
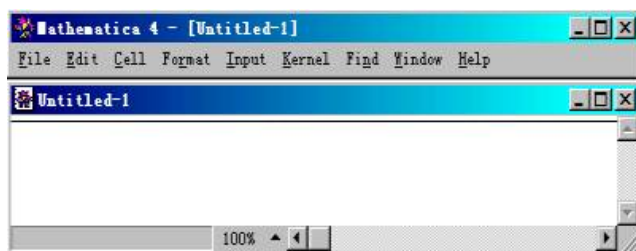


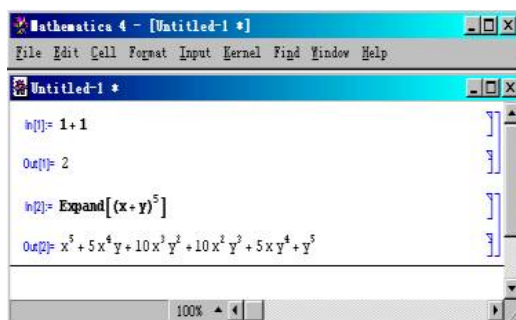
小木虫

Mathematica 是美国 Wolfram 研究公司生产的一种数学分析型的软件，以符号计算见长，也具有高精度的数值计算功能和强大的图形功能。

假设在 Windows 环境下已安装好 Mathematica4.0，启动 Windows 后，在“开始”菜单的“程序”中单击  Mathematica 4，就启动了 Mathematica4.0，在屏幕上显示如图的 Notebook 窗口，系统暂时取名 Untitled-1，直到用户保存时重新命名为止



输入 $1+1$ ，然后按下 **Shift+Enter** 键，这时系统开始计算并输出计算结果，并给输入和输出附上次序标识 **In[1]**和 **Out[1]**，注意 **In[1]**是计算后才出现的；再输入第二个表达式，要求系统将一个二项式展开，按 **Shift+Enter** 输出计算结果后，系统分别将其标识为 **In[2]**和 **Out[2]**。如图



在 Mathematica 的 Notebook 界面下，可以用这种交互方式完成各种运算，如函数作图，求极限、解方程等，也可以用它编写像 C 那样的结构化程序。在 Mathematica 系统中定义了许多功能强大的函数，我们称之为内建函数（built-in function），直接调用这些函数可以取到事半功倍的效果。这些函数分为两类，一类是数学意义上的函数，如：绝对值函数 **Abs[x]**，正弦函数 **Sin[x]**，余弦函数 **Cos[x]**，以 e 为底的对数函数 **Log[x]**，以 a 为底的对数函数 **Log[a,x]**等；第二类是命令意义上的函数，如作函数图形的函数 **Plot[f[x],{x,xmin,xmax}]**，解方程函数 **Solve[eqn,x]**，求导函数 **D[f[x],x]**等。

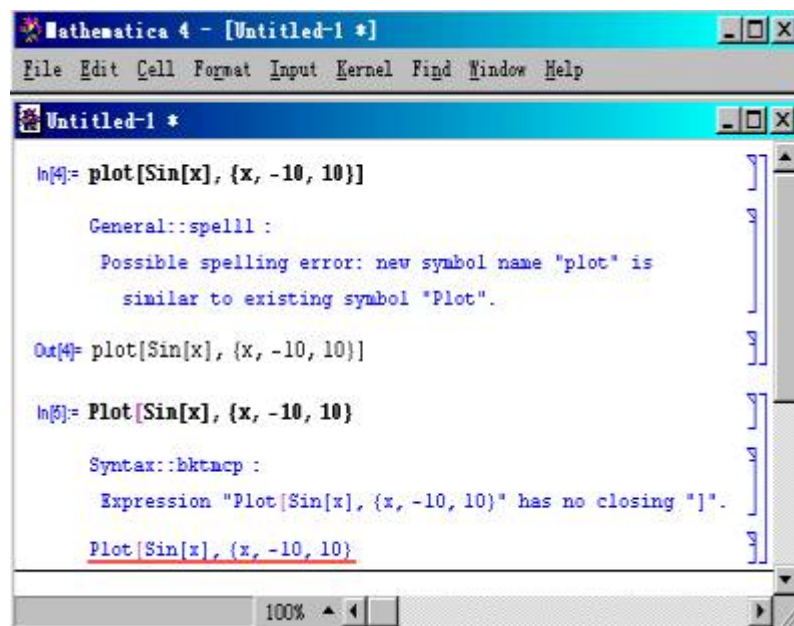


必须注意的是：

Mathematica 严格区分大小写，一般地，内建函数的首写字母必须大写，有时一个函数名是由几个单词构成，则每个单词的首写字母也必须大写，如：求局部极小值函数 **FindMinimum[f[x],{x,x0}]**等。第二点要注意的是，在 Mathematica 中，函数名和自变量之间的分隔符是用方括号“**[]**”，而不是一般数学书上用的圆括号“**()**”，初学者很容易犯这类错误。

如果输入了不合语法规则的表达式，系统会显示出错信息，并且不给出计算结果，例如：要画正弦函数在区间 $[-10, 10]$ 上的图形，输入 **plot[Sin[x],{x,-10,10}]**，则系统提示“可能有拼写错误，新符

号‘plot’很像已经存在的符号‘Plot’”，实际上，系统作图命令“Plot”第一个字母必须大写，一般地，系统内建函数首写字母都要大写。再输入 Plot[Sin[x],{x,-10,10}，系统又提示缺少右方括号，并且将不配对的括号用蓝色显示，如图



一个表达式只有准确无误，方能得出正确结果。学会看系统出错信息能帮助我们较快找出错误，提高工作效率。完成各种计算后，点击 File->Exit 退出，如果文件未存盘，系统提示用户存盘，文件名以“.nb”作为后缀，称为 Notebook 文件。以后想使用本次保存的结果时可以通过 File->Open 菜单读入，也可以直接双击它，系统自动调用 Mathematica 将它打开。

1.1.2 表达式的输入

Mathematica 提供了多种输入数学表达式的方法。除了用键盘输入外，还可以使用工具样或者快捷方式键入运算符、矩阵或数学表达式。

1. 数学表达式二维格式的输入

Mathematica 提供了两种格式的数学表达式。形如 $x/(2+3x)+y*(x-w)$ 的称为一维格式，形如

$$\frac{x}{2+3x} + \frac{y}{x-w}$$

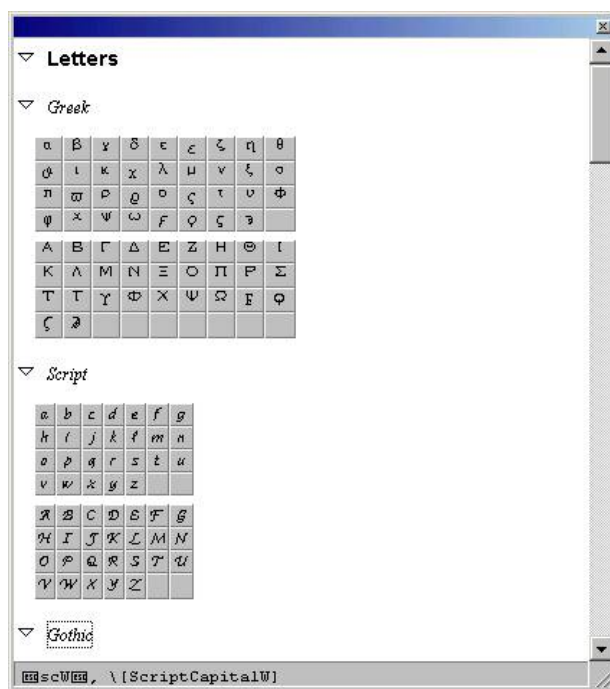
的称为二维格式。

你可以使用快捷方式输入二维格式，也可用基本输入工具栏输入二维格式。下面列出了用快捷方式输入二维格式的方法

数学运算

数学表达式

按键



单击符号后即可输入。

1.2.Mathematica 的联机帮助系统

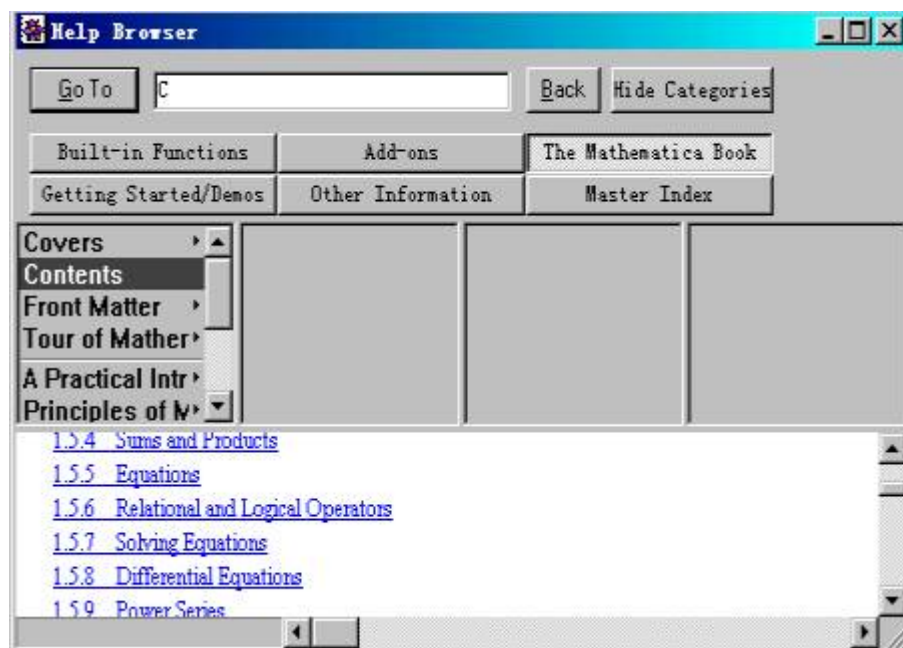
用 Mathematica 的过程中，常常需要了解一个命令的详细用法，或者想知系统中是否有完成某一计算的命令，联机帮助系统永远是最详细、最方便的资料库。

1.获取函数和命令的帮助

在 Notebook 界面下，用 `?` 或 `??` 可向系统查询运算符、函数和命令的定义和用法，获取简单而直接的帮助信息。例如，向系统查询作图函数 `Plot` 命令的用法？`Plot` 系统将给出调用 `Plot` 的格式以及 `Plot` 命令的功能（如果用两个问号“`??`”，则信息会更详细一些）。`?Plot*` 给出所有以 `Plot` 这四个字母开头的命令

2.Help 菜单

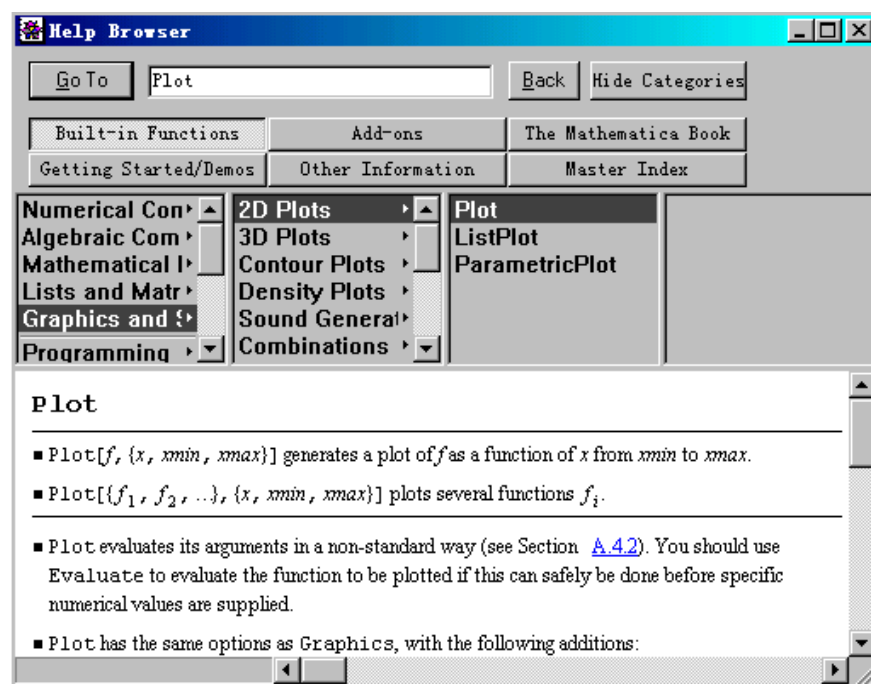
任何时候都可以通过按 `F1` 键或点击帮助菜单项 `Help Browser`，调出帮助菜单，如图



所示，其中的各按钮用途如表所示

Built-in Function	内建函数，按数值计算、代数计算、图形和编程分类存放
Add-ons	有程序包（Standard Packages）MathLink Library 等内容
The Mathematica Book	一本完整的 Mathematica 使用手册
Getting Started/Demos	初学者入门指南和多种演示
Other Information	菜单命令的快捷键，二维输入格式等
Master Index	按字母命令给出命令、函数和选项的索引表

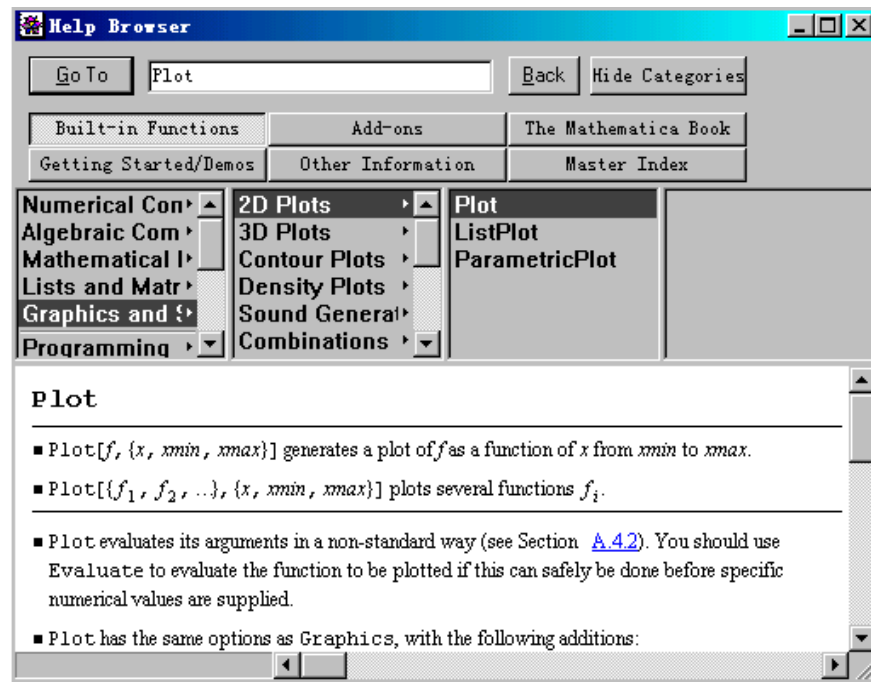
如果要查找 Mathematica 中具有某个功能的函数,可以通过帮助菜单中的 Mathematica 使用手册,通过其目录索引可以快速定位到自己要找的帮助信息。例如: 需要查找 Mathematica 中有关解方程的命令,单击“[The Mathematica Book](#)”按钮,再单击“[Contents](#)”,在目录中找到有关解方程的节次,点击相应的超链接,有关内容的详细说明就马上调出来了。如果知道具体的函数名,但不知其详细使用说明,可以在命令按钮 [Goto](#) 右边的文本框中键入函数名,按回车键后就显示有关函数的定义、例题和相关联的章节。例如,要查找函数 Plot 的用法,只要在文本框中键入 Plot,按回车键后显示如图的窗口,



再按回车键，则显示 Plot 函数的详细用法和例题。如果已经确知 Mathematica 中有具有某个功能的函数，但不知具体函数名，可以点击 Built-in Functions 按钮，再按功能分类从粗到细一步一步找到具体的函数，例如，要找画一元函数图形的函数，点击 Built-in Functions ->Graphics and Sound->2D Plots->Plot，找到 Plot 的帮助信息。

如果知道具体的函数名，但不知其详细使用说明，可以在命令按钮 Goto 右边的文本框中键入函数名，按回车键后就显示有关函数的定义、例题和相关联的章节。例如，要查找函数 Plot 的用法，只要在文本框中键入 Plot，按回车键后显示如图 1-5 的窗口，再按回车键，则显示 Plot 函数的详细用法和例题。

如果已经确知 Mathematica 中有具有某个功能的函数，但不知具体函数名，可以点击 Built-in Functions 按钮，再按功能分类从粗到细一步一步找到具体的函数，例如，要找画一元函数图形的函数，点击 Built-in Functions ->Graphics and Sound->2D Plots->Plot，找到 Plot 的帮助信息。



2.1 数据类型和常数

1 数值类型

在 Mathematic 中，基本的数值类型有四种：整数，有理数、实数和复数

如果你的计算机的内存足够大,Mathematica 可以表示任意长度的精确实数，而不受所用的计算机字长的影响。整数与整数的计算结果仍是精确的整数或是 有理数。例如：2 的 100 次方是一个 31 位的整数：

```
In[1]:=2^100
```

```
Out[1]=1267650600228228229401496703205376
```

在 Mathematica 中允许使用分数，也就是用有理数表示化简过的分数。当两个整数相除而又不能整除时，系统就用有理数来表示，即有理数是由两个整数的比来组成如：

```
Ln[2]:=12345/5555
```

```
Out[2]=2469/1111
```

实数是用浮点数表示的，Mathematica 实数的有效位可取任意位数，是一种具有任意精确度的近似实数，当然在计算的时候也可以控制实数的精度。实数有两种表示方法：一种是小数点另外一种是用指数方法表示的。如：

```
In[3]:=0.239998
```

```
Out[3]=0.23998
```



```
In[4]:=0.12*10^11
```

```
Out[4]=0.12*10^11
```

实数也可以与整数，有理数进行混合运算，结果还是一个实数。

```
Ln[5]:=2+1/4+0.5
```

```
Out[5]=2.75
```

复数是由实部和虚部组成。实部和虚部可以用整数，实数，有理数表示。在 Mathematica 中，用 i 表示虚数单位如：

```
Ln[6]:=3+0.7i
```

```
Out[6]:=3+0.7i
```

2. 不同类型数的转换

在 Mathematica 的不同应用中，通常对数字的类型要求是不同的。例如在公式推导中的数字常用整数或有理数表示，而在数值计算中的数字常用实数表示。在一般情况下在输出行 Out[n] 中，系统根据输入行 In[n] 的数字类型对计算结果做出相应的处理。如果有一些特殊的要求，就要进行数据类型转换。

在 Mathematica 中提供以下几个函数达到转换的目的：

N[x] 将 x 转换成实数

N[x,n] 将 x 转换成近似实数，精度为 n

Rationalize[x] 给出 x 的有理数近似值

Rationalize[x,dx] 给出 x 的有理数近似值，误差小于 dx

[举例]

```
In[1]=N[5.3,20]
```

```
Out[1]=1.66666666666666666667
```

```
In[2]:=N[%,10]
```

```
Out[2]=1.66666667
```

二行输出是把上面计算的结果变为 10 位精度的数字。% 表示上一输出结果。

```
Ln[3]=Rationalize[%]
```


Out[3]=5/3

3.数学常数

Mathematica 中定义了一些常见的数学常数，这些数学常数都是精确数，例如表示圆周率。

Pi	表示 π = 3.14159.....
E	自然对数的底，e=2.71828.....
Degree	$\pi/180$
i	虚数单位
Infinity	无穷大 ∞
- infinity	负的无穷大 $-\infty$
GondenRatio	黄金分割数 0.61803

数学常数可用在公式推导和数值计算中。在数值计算中表示精确值：如：

In[1]:=Pi^2

Out[1]= π^2

In[2]:=Pi^2//N

Out[2]=9.86961

4.数的输出形式

在数的输出中可以使用转换函数进行不同数据类型和精度的转换。另外对一些特殊要求的格式还可以使用如下的格式函数：

NumberForm[expr,n] 以 n 位精度的实数形式输出实数 expr

ScientificFormat[expr] 以科学记数法输出实数 expr

EngineergForm[expr] 以工程记数法输出实数 expr

例如：

In[1]:=N[Pi^30,30]

```
8.21289330402749581586503585434 × 1014
```

```
In[2]:=NumberForm[%,10]
```

```
Out[2]//NumberForm=8.212893304 × 1014
```

下面的函数输出幂值可被 3 整除的实数

```
Ln[3]=EngineeringForm[%%]
```

```
Out[6]//EngineeringForm= 821.289 × 1012
```

1. 变量的命名

Mathematica 中内部函数和命令都是以大写字母开始的标示符。为了不会与它门混淆，我们自定义的变量应该是以小写字母开始，后跟数字和字母的组合，长度不限。例如：a12,ast,aST 都是合法的，而 12a, z*a 是非法的。另外在 Mathematica 中的变量是区分大小写的 在 Mathematica 中，变量不仅可以存放一个数值，还可以存放表达式或复杂的算式。

2. 给变量赋值

在 Mathematica 中用等号 = 为变量赋值。同一个变量可以表示一个数值，一个数组，一个表达式，甚至一个图形。如：

```
Ln[1]:=x=3
```

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

```
Ln[2]:=x^2+2x
```

```
Out[2]=15
```

```
Ln[3]:=x=%+1
```

```
Out[3]=16
```

对不同的变量可同时赋不同的值例如：

```
Ln[4]:={u,v,w}={1,2,3}
```

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

```
Ln[5]:=2u+3v+w
```

```
Out[5]=11
```

对于已定义的变量，当你不再使用它是，为防止变量值的混淆，可以随时用 `=.` 清除他的值，如果变量本身也要清除用函数 `Clear[x]` 例如

```
ln[6]:=u=.
```

```
ln[7]:=2u+v
```

```
Out[7]=2+2u
```

3.变量的替换

在给定一个表达式时其中的变量可能取不同的值，这是可用变量替换来计算表达式的不同值。方法为用 `expr/.` 例如：

```
Ln[1]:=f=x/2+1
```

```
Out[1]=  $1 + \frac{x}{2}$ 
```

```
Ln[2]:=f/.x->1
```

```
Out[2]=  $\frac{3}{2}$ 
```

```
Ln[3]:=f/.->2
```

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

如果表达式中有多个变量也可以同时替换方法为例如有两个：

```
expr/.{x->xval,y->val}
```

```
Ln[4]:=(x+y)(x-y)^2/.{x->3,y->1-a}
```

```
Out[4]=  $(4-a)(2+a)^2$ 
```

2.3 函数



1. 系统函数

在 Mathematic 中定义了大量的数学函数可以直接调用，这些函数其名称一般表达了一定的意义，可以帮助我们理解。下面是几个常用的函数：

Floor[x]	不比 x 大的最大整数
Ceiling[x]	不比 x 小的最小整数
Sign[x]	符号函数
Round[x]	接近 x 的整数
Abs[x]	x 绝对值
Max[x1,x2,x3.....]	x1,x2,x3.....中的最大值
Min[x1,x2,x3.....]	x1,x2,x3.....中的最小值
Random[]	0~1 之间的随机函数
Random[Real,xmax]	0~xmax 之间的随机函数
Random[Real,{xmin,xmax}]	xmin~xmax 之间的随机函数
Exp[x]	指数函数
Log[x]	自然对数函数 $\ln x$
Log[b,x]	以 b 为底的对数函数
Sin[x],Cos[x],Tan[x],Csc[x],Sec[x],Cot[x]	三角函数（变量是以弧度为单位的）
Sinh[x],Cosh[x],Tanh[x],Csch[x],Sech[x],Coth[x]	双曲函数
ArcSech[x],ArcCoth[x]	双曲函数
Mod[m,n]	m 被 n 整除的余数，余数与 n 的符相同
Quotient[m,n]	m/n 的整数部分
GCD[n1,n2,n3.....]或 GCD[s]	n1,n2,...的最大公约数，s 为一数集合
LCM[n1,n2.....]或 LCM[s]	n1,n2.....的最大公倍数，s 为数据集合
N!	n 的阶程
N!!	n 的双阶程

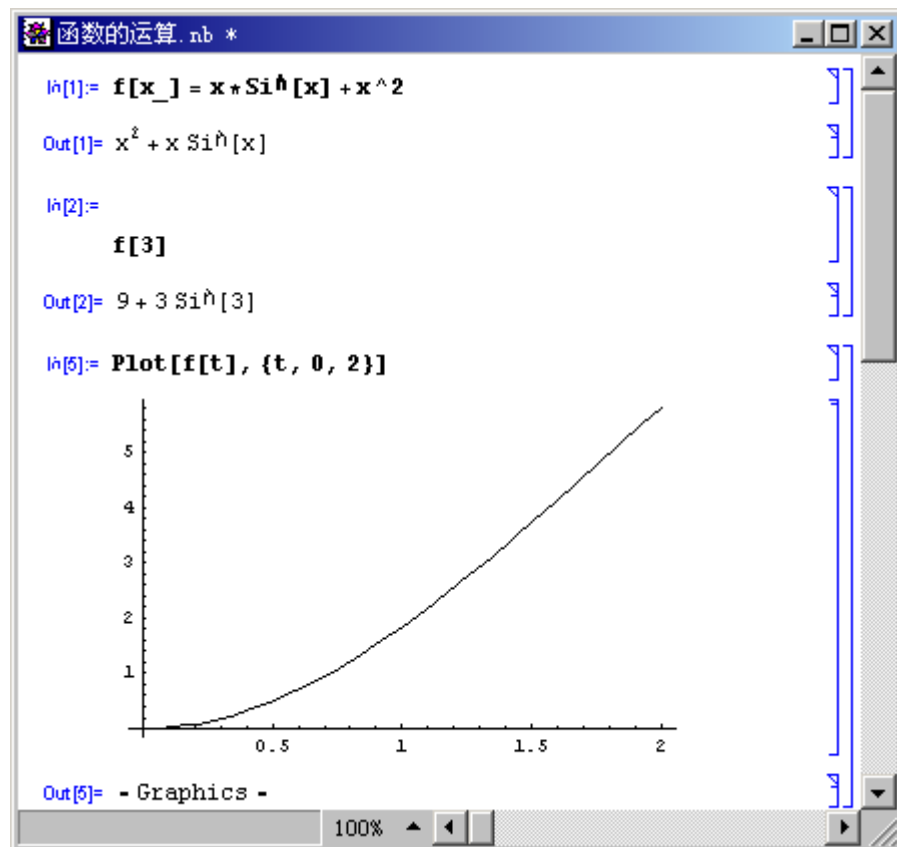
Mathematica 中的函数与数学上的函数有些不同的地方，Mathematica 中函数是一个具有独立功能的程序模块，可以直接被调用。同时每一函数也可以包括一个，或多个参数，也可以没有参数。参数的数据类型也比较复杂。更加详细的可以参看系统的帮助，了解各个函数的功能和使用方法是学习 Mathematica 软件的基础

2. 函数的定义

(1) 函数的立即定义

立即定义函数的语法如下 $f[x_]=\text{expr}$ 函数名为 f , 自变量为 x , expr 是表达式。在执行时会把 expr 中的 x 都换为 f 的自变量 x (不是 $x_$)。函数的自变量具有局部性, 只对所在的函数起作用。函数执行结束后也就没有了, 不会改变其它全局定义的同名变量的值。请看下面的例子

定义函数 $f(x)=x*\text{Sin}x+x^2$ 对定义的函数我们可以求函数值, 也可绘制它的图形。



对于定义的函数我们可以使用命令 `Clear[f]` 清除掉而 `Remove[f]` 则从系统中删除该函数。

(2). 多变量函数的定义

也可以定义多个变量的函数, 格式为 $f[x_, y_, z_, \dots] = \text{expr}$ 自变量为 x, y, z, \dots , 相应的 expr 中的自变量

会被替换。例如定义函数

$$f(x, y) = xy + y \cos x$$

```

函数的运算.nb *
In[9]:= f[x_, y_] = x * y + y * Cos[x]

Out[9]= x y + y Cos[x]

In[10]:= f[2, 3]

Out[10]= 6 + 3 Cos[2]

```

(3) 延迟定义函数

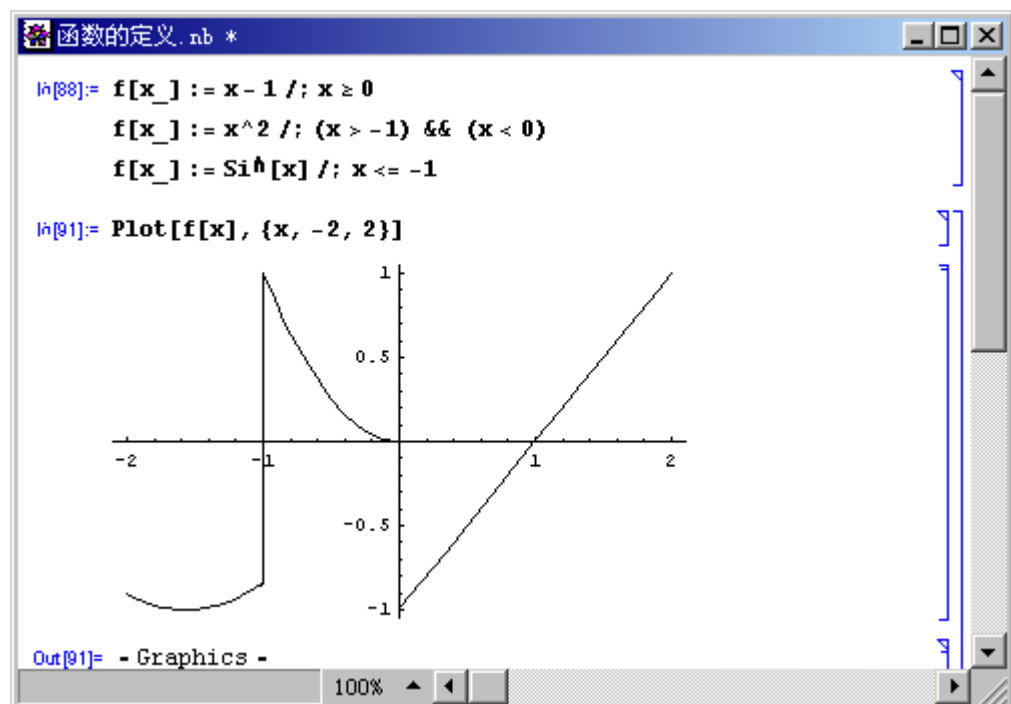
延迟定义函数从定义方法上与即时定义的区别为“=”与“:=”延迟定义的格式为 `f[x_] := expr` 其他操作基本相同。那么延迟定义和即时定义的主要区别是什么？即时定义函数在输入函数后立即定义函数并存放在内存中并可直接调用。延迟定义只是在调用函数时才真正定义函数。

