第三章　线性代数

1 多项式

多项式是代数学中最基本的对象之一，它不但与高次方程的讨论有关，而且是进一步学习代数以及其它数学分支的基础.

1.1 多项式生成及类型测试

在 Maple 中，多项式由名称、整数和其他 Maple 值，通过+、-、*和^等组合而成. 例如：

```
> p1:=5*x^5+3*x^3+x+168;
```

$$p1 := 5x^5 + 3x^3 + x + 168$$

这是一个整系数单变量多项式. 多元多项式和定义在其他数域上的多项式可以类似构造：

```
> p2:=3*x*y^2*z^3+2*sqrt(-1)*x^2*y*z+2002;
```

$$p2 := 3xy^2z^3 + 2Ix^2yz + 2002$$

由此可以看出, Maple 中多项式的生成与“赋值”命令相似.

另外，还可以通过函数 randpoly 生成随机多项式，生成一个关于 vars 的随机多项式的格式如下：

randpoly(vars, opts);

其中, vars 表示变量或者变量列表或集合, opts 为可选项方程或者指定属性的名称. 如：

```
> randpoly(x);      # 随机生成关于 x 的 5 次(默认)多项式
```

$$-42x^5 + 88x^4 - 76x^3 - 65x^2 + 25x + 28$$

```
> randpoly([x, y], terms=8);      # 随机生成关于[x, y]二元 8 项多项式
```

$$-78xy + 62x^3 + 11x^2y + 88x^3y + xy^3 + 30y^4 + 81x^4y - 5x^2y^3$$

```
> randpoly([x, sin(x), cos(x)]);
```

$$-73\sin(x)\cos(x) - 91\sin(x)\cos(x)^2 + x^3\cos(x) + 5x^4\sin(x) - 86x^2\cos(x)^3 + 43\sin(x)^4\cos(x)$$

而要随机生成关于[x, y, z]的密集的、均匀的、度为 2 的多项式的命令为：

```
> randpoly([x,y,z],dense,homogeneous,degree=2);
```

$$-85x^2 - 55zx - 37yx - 35z^2 + 97yz + 50y^2$$

用 type 命令可以测试多项式的类型：

```
> type(p1, polynom(integer, x));          #测试 p1 是否是一个关于 x 的整系数多项式
```

true

```
> type(p2, polynom(complex, {x, y, z}));  #测试 p2 是否是一个关于{x, y, z}的复系数多项式
```

true

1.2 提取多项式系数

coeff函数用来提取一元多项式的系数，而多元多项式所有系数的提取用命令coeffs，指定系数的提取用命令 coftayl.

(1) 提取多项式 p 中 x^n 的系数使用命令：coeff(p, x^n);或 coeff(p, x, n);

(2) 提取多项式 p 中变量 x 的所有系数并将相应的 x 幂存于变量 t 中：coeffs(p, x, 't');

(3) 返回 expr 在 x=a 处的 Taylor 展式中(x-a)^k 的系数: coeftayl(expr, x=a, k);

```
> p:=2*x^2+3*y^3*x-5*x+68;
```

$$p := 2\,x^2 + 3\,y^3\,x - 5\,x + 68$$

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

$$3\,y^3 - 5$$

```
> coeff(x^4-5*x^2-sin(a)*(x+1)^2, x^2);
```

$$-5 - \sin(a)$$

```
> s:=3*x^2*y^2+5*x*y;
```

$$s := 3\,x^2\,y^2 + 5\,x\,y$$

```
> coeffs(s);
```

$$5,\ 3$$

```
> coeffs(s, x, 't');
```

$$5\,y,\ 3\,y^2$$

```
> t;
```

$$x,\ x^2$$

```
> coeftayl(exp(x), x=0, 10);
```

$$\frac{1}{3628800}$$

```
> p:=3*(x+1)^3+sin(Pi/3)*x^2*y+x*y^3+x-6;
```

$$p := 3\,(x+1)^3 + \frac{1}{2}\sqrt{3}\,x^2 y + x\,y^3 + x - 6$$

> **coeftayl(p, x=-1, 1);**

$$1 - \sqrt{3}\,y + y^3$$

> **coeftayl(p, [x, y]=[0, 0], [1, 0]);**

$$10$$

返回默认为降序排列的多元多项式的首项和末项系数分别使用命令 lcoeff、tcoeff：

> **lcoeff(p, x);**

$$3$$

> **tcoeff(p, x);**

$$-3$$

1.3 多项式的约数和根

1.3.1 多项式的最大公约因式(gcd)/最小公倍因式(lcm)

求多项式的最大公约因式/最小公倍因式的命令与求两个整数最大公约数/最小公倍数命令一样，都是 gcd/lcm. 命令格式分别为：

gcd(p1, p2, 't', 's');

lcm(p1, p2, 't', 's');

其中，第 3 个参数 t 赋值为余因子 p1/gcd(p1, p2)，第 4 个参数 s 赋值为余因子 p2/gcd(p1, p2).

> **p1:=x^4+x^3+2*x^2+x+1;**

$$p1 := x^4 + x^3 + 2\,x^2 + x + 1$$

> **p2:=x^2+x+1;**

$$p2 := x^2 + x + 1$$

> **gcd(p1, p2, 't', 's');**

$$x^2 + x + 1$$

> **t, s;**

$$x^2 + 1,\ 1$$

> **lcm(p1, p2);**

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

1.3.2 多项式的平方根(psqrt)和第 n 次方根(proot)

求多项式 p 的平方根，若不是完全平方，则返回_NOSQRT：psqrt(p);

求多项式 p 的 n 次方根，若不是完全 n 次方，则返回_NOROOT：proot(p, n);

> **p:=x^4+4*x^3+6*x^2+4*x+1;**

$$p := x^4 + 4\,x^3 + 6\,x^2 + 4\,x + 1$$

> **psqrt(p);**

$$x^2 + 2\,x + 1$$

> **proot(p, 4);**

$$x + 1$$

> **proot(p, 8);**

$$\_NOROOT$$

1.3.3 多项式相除的余式(rem)/商式(quo)

计算 p1 除以 p2 的余式，将商式赋值给 q 的命令格式为：rem(p1, p2, x, 'q');

计算 p1 除以 p2 的商式，将余式赋值给 r 的命令格式为：quo(p1, p2, x, 'r');

余式和商式满足：p1=p2*q+r，其中 degree(r, x)<degree(p2, x)

> **rem(x^5+x^3+x, x+1, x, 'q');**

$$-3$$

> **q;**

$$x^4 - x^3 + 2\,x^2 - 2\,x + 3$$

> **quo(x^3+x^2+x+1, x-1, x, 'r');**

$$x^2 + 2\,x + 3$$

> **r;**

$$4$$

1.4 多项式转换及整理

1.4.1 将多项式转换成 Horner 形式

将多项式 poly 转换成关于变量 var 的 Horner 形式或者嵌套形式的命令格式如下：

convert(poly, horner, var);

> **convert(x^5+x^4+x^3+x^2+x+1, horner, x);**

$$1 + (1 + (1 + (1 + (x + 1)\,x)\,x)\,x)\,x$$

> **convert(x^3*y^3+x^2*y^2+x*y+1, horner, [x, y]);**

$$1 + (y + (y^2 + y^3\,x)\,x)\,x$$

1.4.2 将级数转换成多项式形式

将级数(series)转换成多项式(polynom)事实上就是忽略函数的级数展开式中的余项，

其命令格式为：

convert(series, polynom);

