An Introduction to Maple

School of Mathematics and Applied Statistics

University of Wollongong

2004

These notes are intended to provide an introduction to Maple. A basic understanding of mathematics is assumed. This manual is based on Version 9.01 of Maple on the PC platform.

Dr Alex Antic & Dr Maureen Edwards

School of Mathematics and Applied Statistics

University of Wollongong

July 2004

Contents

1 What is Maple?

Maple is a powerful general-purpose mathematical problem solving software package. It allows you to solve complex mathematical problems by using its many built-in mathematical solvers, generate two and three dimensional interactive plots and even contains a powerful programming language so you can write your own code.

In the forthcoming sections, the extensive functionality of Maple will be explored by discussing some of its main features. Additional information on any of the topics discussed, or any other feature of Maple, can be found in the comprehensive *Help* menu accessible via the menu bar.

2 Getting Started

The following sections aim to provide a quick and easy introduction to Maple. They outline the basics of Maple including creating a new worksheet, understanding the syntax and using some of the many built-in features.

2.1 Worksheets

A Maple document is called a *worksheet*. All the commands used in Maple are entered into the worksheet.

To create a **new** worksheet select $File \rightarrow New$ from the menu bar. When Maple is launched, a new untitled worksheet will be created by default.

To **save** a worksheet select $File \rightarrow Save$ from the menu bar. All Maple worksheets have *.mw* appended to the end of the filename by default.

To **open** a previously saved worksheet select $File \rightarrow Open$ from the menu bar.

2.1.1 Executing Commands

Every execution statement entered in a Maple worksheet must end with either a semicolon ; if the output is to be displayed, or a colon : if display of the output is to be suppressed.

To **execute** a command the "Enter", or "Return", key is pressed. When executing a Maple command the cursor does not have to be at the end of the command sequence, it can appear anywhere within the command upon entering the "Enter" key. To continue the command sequence on a new line, pressing "Shift"+"Enter" will move the cursor to the next line.

Note: Maple is **case-sensitive**.

2.2 Special Constants

The following characters and words are **protected** in Maple

`I`	complex symbol, $I^2 = -1$
`infinity`	mathematical infinity ∞
`Pi`	constant value of π

`pi` is also used in Maple but it is **not** a protected word as it is only used as a symbol to represent the constant value of π. Furthermore, the characters `e` and `E` are **not** protected in Maple. To use the value of the exponential e, use the following command

```
exp(1);
```

Any symbol or variable name beginning with an **underscore** is effectively reserved for use by Maple library code and should not be used. Also, variable names should not begin or end with a **tilde**.

2.3 Operators

The following **mathematical** operators are used in Maple

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`^`	exponentiation

The following **logical** operators are used in Maple

`<`	less than
`>`	greater than
`<=`	less than or equal to
`>=`	greater than or equal to
`<>`	not equal

Note: The '.' operator performs non-commutative multiplication on its arguments. The following rules apply when using the operator '.' in Maple, and are illustrated via examples:

- 2.5 is a number.
- 2 . 5 equals 10.
- 2 .5 equals 10.
- 2. 5 is an error.

2.3.1 The Ditto Operator

The *ditto* operator is a useful function within Maple which allows previously generated results to be accessed. The three ditto operators, `%`, `%%` and `%%%` respectively, refer to the last, second last and third last, non-null results computed in the worksheet.

For example, if the following command had been executed

```
2 + 4;
```

and it was immediately followed by the execution of the command

`%*5;`

the generated result would be 30.

Note: The results generated do not necessarily have to be the last three successive entries, but the last three commands executed anywhere in the worksheet. Hence, the previous line in a worksheet may not be the one returned by the ditto operator.

For a complete list of all the operators used in Maple select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *General Information* → *expression index*

2.4 Sets and Lists

In Maple, there are two different ways in which to order a sequence of expressions, either by a **set** or a **list**.

A set is an unordered sequence of expressions of the form `{e1, e2, ...}`. When using a set to denote a sequence of expressions do not assume that Maple will preserve the order.

A list is an ordered sequence of expressions of the form `[e1, e2, ...]`, where `e1`, `e2`, . . . , are expressions.

Note: Expressions in a list can be sorted via the `sort` function.

