```
> solve({x*y-y,x^2-y^2});

            {x = 0, y = 0}, {x = 1, y = 1}, {x = 1, y = -1}
```

Sometimes Maple answers using `RootOf`, which is short for "the solution of the equation". This function uses the global parameter `_Z`. To force a (numerical) solution use `fsolve` or `allvalues`.

To solve the differential equation `deq` with `y` as unknown, type `> dsolve(deq,y(x));` An example of `deq`:
`> deq:=diff(y(x),x,x)-y(x)=0;`
Initial conditions are treated like this: `> deq:={diff(y,x)-x=0,y(0)=1};`
It is also possible to solve a DE numerically with a 4-5:th order Runge-Kutta. Type `> ?dsolve[numeric]` for more information about this.

VECTORS AND MATRICES

Before starting matrix calculations, load the linear algebra package:
`> with(linalg):`
and preferably `evalm`:
`> readlib(evalm):`
Vectors and matrices are created with the commands `vector` and `matrix` respectively. A vector (which is not a special case of a matrix) is regarded as a column vector by multiplication. Some examples:

```
> A:=matrix([[4,34],[-a,sqrt(2)]]):
> a12:=A[1,2]:
> v:=vector([x,-12]):
> w:=v[2]:
```

Whenever doing matrix calculations the function `evalm` has to be applied. Matrix multiplication is called `&*` which has to be surrounded by spaces. Multiplication with a scalar is called `*` Thus to perform $A = 2B'C + D$, where the operands are matrices, you type
`> A:=evalm(2* transpose(B) &* C + D);`

The characteristic polynomial of `A` with `s` as variable is obtained by `> charpoly(A,s);` and `> eigenvals(A);` gives the eigenvalues of `A` as an expression sequence. Example:

```
> B:=matrix([[1, sqrt(2)], [-ln(3), 3]]):
> map(evalf,[eig(B)]);

                    [2. + .74404 I, 2. - .74404 I]
```

When applying a function element-wise, `map` should be used.

MATHEMATICAL FUNCTIONS

> `> diff(f,x);` means "differentiate `f` with respect to `x`"
> `> diff(f,x$n);` yields the n:th derivative.
> `> grad(f,[x1,x2]);` computes the gradient of `f` w.r.t. `[x1,x2]`.
> `> jacobian(f,[x1,x2]);` jacobian matrix of the vector `f`.

To do Laplace-transformations first read the `laplace` package: `> readlib(laplace):` Then type, e.g. `> laplace(f,t,s);`

The logarithm function is called `ln`.

Integration: `> int(f,x)` or `> int(f,x=0..10)` For numerical integration, apply `evalf`:

```
> evalf(int(sin(x)/x,x=0..1),10);

                              .9460830704
```

`> limit(ln(x)/x,x=0,right);` is an example of a limit.

```
> sum(binomial(n,k)*x^k,k=0..n);
                                          n
                                   (1 + x)
```

`> k!;` denotes k factorial.

A Taylor expansion of order `ord` of `expr` for `var` close to zero is given by:
`> taylor(expr,var,ordn);`
To obtain the expansion around 1 instead, type
`> taylor(expr,var=1,ordn);`

EQUATIONS

The call `solve(eq,var);` solves the equation `eq` w.r.t. `var`. If there is no `=` in `eq` the right hand side is supposed to be zero and if `var` is left out, all variables occurring are considered unknown. If there are several solutions to `eq` then `solve` returns an expression sequence, i.e. several expressions separated by commas. It is up to the user to collect the solutions in a set or a list etc. An equation system is given as a set of equations; in this case the second argument is a set of variables.

```
> eq:=x^2-x+1=0:
> solve(eq);
                                 1/2                 1/2
                    1/2 + 1/2 I 3   , 1/2 - 1/2 I 3

> S:=[solve(2*z^3-z^2+11*z-21)];

                                 1/2                     1/2
          S := [3/2, - 1/2 + 3/2 I 3   , - 1/2 - 3/2 I 3   ]
```

The value returned from the procedure is the one of the last expression in the procedure - the one before `end;`. If the number of in-arguments is not predetermined, use `args` and `nargs`.

