What is Maple and how to start it up?

- Maple: Canadian product (Waterloo Maple Software Corporation)
- High-level interactive software package for mathematical manipulation and problem solving
- Enables easy manipulation of functions and algebraic expressions
- Particularly tailored to do **symboli**c computations

To start up Maple on CDF type
`% maple`
or
`% xmaple &`
at the Unix shell prompt.
The `maple` command must be run on the machine `werewolf.cdf` and starts up Maple in your current window (most likely an `xterm` or any ASCII terminal); it gives you a simple and light ASCII interface; it can be run remotely over a telnet/ssh session.
The `xmaple` command starts up Maple in a new window of the console you are on, assuming you are running Xwindows; the new window has various menus and buttons, and one (or more) inner windows, called **worksheets**; it gives you a more powerful, but also complex and heavy interface.
The standard Maple prompt (appears on the xterm or ASCII terminals, or on the worksheets) is `>`
In the following, whatever follows `>` on a line starting with `>` is typed by the user at the Maple prompt.

To quit Maple, type
`> quit`

 © C. Christara, 2001-2006

An example of Maple code

```
solve(a*x^2+b*x+c = 0, x); #solve a quadratic equation
                           #with respect to x
a := 1; b := 5; c := 4;    #give values to variables
                           #a, b, c
solve(a*x^2+b*x+c = 0, x); #solve again
```

The above piece of code, first gives us the formulae for the roots, ρ_1 and ρ_2, of the quadratic equation $ax^2 + bx + c = 0$ in terms of a, b and c (symbolic answer), then sets the variables a, b and c to the values 1, 5 and 4, respectively, then gives us the two roots, -1 and -4 (numerical answer), of $x^2 + 5x + 4 = 0$.

- The assignment operator is `:=`
- The `=` sign is used to denote equality of the two sides and to define an equation.
- Whatever follows the # symbol on a line is a comment.
- The semicolon (;) and the colon (:) both denote the end of a Maple statement.
  The semicolon causes the result of the statement to be echoed. The colon suppresses echoing. (See example, further.)
- When a Maple statement is long to fit on one line, continue on the next line, without giving a colon or a semicolon.
- The newline (return) does **not** denote the end of a Maple statement.
- Maple is case sensitive, i.e. variable A is different than variable a.

 © C. Christara, 2001-2006

Using Maple interactively

Recall that whatever follows `>` on a line starting with `>` is typed by the user at the Maple prompt. The rest is output echoed by Maple.

```
> solve(a*x^2+b*x+c = 0, x);
                2         1/2             2         1/2
         -b + (b  - 4 a c)         -b - (b  - 4 a c)
    1/2 --------------------, 1/2 --------------------
                a                         a
> a := 1; b := 5; c := 4;
                          a := 1
                          b := 5
                          c := 4
> solve(a*x**2+b*x+c = 0, x);
                          -1, -4
> r := solve(a*x**2+b*x+c = 0, x);
                       r := -1, -4
> r[1];
                            -1
> r[2];
                            -4
> print(r);
                          -1, -4
> print(r[1]);
                            -1
> a;
                             1
> unassign('a', 'b', 'c');
> a;
                             a
> y := solve(a*x^2+b*x+c = 0, x):
```

 © C. Christara, 2001-2006

```
> y[1];
                          2         1/2
                      -b + (b  - 4 a c)
                  1/2 --------------------
                             a
> y[2];
                          2         1/2
                      -b - (b  - 4 a c)
                  1/2 --------------------
                             a
```

- Exponentiation is carried by the hat `^` or the double star `**` symbols.
- The built-in Maple procedure `solve` solves (linear or nonlinear) equations, with respect to the given unknowns. (In the above example, it solves a quadratic equation w.r.t. x.)
- When `a, b, c` are not assigned a numeric value, `solve` gives the answer in terms of `a, b, c` (symbolic answer).
  When `a, b, c` are assigned a numeric value, `solve` gives the numeric answer.
- To save the answer (symbolic or numeric), assign the result of the procedure to a variable (here `r` or `y`).
  Since the answer consists of two numbers, `solve` returns a **list** with two elements.
  To refer to an element of the list, use the index of the element in the list inside square brackets.
- To see the value of a variable, use `print(variable);` or just type the name of the variable followed by semicolon.
- To clear the variable name `a` use `unassign('a');` (note the **single-quotes**, also called **apostrophe**), while to clear all variable names use `restart;`.