> **s:=series(sin(x), x, 10);**

$$s := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9 + O(x^{10})$$

> **type(s, polynom);**

$$false$$

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

$$p := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 - \frac{1}{5040}x^7 + \frac{1}{362880}x^9$$

> **type(p, polynom);**

$$true$$

1.4.3 将级数转换成有理多项式(有理函数)

将级数 series(laurent 级数或 Chebyshev 类型级数)转换成有理多项式(有理函数)ratpoly 的命令格式为：

convert(series, ratpoly);

> **series(exp(x^2), x, 15);**

$$1 + x^2 + \frac{1}{2}x^4 + \frac{1}{6}x^6 + \frac{1}{24}x^8 + \frac{1}{120}x^{10} + \frac{1}{720}x^{12} + \frac{1}{5040}x^{14} + O(x^{15})$$

> **convert(%, ratpoly);**

$$\frac{\frac{1}{120}x^6 + \frac{1}{10}x^4 + \frac{1}{2}x^2 + 1}{-\frac{1}{120}x^6 + \frac{1}{10}x^4 - \frac{1}{2}x^2 + 1}$$

1.4.4 合并多项式系数(合并同类项)

将多项式具有相同次幂的项的系数合并在一起(包括正的、负的或者分数次幂)，即合并同类项(称为多项式的典范形式)，用命令 collect:

collect(p, x);

collect(p, x, form, func);

collect(p, x, func);

其中 x 是表示单变量 x 或多变量 x1, x2, …, xn 的一个列表或集合.

> **collect(a*ln(x)-ln(x)*x-x, ln(x));**

$$(a - x)\ln(x) - x$$

> **collect(x*(x+1)+y*(x+1), x);;**

$$x^2 + (1 + y)x + y$$

> **collect(x*(x+1)+y*(x+1), y);**

$$x\,(x+1)+y\,(x+1)$$

```
> p := x*y+a*x*y+y*x^2-a*y*x^2+x+a*x:
collect( p, [x, y], recursive );
```

$$(1-a)\,y\,x^2+((1+a)\,y+1+a)\,x$$

```
> collect( p, [y, x], recursive );
```

$$((1-a)\,x^2+(1+a)\,x)\,y+(1+a)\,x$$

```
> collect( p, {x, y}, distributed );
```

$$(1+a)\,x+(1+a)\,x\,y+(1-a)\,y\,x^2$$

其中的参数recureive为递归式的，而distributed为分布式的。

1.4.5 将多项式(或者值的列表)排序

将多项式(或者值的列表)按升(或降)序次方排序时作命令sort. 命令格式：

sort(L);

sort(L, F);

sort(A);

sort(A, V);

其中, L—表示要排序的列表; F(可选项)—带两个参数的布尔函数; A—代数表达式; V(可选项)—变量

sort函数将列表L按升序次方, 将代数表达式A中的多项式按降序次方排列.

如果给定了F, 它将用来定义一个排序的列表. 如果F是符号“<”或者数值, 那么L是一个列表类型, 按数值型降序排列; 如果F是符号“>”, L按数值型升序排列; 如果F是一个词典编纂的符号, 那么字符串和符号列表将按词典编纂的次序排列. 另外, F必须是一个有两个参数的布尔函数.

在Maple中, 多项式并不自动按排序次序存储而是按建立的次序存储. 值得注意的是, sort函数对多项式的排序是破坏性的操作, 因为输入的多项式将会按排序后的次序存储.

用来对列表排序的算法是一种带早期排序序列监测的合并排序的递归算法, 而用于对多项式排序的算法是一个原位替换排序.

```
> sort([3, 2, 1, 5, 4]);
```

$$[1, 2, 3, 4, 5]$$

```
> sort(x^3+x^2+1+x^5, x);
```

$$x^5+x^3+x^2+1$$

```
> sort((y+x)^2/(y-x)^3*(y+2*x), x);
```

$$\frac{(x+y)^2\,(2\,x+y)}{(-x+y)^3}$$

1.4.6 多项式的因式分解

一般情况下，计算带有整数、有理数、复数或代数数系数的多项式的因式分解使用命令 factor，命令格式为：

factor(expr, K);

其中, expr 为多项式, K 为因式分解所在的区域

```
> factor(x^3+y^3);
```

$$(x+y)\,(x^2-x\,y+y^2)$$

```
> factor(t^6-1,  (-3)^(1/2));
```

$$\frac{1}{16}(2\,t+1-\sqrt{-3})\,(2\,t+1+\sqrt{-3})\,(2\,t-1-\sqrt{-3})\,(2\,t-1+\sqrt{-3})\,(t+1)\,(t-1)$$

```
> factor(x^3+5, {5^(1/3),  (-3)^(1/2)});
```

$$\frac{1}{4}(2\,x-5^{(1/3)}-\sqrt{-3}\,5^{(1/3)})\,(2\,x-5^{(1/3)}+\sqrt{-3}\,5^{(1/3)})\,(x+5^{(1/3)})$$

但是factor不分解整数，也不分解多项式中的整数系数，整数或整数系数的分解使用函数ifactor：

```
> ifactor(2^99+1);
```

$$(3)^3\ (19)\ (67)\ (683)\ (242099935645987)\ (20857)\ (5347)$$

1.5 多项式运算

1.5.1 多项式的判别式

多项式判别式在多项式求根中具有重要作用，例如二次多项式的判别式. 求多项式 p 关于变量 x 的判别式的命令格式为：

discrim(p, x);

```
> p1:=a*x^2+b*x+c;
```

$$p1 := a\,x^2+b\,x+c$$

```
> discrim(p1, x);
```

$$-4\,a\,c+b^2$$

```
> p2:=a*x^3+b*x^2+c*x+d;
```

$$p2 := a\,x^3+b\,x^2+c\,x+d$$

```
> discrim(p2, x);
```

$$-27\,a^2\,d^2+18\,a\,d\,c\,b+b^2\,c^2-4\,b^3\,d-4\,a\,c^3$$

1.5.2 多项式的范数

计算关于变元 v 的多项式 p 的第 n 阶范数用命令 norm，格式如下：

norm(p, n, v);

其中 p 是展开的多项式，n 为实常数或无穷大. 对于 n≥1，范数 norm 定义为 norm(p, n, v)=sum(abs(c)^n), c=[coeffs(p, v)]^(1/n).

```
> norm(x^3+x^2+x+1, 1);
```

$$4$$

```
> norm(x-3*y, 2);
```

$$\sqrt{10}$$

```
> norm(x-3*y, infinity);
```

$$3$$

1.5.3 bernoulli 数及多项式

bernoulli 函数计算第 n 阶 bernoulli 数和关于表达式 x 的第 n 阶 bernoulli 多项式 bernoulli(n, x), bernoulli(n, x)通过指数生成函数定义为：

$$\frac{te^{xt}}{e^t-1}=\sum_{n=0}^{\infty}\frac{\mathrm{bernoulli}(n,x)}{n!}t^n$$

第 n 阶 bernoulli 数定义为：bernoulli(n)= bernoulli(n, 0)

```
> bernoulli(6);
```

$$\frac{1}{42}$$

```
> bernoulli(10, x);
```

$$\frac{5}{66}-\frac{3}{2}x^2+5\,x^4-7\,x^6+\frac{15}{2}x^8+x^{10}-5\,x^9$$

```
> bernoulli(6, 1/2);
```

$$\frac{-31}{1344}$$

1.5.4 用 Bernstein 多项式近似一个函数

bernstein 多项式起因于 stone-weierstrass 近似分析理论，该理论认为任何连续函数都可以在一个闭区间用多项式序列来近似，bernstein 多项式就是该理论的一个实例. 若给定 p:=(n, i, x)->binomial(n, i)*x^i*(1-x)^(n-i)时，Bernstein 多项式定义为：

$$\mathrm{Berstein}(n,f,x)=\sum_{i=0}^{n}p(n,i,x)f(\frac{i}{n})$$

其中, binomial(n, r) = GAMMA(n+1)/GAMMA(r+1)/GAMMA(n-r+1)

