

## Maple linalg package

Mth 351 October 27 2002 Maple 6  
Bent E. Petersen

Assignment 3 - Due Nov 5 - see below

### Introduction

Maple 6 comes with two packages for linear algebra, linalg and LinearAlgebra. In this worksheet we will only explore the linalg package. Many of the functions in the linalg package take optional parameters, or have several forms. We will not investigate every, or even many, possibilities.

Note that Maple commands are polymorphic. Their behavior depends on the number and type of parameters that you feed to them. In some situations you may need to be careful about data types and the number of parameters passed.

### The Assignment

Assignment 3 is included near the end of this worksheet.

```
> restart;  
> with(linalg):  
Warning, the protected names norm and trace have been redefined and unprotected
```

### matrix()

Matrices can be entered by listing their rows (lists are denoted by [ ] in Maple) or by giving the size and then listing the entries in the matrix row-wise. Here I enter the matrix A in both ways.

```
> A:=matrix([[1,2,3,4],[5,6,7,8],[9,0,1,2],[4,4,4,4]]);
```

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

```
> A:=matrix(4,4,[1,2,3,4,5,6,7,8,9,0,1,2,4,4,4,4]);
```

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

Personally I prefer the second method for interactive entry. It is also possible to build a matrix by specifying its columns (as vectors or lists).

### Multiplication of matrices, evalm()

Matrix multiplication is denoted by `&*`. The ampersand serves to mark the multiplication as being noncommutative so that Maple will not make incorrect assumptions when simplifying expressions involving matrix multiplication.

```
> B:=matrix(4,2,[3,2,-4,-5,7,6,-3,-5]);
```

$$B := \begin{bmatrix} 3 & 2 \\ -4 & -5 \\ 7 & 6 \\ -3 & -5 \end{bmatrix}$$

```
> A&*B;
```

$$A \&* B$$

Note that Maple often suppresses output of matrices. To force evaluation we use the `evalm()` function

```
> evalm(A&*B);
```

$$\begin{bmatrix} 4 & -10 \\ 16 & -18 \\ 28 & 14 \\ 12 & -8 \end{bmatrix}$$

### transpose()

```
> B:=matrix(2,3,[1,2,3,4,5,6]);
```

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

```
> transpose(B);
```

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

### Matrices with unspecified entries

We can construct matrices with unspecified entries and then specify the entries later:

```
> C:=matrix(3,4);
```

```

C := array(1 .. 3, 1 .. 4, [ ])
> C[1,2] := -14; C[3,4] := 12;
C1,2 := -14
C3,4 := 12
> evalm(C);

$$\begin{bmatrix} C_{1,1} & -14 & C_{1,3} & C_{1,4} \\ C_{2,1} & C_{2,2} & C_{2,3} & C_{2,4} \\ C_{3,1} & C_{3,2} & C_{3,3} & 12 \end{bmatrix}$$


```

### diag(), BlockDiagonal()

The diag() command produces a diagonal matrix.

```

> diag(1,2,3,4);

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

> B1:=matrix(2,2,[1,2,3,4]); B2:=matrix(2,2,[5,6,7,8]);
B1 :=  $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ 
B2 :=  $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$ 
> diag(B1,B2,-7);

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


```

BlockDiagonal() is a synonym for diag(). It is likely to be used in constructions such as the last one above.

### Creating band matrices - band()

The function call band(b,n) returns an n-by-n band matrix. If b is a list of odd length, n is a positive integer, and the number of entries in b is no more than 2n-1 then a matrix in which all the elements of the i<sup>th</sup> subdiagonal are initialized to b[i], for the first half of b. The elements of the main diagonal of the matrix result are initialized to the middle element of b. The last half of b is used to initialize the superdiagonals.

```
> band([1, 3, 1], 6);
```

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

### extend()

The extend function enlarges a matrix. The function call extend(B,m,n,x) adds m rows and n columns to the matrix B. The new entries are initialized to x or left undefined if the value x is not included in the function call.

```
> B:=matrix(2,3,[1,2,3,4,5,6]);
```

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

```
> C0:=extend(B,2,1,0);
```

$$C0 := \begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> C:=extend(B,2,1);
```

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

### The identity matrix

The expression &\*( ) may be used to designate the identity matrix of unspecified size. The size is determined from the context. For example

```
> evalm( &*( ) + matrix(2,2,[1,2,3,4]) );
```

$$\begin{bmatrix} 2 & 2 \\ 3 & 5 \end{bmatrix}$$

To produce an identity matrix of an explicit size we can use the diag() command in conjunction with the seq() command.

```
> diag( seq(1,k=1..5) );
```

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

The function `band()` is even more convenient for creating the identity matrix

```
> band( [1], 5 );
```

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

Note the identity matrix is implicitly assumed when scalars are added to matrices:

```
> evalm( 3 + matrix(2,2, [1,2,3,4]) );
```

$$\begin{bmatrix} 4 & 2 \\ 3 & 7 \end{bmatrix}$$

### Hilbert matrix, `hilbert()`

The (i,j)th entry in the Hilbert matrix is  $1/(i+j-1)$ .

