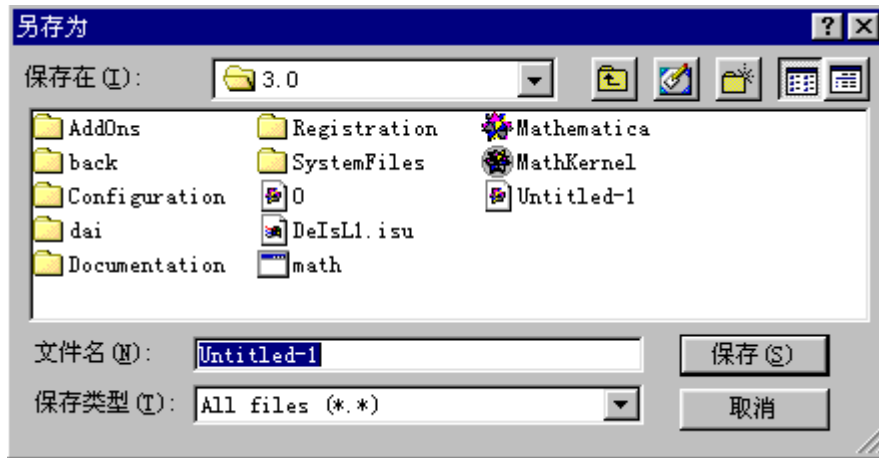
(1) 文件的存储

与大多数应用软件一样，在 Mathematica 中可以保存你所做的工作，包括你的输入、执行的结果、图象输出以及出错信息等。这样就便于你下次继续你的工作。

选"File"菜单，再选"Save"菜单项，出现如下画面：



这时弹出一个对话框，即告诉你当前文件的存储位置，在文件名处键入你所需保存结果文件的文件名（一般可选用一个便于记忆并与文件内容有关的文件名），回车后即完成文件的存储。



当然，你也可以选择一个已经存在的文件来保存你的结果，不过要注意，此时确定以后，原来文件的内容被完全覆盖。

保存的文件以.nb(是 notebook 的缩写)为后缀，是系统默认的 Mathematica3.0 语言程序文件。一般默认保存位置在你安装的 Mathematica3.0 系统所在目录，如：

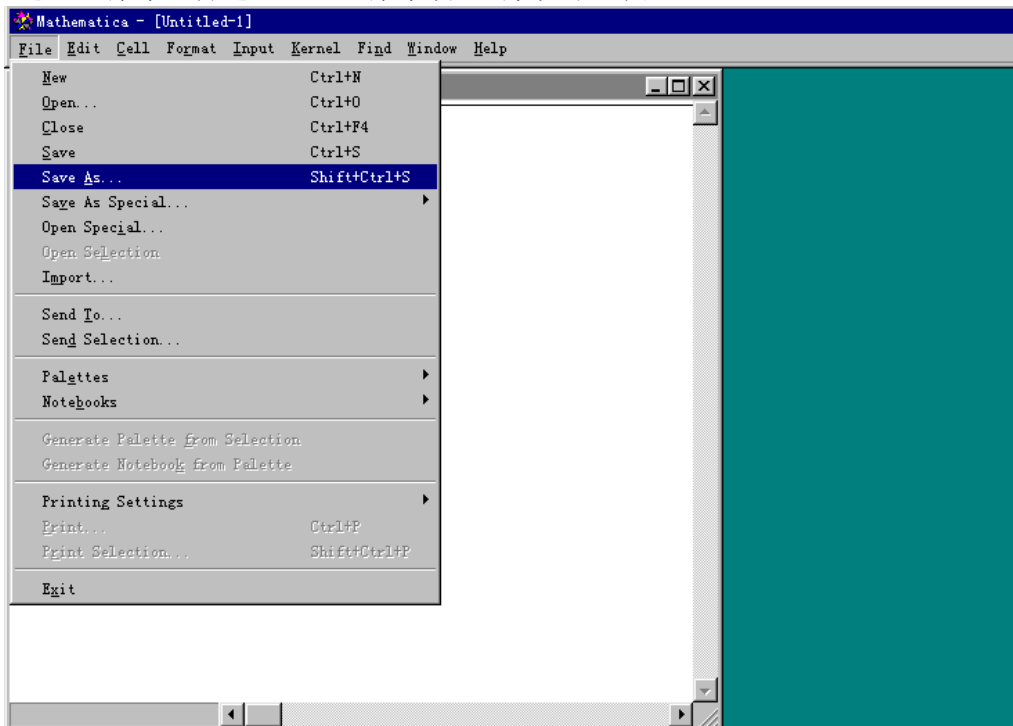
c:\Program Files\Wolfram Research\Mathematica\3.0\

如果你要将文件保存在其他特定工作目录下，请在保存文件时的弹出对话框中选定路径后再保存，下次打开时也必须至相应目录下取出该文件。

(2) 文件的另存

有时文件中有部分内容有变动，但又不想替换掉原文件，此时可选用 Mathematica 的另存功能。

选"File"菜单，再选"Save as"菜单项，出现如下画面：

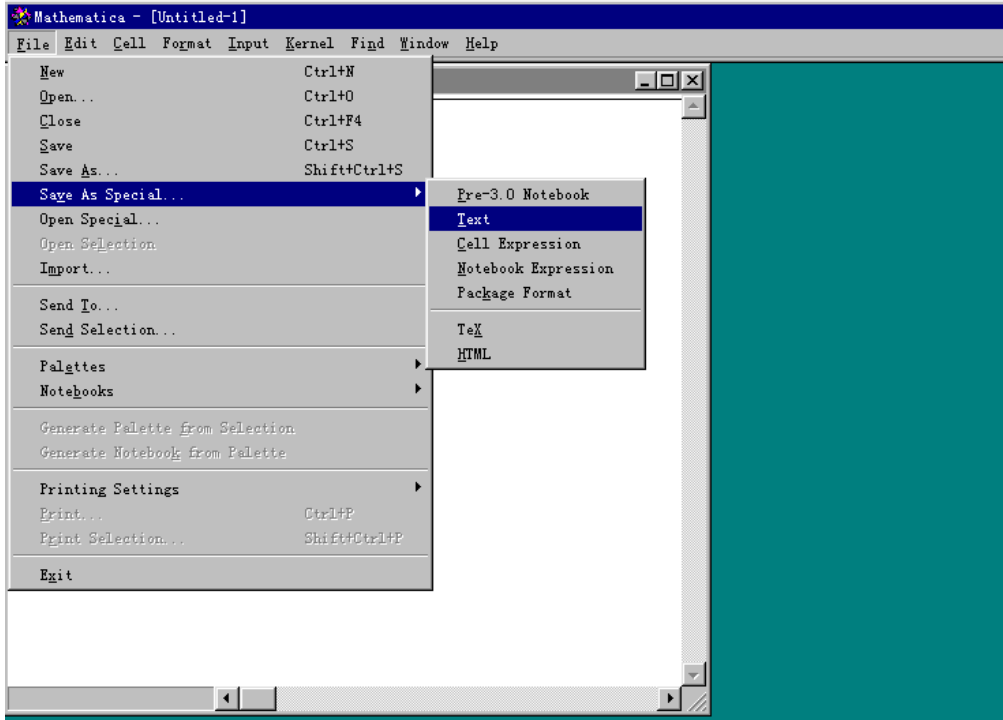


在弹出的对话框中键入一个新的文件名，回车后即完成一个新文件的保存。

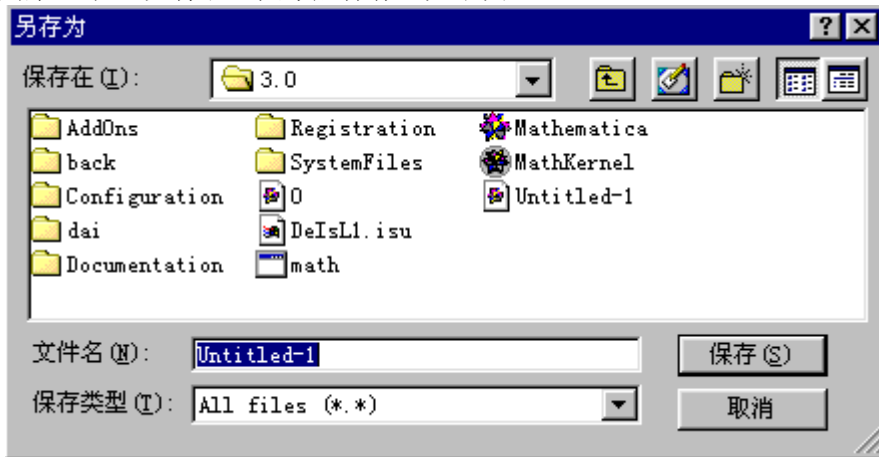
(3) 文件的特殊存储

对于以.nb 为后缀的 Mathematica 文件，在其他应用软件中不可以直接调用，这样的话 Mathematica 的兼容性就表现不出来了。不过，事实上你可以应用 Mathematica 的特殊存储功能将.nb 文件转存为其他格式的文件。Mathematica 系统提供了几种常用的文件格式：Text, TeX, Html 等。

选"File"菜单，再选"Save As Special"菜单项，出现如下画面：



在弹出的下级菜单项中选定你要保存 Mathematica 文件的特定格式，再在弹出的对话框中选定路径，键入文件名，回车后保存，见下图：

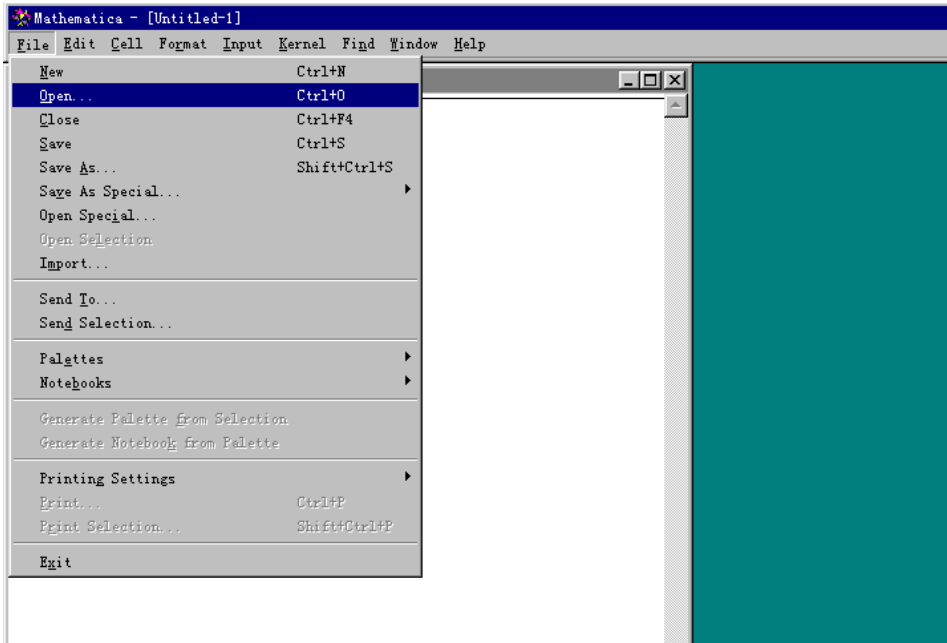


此时，你就可以在其他对应的应用软件中打开你刚才保存的特殊格式文件，进行操作。不过要注意的是，这个文件的后缀名仍然是.nb，若直接双击打开，还是处于 Mathematica 工作环境，必须在相应软件中用 Open 方式打开。

这种特殊存储功能大大方便了 Mathematica 用户更广泛的应用该软件所得到的结果，使你能在不同的应用软件下调用 Mathematica 文件。不过，它与"Save"及"Save As"功能不同的是，当你应用"Save As Special"功能时，Mathematica 会先执行整个文件，再将全部内容（包括结果）进行转存。

(4) 文件的打开

在你继续先前工作时，需打开上一次工作的结果。选"File"菜单，再选"Open"菜单项。在弹出的对话框中写好正确的路径和正确的文件，回车后即打开你需要的文件。注意对于用"Save As Special"得到的结果，最好用"File"菜单下的"Open Special"菜单项打开。

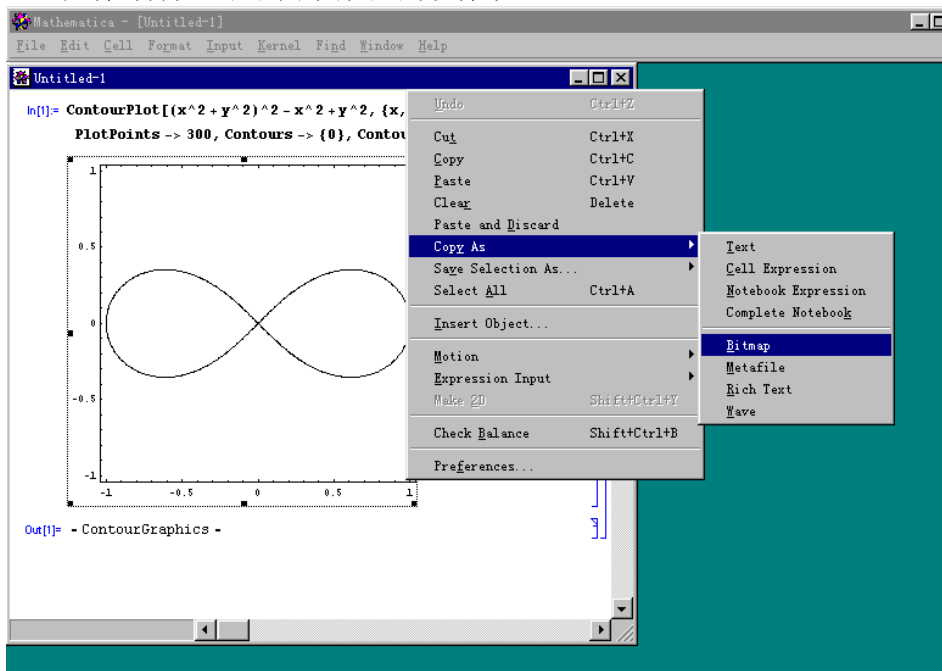


Mathematica 允许打开多个文件，但保存时仅保存最近使用过的一个文件。

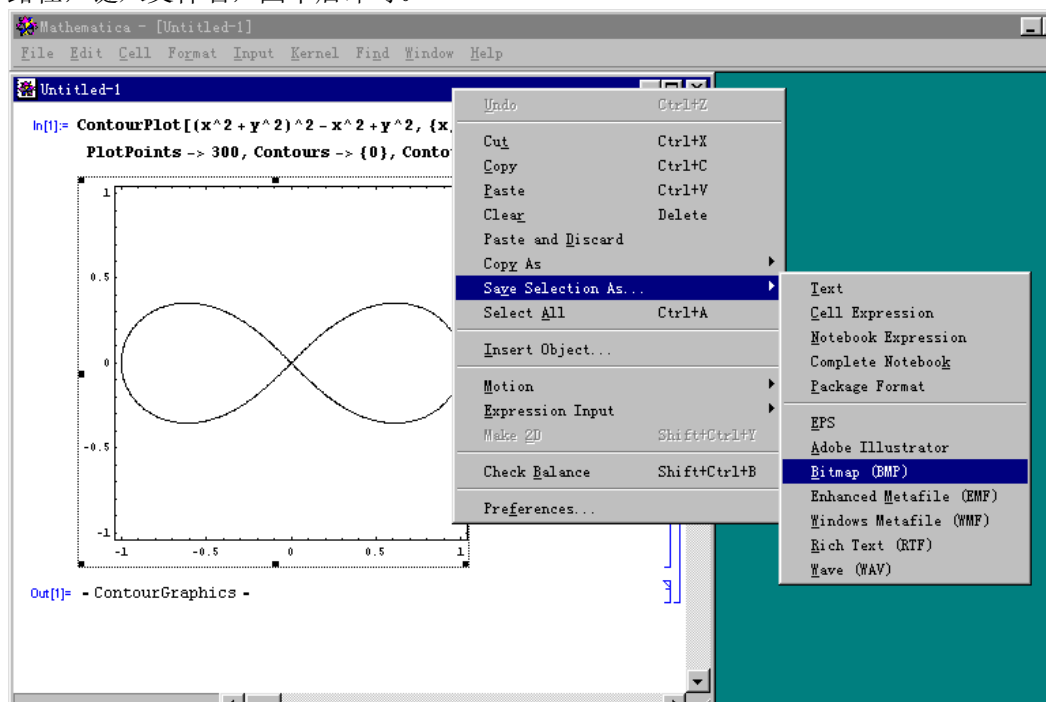
(5) 部分输入或输出的保存

对于一些中间结果，如图表等，你可能需要更方便地在其他应用软件中使用，Mathematica 也提供了这样的功能。

选中你所需的内容，在该内容上单击鼠标右键，在弹出菜单中选取"Copy As"菜单项，再在弹出的下级菜单中选中你所需的格式（文本或图形），按鼠标左键确定。这样就将选中内容以你所要求的格式置入剪贴板，之后你就可以在其他软件中用"Paste"或"粘贴"命令调用。注意，由于 Mathematica 文件格式的特定性，用"File"菜单下的"Copy"菜单项截下的 Mathematica 文件部分无法应用于其他应用软件中。