 © C. Christara, 2001-2006

```
> exp(0);
                              1
> exp(1);
                            exp(1)
> exp(1.);
                         2.718281828
> evalf(exp(1));
                         2.718281828
> Digits := 20:
> exp(1.);
                   2.7182818284590452354
> Digits := 10:
> sin(Pi);
                              0
> sin(Pi/2);
                              1
> sin(Pi/4);
                             1/2
                          1/2 2
> sin(Pi/4.);
                    sin(.2500000000 Pi)
> evalf(sin(Pi/4));
                         .7071067810
> sqrt(-1);
                              I
> a := 2+3*I;
                        a := 2 + 3 I
> b := 5+7*I;
                        b := 5 + 7 I
> a+b;
                          7 + 10 I
```



```
> a := 3/2: b := 5: c := 4:
> a+b;
                             13/2
> z := solve(a*x^2+b*x+c = 0, x);
                        z := -4/3, -2
```

- Maple has all the standard mathematical functions, e.g. exp, ln, log (defaults to ln, but can use other base, log[b]), log10, sin, cos, tan, arcsin, arccos, arctan, sqrt, sinh, cosh, abs, max, min, etc, and many more built-in functions that compute a variety of mathematical results.
- When Maple can't find a simplified symbolic form for the value of an expression, it takes the expression itself as a symbolic form.
  To get a numeric value (in the decimal system), we can either explicitly request it by the `evalf` (evaluate using floating-point arithmetic) function, or by passing the argument in decimal format.
- By default, Maple uses 10 decimal digits for floating-point arithmetic. By assigning a value other than 10 to the reserved variable `Digits` the default number of decimal digits can be changed at the user's choice; it can be brought back to the default value by `Digits := 10:`
- The variable `Pi` is set to the value of the number π, and is **protected**, i.e. the user can not overwrite it.
  Similarly, the variable `I` is set to the value $\sqrt{-1}$, and is protected.
- Complex numbers are handled just as easily as real ones. Complex numbers are output in the form a + b I.
- Maple can work with fractions, just as humans can do.

Differentiation, integration, etc.

```
> # differentiate a function
> diff(sin(x), x);
                            cos(x)
> diff(sin(x), x$2);
                           -sin(x)
> diff(sin(x), x$3);
                           -cos(x)
> diff(sin(x), y);
                              0
> diff(x^4 * y^5, x$2);
                             2  5
                          12 x  y
> diff(x^4 * y^5, x$2, y);
                             2  4
                          60 x  y
> # integrate a function
> int(tan(x), x);
                        -ln(cos(x))
> # integrate a function in an interval
> int(1+x^2, x = 1..2);
                            10/3
```



```
> # expand a product or a math function
> expand((a+b)^2);
                        2           2
                       a  + 2 a b + b
> expand((x+1)^5);
                5      4       3       2
               x  + 5 x  + 10 x  + 10 x  + 5 x + 1
> expand(sin(x+y));
                  sin(x) cos(y) + cos(x) sin(y)
> # factor a polynomial
> factor(a^2 + 2*a*b + b^2);
                              2
                           (a + b)
> # simplify an expression
> simplify((x^5-1)/(x-1));
                       4    3    2
                      x  + x  + x  + x + 1
> 4^(1/2)+3;
                           1/2
                          4    + 3
> simplify(4^(1/2)+3);
                             5
> simplify(sin(x)^2+cos(x)^2);
                             1
```

Do loops

The most typical syntax of the do loop is of the form

```
for i from j by k to l do
    statements
od;
```

where i, the loop counter, starts from j, and proceeds to l, with stepsize k. If k is missing, i.e. the do statement looks like `for i from j to l`, the stepsize takes the default value 1. If j is missing, i.e. the do statement looks like `for i by k to l`, the starting value takes the default value 1. Both j and k may be missing. Note that none of the i, j, k and l variables needs to be integer.