在 Maple 中，在[0, 1]区间上近似等于函数 f(x)的关于 x 的 n 次方的 Bernstein 多项的操作命令是：

bernstein(n, f, x);

其中, f 必须是用一个程序或者操作符指定的单变量函数.

```
> bernstein(3, x->1/(x+1), z);
```

$$1 - \frac{3}{4}z + \frac{3}{10}z^2 - \frac{1}{20}z^3$$

```
> f:= proc(t) if t < 1/2 then 4*t^2 else 2 - 4*t^2 end if end proc:
  bernstein(8, f, x);
```

$$\frac{1}{2}x - 63\,x^5 + \frac{7}{2}x^2 + 119\,x^6 + 20\,x^8 - 82\,x^7$$

1.5.5 获取多项式的最高/最低次方

获取多项式的最高/最低次方的 Maple 命令分别为 degree 和 ldegree，格式如下：

degree(p, x);

ldegree(p, x);

```
> p:=3/x^3+x^3-(x-1)^4;
```

$$p := 3\frac{1}{x^3} + x^3 - (x-1)^4$$

```
> degree(p, x); ldegree(p, x);
```

$$4$$

$$-3$$

```
> degree(x*sin(x), x);
```

$$FAIL$$

```
> degree(x*sin(x), sin(x));
```

$$1$$

1.5.6 Euler 数及多项式

第 n 阶 Euler 多项式 E(n, x)由指数生成函数定义为：

$$\frac{2e^{xt}}{e^t H} = \sum_{n=0}^{\infty} \frac{\mathrm{E}(n,x)}{n!} t^n$$

据此，第 n 阶 Euler 数 E(n)由指数生成函数定义为：

$$\frac{2}{e^t + e^{-t}} = \sum_{n=0}^{\infty} \frac{\mathrm{E}(n)}{n!} t^n$$

由此可见，第 n 阶 Euler 多项式和 Euler 数的关系为：

$$E(n)=2^n E(n,\frac{1}{2})$$

在 Maple 中，获取第 n 阶 Euler 多项式和 Euler 数的命令分别为：

euler(n, x);　　euler(n);

其中, n 为非负整数, x 为表达式.

> **euler(6);**

$$-61$$

> **euler(6, x);**

$$x^6-3x^5+5x^3-3x$$

> **euler(6, 1/2);**

$$\frac{-61}{64}$$

1.5.7 多项式插值

函数 interp 计算次方小于或等于 n 的过插值点(x[1], y[1]), (x[2],　y[2]), …, (x[n+1], y[n+1])关于变量 v 的多项式，命令格式如下：

interp(x, y, v);

值得注意的是，如果相同的 x 值输入了 2 次，不论是否输入了相同的 y 值都会导致错误，也就是说所有插值必须不一样. 如过点(1, 2), (2, 4), (3, 6), (4, 8), (5, 10), (6, 12), (7, 14), (8, 16), (9, 18), (10, 22)的关于 Z 的插值多项式的计算如下：

> **interp([1,2,3,4,5,6,7,8,9,10], [2,4,6,8,10,12,14,16,18,22], z);**

$$\frac{1}{181440}z^9-\frac{1}{4032}z^8+\frac{29}{6048}z^7-\frac{5}{96}z^6+\frac{3013}{8640}z^5-\frac{95}{64}z^4+\frac{4523}{1134}z^3-\frac{6515}{1008}z^2+\frac{9649}{1260}z-2$$

1.5.8 计算自然样条函数

计算一个关于变量 z 的次数是 n(默认为 3)的分段多项式来近似 X, Y 数据值. X 值必须唯一且按升序排列，而对 Y 值没有特别的约定. 命令格式为：

spline(X, Y, z, n);

> **spline([1, 2, 3, 4], [2, 3, 6, 5], x, linear);**#生成线性样条插值

$$\begin{cases} 1+x & x<2 \\ -3+3x & x<3 \\ 9-x & otherwise \end{cases}$$

> **spline([0, 1, 2, 3], [0, 1, 4, 3], x, cubic);**# 生成三次样条插值

$$\begin{cases} \frac{1}{5}x+\frac{4}{5}x^3 & x<1 \\ \frac{14}{5}-\frac{41}{5}x+\frac{42}{5}x^2-2x^3 & x<2 \\ -\frac{114}{5}+\frac{151}{5}x-\frac{54}{5}x^2+\frac{6}{5}x^3 & otherwise \end{cases}$$

1.6 有理式

所谓有理式，是指可以表示成 f/g 形式的式子，其中 f 与 g 均为多项式，且 g≠0.

1.6.1 获取表达式的分子/分母

对于一个有理式 x，可以用 **numer**(x)和 **denom**(x)来获得它的分子(numerator)和分母(denumirator):

```
> f:=x^2+3*x+2: g:=x^2+5*x+6: h:=f/g;
```

$$h := \frac{x^2+3x+2}{x^2+5x+6}$$

```
> numer(h);denom(h);
```

$$x^2+3x+2$$

$$x^2+5x+6$$

1.6.2 有理表达式的标准化

和数字除法不同, Maple 并不自动化简分式，使分子和分母互不可约，除非分子和分母都表示成乘积的形式并且有公共部分. 如果要将有理式化简成最简形式即标准化，需要调用函数 **normal**( ) (也称正则化). 事实上，normal 函数提供了化简的一种基本形式，它首先识别在有理数主域上等于 0 的表达式包括求和、乘积、整数幂次和变量构成的任何表达式. Normal 的结果是使有理式转化为约化的标准形式，即：分子分母是带整数系数的素数多项式.

```
> normal(f/g);
```

$$\frac{x+1}{x+3}$$

```
> normal(x^2-(x+1)*(x-1)-1);
```

$$0$$

```
> normal((f(x)^2-1)/(f(x)-1));
```

$$f(x)+1$$

```
> normal(1/x+x/(x+1), expanded);
```

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

> **normal(2/x+y/3);**

$$\frac{1}{3}\frac{6+x\,y}{x}$$

1.6.3 将有理式转换为多项式与真分式和的形式

convert/parfrac可将有理函数 f 分解为多项式与真分式和的形式，这是有理函数积分的基础. 命令格式如下：

convert(f, parfrac, x, K);

其中, f为x的有理函数, x为主变量名称, K为可选参数(实数—real, 复数—complex, 扩展域—a field extended, 真—true, 假—false, 无平方项—sqrfree)等.

