Maple

David J. Scott

d.scott@auckland.ac.nz

Department of Statistics, University of Auckland

Outline

- Introduction
- A sample session
- Practicalities
- Basic operations
- Calculus
- Procedures

Resources

- The Maple website is the major resource:
  http://www.maplesoft.com/
- There are downloadable manuals on the website. You need to register, but can then download the manuals without cost:
  http://www.maplesoft.com/documentation%5Fcenter/
- Yong Wang has some Maple examples on his website:
  http://www.stat.auckland.ac.nz/~yongwang/782/week10.html and
  http://www.stat.auckland.ac.nz/~yongwang/782/week11.html
- This website has a number of useful-looking tutorials:
  http://www.gmi.edu/acad/scimath/appmath/maple/

Introduction

What is Maple?

- It is a computer algebra system
- Various names are used for this sort of computation: symbolic computation, symbolic manipulation, formula manipulation, computer algebra
- Symbolic computation differs from numeric computation in that it manipulates symbols representing mathematical objects
- The mathematical objects may represent number such as integers, real numbers and complex numbers, but may also be such things as polynomials, functions, algebraic structures such as groups, rings and fields
- The manipulations are carried out exactly using the rules of algebra

Examples of Computations

- The sort of manipulations carried out are:
  - factorisation of polynomials
  - differentiation and integration
  - series expansions
  - determination of limits
  - simplification of expressions
  - analytic solution of differential equations
  - exact solution of systems of equations
- Most computer algebra systems do also allow the numerical evaluation of expressions
- Sometimes computer algebra systems will produce code in a language such as Fortran to numerically evaluate a symbolic expression

A Sample Session

Numerical Calculation

- Maple uses exact arithmetic, not floating point approximations

```
> 105/25;
                                          21/5
> interface(screenwidth=60);
                                  79

> 100!;
933262154439441526816992388562667004907159682643816214 68\
    59296389521759999322991560894146397615651828625369 79\
    20827223758251185210916864000000000000000000000000

> interface(screenwidth=79);
                                           60
```

Built-in Constants and Functions

- Standard mathematical constants are built-in, likewise standard functions such as the trigonometric functions

```
> sin(Pi/2);
```

$$1$$

- Numerical evaluation to any accuracy is possible

```
> evalf(Pi,25);
```

$$3.141592653589793238462643$$

Algebraic Manipulation

- Common manipulations of algebraic expressions are included

```
> expand((1+x)^2);
                                          2
                              1 + 2 x + x

> factor(%);
                                        2
                                 (1 + x)
```

- This includes trigonometric identities

```
> expand(sin(a+b));
                    sin(a) cos(b) + cos(a) sin(b)
> simplify((1+sin(x)+cos(x))/(1+sin(x)-cos(x)));
                               1 + cos(x)
                               ----------
                                 sin(x)
```

Solution of Equations

- Single equations can be solved

```
> solve(x^2-x=1,x);
                                  1/2          1/2
                                 5            5
                           1/2 + ----, 1/2 - ----
                                  2            2
```

- Sets of equations can likewise be solved

Plotting

- Functions can be plotted
- 3-d plots are possible, including animation of plots

```
plot(sin(x),x=-Pi..2*Pi);
expr:=Int(x^2*sin(x-a),x);
answer:=value(expr);
plot3d(answer,x=-Pi..Pi,a=0..1);
with(plots):
animate(answer,x=-Pi..Pi,a=0..1);
```

Limits

- Limits can be evaluated

```
> limit((x^2-4)/(x-2),x=2);
                                        4

> limit(tan(x),x=Pi/2);
                                    undefined
```

- Limits can be taken from a particular side

```
> limit(tan(x),x=Pi/2,`left`);
                                    infinity
```

Differentiation

- Differentiation is a mechanical process, ideal for a computer

```
> Diff(exp(-x^2),x);
```

```
                                  d           2
                                 --- exp(-x~ )
                                 dx~
```

```
> value(%);
```

```
                                              2
                                -2 x~ exp(-x~ )
```

Integration

- Integration is much more difficult
- Maple knows all the rules you were taught in calculus classes

```
> Int(x^2*sin(x),x);
                                    /
                                   |    2
                                   |  x~  sin(x~) dx~
                                   |
                                  /

> value(%);
                          2
                       -x~  cos(x~) + 2 cos(x~) + 2 x~ sin(x~)
```

- The integration constant is omitted

Integration

- Definite integrals may be evaluated

```
> Int(x^2*sin(x),x=0..1);
                                  1
                                 /
                                |     2
                                |    x~   sin(x~) dx~
                                |
                               /
                                 0

> value(%);
                               cos(1) + 2 sin(1) - 2

> evalf(%%,10);
                                    0.223244276
```