你还可以将这些中间结果直接保存为特殊格式文件，以便于你多次调用。选中内容后，在该内容上单击鼠标右键，在弹出菜单中选取"Save Selections As"菜单项，再在弹出的下级菜单中选中你需要的格式（文本或图形），按鼠标左键确定，然后在弹出的对话框中选好路径，键入文件名，回车后即可。



注意，与"File"菜单下的"Save As Special"菜单项的保存结果不同的是，此时生成的文件不是以.nb 为后缀的 Mathematica3.0 默认文件，而是系统默认的格式（文本或图形）文件，双击该文件可直接在应用软件中打开该文件。

(6) 设定 Mathematica 工作目录

Mathematica 具有一个默认当前工作目录，这在前文中介绍保存文件时已经提到，一般默认当前工作目录是 Mathematica 程序的所在位置，如：

c:\Program Files\Wolfram Research\Mathematica\3.0\

其实可以在任何时候，设定一个当前工作目录。在当前目录下可以直接指定文件，在其他目录下，则建立与当前目录的关系来指定文件。

用 Directory[] 命令给定当前工作目录，用 SetDirectory["ddd"] 命令将子目录 ddd 设置为当前工作目录，注意：ddd 不能是根目录，也不能是 Windows 的专用目录，如：c:\temp 等。用 ResetDirectory[] 命令可将当前工作目录退回到上一次设置的工作目录。

用 <<file.nb (或 file.m) 调入相应的文件，类似于 C 语言中 Include。

1.4 数和算术

现在开始我们来学习 Mathematica 的各种功能。你的计算机已经准备好了，它正在等待你的输入。你也通过上面的几页对 Mathematica 有了感性的了解。

Mathematica 的最基本功能之一是做数的运算，我们先来看看这方面的问题。

Mathematica 有几个常用的运算符：

运算符	+	-	*(或空格)	/	^	!
含义	加法	减法	乘法	除法	乘方	阶乘

下表列举了 Mathematica 有几个常用的常数、常量：

常量名	数学含义	解释	示例	结果
Pi	π	圆周率	Sin[Pi/2]	1
E	e	自然对数的底数	Log[E^10]	10
I	i	虚数单位	I^2	-1
Infinity	∞	无穷大	Limit[E^t,t->Infinity]	∞
Degree	$\frac{\pi}{180}$	度数	Cos[60 Degree]	$\frac{1}{2}$
GoldRatio	$\frac{1+\sqrt{5}}{2}$	黄金分割率	N[1/GoldRatio,4]	0.618

Mathematica3.0 常数、常量表

1.4.1 整数、有理数计算

如果你在 Mathematica 提示符状态下输入：

235+87

并按一下 Shift+Enter 键，就能得到 235 与 87 相加的和。235+87 是一个简单的表达式，表示求两个整数的和。从这儿也可以知道，一个整数是由一串数字构成的，要注意的是数字之间不能夹有空格。字符“+”表示求和的加法运算，在加号与两个加数之间有没有空格都不影响计算的结果。为了清楚起见，我们建议你各放一个空格。如果在两个整数之间留一个空格，就表示要计算它们相乘的积。例如：

235 87

可以得到它们的乘积，而不是 23587。放在两个计算对象之间的空格具有特定的意义，它总表示要相乘。当然也可以用“*”号表示乘法，即用 235*87 计算两数相乘。不过为清晰和简单的原因，很少用“*”号。

“-”符号表示减法：

87-235

得到一个负数。而：

87^2

计算 87 的平方，字符“^”表示乘方运算，按一般习惯，乘方符号两侧不留空格，但若留了空格也不影响计算结果。

用算术运算符把算术式进一步连接起来可以构成更复杂的算术式。对复杂算术式的计算顺序与通常数学里一样，先乘方，再乘除，再加减。对于加减乘除，同级运算都是自左向右做。

Mathematica 整数计算的最重要的特点是整数的表示范围没有限制，这也是它同其它各种数学软件的不同地方。例如：

235^87

可以得到一个二百多位的整数，它在你的屏幕上占了两行多。你可以立即算出传说中的印度国王应当付给他的发明国际象棋的大臣多少粒米：

2^64-1

系统会回答：

18446744073709551615

这个数“看起来并不太大”，仅仅只有 1 百亿亿。况且它比起下面的数字，实在很小：

1000!

你大概从来没有写过这么大的数，它有二千多位，结果写下来要一页多。这里的“!”符

号表示阶乘运算。

在 **Mathematica** 里整数是精确数。如果参加运算的都是整数，你永远可以得到精确的结果。但是从另一方面讲，越长越大的数计算起来越花时间。如果你要计算一万的阶乘，那就要等一段时间了。

两个整数相除，如不能整除就会得到有理数。**Mathematica** 把有理数表示为两个整数的商，例如：

285/123

再如：

1/200003+45/60000, 结果是：

$$\frac{604009}{800012000}$$

得到的结果是有理数。你会看到系统把有理数输出成我们写分数的样子，并且做了化简。这里的“/”特号表示除法运算。有理数也是精确数，你可以做出分子分母都非常大的有理数，例如：

285^27/123^13, 结果是：

$$\frac{1197388020571381427747405732554261354898594319820404052734375}{925103102315013629321}$$

有理数和整数经过加减乘除和整数次乘方的运算，得到的结果仍然是有理数或整数；这样计算的结果总是精确的。

1.4.2 浮点数(实数)和复数计算

Mathematica 的另一类数是浮点数，用于表示实数。当然，实际上一般讲浮点数只能表示实数的近似值。这里浮点数的特点是它们的有效位数不受限制。你可以写出有效位数是一百位一千位的浮点数。浮点数表示为包含一个小数点的数字串，它至少包含一个小数点和一个数字。浮点数的运算符与整数、有理数的运算符一样。两个浮点数运算或一个浮点数与一个整数(有理数)运算，得到的结果都是浮点数(近似数参与的运算得到的结果只能是近似的)。你可以试一下：

3.872^2+234

整数、有理数可以转换为浮点数，而且可以得到它们的任意位有效数字的浮点表示。例如要求一个有理数的 50 位精度的近似值：

N[237/4683, 50]

这里用大写的 **N** 表示取近似值，是 **Mathematica** 的系统函数。后面方括号中的两项用逗号分隔开，第一项是被取值的数或式子，第二项是要求的近似值位数；如果只想求近似值而不关心位数，可以写

N[237/4683]或 237/4683 //N

两种写法都可以，后一种方式称为 **Mathematica** 表达式的后缀形式。这时的近似值有十六位有效数字，一般也够用了。若不指定近似值位数，这样的数屏幕上将只显示出小数点后六位。使用这种由计算机确定位数的近似值的优点是计算速度快。试比较：

输 入	输 出
N[1/7]	0.142857
N[1/7, 20]	0.14285714285714285714
1/7 //N	0.142857
N[1.0/7, 20]	0.1428571428571428 (仅 16 位)
1.0...(共 22 个 0)...0/7	0.1428571428571428571429

Mathematica 系统里定义了许多常用数学常数, 如 e 、 π 等 (参见前面的 Mathematica3.0 常数、常量表; 另外还有 3 个数学常数: EulerGamma (欧拉常数), Catalan, Indeterminate 未列表中)。有趣的是它们都是精确数。这听起来有些奇怪, 但是回想一下, 我们在写数学式子时不是总用 π 这个符号表示圆周率吗?

它不是一个“精确”的“数”吗? 在 Mathematica 里 π 的名字是 Pi、 e 用大写 E 表示。你可以试一下:

```
2*Pi(或 2Pi)
```

```
E^2
```

系统回答是 2 Pi 和 E^2。好像什么也没有做, 但这不正是精确的结果吗? Mathematica 的数学常数(比如 Pi)和人们写在纸上的一个不同点是它是可以计算的对象。例如, 你可以求出它们或者包含它们的表达式的(近似)值来(用 N 来做):

```
N[E^2, 50]
```

```
N[2*Pi]
```

如果你想看看 Pi 的前一千位, 可以用以下命令:

```
N[Pi, 1000]
```

从以上的例子中我们可以看出 Mathematica 做计算的基本规则之一是: 如果能够得到精确结果, 就一定保证结果是精确的, 绝不随便丢掉信息、损失精度。实际上, Pi 中所包含的信息要比你刚刚得到的那个 3.14……多得多。

Mathematica 也支持复数的计算。虚数 i 用大写字母 I 表示, 下面是两个复数:

```
2+3I
```

```
3.24+6.17I
```

复数也可以分成精确的(实部、虚部都是整数或有理数)和近似的两类。请试着算:

```
I^I
```

```
I^I^I
```

等等, 再计算它们的数值。

1.4.3 常用数学函数

Mathematica 里定义了许多数学函数, 包括三角函数、指数对数函数、双曲函数和许多特殊函数, 还有一批整函数。这些函数都可以用在表达式里, 例如:

```
Sqrt[100]
```

```
Sin[1.34]
```

```
Log[E^10]
```

你也可以试一下: Sqrt[2], 看看会得到什么结果, 并解释一下其中的原因。

三角函数的名字在这里分别是 Sin、Cos、Tan、Cot、Sec、Csc、ArcSin、ArcCos、ArcTan、……。其他函数的名字可以从手册查到。这里应该注意几点: 函数名都是由字符串表示的, 字符之间不能有空格; 函数名字的第一个字母总是大写的, 后面的字母是小写的, 但如果名字是由几个段构成的(如 ArcSin), 则每段的第一个字母都必须大写, 这些是 Mathematica 内部函数取名的规则。再一点应当特别注意: 函数的参数表是用方括号括起来的, 不像我们在数学中那样用圆括号。这样做是为了避免产生歧义。(主要是为了区分 $f(x)$ 是 $f \cdot x$ 还是函数 $f(x)$)。

对复数有几个特别的函数, 分别是:

Abs[z](求绝对值)、Arg[z](幅角)、Re[z](实部)、Im[z](虚部)、Conjugate[z](z 的共轭)。

所有的实函数(如 Sin、Log 等)都可以用于复数。

其它常用函数有:

Exp[x]、Log[b, x]、Sign[x]、Floor[x]、Round[x]、Max[x,y,z,...]、Min[x,y,z,...]、!(阶乘)、

还可以写出多项式和有理式：

$$x^3+4x^2+5x-4$$
$$(x^2+4x-5)/(2x^2-2x)$$

你可以看到输出表达式的形式与输入的完全不同，项被重新排列了，且采用了数学公式的写法。在写表达式的时候要特别注意的是：写两个变量相乘或一个变量后面乘一个数时，前面变量的名字后面一定要留一个空格，否则系统就会把它们当作一个变量处理了。

任何时候都可以在输入表达式里写百分号“%”表示上一次计算的结果。这样你就可以方便地用前面的计算结果构造新的计算。下面是前后的两行输入，每行后面都要按 Shift+Enter 键，请观察它们的结果：

$$x^2+2x y+5 x y^2$$
$$\% (x^2+y)$$

还可以用%%表示倒数第二个计算结果。更一般的方法是在百分号后面写一个数，表示以这个数为编号的那一次计算的结果(回忆一下提示符的编号)。这样，前面已经得到的任何结果都可以很方便地用在后面的计算里了。

把一个代数式里的变量用某表达式替换，可以生成新的代数式。如果把一个代数式里所有的变量都用数值替换，得到的式子就可以计算出一个数值(原代数式在某点上的值)。例如(假设你刚刚算过的是上面的表达式，否则下式中的“%”应当带上相应的编号)：

$$\% /. \{x \rightarrow y+1\}$$
$$\% \% /. \{x \rightarrow 25, y \rightarrow 0.32\}$$

这第一个式子的结果是前面式子中所有的 x 用 y + 1 取代而得到的表达式，只有一个变量的话，花括号可以不写；后一个式子用两个数值做替换，结果是算出了一个数。这里，替换用花括号括起的几个替换式表示，每个替换式的左边是一个变量，右边是替换的表达式，它们之间用一个减号和一个大于符号连成的串隔开。替换和原表达式之间放上由一个除号和一个圆点符号组成的串。

需要时你还可以把数值或表达式赋给一个变量，赋值操作用“=”符号表示，它类似于我们一般讲的“定义为”。例如定义 a 表示π的二十位近似值的数应当写成：

$$a = N [\text{Pi} , 20]$$

类似地，可以写：

$$b=3.15x$$
$$c=x^2+3 x y+5 x y^2$$
$$a b c$$

这里又分别定义了 b 和 c 的值(是表达式)，最后求出 a、b、c 的乘积。

2.2 多项式计算

请你先定义两个多项式：

$$p1=x^3+5x^2-6$$
$$p2=x^4+3x^2y-4x^2y^2$$

你可以用它们做出新的多项式，例如：

$$p3=p1^2+p2^3$$
$$p4=p1^10$$
$$p5=(p1+p2)^4$$

如果你想展开一个多项式，那么用 Expand 操作：

$$\text{Expand}[p4]$$

这样你得到了一个很长的式子，反过来，你也可能想做因式分解，这时用 Factor。例如，

Factor[%]

比较一下得到的结果是不是与 p4 一样? 不一样! 是这样的, 因为 p1 本身可以因式分解为 $(x-1)(x^2+6x+6)$ 。

Expand 可以展开任意的多项式, Factor 的能力也相当强, 你可以试一下:

Factor[x^100-1]

Factor[x^3141-1]

设想一下, 如果要你自己做后一个因式分解, 你要花多长时间。我们也可以看出, 用计算机做这些事也不是不花代价的。以分解 $x^n - 1$ 为例, 随着 n 的增大, 计算的时间也在增加, 这也是计算机能力有限度的一个证据。

这里的 Factor、Expand 都是 Mathematica 的代数式操作函数, 这类函数还有很多。它们不是数学意义上的函数, 而是要求计算机做某种工作的命令, 可以用来作用到表达式上, 得到另一个表达式, 完成一种数学演算。在这个系统里有很多这样的“函数”。今后, 我们也不再区分数学意义上的函数和其它函数, 只要是写在方括号前面的表示某种意义的东西都将被称为函数。前一节你已经用过多次的 N 就是一个数值转换函数, 它把一个精确数值转换为一个近似数值。当然, 它还可以带一个表示结果精度的参数。

2.3 向量及矩阵

2.3.1 集合及其运算

集合是 Mathematica 中最灵活有效的元素。你会发现在 Mathematica 中, 集合是数学和计算机科学中一些标准的概念的总结推广。一般地, Mathematica 的集合所做的是为用户提供了一种收集各种表达式的方法。

这是一个数集:

In[1]:={2,3,4}

Out[1]={2,3,4}

这里给出了一个符号表达式的集合:

In[2]:=x^%-1

Out[2]={-1+x^2, -1+x^3, -1+x^4}

你可以对这些表达式进行求导运算:

In[3]:=D[% ,x]

Out[3]={2x, 3x^2, 4x^3}

x 被 3 代替后你可以求得值:

In[4]:=%/.x->3

Out[4]={6,27,108}

从这里可以看出, Mathematica 中大部分数学函数都可以独立作用于集合中的每一个元素。但不是所有的函数都是这样的。除非你规定新建立的函数 f 能够像对待单个的元素一样对待集合。你可以把集合用作根值表。例如, 你可以通过计算一个有序的具有不同参数的表达式来生成一个表。

在生成表的时候, Table 是一个很有用的函数。

这里给出了 i^2 值的表。i 从 1 变化到 6:

In[5]:=Table[i^2,{i,6}]

Out[5]={1,4,9,16,25,36}

你也可以制成一个公式表:

In[6]:=Table[x^i+2i,{i,5}]

Out[6]={2+x, 4+x^2, 6+x^3, 8+x^4, 10+x^5}

你可对从 Table 得到的集合进行其它的操作。TableForm 用表格的形式给出集合。注意 TableForm 的两个词都以大写字母开头。现在所有的例子都是一个参数变化形成的表。你也可以制成包含几个参数的表。

这里制成了一个 x^i+y^j 的表。i从1变化到3(外循环), j从1变化到2(内循环):

```
In[8]:=Table[x^i+y^j,{i,3},{j,2}]
Out[8]={{x+y,x+y^2},{x^2+y,x^2+y^2},{x^3+y,x^3+y^2}}
```

在这个例子中。这个表是集合的集合。外面的集合的元素与连续变量 i 相对应。里面的每个集合的元素 i 不变时, 和连续变量 j 相对应。

Table[f,{imax}]	给出一个 f 的 imax 值的集合
Table[f,{i,imax}]	给出一个 i 从 1 变化到 imax 的 f 值的集合
Table[f,{i,imin,imax}]	给出一个 i 从 imin 变化到 imax 的 f 值的集合
Table[f,{i,imin,imax,di}]	用 di 为步长
Table[f,{i,imin,imax},{j,jmin,jmax},...]	制成多维表格
TableForm[list]	用表格形式显示集合

表 2.3.1.1 Table 函数的用法

这里制成了一个 2×2 的表, 并定义它的名称为 m:

```
In[9]:=m=Table[i-j,{i,2},{j,2}]
Out[9]={{0,-1},{1,0}}
```

