Mathematica表达式及其运算规则

在本节中，我们将主要介绍Mathematica进行数学运算的基本工作原理及特殊符号的输入方式。

1、 西腊字母及命令的直观输入

在Notebook中，有两种输入西腊字母的方法，一种是调用File→Palettes→BasicInput、BaiscTypesetting或CompleteCharacters→Letters→Greek菜单，此时会弹出一个含有西腊字母的数学工具面板，单击此面板的符号即可；另一种是直接通过键盘输入西腊字母所代表的标准名称，其格式为\[Greek_name]，例如，在Notebook中输入\[Beta]后(注意大小写)，将会显示β，下面是一些常用西腊字母的标准名称表。

α	\[Alpha]	β	\[Beta]	γ	\[Gamma]	δ	\[Dalta]
ε	\[Epsilon]	ζ	\[Zeta]	η	\[Eta]	θ	\[Theta]
λ	\[Lambda]	μ	\[Mu]	ξ	\[Xi]	π	\[Pi]
ρ	\[Rho]	σ	\[Sigma]	τ	\[Tau]	ϕ	\[Phi]
φ	\[CurlyPhi]	ω	\[Omega]	Φ	\[CapitalPhi]	Γ	\[CapitalGamma
Π	\[CapitalPi]	ψ	\[Psi]	Σ	\[CapitalSigma]	Ω	\[CapitalOmega]

另外，在刚开始使用**Mathematica**时，一般对有关数学运算命令及数学公式的输入都不是太熟悉，这时可以通过菜单**File→Palettes**的各个下级子菜单输入相关命令及公式，不过这种输入方法效率不高，建议还是少用为好。

2、 表达式与表结构

Mathematica能够处理多种类型的数据形式：数学公式、集合、图形等等，**Mathematica**将它们都称为表达式。使用函数及运算符(+, -, *, /,^等)可组成各种表达式。

```
FullForm[a*b+c]
Plus[Times[a, b], c]
FullForm[{1, 2, 3, 4}]
List[1, 2, 3, 4]
Head[Sin[x]]
Sin
```

FullForm[]可显示出表达式在系统内部存贮的标准格式，而**Head[]**可得到某个表达式的头部，这对我们确定表达式的类型很有用处。

上面的**{1,2,3,4}**称为表(**List**)，表是**Mathematica**中非常有用的结构。首先，表可以理解成数学意义下的集合，例如对集合**{1,{2,3},4,{5,6,7},8,9},**它是含有**6**个元素的子集合**,**其中**{2,3}**及**{5,6,7}**此集合的子集合。

上页 下页 退出

作为集合，有下面的各种集合运算。

•Append[list,element]在集合list的末尾加入元素element

•Apply[Plus,list]将集合list中的所有元素加在一起

•Apply[Times,list]将集合list中的所有元素乘在一起

•Complement[list1,list2]求在list1中而不在list2中元素集合

•Delete[list,{i,j}]删除集合第i,j处的元素

•Delete[list,i]删除集合list的第i个元素

•Flatten[list]展开集合list中的各个子集，形成一个一维表

- FlattenAt[list,n]展开集合list中的第n级子集
- Insert[list,element,{i,j}]插入第i个子集合的第j 个元素处
- Insert[list,element,i]在list第i个元素的前面插入element
- Intersection[list1,list2,…]这是数学意义下的求交集命令
- Join[list1,list2,…]将集合首尾相连，形成一个新的集合
- Length[list]集合list中元素的个数
- list[[i,j]]集合list中第i个子集合的第j个元素

•list[[i]]集合list中第i个元素

•Partition[list,n]将集合list分成n个元素一组

•Prepend[list,element]在集合list的开头加入元素element

•ReplacePart[list,element,{i,j}]替换list中的第i,j处的元素

•ReplacePart[list,element,i]替换集合list中的第i个元素

•Reverse[list]翻转集合list中的元素

•Sort[list]将集合list中的元素按升序排序

- **Table[f,{i,imin,imax},{j,jmin,jmax}]**建立二维表或矩阵
- **Table[f,{i,imin,imax}]**建立一个一维表或向量
- **Take[list,{m,n}]** 给出**list**中从**m**到**n**之间的所有元素
- **Take[list,n]** 给出前**n**个，**Take[list,-n]** 给出后**n**个
- **Union[list]**合并集合**list**中的重复元素
- **Union[list1,list2,…]**这是数学意义下的求集合的并集命令