```
> hilbert(4);
```

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

### Zero matrix

A zero matrix of any size may be easily created by using the sequence command, `seq()`:

```
> matrix(3,5, [seq(0,k=1..3*5)]);
```

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

### Multiplication by zero, scalarmul()

Unfortunately multiplication by zero returns a scalar 0 rather than a zero matrix. There is no problem if the result is added to another matrix of the same size, but there is a potential problem in other contexts.

```
> evalm( 0 * matrix(2,2,[1,2,3,4]) );
```

0

```
> evalm( 2 * matrix(2,2,[1,2,3,4]) );
```

$$\begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

A safe, but inconvenient, way to multiply by scalars is to use the scalarmul() command:

```
> scalarmul( matrix(2,2,[1,2,3,4]), 0 );
```

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

### rank()

The rank() function returns the rank of a matrix.

```
> rank(A);
```

3

### determinant, det()

```
> B:=matrix(4,4);
```

B := array(1 .. 4, 1 .. 4, [ ])

```
> det(B);
```

$$\begin{aligned} & B_{1,1} B_{2,2} B_{3,3} B_{4,4} - B_{1,1} B_{2,2} B_{3,4} B_{4,3} - B_{1,1} B_{3,2} B_{2,3} B_{4,4} + B_{1,1} B_{3,2} B_{2,4} B_{4,3} \\ & + B_{1,1} B_{4,2} B_{2,3} B_{3,4} - B_{1,1} B_{4,2} B_{2,4} B_{3,3} - B_{2,1} B_{1,2} B_{3,3} B_{4,4} + B_{2,1} B_{1,2} B_{3,4} B_{4,3} \\ & + B_{2,1} B_{3,2} B_{1,3} B_{4,4} - B_{2,1} B_{3,2} B_{1,4} B_{4,3} - B_{2,1} B_{4,2} B_{1,3} B_{3,4} + B_{2,1} B_{4,2} B_{1,4} B_{3,3} \\ & + B_{3,1} B_{1,2} B_{2,3} B_{4,4} - B_{3,1} B_{1,2} B_{2,4} B_{4,3} - B_{3,1} B_{2,2} B_{1,3} B_{4,4} + B_{3,1} B_{2,2} B_{1,4} B_{4,3} \\ & + B_{3,1} B_{4,2} B_{1,3} B_{2,4} - B_{3,1} B_{4,2} B_{1,4} B_{2,3} - B_{4,1} B_{1,2} B_{2,3} B_{3,4} + B_{4,1} B_{1,2} B_{2,4} B_{3,3} \\ & + B_{4,1} B_{2,2} B_{1,3} B_{3,4} - B_{4,1} B_{2,2} B_{1,4} B_{3,3} - B_{4,1} B_{3,2} B_{1,3} B_{2,4} + B_{4,1} B_{3,2} B_{1,4} B_{2,3} \end{aligned}$$

```
> C:=matrix(3,3,[1,2,2,6,5,2,7,8,9]);
```

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

```
> det(C);
```

-25

### vector()

A vector in Maple is an ordered n-tuple. There is no concept of rows or columns in connection with a vector; We can have row or column vectors if we wish, but they are matrices and so have two subscripts.

```
> a:=vector([1,2,3,4]);
```

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

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

$$b := [1 \quad 2 \quad 3 \quad 4]$$

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

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

We can convert vectors to matrices and 1-by-n matrices to vectors

```
> convert(a,matrix);
```

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

```
> convert(c,vector);
```

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

### rowspace(), colspace()

The rowspace() function returns a basis for the rowspace of a matrix. The basis is returned as a set of vectors, that is, unordered.

```
> basis:=rowspace(A);
```

$$basis := \{ [1, 0, 0, 0], [0, 1, 0, -1], [0, 0, 1, 2] \}$$

The quickest way to obtain an ordered basis is to convert to a list. However, the order may not be what you want.

```
> convert(basis,list);  
[[1, 0, 0, 0], [0, 1, 0, -1], [0, 0, 1, 2]]
```

The `colspace()` function behaves similarly

```
> colspace(A);  
{[1, 0, 0, -1], [0, 1, 0, 1], [0, 0, 1, 0]}
```

Note both `rowspace()` and `colspace()` return vectors.

### Augmentation - `augment()`, `concat()` and `stackmatrix()`

We can augment an m-by-n matrix by an m-by-q matrix or by a vector or list. The matrices are simply concatenated horizontally. The vector or list is added as a column. The function `concat()` is a synonym for `augment()`. Recall our matrix A

```
> evalm(A);  

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

```

```
> augment([1,2,3,4],A,[5,6,7,8]);  

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

```

```
> augment(A,A^2);  

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 54 & 30 & 36 & 42 \\ 5 & 6 & 7 & 8 & 130 & 78 & 96 & 114 \\ 9 & 0 & 1 & 2 & 26 & 26 & 36 & 46 \\ 4 & 4 & 4 & 4 & 76 & 48 & 60 & 72 \end{bmatrix}$$

```

The `stackmatrix()` command concatenates vertically. A vector or a list is added as a row.