> **p:=(x^5+1)/(x^3+1);**

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

> **convert(p, parfrac, x);**

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

> **convert(x/(x-a)^2, parfrac, x);**

$$\frac{a}{(-x+a)^2} - \frac{1}{-x+a}$$

1.6.4 将浮点数转换为接近的有理数

convert/rational将一个浮点数转换成一个近似的有理数, 转换的精度取决于环境变量Digits的值. 命令格式为：

convert(float, rational, Digits);

> **convert(1.333333333, rational);**

$$\frac{4}{3}$$

> **convert(evalf(Pi), rational,6);**

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

```
> convert(evalf(Pi), rational, 3);
```

$$\frac{22}{7}$$

2 矩阵基本运算

2.1 数组和向量

在 Maple 中，数组(**array**)由命令 **array** 产生，其下标变量(**index**)可以自由指定. 下标由 1 开始的一维数组称为向量(**vector**)，二维以上的数组称为矩阵(**matrix**)，因此，可以说向量与矩阵是数组的特殊形式. 生成数组与向量的命令分别列示如下：

```
array (m, n);                  #产生一个下标变量为 m 到 n 的一维数组
array (m, n, [a1, a2 .. ak]);  #产生一维数组，并依次填入初值
array ([a1, a2 ..ak]);         #产生下标变量为 1 到 k 的一维数组，并依次填入初值
vector(n, [v1, v2, …, vn]);    #产生 n 维向量(n 可省略)
```

```
> A:=array(-1..2);
```

$$A := \text{array}(-1 \ldots 2, [\ ])$$

```
> A[0]:=12;
```

$$A_0 := 12$$

```
> A[1]:=2*x^2+x+4;
```

$$A_1 := 2x^2 + x + 4$$

```
> A;
```

$$A$$

```
> lprint(eval(A));
array(-1 .. 2, [(0)=12, (1)=2*x^2+x+4])
> B:=array(0..2, [x, x^2, x^3]);
```

$$B := \text{array}(0 \ldots 2, [\ ])$$

$$(0) = x$$

$$(1) = x^2$$

$$(2) = x^3$$

```
> C:=array(1..3, [x, x^2, x^3]);
```

$$C := [x, x^2, x^3]$$

```
> print(C);
```

$$[x, x^2, x^3]$$

```
> type(B, vector);
```

$$false$$

```
> type(B, array);
```

$$true$$

```
> type(C, vector);
```

$$true$$

```
> type(C, array);
```

$$true$$

```
> vector(4, [1, x, x^2, x^3]);
```

$$[1, x, x^2, x^3]$$

```
> Vector(1..3, 5);
```

$$\begin{bmatrix} 5 \\ 5 \\ 5 \end{bmatrix}$$

注意 **vector** 命令使用时大小写的区别：矢量 vector(小写)指 linalg 包的基于数组的矢量，而 Vector(大写)指 LinearAlgebra 包的基于 rtable 的矢量.

当我们把一个向量赋给某个变量后，Maple 在显示这个向量时，只会显示出变量名称，而不会显示出向量的内容，这个设计可大大地简化向量的显示方式. 如果想要显示向量的内容，可用 **evalm** 指令来强迫对向量求值. 事实上，**evalm** 不仅可以显示向量，也可以显示矩阵.

```
> A:=array([x, y, z]);
```

$$A := [x, y, z]$$

```
> B:=array([1, 2, 3]);
```

$$B := [1, 2, 3]$$

```
> 2*A+B;
```

$$2A + B$$

```
> evalm(2*A+B);
```

$$[2x + 1, 2y + 2, 2z + 3]$$

另一个有用的命令是将列表、数组或Vector矢量A转换成vector的函数convert：

convert(A, vector);

```
> A:=array(1..2, 1..2, [[1, 2], [3, 4]]):
  convert(A, vector);
```

$$[1, 2, 3, 4]$$

> V:=Vector([1, 6, 8]);

$$V := \begin{bmatrix} 1 \\ 6 \\ 8 \end{bmatrix}$$

> type(V, vector);

$$false$$

> convert(V, vector);

$$[1, 6, 8]$$

> type(%, vector);

$$true$$

2.2 矩阵的建立

在使用 Maple 进行线性代数运算时，需要先载入线性代数工具包—**linalg**，绝大部分线性代数运算函数都存于该工具包中.

在 Maple 中，建立矩阵用 matrix 命令. 如建立一个 m 行 n 列的矩阵的命令格式为：

matrix(m,n);

matrix(m,n,init);

其中，init 可以有很多种选择，如程序、数组、列表、方程组、代数表达式或者矩阵的初值等。

> with(linalg):

> A:=array([[1,2,3],[4,5,6],[7,8,9]]);

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

> f:=(i,j)->i^2+j^3;

$$f := (i, j) \to i^2 + j^3$$

> B:=matrix(4,5,f);

$$B := \begin{bmatrix} 2 & 9 & 28 & 65 & 126 \\ 5 & 12 & 31 & 68 & 129 \\ 10 & 17 & 36 & 73 & 134 \\ 17 & 24 & 43 & 80 & 141 \end{bmatrix}$$

> C:=matrix(2,2,[1,2,3,4]);

$$C := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

而函数 diag 可以生成块对角阵, band 函数可以建立以常数对角元素的带状矩阵：

```
> diag(A,C);
```

$$\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 \\ 7 & 8 & 9 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 & 4 \end{bmatrix}$$

```
> M:=Matrix([[1,1,1],[2,2,2,2],[3,3,3,3]], shape=band[1,3], scan=band[1,3]);
```

$$M := \begin{bmatrix} 2 & 3 & 0 & 0 & 0 \\ 1 & 2 & 3 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix}$$

内建于 linalg 的另一个有用的命令是从线性方程组中提取系数矩阵或增广矩阵，实现这一功能的函数是 genmatrix，其命令格式为：

genmatrix(eqns, vars, flag);

```
> eqns:={x+2*z=a,3*x-5*y=6-z};
```

$$eqns := \{x + 2\,z = a, 3\,x - 5\,y = 6 - z\}$$

```
> A:=genmatrix(eqns,[x,y,z],'flag');
```

$$A := \begin{bmatrix} 1 & 0 & 2 & a \\ 3 & -5 & 1 & 6 \end{bmatrix}$$

```
> B:=genmatrix(eqns,[x,y,z]);
```

$$B := \begin{bmatrix} 1 & 0 & 2 \\ 3 & -5 & 1 \end{bmatrix}$$

由此例可以看出，若加参数**'flag'**则提取增广矩阵，否则提取系数矩阵，这一点对于求解线性方程组很重要.

另外, Maple 中还有一些预先编好的矩阵，可以直接由函数(命令)获得：

```
> hilbert(3);
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

```
> hilbert(3,x-1);
```

$$\begin{bmatrix} \frac{1}{3-x} & \frac{1}{4-x} & \frac{1}{5-x} \\ \frac{1}{4-x} & \frac{1}{5-x} & \frac{1}{6-x} \\ \frac{1}{5-x} & \frac{1}{6-x} & \frac{1}{7-x} \end{bmatrix}$$

```
> toeplitz([1,2,3,4]);
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

```
> vandermonde([1,x-1,(x-1)^2,(x-1)^3]);
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & x-1 & (x-1)^2 & (x-1)^3 \\ 1 & (x-1)^2 & (x-1)^4 & (x-1)^6 \\ 1 & (x-1)^3 & (x-1)^6 & (x-1)^9 \end{bmatrix}$$

```
> fibonacci(3);
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

还有一个有用的命令是随机生成矩阵的函数 randmatrix：

```
> randmatrix(3,3);
```

$$\begin{bmatrix} 63 & 57 & -59 \\ 45 & -8 & -93 \\ 92 & 43 & -62 \end{bmatrix}$$

同样，随机生成向量的函数为 randvector.

2.3 矩阵的基本运算

矩阵的基本运算命令列于下表，而矩阵的代数运算最直观的方法莫过于使用函数 **evalm** 了，只要把矩阵的代数计算表达式作为它的参数，就可以得到结果.