下面是有关集合方面的一些运算:

```
s = Table[i^2 + 1, {i, 0, 7}]
```

{1, 2, 5, 10, 17, 26, 37, 50}

```
Print["length s=", Length[s], "   s[[4]]=", s[[4]]]
```

length s=8   s[[4]]=10

```
s1 = Partition[Sqrt[s], 2]
```

$\{\{1, \sqrt{2}\}, \{\sqrt{5}, \sqrt{10}\}, \{\sqrt{17}, \sqrt{26}\}, \{\sqrt{37}, 5\sqrt{2}\}\}$

```
s1[[3, 2]]
```

$\sqrt{26}$

```
s2 = Flatten[s1]
```

$\{1, \sqrt{2}, \sqrt{5}, \sqrt{10}, \sqrt{17}, \sqrt{26}, \sqrt{37}, 5\sqrt{2}\}$

```
s3 = Insert[s2, 17, 4]
```

$\{1, \sqrt{2}, \sqrt{5}, 17, \sqrt{10}, \sqrt{17}, \sqrt{26}, \sqrt{37}, 5\sqrt{2}\}$

```
Intersection[s, s3]
```

{1, 17}

其次，对于一维表，可以理解成数学意义下的向量，对于二维表，可以理解成矩阵，因此，有如下的矩阵函数，其中a,b为向量，p,q为常量，M为方阵，A,B为同阶普通矩阵,具体例子参见下一节。

- Dot[a,b]或a.b 向量a与b的数量积
- Cross[a,b] 向量a与b的矢量积
- P*A+q*B 矩阵与数的乘法运算
- A*B A与B的对应元素相乘
- M^2 将矩阵M中的每个元素平方
- P.Q 矩阵乘法运算,其中P为m×k阶矩阵，Q为k×n阶矩阵

上页 下页 退出

- Det[M] 求方阵M的行列式
- MatrixForm[A] 以矩阵的形式显示 A
- MatrixPower[M,n] 矩阵M的n次幂
- Transpose[A] 矩阵A的转置矩阵
- Eigenvalues[M] 求矩阵M的特征值
- Eigenvectors[M] 求矩阵M的特征向量
- Eigensystem[M] 求矩阵M的特征值与特征向量
- IdentityMatrix[n] 建立一个$n \times n$的单位阵
- DiagonalMatrix[list] 建立一个对角阵，其对角线元素为表list

- **Inverse[M]** 求方阵M的逆矩阵
- **LinearSolve[A,b]** 求线性方程组AX=b的解
- **NullSpace[A]** 求满足方程AX=0的基本向量组，即零解空间
- **RowReduce[A]** 将矩阵A进行行变换
- **QRDecomposition[M]** 矩阵M的QR分解
- **SchurDecomposition[M]** 矩阵M的Schur分解
- **JordanDecomposition[M]** 矩阵M的Jordan分解
- **LUDecomposition[M]** 矩阵M的LU分解

3、Mathematica中数的类型与精度

在Mathematica中，进行数学运算的“数”有四种类型，它们分别是Integer(整数)、Rational(有理数)、Real(实数)、Complex(复数)。不带有小数点的数，系统都认为是整数，而带有小数点的数，系统则认为是实数。对两个整数的比，如12/13，系统认为是有理数，而a+b*I形式的数，系统认为是复数。Mathematica可表示任意大的数和任意小的数，其它计算机语言比如C、Basic是做不到这一点的，例如

500! // N

$1.220136825991110\times10^{1134}$

A := Table[1/(i+j), {i, 1, 8}, {j, 1, 8}]; Det[A]

$$\frac{1}{470214262250820283325130473472000000}$$

```
(2+I)(1+2I)^2/(2-11I)
```

$$-\frac{3}{5}-\frac{4\,i}{5}$$

其中//N表示取表达式的数值解，默认精度为16位，它等价于N[expr],一般形式为N[expr,n]，即取表达式n位精度的数值解。如

```
N[Det[A], 30]
```

$$2.12669006510607072000158763622\times 10^{-37}$$

```
N[π, 50]
```

3.1415926535897932384626433832795028841971693993751

使用Rationalize[expr,error]命令可将表达式转换为有理数，其中error表示转换后误差的控制范围。例如

```
Rationalize[3.1415926, 10^-5]
```

$$\frac{355}{113}$$

上页 下页 退出

Mathematica中的变量以字母开头，变量中不能含有空格及下划线，因此，上面的**2I**表示**2*I(I**为虚数**)**，乘号可用空格代替，在很多情况下，乘号可以省略，如**(1+I)(1+2I)**中的两个乘号。如果某个表达式的结果为复数，**Mathematica**就会给出复数的结果。对下面的３次方程

```
Solve[x^3 - 2 x^2 + 3 x - 6 == 0, x]
```

$$\{\{x \to 2\}, \{x \to -i\sqrt{3}\}, \{x \to i\sqrt{3}\}\}$$

上面的行列式|**A**|的计算结果，系统给出的是一个分数值，在**Mathematica**中，不同类型的数进行运算，其结果是高一级的数，如有理数与实数运算的结果是实数，复数与实数的运算结果是复数，依此类推。由于整数与有理数的运算级别最低,