(4) 使用条件运算符定义和If命令定义函数

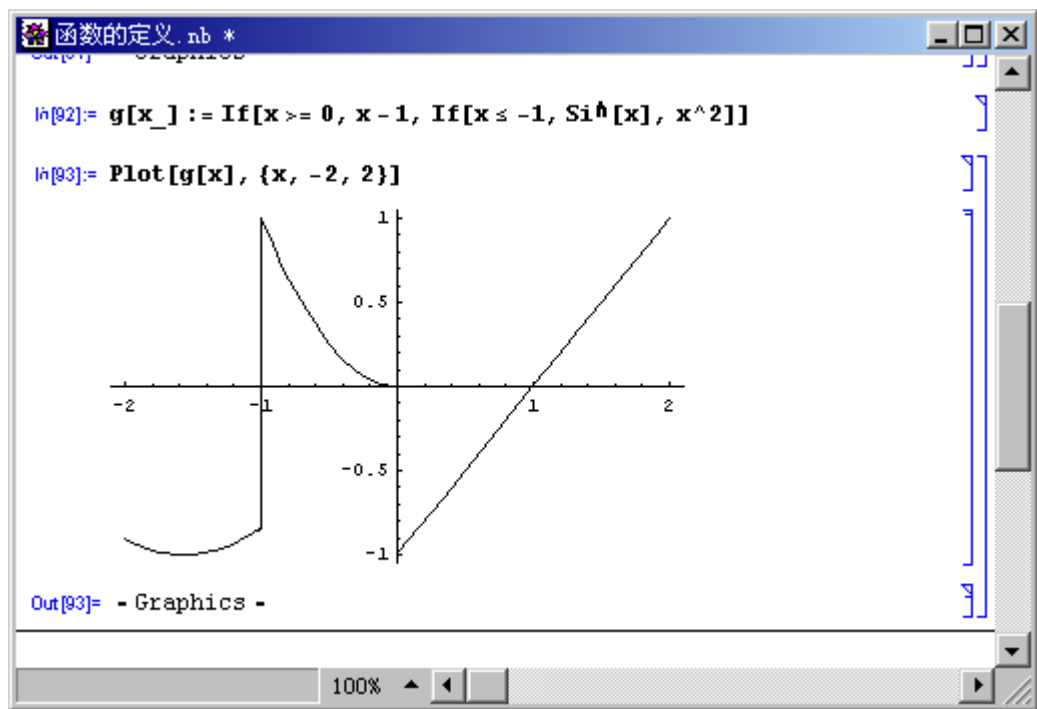
如果要定义如：

$$f(x) = \begin{cases} x-1 & x > 0 \\ x^2 & 0 \geq x > -1 \\ \sin x & x \leq -1 \end{cases}$$

这样的分段函数应该如何定义，显然要根据 x 的不同值给出不同的表达式。一种办法是使用条件运算符，基本格式为 `f[x_] := expr/condition`，当 `condition` 条件满足时才把 `expr` 赋给 `f`。下面定义方法，通过图形可以验证所定义函数的正确性



当然使用If命令也可以定义上面的函数，If语句的格式为 `If[条件, 值1, 值2]` 如果条件成立取“值1”，否则取“值2”，下面用If语句的定义结果



可以看出用If定义的函数g(x)和前面函数f(x)相同，这里使用了两个If嵌套。逻辑性比较强关于其他的条件命令的进一步讨论请看后面的章节

2.4 表



将一些相互关联的元素放在一起，使它们成为一个整体。既可以对整体操作，也可以对整体中的一个元素单独进行操作。在 Mathematica 中这样的数据结构就称作表（List）。表主要有三个用法：表 {a,b,c} 可以表示一个向量；表 {{a,b},{c,d}} 可表示一个矩阵。

1. 建表

在表中元素较少时，可以采取直接列表的方式列出表中的元素，如 {1,2,3}。请看下面的操作

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

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

下面是符号表达式的列表

In[2]:=1+%x+x^%

Out[2]={1+2x,1+2x+x^{2,1+3x+x2}}

下面是对列表中的表达式对x求导


```
Ln[3]:=D[%,x]
```

```
Out[3]={2,2+2x,3+2x}
```

```
Ln[4]:=%/x>1
```

```
Out[4]={2,4,5}
```

如果表中的元素较多时，可以用建表函数进行建表。

`Table[f,{l,min,max,step}]` 以step为步长给出f的数值表，l由min变到max。

`Table[f,{min,max}]` 给出f的数值表，l由min变到max 步长为 1

`Table[f,max]` 给出max个f的表

`Table[f,{l,imin,imax},{j,jmin,jmax},...]` 生成一个多维表

`TableForm[list]` 以表格格式显示一个表

`Range[n]` 生成一个{1,2,...,n}的列表

`Range[n1,n2,d]` 生成{n1,n1+d,n1+2d,...,n2}的列表

下面给出x乘i的值的表，i的变化范围为[2,6]:

```
Ln[1]:=Table[x*i,{i,2,6}]
```

```
Out[1]={2x,3x,4x,5x,6x}
```

```
Ln[2]:=Table[x^2,{4}]
```

```
Out[2]={x2,x2,x2,x2}
```

用Range函数生成一个序列数

```
Ln[3]:=Range[10]
```

```
Out[3]={1,2,3,4,5,6,7,8,9,10}
```

下面这个序列是以步长为 2，范围从 8 到 20

```
Ln[4]:=Range[8,20,2]
```

```
Out[4]={8,10,12,14,16,18,20}
```

上面的参数变化都是只有一个，也可制成包括多个参数的表，下面生成一个多维表：

```
Ln[5]:=Table[2i+j,{i,1,3},{j,3,5]}
Out[5]={{5,6,7},{7,8,9},{9,10,11}}
```

使用函数TableForm可以以表格的方式输出

```
Ln[6]:=TableForm
Out[6]//TableForm=
5 6 7
7 8 9
9 10 11
```

2. 表的元素的操作

当t表示一个表时，t[[i]]表示t中的第i个子表。如果t={1,2,a,b}那么t[[3]]表示“a”。如：

```
ln[1]:=t=Table[I+2,j{I,1,3},{j,3,5]}
Out[1]={{7,9,11},{8,10,12},{9,11,13}}
ln[2]:=t[[2]]
Out[2]={8,10,12}
```

对于表的操作Mathematica提供了丰富的函数，详细的可以查阅后面的附录或者系统帮助。



3.1 多项式的表示形式

可认为多项式是表达式的一种特殊的形式，所以多项式的运算与表达式的运算基本一样，表达式中的各种输出形式也可用于多项式的输出。Mathematica 提供一组按不同形式表示代数式的函数。

Expand[ploy]	按幂次展开多项式 ploy
Expand[ploy]	全部展开多项式 ploy
ExpandAll[ploy]	全部展开多项式 ploy
Factor[ploy]	对多项式 poly 进行因式分解

<code>FactorTerms[poly, {x,y,...}]</code>	按变量 x,y,\dots 进行分解
<code>Simplify[poly]</code>	把多项式化为最简形式
<code>FullSimplify[poly]</code>	把多项式展开并化简
<code>Collect[poly,x]</code>	把多项式 poly 按 x 幂展开
<code>Collect[poly,{x,y,...}]</code>	把多项式 poly 按 x,y,\dots 的幂次展开

1.下面是一些例子

(1). 对 x^{8-1} 进行分解

```

In[1]:= Factor[x^8 - 1]

Out[1]= (-1 + x) (1 + x) (1 + x^2) (1 + x^4)

```

(2). 展开多项式 $(1+x)^5$

```

In[3]:= Expand[(1 + x)^5]

Out[3]= 1 + 5 x + 10 x^2 + 10 x^3 + 5 x^4 + x^5

```

(3). 展开多项式 $(1+x+3y)^4$

```

In[4]:= Expand[(1 + x + 3 y)^4]

Out[4]= 1 + 4 x + 6 x^2 + 4 x^3 + x^4 + 12 y + 36 x y + 36 x^2 y + 12 x^3 y +
        54 y^2 + 108 x y^2 + 54 x^2 y^2 + 108 y^3 + 108 x y^3 + 81 y^4

```

(4). 化简 $(2+x)^4(1+x)^4(3+x)^3$

```

In[7]:= Simplify[Expand[(2 + x)^4 (1 + x)^4 (3 + x)^3]]

Out[7]= (3 + x)^3 (2 + 3 x + x^2)^4

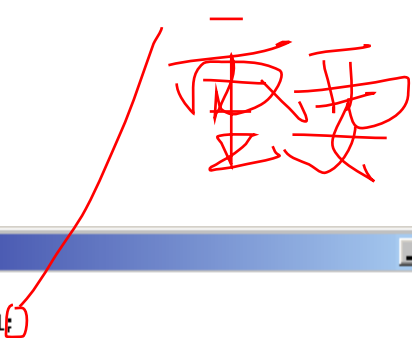
```

李用

2. 多项式的代数运算

多项式的运算有加、减、乘、除运算：+，-，*，/ 下面通过例子说明。

(1). 多项式的加运算 a^2+3a+2 与 $a+1$ 相加（后面例子中也使用这两个多项式运算



```
多项式运算.nb *
In[10]:=
  p1 = a^2 + 3 a + 2; p2 = a + 1;
  p1 + p2
Out[11]= 3 + 4 a + a^2
```

(2). 多项式相减

```
多项式运算.nb *
In[14]:=
  p2 - p1
Out[14]= -1 - 2 a - a^2
```

(3). 多项式相乘

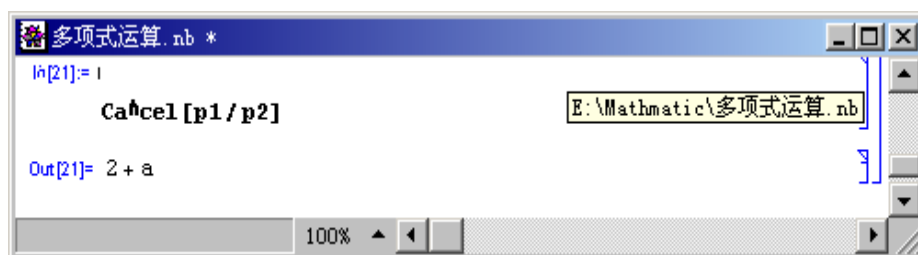
```
多项式运算.nb *
In[15]:=
  p1 * p2
Out[15]= (1 + a) (2 + 3 a + a^2)
```

(4). 多项式相除

```
多项式运算.nb *
  p1 / p2
Out[16]= 
$$\frac{2 + 3 a + a^2}{1 + a}$$

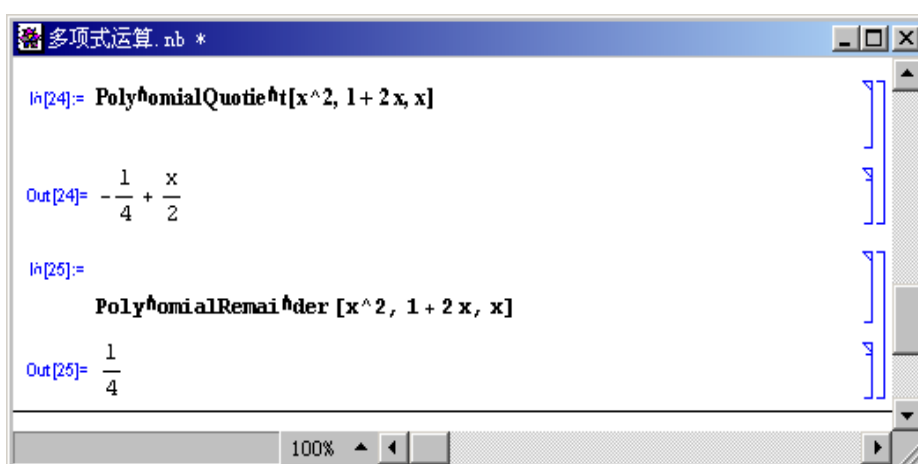
```

(5). 另外使用 Cancel 函数可以去公因式



两个多项式相除，总能写成一个多项式和一个有理式相加Mathematica中提供两个函数PolynomialQuotient和PolynomialRemainder分别返商式和余式。

例如：

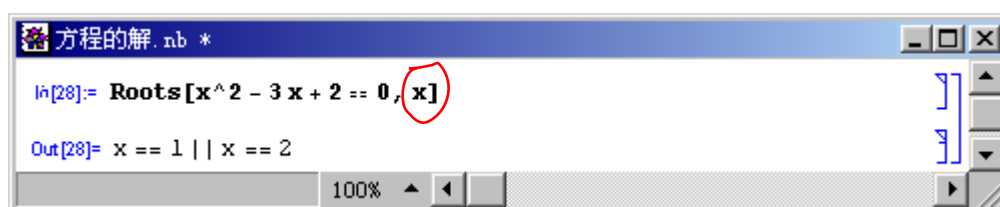


3. 2 方程及其根的表达



因为Mathematica把方程看作逻辑语句。在数学方程式表示为形如“ $x^2-2x+1=0$ ”的形式。在Mathematica中“=”用作赋值

语句，这样在Mathematica中用“==”表示逻辑等号，则方程应表示为“ $x^2-2x+1==0$ ”。方程的解同原方程一样被看作是逻辑语句。例如用Roots求方程 x^2-3x+2 的根显示为



这种表示形式说明x取1或2均可。而用Solve[]可得解集形式。

```

In[29]:=
Solve[x^2 - 3 x + 2 == 0, x]

Out[29]= {{x -> 1}, {x -> 2}}

```

1 求解一元代数方程

下面是常用的一些方程求解函数

Solve[lhs==rhs,vars]	给出方程的解集
NSolve[lhs==rhs,vars]	直接给出方程的数值解集
Roots[lhs==rhs,vars]	求表达式的根
FindRoot[lhs==rhs,{x,x0}]	求x=x0 时，方程的解值

先看Solve函数例子

```

In[32]:= Solve[x^2 + b*x + c == 0, x]

Out[32]= {{x -> 1/2 (-b - Sqrt[b^2 - 4 c])}, {x -> 1/2 (-b + Sqrt[b^2 - 4 c])}}

```

Solve函数可处理的主要方程是多项式方程。Mathematica总能对不高于四次的方程进行精确求解，对于三次或四次方程，解的形式可能很复杂。

例如求 $x^3+5x+3=0$

```

In[33]:= Solve[x^3 + 5 x + 3 == 0, x]

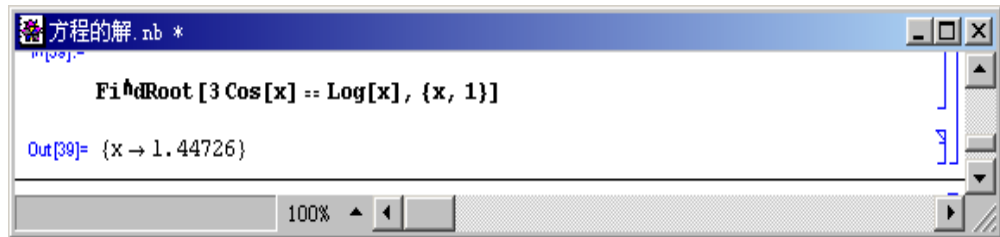
Out[33]= {{x -> -5 (2 / (3 (-27 + Sqrt[2229]))^(1/3) + (1/2 (-27 + Sqrt[2229]))^(1/3))},
  {x -> -((1 + I Sqrt[3]) (1/2 (-27 + Sqrt[2229]))^(1/3) + 5 (1 - I Sqrt[3]) / (2^(1/3) (3 (-27 + Sqrt[2229]))^(1/3))},
  {x -> -((1 - I Sqrt[3]) (1/2 (-27 + Sqrt[2229]))^(1/3) + 5 (1 + I Sqrt[3]) / (2^(1/3) (3 (-27 + Sqrt[2229]))^(1/3))}}

```

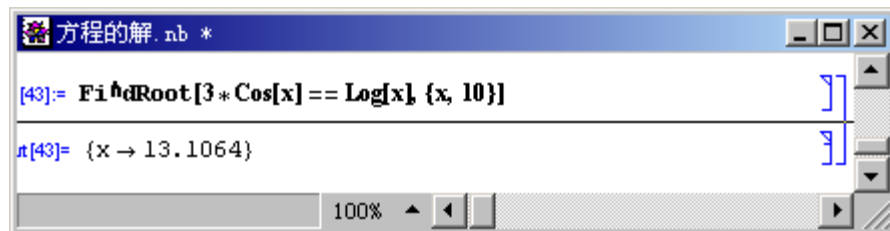
这时可用N函数近似数值解。

当方程中有一些复杂的函数时，Mathematica可能无法直接给出解来。在这种情况下我们可用FindRoot[]来求解，但要给出起始条件。

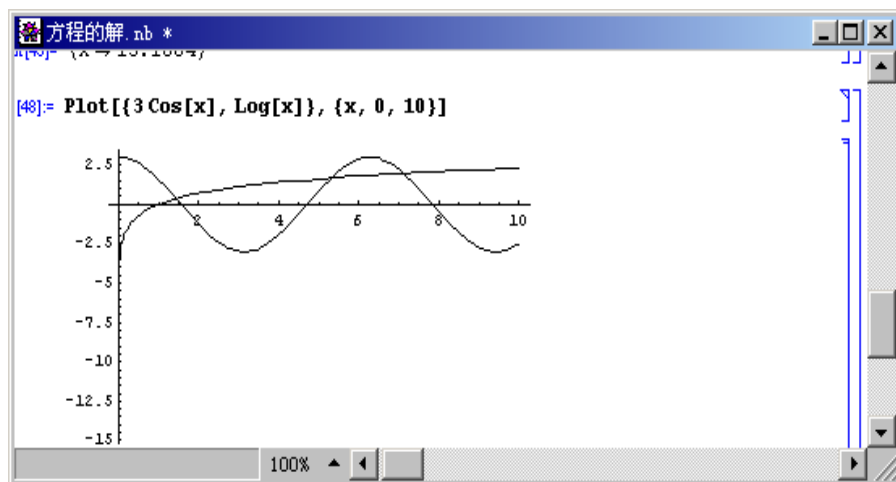
例如：求 $3\cos x = \log x$ 的解



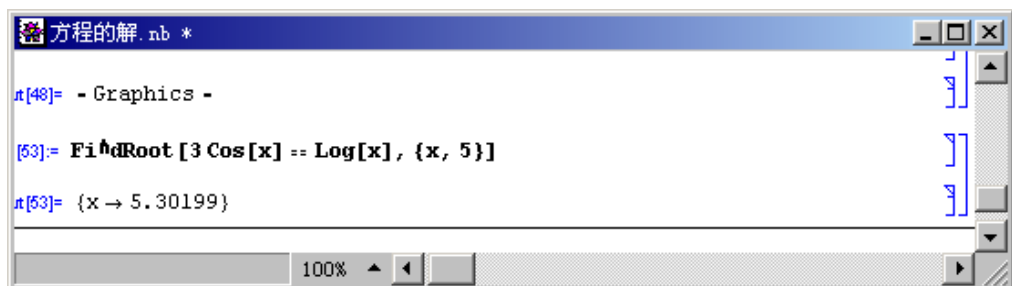
但只能求出 $x=1$ 附近的解，如果方程有几个不同的解，当给定不同的条件时，将给出不同的解。如上例若求 $x=10$ 附近的解命令为：



因此确定解的起始位置是比较关键，一种常用的方法是，先绘制图形观察后再解



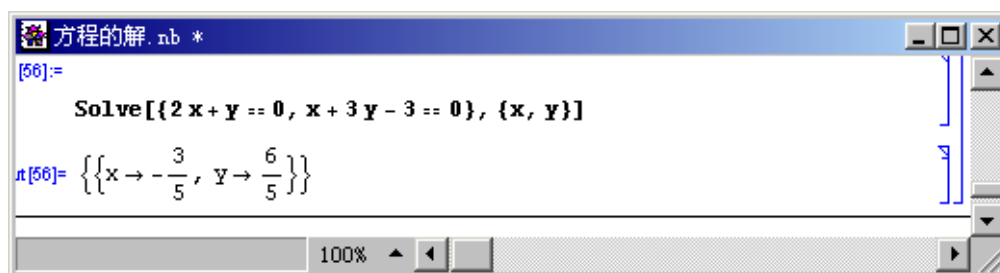
如上例通过图形可断定在 $x=5$ 附近有另一根



2.求方程组的根

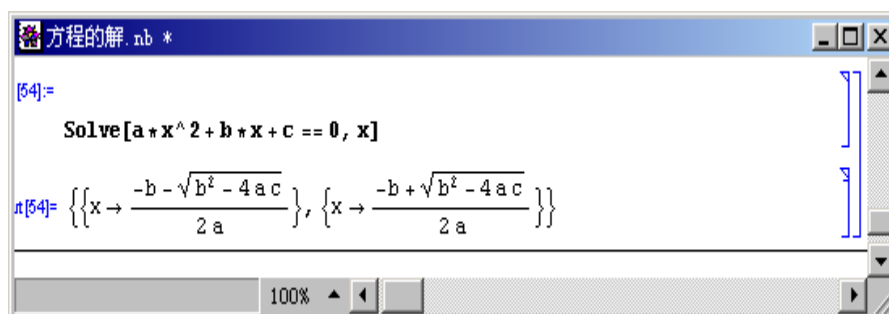
使用 Solve 和 NSolve, FindRoot 也可求方程组的解, 只是使用时格式略有不同下面给出一个 Solve 函数的例子:

求解:
$$\begin{cases} 2x + y = 0 \\ x + 3y - 3 = 0 \end{cases}$$

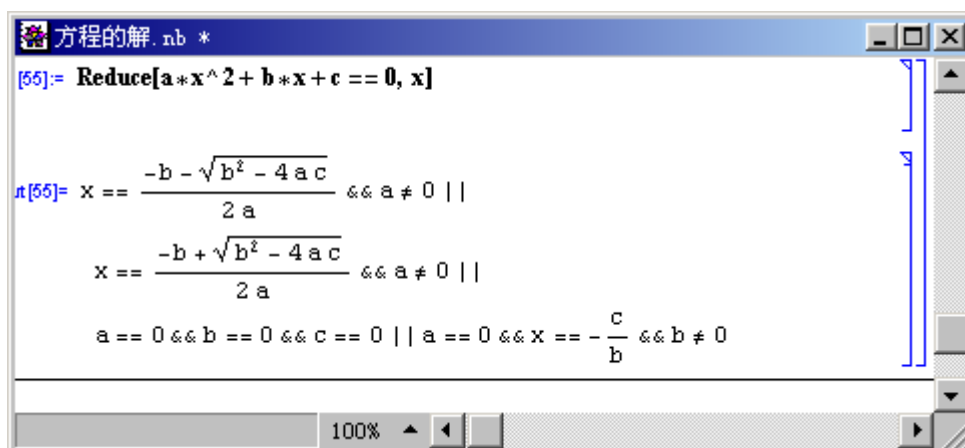


3 求方程的全解

如果我们求 $ax^2+bx+c=0$ 的根我们用 Solve 函数解的结果是:



这显然是不合理的, 因为对不同的 a, b, c 方程的解有不同的情况, 而上面只是给出部分解如果要解决这个问题可用 Reduce 命令, 它可根据 a, b, c 的取值给出全部值。



因此Solve,Roots只给出方程的一般解，而Reduce函数可以给出方程的全部可能解。

4.解条件方程

在作方程计算时，可以把一个方程看作你要处理的主要方程，而把其他方程作为必须满足的辅助条件。你将会发现这样处理很方便。譬如在求解像 $x^4 + bx^2 + c = 0$ 这样的方程

时，通常我们采用 $x^2 = y$ 的代换方法使求解方程

得到简化。在Mathematica中，我们通常是首先命名辅助条件组，然后用名字把辅助条件包含在你要用函数Solve[]求解的方程组中。

用Sc定义方程： $\sin^2 x + \cos^2 x = 1$ ，在这种条件下，求解方程。

```
Untitled-1 *
In[6]:= Sc = Sin[x]^2 + Cos[x]^2 == 1
Out[6]= Cos[x]^2 + Sin[x]^2 == 1
In[5]:= Solve[{Cos[x] + 2 Sin[x] == 1, Sc}, {Sin[x], Cos[x]}]
Out[5]= {{Sin[x] -> 0, Cos[x] -> 1}, {Sin[x] -> 4/5, Cos[x] -> -3/5}}
```

3.3 求和与求积



在 Mathematica 中，数学上的各式符号 \sum 用 Sum 表示，连乘 \prod 用 Product 表示。下面列出求

各与求积函数的形式和意义：

Sum[f,{i,imin,imax}] 求和 $\sum_{i=imin}^{imax} f$

f 表达式

Sum[f,{i,imin,imax,di}] 以步长 di 增加 i 求和

Sum[f,{i,imin,imax},{j,jmin,jmax}] 嵌套求和 $\sum_{i=imin}^{imax} \sum_{j=jmin}^{jmax} f$

Product[f,{i,imin,imax}] 求积 $\prod_{i=\text{imin}}^{\text{imax}} f$

Product[f,{i,imin,imax,di}] 以步长 di 增加 i 求和

Product[f,{i,imin,imax},{j,jmin,jmax}] 嵌套求积 $\prod_{i=\text{imin}}^{\text{imax}} \prod_{j=\text{jmin}}^{\text{jmax}} f$

Nsum[f,{i,imin,Infinity}] 求 $\sum_{i=\text{imin}}^{\infty} f$ 近似值

NProduct[f,{i,imin,Infinity}] 求 $\prod_{i=\text{imin}}^{\infty} f$ 近似值

一些例子

```

(*求1到9的奇数之和 *)
Sum[2 i - 1, {i, 1, 9}]

81

如果下限等于1 则可以忽略

Sum[2 i - 1, {i, 9}]

81

下式构造一个多项式

Sum[i * x^i, {i, 1, 9, 2}]

x + 3 x^3 + 5 x^5 + 7 x^7 + 9 x^9

Mathematic 可以给出和的精确结果

Sum[1/h!, {h, 1, 11}]

8573539
4989600

N[%]

1.71828
  
```

4.1 基本的二维图形



Mathematica 在直角坐标系中作一元函数图形用下列基本命令。

Plot[f, {x, xmin, xmax}, option->value] 在指定区间上按选项定义值画出函数在直角坐标系中的图形。

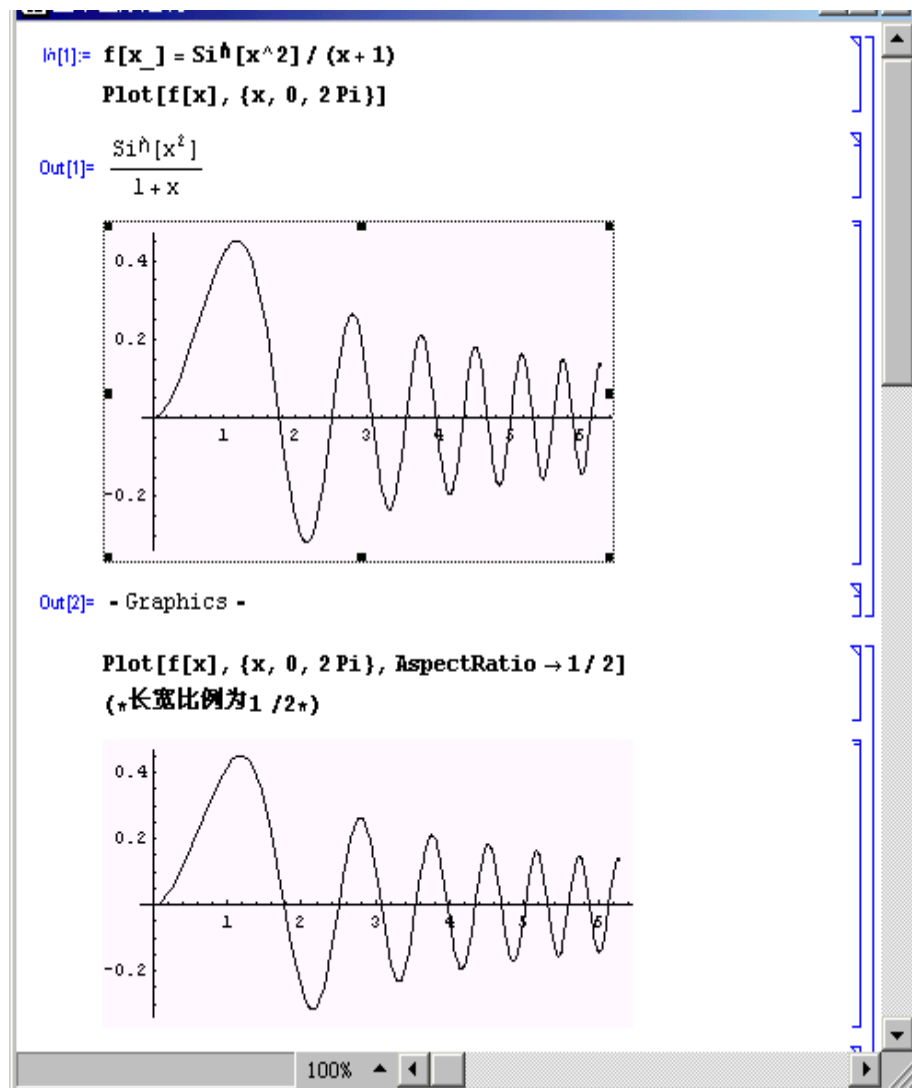
Plot[{f1,f2,f3,...},{x,xmin,xmax},option->>value] 在指定区间上按选项定义值同时画出多个函数在直角坐标系中的图形