The `alias` facility is quite useful. An example:

```
map(alias,[tp=transpose,SE=simplify@expand,choose=binomial]):
```

LOOPS

General for-loop: `for i from 1 to 10 do x[i]:=i^2: y[i]:=i^3 od:`
Simplified: `for i to 10 do x[i]:=i^2: y[i]:=i^3 od:`
While-loop: `> while i<3 do i:=i^2+1 od;`

CONDITIONAL CLAUSES

General syntax: `if x>0 then y:=x else y:=-x fi:`

PLOTTING

If you are interested in plotting, run Maple under X-windows or on a 386, etc. Useful plot commands are: `plot, plot3d, tubeplot, spacecurve`

The syntax of `plot` and `plot3d` is hopefully obvious from the examples below. They can both take a set of functions as their first argument.

The advanced plot-functions have to be loaded with the command `> with(plots);`
Some simple examples:

```
> plot(sin(x), x=0..2*Pi);
> plot({sin(x), x-x^3/6}, 0..2);
> f:=exp(-x^2-2*y^2)-sin(7*x)/8:
> plot3d(f,x=-1..1,y=-1..1);
```

SIMPLIFICATION

Some pretty obvious commands are: `simplify, expand, factor`
Sometimes `simplify(expand(expr))` differs from `simplify(expr)`.
The opposite of `expand` is `combine`, in some sense.

`> normal(expr);` $\Rightarrow$ Simplification of rational expressions.

`> collect(pol,var);` returns the multivariate polynomial `pol` written as a univariate one, in the variable `var`. (More advanced uses are possible).

This principle does not work for vectors and matrices. E.g. no evaluation of the entries of the matrix `A` is made at the call `> op(A);` The best way to circumvent this problem is to use `subs` (see **Elementary Manipulation** below)

For complex evaluation of `expr`, use `> evalc(expr);`

To evaluate the expression `expr` numerically with `n` digits, type `> evalf(expr,n);` If `n` is left out the floating point precision is determined by the variable `Digits`.

The function `evalhf` uses the hardware floating-point of the system, to enhance speed.

ELEMENTARY MANIPULATION

`> op(n,expr);` extracts operand number n.

`> nops(expr);` number of operands, e.g. the length of a list.

`> subs(a=b,expr);` Substitute `a` by `b` in `expr`. If the second argument is a vector, matrix etc. you have to use `op`: `> subs(a=b,op(M));`

`rhs, lhs` right and left hand sides of equations

`Re, Im` Real and imaginary part

`> coeff(pol,var);` gives the coefficient of `var` in the polynomial `pol`.

USER-DEFINED FUNCTIONS

There are several possibilities for writing your own functions in Maple. Simplest is probably the arrow-construct. An example:

```
> f:=x -> x^2-a;
                                          2
                              f := x -> x   - a
> g:=(x,y) -> x*y-1;

                            g := (x,y) -> x y - 1

> g(f(1),2);
                                 - 2 a + 1
```

If the function is more complicated and local variables are needed the function `proc` should be used. In the example below the function `nyqpol` takes one argument (a rational function in the global variable `s`) and returns its corresponding Nyquist polynomial:

```
nyqpol:=proc(G)
   local A,f1,f2,w:
   A:=subs(s=I*w,G):
   f1:=evalc(numer(x1-evalc(Re(A)))):
   f2:=evalc(numer(x2-evalc(Im(A)))):
   grobner[finduni](x1,[f1,f2],[w,x1]);
end:
```

LISTS AND SETS

A list is of the form `[op1, ..., opn]` and a set `{op1,...,opn}`. The `i`:th element of the list `L` is addressed `L[i]`. Examples:

```
> L:=[17,f,ln(y),[1,9]]:  L[2];
                                  f

> L[1..3];
                           17, f, ln(y)

> S:={L[1],L[4][2]};
                           S := {9, 17}
```

`> map(fcn,L);` means "apply the function `fcn` on every element in the list `L`". This also works for sets, vectors, etc. If `fcn` takes several arguments you type

`> map(fkn,L,arg2,arg3);`

If `arg2, arg3, ...` are lists, use `zip` instead.