这是取出制成表格的集合中的集合的第一个分集合, 请注意取元素的符号为双重括号:

```
In[10]:=m[[1]]
Out[10]={0,-1}
```

这是取出第一个分表的第二个元素:

```
In[11]:=m[[1][2]]
Out[11]=-1
```

这是把两个操作结合在一起:

```
In[12]:=m[[1,2]] 或 m[[1]][[2]]
Out[12]=-1
```

这是用表格的形式列出 m:

```
In[13]=TableForm[m]
Out[13]//TableForm=


|   |    |
|---|----|
| 0 | -1 |
| 1 | 0  |


    (或: Out[13]// TraditionalForm=


|   |    |
|---|----|
| 0 | -1 |
| 1 | 0  |

)
```

t[[i]] 或 Part[t,i]	给出 t 的第 i 个分集合
t[[{i ₁ ,i ₂ ,...}]] 或 Part[t,{i ₁ ,i ₂ ,...}]	给出 t 的第 i ₁ ,i ₂ ,... 个部分组成的集合
t[[i,j,...]] 或 Part[t,i,j,...]	给出 t 的与 t[[i]][[j]]... 相对应的部分

表 2.3.1.2 取集合元素的操作

集合的集合就可以看成是二维数组。当你用表的形式写出他们时, 两项标志就像 x,y 坐标一样。你可以用 Table 来生成任意维数的数组。

2.3.2 向量和矩阵

Mathematica 用集合来表示向量和矩阵, 或者更一般的张量。向量用集合表示, 矩阵用集合的集合 (当然还要求每个子集一样长)。在这里应当特别指出, 不仅可以有数值的向量、矩阵, 也可以写出一般表达式的向量和矩阵。在这种表示下, 向量矩阵的加减和数乘运算可以直接用一般的运算符实现。在 Mathematica 里, 两个表相加减, 如果它们的元素个数相同, 则处理为它们的元素分别相加减。如此推论, 两个矩阵相加减也可以直接用加减运算符做。不同大小的向量、矩阵之间不能运算。Mathematica 定义了许多向量、矩阵的运算。

{a,b,c} 向量(a,b,c)

$$\{\{a,b\},\{c,d\}\} \text{矩阵} \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

这是一个 2×2 矩阵:

In[14]:=m={{a,b},{c,d}}

Out[14]={{a,b},{c,d}}

这是第一行:

In[15]:=m[[1]]

Out[15]={a,b}

这是元素 m_{12}

In[16]:=m[[1,2]]

Out[16]=b

这是一个二元向量:

In[17]:=v={x,y}

Out[17]={x,y}

Table[f,{l,n}]	用计算 f 的值建立 n 维向量 $i=1,i=2,\dots,i=n$
Array[a,n]	建立一个 $(a[1],a[2],\dots)$ 形式的 n 维向量
Range[n]	产生一个 $(1,2,3,\dots,n)$ 集合
Range[n ₁ ,n ₂]	产生一个 (n_1,n_1+1,\dots,n_2) 集合
Range[n ₁ ,n ₂ ,dn]	产生一个 (n_1,n_1+dn,\dots,n_2) 集合
list[[i]] or Part[list,i]	给出向量集合的第 i 个元素
Length[list]	给出集合中元素个数
ColumnForm[list]	在一列中写出 list 的元素

表 2.3.2.1 向量的生成及操作

p 和 q 被认为是标量:

In[18]:=v=p*v+q (或 v=p v+q)

Out[18]={q+p x,q+p y}

把分向量加入到向量的分向量中去:

In[19]:=v+{x p,y p}+{x p p,y p p}

Out[19]={x+x p+x p p,y+y p+y p p}

也可以用一个向量去乘一个矩阵。矩阵、向量乘法用圆点(西文句点)表示:

In[20]:=m.v

Out[20]={a x+b y,c x+d y}

矩阵乘矩阵、矩阵乘向量也可以。

因为 Mathematica 用集合来表示向量和矩阵, 你不必分清行向量和列向量。

Table[f,{i,m},{j,n}]	用计算 f 的值建立 $m \times n$ 的矩阵, i 从 1 到 m, j 从 1 到 n
Array[a,{m,n}]	建立一个 $m \times n$ 的矩阵, 第 i 行, j 列的元素是 $a[i,j]$
IdentityMatrix[n]	建立一个 $n \times n$ 的单位方阵
DiagonaMatrix[list]	建立一个对角方阵, 把 list 的元素列于对角线上
list[[i]] or Part[list,i]	给出矩阵 list 的第 i 行
list[[i,j]] or Part[list,i,j]	给出矩阵 list 的第 i 行 j 列
Dimensions[list]	给出矩阵 list 的维数
MatrixForm[list]	用矩阵的形式写出 list

表 2.3.2.2 矩阵的生成及操作

这里形成了一个 3×3 的 s 矩阵, 它的元素是 $s_{ij}=i+j$:

```
In[21]:=s=Table[i+j,{i,3},{j,3}]
Out[21]={{2,3,4},{3,4,5},{4,5,6}}
```

这是用标准的二阶矩阵的形式给出 s:

```
In[22]:=MatrixForm[s]
Out[22]//MatrixForm=
```

$$\begin{pmatrix} 2 & 3 & 3 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \end{pmatrix}$$

这是用元素符号给出的向量，你可以在选用任何分向量进行公式推导的时候使用这个方法:

```
In[23]:=Array[a,4]
Out[23]={a[1],a[2],a[3],a[4]}
```

这里给出了一个由符号元素组成的 3×2 矩阵:

```
In[24]:=Array[p,{3,2}]
Out[24]={{p[1,1],p[1,2]},{p[2,1],p[2,2]},{p[3,1],p[3,2]}}
```

这是前面一行矩阵的维数:

```
In[25]:=Dimensions[%]
Out[25]={3,2}
```

这是一个 3×3 的对角矩阵:

```
In[26]:=DiagonalMatrix[{a,b,c}]
Out[26]={{a,0,0},{0,b,0},{0,0,c}}
```

这里给出上面所定义的符号变量组成的 2×2 矩阵的行列式:

```
In[27]:=Det[m]
Out[27]=-(b c)+a d
```

这是 m 的逆矩阵:

```
In[28]:=Inverse[m]
Out[28]={{  $\frac{d}{-(bc)+ad}$ ,  $-\left(\frac{b}{-(bc)+ad}\right)$ },  $-\left(\frac{c}{-(bc)+ad}\right)$ ,  $\frac{a}{-(bc)+ad}$ }}
```

C m	用向量来乘
a.b	矩阵乘法
Inverse[m]	逆矩阵
MatrixPower[m,n]	矩阵的 n 次幂
Det[m]	行列式
Minors[m,k]	m 的 k×k 阶子矩阵
Transpose[m]	转换(将矩阵的行列交换)
Eigenvalues[m] / Eigenvectors[m]	特征值 / 特征向量
Eigenvalues[N[m]] / Eigenvectors[N[m]]	数字的特征值 / 特征向量
Eigensystem[m]	特征值和对应的特征向量

表 2.3.2.3 矩阵运算

把矩阵和它的逆矩阵相乘得到一个单位阵:

```
In[29]:=%.m
Out[29]={{1,0},{0,1}}
```

生成一个 3×3 矩阵:

```
In[30]:=m=Array[a,{3,3}]
Out[30]= {{a[1, 1], a[1, 2], a[1, 3]},
          {a[2, 1], a[2, 2], a[2, 3]},
          {a[3, 1], a[3, 2], a[3, 3]}}
```

求 m 的行列式:

```
In[31]:=Det[m]
Out[31]=-(a[1,3]a[2,2]a[3,1])+ a[1, 2] a[2, 3] a[3, 1] +
          a[1, 3] a[2, 1] a[3, 2] -a[1, 1] a[2, 3] a[3, 2] -
          a[1, 2] a[2, 1] a[3, 3] +a[1, 1] a[2, 2] a[3, 3]
```

第三节 基本微积分

微积分里的求导函数、求不定积分、级数展开等都是最典型的符号演算。Mathematica 系统提供了一批做这些演算的函数。

3.1 极限、微商

求极限的函数是 `Limit`。先来看一个简单的例子，求一个典型的极限:

```
Limit[Sin[x]/x,x->0]
```

从这里你可以知道, `Limit` 有两个参数, 第一个是被求极限的表达式, 上例中是 `Sin[x]/x`; 第二个说明变量和变量趋向的值; 你可以用 `Plot` 看一看函数的图形, 这有利于理解问题:

```
Plot[Sin[x]/x, {x,-4,4}]
```

可以去求变量趋于无穷大的极限, 例如:

```
Limit[Sqrt[x^2+3x]-x, x->Infinity]
```

也可以通过作图观察一下被求极限的函数表达式在 x 趋向无穷大时的变化情况和趋势:

```
Plot[Sqrt[x^2+3x]-x, {x,0,100}]
```

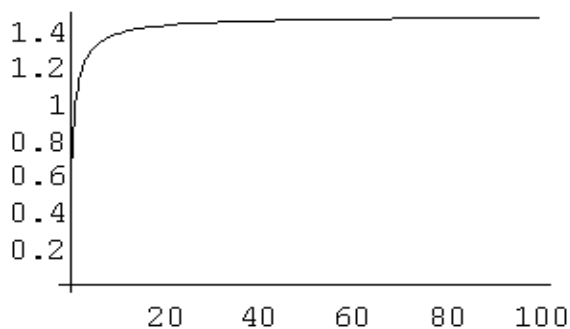


图 3.1.1 `Plot[Sqrt[x^2+3x]-x, {x,0,100}]`

如果愿意, 你可以在更大的范围作这个函数的图。

我们知道不是所有的函数在定点都有确定的极限, 比如, 函数 $\sin(1/x)$ 在 $x=0$ 附近无限振荡, 故在该处无确定极限。尽管如此, 就实数 x 而言, $x=0$ 处的函数值介于 -1 到 1 之间, `Limit` 用 `Interval` 指定边界变化值。通常情况下, `Interval[{xmin,xmax}]` 生成一个介于 $xmin$ 到 $xmax$ 之间的不定值。

`Limit` 返回一个 `ReallInterval` 目标, 指出 $\sin\left[\frac{1}{x}\right]$ 在其奇点 $x=0$ 可能值的范围:

```
In[ ]:=Limit[Sin[1/x],x->0]
```

```
Out[ ]=Interval[{-1,1}]
```

某些函数在一点处的极限随逼近方向不同而不同，可用 **Limit** 的方向选择 **Direction** 指定你想要的方向。

下面给出求 $\tan(x)$ 在 $x = \frac{\pi}{2}$ 不同逼近方向得到的值：

```
In[ ]:=Limit[Tan[x],x->Pi/2,Direction->1]
```

```
Out[ ]=∞
```

```
In[ ]:=Limit[Tan[x],x->Pi/2,Direction->-1]
```

```
Out[ ]=-∞
```

我们从下图中可以看得很清楚。

```
In[ ]:=Plot[Tan[x],{x,1,2}]
```

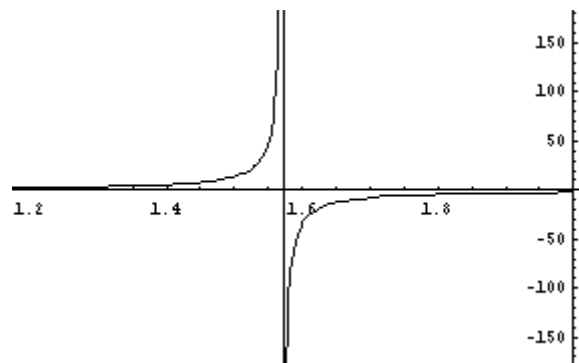


图 3.1.2 Plot[Tan[x],{x,1,2}]

要注意的是，这里的 **Direction->1** 表示 x 沿下方向逼近 x_0 时的极限值，即左极限；而 **Direction->-1** 表示 x 沿上方向逼近 x_0 时的极限值，即右极限。

如果给 **Limit[]** 一个没有定义的函数如 $f[x]$ ，在多数情况下 **Mathematica** 将原来的命令行原样打印，没有具体的运行结果。例如：

```
In[ ]:=Limit[f[x],x->0]
```

```
Out[ ]=Limit[f[x],x->0]
```

Mathematica 可以求出任意函数表达式的微商(函数的导函数)，例如：

```
D[x^3+4x^2-4,x]
```

D 是求导函数的操作。被求导的表达式里可以有其他变量或者符号参数，**D** 总认为它们是与参与求导操作的自变量无关。所以，实际上 **D** 求的是偏导函数是 $\frac{\partial}{\partial x} f$ 。看下面的例子：

```
D[Sin[a*x]Cos[x*y],x]
```

D 还可以用于求多元函数的高阶偏导数：

```
D[Sin[a*x]Cos[x*y],x,y]
```

这个式子表示先对 y 求偏导数，得到的结果再对 x 求偏导数。系统不限制变量出现的次数，同一个变量也可以多次出现。如果是要对一个变量求高阶偏导数，还有一种简洁的写法：

```
D[Sin[x]/(x+Cos[2x]),{x,3}]
```

这表示对 x 求三阶导数。请你把上面式子里的 **3** 换成 **10**，结果会得到一个很长的式子，有几十项，每项都挺复杂。试想一下如果要你自己用笔和纸做这件事，那将是多么费时费力

的事！如果需要，**Mathematica** 也能帮你求出上述函数的 100 阶导数。当然，结果将是非常非常的长。你还必须有足够的耐心，即使是计算机，做这件事也需要相当长的时间。

对于某些特殊函数（如绝对值函数），它在某一点的函数导数不存在，但是在该点的左、右导数存在。此时我们可以应用其数学定义来求出这类函数在特定点的左、右导数。

$$\text{由于 } f'_+(x_0) = \lim_{\substack{x \rightarrow x_0 \\ x > x_0}} \frac{f(x) - f(x_0)}{x - x_0},$$

$$f'_-(x_0) = \lim_{\substack{x \rightarrow x_0 \\ x < x_0}} \frac{f(x) - f(x_0)}{x - x_0}$$

因此我们应用前面介绍的求左、右极限的命令来求左、右导数。

我们以绝对值函数为例：

对绝对值函数求导，仅有符号函数结果。

```
In[ ]:=D[Abs[x],x]
```

```
Out[ ]=Abs'[x]
```

因此，只能应用其定义来求值：

```
In[ ]:=Limit[(Abs[x]-Abs[0])/x,x->0,Direction->1]
```

```
Out[ ]=-1
```

```
In[ ]:=Limit[(Abs[x]-Abs[0])/x,x->0,Direction->-1]
```

```
Out[ ]=1
```

上面分别计算出了绝对值函数在 $x=0$ 处的左、右导数为 -1 和 1。这里要注意的是，和计算左右极限时一样，**Direction->1** 计算出的是左导数，而 **Direction->-1** 计算出的是右导数。

3.2 不定积分和定积分

求积分的函数是 **Integrate**。我们先用它来求一个简单的不定积分：

```
Integrate[x^2+3x+5,x]
```

这个式子的意思很清楚。我们知道原函数是一个函数族，而这里计算给出的只是其中的一个。让我们再来试一个复杂一点的例子：

```
Integrate[(x+1)/(x^2+3x+5),x]
```

怎样验证求出的结果是不是正确呢？你可以用求导函数的 **D** 作用一下，看看得到的是不是原来那个函数。可这次不是，挺复杂的，**Mathematica** 有一个做表达式化简的函数叫 **Simplify**，用它作用一下，常可以得到一个简单些的等价的表达式。这里用 **Simplify** 你就可以得到原来的被积函数了。但要注意，对于复杂的表达式，使用 **Simplify** 常常要花费相当长的时间。

让我们看看被积函数与原函数、导函数的图形：

```
f1=(x+1)/(x^2+3x+5)
```

```
f2=Integrate[f1,x]
```

```
f3=D[f1,x]
```

```
Plot[{f1,f2,f3},{x,-1,1}]
```

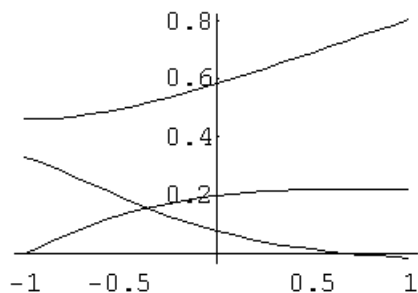


图 3.2.1 Plot[{f1,f2,f3},{x,-1,1}]

你先仔细看看这张图。使用你关于微积分的知识，你能分辨出哪一条曲线属于哪个函数吗？上面我们是先定义了三个函数，然后在一张图里画出了它们的图形。实际上 `Plot` 可以同时画出多个函数的图形，只要你把这些函数写在花括号里，用逗号分隔。例如：

```
Plot[{Sin[x],Cos[x],Tan[x]},{x,-Pi,Pi}]
```

`Integrate` 的积分能力相当强，它可以积出许多不定积分。但是也有许多它积不出来的积分。首先是那些已经证明无法用有限形式表示原函数的积分，例如：

```
Integrate[Sin[Sin[x]],x]
```

那么我们能看看原函数的情况呢？能不能画一下它的图形？如果你紧接着输入：

```
Plot[%,{x,-3,3}]
```

你会看到屏幕上显示出许多错误信息，最后在图上什么也没画出来。究其原因，`Plot` 在作图时先用一系列的自变量取值去计算被画图函数的值。想一想，如果用 $x=-3$ 去算 `Integrate[Sin[Sin[x]],x]` 会得到什么呢？得到的 `Integrate[Sin[Sin[-3]],-3]` 又是什么意思呢？那么我们就没有办法了吗？当然不是。但我们先把这个问题放一下，留待一会再讨论，先来看看定积分。