Mathematica 绘图时允许用户设置选项值对绘制图形的细节提出各种要求。例如：要设置图形的高宽比，给图形加标题等。每个选项都有一个确定的名字，以“选项名->选项值”的形式放在 Plot 中的最右边位置，一次可设置多个选项，选项依次排列，用逗号隔开，也可以不设置选项，采用系统的默认值。

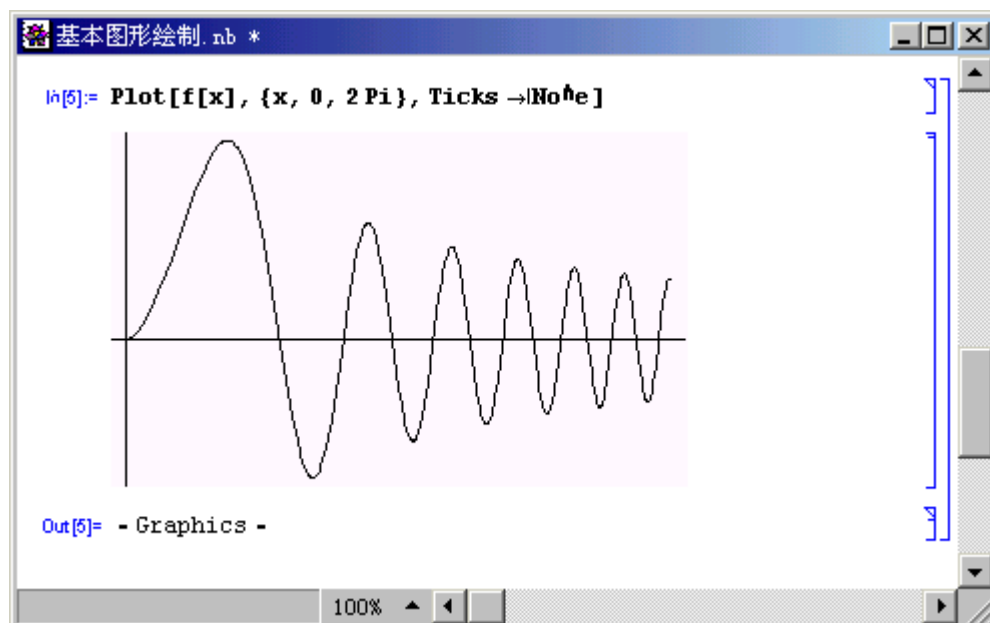
选项	说明	默认值
AspectRatio	图形的高、宽比	1/0.618
AxesLabel	给坐标轴加上名字	不加
PlotLabel	给图形加上标题	不加
PlotRange	指定函数因变量的区间	计算的结果
PlotStyle	用什么样方式作图（颜色，粗细等）	值是一个表
PlotPoint	画图时计算的点数	25

1. 举例

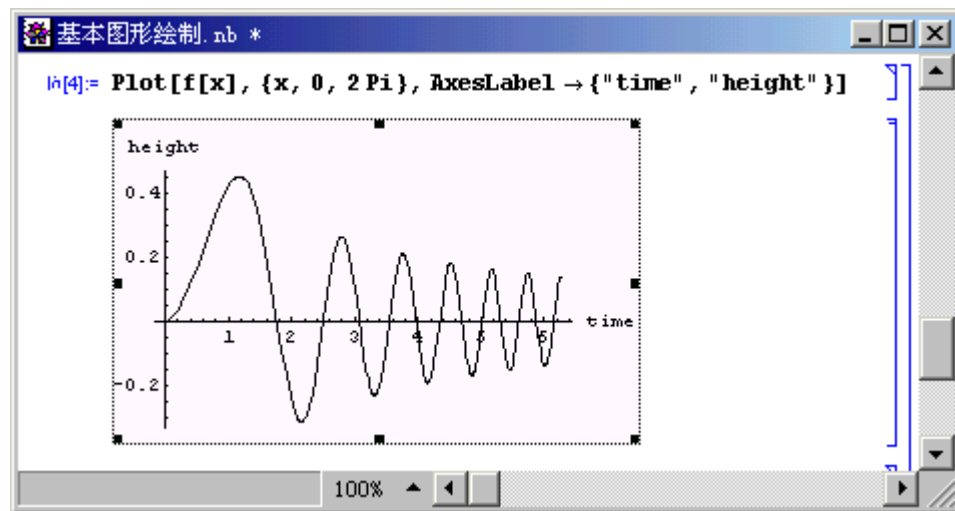
(1). 例如绘制 $f(x) = \frac{\sin x^2}{x+1}$ 的图形。



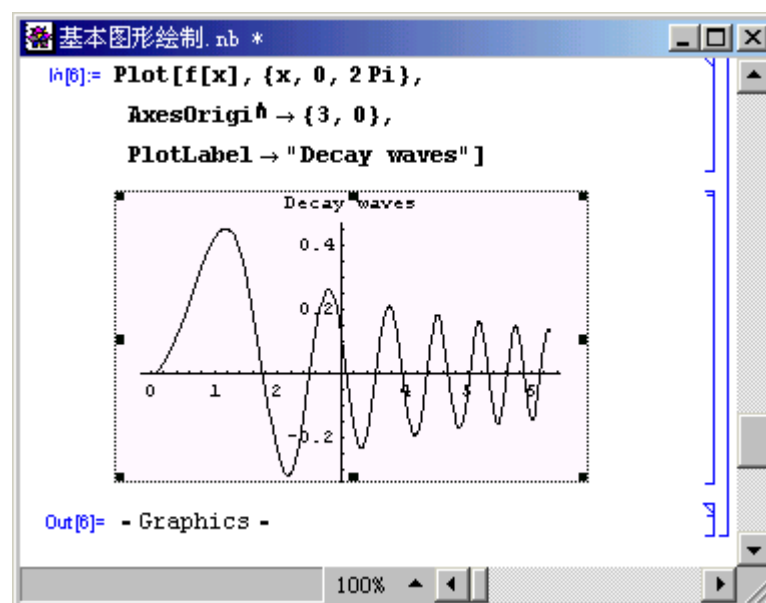
(2). 如果要取消刻度可以使用 Ticks 选项



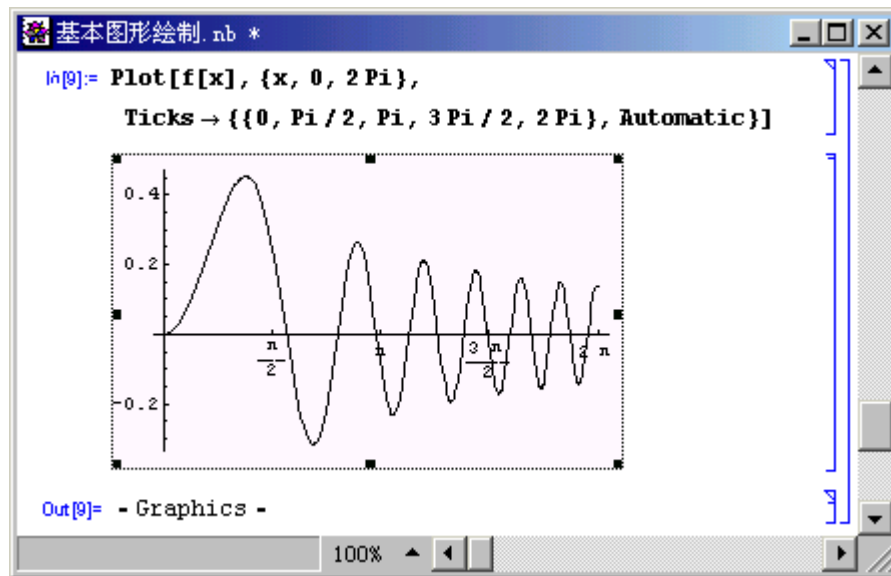
(3). 如果要标注坐标名称 x 轴为“Time”,y 轴为“Height”



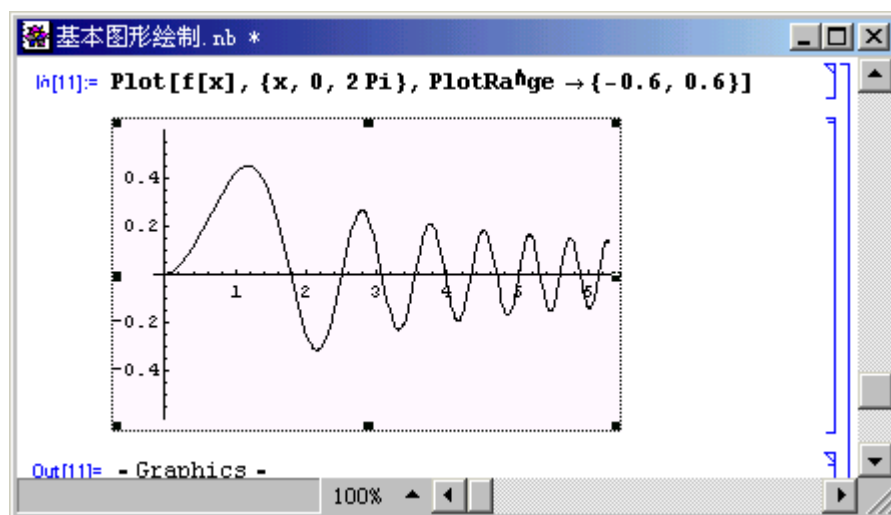
(4). 将坐标原点 (3, 0), 并标注图形名称。



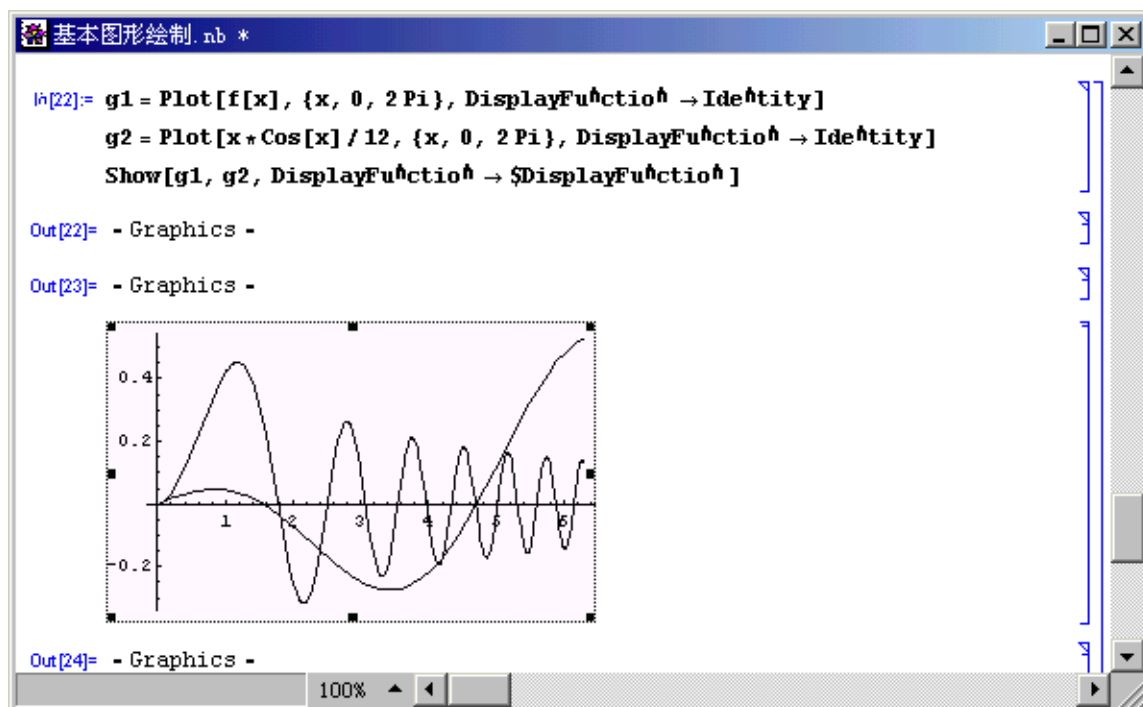
(5). 修改 x 方向的刻度, y 轴方向的刻度则用默认值。



(6). 定义 y 轴的绘图范围



(7). 另外我们也可以将图形结果定义给变量，但不显示图形，后用 Show 命令显示。

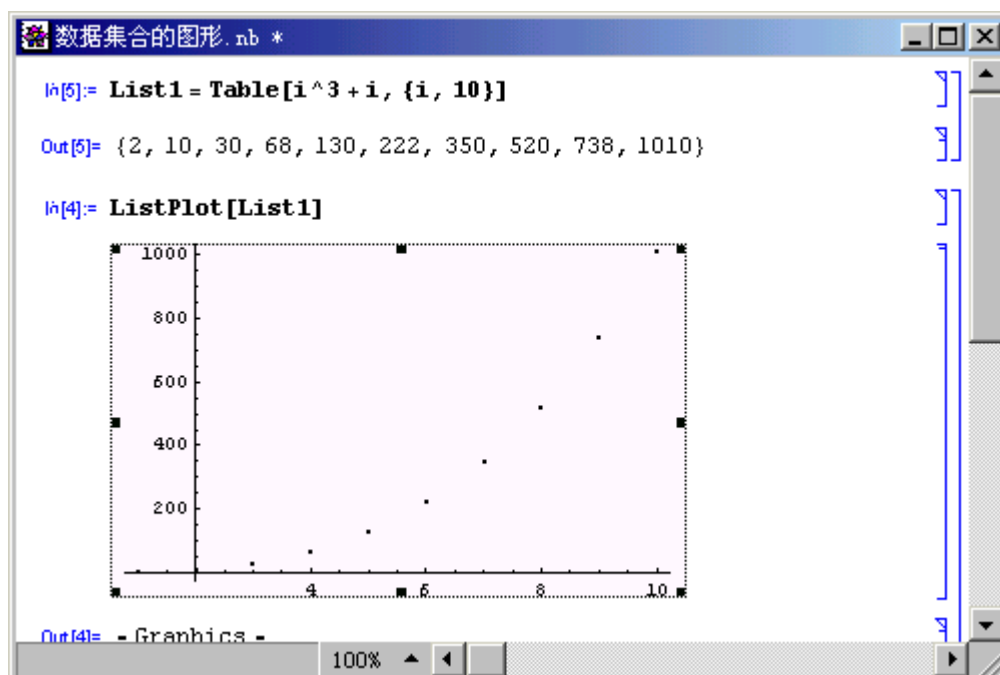


2.数据集合的图形

Mathematica 用于绘数字集合的图形的命令与前而介绍的绘函数图形的命令是相似的。如下:

ListPlot[{y1,y2,...}]	绘出在 x 的值为 1, 2...时 y1,y2,...的图形
ListPlot[{{x1,y1},{x2,y2},...}]	绘出离散点 (xi,yi)
ListPlot[List,PlotJoined->True]	把离散点连成曲线

(1). 下面举例说明下面是一个离散数据的集合的图形

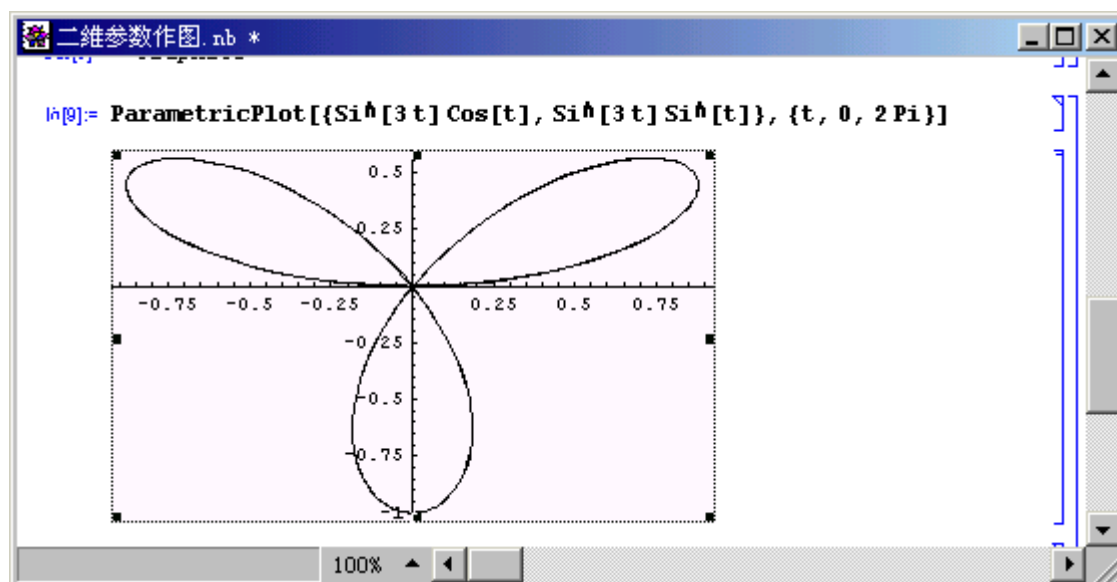


3.二维参数作图

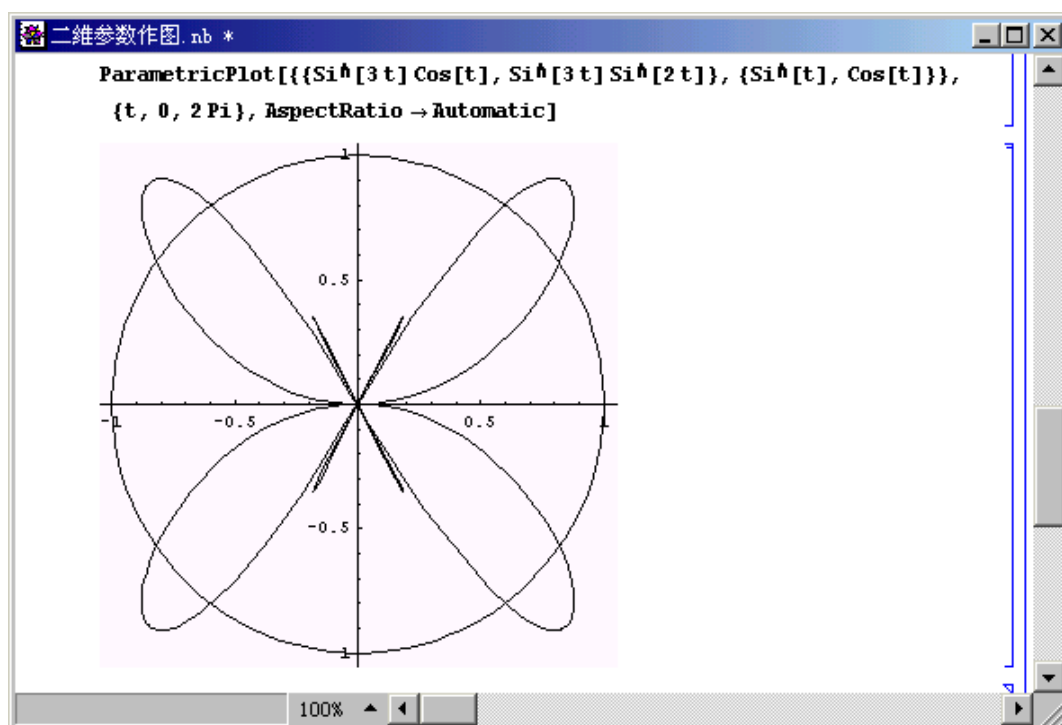
前面我们使用 Plot 命令可以绘出直角坐标系下的函数图形，使用 ParametricPlot 可以绘制参数曲线下面给出 ParametricPlot 的常用形式

<code>ParametricPlot[{fx,fy},{t,tmin,tmax}]</code>	绘出参数图
<code>ParametricPlot[{fx,fy},{gx,gy},...{t,tmin,tmax}]</code>	绘出一组参数图
<code>ParametricPlot[{fx,fy},{t,tmin,tmax},AspectRatio->Automatic]</code>	设法保持曲线的形

(1).绘制参数方程
$$\begin{cases} x = \sin 3t \cos t \\ y = \sin 3t \sin t \end{cases}$$
 的图形



(2). 下面将一个圆与上面参数绘在同一个坐标下，并保证图形的形状正确。



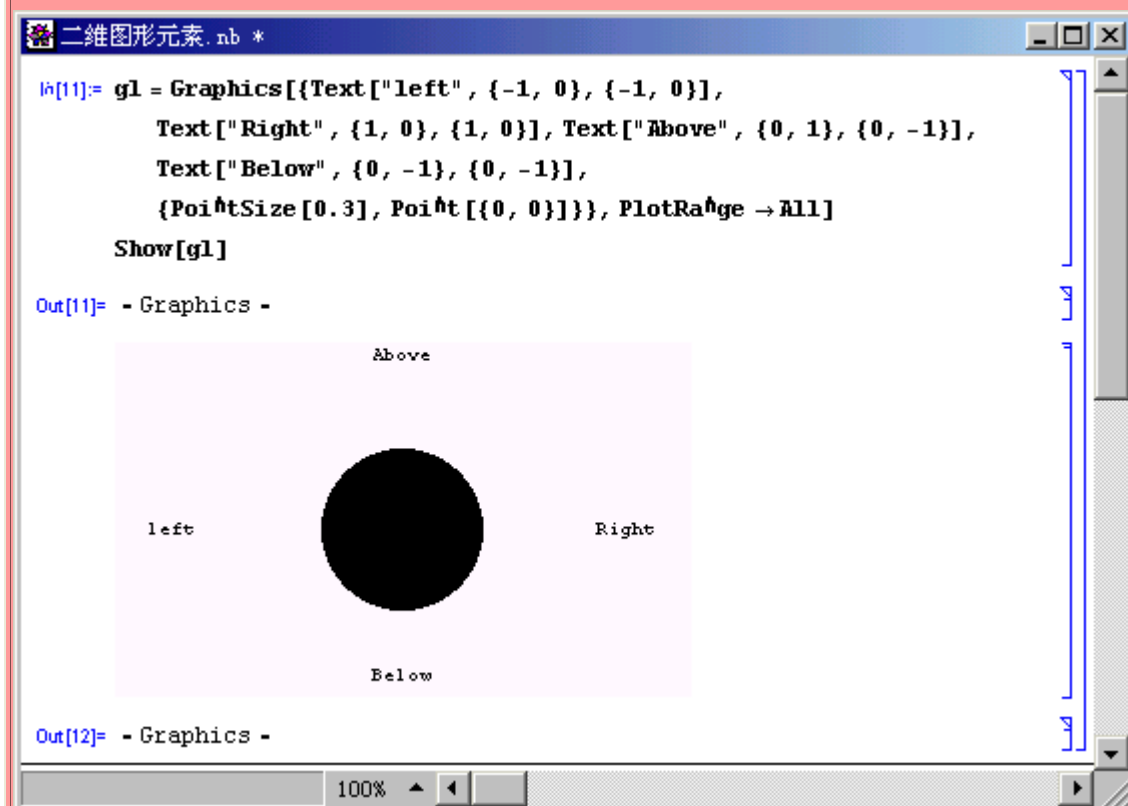
4. 1 二维图形元素

用图形元素绘图适合于绘制结构复杂的图形。Mathematica 中还提供了各种如绘制点、线段、圆弧等函数。同样我们可先用 Graphics 作出平面图形的表达式，再用 Show 显示生成的图形。

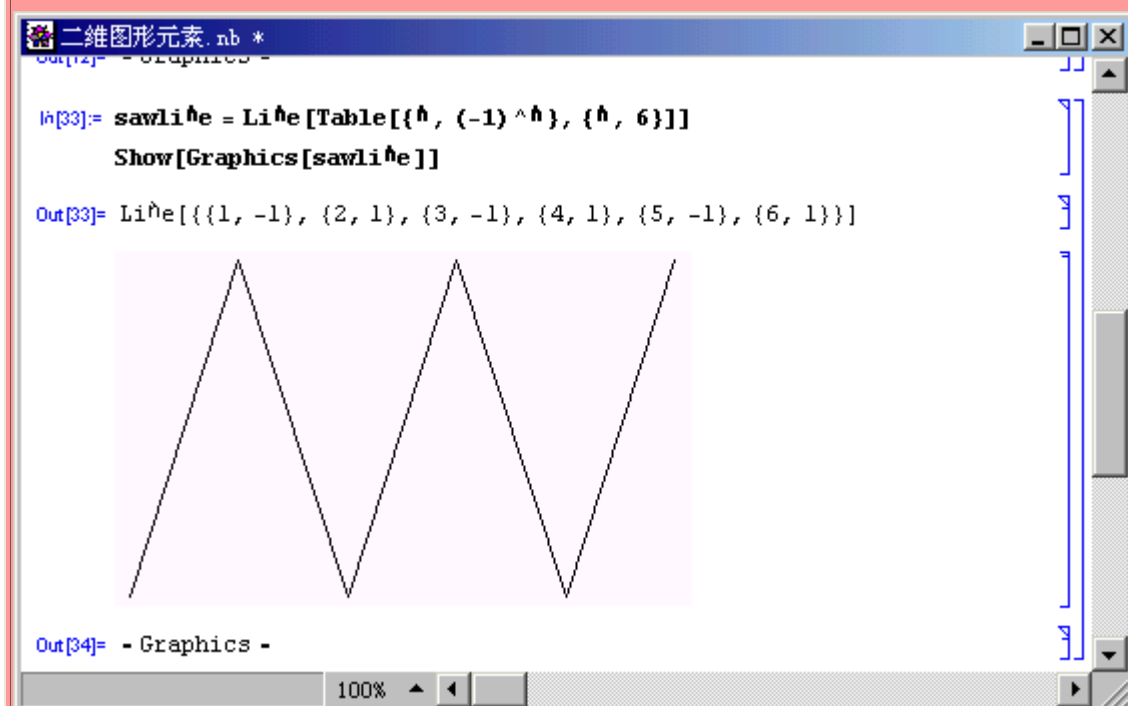
下面给出在 Mathematica 中常用的二维图形元素。

Point[{x,y}]	点
Line[{x1,y1},{x2,y2},...]	线段
Rectangle[{xmin,ymin},{xmax,ymax}]	填充矩形
Polygon[{x1,y1},{x2,y2},.....]	填充多边形
Circle[{x,y},r]	圆
Circle[{x,y},{rx,ry}]	半轴分别为 rx,ry 的椭圆
Circle[{x,y},r,{theta1,theta2}]	圆弧
Circle[{x,y},{rx,ry},{theta1,theta2}]	椭圆弧
Disk[{x,y},r]	填充圆
Raster[{{a11,a12,.....},{a21,.....},.....}]	灰度在 0 到 1 之间的灰层组
Text[Expr,{x,y}]	文本大小