ASSIGNMENT AND EVALUATION

Assignment is made with `:=` whereas the equality sign `=` is used for equations.
To check the current value of a variable type its name: `> a;`
The exceptions are vectors, matrices, arrays, tables and procedures for which the `print` or `op` command has to be used: `> print(A);` or `> op(A);`

To check if the variable `a` has been assigned a value you can type `> assigned(a);`
To unassign the variables `a` and `b` write `> unassign('a,b');`
The `unassign` function first has to be loaded with `> readlib(unassign):`

The main evaluation rule in Maple is that every expression is evaluated as far as possible, i.e. all variables in the expression are substituted by their value if they have been assigned one. Thus the order in which assignments are made is of importance, as is illustrated by the following example.

```
> a:=b;
                               a := b
> b:=3:
> c:=b;
                               c := 3
> b:=4:
> [a,b,c];
                             [4, 4, 3]

> unassign('b'): [a,b,c];
                             [b, b, 3]
```

```
> [0$4];
                              [0, 0, 0, 0]

> {$1..5};
                             {1, 2, 3, 4, 5}

> diff(x(t),t$6): #Sixth derivative of x(t) w.r.t. t
> x.(1..4);
                             x1, x2, x3, x4

> [x.(7-i)$i=0..6];
                      [x7, x6, x5, x4, x3, x2, x1]

> (sin@arcsin)(x);
                                   x
```

"The latest expression" is called by `"`. The latest expression but one is `""` etc. For a history encompassing more than the three latest expressions returned, use the command `history`.

The function `convert(expr,type)` converts between different data types.

In order to clear the screen, type `> !clear;` (Thus `!` means "escape to host".)

For translation of an expression into LaTeX, type `> latex(expr,foo);`
The second argument, which is optional, is a file name.
There are similar functions for C and FORTRAN.

HELP FACILITIES

`> ?fnamn;` displays a help message about `fnamn` and a general help message can be obtained by typing a sole question mark: `> ?`
The call `example(fcn)` gives an example of the use of the function `fcn`:

```
> example(dsolve);

            dsolve({sys, z(0) = 1, y(0) = 0}, fcns, series)
```

The call `> anames();` returns a list of all names that have been assigned a value.

The best way to debug a function is to use `printlevel`, which is a system variable.

READING AND SAVING FILES

To save all data on the file `foo` type `> save(foo);`

If you just want to save some assignments type `> save(name1,name2,foo);`

The command used for reading from a file is called `read`: `> read(foo);`

A Quick Manual to Maple version V

Krister Forsman, Dept. Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden
email: `krister@isy.liu.se`

1992-01-27

Preface

Maple is a nice and powerful computer algebra system. There is also a very readable manual for Maple. However, many people do not do symbolic calculations every day and may need to use the program only occasionally. This is a manual for those people. It is totally incomplete, it has examples instead of general syntax description of the commands, it only accounts for a small fraction of the existing functions.

The basic idea is: look here for the function you need, and if there is not enough information about it here (there typically won't be) use the on-line documentation or read the real manual. **Caveat**: since the functions are mostly explained by examples, many of them have more facilities and are more powerful than it appears here.

STARTING MAPLE

Start Maple by typing `maple`. If you wish the file `foo` to be read when starting maple, type `maple < foo`. If there is a file called `.mapleinit` in your home directory it is automatically read. There is also an X-windows version of Maple, which is started with the command `xmaple`.

BASIC FACTS

The prompter is `>`
Each command is terminated by a semi colon (`;`) or a colon (`:`).
`;` $\Rightarrow$ the result is showed. `:` $\Rightarrow$ the result is not showed.

Maple is case sensitive.

Comments start with `#`

$e^{\pi i}$ would be `> E^(Pi*I);` in Maple. ∞ is called `infinity`.

The iteration operator, `$`, is very useful. The dot-operator (`.`) denotes concatenation. The composition operator is written `@`.