Practicalities

Starting, Stopping etc

- Start the gui version of maple on unix using `xmaple`
- The text version is started simply with `maple`
- Terminate a session with `quit` or in the gui version by using a menu, or a keyboard shortcut
- Obtain help using `help` or `?`
- For help on a particular topic, `help(topic)` or `?topic`
- Help is available from menus in the gui version

Input and Output

- Different input and output displays are possible in the gui version
- Inputs may be either ascii (similar to TEX) or in Maple input format
- Output may be in Maple notation, as character, or in LATEX format
- The worksheet can include text so that the commands used can be documented
- The text can be formatted with titles, headings, different fonts etc
- The command `latex` will write a single expression in LATEX

Use of `latex` *Command*

- Here is a Maple command which produces an integral

```
> expr:=int(exp(-x^2)*ln(x),x);
                              /
                             |         2
                   expr :=   |  exp(-x ) ln(x) dx
                             |
                            /

> latex(expr);
\int \!{e^{-{x}^{2}}}\ln  \left( x \right) {dx}
```

- Which produces

$$\int e^{-x^2} \ln\left(x\right) dx$$

Output

- Maple output as LaTeX requires style files
- Different versions of Maple require different style files
- It appears now a Maple class is available
- The required style files ship with Maple
- On stat12, the files are in `/usr/local/maple9.5/etc`
- Copy the files to your `TEXINPUTS` directory
- Maple will create `.eps` files for any plots in the output, and will include LaTeXcode to include any plots in the typeset document

Batch Processing

- You can use redirection in unix to read Maple commands from a file and save the output in a file
- When in Maple, you can read a file of Maple commands in with

  `> read `*filename*``

- To ensure that commands are included along with the files, set the interface variable `echo` to 2, using

  `interface(echo=2)`

Maple Syntax

- Arguments are given in parentheses `( . . . )`
- Square brackets `[ . . . ]` are reserved for grouping operations: vectors, matrices and lists
- Commands must end with a semicolon (`;`) or colon (`:`)
- The result is displayed if the command is ended with a semicolon, but not if ended with a colon
- Multiplication is represented by an asterisk (`*`): enter `2*x*y` to evaluate $2xy$, not `2xy`
- Powers are entered with `^`
- Use brackets to ensure the correct order of operations

Basic Operations

Exact Calculations

- Basic calculations are fairly obvious

```
> 12315/35;
                                        2463/7
> (22431)*(832748)*(387281);
                                   7234165243235028
> sqrt(27);
                                           1/2
                                        3 3
```

- Expressions are not routinely simplified, but simplification can be explicitly requested

```
> 8^(2/3);
                                         (2/3)
                                        8
> simplify(%);
                                           4
```

Determining Roots

- Care is needed with the order of operations

```
> (-27/64)^2/3;
                                        243
                                        ----
                                        4096
> (-27/64)^(2/3);
                                        2/3   1/3
                                   (-27)    64
                                   --------------
                                         64
```

Determining Roots

- `simplify` doesn't always behave as expected, `surd` is useful in calculating roots

```
> simplify((-27/64)^(2/3));
                                   1/2   2
                          9 (1 + 3     I)
                          ---------------
                                64
> surd((-27/64),3);
                               -3/4
> surd((-27/64),3)^2;
                               9/16
```

- `I` here represents $\sqrt{-1}$

Built-in Constants

- Besides $\sqrt{-1}$, denoted by `I`, Maple has
  - $e \approx 2.71828$ denoted by `exp(1)`
  - $\pi \approx 3.14159$ denoted by `Pi`
  - ∞ denoted by `infinity`
  - Euler's constant $\gamma \approx 0.577216$ denoted by `gamma`

Built-in Functions

- Built-in functions include
  - the exponential function `exp(x)`
  - the natural logarithm `ln(x)`
  - the absolute value function `abs(x)`
  - trigonometric functions `sin(x), cos(x), tan(x), sec(x), csc(x), cot(x)`
  - inverse trigonometric functions `arcsin(x), arccos(x), arctan(x), arcsec(x), arccsc(x), arccot(x)`
  - hyperbolic trigonometric functions and their inverses
  - special functions such as Bessel functions (including `BesselI(v,x), BesselJ(v,x), BesselK(v,x)`)