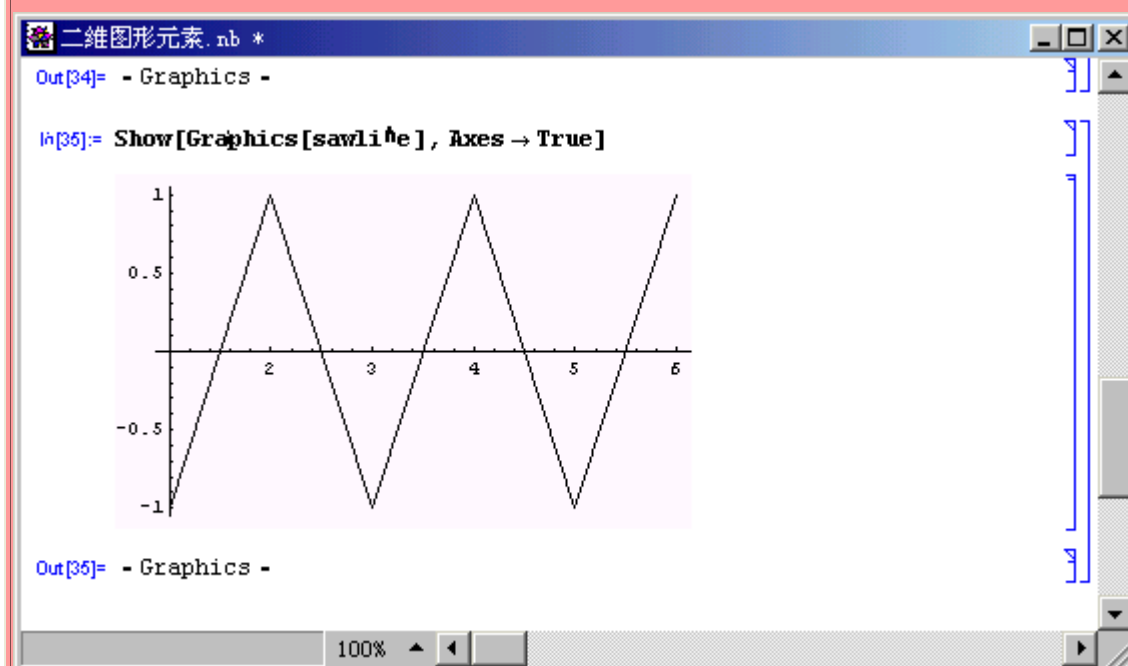
下图绘出一个有颜色和大小的点，且在图形四周插入文本



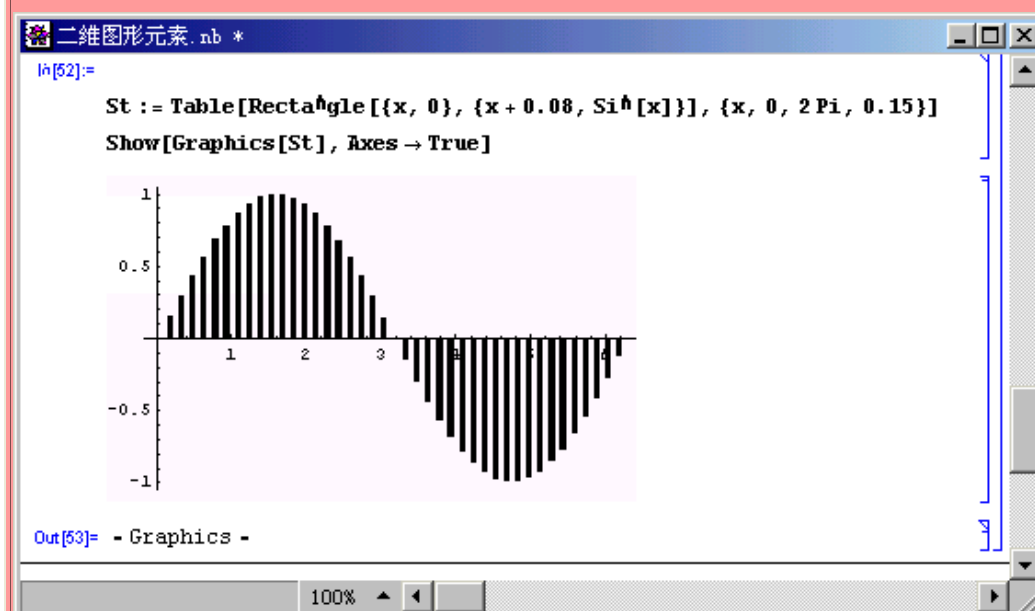
下面绘制一些有线条组成的图形



当然也可以添加坐标轴下面的例子，说明了这一点。



下面的例子，是说明了 Retangle 的图形绘制，例子中用一些小矩形逼近正弦曲线与 x 轴所成面积。程序中生成一个图形集合并显示出来。



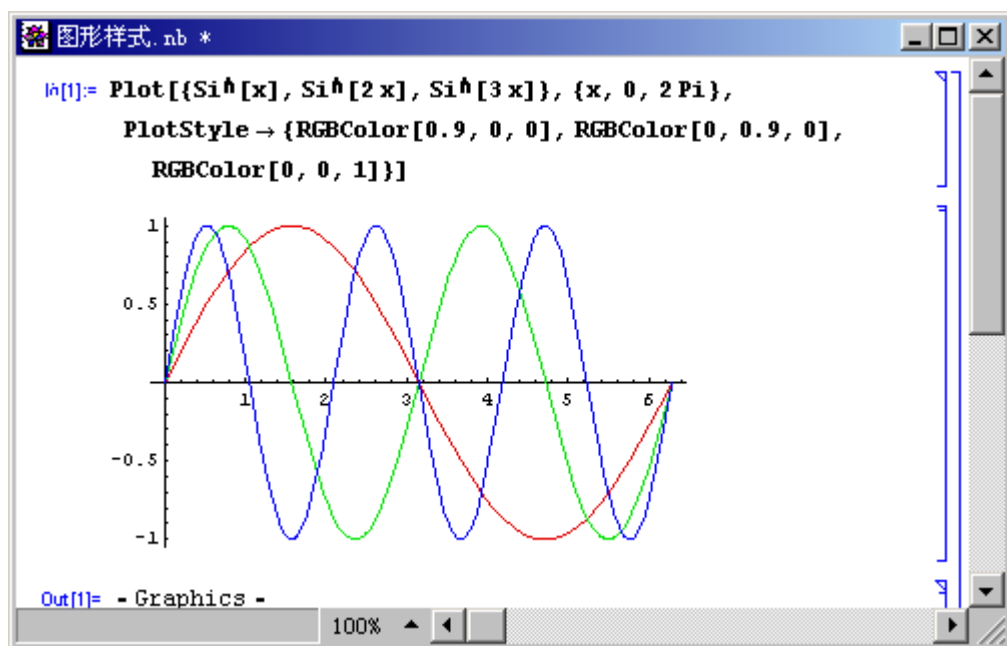
我们称图形的颜色、曲线的形状和宽度等特性为图形样式。在本节中，我们就图形的各种样式，尤其是曲线的样式进行学习。

下面给出选项用于设置图形样式。

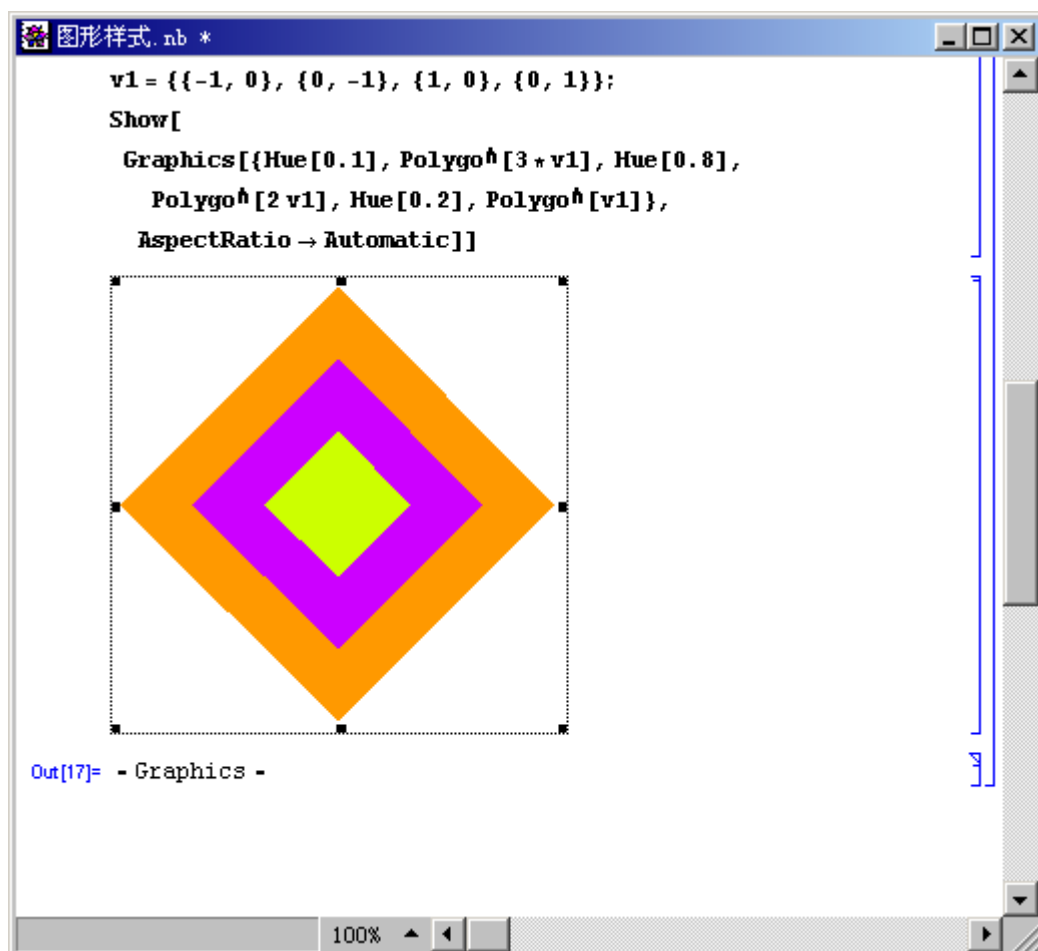
GrayLevel[]	灰度介于 0(黑)到 1(白)之间
RGBColor[r,g,b]	由红、绿、蓝组成的颜色，每种色彩取 0 到 1 之间的数
Hue[A]	取 0 到 1 之间的色彩
Hue[h,s,b]	指定色调，位置和亮度的颜色，每项介于 0 到 1 之间
PointSize[d]	给出半径为 d 的点，单位是 Plot 的一个分数
AbsolutePointSize[d]	给出半径为 d 的点(以绝对单位量取)
Thickness[w]	给所有线的宽度 w，单位是 Plot 的分数
AbsoluteThickness[w]	给所有线的宽度 w，(以绝对单位量取)
Dashing[w1,w2,...]	给所有线为一系列虚线，虚线段的长度为 w1, w2,...
Absolutedashing[{w1,w2,...}]	以绝对单位给出虚线长度
PlotStyle->style	设立 Plot 中所有曲线的风格
PlotStyle->{{Style1},{Style2}.....}	设立 Plot 中一些列曲线的风格
MeshStyle->Style	设立宽度和表面网格的风格

1.图形颜色的设置

在 Mathematica 提供各种图形指令中，对图形元素颜色的设置是一个很重要的设置。下面给出三条不同颜色的正弦曲线，此处以灰度表示，即颜色深浅不同。

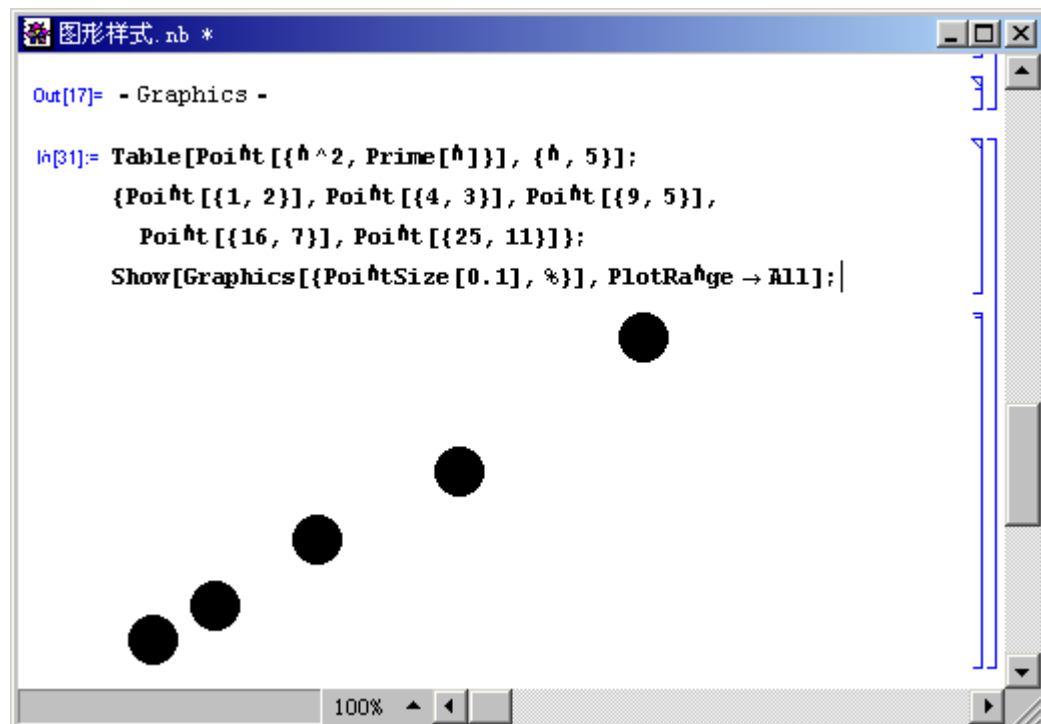


下面用不同的色调对三个菱形进行着色。

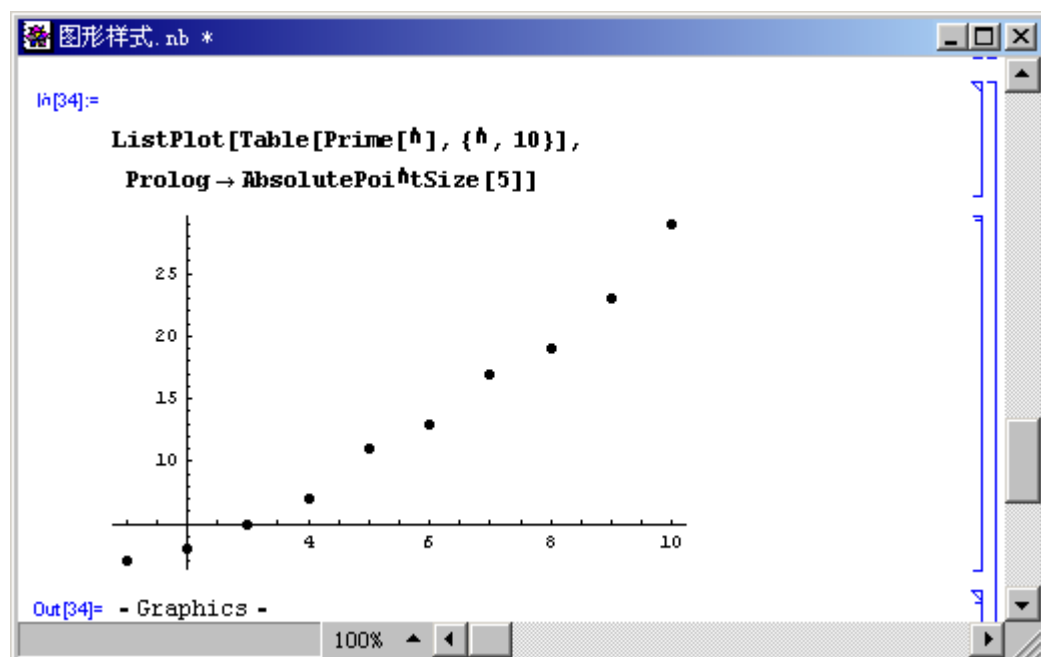


2.图形大小

下面是一些点，注意点大小的控制。

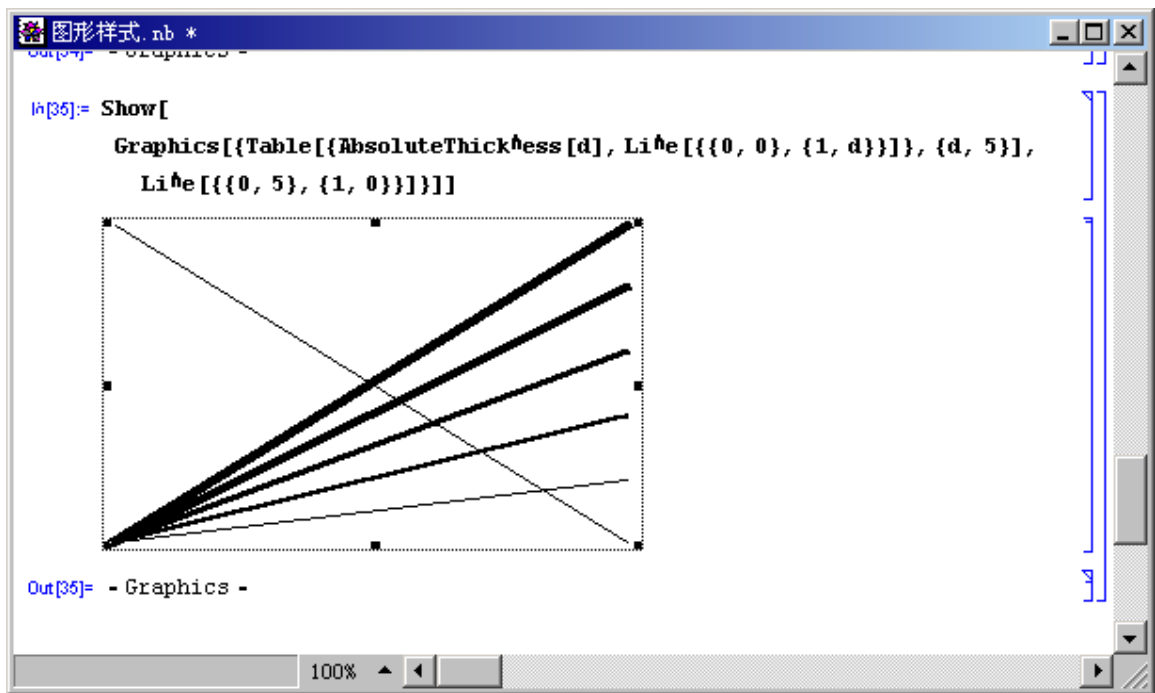


下面的点的控制是用绝对单位



3. 线段的控制

下面的例子是控制线段的宽度，使用的是绝对控制。



Mathematica 提供的虚线指令可生成多种不同的复杂虚线。

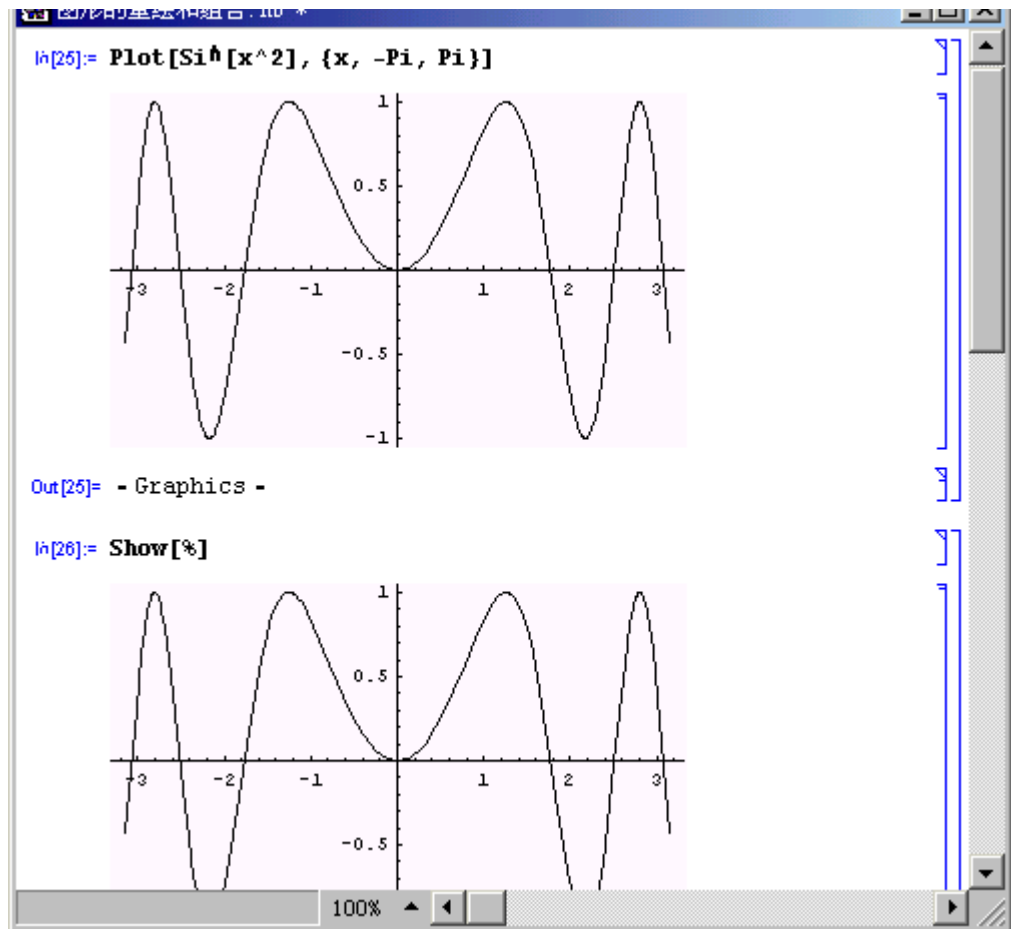
4.4 图形的重绘和组合

每次绘制图形后，Mathematica 保存了图形的所有信息，所以用户可以重绘 这些图形。我们在重绘图形的时候，还可以改变一些使用。下面是常用重绘图形的函数。

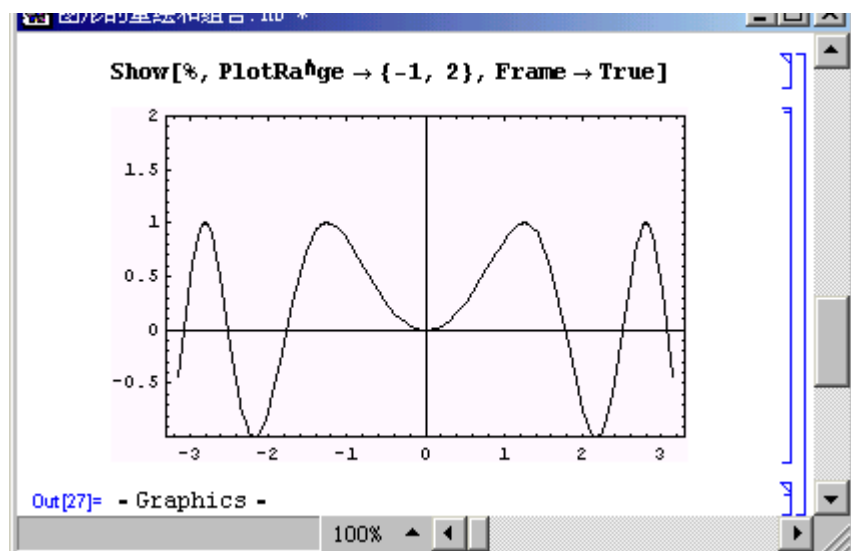
Show[plot]	重绘图形
Show[plot,option->value]	改变方案重绘图形
Show[plot1,plot2,plot3...]	多个图形的绘制
Show[GraphicsArray[{{plot1,plot2,...}...}]]	绘制图形矩阵
InputForm[plot]	给出所有的图形信息

1.使用 Show 显示图形

下面绘制函数 $\sin[x^2]$ 的图形。

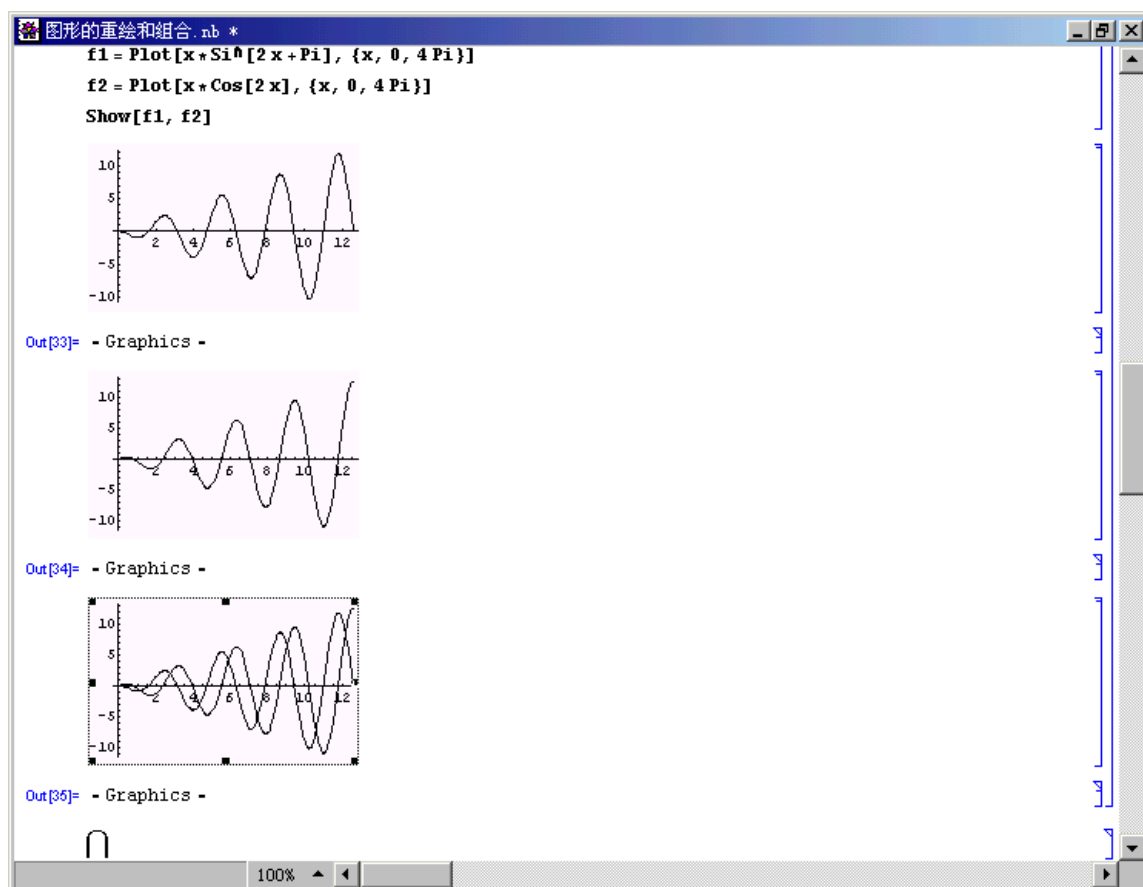


重绘图形时，可以改变命令的设置，下面改变 y 的比例同时给图边框



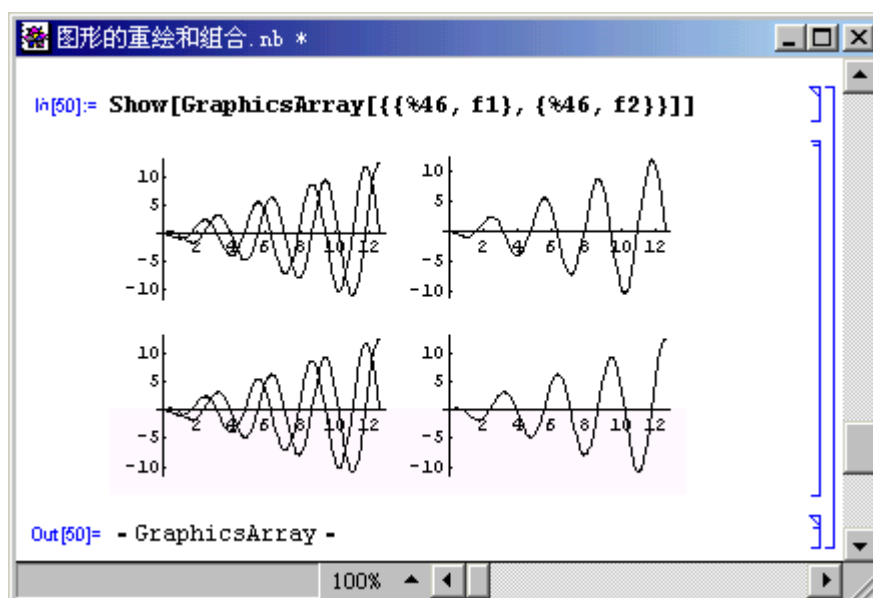
2.使用 Show 命令进行组合

也可使用 Show 进行图形组合。图形组合与图形是否有相同的比例无关，这是 Mathematica 会自动选择新的比例来绘制图形。下面绘制函数 $-\text{xsin}(2\text{x}+\text{Pi})$ 的图形和 $\text{xcos}(2\text{x})$ 然后绘制在一张图时。



3. 将多个图形组合为一个图形

我们也可把图形组合为一个图形，我们还可以用 `GraphicsArray` 把多个图形绘制在一个图形矩阵中如下图。



4. 5 基本三维图形



绘制函数 $f(x, y)$ 在平面区域上的三维立体图形的基本命令是 `Plot3D`，`Plot3D` 和 `Plot` 的工作方式和选项基本相同。`ListPlot3D` 可以用来绘制三维数字集合的三维图形，其用法也类似于 `listPlot`，下面给出这两个函数的常用形式。

`Plot3D[f(x,xmin,xmax), (y,ymin,ymax)]` 绘制以 x 和 y 为变量的三维函数 f 的图形

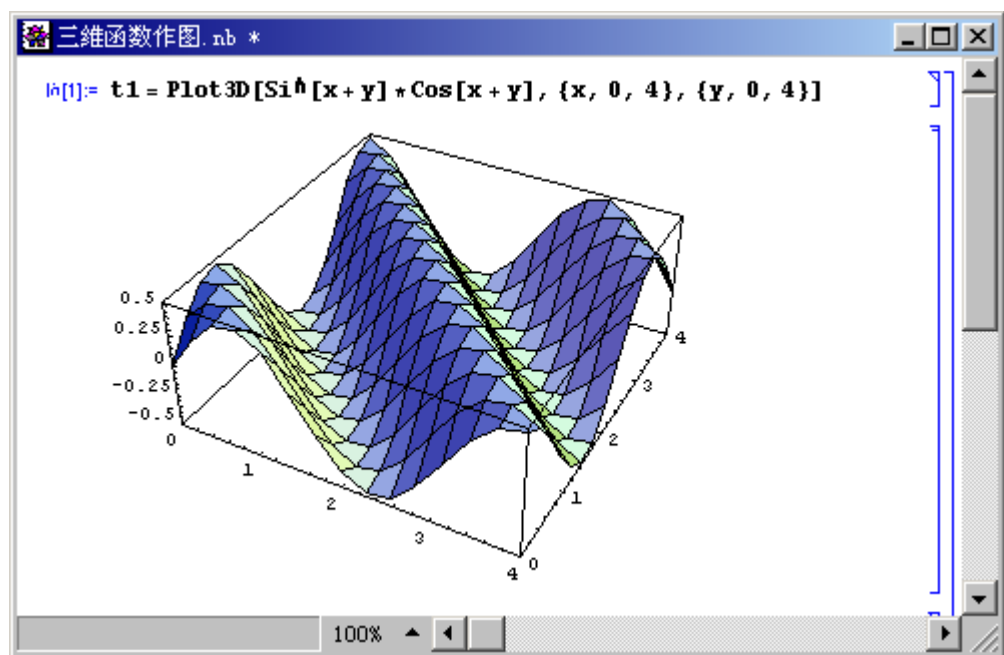
`ListPlot3D[{Z11,Z12,...}, {Z21,Z22,...},.....]` 绘出高度为 Z_{vx} 数组的三维图形