上页 下页 退出

因此，在进行数学计算中，如果可能的话，就尽量用精确数，即整数或有理数。另外，“==”称为逻辑等号，定义一个等式要用逻辑等号。

```
A := {{5, 1, 2}, {1, 2, 6}, {1, 2, 7}}; Inverse[A]
```

$$\left\{\left\{\frac{2}{9}, -\frac{1}{3}, \frac{2}{9}\right\}, \left\{-\frac{1}{9}, \frac{11}{3}, -\frac{28}{9}\right\}, \{0, -1, 1\}\right\}$$

```
B := {{5.0, 1, 2}, {1, 2, 6}, {1, 2, 7}}; Inverse[B]
```

{{0.222222, -0.333333, 0.222222},
{-0.111111, 3.66667, -3.11111}, {0., -1., 1.}}

其中Inverse[]是求逆矩阵命令。在Mathematica中，一行中可以输入多个命令,各命令间用分号分隔。另外，分号还有一个作用是通知Mathematica，只在内存中计算以分号结尾的命令，但不输出此命令的计算结果。

如果表达式太长，一行写不下，可以分２行写，系统会自动判断一个表达式是否输入完毕。对于需要多行输入的表达式，建议每行用运算符结尾。下面我们简要说明一下**Mathematica**的赋值符号及相关命令。在**Mathematica**中，对变量赋值，有两种方法。**A:=expr**的意思是将表达式**expr**的值赋给**A**，但**Mathematica**并不立即执行此项操作，一直到用到**A**的值时，**Mathematica**才真正的将**expr**的值赋给**A**，即所谓的延迟赋值。在大部分情况下，我们都采用延迟赋值的形式为表达式赋值。另一种赋值方法是我们所熟悉的赋值形式，即**A=expr**或**A=B= expr**的形式，一般称为立即赋值。只要一执行该命令，**Mathematica**将**expr**

的值赋给**A**。

另外，对于变量，**Mathematica**不像**C**语言那样，需要申请后再使用，也不用事先确定变量的类型，这些问题都由**Mathematica**来自动处理。对于不需要的变量，可以使用**Clear**命令将变量从内存中清除出去，以节省内存空间，例如

- **Clear[A]** 清除变量**A**，其简写形式是**A=.**

- **Clear[A,B,W]** 清除变量**A**、**B**、**W**

- **Clear["A*","B*"]** 清除以**A**、**B**开头的所有变量

可以使用**Precision[expr]**或**Accuracy[expr]**返回表达式的精度

下面的变量 a 是计算 $\int_1^2 e^{-x^2}\,dx$ 的数值积分，b 是计算其符号积分，c 和 d 只是输入的形式不同，但精度却不一样。

```
a := NIntegrate[Exp[-x^2], {x, 1, 2}]; Precision[a]
16
b := Integrate[Exp[-x^2], {x, 1, 2}]; Accuracy[b]
∞
c = 1.23; d = 123 / 100; Print["c=", Accuracy[c], "   d=", Accuracy[d]]
c=16   d=∞
```

其中，∞在系统中是一个内部常数，其完整的命令是 **Infinity**，这样的常数有：**Pi(π)**、E(实数e)、**ComplexInfinity(**复数的无穷大)、**I(**复数i)、**Degree(1。=π/180)**、C(不定积分的任意常数)，另外，D(导数运算符)，N(取精度运算符)、**O(**泰勒展开的高阶无穷小量)。

上面Print[]命令的功能是打印表达式或者字符串，其格式为

Print[expr1，expr2，……]

expr1,expr2,……可以为任意合法的Mathematica表达式，如果为字符串，则需要双引号将字符串括起来。在实际计算过程中，可能得到的结果中含有很小的数，为了以后计算上的方便，我们如果想去掉这样的数，可以使用命令

- **Chop[expr,dx]** 若expr中的某个数小于dx，则用0来代替该数
- **Chop[expr]**若expr中的数小于10-10，则用0来代替该数