Examples: What values does `i` take in the following loops, and what value does it have after the loop executes?

```
> for i to 3 do print(i); od; print(i);
```

1, 2, 3 after loop value 4

```
> for i by 2 to 6 do print(i); od; print(i);
```

1, 3, 5, after loop value 7

```
> for i from 1.5 by 1.5 to 3 do print(i); od; print(i);
```

1.5, 3.0, after loop value 4.5

```
> for i from 3.5 by -1.5 to 0 do print(i); od; print(i);
```

3.5, 2.0, 0.5, after loop value -1.0

```
> for i from 1.5 by -1.5 to 1.5 do print(i); od; print(i);
```

1.5, after loop value 0

```
> for i from 1.5 by -1.5 to 3 do print(i); od; print(i);
```

No values inside the loop! The statements inside the loop (whatever they were) do not get executed.
After loop value 1.5

 © C. Christara, 2001-2006

A two-level nested loop: (assume `n` is already assigned a value)

```
for i to n do
    statements that possibly use i
    for j to n do
        statements that possibly use i and j
    od
od
```

The `break` statement can be used to terminate the (innermost) loop prematurely. (Note: earlier versions of Maple, e.g. Maple V, did not have this keyword.)

 © C. Christara, 2001-2006

If (conditional) statement, relational and logical operators

The general form of the if statement is

```
if conditional expression then
    statements
elif conditional expression then
    statements
else
    statements
fi
```

The statements following the (first) if are executed, if the (first) conditional expression of the if statement is true.
The statements following each `elif` are executed, if the expression of the respective branch is true and none of the previous branches resulted in execution. The `else` branch is executed if none of the previous branches resulted in execution. The `else` and `elif` parts are optional. Zero or more `elif` parts can be used as well as nested `if`'s.
Each conditional expression is a boolean expression formed by
the relational operators `=`, `<`, `>`, `<=`, `>=`, or `<>`,
the logical operators `and`, `or`, `xor`, `implies` or `not`,
and the logical names `true`, `false` and `FAIL`.
`FAIL` is used in Maple to denote an unknown or undetermined value. As a boolean expression in an if statement, `FAIL` behaves like `false`.

Examples of if statements:

```
if (i < j) and (i > j-5) then print(1);
elif i < j then print(2);
else print(3); fi;
```

Which branch will be executed if the above if statement is preceeded by
`i := 2: j := 7:` or `i := 3: j := 7:` or `i := 7: j := 7:`

 © C. Christara, 2001-2006

Assume x and y have not been assigned a value.

```
> if (x = x) then print(1); else print(2); fi;
                                1
> if (x = y) then print(1); else print(2); fi;
                                2
> x := 1:
> if (x = y) then print(1); else print(2); fi;
                                2
> y := 1:
> if (x = y) then print(1); else print(2); fi;
                                1
```

Assume a and b have not been assigned a value.

```
> if (a^2 + 2*a*b + b^2 = (a+b)^2) then print(1);
> else print(2); fi;
                                2
> if (a^2 + 2*a*b + b^2 = expand((a+b)^2)) then print(1);
> else print(2); fi;
                                1
> a := 1: b := 2:
> if (a^2 + 2*a*b + b^2 = (a+b)^2) then print(1);
> else print(2); fi;
                                1
> if (a^2 - 2*a*b + b^2 = (a-b)^2) then print(1);
> else print(2); fi;
                                1
> a := 15.6: b := 15.7: Digits := 3:
> if (a^2 - 2*a*b + b^2 = (a-b)^2) then print(1);
> else print(2); fi;
                                2
> a^2 - 2*a*b + b^2;
                               -1.
> (a-b)^2;
                               .01
```

 © C. Christara, 2001-2006

Arrays, Matrices, Lists

Matrices in Maple are represented by two-dimensional arrays with row and column indices starting from 1.