`Plot3D` 同平面图形一样，也有许多输出选项，你可通过多次试验找出你所需的最佳图形样式。

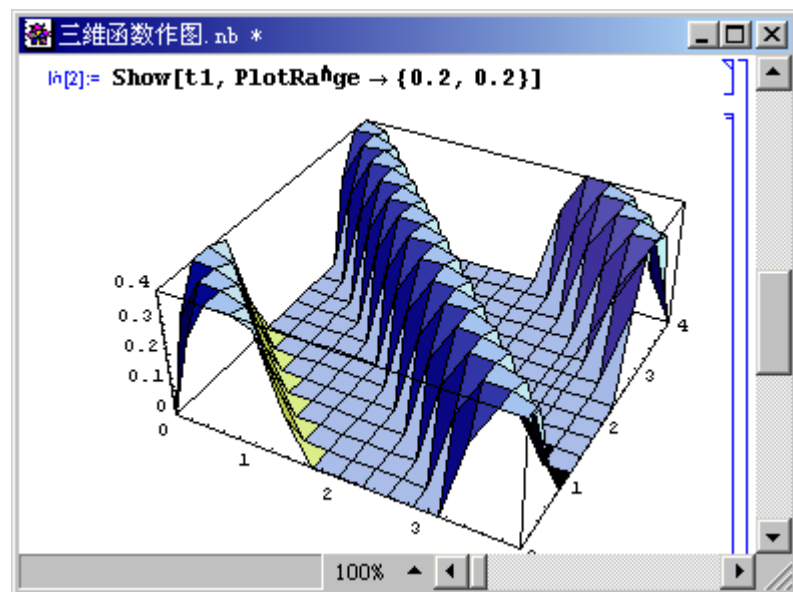
选项	取值	意义
Axes	True	是否包括坐标轴
AxesLabel	None	在轴上加上标志：zlabel 规定 z 轴的标志，{xlabel,ylabel,zlabel} 规定所有轴的标志
Boxed	True	是否在曲面周围加上立方体
ColorFunction	Automatic	使用什么颜色的明暗度；Hue 表示使用一系列颜色
TextStyle	STextStyle	用于图形文本的缺省类型
ormatType	StandardForm	用于图形文本的缺省格式类型
DisplayFunction	SdisplayFunction	如何绘制图形，Identity 表示不显示
FaceGrids	None	如何在立体界面上绘上网格；All 表示在每个界面上绘上网格
HiddenSurface	True	是否以立体的形式绘出曲面
Lighdng	True	是否用明暗分布米给表面加色
Mesh	True	是否在表面上绘出 xy 网格
PlotRange	Automatic	图中坐标的范围；可以规定为 All, {zmin,zmax} 或 {xminn,xmax}, {ymin,ymax},{zmin,zmax}
Shading	True	表面是用阴影还是留空白
ViewPoint	{1. 3, -2. 4,2}	表面的空间观察点

1. 三维绘图举例

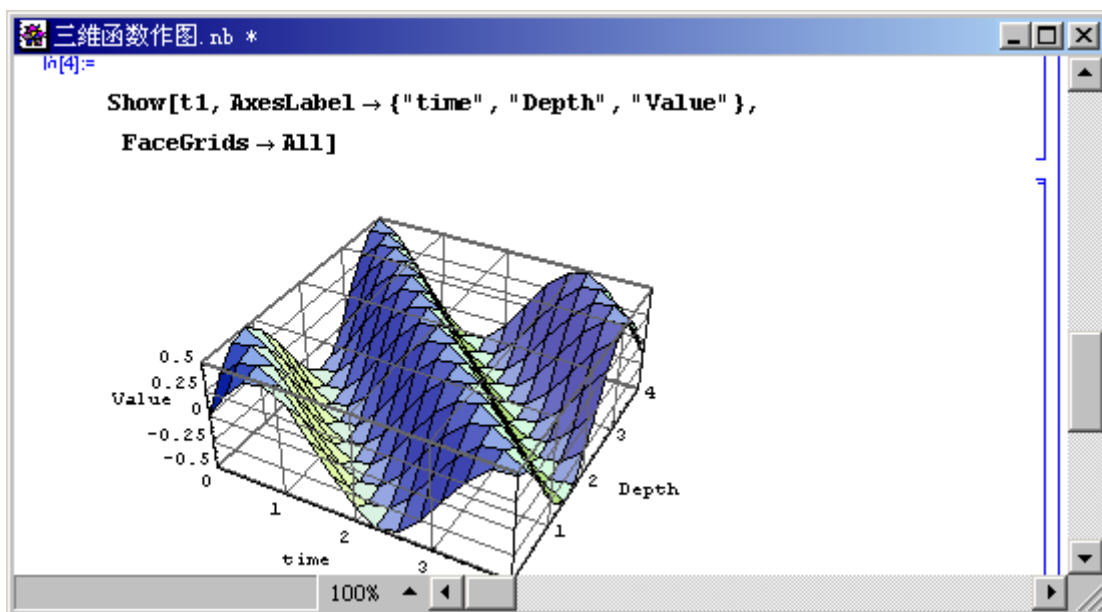
(1). 函数 $\sin(x+y)\cos(x+y)$ 的立体图



(2). 对于三维图形中 Axes、Axeslabel、Boxed 等操作同二维图形的一些操作很相似。用 PlotRange 设定曲线的表面的变化范围。

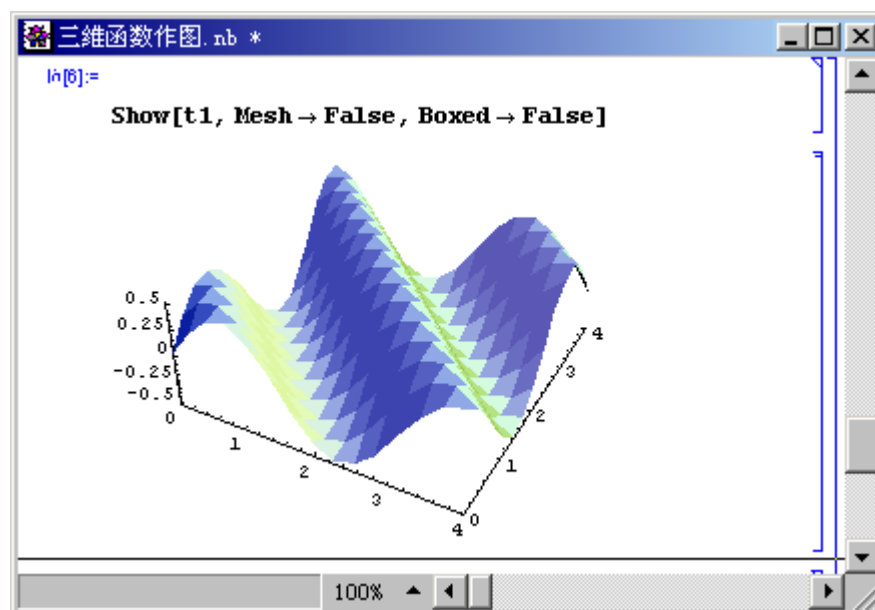


(3). 图形轴上加上标记，且在每个平面上画上网格。

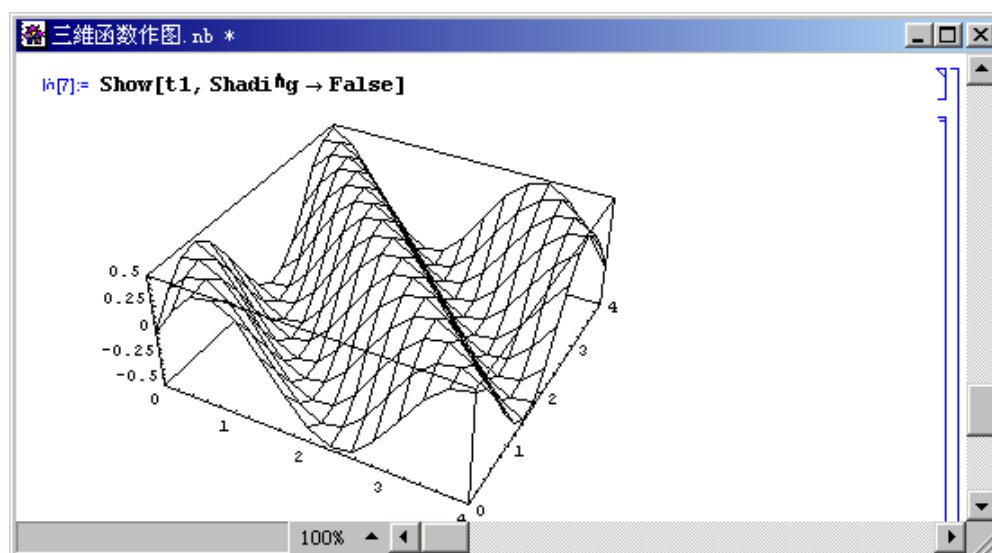


从上面我们可以看出，观察点位于曲面的上方有利于看清对于图形全貌。对于较复杂的图形，我们在所绘的图形上包括尽可能多的曲线对于我们观察很有帮助。同时，在曲面的周围直接绘出立方体盒子也有利于我们认清曲面的方位。

(6). 下面是没有网格和立体盒子的曲面图，它看起来就不如前面的图形清晰明了。



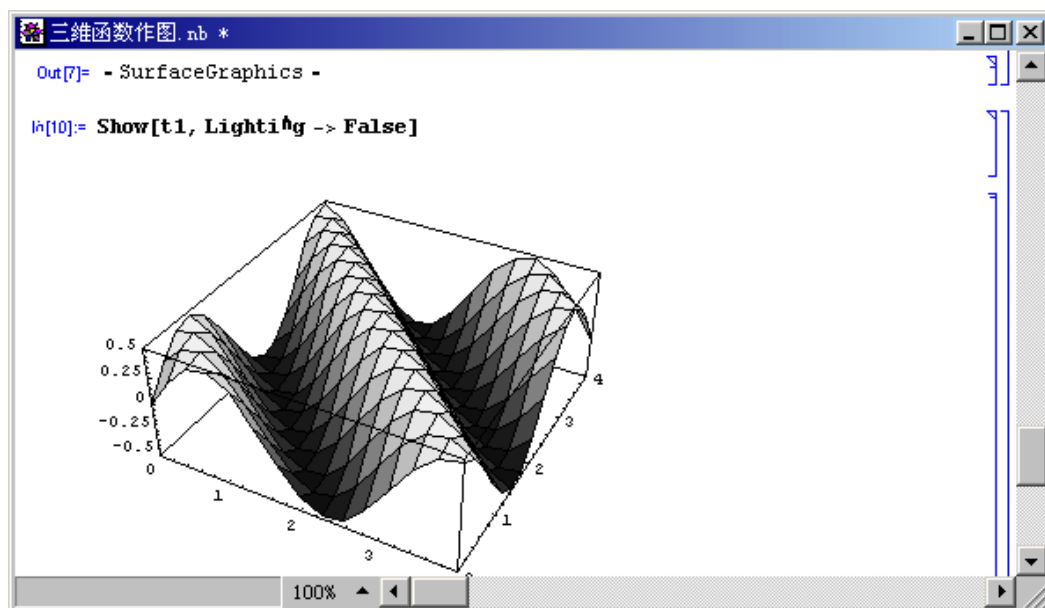
(7). 下图给出没有阴影的曲面



带有阴影和网格的图形对于理解曲面的形状是很有好处的。在有些矢量图形的输出装置中，你可能得不到阴影，但是当有阴影时，输出装置可能要花很长时间来输出它。

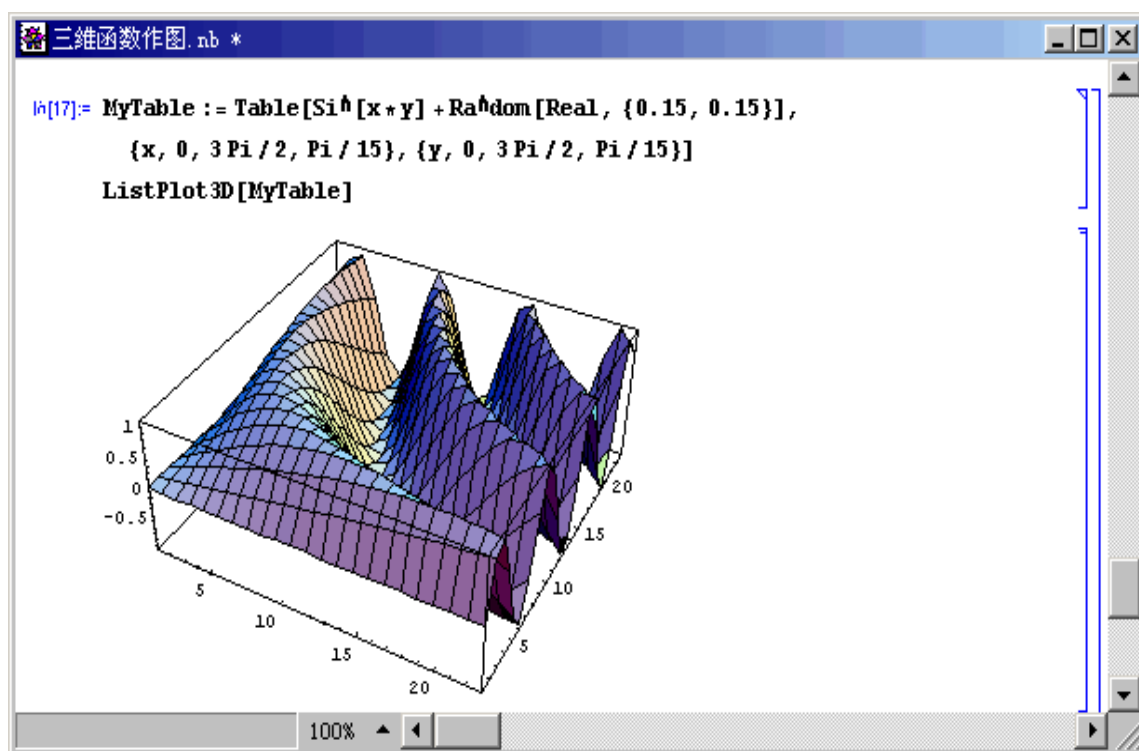
(8). 给空间立体曲面着色

通常情况下，Mathematica 为了使图形更加逼真而用明暗分布的形式给空间立体曲面着色。在这种情况下，Mathematica 假定在图形的右上方有三种光源照在物体上。但有时这种方法会造成混乱，此时你可用 `Lighting->False` 来采取根据高度在表面上涂以不同灰度的阴影的方法。



2 用数据来进行绘图

同二维绘图一样，三维图形也可用数据来进行绘图。下面给出数据矩阵，因其较大未表示其结果。



3. 三维空间的参数方程绘图

三维空间中的参数绘图函数 `ParametricPlot3D[{fx,fv,fz},{t,tmin,tmax}]` 和二维空间中的 `ParametricPlot`

很相仿。在这种情况下，Mathematica 实际上都是根据参数 t 来产生系列点，然后再连接起来。

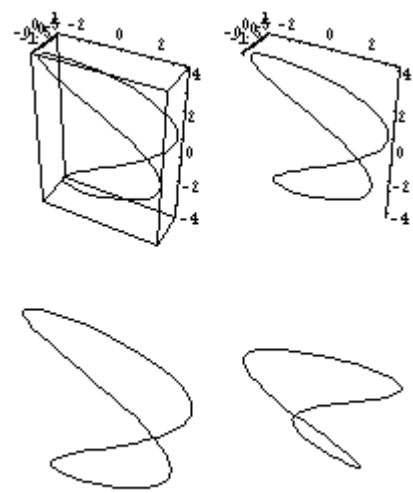
三维参数作图的基本形式为:

<code>ParametricPlot3D[{fx,fv,fz},{t,tmin,tmax}]</code>	给出空间曲线的参数图
<code>ParametricPlot3D[{fx,fv,fz},{t,tmin,tmax},{u,umin,umax}]</code>	给出空间曲面的参数图
<code>ParametricPlot3D[{fx,fv,fz,s}.....]</code>	按照函数关系 s 绘出参数图的阴影部分
<code>ParametricPlot3D[{fx,fv,fz},{gx,gy,gz}.....]</code>	把一些图形绘制在一起

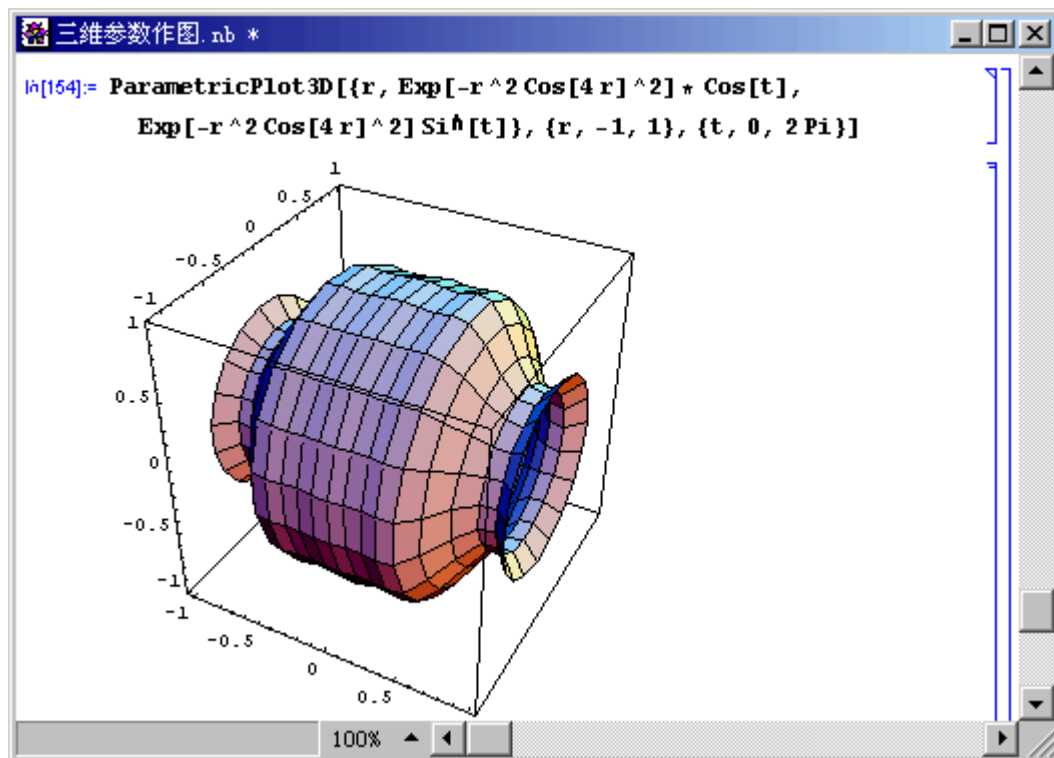
下面是一些空间曲线的例子

```
pp1 := ParametricPlot3D[{3 * Cos[4 * t + 1], Cos[2 * t + 3], 4 Cos[2 * t + 5]}, {t, 0, Pi}];
pp2 := ParametricPlot3D[{3 Cos[4 t + 1], Cos[2 t + 3], 4 Cos[2 t + 5]}, {t, 0, Pi}, Boxed -> False];
pp3 := ParametricPlot3D[{3 Cos[4 t + 1], Cos[2 t + 3], 4 Cos[2 t + 5]}, {t, 0, Pi}, Boxed -> False,
    Axes -> False];
pp4 := ParametricPlot3D[{3 Cos[4 t + 1], Cos[2 t + 3], 4 Cos[2 t + 5]}, {t, 0, Pi}, Boxed -> False,
    Axes -> False, BoxRatios -> {1, 1, 1}];
Show[GraphicsArray[{{pp1, pp2}, {pp3, pp4}}]]
```

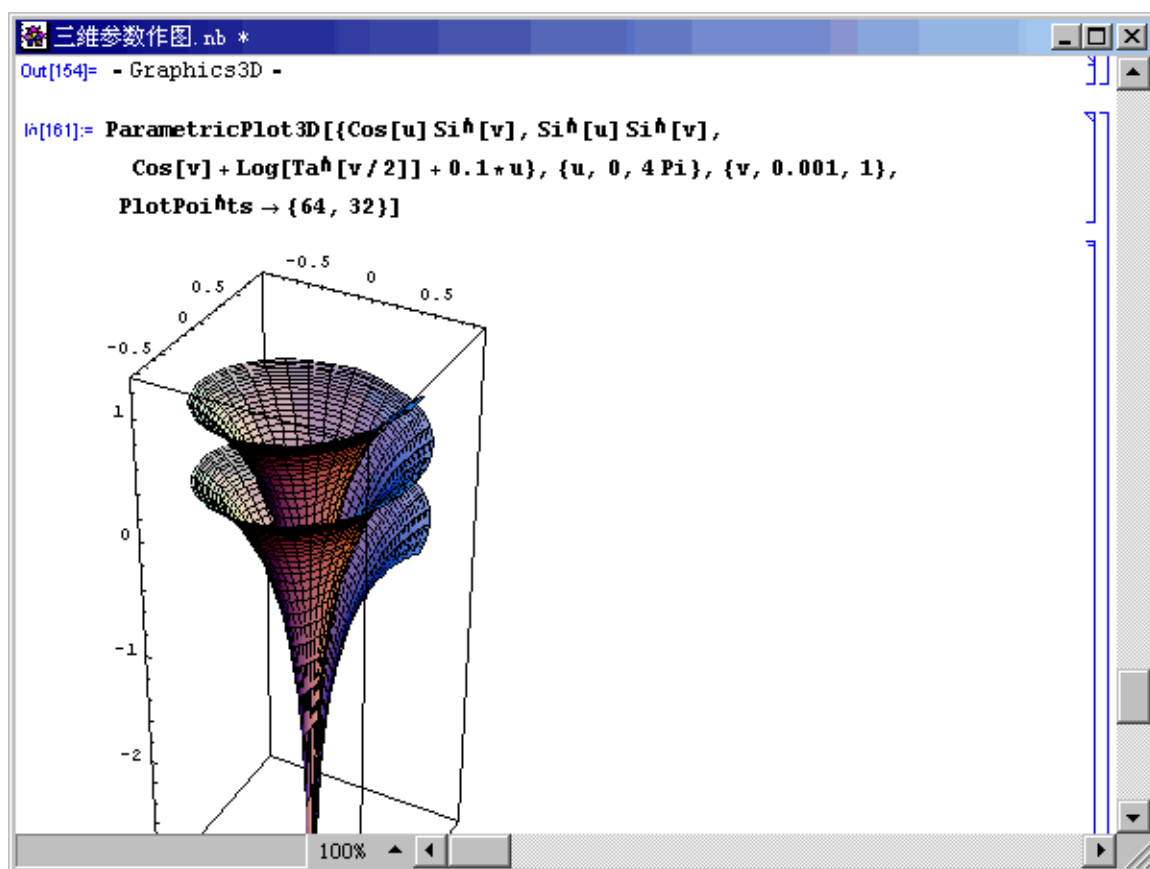
结果为



命令 `ParametricPlot3D[{fx,fv,fz},{t,tmin,tmax}, {u,umin,umax}]` 产生一个曲面而不是一条曲线曲面是由四边形组成。



下面这个图形也很漂亮



5.1 极限



Mathematica 计算极限的命令是 Limit 它的使用方法主要有

Limit[expr,x->x0]	当 x 趋向于 x0 时求 expr 的极限
Limit[expr,x->x0,Direction->1]	当 x 趋向于 x0 时求 expr 的左极限
Limit[expr,x->x0,Direction->-1]	当 x 趋向于 x0 时求 expr 的右极限

趋向的点可以是常数，也可以是 $+\infty$ ， $-\infty$ 例如

1. 求 $\lim_{x \rightarrow \infty} \frac{\sqrt{x^2 + 2}}{3x - 6}$

```
In[1]:= Limit[Sqrt[x^2 + 2] / (3 x - 6), x -> Infinity]
Out[1]:= 1/3
```

2. 求 $\lim_{x \rightarrow 0} \frac{\sin^2 x}{x^2}$

```
In[2]:= Limit[Sin[x]^2 / x^2, x -> 0]
Out[2]:= 1
```

3. 求 $\lim_{x \rightarrow 0^+} \frac{\ln|x|}{x}$

```

求极限.nb *
Out[2]= 1

In[3]:= Limit[Log[x], x -> 0, Direction -> -1]

Out[3]= -∞

```

5.2 微分



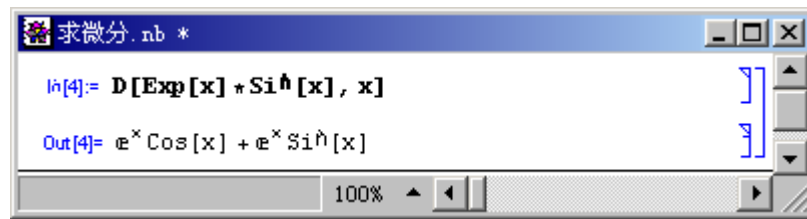
1. 函数的微分

在 Mathematica 中，计算函数的微分或是非常方便的，命令为 $D[f,x]$, 表示对 x 求函数 f 的导数或偏导数。该函数的常用格式有以下几种

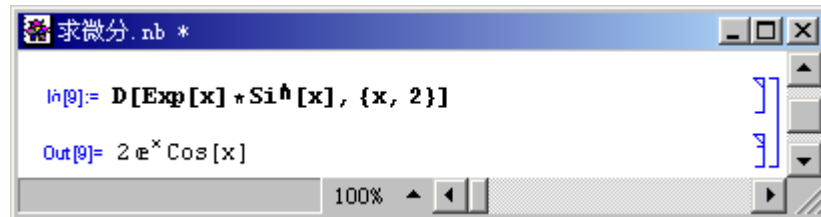
$D[f,x]$	$\frac{\partial f}{\partial x}$ 计算微分
$D[f,x_1,x_2,\dots]$	$\frac{\partial}{\partial x_1} \frac{\partial}{\partial x_2} f$ 计算多重偏微分
$D[f,\{x,n\}]$	$\frac{\partial^n}{\partial x^n} f$ 计算 n 阶微分
$D[f,x, \text{NonConstants} \rightarrow \{v_1, v_2, \dots\}]$	$\frac{\partial f}{\partial x}$ 其中 v_1, v_2, \dots 依赖于 x 计算微分

例如

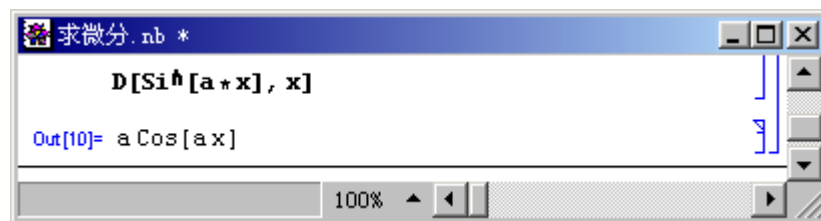
1. 求函数 $\sin x$ 的导数



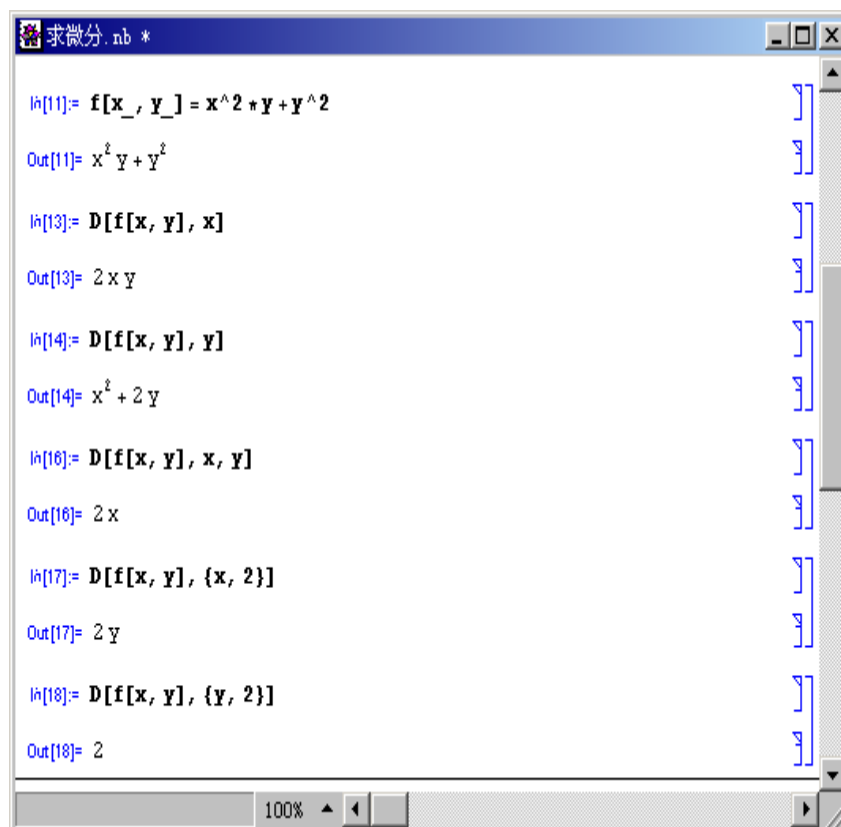
2.求函数 $e^x \sin x$ 的2阶导数



3.假设 a 是常数可以对 $\sin ax$ 求导

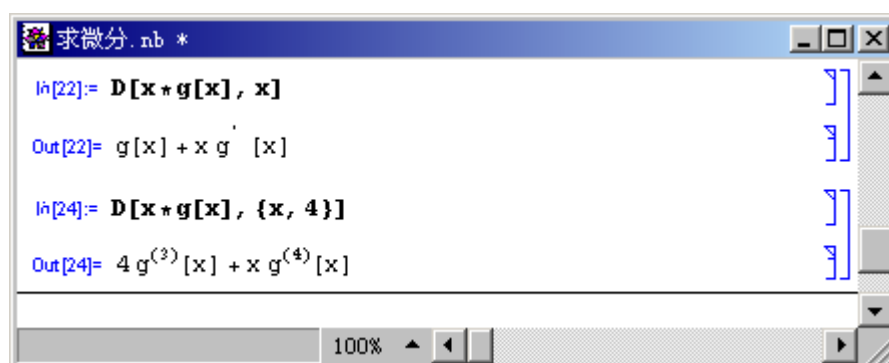


4.如果对二元函数 $f(x,y)=x^2*y+y^2$ 求对 x,y 求一阶和二阶偏导

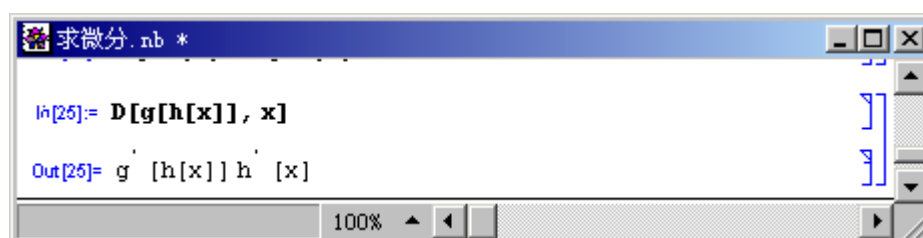


Mathematica可以求函数式未知的函数微分，通常结果使用数学上的表示法

例如：



对链导法则同样可用



如果要得到函数在某一点的导数值可以把这点代入导数如：

```

求微分.nb *

In[30]:= D[Exp[x]*Sin[x], x] /. x -> 2
Out[30]= e^2 Cos[2] + e^2 Sin[2]

In[31]:= N[%]
Out[31]= 3.64392