For example, the command

```
{a, a, b};
```

will return

```
{a, b}
```

while the command

```
[a, a, b];
```

will return

```
[a, a, b]
```

2.5 Commands and Functions

To **assign** a value to a variable the command `:=` is used.

For example, to assign the value of 2 to the variable x and to display the output, the following command is used

```
x := 2;
```

To **unassign**, or clear, a previously assigned variable, the `unassign` function is used.

For example, if the variable x had previously been assigned the value of 2, then to clear this value the following command would be used

```
unassign('x');
```

To clear **all** stored information in a Maple worksheet the following command is used

```
restart:
```

2.5.1 Evaluation

When using special constants, such as `Pi`, Maple will by default represent them by their symbol, that is, π, and not by their numerical value. To obtain the value of a constant, you need to ask Maple to evaluate it by using the `evalf` function, which converts an expression from its symbolic form to its numeric form.

For example, to obtain the value of π, the following command is used

```
evalf(Pi);
```

Note: As `pi` is used to denote the symbol of the constant π only, you **cannot** obtain the numerical value of π via the `evalf` function.

Maple represents **fractions** in rational form. To obtain the numerical value of a fraction, the function `evalf` is used.

For example, Maple will leave 10/7 in rational form, so to obtain the decimal form of 10/7 the `evalf` function is used. Furthermore, when dealing with **irrational** numbers, the `evalf` function is also used to evaluate them.

Note: If the fraction 10/7 had instead been represented as 10.0/7, or 10/7.0 or 10.0/7.0, the `evalf` function would **not** need to be used as Maple would automatically convert the fraction to decimal form.

Maple has a number of other evaluation functions, such as `Eval`, which evaluates an expression at a point, and `evalb`, which evaluates Boolean expressions.

For a complete list of all the evaluation functions available in Maple select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Evaluation$

2.5.2 Digits

The default number of digits that Maple returns is 10. To change this value to n, where n is a natural number in the range from 1 to 268435448, the following command is used

```
Digits := n;
```

To check the number of digits being used, the following command would be used

```
Digits;
```

2.6 Simplification

There are a number of simplifications which Maple will perform by default for any calculation. For an arbitrary variable x, or expression, these are

$x - x$ $\quad x + x$ $\quad x + 0$ $\quad x \times x$

x/x $\quad x \times 1$ $\quad x^0$ $\quad x^1$

Note: Trying to evaluate ∞/∞ or $\infty - \infty$ will return the following error message

```
undefined
```

Trying to calculate $0/0$ will return the message

```
Error, numeric exception: division by zero
```

2.7 Common Mathematical Functions

Maple has many built-in mathematical functions that are commonly used. These include `sin`, `cos`, `tan`, `exp`, `ln`, `sqrt`, among others.

For example, to evaluate $sin(\pi)$, the following command is used

```
sin(Pi);
```

For a complete list of the standard library of functions available select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow General\ Information \rightarrow function\ index$

2.7.1 Summation

The two most commonly used functions in Maple for calculating summations are `sum` and `add`.

The function `sum` is used for calculating a formula for a definite or indefinite sum. The call to the function is of either of the following two general forms

```
sum('expr', 'k')
```

or

```
sum('expr', 'k' = a..b);
```

where `expr` is the expression to be summed, `k` is the summation index and `a..b` represents the chosen range.

If Maple is unable to calculate a closed form, the function call is returned.

For example, to calculate the general formula for the sum of x^k for k ranging from 0 to ∞, the following command is used

```
sum('x^k', 'k' = 0..infinity);
```

The function `Sum` is the inert form of the `sum` function. It is used for representing summations in general form, and not for evaluating them.

For example, to represent the following summation in Maple

$$\sum_{0}^{n} x^k$$

the following command is used

```
Sum(x^k, k = 0..n);
```

The `add` function is used for adding up a finite sequence of values. The `sum` function can also be used to calculate explicit sums, although it is recommended that the function `add` be used instead. The call to the function is of the general form

```
add(expr, k = a..b);
```

where `expr` is the sequence of values to be summed and `a..b` is the range over which to sum. The values of `a` and `b` must be numerical.

Note: `a` can be `infinity` and `b` can be `-infinity`.