```
> stackmatrix([0,0,1,2],A,[1,2,0,0]);
```

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

```
> stackmatrix(A,A^2);
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 0 & 1 & 2 \\ 4 & 4 & 4 & 4 \\ 54 & 30 & 36 & 42 \\ 130 & 78 & 96 & 114 \\ 26 & 26 & 36 & 46 \\ 76 & 48 & 60 & 72 \end{bmatrix}$$

### Random stuff

```
> randmatrix(3,5);
```

$$\begin{bmatrix} -85 & -55 & -37 & -35 & 97 \\ 50 & 79 & 56 & 49 & 63 \\ 57 & -59 & 45 & -8 & -93 \end{bmatrix}$$

```
> randvector(8);
```

[92, 43, -62, 77, 66, 54, -5, 99]

```
> randmatrix(4,4,symmetric);
```

$$\begin{bmatrix} -61 & -50 & -18 & -62 \\ -50 & -12 & 31 & 1 \\ -18 & 31 & -26 & -47 \\ -62 & 1 & -47 & -91 \end{bmatrix}$$

```
> randmatrix(4,4,symmetric,entries=rand(0..8));
```

$$\begin{bmatrix} 5 & 0 & 8 & 2 \\ 0 & 5 & 0 & 0 \\ 8 & 0 & 6 & 8 \\ 2 & 0 & 8 & 4 \end{bmatrix}$$

### Row reduction - rref(), gaussjord()

The function rref() returns the (completely) row reduced echelon form of a matrix. Note gaussjord() is a synonym for rref().

Recall our sample matrix A

```
> evalm(A);
```

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

```
> rref(A);
```

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

### Gauss elimination - gausselim()

The function gausselim() does Gauss elimination, or row reduction, but stops when a triangular matrix is achieved. It does not go all the way to the fully row reduced row echelon form.

```
> B:=matrix(3,4,[x,x,3,4,x,y,z,8,y+1,y,3,4]);
```

$$B := \begin{bmatrix} x & x & 3 & 4 \\ x & y & z & 8 \\ y+1 & y & 3 & 4 \end{bmatrix}$$

```
> gausselim(B);
```

$$\begin{bmatrix} x & x & 3 & 4 \\ 0 & -1 & 3 \frac{x-y-1}{x} & 4 \frac{x-y-1}{x} \\ 0 & 0 & -\frac{-zx-6yx+3y^2+3y+3x^2}{x} & -4 \frac{-2x-2yx+y^2+y+x^2}{x} \end{bmatrix}$$

### Fraction free row reduction - ffgausselim()

The function ffgausselim does Gauss elimination, or row reduction, without introducing fractions

```
> ffgausselim(B);
```

$$\begin{bmatrix} x & x & 3 & 4 \\ 0 & -x & 3x-3y-3 & 4x-4y-4 \\ 0 & 0 & -zx-6yx+3y^2+3y+3x^2 & -8x-8yx+4y^2+4y+4x^2 \end{bmatrix}$$

### Hermite normal form for integer matrices - ihermite()

The function call ihermite(B) row reduces the matrix B over the integers. The Hermite normal form is obtained by doing only integer elementary row operations. This includes interchanging rows, multiplying a row by -1, and adding an integral multiple of one row to another. The goal is to produce

an upper triangular matrix with as many zeros in each pivotal column as possible.

```
> C1:=matrix(3,4,[13,11,3,0,7,3,5,0,11,2,1,9]);
```

$$C1 := \begin{bmatrix} 13 & 11 & 3 & 0 \\ 7 & 3 & 5 & 0 \\ 11 & 2 & 1 & 9 \end{bmatrix}$$

```
> ihermite(C1);
```

$$\begin{bmatrix} 1 & 14 & 15 & -27 \\ 0 & 19 & 8 & -27 \\ 0 & 0 & 20 & -18 \end{bmatrix}$$

The function call ihermite(B,U) in addition returns an integer matrix U such that  $U \&* B = \text{ihermite}(B)$ .

```
> C2:=matrix(3,3,[1,11,3,5,3,5,1,1,1]);
```

$$C2 := \begin{bmatrix} 1 & 11 & 3 \\ 5 & 3 & 5 \\ 1 & 1 & 1 \end{bmatrix}$$

```
> ihermite(C2,U);
```

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

```
> evalm(U);
```

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 5 \\ 1 & 5 & -26 \end{bmatrix}$$

```
> evalm( U &* C2 );
```

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

```
> C3:=matrix(3,3,[2,2,1,2,2,2,0,0,2]);
```

$$C3 := \begin{bmatrix} 2 & 2 & 1 \\ 2 & 2 & 2 \\ 0 & 0 & 2 \end{bmatrix}$$

```
> ihermite(C3);
```