运算	函数调用	等效的运算符运算
加法	matadd(A, B)	evalm(A+B)
数乘	scalarmul(A, expr)	evalm(A*expr)

乘法	multiply(A, B, …)	evalm(A &*B &* …)
逆运算	inverse(A)	evalm(1/A)或 evalm(A^(-1))
转置	transpose(A)	无
行列式	det(A)	无
秩	rank(A)	无
迹	trace(A)	无

```
> A:=randmatrix(2,2);
```

$$A := \begin{bmatrix} 72 & 66 \\ -29 & -91 \end{bmatrix}$$

```
> B:=randmatrix(2,2);
```

$$B := \begin{bmatrix} -53 & -19 \\ -47 & 68 \end{bmatrix}$$

```
> matadd(A,B);   evalm(A+B);
```

$$\begin{bmatrix} 19 & 47 \\ -76 & -23 \end{bmatrix}$$

```
> multiply(A,B);   evalm(A*B);
```

$$\begin{bmatrix} -6918 & 3120 \\ 5814 & -5637 \end{bmatrix}$$

```
> inverse(%);   evalm(1/%);
```

$$\begin{bmatrix} \frac{-1879}{6952362} & \frac{-520}{3476181} \\ \frac{-323}{1158727} & \frac{-1153}{3476181} \end{bmatrix}$$

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

$$\begin{bmatrix} -.0002702678600 & -.0001495894489 \\ -.0002787541845 & -.0003316858357 \end{bmatrix}$$

```
> transpose(A);
```

$$\begin{bmatrix} 72 & -29 \\ 66 & -91 \end{bmatrix}$$

```
> rank(B); trace(B);
```

$$2$$

$$15$$

值得注意的是，矩阵乘法运算符“&*”有着和数乘“*”、除法“/”相同的优先级，在输入表达式时需要注意，必要时要使用括号，请看下面实验：

```
> evalm(A&*1/A);
```

Error, (in evalm/amperstar) &* is reserved for matrix multiplication

> **evalm(A&*(1/A));**

$$\&*(\ \ )$$

上面最后的输出结果表示单位阵.

2.4 矩阵的求值

数组和单个的表达式不同, 具有数组类型的变量不会自动求值, 而需要用 **eval** 等函数的显式调用来强制地求值. 作为数组的特例, 矩阵和向量也是一样, 但要让矩阵最终完成求值还需使用函数 **map**, **map** 函数的主要功能是, 对于任意一元函数, 不加任何说明, 就可以作用在整个数组上, 得到的结果是每一个元素分别调用函数所得结果所组成的数组.

> **with(linalg):**

> **Q:=matrix([[cos(alpha), sin(alpha)], [-sin(alpha), cos(alpha)]]);**

$$Q := \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

> **alpha:=0;**

$$\alpha := 0$$

> **eval(Q);**

$$\begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

> **map(eval, Q);**

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

另一个对矩阵中元素变量代换的命令是 subs 函数(但只能进行一层求值), 例如:

> **P:=matrix(2, 2, [1, 2, x, x^3]);**

$$P := \begin{bmatrix} 1 & 2 \\ x & x^3 \end{bmatrix}$$

> **subs(x=8, eval(P));**

$$\begin{bmatrix} 1 & 2 \\ 8 & 512 \end{bmatrix}$$

2.5 矩阵分解

矩阵分解在矩阵运算中的作用最明显的一点是可以大幅度提高运算效率. 矩阵分解算法较多，主要有LU分解、QR分解等。

2.5.1 **LU 分解**

LUdecomp(**A**, P='p', L='l', U='u' , U1='u1', R='r', rank='ran', det='d');

其中, A 为矩阵(长方形), P='p'—主元素因子, L='l'—单位下三角因子, U='u'—上三角因子, U1='u1'—修改的 U 因子, R='r'—行减少因子, rank='ran'—A 的秩, det='d'—U1 的行列式. 当然, 此命令的简写形式为：LUdecomp(**A**);

```
> A:=vandermonde([1,3,5,7]);
```

$$A := \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 3 & 9 & 27 \\ 1 & 5 & 25 & 125 \\ 1 & 7 & 49 & 343 \end{bmatrix}$$

```
>LUdecomp(A, P='p', L='l', U='u' , U1='u1', R='r', rank='ran', det='d');
```

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 8 & 26 \\ 0 & 0 & 8 & 72 \\ 0 & 0 & 0 & 48 \end{bmatrix}$$

```
> evalm(l);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 3 & 1 \end{bmatrix}$$

```
> ran;
```

$$4$$

```
> d;
```

$$768$$

对于代数元素矩阵, 只有首项元素为 0 时才选主元素, 主元素矩阵返回为 P.

2.5.2 **QR 分解**

在QR分解中, 矩阵A被看作乘积Q•R, 此处, Q为正交或归一化矩阵, R为上三角阵. 该分解是对一系列线性无关向量作Gram-Schmidt正交化, 因此, Q包含了正交的向量, R的列记录了产生原向量的线性组合. 命令格式为：

QRdecomp(A, Q='q', rank='r', fullspan=value);

其中, A 为矩阵(长方形), Q='q'—A 的 Q 因子, rank='ran'—A 的秩, fullspan=value—在 Q 中包含空格(true 或 false).

```
> A:=matrix(3,3,[1,2,3,4,5,6,7,8,10]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

```
> QRdecomp(A,Q='q',rank='r');
```

$$\begin{bmatrix} \sqrt{66} & \frac{13}{11}\sqrt{66} & \frac{97}{66}\sqrt{66} \\ 0 & \frac{3}{11}\sqrt{11} & \frac{5}{11}\sqrt{11} \\ 0 & 0 & \frac{1}{6}\sqrt{6} \end{bmatrix}$$

```
> r;
```

$$3$$

```
> evalm(q);
```

$$\begin{bmatrix} \frac{1}{66}\sqrt{66} & \frac{3}{11}\sqrt{11} & \frac{1}{6}\sqrt{6} \\ \frac{2}{33}\sqrt{66} & \frac{1}{11}\sqrt{11} & -\frac{1}{3}\sqrt{6} \\ \frac{7}{66}\sqrt{66} & -\frac{1}{11}\sqrt{11} & \frac{1}{6}\sqrt{6} \end{bmatrix}$$

另外，对实正定矩阵可采用Cholesky分解，这是一种形式更对称的LU分解.

3 矩阵的初等变换和线性方程组求解

3.1 矩阵的初等变换

矩阵的初等变换是各种消去法的基础，是求解线性方程组的基础. 对于符号运算，有时候我们不仅要求最后的求解结果，而且要求中间的求解步骤，此时就需调用线性代数工具包中的初等变换函数. 如下表所示：

	函数调用	对应的初等变换
行变换	mulrow(A, r, expr)	用标量 expr 乘以矩阵 A 的第 r 行
	addrow(Ar1, r2, m)	将矩阵 A 的第 r1 行的 m 倍加到第 r2 行上
	swaprow(A, r1, r2)	互换矩阵 A 的第 r1 行和第 r2 行
列变换	mulcol(A, c, expr)	用标量 expr 乘以矩阵 A 的第 c 列
	addcol(A, c1, c2)	将矩阵 A 的第 c1 列的 m 倍加到第 c2 列上
	swapcol(A, c1, c2)	互换矩阵 A 的第 c1 列和第 c2 列

除了这些初等变换外，**Maple** 在 **linalg** 工具包中还提供了一些矩阵的形状变换，可以在利用初等变换解决问题时起到辅助作用，我们也将它们一并列出.