The simplest way to set a matrix is by using the `array` function. For example,

```
> A := array([[1, 2, 2], [3, 2, 6]]);
                [1    2    2]
          A := [             ]
                [3    2    6]
```

creates a 3×2 array with the entries set as shown above, and assings it to `A`. Some arrays with specific properties can be set without listing the entries explicitly. For example,

```
> E := array(identity, 1..2, 1..2);
         E := array(identity, 1 .. 2, 1 .. 2, [])
> print(E);
                [1    0]
                [      ]
                [0    1]
```

To refer to a matrix entry that has been assigned earlier, type the matrix variable name followed by `[i, j]`, where `i` is the row index and `j` is the column index of the entry.

```
> A[2, 1];
                      3
```

Maple has a Linear Algebra package, called `linalg`, which includes several matrix manipulation commands. To use this package, we must first invoke it with

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

Then, we can call any function in the package.

 © C. Christara, 2001-2006

This package gives us another way to set a matrix by using the `matrix` function, and allows us to use many built-in functions for matrix manipulation.

```
> with(linalg):
> # define a matrix
> A := matrix([[1, x], [2, y]]);
                    [1    x]
              A := [       ]
                    [2    y]
> # find the determinant of A
> det(A);
                 y - 2 x
> # find the inverse of A
> inverse(A);
      [     y            x     ]
      [- --------     -------- ]
      [  -y + 2 x     -y + 2 x ]
      [                        ]
      [    2              1    ]
      [ --------     - --------]
      [ -y + 2 x       -y + 2 x]
```

Lists can be viewed as one-dimensional arrays. For example,

```
> L := [1, 2, 3, x, x, y]:
```

defines a (ordered) list or 6 elements, as shown above. To refer to an element of a list, type the name of the list followed by the index of the element in square brackets.

```
> L[2];
                      2
> L[5];
                      x
```

 © C. Christara, 2001-2006

We can also get the elements in a certain range of indices.

```
> L[3..5];
                [3, x, x]
```

We can have nested lists, with elements lists themselves.

```
> L := [1, [2, 3], [4, [5, 6], 7], 8, 9]:
> L[2];
                 [2, 3]
> L[2][1];
                      2
> L[3][2][1];
                      5
```

To append or prepend elements to a list, or extract sublists and form new lists, we use the `op` function.

```
> L := [op(L), 10];
        L := [1, [2, 3], [4, [5, 6], 7], 8, 9, 10]
> L[6];
                     10
> L := [0, op(L)];
        L := [0, 1, [2, 3], [4, [5, 6], 7], 8, 9, 10]
> L[1];
                      0
> L[6];
                      9
> L[5..6];
                [8, 9]
> L := [-1, op(L[1..6]), 11];
     L := [-1, 0, 1, [2, 3], [4, [5, 6], 7], 8, 9, 11]
```

 © C. Christara, 2001-2006

Procedures and scripts in Maple

Besides the built-in functions and procedures in Maple, we can have user-defined procedures. Procedures may have one or more input parameters and may return one or more output variables. User-defined procedures are often saved in (text) files (edited by standard text editors), named at the choice of the user. One or more procedures can be in the same file.

A sequence of Maple statements is often called a script. Scripts have neither formal input parameters, nor formal output variables. Scripts can also be saved in (text) files (edited by standard text editors).

- To call any user-defined procedure saved in some file, the file must be read by
  `> read ‘myfile‘;`
  or
  `> read ‘mydir/myfile‘;`

  Note the **backquotes**. Maple 7 allows the use of doublequotes in place of backquotes.
- To call (execute) a user-defined procedure (assuming the appropriate file has been read), type the name of the procedure, followed by the desired input and output arguments in parentheses.
- To save the default output variable in a name, precede the call by the output variable name followed by the `:=` sign.
- To execute the statements of some script saved in some file, just read the file as above.