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

### LU decomposition - LUdecomp()

The function LUdecomp() returns only the U in the LU decomposition. To get the permutation matrix

P and the lower triangular matrix L we have to specify names for them in the function call.

```
> B:=hilbert(4);
```

$$B := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

```
> LUdecomp(B, P='Ph', L='Lh', U='Uh');
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ 0 & \frac{1}{12} & \frac{1}{12} & \frac{3}{40} \\ 0 & 0 & \frac{1}{180} & \frac{1}{120} \\ 0 & 0 & 0 & \frac{1}{2800} \end{bmatrix}$$

```
> evalm(Ph); evalm(Lh); evalm(Uh);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{3} & 1 & 1 & 0 \\ \frac{1}{4} & \frac{9}{10} & \frac{3}{2} & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ 0 & \frac{1}{12} & \frac{1}{12} & \frac{3}{40} \\ 0 & 0 & \frac{1}{180} & \frac{1}{120} \\ 0 & 0 & 0 & \frac{1}{2800} \end{bmatrix}$$

> `evalm(Lh&*Uh);`

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

We see it checks,  $B = LU$  and  $P$  is the identity.

### Cholesky decomposition - `cholesky()`, `definite()`

Here is an example of a positive definite matrix

> `B:=hilbert(4);`

$$B := \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

> `definite(B,positive_def);`

*true*

In the Cholesky decomposition we have a lower triangular matrix  $C$  such that  $B = C \&* \text{transpose}(C)$ .

> `C:=cholesky(B);`

$$C := \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{6}\sqrt{3} & 0 & 0 \\ \frac{1}{3} & \frac{1}{6}\sqrt{3} & \frac{1}{30}\sqrt{5} & 0 \\ \frac{1}{4} & \frac{3}{20}\sqrt{3} & \frac{1}{20}\sqrt{5} & \frac{1}{140}\sqrt{7} \end{bmatrix}$$

Let's verify that it works -

```
> evalm( C &* transpose(C) );
```

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

Sure enough!

We can produce a positive definite matrix from any nonsingular matrix by multiplying by the transpose (check this):

```
> B0:=matrix(3,3,[1,2,2,3,3,3,2,2,1]); det(B0);
```

$$B0 := \begin{bmatrix} 1 & 2 & 2 \\ 3 & 3 & 3 \\ 2 & 2 & 1 \end{bmatrix}$$

3

```
> B:= evalm(transpose(B0) &* B0);
```

$$B := \begin{bmatrix} 14 & 15 & 13 \\ 15 & 17 & 15 \\ 13 & 15 & 14 \end{bmatrix}$$

```
> C:=cholesky(B);
```

$$C := \begin{bmatrix} \sqrt{14} & 0 & 0 \\ \frac{15}{14}\sqrt{14} & \frac{1}{14}\sqrt{182} & 0 \\ \frac{13}{14}\sqrt{14} & \frac{15}{182}\sqrt{182} & \frac{3}{13}\sqrt{13} \end{bmatrix}$$

## QR decomposition - QRdecomp()

The function QRdecomp() returns only the upper triangular matrix R in the factorization  $A = QR$ . To get the orthogonal matrix Q we have to pass a name for it in the function call.

Recall our sample matrix A

```
> evalm(A);
```

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

```
> R:=QRdecomp(A,Q='QA');
```

$$R := \begin{bmatrix} \sqrt{123} & \frac{16}{41}\sqrt{123} & \frac{21}{41}\sqrt{123} & \frac{26}{41}\sqrt{123} \\ 0 & \frac{2}{41}\sqrt{15662} & \frac{404}{7831}\sqrt{15662} & \frac{426}{7831}\sqrt{15662} \\ 0 & 0 & \frac{10}{191}\sqrt{382} & \frac{20}{191}\sqrt{382} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> evalm(QA);
```

$$\begin{bmatrix} \frac{1}{123}\sqrt{123} & \frac{33}{15662}\sqrt{15662} & \frac{15}{382}\sqrt{382} & \frac{1}{3}\sqrt{3} \\ \frac{5}{123}\sqrt{123} & \frac{83}{15662}\sqrt{15662} & \frac{3}{382}\sqrt{382} & -\frac{1}{3}\sqrt{3} \\ \frac{3}{41}\sqrt{123} & -\frac{36}{7831}\sqrt{15662} & \frac{1}{191}\sqrt{382} & 0 \\ \frac{4}{123}\sqrt{123} & \frac{25}{7831}\sqrt{15662} & -\frac{6}{191}\sqrt{382} & \frac{1}{3}\sqrt{3} \end{bmatrix}$$

## Linear equations - linsolve()

To solve the linear equation  $Ax=b$  we use linsolve(A,b). If b is a vector then the solution x is returned as a vector, that is, an array with one index. If we express the inhomogeneous term b as a column, that is, an m-by-1 matrix then the solution x is returned as a one column matrix, which is really inconvenient.

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

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

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

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

