LINEAR ALGEBRA USING MAPLE

Prepared by Victoria Bush

This handout is designed to help you learn how to do linear algebra using Maple on the Ultrix computers at IU Bloomington. However, most of the information contained in this document is also relevant to Maple on other platforms (MAC, DOS, etc.)

If you have never used Maple on a Unix machine before, you can sign up for a "Using Maple V under UNIX" class. There is also an interactive tutorial available. It can be accessed by first starting a maple session and then typing tutorial();.

I. A Few Preliminaries.

All the linear algebra functions that are used by Maple are in a separate package called linalg. So if you try to find the determinant of a matrix by just typing in det(A); you will see a prettyprinted version of what you just typed! This means that Maple, not being able to assign what you typed to a function, assumed you typed in text and it echoes that text on the screen.

So the first thing you MUST do whenever you want to do linear algebra is read into Maple all the functions in linalg. At the Maple prompt (>), type:

> with(linalg); [enter]

and you will see a list of all the functions now included in Maple's memory. If you only want to use one linear algebra function and don't really want to read in all the functions, you can use just one by using this command instead:

>linalg[function] (arguments); [enter]

So if I want to take the determinant of a predefined matrix A and not read in all the linear algebra functions, I would type:

> linalg[det](A); [enter] .

As with any Maple function or command, you can always get online information using the ? command. To get a listing of all the functions contained in linalg, type:

> ?linalg; [enter] .

If you want help on a specific function, type:

> ?function_name; [enter] .

For each help screen there is usually a list of examples at the end which often helps to clarify the technical text given.

There is one thing which you must remember. All the arguments passed to these programs must be matrices or vectors. This means that if you type a matrix in by hand, it must be called an array, contained in brackets [and], and each row must be contained in another set of brackets, as in the following assignment:

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

If you just want to define the matrix without defining the entries, use the array command and be sure that both ranges of numbers begin with 1.

> B:= array(1..4, 1..3);

For vectors, the same rules apply, except that you use only one set of brackets or one range of numbers.

> u:= array([1,2,3]); [enter]
> v:= array(1..5); [enter]

While array provides a general way to define multidimensional arrays there are also two functions that define vectors and arrays explicitly and are somewhat easier to use. They are vector and matrix:

> a:=vector([5,4,6,3]);
a := [5, 4, 6, 3]

>> matrix(2,2,[5,4,6,3]);

```
[ 5   4 ]
[       ]
[ 6   3 ]
```

You can also use functions to define vector and matrices:

> f := x -> x^2:
> v := vector(4,f);
v := [1, 4, 9, 16]

Vector(n,f) produces a vector of length n whose elements are the result of the function f acting on the index of the vector. Thus vector(n,f) defines the vector [f(1), f(2), ..., f(n)].

Similarly with matrix:

> f := (i,j) -> x^(i+j-1):
> A := matrix(2,2,f);

```
          [         2 ]
          [   x   x   ]
     A := [           ]
          [   2     3 ]
          [  x     x  ]
```

This defines a 2 by 2 matrix whose (i,j)th entry is f(i,j).

If you want to define a large matrix you might want to use entermatrix function that will prompt you for the values to be inserted:

> B:=array(1..2,1..2);

B := array(1 .. 2, 1 .. 2, [])

(This defines an "empty" 2 by 2 matrix, you always have to specify the size of the matrix before calling entermatrix)

> entermatrix(B);

Note: each matrix entry entered MUST BE FOLLOWED BY A SEMICOLON

enter element 1,1 > 10;
enter element 1,2 > 20;
enter element 2,1 > 30;
enter element 2,2 > 40;

```
[ 10   20 ]
[         ]
[ 30   40 ]
```

Entermatrix can also be used to chang values of some elements in the matrix or specify matrix with special properties (symmetric, etc.). You can find out more about it by typing ?entermatrix; .

