MAPLE Notes for Algebraic Geometry

Michael Monagan
Department of Mathematics
Simon Fraser University
August, 1998.
Updated August 2002, September 2004, January 2009, January 2014.

```
> restart;
```

These notes are for Maple V Release 10. They are platform independent, i.e., they are the same for the Macintosh, PC, and Unix versions of Maple. These notes should be backwards compatible with Maple 9 and forwards compatible with Maple 11.

Maple as a Calculator

Input of a numerical calculation uses +, -, *, /, and ^ for addition, subtraction, multiplication, division, and exponentiation respectively.

```
> 1+2*3-2;
```

$$5$$

```
> 2^3;
```

$$8$$

```
> 120/105;
```

$$\frac{8}{7}$$

Because the input involved integers, not decimal numbers, Maple calculates the exact fraction when there is a division, automatically cancelling out the greatest common divisor (GCD). In this case the GCD is 15, which you can calculate specifically as

```
> igcd(120,105);
```

$$15$$

Observe that every command ends with a semicolon ; This is a gramatical requirement of Maple. If you forget, Maple will assume that the comand is not complete. This allows you to break long commands across a line. For example

```
> 1+2*3/
> (2+3);
```

$$\frac{11}{5}$$

For decimal numbers, the presence of a decimal point . in a number means that the number is a decimal number and Maple will, by default, do all calculations to 10 decimal places.

```
> 120/105.0;
```

$$1.142857143$$

```
> sqrt(2.0);
```

$$1.414213562$$

```
> s := sqrt(2);
```

$$s := \sqrt{2}$$

Notice the difference caused by the presence of a decimal point in these examples. Now, if you have input an exact quantity, like the $\sqrt{2}$ above, and you now want to get a numerical value to 3 decimal digits, use the evalf command to evaluate to floating point. Use the % character to refer to the previous Maple output.

```
> evalf(s,3);
```

$$1.41$$

To input a formula, just use a symbol, e.g. x and the arithmetic operators and functions known to Maple. For example, here is a quartic polynomial in x.

```
> x^4-3*x+2;
```

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

We are going to use this polynomial for a few calculations. We want to give it the name f so we can refer to it later. We do this using the assignment operation in Maple as follows

```
> f := x^4-3*x+2;
```

$$f := x^4 - 3\,x + 2$$

The name f is now a variable. It refers to the polynomial. Here is it's value

```
> f;
```

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

To evaluate this as a function at the point $x = 2$ use the eval command as follows

```
> eval(f,x=2);
```

$$12$$

The following commands differentiate *f* with respect to *x* and factor *f* into irreducible factors over the field of rational numbers.

```
> diff(f,x);
```

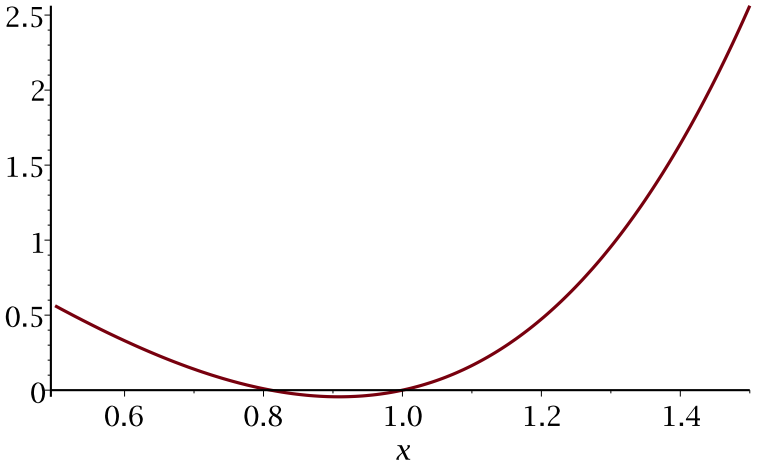
$$4\,x^3 - 3$$

```
> factor(f);
```

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

You can graph functions using the plotting commands. The basic syntax for the **plot** command for a function of one variable is illustrated as follows. To graph *f(x)* on the range $a \le x \le b$ we use $x = a..b$.

```
> plot(f,x=0.5..1.5);
```