下面是一个多项式曲线拟合问题的实际例子

```
data = {{-3, 9}, {-2, 4}, {-1, 1}, {0, 0}, {1, 1}, {2, 4}, {3, 9}};
f[x_] = Fit[data, {1, x, x^2, x^3}, x]
```

$$1.33227\times10^{-15} - 6.66134\times10^{-16}\,x + 1.\,x^2 + 1.38778\times10^{-16}\,x^3$$

```
Chop[f[x]]
```

$$1.\,x^2$$

可以用下面的几个函数来判断表达式运算结果的类型，其中True和False是系统内部的布尔常量。

•NumberQ[expr]判断表达式是否为一个数，返回True或False

•IntegerQ[expr]判断表达式是否为整数, 返回True或False

•EvenQ[expr]判断表达式是否为偶数，返回True或False

•OddQ[expr]判断表达式是否为奇数，返回True或False

•PrimeQ[expr]判断表达式是否为素数，返回True或False

•Head[expr]判断表达式的类型

```
Print[Head[0.5], "  ", Head[1/2], "  ", Head[{1, 2, 3}]]
Real  Rational  List
```

4、 常用数学函数

Mathematica的数学运算，主要是依靠其内部的大量数学函数完成的，下面我们依次列出常用的数学函数，其中**x**、**y**、**a**、**b**代表实数，**z**代表复数，**m**、**n**、**k**为整数。所有的函数或者是它的英文全名，或者是其它计算机语言约定俗成的名称，函数的参数表用方括号**[]**括起来，而不是用圆括号。另外，**Mathematica**对大小写敏感。

数值函数

- Round[x] 最接近x的整数
- Floor[x] 不大于x的最大整数
- Celing[x] 不小于x的最小整数
- Sign[x] 符号函数
- Abs[z] 若z为实数，则求绝对值，为复数，则取模
- Max[x1,x2,…]或Max[{x1,x2, …},…] 求最大值
- Min[x1,x2,…]或Min[{x1,x2, …},…] 求最小值
- x+Iy,Re[z],Im[z],Conjugate[z],Arg[z] 关于复数的基本运算

随机函数

- Random[] 返回一个区间[0,1]内的一个随机数
- Random [Real,{xmin,xmax}]返回一个区间[xmin,xmax]内的随机数
- Random[Integer] 以1/2的概率返回0或1
- Random[Integer,{imin,imax}]返回位于[imin,imax]间的一个整数
- Random[Complex] 模为1的随机复数
- Random[Complex,{zmin,zmax}] 复平面上的随机复数
- SeedRandom[] 使用系统时间作为随机种子
- SeedRandom[n] 使用整数n作为随机种子

整数函数及组合函数

- Mod[m,n],Quotient[m,n] m/n的余数及商
- GCD[n1,n2,…],LCM[n1,n2, …] 最大公约数及最小公倍数
- FactorInteger[n] 返回整数n的所有质数因子表
- PrimePi[x],Prime[k]返回小于x的质数个数及第k个质数
- n!, n!! 整数n的阶乘及双阶乘
- Binomial[n,m]计算 排列组合数 $\dfrac{n!}{m!\cdot(n-m)!}$
- Signature[{i1,i2,…}]排列的正负符号

初等超越函数

这些函数的名称一目了然，我们不多加解释。它们是：

Sqrt[z]、z1^z2、Exp[z]、Log[z]、Log[b,z]、Sin[z]、Cos[z]、Tan[z]、Cot[z]、Csc[z]、Sec[z]、ArcSin[z]、ArcCos[z]、ArcCsc[z]、ArcSec[z]、ArcTan[z]、ArcCot[z]、Sinh[z]、Cosh[z]、Tanh[z]、Coth[z]、 Csch[z]、Sech[z]、ArcSinh[z]、ArcCosh[z]、ArcTanh[z]、ArcCoth[z]、ArcCsch[z]、ArcSech[z]。

正交多项式

- LegendreP[n,x], LegendreP[n,m,x] 勒让德多项式
- ChebyshevT[n,x],ChebyshevU[n,x] 切比雪夫多项式
- HermiteH[n,x] Hermite多项式
- LaguerreL[n,x] Laguerrel[n,a,x] 拉盖尔多项式
- JacobiP[n,a,b,x] 雅可比多项式