```

2.全微分

在 Mathematica 中, $D[f,x]$ 给出 f 的偏导数, 其中假定 f 中的其他变量与 x 无关。当 f 为单变量时, $D[f,x]$ 计算 f 对 x 的导数。函数 $Dt[f,x]$ 给出 f 的全微分形式, 并假定 f 中所有变量依赖于 x 。下面是 Dt 命令的常用形及意义

$Dt[f]$	求全微分 df
$Dt[f,x]$	$\frac{df}{dx}$ 求全微分
$Dt[f,x_1,x_2,\dots]$	$\frac{d}{dx_1} \frac{d}{dx_2} f$ 求多重全微分
$Dt[f,x,Constants \rightarrow \{c_1, c_2, \dots\}]$	求全微分其中 c_1, c_2, \dots 是常数

下面我们求 x^2+y^2 的偏微分 and 全微分

```

求微分.nb *

In[32]:= D[x^2 + y^2, x]
Out[32]= 2 x

In[33]:= Dt[x^2 + y^2, x]
Out[33]= 2 x + 2 y Dt[y, x]

```

可以看出第一种情况 y 与 x 没有关系, 第二种情况 y 是 x 的函数。再看下列求多项式 x^2+xy^3+yz 的全微分并假定 z 保持不变是常数。


```

求微分.nb *
In[36]:= Dt[x^2 + xy^3 + yz, Constants -> {z}]
Out[36]:= 2 x Dt[x, Constants -> {z}] +
          3 xy^2 Dt[y, Constants -> {z}] + Dt[yz, Constants -> {z}]

```

如果 y 是 x 的函数那么，y 被看成是常数

```

求微分.nb *
In[38]:=
          Dt[x^2 + xy[x] + y[x] z]
Out[38]:= 2 x Dt[x] + Dt[z] y[x] + Dt[x] xy'[x] + z Dt[x] y'[x]

```

5.3 计算积分

1. 不定积分

在 Mathematica 中计算不定积分命令为 Integrate[f,x],当然也可使用工具栏直接输入不定积分式。来

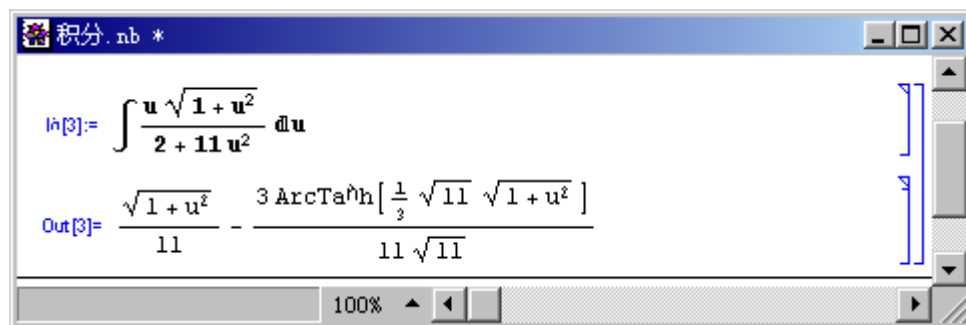
求函数的不定积分。当然并不是所有的不定积分都能求出来。例如若求 $\int \sin x \sin x dx$ Mathematica 就无能为力。

```

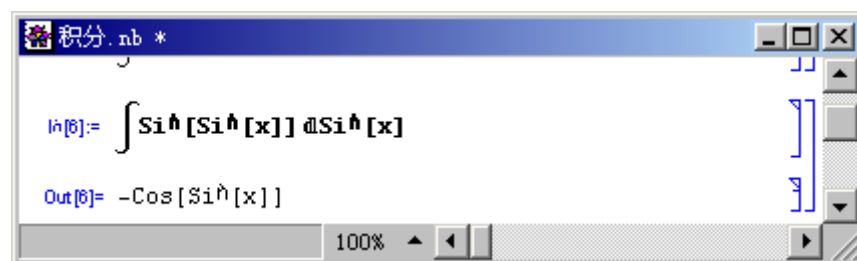
积分.nb *
In[2]:= Integrate[Sin[Sin[x]], x]
Out[2]:= ∫ Sin[Sin[x]] dx

```

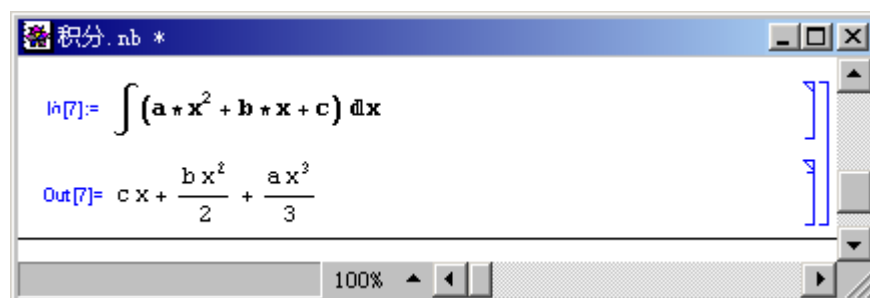
但对于一些手工计算相当复杂的不定积分，MatheMatica 还是能轻易求得，例如求 $\int \frac{u\sqrt{1+u^2}}{2+11u^2} du$



积分变量的形式也可以是一函数，例如



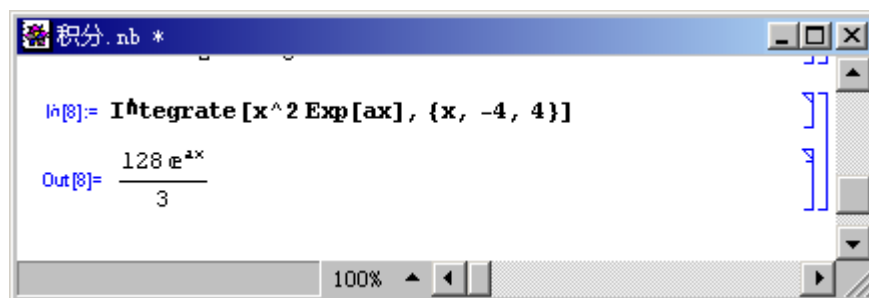
输入命令也可求得正确结果。对于在函数中出现的除积分变量外的函数，统统当作常数处理，请看下面例子。



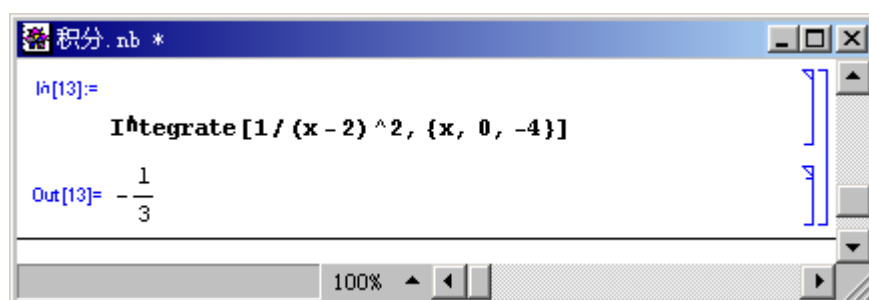
2.定积分

定积分的求解主要命令也是用 Integrate 只是要在命令中加入积分限 Integrate[f,{x,min,max}]

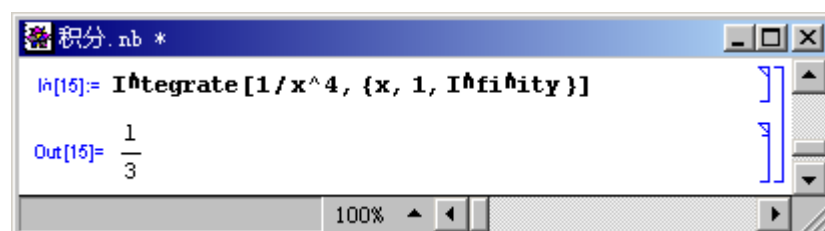
或者使用式具栏输入也可以。例如求 $\int_4^4 x^2 e^{ax} dx$



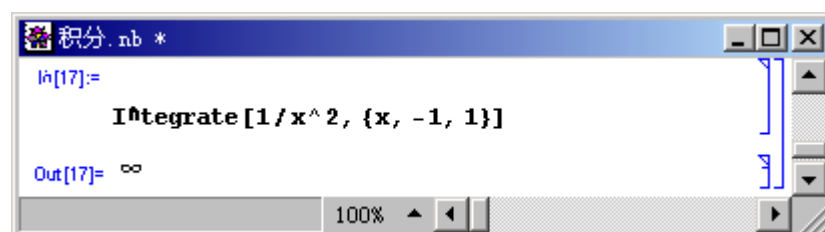
显然这条命令也可以求广义积分例如：求 $\int_0^4 \frac{1}{(x-2)^2} dx$



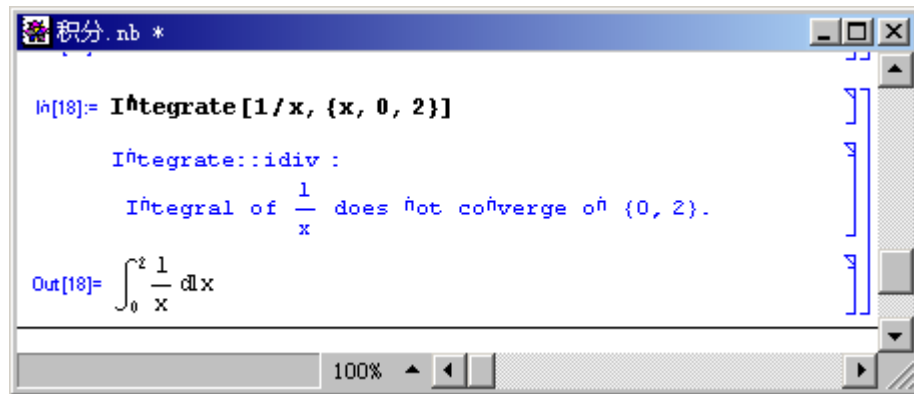
求无穷积也可以例如 $\int_1^{+\infty} \frac{1}{x^4} dx$



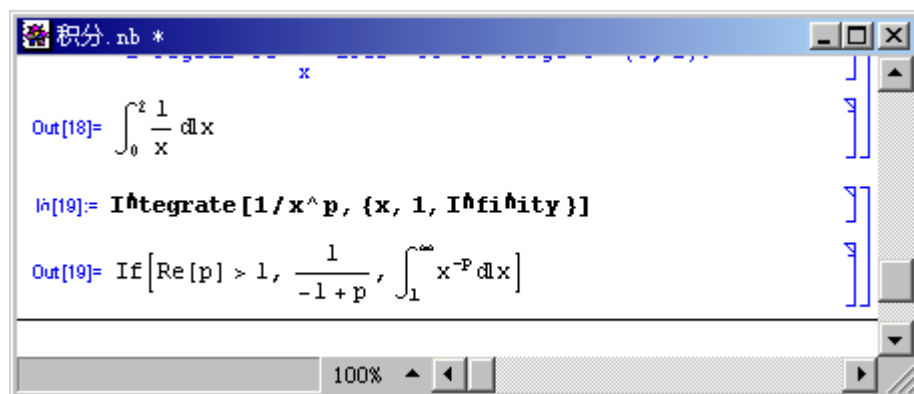
如果广义积发散也能给出结果，例如



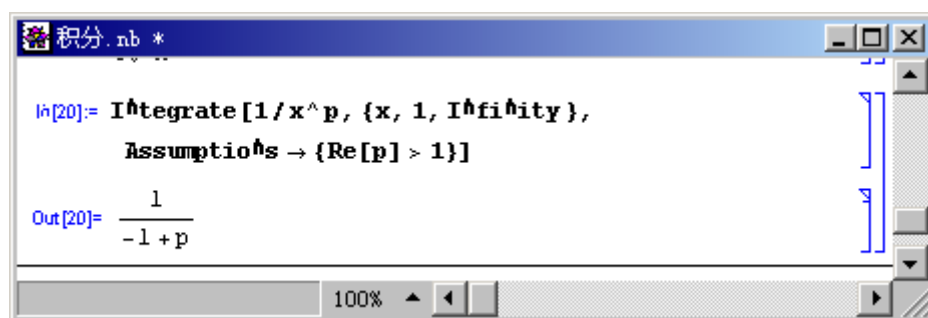
如果无法判定敛散性，就用给出一个提示，例如



如果广义积分敛散性与某个符号的取值有关，它也能给出在不同情况下的积分结果例如 $\int_1^{+\infty} \frac{1}{x^p} dx$



结果的意义是当 $|p|>1$ 时，积分值为 $1/1-p$,否则不收敛。在 Integrate 中可加两个参数 Assumptions 和 GenerateConditions 例如上例中，只要用 Assumptions->{Re[p]>1}就可以得到收敛情况的解



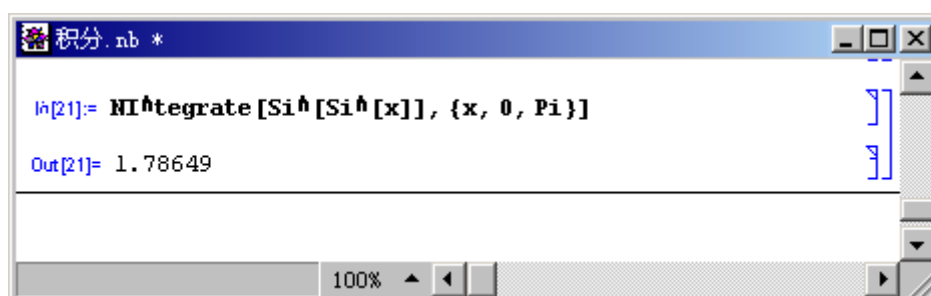
3.数值积分

数值积分是解决求定积分的另一种有效的方法，它可以给出一个近似解。特别是对于用 Integrate 命令无法求出的定积分，数值积分更是可以发挥巨大作用。

它的命令格式为

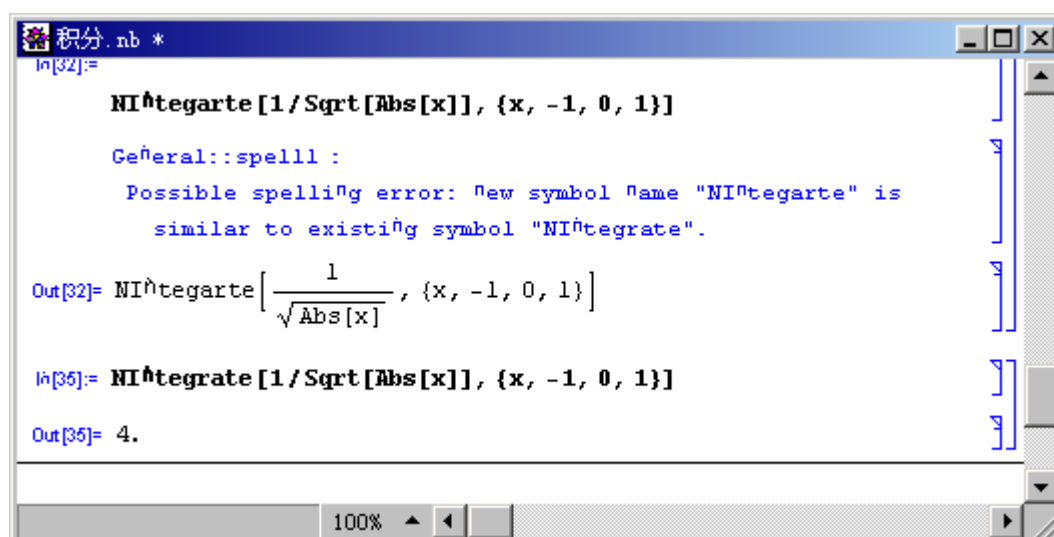
<code>NIntegrate[f,{x,a,b}]</code>	在[a,b]上求 f 数值积分
<code>NIntegrate[f,{x,a,x1,x2,...,b}]</code>	以 x1,x2...为分割求[a,b]上的数值积分
<code>NIntegrate[f,{x,a,b},MaxRecursion->n]</code>	求数值积分时指定迭代次数 n.

下面我们求 $\sin \sin x$ 在 $[0, \pi]$ 上的积分值, 由于这个函数的不定积分求不出, 因此使用 `Integrate` 命令无法得到具体结果, 但可以用数值积分求




如果积分函数存在不连续点, 或存在奇点我们可对积分进行分段求解。例如函数 $\frac{1}{\sqrt{|x|}}$ 在 $[-1, 1]$ 上, 显然 $x=0$ 点是一

个无穷间断点。因此若要求其数值积分, 必须在其插入点 0



对无穷积分, 也可求数值积分, 例如。

```
积分.nb *
In[37]:= NIntegrate[Exp[-x^2], {x, 0, Infinity}]
Out[37]= 0.886227
```



多变量函数的微分

下面是计算多变量函数的偏导数及全微分的命令与单变量基本相同，通过分析下面的例子我们可以轻松掌握。

(1) $D[f, x_1, x_2, \dots, x_n]$ 计算偏导数 $\frac{\partial^n f}{\partial x_1 \partial x_2 \dots \partial x_n}$

下面是实际的例子：

```

多变量函数的微分.nb *

(* 求函数 Sin[x+y^2] 对 x 的偏导数 *)
In[1]:= D[Sin[x+y^2], x]
Out[1]= y^2 Cos[x y^2]

(* 求函数 Sin[x+y^2] 对 x 的二阶偏导数 *)
In[2]:= D[Sin[x+y^2], x, x]
Out[2]= -y^4 Sin[x y^2]

(* 上述命令也可写成如下形式 *)
In[3]:= D[Sin[x+y^2], {x, 2}]
Out[3]= -y^4 Sin[x y^2]

(* 求函数 Sin[x+y^2] 对 x 的二阶对 y 的一阶混合偏导数 *)
In[4]:= D[Sin[x+y^2], x, x, y]
Out[4]= -2 x y^5 Cos[x y^2] - 4 y^3 Sin[x y^2]

(* 上述命令也可写成如下形式 *)
In[5]:= D[Sin[x+y^2], {x, 2}, y]
Out[5]= -2 x y^5 Cos[x y^2] - 4 y^3 Sin[x y^2]

Time: 0.32 seconds 100%

```

(II) $D[f, x, \text{NonConstants} \rightarrow \{v_1, v_2, \dots\}]$ 中 v_i 依赖于 x .

下面是实际的例子:

```

多变量函数的微分.nb *

In[31]:= D[x^2 + y^2, x, NonConstants -> {y}]
Out[31]= 2 x + 2 y D[y, x, NonConstants -> {y}]

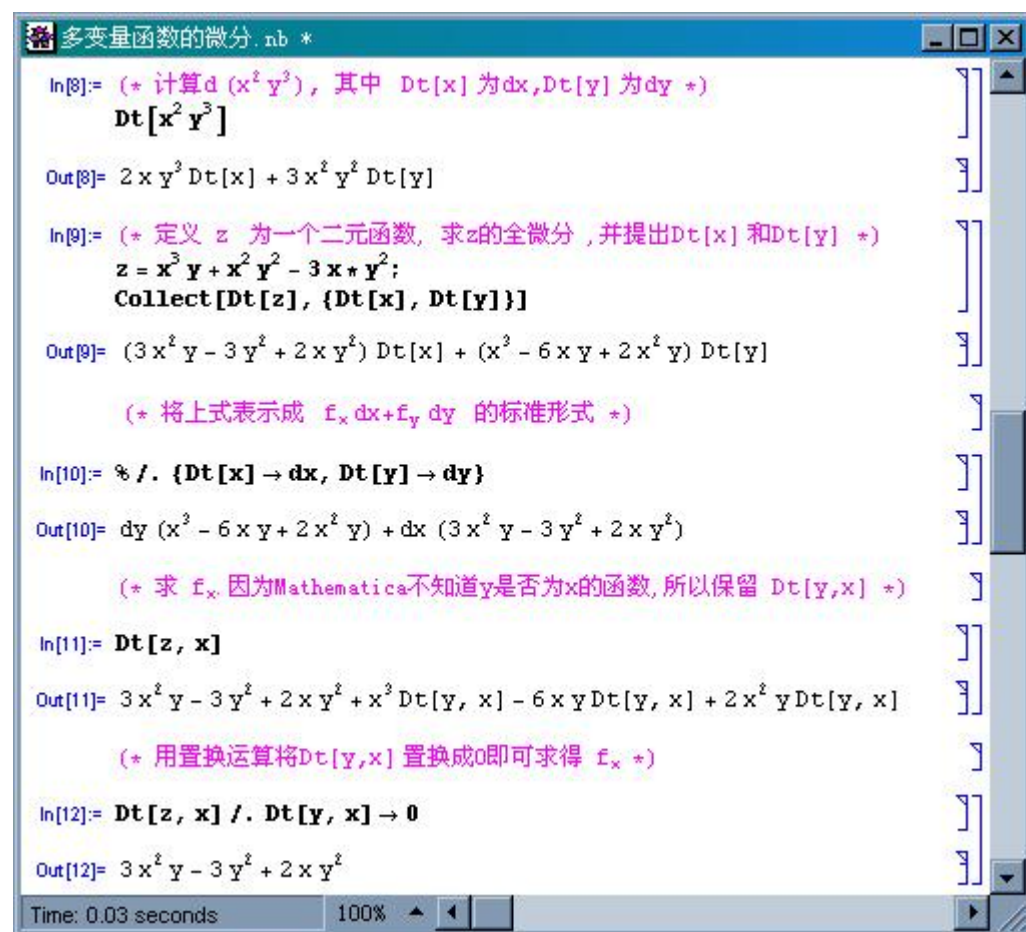
(* 注: D[y, x, NonConstants -> {y}] 表示  $\frac{\partial y}{\partial x}$ , 其中 y 是 x 的函数 *)

Time: 0.03 seconds 100%

```

(III) $D[f]$ 计算全微分 df .

下面是实际的例子:



```
多变量函数的微分.nb *

In[8]:= (* 计算d (x^2 y^3), 其中 Dt[x] 为dx, Dt[y] 为dy *)
Dt[x^2 y^3]

Out[8]= 2 x y^3 Dt[x] + 3 x^2 y^2 Dt[y]

In[9]:= (* 定义 z 为一个二元函数, 求z的全微分, 并提出Dt[x] 和Dt[y] *)
z = x^3 y + x^2 y^2 - 3 x y^2;
Collect[Dt[z], {Dt[x], Dt[y]}]

Out[9]= (3 x^2 y - 3 y^2 + 2 x y^2) Dt[x] + (x^3 - 6 x y + 2 x^2 y) Dt[y]

(* 将上式表示成 f_x dx+f_y dy 的标准形式 *)

In[10]:= % /. {Dt[x] -> dx, Dt[y] -> dy}

Out[10]= dy (x^3 - 6 x y + 2 x^2 y) + dx (3 x^2 y - 3 y^2 + 2 x y^2)

(* 求 f_x, 因为Mathematica不知道y是否为x的函数, 所以保留 Dt[y, x] *)

In[11]:= Dt[z, x]

Out[11]= 3 x^2 y - 3 y^2 + 2 x y^2 + x^3 Dt[y, x] - 6 x y Dt[y, x] + 2 x^2 y Dt[y, x]

(* 用置换运算将Dt[y, x] 置换成0即可求得 f_x *)

In[12]:= Dt[z, x] /. Dt[y, x] -> 0

Out[12]= 3 x^2 y - 3 y^2 + 2 x y^2

Time: 0.03 seconds 100%
```

(IV) 求隐函数的导数

下面是实际的例子:


```

(* 求隐函数  $5y^2 + \sin[y] = x^2$  的导数 *)
In[32]:= Dt[5 y^2 + Sin[y] == x^2, x]
Out[32]= 10 y Dt[y, x] + Cos[y] Dt[y, x] == 2 x
In[33]:= Solve[%, Dt[y, x]]
Out[33]= {{Dt[y, x] ->  $\frac{2x}{10y + \cos[y]}$ }}
Time: 0.47 seconds    100%

```

(V) $\text{Dt}[f, x, \text{Constants} \rightarrow \{c_1, c_2, \dots\}]$ 计算全微分 df , 其中 c_i 是常数.

下面是实际的例子:

```

In[20]:= Clear[y];
          Dt[x^2 + y^2 + z^2, x, Constants -> {z}]
Out[21]= 2 x + 2 y Dt[y, x, Constants -> {z}]
Time: 0.02 seconds    100%

```

(VI) $\text{Dt}[f, x_1, x_2, \dots, x_n]$ 计算多重全微分 $\frac{d^n f}{dx_1 dx_2 \dots dx_n}$.

下面是实际的例子:

```

多变量函数的微分.nb *
In[26]:= z = x^3 y + x^2 y^2 - 3 x * y^2;
(* 计算多重全微分  $\frac{dz}{dx dy}$  *)

In[27]:= Dt[z, x, y]
Out[27]= 3 x^2 - 6 y + 4 x y + 6 x y Dt[x, y] + 2 y^2 Dt[x, y] -
        6 x Dt[y, x] + 2 x^2 Dt[y, x] + 3 x^2 Dt[x, y] Dt[y, x] -
        6 y Dt[x, y] Dt[y, x] + 4 x y Dt[x, y] Dt[y, x]
Time: 0.05 seconds    100%

```



多变量函数的积分 (重积分)

多变量函数的积分类似于一元函数的积分,可以利用 Integrate 函数来完成.命令如下:

Integrate[f,{x,a,b},{y,c,d},...,{z,m,n}] 计算重积分 $\int_a^b \int_c^d \dots \int_m^n f(x,y,\dots,z) dz \dots dy dx$.

NIntegrate[f,{x,a,b},{y,c,d},...,{z,m,n}] 数值积分或重积分的数值解.