One more thing: ALWAYS remember to end each Maple command with a semicolon. If no semicolon is entered at the end of each command, then Maple will just sit there, waiting for you! If this happens, you may enter the semicolon on the next line and Maple will continue as normal.

II. A Few Linear Algebra Functions.

Here is a list of some of the most used functions within the linalg package. The arguments follow the same rules in most cases. Here I have assigned the matrices and vectors to variables, but you can just type in the matrix or vector directly into the function.

a) add

Use add to add two matrices of the same dimensions. You have the option of adding multiples of matrices together. If two matrices A and B have been defined and they have the same dimensions, you can add them by typing:

> add(A, B); [enter] .

If instead you want to add 3*A and 4*B, you could type:

> add(A, B, 3, 4); [enter] .

As a better example of this, look at the following lines typed into Maple and Maple's response (a colon is used in the array assignments to prevent Maple from prettyprinting the assignment):

```
> A:= array([[1,2,3], [2,3,4], [3,4,5]]): [enter]
> E:= array([[1,0,0], [0,1,0], [0,0,1]]): [enter]
> add( A, E, 1, 10);

                  [  11    2     3   ]
                  [                  ]
                  [   2   13     4   ]
                  [                  ]
                  [   3    4    15   ]
```

b) charmat

If you need to construct the characteristic matrix of a matrix A (A - lambda * I, where I is the identity matrix), use this command to generate the matrix with your choice of what to call lambda. If you type:

> charmat(A, lambda); [enter]

where A is defined as above in add, you will get this response from Maple:

```
        [ 1 - lambda      2            3       ]
        [                                      ]
        [     2        3 - lambda      4       ]
        [                                      ]
        [     3           4       5 - lambda ]
```

whereas if you type:

> charmat(A, x);

you will see this:

```
        [    1 - x       2          3      ]
        [                                  ]
        [      2       3 - x        4      ]
        [                                  ]
        [      3         4        5 - x    ]
```

c) charpoly

This function will generate the characteristic polynomial of a matrix A ((-1)^n * det(A - lambda*I), where I is the identity matrix and n is the dimension of A) which you can assign to a variable name.

If we define X to be this matrix:

X:= array([[1,2,3], [1,2,3], [1,5,6]]); [enter]

then using charpoly, we can assign the characteristic polynomial to the variable eqn:

> eqn:=charpoly(X, y); [enter]

and Maple's reply would be:

$$eqn := -9y^2 + y^3$$

and then you could use the solve command to get the eigenvalues.

d) colspace

This command will take a matrix and, after performing Gaussian elimination, will return the nonzero rows as a set. When using this function make sure that all the entries in the matrix are defined, because otherwise Maple will return an error message.

If we define a matrix A:

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

and apply colspace to A:

> colspace(A); [enter]

Maple responds with:

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

e) crossprod

When you need the cross product of two vectors (each with three entries), say u and v, type in:

> crossprod(u,v); [enter]

and Maple will return the result as a vector.

f) det

This command will find the determinant of a square matrix. You do not need to have numerical entries in the matrix for det to work. As long as each entry is defined as a number or some expression in variables, you can get the determinant.

To find the determinant of a matrix A, type:

> det(A); [enter] .

g) dotprod

Using this command, you can find the dot (or scalar) product of two vectors. The result returned will be numerical. (If using complex numbers, check the help screen using ?dotprod; to see how Maple deals with the dot product over the complex field.)

For two defined vectors u and v, find the dot product by typing:

> dotprod(u, v); [enter] .

h) eigenvals

This function will return the eigenvalues of a square matrix, say A. The eigenvalues are computed by solving the characteristic polynomial det(A - lambda I) for the variable lambda, where I is the identity matrix.

Find the eigenvalues of A by typing:

> eigenvals(A); [enter] .