In the graph I can see a local minimum near x=0.9. We can find this point using calculus. The command **fsolve**(f(x)=0, x), on input of a polynomial *f(x)* computes 10 digit numerical approximations for the real roots of *f(x)*.

```
> diff(f,x)=0;
```

$$4x^3 - 3 = 0$$

```
> fsolve(diff(f,x)=0,x);
```

$$0.9085602964$$

Here are some other polynomials and functions

```
> f := x^2-3*x*y+y^2+x*y;
```

$$f := x^2 - 2xy + y^2$$

```
> factor(f);
```

$$(x-y)^2$$

```
> f := (1-t^2)/(1+t^2);
```

$$f := \frac{-t^2+1}{t^2+1}$$

```
> limit(f,t=infinity);
```

$$-1$$

```
> f := 2*t*sin(omega*t);
```

$$f := 2t\sin(\omega t)$$

```
> diff(f,t);
```

$$2\sin(\omega t) + 2t\cos(\omega t)\,\omega$$

We have used the name f as variable to refer to formulae and the symbols x for an uknown in a formula. Often you will have assigned to a name like we have done here to f but you want now to use the name f as a symbol again, not as a variable. You can unassign the value of a name as follows

```
> f;
```

$$2t\sin(\omega t)$$

```
> f := 'f';
```

$$f := f$$

```
> f;
```

$$f$$

Finally, the complex unit in Maple is I, not i, and the value 3.14159... is Pi not pi. The symbols i and pi do not have any special meaning - so you can use them as variables.

```
> solve(x^2+1=0,x);
```

$$\mathrm{I},\ -\mathrm{I}$$

```
> I^2;
```

$$-1$$

```
> i^2;
```

$$i^2$$

```
> sin(Pi);
```

$$0$$

```
> sin(pi);
```

$$\sin(\pi)$$

Varieties in R^2 and R^3

Equations are input using = .

```
> eqn := x^2+y^2=4;
```

$$eqn := x^2 + y^2 = 4$$

```
> lhs(eqn);
```

$$x^2 + y^2$$

```
> rhs(eqn);
```

$$4$$

```
> lhs(eqn)-rhs(eqn) = 0;
```

$$x^2 + y^2 - 4 = 0$$

Here is a linear system in R^2, it's solution and a graph of it.

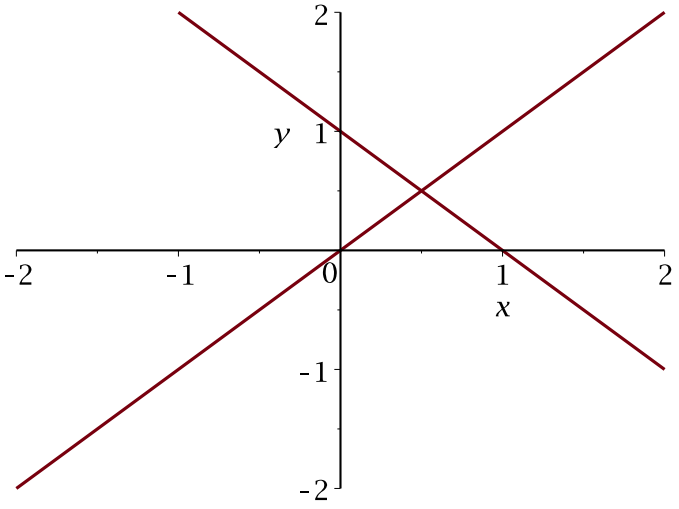
```
> S := { x+y=1, x-y=0 };
```

$$S := \{x - y = 0,\ x + y = 1\}$$

```
> solve(S,{x,y});
```

$$\left\{x = \frac{1}{2},\ y = \frac{1}{2}\right\}$$

```
> plots[implicitplot](S, x=-2..2, y=-2..2 );
```



The following shows that solve does not return solutions as radicals automatically - one has to set a magic varaible (_EnvExplicit) to get radical solutions.

```
> S := { x^2+y^2=1, x+y=0 };
```