Algebraic Operations on Expressions

- `factor(expression)`
- `expand(expression)`
- `simplify(expression)`
- `normal(expression)` provides a basic simplification of rational functions. The numerator and denominator are relatively prime polynomials with integer coefficients
- `convert(expression,parfrac,variable)` computes the partial fraction decomposition of `expression` in terms of the variable `variable`
- `convert(expression,exp)` converts a trigonometric expression to an exponential expression
- Many other conversions are possible—see `?convert`

Algebraic Operations

```
> factor(12*x^2+27*x*y-84*y^2);
                              3 (x + 4 y) (4 x - 7 y)

> expand((x+y)^2*(3*x-y)^3);
                   5        4          3   2         2   3          4    5
               27 x  + 27 x  y - 18 x  y  - 10 x  y  + 7 x y  - y

> simplify(cos(x)^5+sin(x)^4+2*cos(x)^2);
                                    5             4
                               cos(x)  + 1 + cos(x)
```

Algebraic Operations

```
> normal((x^2-y^2)/(x-y)^3);
                                      x + y
                                    --------
                                          2
                                    (x - y)

> normal((x^2-y^2)/(x-y)^3,`expanded`);
                                       x + y
                                 ---------------
                                  2             2
                                 x  - 2 x y + y
```

Algebraic Operations

```
> convert(1/((x-3)*(x-1)),parfrac);
                                  1           1
                            - --------- + ---------
                              2 (x - 1)   2 (x - 3)

> convert(sin(x),exp);
                                 /                1    \
                         -1/2 I |exp(x I) - --------|
                                 \             exp(x I)/
```

Naming and Evaluating Expressions

- Objects can be named
- Syntax is `name:=expression`
- Reduces typing, expressions can be referenced throughout your Maple session
- Expressions can be evaluated using `subs` or `eval`
- Numerical evaluation uses `evalf`

Evaluating Expressions

```
> exp1:=x^2;
                                  2
                          exp1 := x
> exp1:=x^3;
                                  3
                          exp1 := x
> exp:=x^2;
Error, attempting to assign to `exp` which is protected
> f:=(x^3+2*x^2)/(x^3+x^2-4*x-4);
                               3      2
                              x  + 2 x
                    f := -----------------
                           3    2
                          x  + x  - 4 x - 4
> subs(x=4,f);
                               8/5
```

Expressions

- `expression1:=expression2;` causes `expression1` to be set to `expression2` and evaluated
- `expression1:='expression2';` causes `expression1` to be set to `expression2`, overwriting any previous assignment
- An assignment can also be annulled using `unassign('expression')`
- `restart` clears Maple's internal memory of all previously defined symbols

Evaluating Expressions

```
> x:=2;
```

$$x := 2$$

```
> exp1:=x^2;
```

$$exp1 := 4$$

```
> exp1:='x^3';
```

$$exp1 := x^3$$

```
> exp1;
```

$$8$$

```
> unassign('exp1');
> exp1;
```

$$exp1$$

Defining and Evaluating Functions

- Functions, expressions and graphics can be given any name that is not a built-in function or command
- An elementary function of a single variable $y = f(x)$ is typically defined using the form

```
f:=x->expression in x
```

- Then `f(x)` evaluates the function f at x
- `subs` can also be used to evaluate a function
- The function can be evaluated at some set of values using a list
- A list takes the form `[a1,a2,...,an]`

Defining and Evaluating Functions

```
> f:=x->x/(x^2+1);
                                 x
                  f := x -> ------
                                2
                               x  + 1
> f(3);
                           3/10
> f(3+h);
                         3 + h
                      ------------
                                2
                      (3 + h)  + 1
```

Defining and Evaluating Functions

```
> n1:=simplify((f(3+h)-f(3))/h);
                                  8 + 3 h
                    n1 := - -------------------
                                           2
                            10 (10 + 6 h + h )
> subs(h=0,n1);
                                  -2
                                  --
                                  25
> map(f,[0,1,2,3]);
                       [0, 1/2, 2/5, 3/10]
> [seq(f(n),n=0..3)];
                       [0, 1/2, 2/5, 3/10]
```

Solving Equations

- `solve(equation)` can be used to solve an equation with a single unknown
- `solve(equation,variable)` can be used to make `variable` the subject of the equation
- These extend to systems of equations
- A system of equations is specified using a set
- A set takes the form `{a1,a2,...,an}`

Solving Equations

```
> solve(x^3+x^2+x+1=0);
                          -1, I, -I
> solve(y=(x-5)^3/8,x);
   (1/3)           (1/3)     1/2  (1/3)
2 y      + 5, -y        + 3    y        I + 5,


      (1/3)     1/2  (1/3)
  -y        - 3    y        I + 5
> solve(y=(x-5)^3/8,x)[1];
                              (1/3)
                         2 y      + 5
> sys:={3*x-y=4,x+y=2};
               sys := {3 x - y = 4, x + y = 2}
> sols:=solve(sys);
                  sols := {x = 3/2, y = 1/2}
> subs(sols,sys);
                        {4 = 4, 2 = 2}
```