定积分也是用 `Integrate` 做，只是要给出积分的上下限。例如：

```
Integrate[x^2Sin[x],{x,0,2}]
```

`Integrate` 做定积分的方式与人一样，先求出原函数，然后再用上下限代入。因此，这里的上下限可以用任意的代数表达式，原函数中也可以带有其他参数，用 `Integrate` 还可以做多重积分，例如：

```
Integrate[x^2+y^2,{x,0,a},{y,0,x}]
```

这个表达式里使用了符号参数描述积分区域的边界。对于求不出原函数的积分，可以用数值积分的方法求它在某区间定积分的近似值。在 `Mathematica` 里求数值积分的函数是 `NIntegrate`，举个例子：

```
NIntegrate[Sin[Sin[x]},{x,0,Pi}]
```

如果我们想到可以用变上限积分表示原函数，那么我们就知道如何画原函数的图形了。`Plot` 在作图时总是要对被画图的函数表达式在一些点上求值，而一旦取定了一个点，这里要求值的就是一个定积分。你可以用数值积分作为它的近似来作图：

```
Plot[NIntegrate[Sin[Sin[x]},{x,0,y}},{y,-3,3}]
```

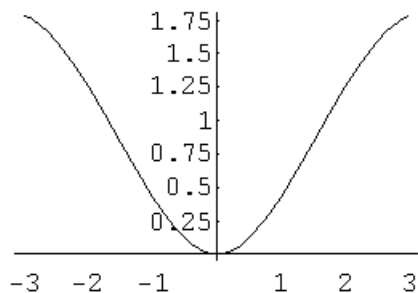


图 3.2.2 Plot[NIntegrate[Sin[Sin[x]],{x,0,y}],{y,-3,3}]

这样画图并不很麻烦。

关于 **Mathematica** 的积分我们还要注意一点：还有一些积分它也做不出来。例如对于分母的次数高于 5 次的整系数有理分式，如果 **Mathematica** 不能做出它的因式分解，就求不出它的积分。还有些人可以求出积分的函数，而 **Mathematica** 也求不出来。但是，这里应该强调的是：即使不能求出原函数(仍然用包含 **Integrate** 的形式给出结果)，**Mathematica** 仍然支持我们对这个“原函数”做操作，例如，你可以对它求导，与其他公式做运算，等等。下节你会看到，还可以把这个原函数展开成幂级数。可以说，求不出有限形式的原函数并不妨碍你使用或者研究这个“原函数”。

NIntegrate 函数的一个重要特性是能处理在已知点被“放大”的函数，**NIntegrate** 在积分区的端点自动检查这类错误。

函数 $\frac{1}{\sqrt{x}}$ 在 $x=0$ 点处被放大，但 **NIntegrate** 函数仍能得到积分的正确结果：

```
In[ ]:=NIntegrate[1/Sqrt[x],{x,0,1}]
```

```
Out[ ]:=2
```

当然 **Mathematica** 也可给出这个函数的精确积分：

```
In[ ]:=Integrate[1/Sqrt[x],{x,0,1}]
```

```
Out[ ]:=2
```

对函数 $\sin(\sin(x))$ ，**Mathematica** 仅能给出数字值：

```
In[ ]:=NIntegrate[Sin[Sin[x]],{x,0,1}]
```

```
Out[ ]:=0.430606
```

我们可以得到 $\sin(1/x)$ 在 $[0.5,1]$ 上的积分：

```
In[ ]:=NIntegrate[Sin[1/x],{x,0.5,1}]
```

```
Out[ ]:=0.472399
```

但 **Mathematica** 能用函数 **NIntegrate** 检验出 $\sin(1/x)$ 在 $x=0$ 处不可积：

```
In[ ]:=NIntegrate[Sin[1/x],{x,0,1}]
```

```
NIntegrate::slwcon:
```

```
Numerical integration converging too slowly;
suspect one of the following:singularity,
value of the integration being 0, oscillatory
integrand, or insufficient Working Precision.
If your integrand is oscillatory try using
the option Method->Oscillatory in NIntegrate.
```

```
NIntegrate::ncvb
```

```
NIntegrate failed to converge to prescribed
accuracy after 7 recursive bisections
```

in x near x=0.0035126778890767333`

Out[]=0.504894

有几种方法可控制 NIntegrate 的操作，首先，你可能想得到你希望的结果的精度。如果你用 Integrate 函数，然后用 N[]去得到结果，同时用 N[]的第二个变量定义数值积分的计算精度。你也可用 WorkingPrecision->n 来指定 NIntegrate 的计算精度。

计算 sin(1/x)在[1,2]上的积分，计算精度为 30 位：

In[]:=NIntegrate[Sin[1/x],{x,1,2},WorkingPrecision->30]

Out[]=0.6325680941080905298925

除 WorkingPrecision 外，在 Mathematica 中，函数 NIntegrate 还有很多这样的选项，参见下表。

选项	缺省值	意义
AccuracyGoal	Infinity	尽量得到最终答案的准确度
PrecisionGoal	Automatic	尽量得到最终答案的精确度
MinRecursion	0	再分积分区的最小递归数
MaxRecursion	6	再分积分区的最大递归数
SingularityDepth	4	在端点变量变换前使用的递归再分数
MaxPoints	Automatic	被积函数最大样本数
WorkingPrecision	\$MachinePrecision	内部计算中使用的小数位数
Compiled	TRUE	积分式是否被编译

我们要注意 WorkingPrecision 仅能指定 NIntegrate 函数内部的计算精度。除此外，我们还可以设置 PrecisionGoal 和 AccuracyGoal 得到 NIntegrate 函数的指定精度。

当用 NIntegrate 函数计算一数值积分时，在一系列点上取积分值样本点，如发现在某一区域积分式变化迅速，则在该区域增加取样点，参数 MinRecursion 和 MaxRecursion 指定递归再分的最小和最大数目。增加 MinRecursion 值，NIntegrate 能够使用的样本点数目增多，MaxRecursion 限制了 NIntegrate 能够使用的样本点数目。增加 MinRecursion 或 MaxRecursion 都使 NIntegrate 运行速度减慢。若积分式在某一点被放大，则变量替换前，SingularityDepth 指定 NIntegrate 应进行的递归再分级数。

参数 MinRecursion 与 MaxRecursion 都在缺省情况下，NIntegrate 忽略 exp(-x^2)在 x=0 处的峰值而给出错误的积分结果：

In[]:=NIntegrate[Exp[-x^2],{x,-1000,1000}]

NIntegrate ::ploss:

Numerical integration stopping due to loss of precision. Achieved neither the requested PrecisionGoal nor AccuracyGoal; suspect one of the following: highly oscillatory integrand or the true value of the integral is 0. If your integrand is oscillatory try using the option Method->Oscillatory in NIntegrate.

Out[]=1.34946 × 10⁻²⁶

在选择 MinRecursion->3 下，NIntegrate 抽取足够点以计算 x=0 处的峰值。但在系统默认的 MaxRecursion 值下，NIntegrate 不能使用足够样点而得到准确答案：

In[]:=NIntegrate[Exp[-x^2],{x,-1000,1000},MinRecursion->3]

NIntegrate::ncvb: NIntegrate failed to converge to prescribed accuracy after 7 recursive bisections in x near x=7.8125`

```
Out[ ]=0.99187
```

如也对 `MaxRecursion` 进行设置, 则 `NIntegrate` 可得到积分的正确答案:

```
In[ ]:=NIntegrate[Exp[-x^2],{x,-1000,1000},MinRecursion->3,MaxRecursion->10]
```

```
Out[ ]=1.77245
```

还有另一种解决问题的方法: 用 `NIntegrate` 把积分区分成几个小块, 其中一个小块覆盖峰值区:

```
In[ ]:=NIntegrate[Exp[-x^2],{x,-1000,10,10,1000}]
```

```
Out[ ]=1.77245
```

对于多维函数的积分, `NIntegrate` 可能要花费较长的时间去得到正确的答案。但是, 通过设定选项 `MaxPoints`, 你可得到一个粗略的估计值, 被积函数的取样次数仅有有限值。

给出三维球平面的近似积分值:

```
In[ ]:=NIntegrate[If[x^2+y^2+z^2<1,1,0],{x,-1,1},{y,-1,1},{z,-1,1},MaxPoints->10000]
```

```
Out[ ]=4.18106
```

下面给出精确值:

```
n[ ]:=N[4/3 Pi]
```

```
Out[ ]=4.18879
```

3.3 幂级数展开

一个函数描述了在某个区域内值的对应关系, 这种关系曲线描述有时可能很复杂, 有时必须借助于积分等形式而无法用初等函数的有限形式表示。当我们只需要考察一个函数在某一点附近的性质时, 我们可以用一个有限次的多项式作为这个函数的近似, 这就是幂级数展开(Taylor 展开)的意义。使用 `Mathematica`, 你可以非常方便地求出任一个复杂函数表达式的任意阶幂级数展开。先试一个简单的例子, 对大家熟悉的 `Sin[x]` 在 0 点展开 10 阶:

```
s=Series[Sin[x],{x,0,10}]
```

这样得到的不是一个一般的多项式, 你能看到它有一个形如 $O[x]^{11}$ 的余项, 这样的式子不能求出在某一点的值, 也无法作图。如果想做这些, 必须先去除展式的余项, 把它转变为一个正规的多项式(原来展式的近似多项式), 这用函数 `Normal` 实现:

```
f1=Normal[s]
```

你看见过 `Sin[x]` 的 10 阶展式的图形吗? 请看:

```
Plot[f1,{x,-5,5}]
```

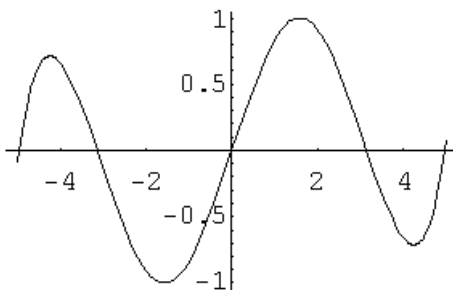


图 3.3.1 Plot[f1,{x,-5,5}]

得到的这个多项式真是 `Sin[x]` 在某种意义下的近似吗? 在什么样的范围内它可以作为 `Sin[x]` 的替代物? 在一般的数学书上有 `Sin[x]` 与它的 10 阶、20 阶、...Taylor 展式的比较吗? 你可以立刻把它们做出来, 这是第一个:

```
Plot[{f1,Sin[x]},{x,-5,5}]
```

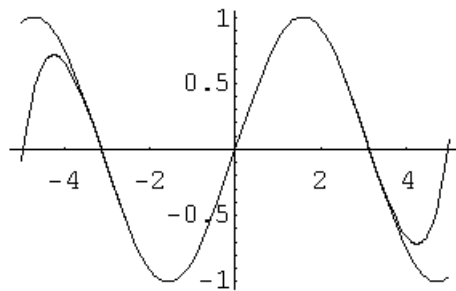


图 3.3.2 Plot[(f1, Sin[x]), {x, -5, 5}]

你还可以画出两个函数的差:

`Plot[f1-Sin[x], {x, -5, 5}]`

选取你所关心的区域去画两个函数的曲线或者它们的差的曲线。你能很快确定在哪个范围里两个函数的差的最大值小于 10。试用 `Sin[x]` 的更高阶的展式做上面同样的事情。根据你的观察, 能认为阶数高的展式对于原函数的逼近程度优于阶数低的展式吗? 如果想在区间 `[-10, 10]` 代替 `Sin[x]` (自己制定一个标准), 那么应该用多少项的展式呢? 用其他复杂些的表达式试一试 `Series` 的能力。

`Series` 也可以展开含有多个变量的表达式, 它自动地认为其他变量是与它正处理的变量无关的。因此, 你可以用它生成某些特殊的序列, 这只要展开对应的生成函数就可以了。`Series` 还可以做多元函数的级数展开, 例如:

`Series[Sin[x*y], {x, 0, 3}, {y, 0, 3}]`

结果为:

$$(y + o[y]^4)x + \left(\frac{-y^3}{6} + o[y]^4\right)x^3 + o[x]^4$$

第四节 拟合、自定义函数与图形

4.1 数值计算与函数拟合

4.1.1 数值计算

`Mathematica` 系统也有一批数值计算函数, 如前面讲过的 `N`、`NIntegrate` 等。再比如求函数极小值的 `FindMinimum`:

`FindMinimum[2Cos[x]+Sin[x], {x, 1}]`

这个表达式表示从初始点 1 出发寻找函数的一个极小值。`FindMinimum` 还可以求多变量函数的极小值:

`FindMinimum[2Cos[x]+Sin[y], {x, 1}, {y, 1}]`

`NSum` 和 `NProduct` 用于求序列的和、积的数值近似, 它们的使用形式与 `Sum`、`Product` 一样, 但同时可用于求无穷级数的近似值:

`NSum[1/n^2, {n, 1, Infinity}]`

4.1.2 函数拟合

在做数据处理时人们常想的一件事是希望用一个函数去反映客观的数据, 这叫函数拟合。做这种拟合一般是企图发现数列中的某种规律性, 找出这种规律性的一种表达式表示。

或者为了从实测数据得到有关数学模型的参数。然后就可以用这些表示或模型预测可能的结果。在 Mathematica 中，最基本的拟合操作是 Fit。为了试一试它的功能，我们先准备一组数据(也可以用现场实测的数据)：

```
data=Table[{x,Cos[x]},{x,-2.0,2.0,0.5}]
```

现在有了 Cos 在几个点的坐标值，而所得的形式也正是 Fit 函数所要求的。现在试试用多项式来拟合这几个点。由于 Cos 是偶函数，可以考虑用含有五个系数的多项式。根据这些考虑，我们可以写出：

```
f=Fit[data,{1,x^2,x^4,x^6,x^8},x]
```

拟合出的多项式为：

$$1.0 - 0.499999x^2 + 0.0416633x^4 - 0.00138442x^6 + 0.0000228146x^8$$

你可以看一看你做出的东西与 Cos[x] 比较起来怎样。命令为：Plot[{Cos[x],f},{x,-5,5}]

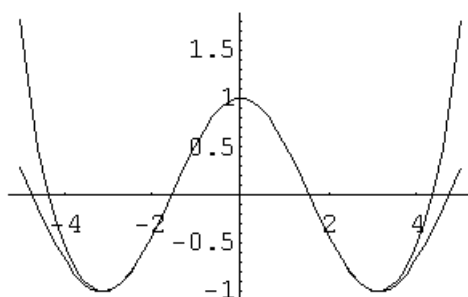


图 4.1.2.1 Plot[{f,Cos[x]},{x,-5,5}]

你还可以在更小的范围里比较两个函数。你可以想一下以下的问题：

1. 使用次数更高的多项式做拟合试验，结果更好吗？如果用的全是 x 的奇次数的项，情况会怎样？(如果 x 的项很多，可以考虑用 Table 生成函数表。)
2. 将上面得到的 f 与 Cos[x] 在 x=0 点的 8 阶幂级数展式作比较。你怎样评价它们对 Cos[x] 的近似的优劣？考虑如何解释你看到的现象。
3. 另外做些别的例子，例如用 Sin、Exp 做试验。你觉得用 Mathematica 做函数拟合有什么优越的地方？

假如你有一组实测数据，例如：

x	0	0.2	0.3	0.52	0.64	0.7	1.0
y	0.3	0.45	0.47	0.50	0.38	0.33	0.24

怎样做它们的函数拟合呢？首先应当把数据做成一个表，其中每个元素是 x、y 值的对：
 $L = \{\{0,0.3\},\{0.2,0.45\},\{0.3,0.47\},\{0.52,0.50\},\{0.64,0.38\},\{0.7,0.33\},\{1.0,0.24\}\}$

然后就可以做拟合了。例如做一个二次拟合：

```
base={1,x,x^2}
```

```
f=Fit[L,base,x]
```

怎样知道拟合的好不好呢？最好的方法是在一个图上画出数据点和拟合函数。

Mathematica 有一个函数叫 ListPlot，它是用来画数据点的图的，于是：

```
ListPlot[L,PlotJoined->True]
```

```
Plot[f,{x,-0.2,1.2}]
```

```
Show[%,%%]
```

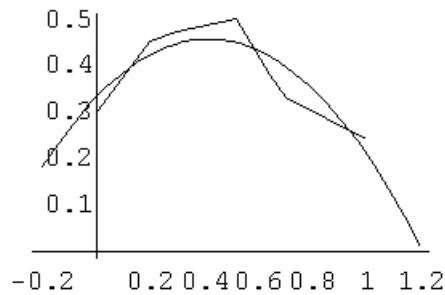


图 4.1.2.2 ListPlot

如果你觉得拟合的不好,或者想再试试,可以自己做做这集数据的三次或更高次数的拟合。

Mathematica 的 **Fit** 函数没有什么希奇的,它设法用你给的函数集的线性组合尽量地去逼近你的原始数据,有许多数值程序包都能做这样的事。但是由于 **Fit** 出现在 **Mathematica** 的整个环境里,它的能力就大大地提高了。你可以很快地用各种函数集做试验,然后可以立刻通过作图等手段观察比较拟合的结果。如果你发现被拟合数据的分布具有指数的性质,那么就可以在函数集中增加一个指数函数;如果认为它们有周期性,那么加一个具有适当周期性的函数。更重要的是,拟合得到的函数是一个正常的表达式,是可以操作的对象。你不但可以直接求它在某些点的值,画它的图形,还可以求它的导函数、积分,把它与别的表达式或函数作运算。