$$S := \{x + y = 0, x^2 + y^2 = 1\}$$

```
> solve(S,{x,y});
```

$$\{x = -RootOf(2\_Z^2 - 1), y = RootOf(2\_Z^2 - 1)\}$$

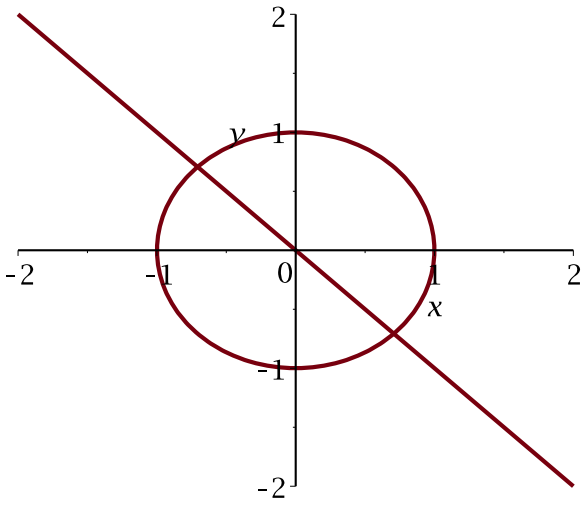
```
> _EnvExplicit := true;
  solve(S,{x,y});
```

$$\_EnvExplicit := true$$

$$\left\{x = -\frac{1}{2}\sqrt{2}, y = \frac{1}{2}\sqrt{2}\right\}, \left\{x = \frac{1}{2}\sqrt{2}, y = -\frac{1}{2}\sqrt{2}\right\}$$

The default for the grid option is [25,25]. The higher the values, the more accurate (less jagged) the plot will be. See ?plots[implicitplot]

```
> plots[implicitplot]( S, x=-2..2, y=-2..2, thickness=2, grid=[50,50]
  );
```



Everything can be done in 3 dimensions. With surfaces. Note, in Maple 11, there is a new command intersectplot that shows the intersection of two surfaces in R^3.

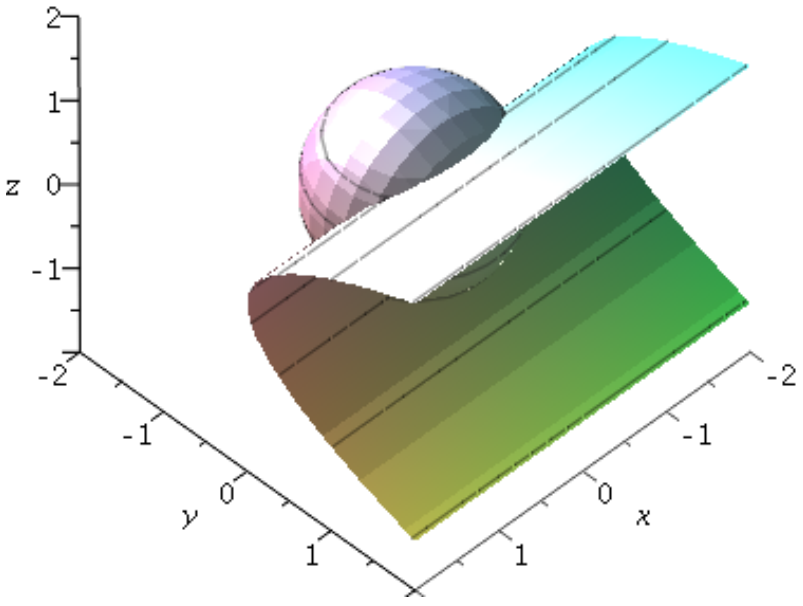
```
> S := {x^2+y^2+z^2=1, y-z^2};
```

$$S := \{-z^2 + y,\ x^2 + y^2 + z^2 = 1\}$$

```
> solve(S,{x,y,z});
```

$$\left\{x = \sqrt{-z^4 - z^2 + 1},\ y = z^2,\ z = z\right\},\ \left\{x = -\sqrt{-z^4 - z^2 + 1},\ y = z^2,\ z = z\right\}$$

```
> plots[implicitplot3d]( S, x=-2..2, y=-2..2, z=-2..2, grid=[20,20,
  20], axes=frame, style=patchcontour);
```