下面是具体的例子:

```

多变量函数的积分.nb *

(* 计算重积分  $\iint \frac{1}{x^2+y+1} dx dy$  *)

In[35]:=  $\iint \frac{1}{x^2+y+1} dx dy$ 

Out[35]=  $2\sqrt{1+y} \operatorname{ArcTan}\left[\frac{x}{\sqrt{1+y}}\right] + x \operatorname{Log}[1+x^2+y]$ 

(* 我们也可以直接输入Integrate命令进行积分, 但要注意x与y 的顺序 *)

In[36]:= Integrate[1/(x^2+y+1), y, x]

Out[36]=  $2\sqrt{1+y} \operatorname{ArcTan}\left[\frac{x}{\sqrt{1+y}}\right] + x \operatorname{Log}[1+x^2+y]$ 

(* 计算二重积分  $\int_0^a \int_0^b (x^2+y^2) dx dy$  *)

In[37]:= Integrate[x^2+y^2, {x, 0, a}, {y, 0, b}]

Out[37]=  $\frac{a^3 b}{3} + \frac{a b^3}{3}$ 

(* y 的积分限也可以是 x 的函数 *)

In[38]:= Integrate[x^2+y^2, {x, 0, a}, {y, 0, x^2}]

Out[38]=  $\frac{a^5}{5} + \frac{a^7}{21}$ 

Time: 0.09 seconds 100%

```

以下是数值积分的例子:

```

(* 在重积分中,无法求出某个变量的积分值, Mathematica
会求出可积分的部分,再输出运算的结果 *)

In[52]:= Integrate[Sqrt[x + y], {x, 0, 2}, {y, 0, Sqrt[x + 2]}]

Out[52]=  $\int_0^2 \left( -\frac{2x^{3/2}}{3} + \frac{2}{3} (x + \sqrt{2+x})^{3/2} \right) dx$ 

In[53]:= (* 将上式转换成数值解 *)
N[%]

Out[53]= 4.65557

(* 直接利用NIntegrate命令求解, 也可以得到相同的答案 *)

In[54]:= NIntegrate[Sqrt[x + y], {x, 0, 2}, {y, 0, Sqrt[x + 2]}]

Out[54]= 4.65557

(* 以下是一个三重积分  $\int_{-2}^2 \int_{x^2}^4 \int_{-\sqrt{y-x^2}}^{\sqrt{y-x^2}} \sqrt{x^2+z^2} dz dy dx$  *)

In[55]:= Off[NIntegrate::slwcon]
NIntegrate[Sqrt[x^2 + z^2], {x, -2, 2}, {y, x^2, 4},
{z, -Sqrt[y - x^2], Sqrt[y - x^2]}]

Out[56]= 26.8083

(* 注:命令 Off[NIntegrate::slwcon] 的作用是不显示提示信息 *)

```

Time: 13.07 seconds 100%



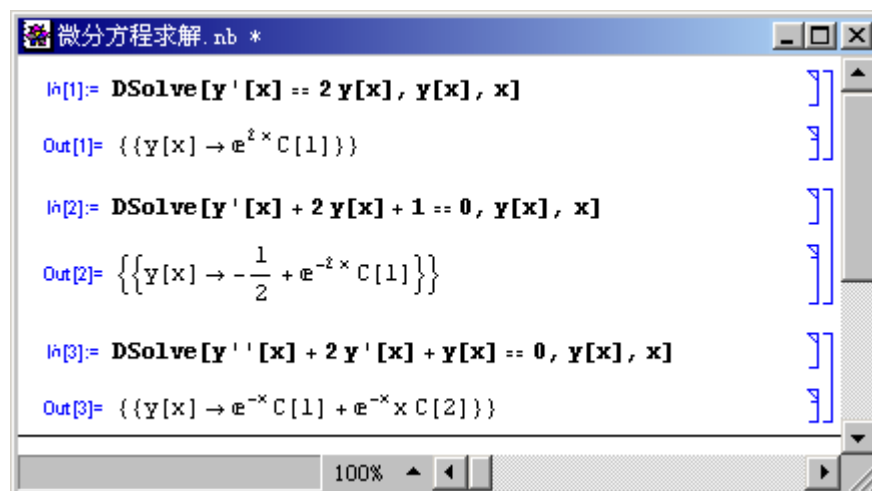
6. 1 微分方程解

在 Mathematica 中使用 Dsolve[] 可以求解线性和非线性微分方程, 以及联立的微分方程组。在没有给定方程的初值条件下, 我们所得到的解包括 C[1], C[2] 是待定系数。求解微分方程就是寻找未知的函数的表达式, 在 Mathematica 中, 未稳中有降函数用 y[x] 表示, 其微分用 y'[x], y''[x] 等表示。

下面给出微分方程 (组) 的求解函数。

Dsolve[eqn, y[x], x]	求解微分方程 y[x]
Dsolve[eqn, y, x]	求解微分方程函数 y
Dsolve[{eqn1, eqn2, ...}, {y1, y2, ...}, x]	求解微分方程组

1. 用 Dsolve 求解微分方程 y[x]



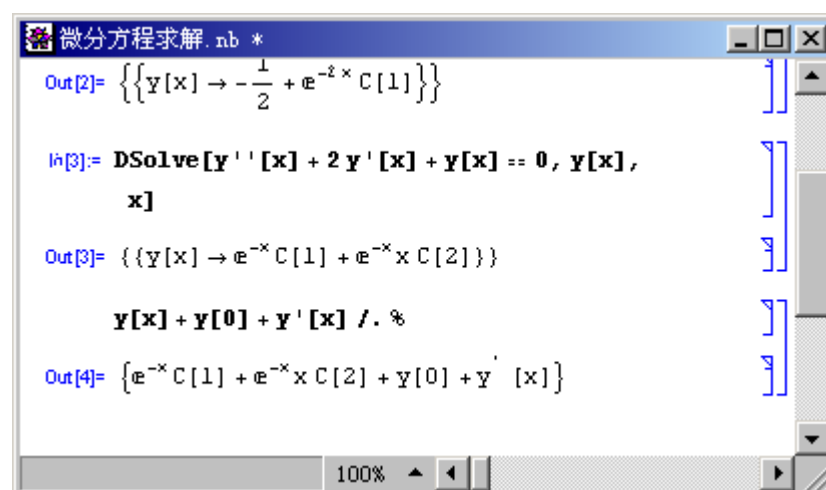
```
微分方程求解.nb *

In[1]:= DSolve[y' [x] == 2 y[x], y[x], x]
Out[1]= {{y[x] -> e^{2 x} C[1]}}

In[2]:= DSolve[y' [x] + 2 y[x] + 1 == 0, y[x], x]
Out[2]= {{y[x] -> -\frac{1}{2} + e^{-2 x} C[1]}}

In[3]:= DSolve[y' '[x] + 2 y' [x] + y[x] == 0, y[x], x]
Out[3]= {{y[x] -> e^{-x} C[1] + e^{-x} x C[2]}}
```

解 y[x]仅适合其本身，并不适合于 y[x]的其它形式，如 y'[x], y[0]等，也就是说 y[x]不是函数，例如我们如果有如下操作，y'[x],y[0]并没有发生变化。



```
微分方程求解.nb *

Out[2]= {{y[x] -> -\frac{1}{2} + e^{-2 x} C[1]}}

In[3]:= DSolve[y' '[x] + 2 y' [x] + y[x] == 0, y[x],
              x]
Out[3]= {{y[x] -> e^{-x} C[1] + e^{-x} x C[2]}}

y[x] + y[0] + y' [x] /. %
Out[4]= {e^{-x} C[1] + e^{-x} x C[2] + y[0] + y' [x]}
```

2. 解的纯函数形式

使用 Dsolve 命令可以给出解的纯函数形式，即 y,请分析下面的例子

```

In[4]:= DSolve[Y'[x] == 2 Y[x], Y[x], x]
Out[4]= {{Y[x] -> e^{2 x} C[1]}}

In[5]:= DSolve[Y'[x] + 2 Y[x] + 1 == 0, Y, x]
Out[5]= {{Y -> (-1/2 + e^{-2 #1} C[1]) &}}

In[6]:= DSolve[Y''[x] == 2 Y[x], Y, x]
Out[6]= {{Y -> (e^{x #1} C[1] &)}}

In[7]:= DSolve[Y''[x] + 2 Y'[x] + Y[x] == 0, Y, x]
Out[7]= {{Y -> (e^{-#1} C[1] + e^{-#1} C[2] #1 &)}}

```

这里 y 适合 y 的所有情况下面的例子可以说明这一点

```

In[13]:= DSolve[Y''[x] + 2 Y'[x] + Y[x] == 0, Y, x]
Out[13]= {{Y -> (e^{-#1} C[1] + e^{-#1} C[2] #1 &)}}

In[14]:= Y[x] + Y'[x] + Y[0] /. %
Out[14]= {C[1] + e^{-x} C[2]}

```

在标准数学表达式中，直接引入亚变量表示函数自变量，用此方法可以生成微分方程的解。如果需要的是解的符号形式，引入这样来变量很方便。然而，如果想在其他的计算中使用该结果，那么最好使用不带亚变量的纯函数形式的结果。

3. 求微分方程组

请分析下面的例子

```

In[16]:= DSolve[{y[x] == -z'[x], z[x] == -y'[x]},
               {y[x], z[x]}, x]

Out[16]:= {{y[x] -> 1/2 e^{-x} (C[1] + e^{2x} C[1] + C[2] - e^{2x} C[2]),
            z[x] -> -1/2 e^{-x} (-C[1] + e^{2x} C[1] - C[2] - e^{2x} C[2])}}

```

当然微分方程组也有纯函数形式。

```

In[16]:= DSolve[{y[x] == -z'[x], z[x] == -y'[x]},
               {y[x], z[x]}, x]

Out[16]:= {{y[x] -> 1/2 e^{-x} (C[1] + e^{2x} C[1] + C[2] - e^{2x} C[2]),
            z[x] -> -1/2 e^{-x} (-C[1] + e^{2x} C[1] - C[2] - e^{2x} C[2])}}

```

4. 带初始条件的微分方程的解

当给定一个微分方程的初始条件可以确定一个待定系数。请看下面的例子

```

In[18]:= DSolve[{y'[x] == y[x], y[0] == 5}, y[x], x]

Out[18]:= {{y[x] -> 5 e^x}}

In[19]:= DSolve[{y'[x] == y[x], y'[0] == 0}, y[x], x]

Out[19]:= {{y[x] -> e^{-x} (C[2] + e^{2x} C[2])}}

```

第二个例子由于给出一个初始条件所以只能确定 C[1].

5. 进一步讨论

对于简单的微分方程的解比较简单，对一些微分方程它的解就复杂的多。特别是对一些微分方程组或高阶微分方程，不一定能得具体的解，其解中可能含有一些特殊函数。并且很多特殊函数的提出就是为了解这些方程的如：

```

In[23]:= DSolve[y'[x] - 2 x * y[x] == 1, y[x], x]
Out[23]= {{y[x] -> e^{x^2} C[1] + 1/2 e^{x^2} sqrt(pi) Erf[x]}}

In[24]:= DSolve[y''[x] - x * y[x] == 0, y[x], x]
Out[24]= {{y[x] -> AiryAi[x] C[1] + AiryBi[x] C[2]}}

In[25]:= DSolve[y''[x] - Exp[x] y[x] == 0, y[x], x]
Out[25]= {{y[x] ->
          BesselI[0, 2 sqrt{e^x}] C[1] + BesselK[0, 2 sqrt{e^x}] C[2]}}

```

上面三个方程中分别使用了三种类型的函数，可以查看系统帮助了解他们的性质和含义。对于非线性微分方程，仅有一些特殊的情况可用标准数学函数得到解。Dsolve 能够处理所有在标准数学手册有解的非线性微分方程。例如：

```

In[31]:= DSolve[y'[x] - y[x]^2 == 0, y[x], x]
Out[31]= {{y[x] -> 1/(-x + C[1])}}

In[33]:= DSolve[y'[x] - y[x]^2 == x, y[x], x]
Out[33]= {{y[x] -> -((-1)^{1/3} (AiryBiPrime[(-1)^{1/3} x] +
          AiryAiPrime[(-1)^{1/3} x] C[1])) /
          (AiryBi[(-1)^{1/3} x] + AiryAi[(-1)^{1/3} x] C[1])}}

```


可以看出第二个方程的解已经非常复杂。

6.2 微分方程的数值解

在 Mathematica 中用函数 DSolve[] 得到微分方程的准确解, 用函数 NDSolve 得到微分方程的数值解, 当然在此处要给出求解区间 (x, xmin, xmax)。

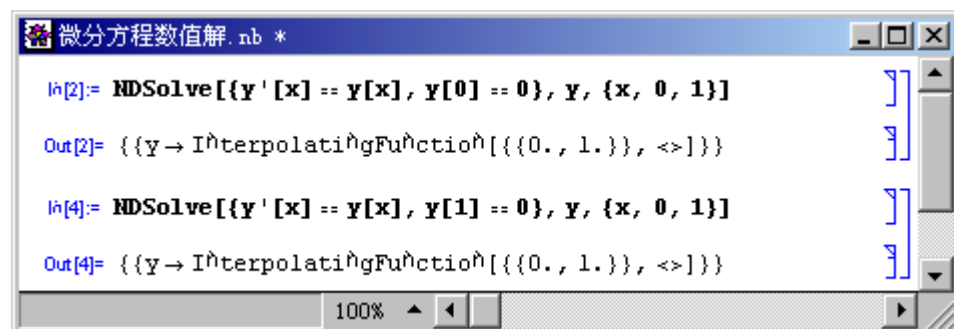
NDSolve 也是既能计算单个的微分方程, 也能计算联立微分方程组。它可对大多数的常微分方程和部分偏微分方程求解。在常微分可能有一些未知函数 yi, 但这些未知函数都依赖于一个单变量 x。

`NDSolve[{eqn1, eqn2, ...}, y, {x, xmin, xmax}]` 求函数 y 的数值解, x 属于 [xmin, xmax]

`NDSolve[{eqn1, eqn2, ...}, {y1, y2, ...}, {x, xmin, xmax}]` 求多个函数 yi 的数值解

NDSolve 以 InterpolatingFunction 为目标生成函数 yi 的解, InterpolatingFunction 目标提供在独立变量 x 的 xmin 到 xmax 范围内求出的近似值。NDSolve 用迭代法求解, 它以某一个 x 值开始, 尽可能覆盖从 xmin 到 xmax 的全区间。

为使迭代开始, NDSolve 指定 yi 及其导数为初始条件。初始条件给定某点 x 处的 yi[x] 及尽可能的导数 yi'[x], 一般情况下, 初始条件可在任意 x 处, NDSolve 将以此起点自动覆盖 xmin 到 xmax 的全区域。下面对初始条件 y[0]=0 和 y[1]=0 分别求出 x 从 0 到 1 的范围内 y'[x]=y[x] 的解。



```
In[2]:= NDSolve[{y'[x] == y[x], y[0] == 0}, y, {x, 0, 1}]
Out[2]:= {{y -> InterpolatingFunction[{{0., 1.}}, <>]}}

In[4]:= NDSolve[{y'[x] == y[x], y[1] == 0}, y, {x, 0, 1}]
Out[4]:= {{y -> InterpolatingFunction[{{0., 1.}}, <>]}}
```

再看下面的微分方程的数值解

```
微分方程数值解.nb *

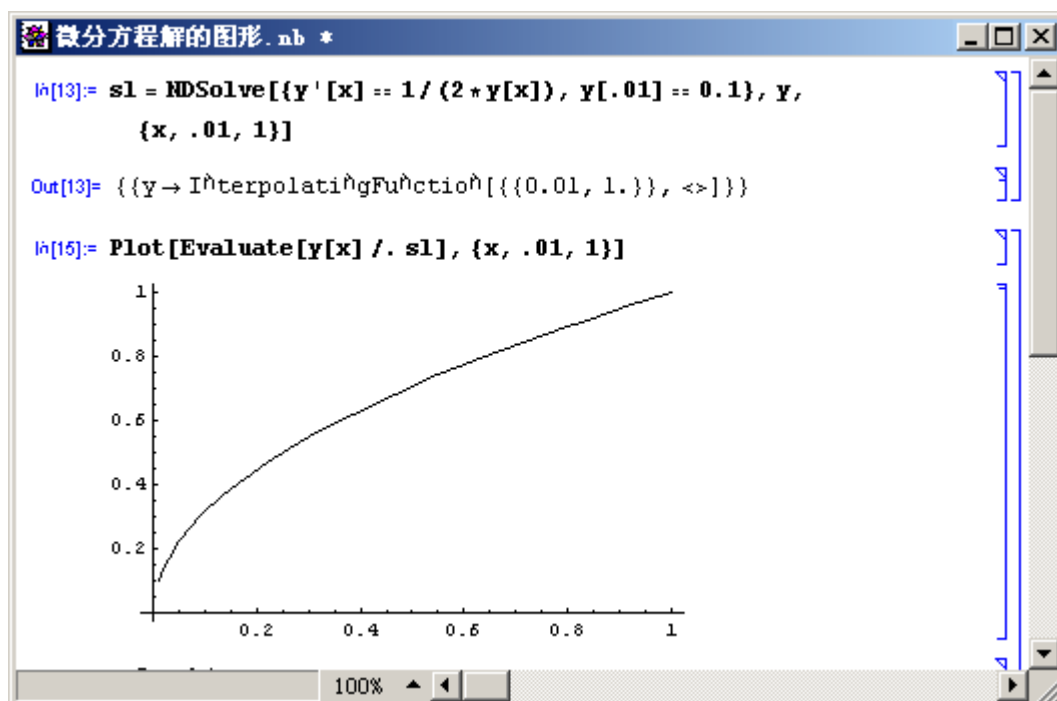
In[5]:= s = NDSolve[{y'[x] == y[x]^2, y[0] == 1}, y, {x, 0, 1}]

NDSolve::npsz: At x == 0.9999817475261439, step
size is effectively zero; singularity suspected.

Out[5]:= {{y -> InterpolatingFunction[{{0., 0.999982}}, <>]}}
```

使用Mathematica可以很容易的得到解的图形。这儿给出如何观察微商的逆函数的近似值图形。我们使用命令Evaluate代替InterpolatingFunction能够节省时间。

例如:



7.1 模块和块中的变量

前面我们学习了有关 Mathematica 的各种基本运算及操作,为了使 Mathematica 更有效的工作,我们可对 Mathematica 进行模块化运算。在模块内部通过编写一系列表达式语句,使其实现一定的功能。在 Mathematica 内部也提供了很多程序包,我们将学习如何调用它们。

一般情况下,Mathematica 假设所有变量都为全局变量。也就是说无论何时你使用一个你定义的变量,Mathematica 都假设你指的是同一个目标。然而在编制程序时,你则不会想把所有的变量当作全局变

量,因为如果这样程序可能就不具有通用性,你也可能在调用程序时陷入混乱状态。给出定义模块或块和局部变量的常用

形式:

Module[{x,y,...},body]	具有局部变量 x,y... 的模块
Module[{x=x0,y=y0,...},body]	具有初始值的局部变量的模块
lhs:=Module[vars,rhs/:cond]	rhs 和 cond 共享局部变量
Block[{x,y,...	},body]运用局部值 x,y, ... 计算 body
Block[{x=x0,y=y0,...},bddy]	给 x,y,..赋初始值

Mathematica 中的模块工作很简单,每当使用模块时,就产生一个新的符号来表示它的每一个局部变量。产生的新符号具有唯一的名字,互不冲突,有效的保护了模块内外的每个变量的作用范围。首先我们来看 Module 函数,这个函数的第一部分参数,里说明的 变量只在 Module 内起作用, body 执行体,包含合法的 Mathematica 语句, 多个语句之间可用“;”分割下面定义有初值的变量 t,Mathematica 默认它为全局变量:

In[1]:=t=10

Out[1]=10

模块中的 t 为局部变量,因此它独立于全局变量 t

In[2]:=Module[{t},t=8;Print[t]]

全局变量 t 的值仍为 10:

In[3]=t=10

Out[3]=10

下面定义函数中的中间变量 t 为局部变量并调用 f:

```

In[19]:= f[v_] := Module[{t}, t = (1 + v) ^ 2; Expand[t]]

In[20]:=
      f[a]

Out[20]= 1 + 2 a + a^2

```

全局变量 t 的值仍为 10:

In[6]:=t=10

Out[6]=10

我们可以对模块中的任意局部变量进行初始化,这些初始值总是在模块执行前就被计算出来。下面给局部变量 t 赋初值 u:调用函数 g;

```

In[24]:= g[u_] := Module[{t = u}, t = t + t / (1 + u)]

In[25]:= g[a + b]

Out[25]= a + b +  $\frac{a + b}{1 + a + b}$ 

```

Mathematica 中的模块允许你把某变量名看作局部变量名。然而又存在有时你又希望它们为全局变量时,但变量值为局部的矛盾,这时我们可以用 Block[]函数。下面是一个含有全局变量 x 表达式,使用 x 的局部值计算上面的表达式:

```

In[26]:= x^2 + 1

Out[26]= 1 + x^2

In[27]:= Block[{x = a + 1}, %]

Out[27]= 1 + (1 + a)^2

In[28]:= x

Out[28]= x

```

在 Mathematica 中编制程序时,必须使程序中的各个部分尽可能的独立,这样程序才便于读懂、维护和修改。确保程序各部分不相干的主要方法是设置具有一定作用域的变量。在 Mathematica 中有两种限制变量作用域的基本方法:模块(Module)和块(Block)。我们在书写实际程序中,模块比块更具普遍性。然而在交互式计算中需要定义作用域时,块更实用。

Module[vars,body]所要做的是把执行模块时表达式 body 的形式看成 Mathematica 程序的“代码”。然而当“代码”中直接出现变量 vats 时,这些 vars 都将被看作局部的。Block[vats,body]并不查看表达式 body 的形式,而在整个计算 Body 的过程中,实用 vars 的局部值。

下例中我们根据 i 定义 m:

In[12]:=m=i^2 ,

Out[12]:=i²

在计算i+m的整个过程中使用块中i的局部值:


h[13]:=Block[{i; a}, !+m]

Out[13]=a+a2

而对于下面的例子,只有直接出现在i+m中的i,才被看作局部变量:

In[14]:=Module[{i=a}, i+m]

Out[14]=a+i



8.2.1 条件结构

我们在用计算机语言进行编程时,常用到条件语句。在 Mathematica 中也提供了多种设置条件的方法,并规定只有在该条件满足时才计算表达式。

下面条件结构的常用形式。

lhs:=rhs/:test	当 test 为真时使用定义
If[test,then,else]	如 test 为真计算 then, 反之计算 else

<code>which[test1,value1, test2, ...]</code>	依次计算 test1, 给出对应的第一个为真的值
<code>Switch[expr,form1,value1,form2,...]</code>	expr 与每一个 formi 相比较, 给出第一个相匹配的值
<code>Switch[expr,form1, value1,form2,..., _, def]</code>	用 def 为系统默认值

1.If 命令

下面的 test 为真,故返回第一表达式的值:

```
In[1]:=If [1>0,1+2,2+3]
```

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

用 Mathematica 编程时,不可避免的要在单个或多个定义之间进行选择。单个定义的右边包含多个由 If 函数控制的分支,多个定义是用 `/;condition` 来表示的。运用多个定义进行编程你常能得到结构很好的程序。下面定义了一个阶跃函数,即当 $x>0$ 时值为 1,反之值为 -1:

```
In[2]:=If[x>0,1,-1]
```

下面运用 `/; condition` 形式分别定义阶跃函数的正数和负数部分:

```
In[3]:g=1/:x>0
```

```
In[4]:g=-1/:x<0
```

用“?”显示用 `/;condition` 定义的函数 g 的完整信息:

```
In[5]:=?g
```

```
Global`g
```

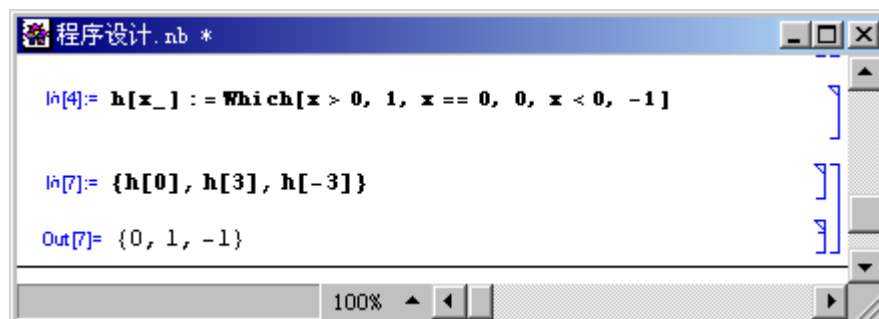
```
g[x_]:=1/:x>0
```

```
g[x_]:=-1/:x<0
```

我们用函数 If 时,还可以用 `if(test,expr)` 结构,即当 test 真时,计算表达式 expr,表达式 expr 的值就是整个 If 结构的值,反之返回空值。

2. Which 命令

对于一般情况函数 If 提供一个两者择一的方法。然而,有时条件多于两个,在这种情况下可用 If 函数的嵌套方式来处理,但在这种情况下使用 Which 或 Switch 函数将更合适。下面用 Which 定义具有三个条件的函数,调用这个函数:



```

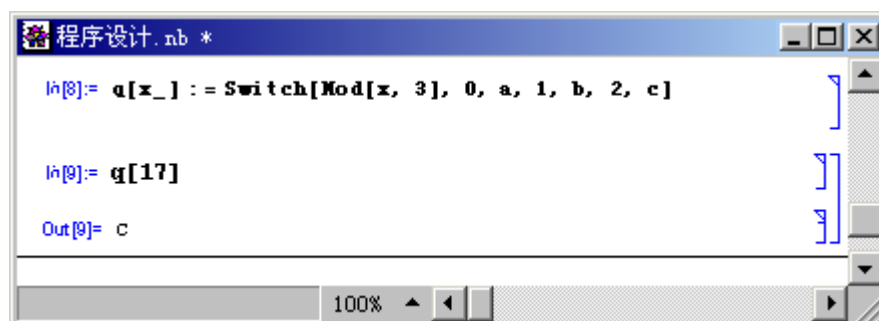
In[4]:= h[x_] := Which[x > 0, 1, x == 0, 0, x < 0, -1]

In[7]:= {h[0], h[3], h[-3]}

Out[7]= {0, 1, -1}

```

用 Switch 定义一个与模的余数有关的函数:



```

In[8]:= q[x_] := Switch[Mod[x, 3], 0, a, 1, b, 2, c]

In[9]:= q[17]

Out[9]= c

```

Mod[17,3]=2,因此运用了 Switch 中的第三种情况:

3 符号条件

在 Mathematica 中,有一种可能的情况就是你给出的条件结果既不是真也不为假。下面测试的结果既不是真也不是假,因此 If 的两个分支保持不变:

```

In[1]:= If[x==y,a,b]

Out[1]:= If[x==y,a,b]

```

你可以给 If 加上第三个条件结果,这允许你测试的结果既不是真也不是假的情况下使用它:

```

In[2]:= If[x==y,a,b,c]

Out[2]=c

```

下面给出处理符号条件的函数。因 $x=x$,故 Mathematica 给出结果为真:

但 Mathematica 在下面情况下以符号等式输出:

```
In[4]:=x==y
Out[4]:=x==y
```

除非表达式能得出真,否则都被假设为假:

```
In[5]:=TrueQ[x==x]
Out[5]=True

In[6]:=TrueQ[x==y]
Out[6]=false
```

我们用“==”可直接测试两个表达式的等同性:

```
In[7]:=x===y
Out[7]=False
```

一般情况下,“==”返回值为真(True)或假(False),而“=”为符号形式输出,表示一个符号等式。在特殊情况下可用“==”测试一个表达式的结构,而用“=”测试数学上的等同性。下例用“==”来测试表达式的结构:

下面给出一个无用结果:

在建立条件时,你常需要运用组合条件,如 test1&&test2&&…。对于这些组合条件,如果其中有一个为假,则最后结果为假。Mathematica 依次对 test 进行计算,直到其中有一个为假为止。

4.是逻辑表达式的运算形式。

逻辑表达式

expr1&&expr2&&expr3	计算 expr1,直到其中有一个为假为止
expr1 expr2 expr3	计算 expr1,直到其中有一个为真为止

下面的函数包括两个组合条件:

```
In[10]:=t[x_]:= (x!=0&&1/x<3)
```



对这两个测试条件进行计算,下面的第一次测试得出为假,因此不进行第二个条件的测试,第二测试结果可能为 1 或 0,因此

输出结果为假:

```
In[12]:=t[0]

Out[12]=False
```

Mathematica 处理逻辑表达式的方法允许你组合一系列的测试条件,且只有当前面条件满足时才处理后面的条件。



8.3 循环结构

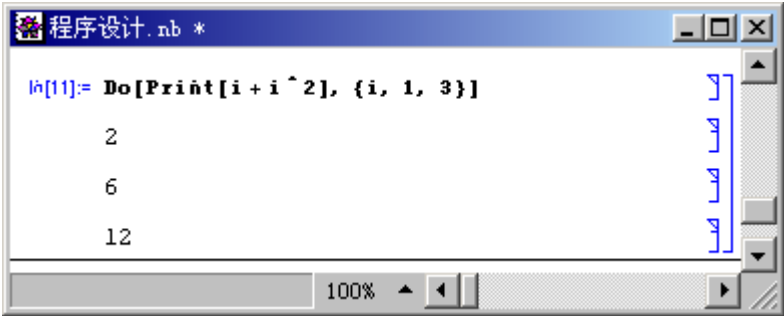
Mathematica 程序的执行包括对一系列 Mathematica 表达式的计算。对简单程序,表达式的计算可用分号“;”来隔开,然后一个接一个地进行计算。然而,有时你需要对同一表达式进行多次计算,即循环计算。

1 Do 循环结构

简单地 Do 循环结构形式:

Do[expr,{i,imax}]	循环计算 expr,以步长 1,i 从 1 增加到 imax
Do[expr,{i,imin,imax,di}]	循环计算 expr,以步长 di,i 从 imin 增加到 imax
Do[expr,{n}]	循环计算 expr n 次

计算 Print[i+i^2],i 从 1 增加到 3:

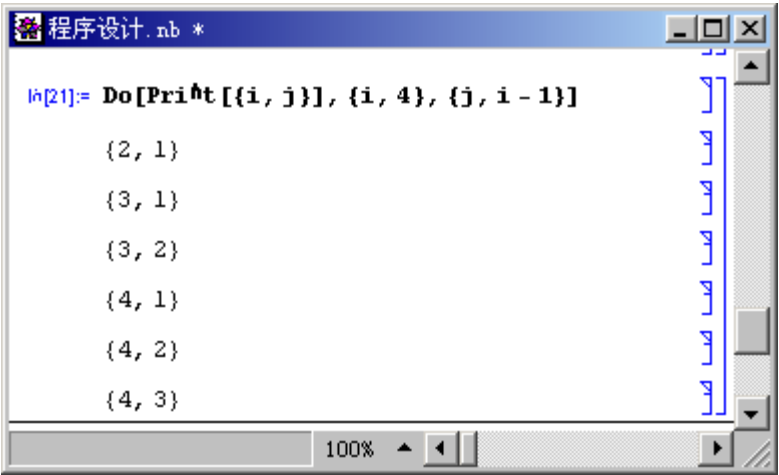


```
程序设计.nb *

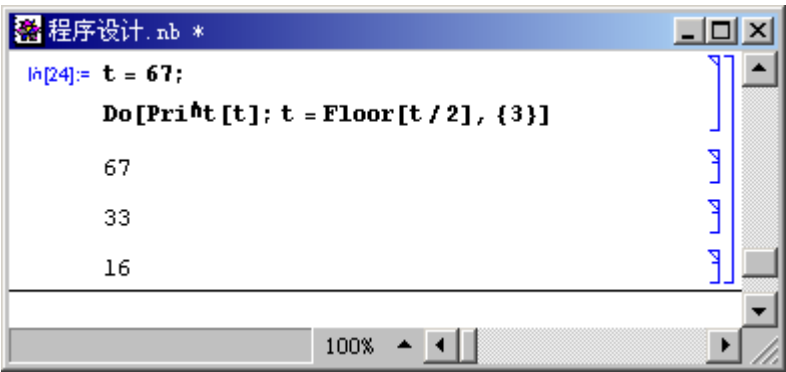
In[11]:= Do[Print[i + i ^ 2], {i, 1, 3}]

2
6
12
```

Do 中的定义的循环方式与函数 Table 和 Sum 中的定义一样。在函数 Do 中，你同样能建立重循环。
下面给出的 i 从 1 到 4 进行循环，而对于每个 i, j 又从 1 到 i-1 进行循环：



我们还可把一个过程放入 Do 函数中：



2 While 与 For 结构

在 Mathematica 程序中，Do 是以结构方式进行循环的，然而有时你需要生成非结构循环。此时，运用函数 While 和 For 是合适的。下面是 While 和 For 函数的循环结构形式：

While[test,body]	只要 test 为真，就重复计算 body
For[start,test,incr,body]	以为 start 起始值，重复计算 body 和 incr，直到 test 为假为止

当条件满足时，While 循环一直进行,因此为了防止死循环，在 While 中应包括命令能改变 test 的值。

```

程序设计.nb *
In[29]:= h = 25;
While[{h = Floor[h / 3]} != 0, Print[h]]

8
2