Fit 还能做多元拟合,我们来看一个例子。首先来准备一组数据点:

```
A=Table[{x,y,z,Sin[x*y*z]},{x,0,2,0.5},{y,0,2,0.5},{z,0,2,0.5}]
```

注意大括号内有四项,第四项为表格步长。

这样得到的数据在形式上不符合 **Fit** 的要求。**Fit** 要求的参数是数据点(现在由 4 个数组成)的一个表,而我们做出来的是数据点的一个三层的表。函数 **Flatten** 可以把表展开几层。我们要展开两层,应当写

```
A=Flatten[A,2]
```

于是你就可以试着做拟合了:

```
Fit[A,{x^2,y^2,z^2,x*y,x*z,y*z},{x,y,z}]
```

结果为:

$$0.0430301x^2 - 0.00330202xy + 0.0430301y^2 - 0.00330202xz - 0.00330202yz + 0.0430301z^2$$

再做个例子。首先生成前 20 个素数的表:

```
B=Table[Prime[n],{n,20}]
```

对 **B** 做二次拟合:

```
f = Fit[B, {1,x,x^2},x]
```

这里数据表 **B** 中只有 **y** 坐标值, **Fit** 假定这时对应的 **x** 值分别是 1、2、3、……,画出它们的图:

```
ListPlot[B]
Plot[f,{x,0,20}]
Show[%,%%]
```

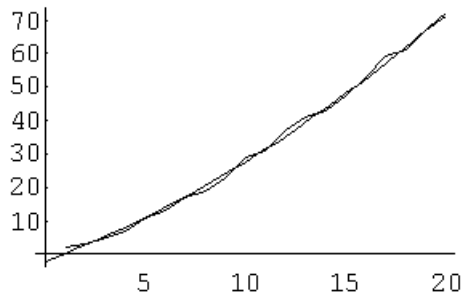


图 4.1.2.3 素数分布拟合

4.2 转换规则和函数

我们已经知道在 **Mathematica** 中，所有的输入都是表达式，所有的操作都是调用转换规则对表达式求值。表达式的求解过程就是将一种表示形式转变为另一种表示形式的过程。

Mathematica 在求解表达式的值时，把各种转化规则作用于表达式，一直到找不到可用的规则时停止，这时所产生的表达式就是表达式求值的结果。

一个函数即对应一个转换规则。**Mathematica** 内容建立了许多转换规则，如算术运算、化简表达式、做微积分运算、绘图等。在前面的介绍中我们主要了解了如何运用这些系统定义的规则，在这里我们将介绍如何建立自己的转换规则。

4.2.1 转换规则

在前文中我们已经知道变换规则是指表达式的变量替换或赋值运算，**Mathematica** 提供了变换规则的运算与定义等功能。

下表给出转换规则的常用形式。

转换规则	意义
<code>expr/.lhs->rhs</code>	对表达式 <code>expr</code> 运用转换规则
<code>expr/.{lhs1->rhs1,lhs2->rhs2,...}</code>	对表达式 <code>expr</code> 中的各项运用这一系列转换规则
<code>expr/.{rules1,rules2,...}</code>	给出依次使用转换规则集合而得到的结果集合
<code>expr/.rules</code>	对表达式 <code>expr</code> 中的所有子项使用一次转换规则
<code>expr//.rules</code>	重复运用转换规则直到结果不变
<code>Replace[expr,rules]</code>	仅对整个表达式 <code>expr</code> 运用转换规则
<code>Replace[expr,rules,levspec]</code>	通过 <code>levspec</code> 指定应用转换规则的层次
<code>ReplaceList[expr,rules]</code>	在所有可能的地方应用转换规则

运算符“/.”对表达式应用转换规则，每次你可以只给出一个转换规则，也可以同时给出一系列的转换规则：

```
In[1]:=x+2y/.x->1
Out[1]=1+2y
In[2]:=x+2y/.{x->1,y->a}
Out[2]=1+2a
```

如果你以集合的形式给出转换规则，那么你也得到集合的结果形式：

```
In[3]:=x+2y/.{x->1,y->a},{x->b,y->c}
Out[3]={1+2a,b+2c}
```

当你使用 `expr/.rules` 时，表示对表达式中的每一项逐个使用每个转换规则。一旦运用了规则，变换也就开始了，结果就跟着出现。注意比较下面的例子（有关符号“_”在后面将做详细的讨论）：

```
In[4]:= {x^2,x^4,x^6}/.{x^2->u,x^n_>p[n]}
```

```
Out[4]={u,p[4],p[6]}
In[5]:=h[x+h[y]]/.h[u]->u^2
Out[5]=(x+h[y])2
```

上面结果的出现是因为 `expr/.rules` 表示每个转换规则对表达式中的每一项只使用一次。你可以运用一系列的规则对表达式中的每一项进行转换：

```
In[6]:=h[x+h[y]]/.h[u_]->u^2/.h[u_]->u^2
Out[6]=(x+y2)2
```

当然 Mathematica 提供了更有效的方法来进行这种转换：

```
In[7]:=h[x+h[y]]//.h[u_]->u^2
Out[7]=(x+y2)2
```

同样也可用在以集合形式表示的转换规则：

```
In[8]:={x^2,x^4,x^6} /. {x->u+1,u->2}
Out[8]={9,81,729}
```

下面再给出有关 “/.” 和 “//.” 的例子，请注意比较：

```
In[9]:=log[a b c] /. log[x_ y_ ]->log[x]+log[y]
Out[9]=log[a]+log[b c]
In[10]:=log[a b c] // .log[x_ y_ ]->log[x]+log[y]
Out[10]=log[a]+log[b]+log[c]
```

当你使用 “//.” 时，Mathematica 会设法运用每一个法则重复地对你的表达式进行转换，直到连续两次结果相同为止。

替换运算符 “/.” 和 “//.” 对于表达式的每个子项表现出相同的特征。相反，`Replace[expr,rules]` 是对整个表达式的替换，而不是对每个子项。

注意比较下面的三个例子：

```
In[11]:=Cos[x^2] /. x->a
Out[11]=Cos[a2]
In[12]:=Replace[Cos[x^2],Cos[x^2]->b]
Out[12]=b
In[12]:=Replace[Cos[x^2],x^2->b]
Out[13]=Cos[x2]
```

但可用 `Replace` 通过指定规则应用到特定层数：

```
In[14]:=Replace[Cos[x^2],x->a,2]
Out[14]=Cos[a2]
```

下面的例子表示 `Replace` 应用了规则表中的第一个转换规则：

```
In[15]:=Replace[f[u],{f[x_]->x^2,f[x_]->x^3}]
Out[15]=u2
```

而 `ReplaceList` 则应用每一个转换规则，以集合的形式给出结果：

```
In[16]:=ReplaceList[f[u],{f[x_]->x^2,f[x_]->x^3}]
Out[16]={u2,u3}
```

如果一个规则可以有多种可能的应用形式，则 `ReplaceList` 以集合的形式给出每一种可能的结果：

```
In[17]:=ReplaceList[a+b+c,x_+y_->g[x,y]]
Out[17]={g[a,b+c],g[b,a+c],g[c,a+b],g[a+b,c],g[a+c,b],g[b+c,a]}
```

4.2.2 函数定义

在前文中，我们已经看到许多建立 Mathematica 函数的例子。在我们学习工作中，有时会经常用到某种类型的表达式，这时我们可以把它定义为一个函数。在这里，我们主要讨论如何自己定义函数。

下面给出一个简单的例子，即定义自变量加上 1 的函数，用 Mathematica 命令定义这个函数 $f[x_]:=x+1$ （在这里我们对函数定义时对“:=”和“=”未加以区别，在下一部分，我们将作相应讨论）：

In[1]:=f[x_]:=x+1

注意：“_”是在 x 的右边，它在定义函数时是非常重要的。

$f[x_]:=x+1$ 规定无论在何时 Mathematica 遇到和模型 $f[x_]$ 相匹配的表达式，Mathematica 都用表达式 $x+1$ 来替换。

自变量可为任意数字或表达式：

In[2]:=f[2]

Out[2]=3

In[3]:=f[a+1]

Out[3]=2+a

In[4]:=f[a^2+2a+1]

Out[4]=2+2a+a²

其实你也可用 $f[x]=x+1$ 来定义函数，不过 $f[x]=x+1$ 规定无论何时出现表达式 $f[x]$ ，它都可用 $x+1$ 来代替，但该定义不适合像 $f[y]$ 这样的表达式。

下表中给出函数定义有关的命令。

命令	意义
$f[x]=value$	对一个特定的表达式 $f(x)$ 进行定义
$f[x_]=value$	对用 $f(x)$ 来指代的任何表达式进行定义
?f	显示 f 的定义
Clear[f]	清除所有有关 f 的定义

$f[1]$ 或 $f[2]$ 的定义可被看作对数组 f 的不同元素的赋值。而 $f[x_]$ 的定义可被看作是对一系列具有代数符号的数组元素赋值。事实上，任何函数的定义都是如此。

下面给特定表达式 $f[x]$ 定义一个转换规则：

In[5]:=f[x]=a

Out[5]=a

则在表达式运算中一旦出现 $f[x]$ 的形式，它就用 a 来替换，而其他表达式则不变，即使它们的形式类似：

In[6]:=3 f[x]+2 f[y]+f[x] f[xy]

Out[6]=3a+a f[xy]+2f[y]

下面定义了可以具有任何表达式变量的 f ：

In[7]:=f[z_]=z+1

Out[7]=1+z

在下面定义的表达式运算中同时使用了两种定义

In[8]:=3 f[x]+2 f[y]+f[x] f[xy]

Out[8]=3a+a(1+xy)+2(1+y)

我们也可以只用 $f[x_]$ 的定义，不过这时要清除 $f[x]$ 的定义，Clear[f] 同时清除所有有关 f 的定义，故要重新定义 $f[x_]$ ：

In[9]:=Clear[f]

In[10]:=f[z_]=z+1

Out[10]=1+z

In[11]:=3 f[x]+2 f[y]+f[x] f[xy]

Out[11]=3(1+x)+(1+x)(1+xy)+2(1+y)

用 **Clear** 清除你定义的函数是一件很重要的工作，因为在后面的计算中，你可能不再需要这个转换，但 **Mathematica** 还会记住且在你清除前会一直调用它。

有时，我们忘了我们定义的函数形式，我们可以用命令“？”来查看函数的类型，**Mathematica** 的内部函数也可用这个命令进行查看：

```
In[12]:=?f
Global`f
f[z_]=1+z
```

```
In[13]:=Clear[f]
```

“？”也可用与查看内部函数

```
In[14]:=?Plot
```

```
Plot[f,{x,xmin,xmax}] generates a plot of f as a function of x from xmin to xmax,
Plot[{f1,f2,...},{x,xmin,xmax}] plots several functions fi.
```

如果你想知道 P 打头的内部函数有哪些，可以写：

```
?P*
```

这里的星号代表任意字符串。你可以看到屏幕上列出了许多函数的名字。于是你可以进一步找出你感兴趣的函数，去看它们的说明。

如果对一个函数的选项不清楚，可以用 **Options** 命令，它的用法是：

```
Options[fun_name]
```

如 **Options[ListPlot]**、**Options[Plot3D]** 等。

一个特定的模式在使用转换规则时可以代表不同的表达式，函数中的模式可以有多个或多种形式。下表给出常用的模式。

函数模式	意义
f[x_]	变量名为x 的函数f
f[x_,y_]	变量名为x 和y 的函数f
f[x_,x_]	具有两个恒等变量的函数f
x^n_	x的任意次幂，幂次为n
x_^n_	任意表达式的任意次幂
a_+b_	两个表达式之和
a1_,a2_	两个表达式的集合

下面首先给出两个变量函数 **h1** 的定义，然后调用它，对于一个变量的 **h1** 函数，我们定义的 **h1** 不起作用：

```
In[15]:=h1[x_,y_]:=x^y
```

```
In[16]:=h1[a,2]
```

```
Out[16]=a2
```

```
In[17]:=h1[a]+h1[b,2]
```

```
Out[17]=b2+h1[a]
```

下面是两个恒等变量的简单例子：

```
In[18]:=h[x_,x_]:=x^x
```

```
In[19]:=h[3,3]
```

```
Out[19]=27
```

注意比较在下面的转换规则中定义的不同，即在表达式调用时体现不同的性质：

```
In[20]:={{x^2,x^3},{y^2,y^3}}/.x^n_>r[n]
```

```

Out[20]={{r[2],r[3]},{y^2,y^3}}
In[21]:= {{x^2,x^3},{y^2,y^3}}/.x_^n_>r[n]
Out[21]={{r[2],r[3]},{r[2],r[3]}}
In[22]:= {a+b,c+d}/.x_+y_>xy
Out[22]={xy,xy}

```

在 **Mathematica** 中允许你对任何表达式或模式定义转换规则。在 **In[18]**中我们已经看出我们可以混用两种函数定义形式。其实在 **Mathematica** 中，许多数学函数都是通过特殊定义和一般定义的混合使用建立的。譬如阶乘函数，虽然这个函数 **Mathematica** 已经定义了，但你能用 **Mathematica** 中的定义来建立你自己的函数。

在 **Mathematica** 中，阶乘的定义采用了标准数学中的定义，即

$$f[n_]:=n f[n-1];f[1]=1;$$

这个定义的含义是：对于任何 n ， $f[n]$ 必须用 $n f[n-1]$ 来替代，但 $n=1$ 的情况除外，规定当 $n=1$ 时 $f[n]=1$ 。

下面定义了变量为 1 的阶乘函数值，以及阶乘函数的一般递推关系公式：

```

In[23]:=f[1]=1
Out[23]=1
In[24]:=f[n_]:=n f[n-1]

```

现在你可用阶乘函数来求任意阶乘的值，譬如求 15 的阶乘：

```

In[25]:=f[15]
Out[25]=1307674368000

```

结果与 **Mathematica** 内部给出的阶乘模型得到的结果相同：

```

In[26]:=15!
Out[26]=1307674368000

```

当你在 **Mathematica** 中建一系列的定时时，其中有时采用一般函数（如上例中的 $f[n_]:=n f[n-1]$ ），有时采用特殊函数（如上例中的 $f[1]=1$ ）。**Mathematica** 一般遵循这样的原则，即特殊函数优先级高于一般函数。

4.2.3 立即赋值和延迟赋值

你也许注意到在 **Mathematica** 中有两种不同的赋值方式： $lhs=rhs$ 和 $lhs:=rhs$ 。这两种方式的不同之处是对 rhs 进行计算的时间不同。 $lhs=rhs$ 是立即赋值，这个等式中的 rhs 在赋值的同时也就被计算了。 $lhs:=rhs$ 是延时赋值，在这个式子中的 rhs 并不在赋值的同时赋值，它是在需要 lhs 值时才进行计算的。

下表给出它们的形式和意义：

函数	意义
$lhs=rhs$	立即赋值；赋值的同时计算 rhs ，即 rhs 被假设为 lhs 的最终值
$lhs:=rhs$	延时赋值；当要求得到 lhs 时计算 rhs

下面用运算符“:=”来定义函数 **ex**，由于使用的是“:=”运算符，因此当用“？”查看函数 **ex** 时可发现定义仍保持没有被计算之前的形式：

```

In[1]:=ex[x_]:=Expand[(a+b x)^2]
In[2]:=?ex
Global`x
Ex[x_]:=Expand[(a+b x)^2]

```

下面用运算符“=”来定义函数 **ie**，由于使用的是“=”运算符，因此当用“？”查

看函数 `iex` 时可发现保存的定义是 `Expand` 命令展开的形式:

```
In[3]:=iex[x_]=Expand[(a+bx)^2]
```

```
Out[3]=a2+2 a b x+b2 x2
```

```
In[4]:=?iex
```

```
Global`iex
```

```
iex[x_]=a2+2 a b x+b2 x2
```

下面分别调用这两个函数，试比较它们的结果:

```
In[5]:=ex[y+2]
```

```
Out[5]=a2+4 a b+4b2+2 a b y+4b2 y+b2 y2
```

```
In[6]:=iex[y+2]
```

```
Out[6]=a2+2 a b(2+y)+b2 (2+y)2
```

正如你上面看到的那样，在定义函数时，“:=”和“=”都很有用，但我们一定要注意它们具有的不同意义。

在使用“:=”和“=”的一个经验规则就是：当你想得到表达式的最终结果时，可用“=”；当你希望得到的是一个特殊的“命令”，那么你最好用“:=”运算符。

一般情况下，定义函数时“:=”比“=”常用，但对于下面这种情况，你则必须用“=”来定义函数。如果你在计算时得到关于符号参数 x 的结果，你会想要继续进行并且要得到 x 的各个不同特征值，这时用“=”来定义变量 x 更为合适了。

下面是一个含有 x 的表达式:

```
In[7]:=D[Log[Sin[x]]^2,x]
```

```
Out[7]=2 Cot[x] Log[Sin[x]]
```

下面定义了一个函数，该函数的变量是关于 x 的值:

```
In[8]:=dlog[x_]=%
```

```
Out[8]=2 Cot[x] Log[Sin[x]]
```

下面调用这个函数:

```
In[9]:=dlog[1+a]
```

```
Out[9]=2 Cot[1+a] Log[Sin[1+a]]
```

前面我们所讨论的主要是用“:=”和“=”来定义函数，其实也可用它们对变量进行赋值。同前面所说的一样，用“=”赋值时，结果立即被计算出来，而用“:=”赋值时，它保持没有被计算时的形式。