```
> soln:=linsolve(A,b);
```

$$\text{soln} := \left[ -1 - t_1, -t_1, \frac{9}{2}, -3 \right]$$

Maple returns a list of values. The underscore t sub 1 is a parameter. You assign it arbitrary values to obtain all the solutions. If you find the expression difficult to read you can always substitute something more agreeable, for example

```
> soln:=linsolve(A,b,rnk,s);
```

$$\text{soln} := \left[ -1 - s_1, s_1, \frac{9}{2}, -3 \right]$$

Note the parameter name has to appear as the fourth variable, so we had to insert a third variable, `rnk`. It turns out that `linsolve` stuffs the rank of `A` into the third variable. Thus

```
> rnk;
```

3

One can pick out the individual components easily

```
> soln[1]; soln[2]; soln[3]; soln[4];
```

$-1 - s_1$

$s_1$

$\frac{9}{2}$

-3

We can also solve directly by row reduction of course -

```
> M:=augment(A,b); R:=rref(M);
```

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

$$R := \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & \frac{9}{2} \\ 0 & 0 & 0 & 1 & -3 \end{bmatrix}$$

Here we can read the solution easily. Moreover, we get the canonical form of the solution, with the free variable(s) as the parameter. This is not always the form that Maple returns (though in the present example it is).

```
> x[1] := R[1,5] - R[1,2]*s; x[2] := s; x[3] := R[2,5]; x[4] := R[3,5];
```

$$x_1 := -1 - s$$

$$x_2 := s$$

$$x_3 := \frac{9}{2}$$

$$x_4 := -3$$

### Least squares solution - leastsqr()

The function call leastsqr(A,b) returns the solutions which minimize the Euclidean norm of Ax-b.

First here is an example that has an actual solution:

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

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

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

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

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

$$\left[ -1 - t_1, -t_1, \frac{9}{2}, -3 \right]$$

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

$$\left[ -1 - t_1, -t_1, \frac{9}{2}, -3 \right]$$

Now here is an example which has no actual solution. Note Maple indicates the lack of a solution by returning the empty set.

```
> c := vector([1,2,3,4]);
```

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

```
[ > linsolve(transpose(A), c);  
> leastsqrs(transpose(A), c);
```

$$\left[ \frac{5}{2}, -2, 1 \right]$$

### Row reduction by hand - addrow(), mulrow(), swaprow()

It may be hard to imagine, but possibly you may wish to do elementary row operations by hand. There are also analogous column operations: addcol(), mulcol() and swapcol().

```
[ > evalm(M);
```

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

Add -3 times row 1 to row 2

```
[ > M1:=addrow(M, 1, 2, -3);
```

$$M1 := \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & -2 & -2 & -3 \\ 5 & 5 & 6 & 7 & 1 \end{bmatrix}$$

Add -5 times row 1 to row 3

```
[ > M2:=addrow(M1, 1, 3, -5);
```

$$M2 := \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & -2 & -2 & -3 \\ 0 & 0 & -4 & -3 & -9 \end{bmatrix}$$

Multiply row 2 by -1/2

```
[ > M3:=mulrow(M2, 2, -1/2);
```

$$M3 := \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & \frac{3}{2} \\ 0 & 0 & -4 & -3 & -9 \end{bmatrix}$$

At this point we should add -4 times row 2 to row 3, but let's suppose we didn't notice and instead decided to swap row 2 and row 3

```
[ > M4:=swaprow(M3, 2, 3);
```

$$M4 := \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & -4 & -3 & -9 \\ 0 & 0 & 1 & 1 & \frac{3}{2} \end{bmatrix}$$

> `M5:=mulrow(M4,2,-1/4);`

$$M5 := \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & \frac{3}{4} & \frac{9}{4} \\ 0 & 0 & 1 & 1 & \frac{3}{2} \end{bmatrix}$$

> `M6:=addrow(M5,2,3,-1);`

$$M6 := \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & \frac{3}{4} & \frac{9}{4} \\ 0 & 0 & 0 & \frac{1}{4} & \frac{-3}{4} \end{bmatrix}$$

> `M7:=mulrow(M6,3,4);`

$$M7 := \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & \frac{3}{4} & \frac{9}{4} \\ 0 & 0 & 0 & 1 & -3 \end{bmatrix}$$

> `M8:=addrow(M7,3,1,-2);`

$$M8 := \begin{bmatrix} 1 & 1 & 2 & 0 & 8 \\ 0 & 0 & 1 & \frac{3}{4} & \frac{9}{4} \\ 0 & 0 & 0 & 1 & -3 \end{bmatrix}$$

> `M9:=addrow(M8,3,2,-3/4);`

$$M9 := \begin{bmatrix} 1 & 1 & 2 & 0 & 8 \\ 0 & 0 & 1 & 0 & \frac{9}{2} \\ 0 & 0 & 0 & 1 & -3 \end{bmatrix}$$

> `M10:=addrow(M9,2,1,-2);`

$$M10 := \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & \frac{9}{2} \\ 0 & 0 & 0 & 1 & -3 \end{bmatrix}$$

Let's check our work

> `rref(M);`

$$\begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & \frac{9}{2} \\ 0 & 0 & 0 & 1 & -3 \end{bmatrix}$$

### More row reduction - pivot()

The function call `pivot(A, i, j)` will add multiples of the *i*th row to every other row in the matrix, with the result that the *j*th column of the matrix will be all zeros, except for the (*i*,*j*)th element. The function call `pivot(A, i, j, r..s)` acts like `pivot(A, i, j)` except that only rows *r* through *s* are set to zero in the *j*th column (except row *i* if it is in the range).

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

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