Lists and Sets

The simplest data structure in Maple is a list. The elements of a list may be of any type. To create a list of values enclose them in square brackets [,]. Lists may be nested of course and the entries may be of any type.

```
> restart;
```

```
> E := []; # the empty list
```

$$E := [\,]$$

```
> L := [1,2,-3,4,1];
```

$$L := [1, 2, -3, 4, 1]$$

```
> M := [[1,2,3],[x,y,z]];
```

$$M := [[1, 2, 3], [x, y, z]]$$

To count the number of entries in a list use nops(L) command.

```
> nops(L);
```

$$5$$

```
> nops(M);
```

$$2$$

To access the i'th element of a list (counting from 1) use a subscript. A negative subscript counts from the end.

```
> L[3];
```

$$-3$$

```
> L[-1];
```

$$1$$

```
> M[2];
```

$$[x, y, z]$$

```
> M[2][2];
```

$$y$$

Use the following to extract a sublist

```
> L[2..3];
```

$$[2, -3]$$

```
> L[2..-1];
```

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

To append (prepend) elements to a list use the following.

```
> op(L);
```

$$1, 2, -3, 4, 1$$

```
> [op(L),5];
```

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

To test if an element is in a list use

```
> member(2,L);
```

$$true$$

Although you can assign to an entry of a list (as if it were an array) if the list has less than 100 elements, do not do this. It creates a copy of the entire list. So it's not efficient. Use Arrays .

```
> L[2] := 10;
```

$$L_2 := 10$$

```
> L;
```

$$[1, 10, -3, 4, 1]$$

Maple sets are like mathematics sets, duplicates are removed. To create a set use squigley brackets { and } .
The examples should be self explanatory.

```
> {};  # The empty set
```

$$\{\}$$

```
> S := {1,5,3,1};
  T := {3,4,5,6};
```

$$S := \{1, 3, 5\}$$

$$T := \{3, 4, 5, 6\}$$

```
> S[2];
```

$$3$$

```
> nops(S); # cardinaltiy
```

$$3$$

```
> S union T;
```

$$\{1, 3, 4, 5, 6\}$$

```
> S intersect T;
```

$$\{3, 5\}$$

```
> S union {7};
```

$$\{1, 3, 5, 7\}$$

```
> S minus T;
```

$$\{1\}$$

```
> member(5,S); # set membership
```

$$true$$

The seq, add and mul commands.

The seq(f(i), i=a..b), mul(f(i), i=a..b), add(f(i), i=a..b) useful for working with sets, lists and polynomials.

```
> restart;
```

```
> seq( i, i=1..4 );
```

$$1, 2, 3, 4$$

```
> seq( f(i), i=1..4 );
```

$$f(1), f(2), f(3), f(4)$$

creates a sequence of values which you can put in a list or set, e.g,

```
> L := [ seq(x[i], i=1..4) ];
```

$$L := [x_1, x_2, x_3, x_4]$$

The add and mul commands work the same way except they create a sum (product) respectively.

```
> L := [x,y,z];
```

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

```
> add( L[i], i=1..3 );
```

$$x + y + z$$

```
> mul( L[i], i=1..3 );
```

$$x\,y\,z$$

```
> f := x^3-3*x^2+5;
```

$$f := x^3 - 3\,x^2 + 5$$

```
> degree(f,x);
```

$$3$$

```
> coeff(f,x,2);
```

$$-3$$

```
> [seq( coeff(f,x,i), i=0..degree(f,x) )];
```

$$[5, 0, -3, 1]$$

```
> C := [1,0,3,5];
```

$$C := [1, 0, 3, 5]$$

```
> add( C[i]*x^(i-1), i=1..nops(C) );
```

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

Read the help pages

```
> ?seq
> ?add
```

Loops and If statements.

```
> restart;
```

To do a sequence of calculations it will be handy to know how to use some of Maple's looping commands and also the **if** command. To execute a command in Maple conditionally use the if command which has either of the following forms

if <condition> **then** <statements> **else** <statements> **fi**

or just

if <condition> **then** <statements> **fi**

```
> if 2>1 then print(good) else print(bad) fi;
```

good

To execute one or more statements zero or more times in a loop use the for command. It has the following form

for <variable> **from** <start> **to** <end> **do** <statements> **od**

```
> for i from 1 to 4 do i^2; od;
```

$$1$$

$$4$$

$$9$$

$$16$$

If a loop index starts from 1 then you can omit the "from 1" clause. So the above loop may be written

```
> for i to 4 do i^2; od;
```

$$1$$

$$4$$

$$9$$

$$16$$

As an example, the Maple gcd command computes the gcd of two polynomials.
To compute the gcd of a list of 1 or more polynomials we can use a loop as follows.

```
> L := [x^6-1,x^4-1,x^12-1,x^3-1];
```