下面通过计算 `Random[]` 来得到一个随机数，并且把它赋给 `rd1`:

```
In[10]:=rd1=Random[ ]
```

```
Out[10]=0.327946
```

对于下面的定义方式，`Random[]`保持原来的形式，只有当使用 `rd2` 时才进行计算:

```
In[11]:=rd2:=Random[ ]
```

下面两次调用 `rd1` 和 `rd2`，`rd1` 的值保持不变，`rd2` 则在变化:

```
In[12]:= {rd1,rd2}
```

```
Out[12]={0.537946,0.450938}
```

```
In[13]:= {rd1,rd2}
```

```
Out[13]={0.327946,0.334043}
```

当你进行一系列的赋值时，立即赋值和延时赋值之间的差别是非常重要的。这一点，我们在实际计算时一定要引起重视。

请参看下面的例子，`a` 定义了两次，`ia` 保持原结果不变，`da` 则随着 `a` 的变化而变化:

```
In[14]:=a=1
```

```

Out[14]=1
In[15]:=ia=a+2
Out[15]=3
In[16]:=da:=a+2
In[17]:={ia,da}
Out[17]={3,3}
In[18]:=a=2
Out[18]=2
In[19]:={ia,da}
Out[19]={3,4}

```

因为延时赋值有保持表达式的不变性，故我们可用它来设置在不同环境下有不同值的变量。

在你使用“:=”建立一个函数的定义以后，一旦你要调用函数，**Mathematica** 就重新计算该函数。在许多计算中，你可能要对同一函数访问多次，在这种情况下，你可以通过让 **Mathematica** 记住它发现的所有函数值的方法来节省时间。

下表给出定义这样函数的形式。

函数	意义
$f[x_]:=f[x]=rhs$	定义一个能记住它发现的所有值的函数

定义函数 f ，该函数存取它发现的所有值，我们同时给出这个递推函数 f 的结束条件：

```

In[20]:=f[x_]:=f[x]=f[x-1]+f[x-2]
In[21]:=f[0]=f[1]=1
Out[21]=1

```

下面是显示 f 的原始定义：

```

In[22]:=?f
Global`f
f[0]=1
f[1]=1
f[x_]:=f[x]=f[x-1]+f[x-2]

```

下面计算 $f[5]$ ，这个计算同时包括寻找一系列的值，即 $f[5], f[4], f[3], f[2]$ ：

```

In[23]:=f[5]
Out[23]=8

```

下式显示的函数 f 的所有值都被保存起来：

```

In[24]:=?f
Global`f
f[0]=1
f[1]=1
f[2]=2
f[3]=3
f[4]=5
f[5]=8
f[x_]:=f[x]=f[x-1]+f[x-2]

```

如果你再次调用 $f[5]$ ，**Mathematica** 可以立即得到 $f[5]$ 的值，而不需要重新计算：

```

In[25]:=f[5]
Out[25]=8

```

4.2.4 具有不定数目变量的函数

我们在前面讨论过如何定义函数，但像 $f[x_]$ 的函数模型只代表具有一个变量的函数， $f[x_,y_]$ 也只代表具有两个变量的函数。而有时你需要建立任意数目变量的函数模型。这时，你也可通过使用多个间隔号来实现。一个间隔号如 “ $x_$ ” 代表一个 Mathematica 表达式，两个间隔号如 “ $x_ _$ ” 表示一系列表达式。

下表列出函数定义中表示形式参量的数目的标记。

函数	意义
$_$	任何单一表达式
$x_$	任何名为 x 的单一表达式
$_ _$	任何一个或多个表达式序列
$x_ _$	序列名为 x 的表达式
$x_ _ h$	所有序列头部为 h 表达式
$_ _ _$	0个或多个表达式序列
$x_ _ _$	任何名为 x 的0个或多个表达式序列
$x_ _ _ h$	头部为 h 的0个或多个表达式序列

下面的 $x_ _ _$ 代表序列表达式 $\{a,b,b\}$:

```
In[1]:=f[a,b,c]/.f[x_ _ ]->p[x,x,x]
```

```
Out[1]=p[a,b,c,a,b,c,a,b,c]
```

其实我们可用函数的形式来定义，这样看起来就更清楚了：

```
In[2]:=f[x_ _ ]=p[x,x,x]
```

```
Out[2]=p[x,x,x]
```

```
In[3]:=f[a,b,c]
```

```
Out[3]=p[a,b,c,a,b,c,a,b,c]
```

为更好的搞清楚多个间隔号的意义，下面分别定义 $f1[x_,y_]$ 和 $f2[x_ _,y_ _]$ ，请注意比较它们的异同：

```
In[4]:=f1[x_,y_]:=2(x+y)
```

```
In[5]:=f2[x_ _,y_ _]:=2(x+y)
```

当为两个变量时，他们返回的结果相同：

```
In[6]:=f1[a,b]
```

```
Out[6]=2(a+b)
```

```
In[7]:=f2[a,b]
```

```
Out[7]=2(a+b)
```

但当变量为多于两个时， $f2[x_ _,y_ _]$ 能给出结果，而 $f1[x_,y_]$ 则不能：

```
In[8]:=f1[a,b,c]
```

```
Out[8]=f1[a,b,c]
```

```
In[9]:=f2[a,b,c]
```

```
Out[9]=2(a+b+c)
```

下面定义一个变量的函数 $g[x_ _]$ ，它给出加法的形式，注意体会其用法：

```
In[10]:=g[x_ _]:=x+y
```

```
In[11]:=g[a]
```

```
Out[11]=a+y
```

```
In[12]:=g[a,b,c]
```

```
Out[12]=a+b+c+y
```

下面给出函数 $g1[_ _]$ 的定义，则对任意一个或多个表达式序列，它都可给出确定的结果，不过结果形式不能随变量的改变而改变：

```
In[13]:=g1[_ _]:=x+y
```

```
In[14]:=g1[a]
```

```
Out[14]=x+y
```

```
In[15]:=g1[a,b,c]
```

```
Out[15]=x+y
```

下面我们再给出一个复杂的例子：

```
In[16]:=h[a_.,x,b_.,x_,c_]:=hh[x] h[a,b,c]
```

运用定义两次得出两个配对元素：

```
In[17]:=h[2,3,2,4,5,3]
```

```
Out[17]=h[4,5]hh[2]hh[3]
```

双间隔号“`._.`”表示一个或多个表达式序列，三个间隔号“`._._.`”表示 0 个或多个表达式序列。无论何时你使用三个间隔号模型都要特别小心，因为这样很容易陷入死循环状态。例如，如你定义 $f[x_,y_._._]=f[x]g[y]$ ，然后输入 $f[a]$ 则将陷入死循环，这是因为将重复地用 0 序列来表示。除非你想包括 0 元素，否则你应尽量使用两个间隔号。

当你使用多个间隔号时，对于特定的表达式可能有多个匹配形式。一般来说，Mathematica 总是设法寻找与第一个间隔号相匹配的最短变量序列的表达式。

4.2.5 以函数作为参数的系统函数

这里我们来看几个以函数作为参数的系统内部函数。第一个函数是 **Map**，它的功能是把一个函数(一元的，无论是系统内部函数还是你自己定义的函数)作用到一个表里所有的元素上，得到一个新的表。例如：

```
myplot[r_]:=ParametricPlot[{r*Cos[t],r*Sin[t]},{t,0,2Pi}]
Map[myplot,{1,2,3,4,5}]
Show[%]
```

你应当顺序得到五个圆的图，后面的 **Show** 使你得到套在一起的五个圆的图。也可以用一下前面定义的 **f**：

```
Map[f, Table[n,{n,10}]]
```

你得到了一个包含 10 个整数的表，这些整数分别是 **f** 作用到 1、2、...，10 的结果。

再一个应当知道的函数是 **Nest**，它把一个函数复合指定的次数，作用到给定参数上，例如：

```
Nest[Sin,x,10]
```

让我们定义一个新函数：

```
g[x_]:=1/(1+x)
```

你可以做出一个连分式：

```
Nest[g,x,5]
```

让我们来做一个表：

```
Table[Nest[g,1,n],{n,0,10}] //N
```

你认识这个序列中的数吗？如果你取更多的项，你会确信这个系列趋向一个极限，黄金分割。系统里正好有这个常数：

```
N[1/GoldenRatio, 30]
```

系统函数 **NestList** 做的正是我们用 **Table** 和 **Nest** 合起来做的事，请看：

```
Nestlist[g,1,10] //N
```

4.2.6 函数属性

在 Mathematica 中，我们不仅可以定义函数的运算规则，还可以自己设置函数的属性。函数的属性对函数的运算规则和调用时的模式匹配有直接的作用和影响。在本小节中我们主要学习有关函数的属性及如何定义函数的属性，另外我们还将学习如何修改 Mathematica

的内部函数等。

(1) 函数的属性

下表给出 Mathematica 中函数的属性及其意义。

函数	意义
Orderless	可交换函数
Flat	可结合函数
OneIdentity	$f[f[a]]$ 等与a模式上匹配
Listable	函数自动分配到列表中（例如 $f[\{a,b\}]$ 变成 $\{f[a],f[b]\}$ ）
Constant	函数的所有导数为0，即是常数函数
NumericFunction	当函数被赋予数值特性时，就假定它有一个数字值
Protected	函数的值不能被改变
Locked	函数的属性值不能被改变
ReadProtected	函数值不能被读取
HoldFirst	函数的第一个参量不求值
HoldRest	除第一个变量外，所有的函数参量不求值
HoldAll	所有的函数参量不求值
HoldAllComplete	函数的参数被当成惰性参数
NholdFirst	函数的第一个参量不被命令N作用
NholdRest	除第一个变量外，所有的函数参量不被命令N作用
SequenceHold	所有的函数参量不被命令N作用
Temporary	函数作为一个局部变量，当不再使用时将被去除掉

Mathematica 系统定义的函数都相应地定义了属性，用户可以根据具体情况给自己定义的函数设置属性或修改系统函数的属性。

Mathematica 提供了一系列的属性函数，你可以使用这些函数来规定函数的不同属性。下表给出了有关函数属性的函数。

函数	意义
Attributes[f]	显示函数f的属性
Attributes[f]={attr1,attr2,...}	设置函数f的属性
Attributes[f]={}	设置函数f无任何属性
SetAttributes[f,attr]	把attr加入到f的属性表中
ClearAttributes[f,attr]	清除函数f属性表中的attr属性
Clear[f]	清除函数f的值，但不清除其属性
ClearAll[f]	同时清除函数f的值和属性

函数的属性表示这些函数在计算和模型匹配时是如何处理的。例如函数能被赋以 Orderless 属性，这表示函数具有交换性和对称性，并且允许函数变量被重新组合。

加法函数 Plus[]具有这种可交换性，Plus[a+b]就相当于“a+b”，我们知道在 Mathematica 中 a+b 等于 b+a，因此我们称 Plus 有交换性。下面给出 Plus 的属性：

```
In[1]:=Attributes[Plus]
```

Out[1]={Flat,Listable,NumericFunction,OneIdentity,Orderless,Protected}

从这我们还可看出, Plus[]函数除了具有交换性外,还具有其他一些如可结合性等属性。

下面定义 hs 为具有可交换性函数:

In[2]:=SetAttributes[hs,Orderless]

hs 的自变量自动的按 Mathematica 规定的顺序自动排列:

In[3]:=hs[b,a,c]

Out[3]=hs[a,b,c]

函数 Plus[]除了具有交换性外,从它的属性表中,我们知道它还具有结合性。这是指能给变量加上圆括号,如 $x+(y+z)$ 等价于 $x+y+z$ 等。

在匹配模型时, Mathematica 考虑了结合性。因此模型 $g[x_+y_]$ 能与 $g[a+b+c]$ 相匹配,此时 $x \rightarrow a$, $y \rightarrow (b+c)$ 。

在下例中 g 的变量被写成 $a+(b+c)$, 以便模型相匹配:

In[4]:=g[a+b+c]/.g[x_+y_]->p[x,y]

Out[4]=p[a,b+c]

在不具有结合性的函数中,像 $x_$ 的模型匹配具有一个独立变量的函数。但在有结合性的函数 $f[a,b,c,\dots]$ 中, $x_$ 能匹配如 $f[b,c]$ 的函数,函数 $f[b,c]$ 具有多个变量。然而在 $x_$ 只表示结合性的函数中的单个变量的情况下,存在 $x_$ 是匹配 a 本身还是 $f[a]$ 这样的问题。若函数带有属性 OneIdentity, 则 Mathematica 首先选用第一种情况,然后在选用第二种情况。

函数 Plus[]具有这种属性:

In[5]:=Plus[a+b]

Out[5]=a+b

在 Mathematica 内部数学函数中,有一个重要的属性就是 Listable。该属性规定一个函数必须被自动的散布开来,这就意味着该函数能有效的适用以变量形式出现的集合中的所有元素。

内部正弦函数具有 Listable 属性:

In[6]:=Sin[{1,2,3}]

Out[6]={Sin[1],Sin[2],Sin[3]}

对于一任意函数 q,我们先不给它任何属性,则:

In[7]:=q[{a,b,c},d]

Out[7]=q[{a,b,c},d]

下面定义函数 q 为 Listable:

In[8]:=SetAttributes[q,Listable]

In[9]:=q[{a,b,c},d]

Out[9]={q[a,d],q[b,d],q[c,d]}

在 Mathematica 中,许多你赋予函数的属性直接影响着这些函数的运算。当然,有些属性只影响函数处理的一些其他方面。例如,上面提到的 OneIdentity 只影响模型匹配,同样 Constant 只与函数求导有关,并且它的运算依赖于求导。

Protected 属性影响赋值, Mathematica 不允许你对具有该属性的函数进行定义。Mathematica 中的大部分内部函数都被保护起来以避免我们可能对它进行错误定义,我们将在下一部分对它进行专门讨论。

下面是定义函数 p:

In[10]:=p[x_]=x^2

Out[10]=x²

给函数 p 加上 Protected:

```
In[11]:=Attributes[p]={Protected}
```

```
Out[11]={Protected}
```

现在你就不能对 p 进行修改了：

```
In[12]:=p[x_]=x
```

```
Set::write::Tag p in p[x_] is Protected.
```

```
Out[12]=x
```

我们可通过“? 函数名”或者使用其他内部函数就可以显示你建立的定义。然而，如果你设置了属性 `ReadProtected`。Mathematica 就不再允许你显示特定函数的定义。当然在你进行计算时仍然使用这一定义。

虽然我们不能再对 p 进行再定义，但我们可以显示这个定义：

```
In[13]:=?P
```

```
Global`p
```

```
Attributes[p]={Protected}
```

```
p[x_]=x2
```

下面给 p 设置了 `ReadProtected` 属性：

```
In[14]:=SetAttributes[p,ReadProtected]
```

现在我们不能看到 p 的定义了：

```
In[15]:=?p
```

```
Global`p
```

```
Attributes[p]={Protected,ReadProtected}
```

我们可以用函数 `SetAttributes` 和 `ClearAttributes` 来修改某一函数的属性，但是一旦你对某函数设置了 `Locked` 属性，那么 Mathematica 也就不允许你对该函数的属性进行修改了。不过我们可以通过 `ClearAll` 来清除函数的所有属性。

下面清除 q 的值和属性，则 q 不再具有 `Listable` 属性了：

```
In[16]:=ClearAll[q]
```

```
In[17]:=q[{a,b,c},d]
```

```
Out[17]=q[{a,b,c},d]
```

(2) 修改内部函数

Mathematica 不仅允许你对你自己定义的函数进行转换规则的定义，还允许你对计算机内部的函数进行定义。不过这样做你虽然能增强、修改内部函数的特性，但同时它也是很危险的，因为 Mathematica 将遵从你给出的规则。这也就意味着你给出的转换规则不正确，那么 Mathematica 也将给你一个错误的结果。

在 Mathematica 中，为了避免你不经意的情况下改动内部函数，它对其内部所有函数都进行了保护。如果你想修改内部某个函数的话，你首先得去掉函数的保护属性。同样在你定义之后，你必须恢复这种保护属性以防止误操作。

下表是给函数加保护或去掉保护的命令：

函数	意义
<code>Protect[f]</code>	加上保护
<code>Unprotect[f]</code>	去掉保护

因为内部函数都被保护了，故你不能对它们进行重新定义：

```
In[1]:=Sqrt[10]=3
```

```
Set::Write::Tag Sqrt in  $\sqrt{10}$  is Protected
```

```
Out[1]=3
```

去掉 Sqrt 的保护 Protected:

```
In[2]:=Unprotect[Sqrt]
Out[2]={Sqrt}
```

现在你可以对 Sqrt 进行再定义。虽然定义不正确，但 Mathematica 也将遵循你的这一定义:

```
In[3]:=Sqrt[10]=3
Out[3]=3
In[4]:=Sqrt[10]+Sqrt[5]
Out[4]=3+ $\sqrt{5}$ 
```

清除 Sqrt 错误的定义，还原其原定义，并重新保护 Sqrt 的内部定义:

```
In[5]:=Sqrt[10]=.
In[6]:=Protect[Sqrt]
Out[6]={Sqrt}
In[7]:=Sqrt[10]+Sqrt[5]
Out[7]= $\sqrt{5} + \sqrt{10}$ 
```