特殊函数

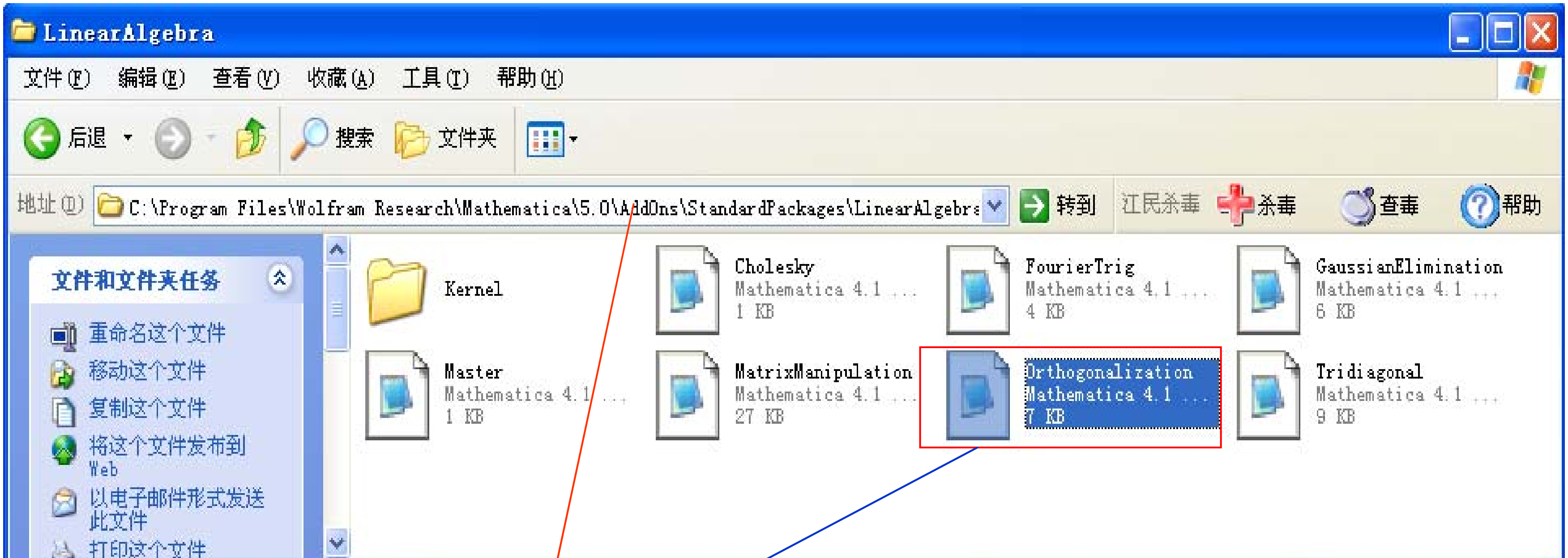
此处我们将不给出特殊函数的具体表达式，读者可查阅相关资料。

- **Beta[a,b],Beta[z,a,b] Bata**函数及不完全**Beta**函数
- **Gamma[z],Gamma[a,z] Gamma**函数及不完全**Gamma**函数
- **Erf[z],Erf[z0,z1]** 误差函数及广义误差函数
- **BesselJ[n,z],BesselY[n,z]** 贝赛尔函数
- **BesselI[n,z],BesselK[n,z]** 修正的贝赛尔函数
- **ExpIntegralE[n,z],LogIntegral[z]** 指数积分与对数积分

数学软件包的读入方法:

在讲义上的此部分,提供了一种数学软件包的读入mathematica中的方法,下面是一种更为简单的方法

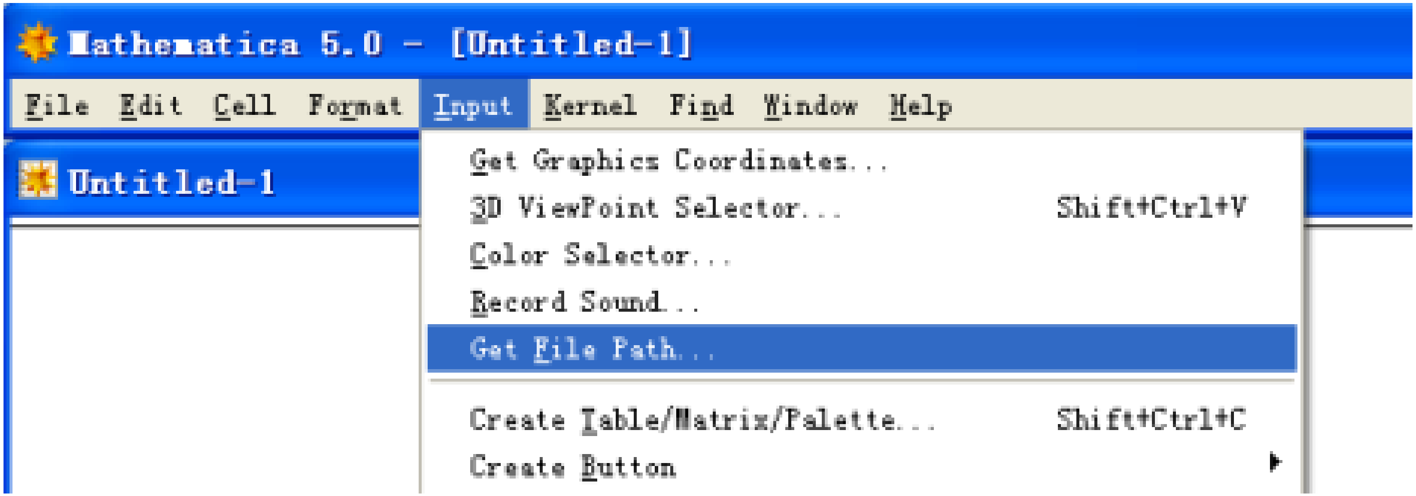
我们下面要读入软件包:



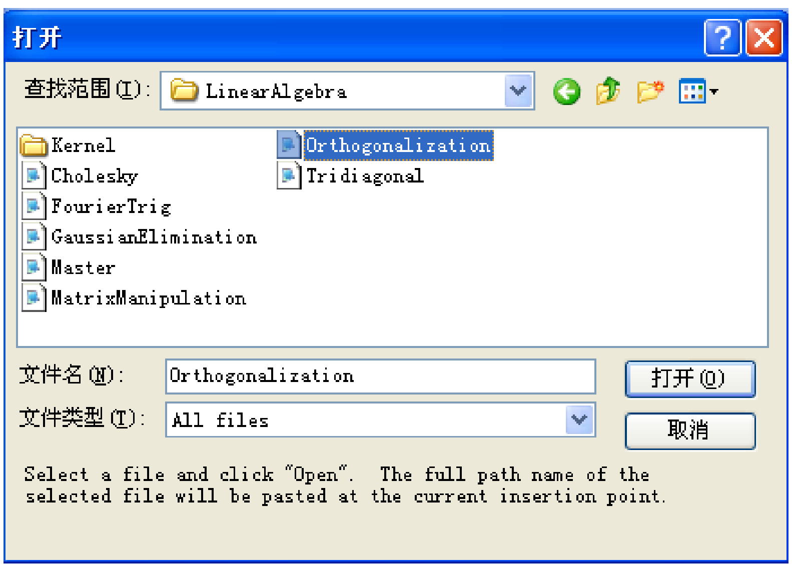
目录:C:\Program Files\Wolfram Research\Mathematica\5.0\AddOns\StandardPackages\LinearAlgebra

文件名: **Orthogonalization.M**

执行以下的mathematica菜单:



在弹出的文件打开框中,一直找到上面的文件为止.



在此文件框中打开此文件,则**mathematica**并没有真正打开文件,而是返回了文件所在的路径

```
"C:\\Program Files\\Wolfram Research\\Mathematica\\
  5.0\\AddOns\\StandardPackages\\LinearAlgebra\\
  Orthogonalization.m"
```

最后,在此返回路径的前面加上2个“<<”并运行,即可读入此软件包,它是关于向量正交化的.

```
<< "C:\\Program Files\\Wolfram Research\\Mathematica
   \\5.0\\AddOns\\StandardPackages\\LinearAlgebra\\
   Orthogonalization.m"
```

使用任何一个编辑软件打开此软件包文件,你会发现如下:

GramSchmidt::usage ="GramSchmidt[vectors] performs the GramSchmidt orthogonalization process on a list of vectors. Note that an inner product can be specified,allowing, for instance, a function space to used. Also, the option

Normalized can be set to determine whether or not the basis is orthonormal."

它是线性代数中的施密特向量正交化函数,当然,此软件包中还有许多其它的函数.

下面是一个例子,此例子一定要在读入软件包后,才能使用.

```
In[9]:= GramSchmidt[{{1, 1, 1}, {2, 1, 3}, {4, 3, 2}}]
```

Out[9]=
$$\left\{\left\{\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right\}, \left\{0, -\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}, \left\{\sqrt{\frac{2}{3}}, -\frac{1}{\sqrt{6}}, -\frac{1}{\sqrt{6}}\right\}\right\}$$

5、自定义函数

在Mathematica中定义一个新函数后，其用法与内部函数是一样的，其定义形式为

fun[var1_,var2_,…]:=expr 或

fun[var1_,var2_,…]=expr

其中函数变量后面的下划线必不可少，以上面的**var1_**为例，其意思是让**var1**匹配所有表达式，但我们可以在下划线的后面限定变量的类型，如**f[n_Integer]**的意思是变量n是一个整数。例如

```
f[x_] = Simplify[D[Exp[2 x] Sin[x], {x, 3}]]
```

$$e^{2x}\,(11\,\mathrm{Cos}[x] + 2\,\mathrm{Sin}[x])$$

```
g[x_, y_] := x^2 + f[y]; g[1, 0]
```

12

上页 下页 退出

Mathematica中的函数调用是递归的，就是说，函数可以调用自身，下面是计算阶乘的函数子程序。

```
Clear[a, k]; a[k_Integer] := k*a[k - 1]; a[0] = 1;
Print["30!=", a[30], "   a[10.0]=", a[10.0]];
30!=265252859812191058636308480000000    a[10.0]=a[10.]
```