For example, to add up the expression x^2 for x ranging from 0 to 4, the following command is used

```
add(x^2, x = 0..4);
```

For more information on mathematical operations, including `sum` and `add`, select

Help → Table of Contents

from the toolbar, and under the *Contents* tab select

Mathematics → Basic Mathematics → Arithmetic Operations

Other commonly used commands include `mul`, which multiplies a sequence of values, `product`, which computes a symbolic product, `expand`, which expands an expression and `factor`, which factorises a multivariate polynomial.

2.7.2 Limits

Maple can calculate directional and bidirectional limits. To attempt to calculate the limiting value of an expression `expr`, a function of `x`, as `x` approaches `a`, the `limit` functions is used. The call to the function is of the general form

```
limit(expr, x = a, dir);
```

where `dir` (optional) specifies if a particular direction is required. The options are `left`, `right`, `real` or `complex`. If no direction is specified, the real bidirectional limit is evaluated.

If Maple cannot evaluate the limit, the function call will be returned.

Note: `a` can be `infinity` or `-infinity`.

For example, to calculate the limit of $ln(x)$ as $x \to 0$, the following command is used

```
limit(ln(x), x = 0);
```

For example, to calculate the limit of $1/x$ as $x \to 0^-$, the following command is used

```
limit(1/x, x = 0, left);
```

The function `Limit`, known as the **inert** form of the function `limit`, is used in place of `limit` to only express the limit, rather than evaluate it.

For example, to represent the following limit in Maple

$\lim_{x \to 0} f(x)$

the following command would be used

```
Limit(f(x), x = 0);
```

For more information on evaluating limits in Maple select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Calculus* → *Limits*

2.7.3 Complex Variables

Maple contains functionality related to complex variables.

For example, to represent the complex variable $z = 3 + 5i$, the following command is used

```
z := 3 + 5*I;
```

To then select the real and imaginary components, the following commands would be used

```
Re(z);
```

and

```
Im(z);
```

If the complex expression is not in the form $z = a + ib$ then the function `evalc` can be used to attempt to split the expression into its real and imaginary components. For example, to represent the expression $1/(a - i)$, where a is an arbitrary constant, in the form $z = a + ib$, the following command is used

```
evalc(1 / (a - I));
```

which would return

$$\frac{a}{a^2 + 1} + \frac{i}{a^2 + 1}.$$

3 Packages

Maple has a large collection of built-in functions that can easily be accessed. To use the most common functions, such as `sin` and `evalf` they are simply entered as a command. However, there are also available **packages** of functions. Prior to using one of these particular functions, the relevant package containing the chosen function must first be **imported** using the `with` command. The general procedure

for importing a package is

```
with(package_name);
```

Note: Upon importing a package, there will appear a list (assuming you have ended the command with a semicolon and not a colon) of all the functions contained within that package.

For example, to use the `contourplot` function, contained within the `plots` package, the following sequence of commands is used

```
with(plots):
contourplot(sin(x*y), x = -2..2, y = -2..2);
```

For a complete list of all the packages available select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow General\ Information \rightarrow packages$

4 Graphics

Maple contains functionality to make plotting two and three dimensional graphs easy, and even allows for the animation of plots.

When generating a plot in Maple you can choose to have it displayed either `Inline`, that is, in the worksheet, or `Window`, that is, in a separate window. The default

setting is `Inline`.

To toggle between the two select *Tools* → *Options* from the toolbar, then select the *Plotting* tab and under *Plot Display* select either `Window` or `Inline`.

4.1 Two Dimensional Plots

To create a two dimensional plot, the `plot` function is used. The call to the function is of the general form

```
plot(f, h, v, o);
```

where `f` is the function to be plotted, `h` is the horizontal range, `v` (optional) is the vertical range and `o` (optional) is a list of options.

Note: The `plots` package does not need to be imported in order to use the function `plot` as it is one of the standard library of functions available.

For example, to plot the function $sin(x)$ for x in the range 0 to 2π, the following command is used

```
plot(sin(x), x = 0..2*Pi);
```

Functions with infinite domains can also be plotted in Maple.