$$L := [x^6 - 1, x^4 - 1, x^{12} - 1, x^3 - 1]$$

```
> g := L[1];
  for i from 2 to nops(L) do g := gcd(g,L[i]) od;
```

$$g := x^6 - 1$$

$$g := x^2 - 1$$

$$g := x^2 - 1$$

$$g := x - 1$$

To execute some statements while a condition is true use the while loop. It has the syntax

while <condition> **do** <statements> **od**

In the following example we repeatedly divide an integer n by 2 until it is odd.

```
> n := 12;
  while irem(n,2) = 0 do n := iquo(n,2); od;
```

$$n := 12$$

$$n := 6$$

$$n := 3$$

The value of n at the end of the loop is

```
> n;
```

$$3$$

Maple Programming

A simple function, like the function f(x) = x^2 may be input using the arrow notation in Maple, as follows

```
> f := x -> x^2;
```

$$f := x \to x^2$$

```
> f(2);
  f(y);
```

$$4$$

$$y^2$$

A procedure in Maple takes the form

```
proc( p1, p2, ... )
local l1, l2, ... ;
global g, g2, ... ;
   statement1;
   statement2;
    ....
   statementn;
end proc
```

There may be zero or more parameters, one or more locals, one or more globals and one or more statements in the procedure body.
The local and global statements are optional. Variables in the procedure body that are not explicitly declared as parameters, locals, or globals are declared to be local automatically if assigned to, otherwise they are global. The value returned by the procedure is the value of *statementn*, the last statement in the body of the procedure or the value of an explicit return statement. Type declarations for parameters and local variables need not be explicitly given. Some examples will help.

```
> f := proc(x)
      y := x^2;
      y-1;
  end proc;
```

Warning, `y` is implicitly declared local to procedure `f`

$$f := \mathbf{proc}(x)\ \mathbf{local}\ y;\ y := x\text{^}2;\ y-1\ \mathbf{end\ proc}$$

```
> f(2);
```

$$3$$

Here is a Maple procedure to compute the LT(f), the leading term of a univariate polynomial f(x). It illustrates the use of the return statement. This time I've explicitly declared c and d to be local variables.

```
> LT := proc(f,x) local d,c;
     if f=0 then return 0; fi;
     d := degree(f,x);
     c := coeff(f,x,d);
     c*x^d;
  end proc;
```

$$LT := \mathbf{proc}(f, x)$$
$$\mathbf{local}\ d, c;$$
$$\mathbf{if}\ f = 0\ \mathbf{then\ return}\ 0\ \mathbf{end\ if};\ d := degree(f, x);\ c := coeff(f, x, d);\ c*x\text{^}d$$
$$\mathbf{end\ proc}$$

```
> LT(0,x);
```

$$0$$

```
> f := 3*x^3-5*x+1;
```

$$f := 3\,x^3 - 5\,x + 1$$

```
> LT(f,x);
```

$$3\,x^3$$

This next example is an implementation of the Euclidean algorithm in Q[x]. It uses the rem command to compute the remainder of f divided by g. It also use multi-assignments.

```
> (f,g) := (x^6-1,x^4-1);
```

$$f, g := x^6 - 1, x^4 - 1$$

```
> f;
```

$$x^6 - 1$$

```
> g;
```

$$x^4 - 1$$

```
> rem(f,g,x);
```

$$x^2 - 1$$

```
> EuclideanAlgorithm := proc(f,g,x) local c,d,r;
      (c,d) := (f,g);
      while d <> 0 do
          r := rem(c,d,x);
          (c,d) := (d,r);
      od;
      c;
  end proc;
```

$$EuclideanAlgorithm := \mathbf{proc}(f, g, x)$$

$$\mathbf{local}\ c, d, r;$$

$$c, d := f, g;\ \mathbf{while}\ d <> 0\ \mathbf{do}\ r := rem(c, d, x);\ c, d := d, r\ \mathbf{end\ do};\ c$$

$$\mathbf{end\ proc}$$

```
> EuclideanAlgorithm(f,g,x);
```

$$x^2 - 1$$

Procedures may be nested, nested lexical scoping is used (a la Pascal).
Procedures may be returned and passed freely as parameters.
The simplest debugging tool is to insert print statements in the procedure.
For example

```
> EuclideanAlgorithm := proc(f,g,x) local c,d,r;
      (c,d) := (f,g);
      while d <> 0 do
          r := rem(c,d,x);
          print(REM(c,d) = r);
          (c,d) := (d,r);
      od;
      c;
  end proc;
```