函数调用	所作的变换或操作
contat(A, B, …);	将矩阵 A, B, …在水平方向上合并成一个矩阵
stackmatrix(A, B, …)	将矩阵 A, B, …在竖直方向上合并成一个矩阵
row(A, i)	取矩阵 A 的第 i 行
col(A, i)	取矩阵 A 的第 i 列
row(A, i .. k)	取矩阵 A 的第 i 到 k 行
col(A, i .. k)	取矩阵 A 的第 i 到 k 列
delrows(A, i .. k)	删除矩阵 A 中 i 到 k 行剩下的子矩阵
delcols(A, i .. k)	删除矩阵 A 中 i 到 k 列剩下的子矩阵
extend(A, m, n, x)	扩展矩阵 m 行 n 列并用 x 填充
submatrix(A, m .. n, r .. s)	取矩阵 A 的 m 到 n 行、r 到 s 列组成的子矩阵

例：利用矩阵的初等变换判断矩阵 A 是否可逆，若可逆，求 A^{-1}

$$A=\begin{pmatrix}-2 & 1 & 3\\ 0 & -1 & 1\\ 1 & 2 & 0\end{pmatrix}$$

```
> with(linalg):
A:=matrix([[-2, 1, 3], [0, -1, 1], [1, 2, 0]]);
```

$$A := \begin{bmatrix}-2 & 1 & 3\\ 0 & -1 & 1\\ 1 & 2 & 0\end{bmatrix}$$

```
> AI:=concat(A, array(identity, 1..3, 1..3));
```

$$AI := \begin{bmatrix}-2 & 1 & 3 & 1 & 0 & 0\\ 0 & -1 & 1 & 0 & 1 & 0\\ 1 & 2 & 0 & 0 & 0 & 1\end{bmatrix}$$

```
> mulrow(AI, 1, -1/2);
```

$$\begin{bmatrix} 1 & \frac{-1}{2} & \frac{-3}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 \\ 1 & 2 & 0 & 0 & 0 & 1 \end{bmatrix}$$

> **addrow(%,1, 3, -1);**

$$\begin{bmatrix} 1 & \frac{-1}{2} & \frac{-3}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 \\ 0 & \frac{5}{2} & \frac{3}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

> **mulrow(%, 2, -1);**

$$\begin{bmatrix} 1 & \frac{-1}{2} & \frac{-3}{2} & \frac{-1}{2} & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & \frac{5}{2} & \frac{3}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

> **addrow(%, 2, 1, 1/2);**

$$\begin{bmatrix} 1 & 0 & -2 & \frac{-1}{2} & \frac{-1}{2} & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & \frac{5}{2} & \frac{3}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix}$$

> **addrow(%, 2, 3, -5/2);**

$$\begin{bmatrix} 1 & 0 & -2 & \frac{-1}{2} & \frac{-1}{2} & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 4 & \frac{1}{2} & \frac{5}{2} & 1 \end{bmatrix}$$

> **mulrow(%, 3, 1/4);**

$$\begin{bmatrix} 1 & 0 & -2 & \frac{-1}{2} & \frac{-1}{2} & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

> **addrow(%, 3, 1, 2);**

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-1}{4} & \frac{3}{4} & \frac{1}{2} \\ 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

> **addrow(%, 3, 2, 1);**

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-1}{4} & \frac{3}{4} & \frac{1}{2} \\ 0 & 1 & 0 & \frac{1}{8} & \frac{-3}{8} & \frac{1}{4} \\ 0 & 0 & 1 & \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

> **AA:=submatrix(%, 1..3, 4..6);**

$$AA := \begin{bmatrix} \frac{-1}{4} & \frac{3}{4} & \frac{1}{2} \\ \frac{1}{8} & \frac{-3}{8} & \frac{1}{4} \\ \frac{1}{8} & \frac{5}{8} & \frac{1}{4} \end{bmatrix}$$

上述判断矩阵可逆并求逆矩阵的方法即高斯—约当消去法，从例中可以看出，按高斯—约当消去法步骤逐一在 Maple 下操作即可实现. 在这里，数学基础显然是最重要的. 当然，上述例子可通过下述两句命令即可完成：

> **det(A);**

若 det(A)不为 0 执行下句即可求出 A 的逆矩阵：

> **inverse(A);**

3.2 线性方程组的解

线性方程组求解常常转化为与其等价的矩阵问题来解决的. Maple中可借助函数**genematrix**及高斯消去法函数**gausselim**联合完成：

> **eqns:={x+2*y+3*z=a,8*x+9*y+4*z=b,7*x+6*y+5*z=c};**

$$eqns := \{x + 2\,y + 3\,z = a, 8\,x + 9\,y + 4\,z = b, 7\,x + 6\,y + 5\,z = c\}$$

> **A:=genmatrix(eqns,[x,y,z],'flag');**

$$A := \begin{bmatrix} 1 & 2 & 3 & a \\ 8 & 9 & 4 & b \\ 7 & 6 & 5 & c \end{bmatrix}$$

> **gausselim(%);**

$$\begin{bmatrix} 1 & 2 & 3 & a \\ 0 & -7 & -20 & b-8a \\ 0 & 0 & \frac{48}{7} & c+\frac{15}{7}a-\frac{8}{7}b \end{bmatrix}$$

可以看出，在高斯消去法的结果矩阵中出现了许多分数，这使表达式看起来不美观，而且随着矩阵的增大，分数的分母会越来越大，如果矩阵中含有符号，结果矩阵中将出现分式．为了避免出现上述情况，可采用无分式高斯消去法(fraction-free Gaussian elimination)，相应的命令为 **ffgausselim**：

> **ffgausselim(%%);**

$$\begin{bmatrix} 1 & 2 & 3 & a \\ 0 & -7 & -20 & b-8a \\ 0 & 0 & -48 & -7c-15a+8b \end{bmatrix}$$

在把系数矩阵转化成相应的上三角阵后，就可以用回代(back substitution)的方法求出各个未知数的值—**backsub**：

> **backsub(%);**

$$\left[-\frac{7}{16}a-\frac{1}{6}b+\frac{19}{48}c, \frac{1}{3}b+\frac{1}{4}a-\frac{5}{12}c, \frac{7}{48}c+\frac{5}{16}a-\frac{1}{6}b\right]$$

和回代相对，也有前代(forward substitution)的概念，可用前代函数 **forwardsub** 法语解下三角矩阵．有了这一对函数，我们就可以用 LU 分解来线性方程组了，相应的命令函数为 **LUdecomp**．关于上一问题也可以利用高斯—约当消去法(Gauss-Jordan elimination)，把系数矩阵变换成单位阵直接得到结果：

> **gaussjord(A);**

$$\begin{bmatrix} 1 & 0 & 0 & -\frac{7}{16}a-\frac{1}{6}b+\frac{19}{48}c \\ 0 & 1 & 0 & \frac{1}{3}b+\frac{1}{4}a-\frac{5}{12}c \\ 0 & 0 & 1 & \frac{7}{48}c+\frac{5}{16}a-\frac{1}{6}b \end{bmatrix}$$

上面介绍的实际上都是线性方程求解的中间步骤，事实上我们可以一步到位，不管它到底用的是什么方法，而只要求得最终的结果．这方面，Maple 提供了一个非常好的函数 **linsolve**，这个函数不仅可以用来求解具有唯一解的线性方程组，而且可以求解有无穷多解的方程组并给出通解．试看下面的实验：

例：求解线性方程组：$\begin{cases} x_1+3x_2+3x_3+2x_4=-1 \\ 2x_1+6x_2+9x_3+5x_4=4 \\ -x_1-3x_2+3x_3=13 \end{cases}$

```
> A:=matrix([[1, 3, 3, 2], [2, 6, 9, 5], [-1, -3, 3, 0]]);
```

$$A := \begin{bmatrix} 1 & 3 & 3 & 2 \\ 2 & 6 & 9 & 5 \\ -1 & -3 & 3 & 0 \end{bmatrix}$$

```
> B:=vector([-1, 4, 13]);
```

$$B := [-1, 4, 13]$$

```
> linsolve(A, B);
```

$$[-13 - 3\ \_t_2 + 3\ \_t_1, \_t_2, \_t_1, 6 - 3\ \_t_1]$$

可以看到, Maple 用辅助变量$_t_1$, $_t_2$给出了方程组的通解.