For example, to plot $sin(x)$ in the range from $x = 0$ to ∞, the following command is used

```
plot(sin(x), x = 0..infinity);
```

Some of the options available for plotting functions include

- `linestyle`, which include `SOLID`, `DOT`, `DASH` or `DASHDOT`,
- `labels = [x,y]`, which allows you to label the horizontal and vertical axes
- `scaling`, which can be either `constrained` or `unconstrained` (default).

Maple scales plots to fit the window. If a plot seems distorted as a result of this, the scaling option needs to be set to `constrained`.

For example, to plot the function $cos(x)$ from -2π to 2π with a dotted line style and with scaling eliminated, the following command would be used

```
plot(cos(x), x = -2*Pi..2*Pi, linestyle = DOT, scaling = constrained);
```

For a complete list of options available for two dimensional plots select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Graphics \rightarrow 2-D \rightarrow Options$

Other graphing functionality includes plotting functions in **polar coordinates**, for instance. To do so, the `polarplot` function is used, which is contained within the `plots` package. The `polarplot` function defines one or more curves in two dimensional space given in polar coordinates. The call to the function is of the general form

```
polarplot(c, o)
```

where `c` is a set of two dimensional curves and `o` is a list of options. If no range is specified, the default angle range used is $-\pi \leq \theta \leq \pi$.

Note: The same set of options that apply to the `plot` function also apply to the `polarplot` function.

For example, to plot a circle of radius 1, the following sequence of commands is used

```
with(plots);

polarplot(1, theta = 0..2*Pi);
```

For a complete list of all the library of functions contained within the `plots` package select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Packages* → *plots*

Note: The `plots` package contains functions related to both two and three dimensional plots.

4.2 Multiple Plots

It is possible to display more than one function on the same set of axes by using the `plot` function. The call to the function is either of the general form

```
plot({f1, f2, ...}, h, v, o);
```

or

```
plot([f1, f2, ...], h, v, o);
```

depending on whether the sequence of functions to be superimposed are given as a set or a list, where `f1`, `f2`, ... are a sequence of the functions to be plotted, `h` is the horizontal range, `v` (optional) is the vertical range and `o` (optional) is a list of options.

For example, to display the functions $sin(x)$ and $cos(x)$ from $-\pi$ to π on the same plot, the following command is used

```
plot([sin(x), cos(x)], x = -Pi..Pi);
```

4.2.1 Display

For greater flexibility when displaying multiple plots on the same graph, the `display` function is used. It allows a number of plots with different ranges, and different line styles for instance, to be displayed together. The call to the function is either of the general form

```
display({p1, p2, ...}, o);
```

or

```
display([p1, p2, ...], o);
```

depending on whether the sequence of plots to be superimposed are provided as either a set or a list, where `p1`, `p2` ... are a sequence of the plot structures to be superimposed and `o` is a set of options, which are the same as that for the function

`plot`.

Note: The function `display` is contained within the `plots` package.

For example, to plot the functions $sin(x)$ and $cos(x)$ on the same graph, with different line styles and ranges of x, the following sequence of commands could be used

```
with(plots);
plot1 := plot(sin(x), x = 0..4*Pi, linestyle = SOLID):
plot2 := plot(cos(x), x = -2*Pi..2*Pi, linestyle = DASH):
display([plot1, plot2]);
```

A useful option for `display`, and `plot`, is `view = [xmin..xmax, ymin..ymax]`, which indicates the minimum and maximum range of x and y values to be displayed.

Note: The same set of options that apply to the `plot` function also apply to the `display` function.

4.3 Three Dimensional Plots

Maple has the ability to generate plots in three dimensions. For simple three dimensional plots, the `plot3d` function is used instead of the `plot` function. As with the `plot` function, the `plots` package does not need to be imported as it is part of the standard library of functions available.

For example, to generate a three dimensional plot of $cos(xy)$ for x and y both ranging from -3 to 3, the following command would be used

```
plot3d(cos(x*y), x = -3..3, y = -3..3);
```

The default coordinate system for three dimensional plots is the **Cartesian** coordinate system. To change the coordinate system to one of many others, which include `spherical` and `cylindrical`, the `coords` option is used.