If the degree of the characteristic polynomial is greater than four, Maple may not be able to find all the eigenvalues. In that case, you can try using evalf(Eigenvals(A)); which uses a more general algorithm.

i) eigenvects

This procedure solves the system of equations given by

(A - lambda*I)*C = 0

where Maple solves for the vector C for each of the eigenvalues of A, which are substituted into lambda. Maple's answer is given in the form of a list. For each eigenvalue, the list is:

{eigenvalue, multiplicity, eigenvectors, ...}

where multiplicity may or may not be given and eigenvectors are the vectors corresponding to eigenvalue. (For information about eigenspaces and what the vectors represent, type ?eigenvects;.)

The following represents a sample use of eigenvects:

```
> mat:= array([[1,-3,3], [3,-5,3], [6,-6,4]]):
> eigenvects(mat);

       [-2, 2, {[0,1,1], [1,1,0]}, 4, 1, {[1,1,2]} ]
```

j) gausselim

For an m by n matrix A gausselim will reduce the matrix by using fraction-free Gaussian elimination with row pivoting. The result is an upper triangular matrix.

> A:= array([[x,1,0], [0,0,1], [1,y,1]]); > gausselim(A);

```
            [ x       1        0    ]
            [                       ]
            [ 0    y x - 1     x    ]
            [                       ]
            [ 0       0     y x -  1 ]
```

k) inverse

Use this function if you need to find the inverse of a square matrix. If the matrix does not have an inverse, an error will occur. The command is:

> inverse(A); [enter] .

l) iszero

iszero(A); checks to see if the all the entries in the matrix A are zero. If so, then Maple responds with true; otherwise the result will be false. (iszero is a boolean operator.)

m) linsolve

When you type:

> linsolve(A, b); [enter]

where A is an m by n matrix and b is a vector, Maple will find the vector x which satisfies the matrix equation A*x = b. Maple will check to make sure that the number of rows in A is the same as the dimension of b; otherwise an error will occur. If there is no solution, Maple will respond with the word NULL. If there are many solutions, Maple will present the result parametrically using the parameters t1, t2, t3, etc.

You can also use this function where b could be a matrix. The same results occur for no solution and many solutions, and for mismatched matrices you will get an error message.

n) multiply

This command will work with either matrices or a matrix and a vector. When you multiply matrices, Maple will multiply them in the order you give them (keeping in mind that matrix multiplication is not commutative). For example:

```
> A:= array([[1,2], [3,4]]);
> B:= array([[0,1], [1,0]]);
> C:= array([[1,2], [4,5]]);
> multiply(A,B,C);
```

```
[ 6    9 ]
[        ]
[16   23 ]
```

If you need to multiply the same matrix a number of times, a cleaner way to do it is to design a do..od loop. Let's say I want to multiply a matrix A by itself 10 times. I also want to be able to call each product and check the entries. In the following program, I use the concatenation operator (.) to keep track of all the matrices that this loop creates. Also I define A1 by typing A.1 so I can multiply A by itself within the loop. The print statement allows me to specify exactly which output Maple should print out since I have used a colon throughout to prevent Maple from prettyprinting an answer like it normally does when one uses a semicolon.

```
> with(linalg):
```

```
> A:= array([[.5, .5], [.2, .8]]):
> A.1:=A:
> for i to 9 do

>      j:=i+1:
>      A.j:=multiply(A.i, A):
>      print( op(A.j) ):

> od:
```

Now when I run this, the matrices A2 to A10 will show up on the screen in order. If you just want to see one of these matrices, you cannot type A5 [return] because Maple will respond by prettyprinting A5. To see the entries in the matrix A5, you can type:

```
> op(A5); [enter] or
> print(A5); [enter]
```

and you will see the matrix. (op stands for operand. To see how this works, type ?op; for information.)

o) randmatrix

This function generates a random matrix. If you enter:

```
> randmatrix(4,5); [enter]
```

then Maple will generate a 4 by 5 matrix where all entries are between -99 and 99. You can define what function will be used to generate the entries, but this is sometimes difficult. (For more information type ?randmatrix;.)