 © C. Christara, 2001-2006

Example:

Assume the following is in file `mortgage.mp` in the current directory

```
mortgage := proc(D, r, A, n, f)
local L, i;
L := [D];
for i from 1 to n do
    L := [op(L), L[i] + r*L[i] - A];
od;
if type(L[n+1], numeric) then
    if L[n+1] > 0 then f := "unpaid";
    else f := "paid"; fi;
else f := FAIL; fi;
return(L);
end;
```

and the following in file `script.mp` in the current directory

```
read ‘mortgage.mp‘:
unassign(’f’);
mortgage(D, r, A, 2, f);
f;
unassign(’f’);
L := mortgage(10000, 0.06, 1000, 10, f);
f;
unassign(’f’);
L := mortgage(10000, 0.06, 1000, 20, f);
f;
```

In Maple, we have

 © C. Christara, 2001-2006

```
> Digits := 6:
> read ‘script.mp‘;
     [D, D + r D - A, D + r D - 2 A + r (D + r D - A)]
                          FAIL

L := [10000, 9600.0, 9176.0, 8726.56, 8250.15, 7745.16,
       7209.87, 6642.46, 6041.01, 5403.47, 4727.68]
                        "unpaid"

L := [10000, 9600.0, 9176.0, 8726.56, 8250.15, 7745.16,
       7209.87, 6642.46, 6041.01, 5403.47, 4727.68,
       4011.34, 3252.02, 2447.14, 1593.97, 689.61,
      -269.013,-1285.15,-2362.26,-3504.00,-4714.24]
                        "paid"
> L[2];
                        9600.0
```

- The `mortgage` procedure is a modification of the one given in the notes. This modification, besides calculating the remaining debt after n years of payments of amount A, with interest rate r, and initial debt D, it also keeps a history of debt amounts for each of the n years (prepended by the original amount), in the form of a list L, where L[i] is the debt amount before year i; it also sets a flag f, whether the debt is paid or not.
- When the procedure is used for symbolic calculations, the flag is set to the predefined value FAIL.
- The default output variable is the list of debt amounts, representing the history of debt amounts. The flag f is also an output variable (but not the default one). It must be unassigned before it is passed to a call to `mortgage`.

 © C. Christara, 2001-2006

A simple plot

If we want a plot to visualize the reduction of the debt as the years pass, assuming we have run the previous script, we can ask for the plot as follows:

```
> with(plots):
> interface(plotdevice = x11);
> listplot(L);
> interface(plotdevice = default);
```

Assuming we are running Xwindows, the above will open up a new window, with the graph of L versus the year index. Then, the plotdevice will be restored to the default one (usually `char`, an ASCII text screen).

If we prefer to get a postscript file of the plot, we ask

```
> interface(plotdevice = ps, plotoutput = ‘myfile.ps‘);
> listplot(L);
> plotsetup(default);
```

The above will write to file `myfile.ps` (will **overwrite!**) the postscript code that produces the plot. Then, both the plotdevice and the output will be restored to the default ones (usually the text screen and the terminal, respectively).

We will not view the plot. We can, though, print it by

```
% lpr myfile.ps
```

on the Unix prompt.

If we just want an ASCII approximation to the plot, just type

```
> listplot(L);
```

 © C. Christara, 2001-2006

On-line help

Maple offers a lot of on-line help. Type

```
> ?topic
```

or

```
> ?topic, subtopic;
```

to get information on any Maple topic or subtopic, respectively. For example,

```
> ?linalg
```

will give you general information on the Linear Algebra package and the names of all its subtopics and

```
> ?linalg, inverse;
```

will tell you how to use the `inverse` built-in function that gives the inverse of a matrix. Similarly, you can use `help(topic)` or `help(topic, subtopic)`.

 © C. Christara, 2001-2006