For example,

```
plot3d(theta, theta = 0..2*Pi, phi = 0..Pi, coords = spherical);
```

For a complete list of all the coordinate systems available select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Graphics \rightarrow 3-D \rightarrow Options \rightarrow coords$

One of the functions found in the `plots` package related to three dimensional plots is the `spacecurve` function, which is used for plotting one or more curves in three dimensional space.

For example,

```
with(plots);
spacecurve([cos(t), sin(t), t], t = 0..6*Pi);
```

For a complete list of all the options available for three dimensional plots select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Graphics \rightarrow 3-D \rightarrow Options$

To display more than one three dimensional plot on the same graph, the `display3d` function can be used, which is an **alias** to the `display` function. Either function can be used for superimposing three dimensional plots, but the `plots` package must first be imported.

5 Number Theory

Maple contains an assortment of functions pertaining to Number Theory. Examples of some of the most commonly used include

- `roots`, which calculates the exact roots of a polynomial. For example, to calculate the roots of $x^2 + 4x + 4$ you would use the command

  ```
  roots(x^2 + 4*x + 4);
  ```

- `igcd`, which calculates the greatest common divisor of an arbitrary number of integers. For example, to calculate the greatest common divisor of 4 and -12 you would use the command

  ```
  igcd(4,-12);
  ```

- `irem(a,b)`, which calculates the integer remainder of a divided by b, where a

and b are both integers. For example,

```
irem(17,5);
```

The `numtheory` package contains functions related to Number Theory. These include

- `pi(n)`, which calculates the number of prime numbers less than or equal to the integer n
- `divisors(n)`, which calculates the set of positive divisors of the integer n

For a complete list of all the function available related to Number Theory select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Number\ Theory$

6 Series Expansions

Maple provides the functionality to easily perform different types of series expansions. Two of the most commonly used series expansion functions in Maple are `series` and `taylor`.

For a complete list of all the series expansions available select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

Mathematics → Power Series

6.1 Series

The `series` function calculates a generalized series expansion. The call to the function is of the general form

```
series(expr, x = a, n)
```

The result is a truncated series expansion of the expression `expr` (a function of `x`) with respect to the variable `x` about the point `a`, which is assumed to be 0 if not provided, and up to order `n` (optional), which is a non-negative integer and has a default value of 6.

Note: If `a` is `infinity`, an asymptotic expansion is generated.

For example, the command

```
series(exp(x), x, 3);
```

will generate a series expansion of e^x about $x = 0$ up to order 3, while the command

```
series(1/y, y = 3);
```

will generate a series expansion of $1/y$ about $y = 3$ up to order 6.

6.2 Taylor

The `taylor` function calculates a Taylor series expansion. The call to the function is of the general form

```
taylor(expr, x = a, n)
```

The result is a Taylor series expansion of the expression `expr` (a function of `x`) with respect to the variable `x` about the point `a`, which is assumed to be 0 if not provided, and up to order `n` (optional), which is a non-negative integer and has a default value of 6.

For example, the command

```
taylor(exp(x), x = 3, 7);
```

will generate a Taylor series expansion of e^x about $x = 3$ up to order 7.

7 Solving Equations

There exist a number of functions in Maple to solve different forms of equations.

The function `solve` is used to solve a given equation, or set of equations, for an unknown, or set of unknown, variables. The call to the function is of the general form

```
solve(eqns, vars);
```

where `eqns` is an equation or inequality, or set of equations or inequalities, to be solved. If only an expression is provided, such as $x^2 - 2$, then Maple assumes $x^2 - 2 = 0$. `vars` (optional) is a variable, or set or variables, to be solved for. If no variables are specified, Maple will solve for **all** the variables that appear in the equation(s).

For example, to solve the equation $x^2 + 4x + 4 = 0$ for x, either of the following

three commands can be used

```
solve(x^2 + 4*x + 4 = 0, x);
```

or

```
solve(x^2 + 4*x + 4 = 0);
```

or

```
solve(x^2 + 4*x + 4);
```

The output to `solve` is either a solution, or a sequence of solutions, if more than one exists. If no solutions exist, or Maple cannot find any solutions, then the empty sequence `NULL` is returned, that is, no output is given.