由于限制k为整数，所以对**a[10.0]**，**Mathematica**是不会计算的。系统中的许多内部函数都是利用递归调用实现的，**$RecursionLimit**是系统进行递归调用的最大次数，默认值为**256**，你可以将它修改为一个合适的值，这只需对**$RecursionLimit**重新赋值即可。

对于复杂的函数定义，可以用模块**Module[]**定义，其形式为

```
fun[var1_,var2_,…]:=
      Module[{x,y,…},statement 1;
                     statement 2;
                         …;
                     statement N];
```

其中变量x,y称为局部变量，它只在此函数定义的内部起作用(实际上，Module[]就是其它计算机语言中的函数子程序，更进一步解释见第５节)。另外，对于复杂的函数定义，一般要应用条件判断及循环结构，第５节我们将要详细介绍这方面的内容。例如，上面计算阶乘的例子可用模块形式书写为

```
f[n_] := Module[{a, k}, a[0] = 1; a[k_] := k a[k - 1]; a[n]]; f[50]
3041409320171337804361260816606476884437764156896051200000000000
```

如果没有**Return[expr]**命令，**Module[]**返回最后一次计算结果作为函数值。还有，在某些情况下，你可能需要更改**Mathematica**内部函数定义，以适合自己某种特殊要求。例如对**Log[x^s]**和**Log[x y]**，系统并不直接化成**s Log[x]**和**Log[x]+Log[y]**的形式，这我们可以通过更改**Mathematica**对函数**Log[]**的定义来做到这一点，这要用到以下函数

- **Unprotect[command]** 移去系统对命令**command**的保护状态

- **Protect[command]** 加上系统对命令**command**的保护状态

上页 下页 退出

请看下面的具体做法:

```
Clear[a, b, s]; Print[Log[a b], "  ", Log[a^s]]; Unprotect[Log];
Log[a b]   Log[a^s]
Log[a_ b_] := Log[a] + Log[b]; Log[a_^s_] := s Log[a]; Protect[Log]
{Log}
{Log[a b], Log[a^s]}
{Log[a] + Log[b], s Log[a]}
```

Mathematica中的函数定义还有以下形式：

- **Function[x,body]** 定义以**body**为函数体的纯函数，其中**x**可以为用户提供的任何变量来代替

- **Function[{x1,x2,…},body]** 同上，但定义多个变量的纯函数

•**Body&** 若函数体body是单变量函数，此变量规定为#，若为多个变量，则第一个变量为#1，第二个变量#2，依此类推

```
Nest[Function[w, 1/(1+w)], x, 3]
```

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

```
#^2&[1+a]
```

$$(1+a)^2$$

```
Map[#^2&, {a, b, c}]
```

$$\{a^2,\ b^2,\ c^2\}$$

```
(#1^2+#2^2)&[x, y]
```

$$x^2+y^2$$

6、函数及表达式的变换规则

• expr/.rules 变换法则rules只对expr中的每项使用一次

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

$$-\frac{2}{n^3}-\frac{(-2+n^2\pi^2)\,\mathrm{Cos}[n\pi]}{n^3}+\frac{2\pi\,\mathrm{Sin}[n\pi]}{n^2}$$

```
Simplify[%] /. {Cos[n Pi] → (-1)^n, Sin[n Pi] → 0}
```

$$\frac{-2+(-1)^n\,(2-n^2\pi^2)}{n^3}$$

```
x + y /. {{x → a1, y → b1}, {x → a2, y → b2}, {x → a3, y → b3}}
```

{a1 + b1, a2 + b2, a3 + b3}

其中“→”是键入“->”的结果。另外，如果变换条件只有一个，可以不用集合定界符{}，例如

```
x + y + z /. x → 1
```

1 + y + z

•expr//.rules 反复对expr使用rules，直到结果不变为

```
Sin[4a] //. {Sin[n_x_] → Sin[(n-1) x] Cos[x] + Cos[(n-1) x] Sin[x],
             Cos[n_x_] → Cos[(n-1) x] Cos[x] - Sin[(n-1) x] Sin[x]}
```

$$\mathrm{Sin}[a]\,(-2\,\mathrm{Cos}[a]\,\mathrm{Sin}[a]^2+\mathrm{Cos}[a]\,(\mathrm{Cos}[a]^2-\mathrm{Sin}[a]^2))+\mathrm{Cos}[a]\,(2\,\mathrm{Cos}[a]^2\,\mathrm{Sin}[a]+\mathrm{Sin}[a]\,(\mathrm{Cos}[a]^2-\mathrm{Sin}[a]^2))$$