```
> pivot(A,2,3);
```

$$\begin{bmatrix} -\frac{1}{3} & 0 & 0 & \frac{2}{3} \\ 2 & 3 & 3 & 5 \\ \frac{16}{3} & 3 & 0 & \frac{1}{3} \end{bmatrix}$$

```
> pivot(A,2,3,1..2);
```

$$\begin{bmatrix} -\frac{1}{3} & 0 & 0 & \frac{2}{3} \\ 2 & 3 & 3 & 5 \\ 6 & 4 & 1 & 2 \end{bmatrix}$$

### GramSchmidt()

The function `GramSchmidt()` returns a set of orthogonal vectors with the same span as a given list or set of linearly independent vectors. The vectors are normalized if that option is specified.

```
> v1:=vector([1,2,3]);
```

$$v1 := [1, 2, 3]$$

```
> v2:=vector([1,2,0]);
```

$$v2 := [1, 2, 0]$$

```
> v3:=vector([1,0,0]);
```

$$v3 := [1, 0, 0]$$

```
> GramSchmidt([v1,v2,v3],normalized);
```

$$\left[ \left[ \frac{1}{14}\sqrt{14}, \frac{1}{7}\sqrt{14}, \frac{3}{14}\sqrt{14} \right], \left[ \frac{1}{70}\sqrt{45}\sqrt{14}, \frac{1}{35}\sqrt{45}\sqrt{14}, -\frac{1}{42}\sqrt{45}\sqrt{14} \right], \right. \\ \left. \left[ \frac{1}{5}\sqrt{4}\sqrt{5}, -\frac{1}{10}\sqrt{4}\sqrt{5}, 0 \right] \right]$$

> `GramSchmidt({v1,v2,v3}, normalized);`

$$\left\{ \left[ \frac{1}{5}\sqrt{4}\sqrt{5}, -\frac{1}{10}\sqrt{4}\sqrt{5}, 0 \right], \left[ \frac{1}{5}\sqrt{5}, \frac{2}{5}\sqrt{5}, 0 \right], [0, 0, 1] \right\}$$

In the case that we pass a list of vectors then  $k$ th vector returned is a linear combination of  $v_1, \dots, v_k$  for each  $k$ . In the case of a set there is no guarantee concerning the order.

### adj(), adjoint()

The function `adj()` returns the classical adjoint. It is characterized by  $A \&* \text{adj}(A) = \det(A) \&* I$

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

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

> `B:=adj(A);`

$$B := \begin{bmatrix} 3 & -1 & -4 \\ -4 & -4 & 8 \\ -1 & 3 & -4 \end{bmatrix}$$

> `evalm(A &* B);`

$$\begin{bmatrix} -8 & 0 & 0 \\ 0 & -8 & 0 \\ 0 & 0 & -8 \end{bmatrix}$$

### inverse()

This function computes the inverse of a nonsingular square matrix.

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

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

> `inverse(A);`

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

```
> B:=matrix(3,3,[x,y,z,3,2,1,2,1,2]);
```

$$B := \begin{bmatrix} x & y & z \\ 3 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$$

```
> inverse(B);
```

$$\begin{bmatrix} 3 \frac{1}{3x-4y-z} & -\frac{2y-z}{3x-4y-z} & \frac{y-2z}{3x-4y-z} \\ -4 \frac{1}{3x-4y-z} & 2 \frac{x-z}{3x-4y-z} & -\frac{x-3z}{3x-4y-z} \\ -\frac{1}{3x-4y-z} & -\frac{x-2y}{3x-4y-z} & \frac{2x-3y}{3x-4y-z} \end{bmatrix}$$

### angle()

The angle() function returns the angle between two vectors.