```

下面给出 For 循环的例子, $i++$ 表示 i 的值加 1(在本节的最后我们给出在编程时常会 用到的赋值方法):

```

程序设计.nb *
In[31]:=
For[i = 1, i < 4, i++, Print[i]]

1
2
3

```

下面再给出一个较复杂的 For 循环的例子，一旦 $i^2 < 10$ 不成立，就中止循环:

```

程序设计.nb *
In[32]:= For[i = 1; t = x, i ^ 2 < 10, i ++, t = t ^ 2 + i;
Print[t]]

1 + x2
2 + (1 + x2)2
3 + (2 + (1 + x2)2)2

```

Mathematica 中的函数 While 和 For 循环总是在执行循环体前对循环条件进行测试。一旦测试结果为假。就中止 While 和 For 循环。因此，循环体的计算总是在测试结果为真 的情况下进行的

3.一些特殊的赋值方式

一些赋值方式在循环结构中有时能带来一些方便。

$i++$	变量 i 加 1
$i--$	变量 i 减 1

<code>++i,</code>	变量 i 先加 1
<code>--i</code>	变量 i 先减 1
<code>i+=di</code>	i 加 di
<code>i-=di</code>	i 减 di
<code>x*=C</code>	x 乘以 C
<code>x /=c</code>	x 除以 c
<code>{x,y}={y,x}</code>	交换 x 和 y 值

4 重复运用函数

我们除了可用 Do、While、For 等进行循环计算外，我们还可以运用函数进行编程。运用函数编程结构你能得出非常有效的程序。例如 Nest[f,x,n]允许你对某一表达式重复运用函数 f

给出重复运用函数的方式。

<code>Nest[expr,n]</code>	对表达式 expr 重复调用函数 fn 次
<code>FixedPoint[y, expr]</code>	对表达式 expr 重复调用函数 fn 次，直到结果不变为止
<code>NestWhile[f,expr,test]</code>	对表达式 expr 重复调用函数 f，直到产生的结果为假时为止

下面对函数 f 迭代 5 次：

```

程序设计.nb *
In[33]:= Nest[f, x, 5]
Out[33]:= f[f[f[f[f[x]]]]]

```

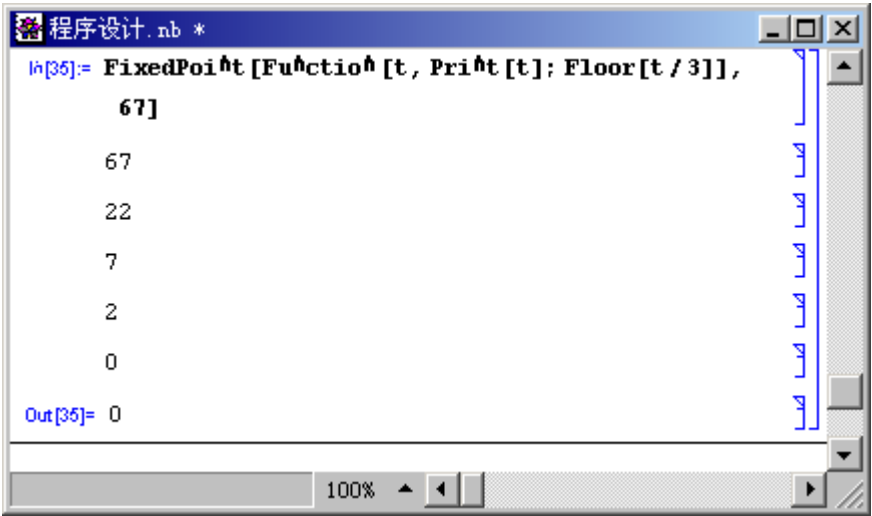
对纯函数进行迭代，你能得出与运用 Do 函数得出的结果一样：

```

程序设计.nb *
In[34]:= Nest[Function[t, 1/Sqrt[1+t]], x, 2]
Out[34]= 1 / Sqrt[1 + 1 / Sqrt[1+x]]

```

Nest 函数允许你重复运用某函数。然而,有时你想在结果不再发生变化的情况下就中止对函数的使用,此时立刻使用函数 FixPoint[f,x]。FixPoint 函数重复运用某一函数直到结果不再发生变化:



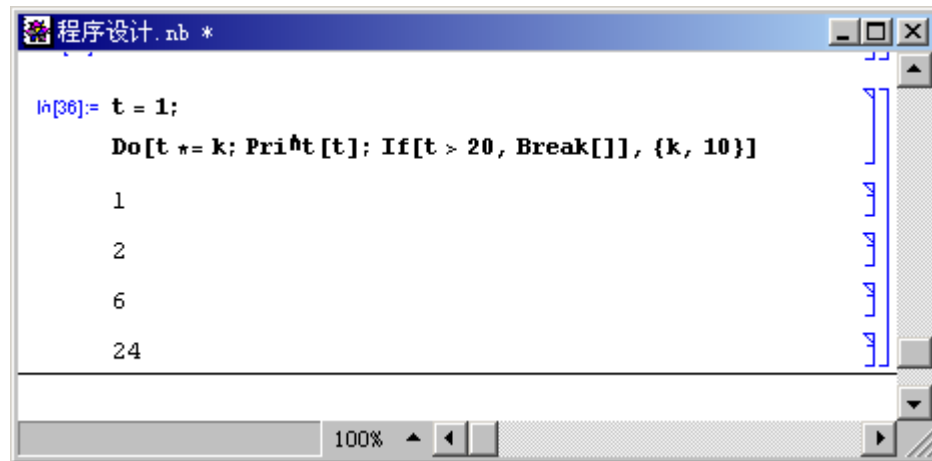
8. 3.5 流程控制

函数程序结构的流程控制一般来说比较简单,但是在应用 While 或 For 等循环时就比较复杂了,这是因为他们的流程控制依赖于表达式的值。而且在这样的循环中,流程的控制并不依赖于循环体中表达式的值。有时你在编制 Mathematica 程序时,在该程序中,流程控制受某一过程或循环体执行结果的影响。这时,我们可用 Mathematica 提供的流程控制函数来控制流程。这些函数的工作过程与 C 语言中的很相似。

常用的流程控制函数。

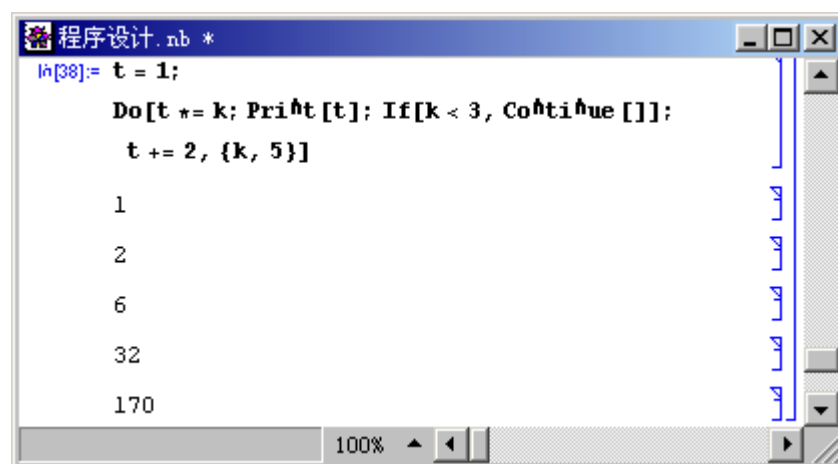
Break[]	退出本层的循环
Continue[]	转入当前循环的下一步
Return[expr]	退出函数中的所有过程及循环,并返回 expr 值
Goto[name]	转入当前过程中的元素 Label[name]
Throw[value]	返回 expr 值

当 t>20 时, Break[]就引起循环体的中断:



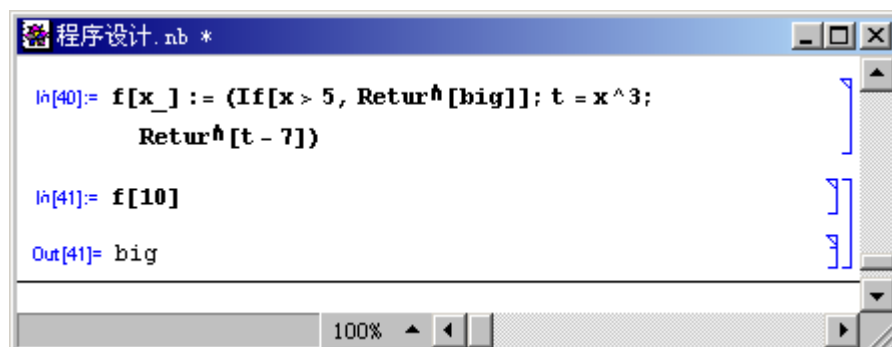
```
in[38]:= t = 1;  
Do[t *= k; Print[t]; If[t > 20, Break[]], {k, 10}]  
  
1  
2  
6  
24
```

当 $k < 3$ 时, Continue[] 继续执行循环:



```
in[38]:= t = 1;  
Do[t *= k; Print[t]; If[k < 3, Continue[]];  
t += 2, {k, 5}]  
  
1  
2  
6  
32  
170
```

下面给出 Return 的一个例子:




```
in[40]:= f[x_] := (If[x > 5, Return[big]]; t = x^3;  
Return[t - 7])  
  
in[41]:= f[10]  
  
Out[41]= big
```

Return[]允许你退出一函数，并返回一个值。Mathematica 可以进行局部返回，这可 允许你退出一列迭代函数。非局部返回在错误处理时是很有用的。下面给出的例子中如函数变量小于 0 则输出 error~

```
程序设计.nb *
In[42]:= h[x_] := If[x < 0, Throw[error], Sqrt[x]]
In[43]:= Catch[h[6]]
Out[43]=  $\sqrt{6}$ 
In[44]:= Catch[h[-6]]
Out[44]= error
```

In[6]不产生 error，且出示 Catch 的结果无效： 当变量小于 0 时输出 error

[运算符及特殊符号](#)[系统常量](#)



1.运算符及特殊符号

Line1	执行 Line，不显示结果
Line1,line2	顺次执行 Line1， Line2，并显示结果
?name	关于系统变量 name 的信息
??name	关于系统变量 name 的全部信息
!command	执行 Dos 命令
N!	N 的阶乘
!!filename	显示文件内容
<<filename	读入文件并执行
Expr: >>filename	打开文件写
Expr>>>filename	打开文件从文件末写
()	结合率
[]	函数
{}	一个表
< *MathFun*>	在 c 语言中使用 math 的函数

(*Note*)	程序的注释
#n	第 n 个参数
##	所有参数
Rule&	把 role 作用于后面的式子
%	前一次的输出
%%	倒数第二次的输出
Var::note	变量 var 的注释
“Astring”	字符串
Context	上下文
A+b	加
a-b	减
A*b 或 ab	乘
A/b	除

2.系统常量

Pi	3.1415 的无限精度数值
E	2.17828 的无限精度数值
Catalan	0. 915966Catalan 常数
EulerGamma	0. 5772Euler 常数
Khinchin	2. 68545Khinchin
Glaisher	0. 915966Glaisher
GoldenRatio	1. 61803 黄金分割数
Degree	Pi/180 角度弧度换算
I	复数单位
Infinity	无穷大
-Infinity	负无穷大
ComplexInfinity	复无穷大
Indeterminate	不定式

代数计算



Expand[expr]	展开表达式
Factor[expr]	展开表达式
Simplify[expr]	化简表达式

FullSimplify[expr]	将特殊函数也进行化简
PowerExpand[expr]	展开所有的幂次形式
ComplexExpand[expr,{x1, x2...}]	按复数实部虚部展开
FunctionExpand[expr]	化简表达式中的特殊函数
Collect[expr,x]	合并同次项
Collect[expr,{x1, x2, ...}]	合并 x1, x2,...的同次项
Together[expr]	通分
Apart[expr]	部分分式展开
Apart[expr,var)	对 var 的部分分式展开
Cancel[expr]	约分
xpandAll[expr]	展开表达式
ExpandAll[expr,patt]	展开表达式
FactorTermspoly]	提出共有的数字因子
FactorTerms[poly, x]	提出与 x 无关的数字因子
FactorTerms[poly,(x1,x2...)]	提出与 xi 无关的数字因子
Coefficient[expr,form]	多项式 expr 中 form 的系数
Coefficient[expr,form, n)	多项式 expr 中 form^n 的系数
Exponent[expr,form]	表达式 expr 中 form 的最高指数
Numerator[expr]	表达式 expr 的分子
Denominator[expr]	表达式 expr 的分母
ExpandNumerator[expr]	展开 expr 的分子部分
Solve[eqns,vats]	从方程组 eqns 中解出 Vats
Solve[eqns,vats,elims]	从方程组 eqns 中削去变量 elims, 解出 vats
DSolve[eqn,y,x]	解微分方程, 其中、y 是 x 的函数
DSolve[{eqn1,eqn2,...}, {y1, y2...},]	解微分方程组, 其中 yi 是 x 的函数
DSolve[eqn,y,{x1,x2...}]	解偏微分方程
Eliminate[eqns, Vats]	把方程组 eqns 中变量 vars 约去
SolveAlways[eqns, vars]	给出等式成立的所有参数满足的条件
Reduce[eqns, Vats]	化简并给出所有可能解的条件
LogicalExpand[expr]	用&&和, , 将逻辑表达式展开
InverseFunctionI 刀	求函数 f 的反函数
Root[f,k1	求多项式函数的第 k 个根
Roots[lhs==rhs, var]	得到多项式方程的所有根

微积分



D[f,x]	求 f[x] 的微分
D[f,{x, n}]	求 f[x] 的 n 阶微分
D[f,x1,x2...]	求 f[x]x1,x2... 偏微分
Dt[f,x]	求 f[x] 的全微分 df/dx
Dt(f)	求 f[x] 的全微分 df
Dt[f,{x,n}]	n 阶全微分 df^n/dx^n
Dt[f,x1,x2..]	对 x1,x2.. 的偏微分
Integrate[f,x]	f[x] 对 x 在的不定积分
Integrate[f,{x, xmin, xmax}]	f[x] 对 x 在区间(xmin,xmax)的定积分
Integrate[f,{x, xmin, xmax},{y,ymin,ymax}]	f[x,y] 的二重积分
Limit[expr,x->x0]	x 趋近于 x0 时 expr 的极限
Residue[expr,{x,x0}]	expr 在 x0 处的留数
Series[f,{x,x0,n}]	给出 f[x] 在 x0 处的幂级数展开
Series[f,{x, x0,nx}, {y, y0, ny}]	先对 y 幂级数展开, 再对 x 幂级数展开
Normal[expr]	化简并给出最常见的表达式
SeriesCoefficient[series, nJ]	给出级数中第 n 次项的系数
SeriesCoefficient[series,{n1,n2...}]	一阶导数
InverseSeries[s,x]	给出逆函数的级数
ComposeSeries[serie1,serie2...]	给出两个基数的组合
SeriesData[x,x0,{a0,a1,...},nmin,nmax,den]	表示一个 x0 处 x 的幂级数
O[x]^n	n 阶小量 x^n

多项式函数



Variables[poly]	给出多项式 poly 中独立变量的列表
CoefficientList[poly,var]	给出多项式 poly 中变量 var 的系数
CoefficientList[poly,{var1,var2...})	给出多项式 poly 中变量 var(i) 的系数列
PolynomialMod[poly,m]	poly 中各系数 mod m 同余后得到的多项式, m 可为整式
PolynomialQuotient[p,q,x]	以 x 为自变量的两个多项式之商式 p/
PolynomialRemainder[p,q,x]	以 x 为自变量的两个多项式之余式
PolynomialGCD[poly1,poly2,...]	poly(i) 的最大公因式

PolynomialLCM[poly1,poly2, . . .]	poly(i)的最小公倍式
PolynomialReduce[poly, {poly1,Poly2,...}, {x1,x2...}]	得到一个表 I(a1, a2, ·), b)其中 Sumld*polyi]+b=poly
Resultant[poly1,poly2,var]	约去 poly1,poly2 中的 var
Factor[poly]	因式分解(在整式范围内)
FactorTerms[poly]	提出 poly 中的数字公因子
FactorTermslpoly,{x1, x2...})	提出 poly 中与 xi 无关项的数字公因子
FactorList[poly], FactorSquareFreeList[p01y], FactorTermsList[poly,{x1, x2...}]	给出各个因式列表
Cyclotomic[n,x]	n 阶柱函数
Decomposet[poly,x]	迭代分解, 给出 {p1,p2,...}, 其中 P1(p2(...))=poly
InterpolafinSPolynomial[data, Var]	在数据 data 上的插值多项式
RootSum[f,form]	得到 f[x]=0 的所有根, 并求得 Sum[form[xi]]

随机函数

RandomCtype,range]	产生 type 类型且在 range 范围内的均匀分布随机数
Random[]	0-1 上的随机实数
SeedRandom[n1	以 n 为 seed 产生伪随机数
Randomldistribution]	可以产生各种分布

数值函数

N[expr]	表达式的机器精度近似值
N[expr,n)	表达式的 n 位近似值, n 为任意正整数
NSolve[lhs=---rhs, val]	求方程数值解
NSolveleqn, Var,n1	求方程数值解, 结果精度到 n 位
NDSolve[eqns, y, {x, xmir1, xmax}]	微分方程数值解
NDSolve[eqns, {y1, y2, ·. 1, {x, xmin, xmax}}	微分方程组数值解
FindRoot[lhs==rhs, {x,x0)1	以 x0 为初值, 寻找方程数值解
FindRoot[lhs=--rhs, {x, xstart, xmin, xmax}]	以 xstart 为初值, 在[xmin,xmax]范围内寻找方程数值解

NSum[f,{imin,imax, di}]	数值求和，出为步长
NSum[f,{imin,imax, di},{j,..},..]	多维函数求和
NProduct[f,{i,imin,imax,di}]	函数求积
NIntegrate[f, {x,xmin,xmax}]	函数数值积分
FindMinimum[f, {x,x0}]	以 x0 为初值，寻找函数最小值
FindMinimum[f, {x, xstart, xmin, xmax}]	以 xstart 为初值，在[xmin,xmax]范围内寻找方程解
ConstrainedMin[f,{inequ}, {x, y, ..}]	inequ 为线性不等式组，f 为 x, y, . 之线性函数，得到最小值及此时的 x, y, . 取值
ConstrainedMax[f, {inequ), {x, y, ..}]	同上
LinearProgramming[C, m, b]	解线性组合 c.x 在 m. $x \geq b$ 且 $x \geq 0$ 约束下的最小值，x, 为向量，m 为矩阵
LatticeReduce[{v1,v2...}]	向量组 Vi 的极小无关组
Fit[data,funs,vats]	用指定函数组对数据进行最小二乘拟合
Interpolation[data]	对数据进行插值
Lisinterpolation[array]	对离散数据插值，array 可为 n 维
ListInterpolation[array,{ {xmin,xmax}, {min,ymax},...}]	在特定网格上进行插值
FunctionInterpolation[expr,{x,xmin,xmax}, {y,ymin,ymax},...]	以对应 expr[xi,yi]的数值为数据进行插值
Fourier[list]	对复数数据进行傅氏变换
InverseFourierof(St)	对复数数据进行傅氏逆变换

[制表函数](#)
[元素操作](#)
[表的操作](#)


1.制表函数

{e1,e2,...}	一个表，元素可以为任意表达式，无穷嵌套
Table[expr,{imax}]	生成一个表，共 imax 个元素
Table[expr,{i,imax}]	生成一个表，共 imax 个元素 expr 间
Table(expr,{i, imin,imax}, {j,jmin,jmax},...]	多维表
Range[imax]	简单数表 f1, 2+, imax)
Range[imin, imax, di]	以 di 为步长的数表
Array[f,n]	一维表，元素为 f[i](i 从 1 到 n)

Array[f,{n1,n2...}]	多维表，元素为 f ， i 小，1 (各自从 1 到 n_i)
IdentityMatrix[n]	n 阶单位阵
DiagonalMatrix[list]	对角阵

2、元素操作

Part[expr,i]或 expr[[i]]	第 i 个元素
expr[[-i]]	倒数第 i 个元素
expr[{i,j,...}]	多维表的元素
expr[{i1,i2,...}]	返回由第 $i(n)$ 的元素组成的子表
FirstCexpr]	第一个元素
Last[expr]	最后一个元素
Head[expr]	函数头，等于 $\text{expr}[[0]]$
Extract[expr,list]	取出由表 list 指定位置上 expr 的元素值
Take[list,n]	取出表 list 前 n 个元素组成的表
Take[list, {m,n}]	取出表 list 从 m 到 n 的元素组成的表
Drop[list,n]	去掉表 list 前 n 个元素组下的表
Rest[expr]	去掉表 list 第一个元素剩下的表
Select[USt, crit]	把 crit 作用到每一个 list 的元素上，为 True 的所有元素组成的表
Length[expr]	expr 第一层元素的个数
Dimensions[expr]	表的维数返回 $\{n1,n2...\}$, expr 为一个 $n1*n2...$ 的阵
TensorRank[expr]	秩
Depth[expr]	expr 最大深度
Level[expr,n]	给出 expr 中第 n 层子表达式的列表
Count[USt, paUem]	满足模式的 list 中元素的个数
MemberQ[list, form]	list 中是否有匹配 form 的元素
FreeQ[expr,form]	MemberQ 的反函数
FreeQ[expr,form]	表中匹配模式 pattern 的元素的位置列表
Cases[{e1,e2...}, pattem]	匹配模式 pattem 的所有元素 e_i 的表

3.表的操作

Append[exp[elem]	返回在表 expr 的最后追加 elem 元素后的表
Prepend[expr,elem]	返回在表 expr 的最前添加 elem 元素后的表
Insert[list, elem, n]	在第 n 元素前插入 elem
Insert[expr,elem,{i,j,...}]	在元素 $\text{expr}[[i,j,...]]$ 前插入 elem

Delete[expr,{i,j,...}]	删除元素 expr[[{i,j,...}]]后剩下的表
DeleteCases[expr,pattern]	删除匹配 pattern 的所有元素后剩下的表
ReplacePart[expr,new,n]	将 expr 的第 n 元素替换为 new
Sort[list]	返回 list 按顺序排列的表
Reverse[expr]	把表 expr 倒过来
RotateLeft[expr,n]	把表 expr 循环左移 n 次
RotateRight[expr,n]	把表 expr 循环右移 n 次
Partition[list,n]	把 list 按每 n 个元素为一个子表分割后再组成的大表
Flatten[list]	抹平所有子表后得到的一维大表
Flatten[list,n]	抹平到第 n 层
Split[list]	把相同的元素组成一个子表，再合成的大表

[二维绘图](#) [二维绘图设置](#)

[三维绘图](#) [三维绘图设置](#) [密度图](#) [图形显示](#) [图元函数](#) [着色及其他](#)

1.二维绘图

Plot[f,{x,xmin,xmax}]	一维函数 f[x]在区间[xmin,xmax]上的函数曲线
Plot[{f1,f2..},{x,xmin,xmax}]	在同一图形上画几条曲线
ListPlot[{y1,y2,...}]	绘出由离散点对(n, yn)组成的图
ListPlot[{x1,y1},{x2,y2},...]	绘出由离散点对(xrl,yrl)组成的图
ParametricPlot[{fx,fy},{t,tmin,tmax}]	由参数方程在参数变化范围内产生的曲线
ParametricPlot[{fx,fy},{gx,gy},...], {t,tmin,truax}]	

2.二维设置

PlotRange->{0,1}	作图显示的值域范围
AspectRatio->1/GoldenRatio	生成图形的纵横比
PlotLabel->label	标题文字
Axes->{false,True}	分别制定是否画 x,y 轴
AxesLabel->{xlabel,ylabel}	x,y 轴上的说明文字
Ticks->None,Automatic,fun	用什么方式画轴的刻度
AxesOrigin->{x,y}	坐标轴原点位置
AxesStyle->{{xstyle},{ystyle}}	设置轴线的线性颜色等属性
Frame->True,False	是否画边框
FrameLabel->{xlabel,ylabel,xlabel,ylabel}	边框四边上的文字

FrameTicks 同 Ticks	边框上是否画刻度
GridLines 同 Ticks	图上是否画栅格线
Framestyle->{{xmstyle}, {ymstyle}}	设置边框线的线性颜色等属性
ListPlot[data,PlotJoined->True]	把离散点按顺序连线
Plotsyle->{{style1}, {style2},...}	曲线的线性颜色等属性
PlotPoints->15	曲线取样点，越大越细致

3.三维绘图

Plot3D[f,{x,xmin,xmax}, {y,ymin,ymax}]	二维函数 $f(x,y)$ 的空间曲面
Plot3D[{f, s}, {x,xmin,xmax}, {y,ymin,ymax}]	同上，曲面的染色由 $s(x,y)$ 值决定
ListPlot3D[array]	二维数据阵 array 的立体高度图
ListPlot3D[array,shades]	同上，曲面的染色由 shades[数据]值决定
ParametricPlot3D[{fx,fy,fz},{t,tmin,tmax}]	三维参数图形
ContourPlot[f,{x,xmin,xmax}, {y,ymin,ymax}]	二维函数 $f(x,y)$ 在指定区间上的等高线图
ListContourPlot[array]	二维函数 $f(x,y)$ 在指定区间上的等高线图

4.三维设置

Contours->n	画 n 条等高线
Contours->{z1,z2, ...}	在 z_i 处画等高线
ContourShading->False	是否用深浅染色
ContourLines->True	是否画等高线
ContourStyle->{{style1},{style2},...}	等高线线性颜色等属性

5.密度图

DensityPlot[f,{x,xmin,xmax},{y,ymin,ymax}]	二维函数 $f(x,y)$ 在指定区间上的密度图
ListDensityPlot[array]	二维函数 $f(x,y)$ 在指定区间上的密度图

5.图形显示

Show[graphics,options]	显示一组图形对象，options 为选项设置
Show[g1, g2...]	在一个图上叠加显示一组图形对象
GraphicsArray[{g1, g2,...}]	在一个图上分块显示一组图形对象
SelectionAnimate[notebook,t]	把选中的 notebook 中的图画循环放映

6.图元函数

Graphics[prim, options]	prim 为下面各种函数组成的表, 表示一个二维图形对象
Graphics3D[prim, options]	prim 为下面各种函数组成的表, 表示一个三维图形对象
SurfaceGraphics[array, shades]	表示一个由 array 和 shade 决定的曲面对象
ContourGraphics[array]	表示一个由 array 决定的等高线图对象
DensityGraphics[array]	表示一个由 array 决定的密度图对象
Point[p]	p={x, y}或{x, y, 2}, 在指定位置画点
Line[{p1, p2, ...}]	经由 Pi 点连线
Rectangle[{xmin, ymin}, {xmax, ymax}]	画矩形
Cuboid[{xmin,ymin, zmin}, {xmax,ymax,zmax}]	由对角线指定的长方体
Polygon[{p1, p2, ...}]	封闭多边形
Circle[{x,y}, r]	画圆
Circle[{x,y},{rx,ry}]	画椭圆, rx,ry 为半长短轴
Circle[{x, y},r,{a1,a2}]	从角度 a1-a2 的圆弧
Disk[{x, y},r]	填充的圆、椭圆、圆弧等参数同上
Raster[array,ColorFunction->f]	颜色栅格
Text[expr,coords]	在坐标 coords 上输出表达式
PostScript["string"]	直接用 Postscript 图元语言写
Scaled[{x, y,...}]	返回点的坐标, 且均大于 0 小于 1

7.着色及其他

GrayLevel[level]	灰度 level 为 0~1 间的实数
RGBColor[red, green, blue]	RGB 颜色, 均为 0~1 间的实数
Hue[h, s, b]	亮度, 饱和度等, 均为 0~1 间的实数
CMYKColor[cyan, magenta,yellow,black]	CMYK 颜色
Thickness[r]	设置线宽为 r
PointSize[d]	设置绘点的大小
Dashing[{r1,r2,...}]	画一个单元的间隔长度的虚线
ImageSize->{x, y}	显示图形大小(单位为像素)

流程控制



If[condition,t,f]	如果 condition 为 True,执行 t, 否则执行 f 段
if[condition, t, f, u]	如果 condition 为 True, 执行 t, 为 False 执行 f, 既非 True 又非 False, 则执行 u 段
Which[test1,block1,test2,block2..]	执行第一为 True 的 test _i 对应的 block _i
Switch[expr,form1,block1,form2,block2...]	重复执行 expr imax 次
Do[expr,{imax}]	重复执行 expr imax 次
Do[expr,{i,imin,imax}, {j,jmin,jmax}]	多重循环
While[test, body]	循环执行 body 直到 test 为 False
For[start,test,incr,body]	循环执行 body 直到 test 为 False
Throw[value]	停止计算, 把 value 返回给最近一个 Catch 处理
Throw[value, tag]	停止计算, 把 value 返回给最近一个 Catch 处理
Catch[expr1	计算 expr,遇到 Throw 返回的值则停止
Catch[expr, form]	当 Throw[value, tag]中 Tag 匹配 form 时停止
Return[expr)	从函数返回, 返回值为 expr
Return[]	返回值 Null
Break1[]	结束最近的一重循环
Continue1[]	停止本次循环, 进行下一次循环
Goto[tag]	无条件转向 Label[Tag]处
Label[tag]	设置一个断点
Check[expr,fmlexpr]	计算 expr,如果有出错信息产生, 则返回 failexpr 的值
Check[expr,failexpr,s1::t1, s2::t2,...]	当特定信息产生时则返回 failexpr
CheckAbort[expr,failexpr]	当产生 abort 信息时返回 failexpr
Interrupt[]	中断运行
Abort[]	中断运行
TimeConstrained[expr,t]	计算 expr, 当耗时超过 t 秒时终止

MemoryConstrained[expr,b]	计算 expr，当耗用内存超过 b 字节时终止运算
Print[expr1,expr2,...]	顺次输出 expri 的值
Input[]	产生一个输入对话框，返回所输入的任何表达式
Input["prompt"]	同上，prompt 为对话框的提示
Pause[n]	运行暂停 n 秒