$$EuclideanAlgorithm := \mathbf{proc}(f, g, x)$$

$$\mathbf{local}\ c, d, r;$$

$$c, d := f, g;\ \mathbf{while}\ d <> 0\ \mathbf{do}$$

$$r := rem(c, d, x);\ print(REM(c, d) = r);\ c, d := d, r$$

$$\mathbf{end\ do};$$

```
    c
end proc
```

```
> g := EuclideanAlgorithm(f,g,x);
```

$$REM(x^6-1, x^4-1) = x^2-1$$

$$REM(x^4-1, x^2-1) = 0$$

$$g := x^2-1$$

The next simplest debugging tool is the trace command. All assigment statements are displayed.

```
> trace(EuclideanAlgorithm);
```

$$EuclideanAlgorithm$$

```
> g := EuclideanAlgorithm(f,g,x);
{--> enter EuclideanAlgorithm, args = x^6-1, x^2-1, x
```

$$c, d := x^6-1, x^2-1$$

$$r := 0$$

$$REM(x^6-1, x^2-1) = 0$$

$$c, d := x^2-1, 0$$

$$x^2-1$$

```
<-- exit EuclideanAlgorithm (now at top level) = x^2-1}
```

$$g := x^2-1$$

The printf command can be used to print more detailed information in a controlled format. It works just like the printf command in the C language. The main difference is the %a option for printing algebraic objects like polynomials. E.g.

```
> f := x^3-2*x^2+1;
```

$$f := x^3-2x^2+1$$

```
> printf( "The polynomial f=%a has degree %d in %a\n", f, 3, x );
The polynomial f=x^3-2*x^2+1 has degree 3 in x
```

As a final example, here is a Maple procedure for the division algorithm which uses the LT procedure.

```
> DIVIDE := proc(f,g,x) # divide f by g
  local q,r,t;
      if g=0 then error "division by zero"; fi;
      q := 0;
      r := f;
      while r <> 0 and degree(r) >= degree(g) do
           t := LT(r,x)/LT(g,x);
           q := q+t;
           r := r-expand(t*g);
      od;
      return(q,r); # return both values
  end:
> f := 2*x^4-3*x^3+1;
```

```
> g := 2*x^2+1;
```

$$f := 2x^4 - 3x^3 + 1$$

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

```
> q,r := DIVIDE(f,g,x):
> q;
```

$$x^2 - \frac{3}{2}x - \frac{1}{2}$$

```
> r;
```

$$\frac{3}{2} + \frac{3}{2}x$$

Check that the result is correct.

```
> expand( f-q*g-r );
```

$$0$$

Subscripted Names and Arrays

Variables may be subscripted. For example, here is a polynomial in x_1, x_2, x_3. You can assign to the subscripts.

```
> restart;
> f := 1-x[1]*x[2]*x[3];
> x[1] := 3;
```

$$f := -x_1 x_2 x_3 + 1$$

$$x_1 := 3$$

```
> f;
```

$$-3x_2 x_3 + 1$$

There may be more than one subscript and the subscripts may be any value.
Arrays are like arrays from computing science. Here is how to create a one-dimensional array A with values indexed from 1 to 5.

```
> A := Array(1..5);
```

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

By default, the entries in the array A are initialized to 0. You access and assign array entries using subscripts. For example

```
> A[1];
```

$$0$$

```
> A[1] := 3;
```

$$A_1 := 3$$

```
> A[1];
```

$$3$$

```
> for i from 2 to 5 do A[i] := x^i od;
```

$$A_2 := x^2$$

$$A_3 := x^3$$

$$A_4 := x^4$$

$$A_5 := x^5$$

```
> A;
```

$$\begin{bmatrix} 3 & x^2 & x^3 & x^4 & x^5 \end{bmatrix}$$

Arrays can be multidimensional. Here is a two dimensional Array with 2 rows and 3 columns

```
> A := Array(1..2,1..3);
```

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

```
> for i to 3 do A[1,i] := i; od;
```

$$A_{1,1} := 1$$

$$A_{1,2} := 2$$

$$A_{1,3} := 3$$

```
> A;
```

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

```
> numelems(A);
```

$$6$$

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

$$1..2, 1..3$$