For example, to solve the equation $x^2 + x - 6 = 0$ for x, and to access the solutions, the following commands would be used

```
sols := solve(x^2 + x - 6 = 0, x);
```

Maple would return

```
sols := 2, -3
```

To then access the values 2 and -3, that is, the first and second solutions in the sequence, the following commands would be used

```
sols[1];
```

and

```
sols[2];
```

Other functions available for solving equations include `fsolve`, which returns only floating-point, or decimal value, solutions, and `isolve`, which returns only integer

solutions, if they exist.

For example, consider solving the equation $x^2 - 2 = 0$ for x:

The following command

```
sols := solve(x^2 - 2 = 0, x);
```

will return

```
sols := 2^(1/2), -2^(1/2)
```

The following command

```
sols := fsolve(x^2 - 2 = 0, x);
```

will return

```
sols := -1.414213562, 1.414213562
```

The following command

```
sols := isolve(x^2 - 2 = 0, x);
```

will return

```
sols :=
```

as no integer solutions exist.

For a complete list of all the functionality available for solving equations select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Factorization and Solving Equations*

8 Differentiation and Integration

Maple provides the scope for differentiating and solving differential equations both analytically and numerically. It also allows for integration to be performed, both analytically and numerically.

8.1 Differentiation

There is provision within Maple to both differentiate expressions and to express differential equations.

The most commonly used function for differentiating expressions in Maple is the `diff` function. The call to the function is of one of the following two general forms

```
diff(expr, x1, x2, ..., xn);
```

or

```
diff(expr, [x1, x2, ..., xn]);
```

which calculates the partial derivative of the algebraic expression `expr` with respect to `x1`, `x2`, ... `xn`, respectively.

For example, to calculate the derivative of some function $f(x)$ with respect to x, the following command is used

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

Note: The following three commands are equivalent

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

```
diff(f(x,y), [x, y]);
```

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

For calculating **higher order derivatives** the sequence operator `$` is used.

For example, the command

```
diff(f(x), x$3);
```

is equivalent to

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

while the command

```
diff(f(x,y), x$2, y$3)
```

is equivalent to

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

For example, the command

```
diff(cos(x), x);
```

will return

```
-sin(x)
```

while the command

```
diff(cos(x), x$2);
```

will return

```
-cos(x)
```

For more information on differentiation in Maple select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Calculus \rightarrow Differential\ Calculus$

8.1.1 Expressing Differentials

The function `Diff` is used, which is the inert form of function `diff`, to express a differential rather than to evaluate it. `Diff` has the same call as that of the function `diff`.

For example, to represent the following equation in Maple

$$\frac{dy(x)}{dx} + y(x)^2 = 0$$

the following command is used

```
diff(y(x), x) + y(x)^2 = 0;
```

8.1.2 D Operator

D operator notation is also available in Maple via the `D` function.

For example, the command

```
D(sin)(x);
```

will return

```
cos(x)
```

For more information on operator D notation select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Calculus \rightarrow Differential\ Calculus \rightarrow D$

8.1.3 Solving Differential Equations

It has been detailed prior, that to solve a general equation, or system of general equations, the function `solve` is used. When dealing with differential equations, there exist a number of specialized functions to choose from, including `dsolve` and `pdsolve`, among others, depending on the type of differential equation to be solved. The `dsolve` function is used to solve, either analytically or numerically, an **ordinary differential equation (ODE)**, or system of ODE's.

To attempt to obtain an **analytical** solution to an ODE, or system of ODE's, the call to the function is of the general form

```
dsolve({eqns, IC}, y(x), args);
```

where `eqns` is the ODE, or system of ODE's, which are functions of x, you wish to solve, `IC`, are the initial conditions, if any, `y(x)` is an indeterminate function, or set of functions, and `args` (optional) are any extra arguments, such as `explicit`, which requests the solutions be in explicit form.

For example, to solve the following ODE for $y(x)$ analytically

$$\frac{dy(x)}{dx} - x = 0$$

the following command is used

```
dsolve(diff(y(x), x) - x = 0, y(x));
```

Note: Maple uses the notation `_C` to represent arbitrary constants.