在 Mathematica 中，你的定义可以超越内部函数的定义。一般来说，Mathematica 总是在使用内部函数定义之前使用你的定义。Mathematica 遵循这样的法则有利于你扩大计算领域。在有些特殊情况下，你可以不按内部规则去执行。在这种情况下，你可以定义你自己的规则去超越内部函数的规则，参见下例。

下面是内部函数化简 Exp[Log[expr]] 的规则:

```
In[8]:=Exp[Log[y]]
Out[8]=y
```

现在建立你自己的化简规则来替代内部规则，然后 Mathematica 将使用你的定义，而不是使用其内部函数:

```
In[9]:= (
  Unprotect[Exp];
  Exp[Log[expr_]]:=explog[expr];
  Protect[Exp];
)
In[10]:=Exp[Log[y]]
Out[10]=explog[y]
```

4.2.7 纯函数

我们对于多次使用的数学函数或要用一系列的运算才能完成的工作，我们可以用规则定义函数。在定义函数时我们要确定函数的变量和函数名字，但对于只在某处使用的特殊函数，这样定义函数显得有些多余，但是不定义又难以描述表达式和自变量的关系。在本小节中我们将要讨论的纯函数就允许你给出能够被运用于变量的函数，而又不必定义显式函数名。定义纯函数时，我们还可以用标记“#”表示的变量，即省略变量名。

下表给出定义纯函数的定义形式:

函数	意义
Function[x,body]	一个变量的纯函数，你可以用任何变量来替代x
Function[{x1,x2,...},body]	多个变量的纯函数
Body&	变量被规定为#或#1,#2,#3等

纯函数经常用到 Nest, Map, Apply 等函数。我们在使用这些函数时，也必须规定所运用的函数。

下面定义一个函数 h[x_], 在 Map 中你可以使用它的函数名 h:

```
In[1]:=h[x_]:=h1[x]
In[2]:=Map[h,{a,b,c}]
```

```
Out[2]={h1[a],h1[b],h1[c]}
```

应用纯函数可以得到同样的结果:

```
In[3]:=Map[h1[#] &,{a,b,c]}
```

```
Out[3]={h1[a],h1[b],h1[c]}
```

利用纯函数我们也可以直接计算出一个特定的函数值:

```
In[4]:=Function[x,Sin[x]]
```

```
Out[4]=Function[x,Sin[x]]
```

```
In[5]:= % [2]
```

```
Out[5]=Sin[2]
```

我们可在 Map, Nest 等中使用纯函数:

```
In[6]:=Map[Function[x,Sin[x]],a+b+c]
```

```
Out[6]=Sin[a]+Sin[b]+Sin[c]
```

```
In[7]:=Nest[Function[q,1/(1+q)],x,3]
```

```
Out[7]=
$$\frac{1}{1+\frac{1}{1+x}}$$

```

下面是多个变量的纯函数:

```
In[8]:=Function[{x,y},x-y^2][1,b]
```

```
Out[8]=1-b2
```

就像如果你不再借助函数的话,那么函数名也就不相干一样,在纯函数中的变量名也是不相干的。Mathematica 允许你避免使用纯函数的变量显式名,并且允许你用“#n”来确定变量。在 Mathematica 的纯函数中,“#n”代表你提供的第 n 个变量,仅一个“#”则代表第一个变量。

下表给出我们常用的形式:

函数	意义
#	纯函数中的第一个变量
#n	纯函数中的第n个变量
##	纯函数中的所有变量
##n	纯函数中从第n个往后的变量

#2&是求变量平方的纯函数的短格式:

```
In[9]:=Map[#2&,{a,b,c}]
```

```
Out[9]=a2+b2+c2
```

下面是使用了提取集合前两个元素的函数,如果使用纯函数的话,那么你可以避免单独定义函数:

```
In[10]:=Map[Take[#,2]&,{2,1,7},{4,1,5},{3,1,2}]
```

```
Out[10]={{2,1},{4,1},{3,1}}
```

注意:“&”在使用纯函数的短格式时是非常重要的,它是纯函数短格式的标志。

通常情况下,“&”的优先级较低,在必要的情况下或你不清楚哪个优先级较高的情况下,我们建议你使用圆括号来明确优先级,例如 Option->(fun&)。

Mathematica 中的纯函数能够处理任意多的变量。

下面用“##”代表所有变量:

```
In[11]:=f[##,##]&[x,y]
```

```
Out[11]=f[x,y,x,y]
```

“##2”代表除第一个变量外的所有变量:

```
In[12]:=Apply[f[##2,#1]&,{a,b,c},{ap,bp}},{1}]
Out[12]={f[b,c,a],f[bp,ap]}
```

4.3 函数与图形

我们已经知道如何用 **Mathematica** 做一元显函数的图形和以参数方程形式表示的一元函数的图形。实际上，**Mathematica** 还有许多作图函数，可以画多种其他图形，包括二元函数的三维图形、密度图、等值线图，利用系统的程序包还可以画出参数表示的函数的三维图形和制作动画等。这一节我们讨论这方面的内容。

4.3.1 三元函数的三维图形

画三维图形最常用的函数是 **Plot3D**。你可以用它画出任何二元显函数表达式的图形来。例如：**Plot3D[x^2 Sin[y], {x,-1,1}, {y,-Pi,Pi}]**

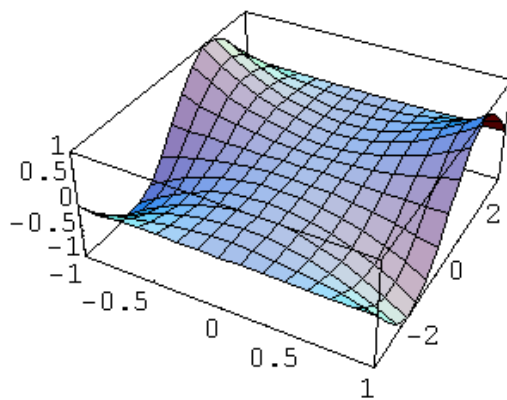
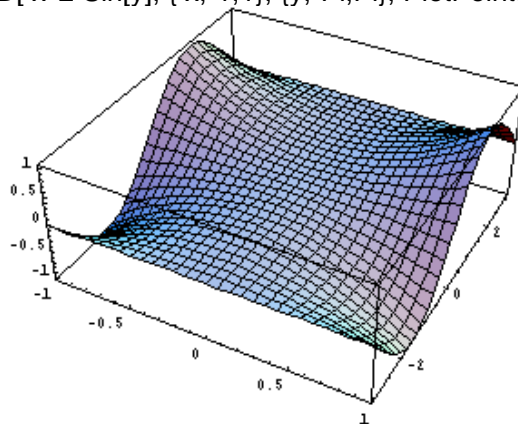


图 4.3.1.1 **Plot3D[x^2 Sin[y], {x,-1,1}, {y,-Pi,Pi}]**

Mathematica 做三维图形的方法是在给定区域中计算出系列格点的值，把它们作为拼出曲面用的一小块“小瓦片”的角的位置。像 **Plot** 一样，**Plot3D** 也有许多选择项，这里介绍几个最常用的。首先，由于图中的曲面是由小瓦片拼成的，它只是实际曲面的一个近似。如果实际曲面的表面变化复杂，画出的曲面就可能反映不出实际曲面的某些情况。这样的问题可以通过对区域做更细的剖分，用更多的小瓦片拼图的方法来改善。**Plot3D** 也是用 **PlotPoints** 说明剖分数的：

```
Plot3D[ x^2 Sin[y], {x,-1,1}, {y,-Pi,Pi}, PlotPoints->30]
```



让我们来考虑另一个图形：

```
Plot3D[Cos[x^2+y^2]^80,{x,-0.5,0.5},{y,-0.5,0.5},PlotRange->All]
```

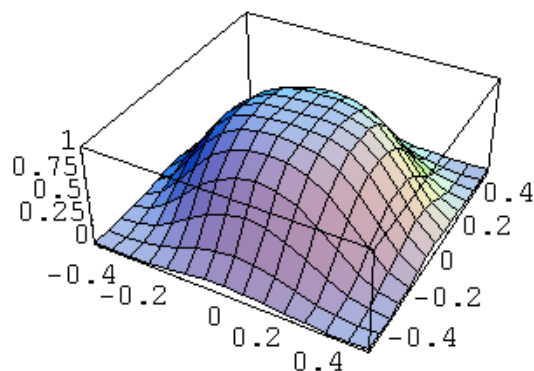


图 4.3.1.2 `Plot3D[Cos[x^2+y^2]^80,{x,-0.5,0.5},{y,-0.5,0.5},PlotRange->All]`

这里的选项 `PlotRange->All` 告诉 Mathematica 画出函数图形所有地方的实际情况。如果不讲这个，系统将自动根据情况突出图形中最重要的部分，它可能切掉图形中某些尖峰。你也可以根据自己的需要，要求系统画出某个特定范围的情况，例如：

```
Show[%,PlotRange->{0,0.5}]
```

前面我们说系统预先设置了几个光源，实际上就是设置了选择项 `LightSources` 的内部值。你也可以自己设光源。确定一个光源有两件事要说明：光源的位置和光色。位置用空间坐标表示，写成三个元素的表；光色用函数 `RGBColor` 表示，它的三个参数分别描述光源光里红、绿、蓝三种成分的强度。例如：`{{2,2,2},RGBColor[0.5,0.8,0.3]}` 表示一个放在 $(2,2,2)$ 点的光色为红绿蓝成分分别为 0.5,0.8,0.3 的点光源。让我们重画上面的图：

```
Plot3D[Cos[x^2+y^2]^80,{x,-0.5,0.5},{y,-0.5,0.5},PlotRnge->A11,  
LightSources->{{{2,2,2},RGBColor[0.5,0.8,0.3]}},Lighting->True]
```

实际上，`LightSources` 对应的值是一个表，其中可以设置若干个光源。请试试这些，再画几个你感兴趣的图形。

4.3.2 三维的参数作图

画三维参数曲面的函数是 `ParametricPlot3D`。在现有版本中，它是系统内部函数，可以直接使用。但对 2.0 以下的版本，使用之前必须先把相应程序包调进来：

```
<<"packages\graphics\parametr.m"
```

当你又看到 `In[...]:=` 时，程序包已调入完毕。让我们先试一个最熟悉的图形，圆球面：

```
Clear[u,v]  
ParametricPlot3D[{Sin[u]Cos[v],Sin[u]Sin[v],Cos[u]},{u,0,Pi},{v,0,2Pi}]
```

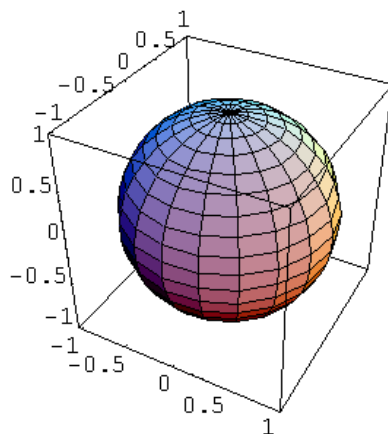


图 4.3.2.1 `ParametricPlot3D[{Sin[u]Cos[v],Sin[u]Sin[v],Cos[u]},{u,0,Pi},{v,0,2Pi}]`

你还可以画些别的图，例如上图的一个变形：

```
ParametricPlot3D[{Sin[u]Cos[v],Sin[u]Sin[v],Cos[u]},{u,Pi/8,7/8Pi},{v,0,3/4Pi}]
```

我们再来看一个复杂点的参数方程图形，画一个莫比乌斯带。莫比乌斯带的一种参数描述是：

$$r(t, v) = a + bv \cos(t/2)$$

$$\begin{cases} x = r(t, v) \cos t \\ y = r(t, v) \sin t \\ z = bv \sin(t/2) \end{cases}$$

这里 a, b 是常量， $t \in [0, 2\pi], v \in [-1, 1]$ 。很容易把上面的式子翻译成 Mathematica 表达式：

```
Clear[r, x, y, z]
a=1.0;b=0.5;
r[t_,v_]=a+b*v*Cos[t/2]
x[t_,v_]=r[t,v]Cos[t]
y[t_,v_]=r[t,v]Sin[t]
z[t_,v_]=b*v*Sin[t/2]
ParametricPlot3D[{x[t,v],y[t,v],z[t,v]},{t,0,2Pi},{v,-1,1},PlotPoints->{20,8}]
```

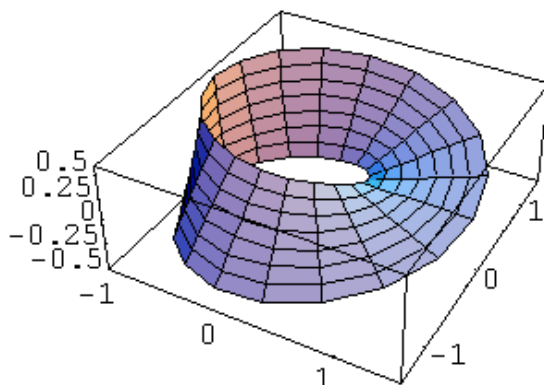


图 4.3.2.2 ParametricPlot3D[{x[t,v],y[t,v],z[t,v]},{t,0,2Pi},{v,-1,1},PlotPoints->{20,8}]

这里有几点需要说明：第一行清除有关变量(可能有)的值。第二行用了两个分号，当你在同一行里写多于一个表达式时，应当用分号把它们分隔开。系统依次计算以分号分隔的几个表达式，返回最后一个表达式的结果作为整个表达式的返回值。语句 $a=1.0;b=0.5;$ 计算表达式 $a=1.0$ 、 $b=0.5$ 和一个“空”表达式，返回“空”。在这里的几个式子里我们用星号表示乘法，你也可以用空格代替它们。

ParametricPlot3D 也可以画空间曲线，用法上的区别表现在它只有一个参数。例如，你可以画一条螺旋线：

```
ParametricPlot3D[{Cos[t],Sin[t],t/10},{t,0,20Pi},PlotPoints->150]
```

4.3.3 等值线图与密度图

画出一个二元函数的一组等值线也是研究函数性质的一种方法。例如，考虑欧拉 Γ 函数，它在 Mathematica 里用 Gamma 表示，也是一个内部函数。让我们画一下它代表的复映射的绝对值的图形：

```
Plot3D[Abs[Gamma[x+l*y]],{x,0.1,3},{y,-3,3}]
```

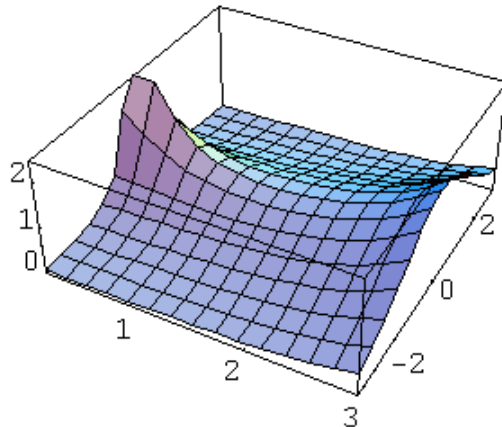


图 4.3.3.1 Plot3D[Abs[Gamma[x+l*y]],{x,0.1,3},{y,-3,3}]

这里的 Abs 是求绝对值的函数。再看看它的等值线圈，画等值线的工作由 ContourPlot 实现(z 值越大，颜色越淡。z 值越小则越黑；z 值越大则越白)：

```
ContourPlot[Abs[Gamma[x+l*y]],{x,0.1,3},{y,-3,3}]
```

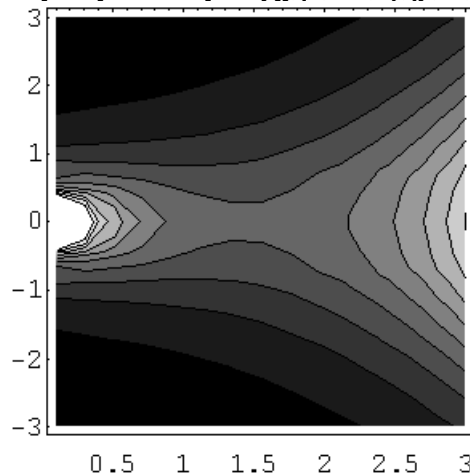


图 4.3.3.2 ContourPlot[Abs[Gamma[x+l*y]],{x,0.1,3},{y,-3,3}]

把这个图与前一个图对照，是不是对于这个函数的某些情况看的更清楚了呢？让我们再来看一个熟悉的图形：

```
ContourPlot[1-x^2-y^2,{x,-2,2},{y,-2,2},PlotPoints->30]
```

你看到了什么？是一组套着的圆。这使你想到了什么？你想到其中值为零的等位线是什么吗？它应该是单位圆。我们能不能画出单位圆呢？当然可以，只要在上述表达式中加上对函数值取值范围的限制：

```
ContourPlot[1-x^2-y^2,{x,-2,2},{y,-2,2},PlotPoints->300,Contours->{0},  
ContourShading->False]
```

现在只关心函数值为 0 的部分，你看到了期望的图形。可是你想到了画出的正是方程

$x^2 + y^2 = 1$ 所表示的 y 与 x 的隐函数关系的图形吗?可能我们发现了一种画一元隐函数的方法。总结一下,使用这种方法的步骤是:先把隐函数关系式 $g(x,y)=h(x,y)$ 变换为 $g(x,y)-h(x,y)=0$;然后用 `ContourPlot` 画等式左边表达式在等值线 0 的图形;一切就都好了。让我们任取一个函数试试这个方法。考虑: $x^3 + y^3 = 6xy$, 我们写出:

```
ContourPlot[x^3+y^3-6xy,{x,-4,4},{y,-4,4},PlotPoints->300,Contours->{0},
ContourShading->False]
```