•Nest[f,x,n] 函数f以x为变量，进行n次复合运算

实质上，f是函数的头，即Head[f]，例如

```
Head[Sin[x^2 + y]]
Sin
Nest[Sin, z, 5]
Sin[Sin[Sin[Sin[Sin[z]]]]]
g[x_] = 1/(1 + x); Nest[g, x, 3]
```

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

•NestList[f,x,n] 同上，但形成一个复合函数序列的集合

•Compose[f,g,…,h,x] 函数复合，生成f[g[…h[x]]]

•Composition[f,g,…,h] 同上，但不带有自变量

•ComposeList[{f,g,…,h},x]生成复合序列 {x,f[x],g[f[x]],…}

```
NestList[f, x, 5]
{x, f[x], f[f[x]], f[f[f[x]]], f[f[f[f[x]]]], f[f[f[f[f[x]]]]]
Compose[f, g, h, x]
f[g[h[x]]]
Composition[f, g, h]
Composition[f, g, h]
%[x]
f[g[h[x]]]
ComposeList[{f, g, h}, x]
{x, f[x], g[f[x]], h[g[f[x]]]}
```

- **FixedPoint[f,x] 对x重复f运用，直到结果不变为止**
- **FixedPointList[f,x] 同上，但列出所有中间计算结果**
- **FixedPointList[f,x,SameTest->Comp] 对两个连续的结果运用比较关系comp，比较结果为真时停止运算**

下面以利用牛顿迭代法求５开平方根为例，说明其用法

```
sqrt5[x_] := 1/2 (x + 5/x) // N;
FixedPoint[sqrt5, 2, SameTest → (Abs[#1 - #2] < 10^(-8) &)]
2.23607
FixedPointList[sqrt5, 2]
{2, 2.25, 2.23611, 2.23607, 2.23607, 2.23607}
```

- **FoldList[f,x,{a,b,…}] 构成集合{x,f[x,a],f[f[x,a],b],…}**
- **Fold[f,x,{a,b,…}] 给出函数FoldList的最后一个元素**

```
FoldList[f, x, {a, b, c}]
{x, f[x, a], f[f[x, a], b], f[f[f[x, a], b], c]}
```

- Apply[f,{a,b,c,…}] 对集合运用f，得到f[a,b,c,…]
- Apply[f,expr] 对表达式的最高层应用f
- Apply[f,expr,level] 对表达式的指定层应用f

```
A := {{a, b}, {c, d}}; {Apply[f, A], Apply[f, A, 2]}
{f[{a, b}, {c, d}], {f[a, b], f[c, d]}}
```

- Map[f,expr] 将函数f作用到表达式第一层的每个部件上
- Map[f,expr,level] 将f作用到表达式第n层的每个部件上
- MapAll[f,expr] 对表达式expr的所有部件应用f
- MapThread[f,{expr1,expr2,…}] 对expr1及expr2的相应元素运用f
- MapThread[f,{expr1,expr2,…},lev]对给定层的表达式运用f

```
A := {{a, b}, {c, d}}; {Map[f, A], Map[f, A, 2]}
{{f[{a, b}], f[{c, d}]}, {f[{f[a], f[b]}], f[{f[c], f[d]}]}}
MapAll[f, A]
f[{f[{f[a], f[b]}], f[{f[c], f[d]}]}]
MapThread[f, A]
{f[a, c], f[b, d]}
```

- **Scan[f,expr] 依次计算对expr中的每个元素运用f的值**
- **Scan[f,expr,level]同上，但在指定层上计算**

```
A := {{a, b}, {c, d}}; Scan[Print, A]
{a, b}
{c, d}
```

- **Array[f,n] 生成表{f[1],f[2],…,f[n]}**
- **Array[f,{n1,n2}] 同上，但生成一个二维表**

```
Array[w, 6]
{w[1], w[2], w[3], w[4], w[5], w[6]}
Array[w, {3, 2}]
{{w[1, 1], w[1, 2]}, {w[2, 1], w[2, 2]}, {w[3, 1], w[3, 2]}}
```

•Seclect[expr,f] 在expr中挑选出函数f为True的元素

•Seclect[expr,f,n] 同上，但只选出前n 个使f为True的元素

```
d1 = {3, 5, 12, 9, 4, 7, 1}; Select[d1, # > 4 &]
{5, 12, 9, 7}
```

•Operate[p,f[x]] 算子函数，给出p[f][x]

•Operate[p,f[x],n] 同上，但在函数的第n层应用p

```
Operate[p, f[x]]
p[f][x]
Operate[p, f[g[x]], 0]
p[f[g[x]]]
Operate[p, f[g[x]], 1]
p[f][g[x]]
Operate[p, f[g[x]], 2]
f[g[x]]
```

上页 下页 退出