```
> angle([0,1,0],[0,0,1]);
```

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

```
> angle([3,4,0,12],[1,0,2,2]); simplify(%);
```

$$\arccos\left(\frac{3}{169}\sqrt{169}\sqrt{9}\right)$$

$$\arccos\left(\frac{9}{13}\right)$$

### exponential()

The function exponential() computes the exponential of a square matrix. The exponential function can only return a symbolic answer if the eigenvalues of A can be found. To get a floating-point approximation, use at least one floating-point entry in A.

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

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

```
> exponential(B*t);
```

$$\begin{bmatrix} \frac{3}{5}e^{(-t)} + \frac{2}{5}e^{(4t)} & \frac{2}{5}e^{(4t)} - \frac{2}{5}e^{(-t)} \\ \frac{3}{5}e^{(4t)} - \frac{3}{5}e^{(-t)} & \frac{2}{5}e^{(-t)} + \frac{3}{5}e^{(4t)} \end{bmatrix}$$

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

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

```
> exponential(C*t);
```

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

```
> E:=matrix(2,2,[0,-b,b,0]);
```

$$E := \begin{bmatrix} 0 & -b \\ b & 0 \end{bmatrix}$$

```
> exponential(E*t); simplify(%);
```

$$\begin{bmatrix} \frac{1}{2}e^{(-bt)} + \frac{1}{2}e^{(bt)} & \frac{1}{2}Ie^{(bt)} - \frac{1}{2}Ie^{(-bt)} \\ -\frac{1}{2}Ie^{(bt)} + \frac{1}{2}Ie^{(-bt)} & \frac{1}{2}e^{(-bt)} + \frac{1}{2}e^{(bt)} \end{bmatrix}$$
$$\begin{bmatrix} \cos(tb) & -\sin(tb) \\ \sin(tb) & \cos(tb) \end{bmatrix}$$

```
> F:=matrix(2,2,[a,b,0,2]);
```

$$F := \begin{bmatrix} a & b \\ 0 & 2 \end{bmatrix}$$

```
> exponential(F*t);
```

$$\begin{bmatrix} e^{(ta)} & -\frac{b(e^{(2t)} - e^{(ta)})}{-2+a} \\ 0 & e^{(2t)} \end{bmatrix}$$

The characteristic polynomial - charpoly()

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

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

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

$$\lambda^3 - 6\lambda^2$$

The minimal polynomial - minpoly()

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

$$-6\lambda + \lambda^2$$

### Norms - norm()

```
> A:=matrix(3,3,[1,2,3,2,2,-3,-1,0,4]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 2 & 2 & -3 \\ -1 & 0 & 4 \end{bmatrix}$$

```
> norm(A,1);
```

10

```
> norm(A,2): evalf(%);
```

5.976940421

```
> norm(A,infinity);
```

7

```
> norm(A,frobenius): evalf(%);
```

6.928203232

### Eigenvalues - eigenvals(), eigenvalues()

The functions eigenvals() and eigenvalues() return the eigenvalues of a matrix. It is not often when we can obtain the eigenvalues exactly and even if we can the expressions may be excessively complicated. We can force Maple to produce a floating point approximation by making sure we have at least one floating point number in the matrix.

```
> A:=hilbert(3): A[1,1]:=1.0: evalm(A);
```

$$\begin{bmatrix} 1.0 & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{bmatrix}$$

```
> eigenvals(A);
```

.002687340333, .1223270659, 1.408318927

### Eigenvectors - eigenvects(), eigenvectors()

```
> eigenvects(A);
```

[.1223270659, 1, {[.5474484307, -.5282902357, -.6490066582]}],

```
[.0026873409, 1, {[.1276593292, -.7137468847, .6886715323]}],  
[1.408318929, 1, { [.8270449269, .4598639046, .3232984350]}]
```

Note the format of the response is [ **eigenvalue, algebraic multiplicity, set of eigenvectors** ].

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

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

```
> eigenvects(B);
```

```
[0, 2, {[ -2, 1, 0], [ -3, 0, 1 ]}], [6, 1, {[ 1, 1, 1 ]}]
```

Note here that the eigenvalue 0 has algebraic multiplicity 2.

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

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

```
> eigenvects(C);
```

```
[0, 3, {[ 0, 0, 1], [ 0, 1, 0 ]}]
```

Here 0 is an eigenvalue of algebraic multiplicity 3, but geometric multiplicity only 2 (just count the number of eigenvectors).

The condition number - cond()

```
> A:=matrix(3,3,[1.0,2,2,4,2,1,0,1,0]);
```

$$A := \begin{bmatrix} 1.0 & 2 & 2 \\ 4 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

```
> cond(A);
```

```
11.00000000
```

The default norm is the infinity norm (maximum row sum) but we can specify other norms.

```
> cond(A,1);
```

```
10.71428572
```

```
> cond(A,2);
```

```
7.240762599
```

```
[ > cond(A, infinity);
                                     11.00000000
[ > cond(A, frobenius);
                                     8.380005357
```

Hilbert matrices tend to have large condition numbers.

```
[ > H:=hilbert(4): H[1,1]:=1.0: cond(H,2);
                                     15513.73852
[ > H:=hilbert(6): H[1,1]:=1.0: cond(H,2);
                                     .1494952999 108
[ > H:=hilbert(8): H[1,1]:=1.0: cond(H,2);
                                     .1541712975 1011
[ > H:=hilbert(10): H[1,1]:=1.0: cond(H,2);
                                     .3148373215 1014
```

Note I forced floating point calculations above to avoid meaningless accuracy in calculating the condition number.