3.3 最小二乘法求解线性方程解

在实际问题中, 由于误差或者其他各方面的原因, 很容易出现无解的方程组. 这样问题的解决则变成求出一个“最优”的近似解. 在线性代数中, 通常采用最小二乘解(least-squares solution). linalg 中对应的函数是 **leastsqrs**, 它有两种调用格式：

leastsqrs(A, b);

leastsqrs(S, v);

其中，A 为矩阵, b 为向量，S 为线性方程组, v 为变量名。

```
> with(linalg):
> S:={c[0]+c[1]+c[2]-3, c[0]+2*c[1]+4*c[2]-10, c[0]-9/10, c[0]-c[1]+c[2]-3};
```

$$S := \{c_0 - c_1 + c_2 - 3, c_0 + 2\,c_1 + 4\,c_2 - 10, c_0 + c_1 + c_2 - 3, c_0 - \frac{9}{10}\}$$

```
> leastsqrs(S,{c[0],c[1],c[2]});
```

$$\{c_1 = \frac{7}{200}, c_0 = \frac{159}{200}, c_2 = \frac{91}{40}\}$$

再如：

```
> A:=matrix(3,2,[0,1,1,1,1,1]);
```

$$A := \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

```
> b:=vector([0,1,-1]);
```

$$b := [0, 1, -1]$$

```
> leastsqrs(A,b);
```

$$[0, 0]$$

对于相容方程组，leastsqrs 命令与 solve 功能一样：

```
> leastsqrs({x+y=2,y+z=3,x+z=3},{x,y,z});
```

$$\{x = 1, y = 1, z = 2\}$$

```
> solve({x+y=2,y+z=3,x+z=3},{x,y,z});
```

$$\{x = 1, y = 1, z = 2\}$$

3.4 正定矩阵

在 Maple 中，判断一个矩阵是否是正(负)定矩阵非常简单，只需要使用命令 **definite** 即可，而且还可求出符号矩阵的正(负)定条件：

```
> with(linalg):
> A := matrix(2, 2, [2,1,1,3]);
```

$$A := \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$$

```
> definite(A, 'positive_def');
```

$$true$$

```
> B:=array(1..2, 1..2, 'symmetric'):
  definite(B, 'negative_semidef');
```

$$B_{1,1} \le 0 \text{ and } -B_{1,1} B_{2,2} + B_{1,2}^{2} \le 0 \text{ and } B_{2,2} \le 0$$

由上述实验可以看出，**definite** 的第 1 个参数是需要判定的矩阵，第 2 个参数是矩阵的正定性，共有 4 种情况：**'positive_def'**(正定)、**'positive_semidef'**(半正定)、**'negative_def'**(负定)、**'negative_semidef'**(半负定). 当判定数值矩阵时，返回布尔值 true/false；判定符号矩阵时，它返回一个布尔表达式，表示正负定的条件.

4 特征值、特征向量和矩阵的相似

4.1 矩阵的相似

在 Maple 中可以利用 **linalg** 中的 **issimilar** 判断两个矩阵是否相似. 命令格式如下：

issimilar(A, B, 'P');

其中 A, B 为方阵, P 为转换矩阵(可选). 如果 A, B 相似，则返回 true，否则返回 false.

```
> with(linalg):
> A:=matrix(2, 2);
```

$$A := \text{array}(1 .. 2, 1 .. 2, [\ ])$$

```
> B:=matrix(2, 2);
```

$$B := \text{array}(1 .. 2, 1 .. 2, [\ ])$$

```
> issimilar(A&*B, B&*A);
```

true

在这个例子中, Maple 作了一定的假设, 由于 A 的行列式是一个符号表达式, Maple 在判定时假设它为非零常数. 不过更多情况下, 这个函数是用在数值矩阵的判定上, 同时也还可以求出转换矩阵 P, P 赋值为满足 A=inverse(P)*B*P 的转换矩阵 P.

```
> with(linalg,matrix,issimilar,eigenvalues,diag):
> A:=matrix(3,3,[1,2,3,4,5,6,7,8,9]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
> B:=diag(eigenvalues(A));
```

$$B := \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{15}{2}+\frac{3}{2}\sqrt{33} & 0 \\ 0 & 0 & \frac{15}{2}-\frac{3}{2}\sqrt{33} \end{bmatrix}$$

```
> issimilar(A,B,P);
```

true

```
> print(P);
```

$$\begin{bmatrix} \frac{1}{6} & \frac{-1}{3} & \frac{1}{6} \\ -\frac{5}{12}+\frac{7}{132}\sqrt{33} & -\frac{1}{6}+\frac{1}{198}\sqrt{33} & \frac{1}{12}-\frac{17}{396}\sqrt{33} \\ \frac{5}{12}+\frac{7}{132}\sqrt{33} & \frac{1}{6}+\frac{1}{198}\sqrt{33} & -\frac{1}{12}-\frac{17}{396}\sqrt{33} \end{bmatrix}$$

```
> map(normal,evalm(P^(-1)&*B&*P));
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

4.2 特征值和特征向量

特征值问题可以说是最常见的线性代数问题了. 在 Maple 中求解特征矩阵(characteristic matrix)和特征多项式(characteristic polynomial)的函数分别是 **charmat** 和 **charpoly**, 求解相应的特征值和特征向量的函数分别是 **eigenvalues(eigenvals)** 和 **eigenvectors**. 命令格式如下:

charmat(A, lambda);

charpoly(A, lambda);

eigenvalues(A);

eigenvectors(A);

下面通过几个实例学习上述 4 个函数的用法:

```
> A:=randmatrix(2,2);
```

$$A := \begin{bmatrix} 17 & 72 \\ -99 & -85 \end{bmatrix}$$

```
> charmat(A,lambda);
```

$$\begin{bmatrix} \lambda - 17 & -72 \\ 99 & \lambda + 85 \end{bmatrix}$$

```
> charpoly(A,lambda);
```

$$\lambda^2 + 68\,\lambda + 5683$$

```
> eigenvalues(A);
```

$$-34 + 3\,I\sqrt{503},\ -34 - 3\,I\sqrt{503}$$

```
> eigenvectors(A);
```

$$\left[-34 + 3\,I\sqrt{503},\ 1,\ \left\{\left[1,\ -\frac{17}{24} + \frac{1}{24}\,I\sqrt{503}\right]\right\}\right],\left[-34 - 3\,I\sqrt{503},\ 1,\ \left\{\left[1,\ -\frac{17}{24} - \frac{1}{24}\,I\sqrt{503}\right]\right\}\right]$$

```
> A:=matrix(3, 3, [1, 2, 2, 2, 1, 2, 2, 2, 1]);
```

$$A := \begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix}$$

```
> charmat(A, lambda);
```

$$\begin{bmatrix} \lambda - 1 & -2 & -2 \\ -2 & \lambda - 1 & -2 \\ -2 & -2 & \lambda - 1 \end{bmatrix}$$