For more information on the function `dsolve` select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Differential Equations* → *dsolve* → *Overview*

Maple also has the ability to find **numerical** solutions to ODE's. The call to the function is of the general form

```
dsolve(odesys, numeric, var, options);
```

where `odesys` is a set or list of ODE(s) and initial or boundary conditions, `numeric` asks Maple to solve the ODE(s) numerically, `var` (optional) is a dependent, or a list of, dependent variables and `options` (optional) is a list of options, which includes instructing Maple to use one of its set of methods to solve the ODE(s). For instance, the default method for solving initial value problems is `rkf45`, which is a Runge-Kutta Fehlberg methods that produces a fifth order accurate solution. Other methods include `dverk78` and `taylorseries`, among others.

The output, by default, is a **procedure** that can be used to obtain solution values for given values of the independent variable.

For example, to obtain a numerical solution for $y(x)$ to the ODE

$$\frac{dy(x)}{dx} - e^{y(x)}y(x) + cos(x) = 0$$

with initial condition $y(0) = 0$, the following sequence of commands could be used

```
ode1 := diff(y(x), x) - exp(y(x))*y(x) + cos(x) = 0;
```

```
sols := dsolve({ode1, y(0) = 0}, numeric);
```

which would return

```
sols := proc(x_rkf45) ... end proc
```

This indicates that the default method `rkf45` was used to obtain a numerical solution. To now obtain solutions for $y(x)$ for any value of x, the following commands are used

```
sols(x_value);
```

For example, to obtain the value of $y(x)$ for $x = 1$, the following command is used

```
sols(1);
```

which would return

```
[x = 1., y(x) = -1.11686488822056496]
```

To then access the value of $y(x)$, for instance, which is the second element in the sequence, the following command is used

```
sols(1)[2];
```

which would return

```
y(x) = -1.11686488822056496
```

To solve the above ODE by another method, such as `dverk78`, the following command would be used

```
sols := dsolve({ode1, y(0) = 0}, numeric, method = dverk78);
```

For more information on solving ODE's numerically, and the different methods available, select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Differential\ Equations \rightarrow dsolve \rightarrow numeric$

To solve a **partial differential equation (PDE)**, or system of PDE's, the function `pdsolve` is used. The call to the function is of the general form

```
pdsolve(pdesys, f, numeric, options);
```

where `pdesys` is the PDE, or system of PDE's, to be solved, `f` is the indeterminate function(s), `numeric` is used when a numerical solution is required and `options` (options) is a list of exta options.

For example, to solve the following PDE in Maple

$$\frac{\partial f(x,y)}{\partial x} + \frac{\partial f(x,y)}{\partial y} = 0$$

the following sequence of commands could be used

```
pde1 := diff(f(x,y), x) + diff(f(x,y), y) = 0;

pdsolve(pde1);
```

Note: Maple uses the notation `_F` to represent arbitrary functions.

For more information on solving PDE's select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Differential\ Equations \rightarrow pdsolve$

8.2 Integration

Maple has the ability to solve definite and indefinite integrals.

The call to the function for performing an **indefinite** integral is of the general form

```
int(expr, x);
```

where `expr` is the expression to be integrated, that is, the integrand, and `x` is the variable which we are integrating with respect to.

Note: A **constant of integration** does not appear in the result.

The call to the function for performing a **definite** integral is of the general form

```
int(expr, x = a..b);
```

where `expr` is the integrand, `x` is the variable which we are integrating with respect to and `a` and `b` are the endpoints.

For example, to calculate the integral of $sin(x)$ between the limits 0 and π, the following command is used

```
int(sin(x), x = 0..Pi);
```

Note: If Maple cannot evaluate the integral, the function call will be returned.

The function `Int` is used to only express the integral, rather than to evaluate it.

For example, to represent the following integral in Maple

$\int_a^b f(x)\,dx$

the following command would be used

```
Int(f(x), x = a..b);
```

For more information on solving integration select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Calculus \rightarrow Integration \rightarrow int$

To perform **numerical** integration in Maple the following function is used

```
evalf( Int(expr, x = a..b) );
```

Note: The inert form of function `int`, the function `Int`, is commonly used in the call to function `evalf` to prevent Maple from invoking the integration routine. If the command `int` returns an unevaluated integral, then the command

```
evalf( int(expr, x = a..b) );
```

can be used.