使用 `ContourPlot` 时要注意,当函数变化比较剧烈复杂而计算的点不够多时,可能会看到不同等值线交叉的情况,这显然是不对的。出现这个问题的原因是计算中在一个小区域里碰到许多取值相近的点,根据已知信息系统无法确定应该把线连到哪里去,有时就弄错了。这时应该增大取点的密度,这样就可能做出正确的图。

画密度图的函数是 `DensityPlot`。让我们用 `Sin[x+y]` 作为例子:

```
DensityPlot[Sin[x+y],{x,0,Pi},{y,0,Pi}]
```

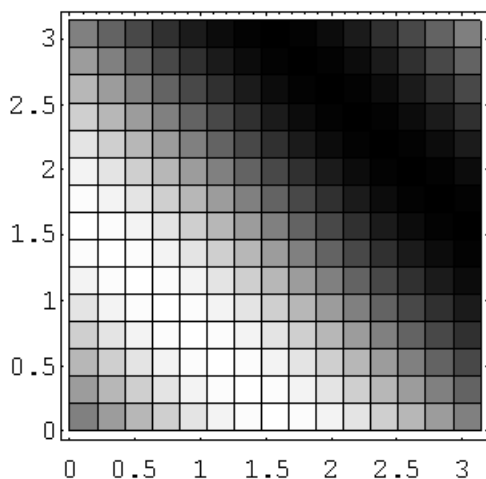


图 4.3.3.3 `DensityPlot[Sin[x+y],{x,0,Pi},{y,0,Pi}]`

这里同样可以加大取点的密度。另外,有人会觉得网格的存在影响观察,那么把它去掉:

```
DensityPlot[Sin[x+y],{x,0,Pi},{y,0,Pi}],PlotPoints->40,Mesh->False]
```

这个图是否也挺有意思?用这个函数可以方便地画出光线通过小孔或光栅形成的干涉图,你不妨自己试一试。顺便说一句,前面讲的 `Plot3D` 等函数也可以通过选择项 `Mesh->False` 去掉网格。

第五节 综合应用示例与练习

5.1 综合应用示例

例一：已知 $y = x \arcsin x$ ，用循环语句求其 1 阶、2 阶、…、10 阶导函数及其在 0 点的值。

解：

```
f[x_]:=x ArcSin[x]
```

```
For[i=1,i<11,
```

```
Print["The ", i, " DEGREE derivative is: ", D[f[x], {x, i}], ", the value is ",  
D[f[x],{x,i}]/.x->0];i++]
```

例二：在 0 点分别用 5, 10, 15, 20 阶 Taylor 级数展开函数 $f(x) = \frac{\sin(x^2)}{x}$ ，并在一张图上绘制出原函数及各展开图像（ $x \in [-\pi, \pi]$ ）。

解：

```
Clear[Taylor,f5,f10,f15,f20,p0,p1,p2,p3,p4]
```

```
Taylor[x_,n_]:=Normal[Series[Sin[x^2]/x,{x,0,n}]]
```

```
f5[x_]=Taylor[x,5]
```

```
f10[x_]=Taylor[x,10]
```

```
f15[x_]=Taylor[x,15]
```

```
f20[x_]=Taylor[x,20]
```

```
t=Pi;
```

```
p0=Plot[f5[x],{x,-t,t},PlotStyle->RGBColor[1,1,0]]
```

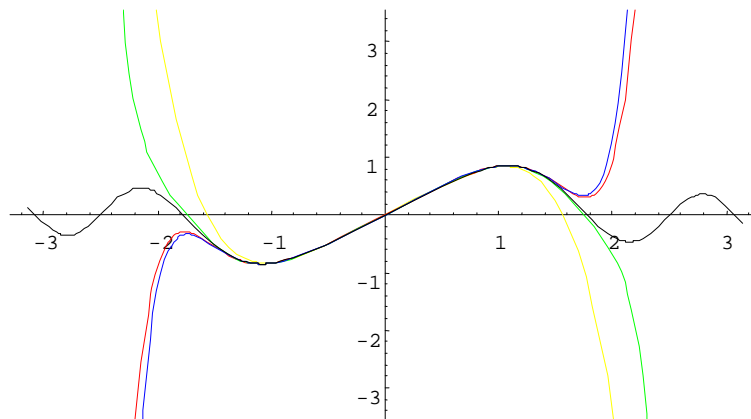
```
p1=Plot[f10[x],{x,-t,t},PlotStyle->RGBColor[1,0,0]]
```

```
p2=Plot[f15[x],{x,-t,t},PlotStyle->RGBColor[0,1,0]]
```

```
p3=Plot[f20[x],{x,-t,t},PlotStyle->RGBColor[0,0,1]]
```

```
p4=Plot[Sin[x^2]/x,{x,-t,t}]
```

```
Show[p0,p1,p2,p3,p4]
```



例三：用矩形法求广义定积分 $\int_0^{+\infty} \frac{\sin x}{x} dx$ 的近似值，并与精确值作比较。

解： $\int_0^{+\infty} \frac{\sin x}{x} dx = \int_0^1 \frac{\sin x}{x} dx + \int_1^{+\infty} \frac{\sin x}{x} dx$,

设 $x = \frac{1}{t}$, 则： $dx = -\frac{1}{t^2} dt$

$$\int_1^{+\infty} \frac{\sin x}{x} dx = \int_1^0 \frac{\sin \frac{1}{t}}{\frac{1}{t}} \left(-\frac{1}{t^2}\right) dt = \int_0^1 \frac{\sin \frac{1}{t}}{t} dt = \int_0^1 \frac{\sin \frac{1}{x}}{x} dx ,$$

故：

$$\int_0^{+\infty} \frac{\sin x}{x} dx = \int_0^1 \frac{\sin x}{x} dx + \int_0^1 \frac{\sin \frac{1}{x}}{x} dx$$

```
Clear[f, x, f1, f2, re, tes, n, s1, s2, diff]
f[x_]:=Sin[x]/x
re=N[Integrate[f[x], {x, 0, Infinity}]]
f1[t_]:=f[x]/.x->(1/t)
f2[t_]:=f1[t]*(1/t^2)
s1[n_]:=N[Sum[f2[i/n]/n, {i, 1, n-1}]]
s2[n_]:=N[Sum[f[i/n]/n, {i, 1, n-1}]]
tes=N[s1[1000]+s2[1000]]
diff=N[re-tes]
```

例四：用雅可比迭代法求线性方程组的解：

对线性方程组 $Ax=b$, A 为非奇异阵且 $a_{ii} \neq 0$ 。将 A 分解为 $A=D-L-U$, 其中 D 为 A 的对角元组成的对角阵, L 为 $(-A)$ 的下三角矩阵, U 为 $(-A)$ 的上三角矩阵, L 和 U 的对角元均为 0。令 $B_0 = I - D^{-1}A = D^{-1}(L+U)$, $f = D^{-1}b$, 则对上述方程组, 先选取初始解向量, 然后按照以下迭代公式进行迭代直至 $|x^{(k+1)} - x^{(k)}| < \varepsilon$

$$\begin{cases} x^{(0)} \text{ 为初始向量} \\ x^{(k+1)} = B_0 x^{(k)} + f \end{cases}$$

此迭代法称为求线性方程组的雅可比迭代法。

取 $\varepsilon = 0.0001$, 对方程组:

$$\begin{bmatrix} 8 & -3 & 2 \\ 4 & 11 & -1 \\ 6 & 3 & 12 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 33 \\ 36 \end{bmatrix}$$

应用此迭代法, 求出方程的解, 并与用直接法计算出的方程的解 $A^{-1}b$ 比较。

解：

```

A={{8, -3, 2},{4, 11, -1},{6, 3, 12}}
D1=DiagonalMatrix[{A[[1,1]], A[[2,2]], A[[3,3]]}]
b={20, 33, 36}
B0=IdentityMatrix[3]-Inverse[D1].A
f=Inverse[D1].b
x1={0, 0, 0};
x2={1, 1, 1};
count = 1;
While[
  Max[Abs[x1-x2]]>10^-4,
  If[Mod[count,2]==1,
    x2 = B0 . x1 + f,
    x1 = B0 . x2 + f
  ];
  count++;
]
x1 //N
Inverse[A].b

```

例五：海底地形图的绘制

为了在一新的海域建立新的航道，需要知道海底深度，并绘制海底地形图。现使用一艘勘测船，对一矩形区域内均匀网格上海底深度进行勘测，得到以下数据（ x 和 y 为点的坐标）：

	$x=0$	$x=20$	$x=40$	$x=60$	$x=80$	$x=100$
$y=0$	8.73	8.32	8.00	7.97	7.77	7.99
$y=40$	8.94	8.78	6.87	7.22	7.92	7.99
$y=80$	8.88	8.91	4.21	6.38	7.37	7.95
$y=120$	8.79	8.79	8.54	5.82	4.88	7.97
$y=160$	8.75	8.80	7.91	5.80	4.77	7.85
$y=200$	8.52	8.31	6.61	6.06	6.49	7.97

假定海底经过海水冲刷，充分光滑，也没有礁石和洞穴。请用插值方法得到此海域内海底深度数据并绘制海底地形图和等高线图。

解：

```

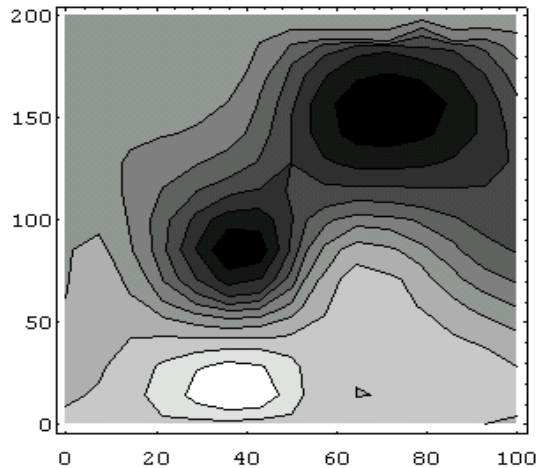
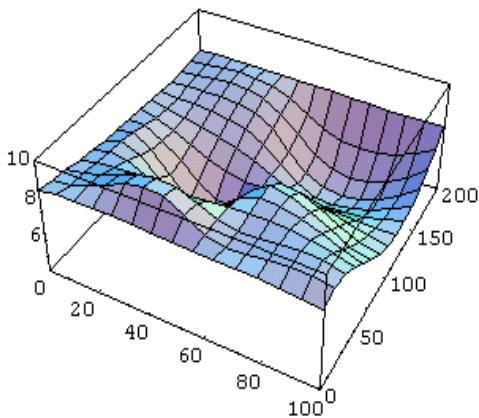
x=Table[20*i, {i, 0, 5}];
y=Table[40*i, {i, 0, 5}];
z = {
  {8.73, 8.32, 8.00, 7.97, 7.77, 7.99},
  {8.94, 8.78, 6.87, 7.22, 7.92, 7.99},
  {8.88, 8.91, 4.21, 6.38, 7.37, 7.95},
  {8.79, 8.79, 8.54, 5.82, 4.88, 7.97},

```

```

      {8.75, 8.80, 7.91, 5.80, 4.77, 7.85},
      {8.52, 8.31, 6.61, 6.06, 6.49, 7.97}
};
data = Table[{x[[i]], y[[j]], z[[i, j]]}, {i, 6}, {j, 6}];
data=Flatten[data, 1];
f=Interpolation[data]
Plot3D[f[x, y], {x, 0, 100}, {y, 0, 200}]
ContourPlot[f[x, y], {x, 0, 100}, {y, 0, 200}]

```



5.2 综合应用练习

- 对 $n=100$ 和 1000 验证斯特林公式: $n! = \sqrt{2n\pi} \frac{n^n}{e^n}$, 确定其相对与绝对误差;
- 对 $1 + \frac{1}{1!} + \frac{1}{2!} + \dots = e$, 试确定这个级数的前多少项之和, 精确到 e 的 10 位有效数字。
对 $\sin 1$ 和 $\cos 1$ 作同样的事情。你也可以试一下收敛到 e 的另一个序列 $(1 + \frac{1}{n})^n$;
- 展开 $(1 + x + 2xy^2 + y)^{20}$, 它有多少项? 最大系数是多少? 最高幂次是多少?
- 分解因式 $x^n - 1$, $n=20, 50, 342, 678$;
- 对有理式: $\frac{-4x + x^2}{-x + x^2} + \frac{-4 + 3x + x^2}{-1 + x^2}$ 进行通分, 因式分解, 拆分, 约分;
- 计算 $6 + 17x + 25x^2 + 45x^5$ 与 $1 + 2x^2$ 的带余除法的商与余数;
- 构造以下的表:
{ 1, 4, 9, 16, ..., 10000 };
{{1,2,3,...,100},{101,102,103,...,200},{901,902,903,...,1000}};
- 计算向量 $a=\{1,2,3,4\}$ 与 $b=\{5,6,7,8\}$ 的内积与外积。(即 ab' 与 $a'b$);
- 集合 $L=\{1,3,5,34,54,78,45,2,35,67\}$, 计算 L 中元素的和, 积;

10. 用 NestList 函数构造集合:

{ Sin(2), Sin(Sin(2)), Sin(Sin(Sin(2))), Sin(Sin(Sin(Sin(2))))},
Sin(Sin(Sin(Sin(Sin(2)))));

11. 计算曲线 $\sin(x)$, $0 \leq x \leq 2\pi$ 绕 x 轴旋转一周所围部分的体积;

12. 求解下面的不定积分和定积分:

$$\int \frac{x^3 + 3x^2 - 5}{(x^2 - 2x - 6)(x^3 + x + 1)} dx \quad \int a^x \sin x (\cos x)^2 dx$$
$$\int_0^{2\pi} \frac{dx}{5 + 3\sin(x)} \quad \int_a^{2a} \frac{\sqrt{x^2 - a^2}}{x^4} dx$$
$$\int_0^{\pi/2} \frac{d\varphi}{12 + 13\cos(\varphi)} \quad \int_3^{\pi/2} \frac{(x-2)^{\frac{2}{3}}}{(x-2)^{\frac{4}{3}} + 3} dx$$

13. 已知 $y = x \operatorname{arctg} x$, 求其高阶导数 $y^{(100)}$ 及其在 0 点的值;

14. 用矩形法求广义定积分 $\int_0^{+\infty} e^{-\frac{x^2}{2}} dx$ 的近似值, 并与精确值作比较;

15. 在 0 点分别用 3, 6, 9 阶 Taylor 级数展开函数 $f(x) = \frac{\sin(x)}{2 + \cos(x)}$, 并在一张图上绘制出原函数及各展开图像 ($x \in [-\pi, \pi]$);

16. 求 $f(x) = x^2 \ln x$ 在 $x_0 = 1$ 处的 5 阶泰勒展开式;

17. 求 $f(x) = e^{-x^2/2}$ 的 4 阶马克劳林展开式;

18. 求方程 $x^2 = \sin x$ 的最小正实根;

19. 求方程 $x^2 = \cos x$ 的最大负实根;

20. 求方程 $x = \cot x$ 的最大负实根;

21. 求方程 $x = \tan x$ 的最小正实根;

22. 用 Exp 函数产生一组数据, 然后用 Random 函数加上随机扰动。用一次, 二次, 三次函数, Exp 函数来拟合它;

23. 用一条语句在一张图上绘制函数 $y = x^n$, $n=1,2,3,4,5,6$ 的图形;

24. 画出参数曲线:

1). $x=\sin(3t)$, $y=\sin(5t)$, $t \in [0, \pi]$

2). $x=a \cos(1+\cos(t))$, $y=a \sin(1+\cos(t))$, a 任意取值, $t \in [0, \pi]$

3) $x=\cos(t)+5\cos(3t)$, $y=6\cos(t)-5\cos(3t)$;

25. 用命令给出前 30 个素数，并用平面折线连接加以示意；
26. 用命令给出斐波那契数列的前 10 项，并画出其平面散点图；
27. 用命令给出数列 $(1 - \frac{1}{2^2})(1 - \frac{1}{2^3})(1 - \frac{1}{2^4}) \cdots (1 - \frac{1}{2^n})$ 的前 100 项，画出平面散点图并估计其极限值；
28. 作 $f(x) = \begin{cases} x^3 + 3, & x > 0 \\ x^3 - 3, & x < 0 \end{cases}$, $x \in [-2, 2]$ 的图形；
29. 作出函数 $f(x) = \sqrt[3]{x}$, $x \in [-10, 10]$ 的图象；
30. 选择合适的作图区间，作出函数 $y = \sin x + \frac{1}{100} \cos(100x)$ 的图象；
31. 作出圆环面 $(\sqrt{x^2 + y^2} - R)^2 + z^2 = r^2$, $R = 6, r = 2$ 的图象；
提示：圆环面的参数方程为：
- $$\begin{cases} x = (R + r \cos u) \cos v \\ y = (R + r \cos u) \sin v \\ z = r \sin u, \end{cases} \quad u \in [0, 2\pi], v \in [0, 2\pi]$$
32. 作出圆环面 $(\sqrt{x^2 + y^2} - R)^2 + z^2 = r^2$, $R = 6, r = 2$ 在第一卦象部分的图象。