```
> charpoly(A, lambda);
```

$$\lambda^3 - 3\lambda^2 - 9\lambda - 5$$

> **eigenvals(A);**

$$5, -1, -1$$

> **eigenvectors(A);**

$$[-1, 2, \{[-1, 1, 0], [-1, 0, 1]\}], [5, 1, \{[1, 1, 1]\}]$$

可以看到，特征向量函数 **eigenvectors** 在给出特征向量的时候，还给出了特征值及特征值重数. 一般情况下，如果同时需要得到特征值和特征向量用 **eigenvectors** 函数即可. 另外，这个函数还可处理符号矩阵，此时，通常以根式的形式给出结果，如果加上可选参数**'implicit'**，结果就以 **RootOf** 的形式给出.

函数 **eigenvalues** 不仅可以求解普通特征值问题，还可求解广义特征值问题，即求解方程 $\det(\lambda C - A) = 0$ 相应的命令函数为 **eigenvals**(A, C).

由特征值理论，任何一个实对称阵，都可以用求特征值的方法将它对角化，但对于非对称阵或者是复矩阵，就没有这样的保证了. 不过对于任意复矩阵 A，还是可以把它化成相似的 Jordan 标准型，相应的命令函数是 **jordan**，而且还可给出转换矩阵 P.

> **A:=matrix(3, 3, [2, -1, 1, 2, 2, -1, 1, 2, -1]);**

$$A := \begin{bmatrix} 2 & -1 & 1 \\ 2 & 2 & -1 \\ 1 & 2 & -1 \end{bmatrix}$$

> **jordan(A, 'p');**

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

> **print(p);**

$$\begin{bmatrix} 0 & 1 & 1 \\ 3 & 2 & 0 \\ 3 & 1 & 0 \end{bmatrix}$$

在 Maple 中，可以调用函数 **JordanBlock**(λ, k) 构造约当块 $J(\lambda, k)$，利用该命令和 diag 函数连用，就可以生成具有约当标准型的矩阵:

> **JordanBlock(3, 4);**

$$\begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

> **linalg[JordanBlock](x, 5);**

$$\begin{bmatrix} x & 1 & 0 & 0 & 0 \\ 0 & x & 1 & 0 & 0 \\ 0 & 0 & x & 1 & 0 \\ 0 & 0 & 0 & x & 1 \\ 0 & 0 & 0 & 0 & x \end{bmatrix}$$

5 线性空间基本理论

5.1 基本子空间

与矩阵 A 相关的 R^n 的四个子空间分别是行空间、列空间、化零空间和左零空间. 化零空间, 实际上就是齐次线性方程组 $AX=0$ 的解空间, 可利用 linsolve 求得其通解, 再将通解中的辅助变量依次替换成 1, 就可以获得解空间的一组基. 对于左零空间, 即方程组 $A^TX=0$ 的解空间, 亦可用同样的方法得到. 行空间和列空间, 在 linalg 中也有专门的函数 rowspace 和 colspace, 可以获得它们的基和维数. 例举如下：

> **with(linalg):**
> **A:=matrix(3, 2,[2, 0, 3, 4, 0, 5]);**

$$A := \begin{bmatrix} 2 & 0 \\ 3 & 4 \\ 0 & 5 \end{bmatrix}$$

> **rowspace(A, 'dim');**

$$\{[0, 1], [1, 0]\}$$

> **dim;**

$$2$$

> **colspace(A);**

$$\left\{\left[0, 1, \frac{5}{4}\right], \left[1, 0, \frac{-15}{8}\right]\right\}$$

函数 rowspace 和 colspace 的第二个参数是可选参数, 用来返回行空间或列空间的维数, 它必须是一个变量名—可以是一个未赋值的新变量, 也可以是已有值的变量, 但要

加上延迟求值符“'”.

而寻找向量空间的基的函数是 basis:

```
> v1:=vector([1,0,0]):
  v2:=vector([0,1,0]):
  v3:=vector([0,0,1]):
  v4:=vector([1,1,1]):
  basis({v1,v2,v3});
```

$$\{v2, v1, v3\}$$

```
> basis({vector([1,1,1]),vector([2,2,2]),vector([1,-1,1]),vector([2,-2,2]),vector([1, 0, 1]),
  vector([0, 1, 1])});
```

$$\{[1, 1, 1], [0, 1, 1], [2, -2, 2]\}$$

5.2 正交基和 Schmidt 正交化

在欧氏空间中，我们可以定义两个向量的内积(inner product)，在此基础上，我们还可以定义两个向量的夹角. 如向量α、β的夹角θ可以定义为：

$$\theta = \arccos\frac{(\alpha,\beta)}{\|\alpha\|\|\beta\|}$$

在 Maple 的 linalg 工具包中，有相应的函数 innerprod、angel 可以计算向量的内积和夹角. 如：

```
> alpha:=vector([1, 2, -1, 11]);
```

$$\alpha := [1, 2, -1, 11]$$

```
> beta:=vector([2, 3, 1, -1]);
```

$$\beta := [2, 3, 1, -1]$$

```
> innerprod(alpha, beta);
```

$$-4$$

```
> angle(alpha, beta);
```

$$\pi - \arccos\left(\frac{4}{1905}\sqrt{127}\sqrt{15}\right)$$

求三维向量的向量积使用命令 crossprod:

```
> alpha:=vector([1, 2, 1]);
```

$$\alpha := [1, 2, 1]$$

```
> beta:=vector([2, 3, 1]);
```

$$\beta := [2, 3, 1]$$

```
> crossprod(alpha,beta);
```

$$[-1, 1, -1]$$

定义内积为零的向量间正交，此时可以利用 Schmit 正交化方法，由欧氏空间中一组普通基得到两两正交的基. Maple 中的相应函数是 **GramSchmidt**，它的输入参数是由一组向量组成的集合(或有序表)，将给出 Schmidt 正交化后的向量集合(或有序表). 输入的向量必须是线性无关的，否则，结果向量之间也将线性相关. 函数并不对向量进行单位化. 如果需要得到一组正交标准基，还需要用 map 方法对这些向量使用单位化函数 **normalize**.

例：用 Schmidt 正交化方法，由下列向量组构造出一组标准正交向量组：

$$(1,1,-1,-2)^T,\ (5,8,-2,-3)^T,\ (3,9,3,9)^T$$

```
> A:={vector([1, 1,-1, -2]), vector([5, 8, -2, -3]), vector([3, 9, 3, 9])};
```

$$A := \{[5, 8, -2, -3], [3, 9, 3, 9], [1, 1, -1, -2]\}$$

```
> simplify(map(normalize, GramSchmidt(A)));
```

$$\left\{\left[\frac{2}{91}\sqrt{2}\sqrt{91}, -\frac{3}{182}\sqrt{2}\sqrt{91}, -\frac{11}{182}\sqrt{2}\sqrt{91}, \frac{3}{91}\sqrt{2}\sqrt{91}\right],\right.$$
$$\left.\left[\frac{1}{7}\sqrt{7}, \frac{1}{7}\sqrt{7}, -\frac{1}{7}\sqrt{7}, -\frac{2}{7}\sqrt{7}\right], \left[\frac{2}{39}\sqrt{39}, \frac{5}{39}\sqrt{39}, \frac{1}{39}\sqrt{39}, \frac{1}{13}\sqrt{39}\right]\right\}$$

事实上，向量的 **Schmidt** 正交化过程实际上给出了矩阵的 *QR* 分解.