Calculus

Limits

- Syntax is `limit(expression,variable=value,direction)`

```
> limit(sin(x)/x,x=0);
                                  1

> limit((1+a/x)^x,x=infinity);
                                exp(a)
```

Differentiation

- `D` and `Diff` are used to differentiate functions
- `diff(f(x),x)` computes and returns $f'(x) = df/dx$
- `D(f)(x)` computes and returns $f'(x) = df/dx$
- `diff(f(x),x$n)` computes and returns $f^{(n)}(x) = d^n f/dx^n$
- `(D@@n)(f)(x)` computes and returns $f^{(n)}(x) = d^n f/dx^n$
- Maple knows all the usual differentiation rules: the product rule, the quotient rule, the chain rule

Differentiation

```
> diff(x^4+4/3*x^3-3*x^2,x);
                                3      2
                             4 x  + 4 x  - 6 x

> diff(x^4+4/3*x^3-3*x^2,x$2);
                                  2
                              12 x  + 8 x - 6
```

Integration

- Syntax is `int(expression,variable)` for an indefinite integral or `int(expression,variable,a..b)` for a definite integral
- The abitrary integration constant is omitted when displaying the indefinite integral

```
> int(1/x^2*exp(1/x),x);
                              -exp(1/x)
```

Integration

- If the integral is a known mathematical function with no closed form, the function will be given
- Otherwise the integral will be returned unevaluated

```
> int(sin(x)/x,x);
                              Si(x)

> int( exp(-x^2)*ln(x), x );
                        /
                       |        2
                       |  exp(-x ) ln(x) dx
                       |
                      /
```

Integration

- Maple doesn't know the normal distribution function, but does know the error function which is very closely related

```
> int(1/sqrt(2*Pi)*exp(-(1/2)*x^2),x=-infinity..a);
                          1/2
                         2    a
                1/2 erf(------) + 1/2
                           2
> int(1/sqrt(2*Pi)*exp(-(1/2)*x^2),x=-infinity..1.96);
                      0.9750021049
```

Procedures

Procedures

- The Maple equivalent of an **R** function is called a procedure
- Procedures can have local and global variables
- They can be defined recursively
- Maple has a full set of control structures: `if . . else`, `for`, `while`
- The type of a variable can be specified
- Syntax of control structures appears to vary from version to version
- Maple can produce Fortran, C and Matlab code, which can be stored in a file

Example

```
> fib:= proc(n::nonnegint)
>   if n<2 then
>     n
>   else
>     fib(n-1)+fib(n-2)
>   end if
> end proc:
>
> seq(fib(n),n=0..10);
          0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

Code Generation Example

```
> polyeqn:=x^3-a*x-1;
                                3
                   polyeqn := x  - a x - 1

> sols:=solve(polyeqn,x);
          1/3
        %1         2 a
sols := ----- + -----,
          6        1/3
                 %1

        1/3                              /  1/3         \
      %1         a                  1/2 |%1        2 a |
    - ----- - ----- + 1/2 I 3       |----- - -----|,
       12        1/3                     |  6        1/3|
               %1                        \         %1   /
```

Code Generation Example

```
        1/3                      /  1/3        \
      %1        a            1/2 |%1       2 a |
    - ----- - ----- - 1/2 I 3    |----- - -----|
       12       1/3              |  6       1/3|
              %1                 \        %1   /

                     3      1/2
%1 := 108 + 12 (-12 a  + 81)
```

Code Generation Example

```
> sol1:=sols[1];
                              3       1/2 1/3
        (108 + 12 (-12 a  + 81)    )
sol1 := ------------------------------
                      6

                    2 a
     + ------------------------------
                             3       1/2 1/3
       (108 + 12 (-12 a  + 81)    )

> with(CodeGeneration);
[C, Fortran, IntermediateCode, Java, LanguageDefinition,

    Matlab, Names, Save, Translate, VisualBasic]
```

Code Generation Example

```
> C(sol1,optimize=true,declare=[a::float]);
t1 = a * a;
t5 = sqrt(-0.12e2 * t1 * a + 0.81e2);
t8 = pow(0.108e3 + 0.12e2 * t5, 0.1e1 / 0.3e1);
t13 = t8 / 0.6e1 + 0.2e1 / t8 * a;
```