## Other functions

There are quite a number of other functions that I have not discussed above. To learn about any one of them simply remove the comment character (#) and execute the command in the list below.

```
[ > # ?JordanBlock
[ > # ?Wronskian
[ > # ?backsub
[ > # ?basis
[ > # ?bezout
[ > # ?blockmatrix
[ > # ?charmat
[ > # ?col
[ > # ?coldim
[ > # ?colspace
[ > # ?colspan
[ > # ?companion
[ > # ?copyinto
[ > # ?crossprod
[ > # ?curl
[ > # ?delcols
[ > # ?delrows
```

```
[ > # ?diverge
[ > # ?dotprod
[ > # ?entermatrix
[ > # ?equal
[ > # ?fibonacci
[ > # ?forwardsub
[ > # ?frobenius
[ > # ?geneqns
[ > # ?genmatrix
[ > # ?grad
[ > # ?hadamard
[ > # ?hermite
[ > # ?hessian
[ > # ?htranspose
[ > # ?indexfunc
[ > # ?innerprod
[ > # ?intbasis
[ > # ?ismith
[ > # ?issimilar
[ > # ?iszero
[ > # ?jacobian
[ > # ?jordan
[ > # ?kernel
[ > # ?laplacian
[ > # ?matadd
[ > # ?minor
[ > # ?multiply
[ > # ?normalize
[ > # ?nullspace
[ > # ?orthog
[ > # ?permanent
[ > # ?potential
[ > # ?ratform
[ > # ?row
[ > # ?rowdim
[ > # ?rowspan
[ > # ?singularvals
[ > # ?smith
[ > # ?submatrix
[ > # ?subvector
[ > # ?sumbasis
[ > # ?sylvester
[ > # ?toeplitz
[ > # ?trace
[
```

```
[ > # ?vandermonde
[ > # ?vecpotent
[ > # ?vectdim
[ > # ?wronskian
```

### Assignment 3 - Due Nov 5, 2020

Try to do these problems with Maple even if you can do some of them with Matlab, or some other way.

#### Problem 1:

Compute the LU factorization of  $L1$  & \*  $U1$  where

```
> L1:=matrix(3,3,[2,0,0,1,2,0,3,2,1]);
```

$$L1 := \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 3 & 2 & 1 \end{bmatrix}$$

```
> U1:=matrix(3,3,[2,3,1,0,3,2,0,0,1]);
```

$$U1 := \begin{bmatrix} 2 & 3 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

#### Problem 2

The matrix B given by

```
> B:=band([-1,2,-1],8);
```

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

is positive definite, as we see from Maple's built-in test

```
> definite(B,positive_def);
```

*true*

Thus the eigenvalues should all be real and positive, yet we have

```
> evalf(eigenvals(B));
1., 3., 3.532088886 - .1 109 I, .120614758 + 0. I, 2.347296354 + 0. I, 3.879385242 + 0. I,
.4679111145 + .1732050808 109 I, 1.652703646 - .1732050808 109 I
```

Some of these numbers are complex! What's happening here? Find the Cholesky decomposition of B. Verify it.

### Problem 3

The matrix A given by

```
> A:=matrix(3,3,[x-y,y,z,0,1,z^2,0,0,z]);
```

$$A := \begin{bmatrix} x-y & y & z \\ 0 & 1 & z^2 \\ 0 & 0 & z \end{bmatrix}$$

appears to have the identity as its row reduced form

```
> rref(A);
```

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

but if we set x equal to y first we get a different result (try it). Why?

### Problem 4

Find the cubic polynomial  $p(t) = x_1 t^3 + x_2 t^2 + x_3 t + x_4$  which passes through the points (1,2), (3,4), (5,1) and (7,-1) by setting up, and solving, a system of linear equations for the coefficients  $x_1$ ,  $x_2$ ,  $x_3$  and  $x_4$ .

### Problem 5

Set Digits equal to 4 and compute the inverse of hilbert(6) exactly and as floating point. Comment on what you observe. Compute the condition number of hilbert(6). What is the connection with your observation. (Don't forget to set Digits back to 10 when you are done.).

### Problem 6

Consider the 4-by-4 Vandermonde matrix

```
> V:=vandermonde([w,x,y,z]);
```

$$V := \begin{bmatrix} 1 & w & w^2 & w^3 \\ 1 & x & x^2 & x^3 \\ 1 & y & y^2 & y^3 \\ 1 & z & z^2 & z^3 \end{bmatrix}$$

Without computing  $\det(V)$  explain why the determinant of  $V$  is zero if any two of the variable  $w, x, y, z$  are equal. Conversely, by computing  $\text{factor}(\det(V))$  deduce if  $\det(V) = 0$  then some two of the variables are equal.

```
>
```