For example, to obtain the numerical value of the integral

$\int_0^1 \frac{e^{-x^2}}{(1+x)}\,dx$

from 0 to 1 the following command could be used

```
evalf( Int( exp(-x^2)/(1 + x), x=0..1 ) );
```

For more information on numerical integration select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Calculus \rightarrow Integration \rightarrow evalf \rightarrow int$

9 Transformations

The `inttrans` package contains a collection of functions that calculate integral transformations, such as Fourier and Laplace transforms.

For a complete list of all the functions contained in the `inttrans` package select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Calculus \rightarrow Transforms \rightarrow inttrans$

9.1 Fourier

To calculate a Fourier transformation in Maple, the `fourier` function is used, which is part of the `inttrans` package.

The call to the function is of the general form

```
fourier(expr, t, w);
```

where `expr` is the equation, expression or set of equations/expressions to be transformed, `t` is the variable that `expr` is being transformed with respect to, and `w` is the parameter of transformation.

For example, to calculate the Fourier transformation of $cos(x)$ with respect to x, the following command is used

```
with(inttrans):
fourier(cos(x), x, w);
```

To calculate the **inverse Fourier transformation** of an expression or equation `expr` with respect to `w`, the `invfourier` function is used, which is also part of the `inttrans` package, and the call is of the following general form

```
invfourier(expr, w, t);
```

where `t` is the parameter of transformation.

The `fourier` and `invfourier` functions recognise the derivative functions `diff` and `Diff`, and the integral function `int` and `Int`.

Note: The `inttrans` package also contains function that perform Fourier Cosine and Fourier Sine transformations. The functions are `fouriercos` and `fouriersin`, respectively.

For more information on performing Fourier transformations in Maple select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Calculus \rightarrow Transforms \rightarrow fourier$

9.2 Laplace

To calculate a Laplace transformation in Maple, the `laplace` function is used, which is part of the `inttrans` package.

The call to the function is of the general form

```
laplace(expr, t, s);
```

where `expr` is the equation, expression or set of equations/expressions to be transformed, `t` is the variable that `expr` is being transformed with respect to, and `s` is the parameter of transformation.

For example, to calculate the Laplace transformation of $sin(x)$ with respect to x, the following command is used

```
with(inttrans):
laplace(sin(x), x, s);
```

To calculate the **inverse Laplace transformation** of an expression or equation `expr` with respect to `s`, the `invlaplace` function is used, which is also part of the `inttrans` package, and the call is of the following general form

```
invlaplace(expr, s, t);
```

where `t` is the variable in the transformed expression.

The `laplace` and `invlaplace` functions recognise the derivative functions `diff` and `Diff`, and the integral function `int` and `Int`.

Note: When taking the Laplace transformation of an expression such as $diff(f(x), x, s)$, the `laplace` function will insert the initial conditions $f(0)$, $D(f)(0)$, etc.

For more information on performing Laplace transformations in Maple select

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Calculus \rightarrow Transforms \rightarrow laplace$

10 Linear Algebra

Maple offers two packages for performing linear algebra computations, namely, the `linalg` package and the `LinearAlgebra` package. The `linalg` package has been **superseded** by the `LinearAlgebra` package, and along with the `VectorCalculus` package, but both are available in the latest version of Maple (Version 9.01). The packages differ as follows:

- The `linalg` package is better suited to computations involving abstract linear algebra
- The `LinearAlgebra` package has the following features:
  - Includes special matrix constructor commands
  - Is more user-friendly in matrix algebra calculations

- Is more powerful and efficient for doing numeric linear algebra calculations involving large matrices

10.1 linalg

The `linalg` package contains an extensive set of functions designed to perform numerous types of calculations in linear algebra.

For more information on the `linalg` package select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Packages* → *linalg*

The basic data structure used by the `linalg` package is the **array**, which is created by using the `array` command. Both matrices and vectors are created using the `array` function.

10.1.1 Matrices

A matrix is represented as a two-dimensional array with row and column indices indexed from 1. There are two ways in which a matrix can be defined, either via the `array` function or the `matrix` function, which is part of the `linalg` package.

For Example, to create an empty $m \times n$ matrix, where m and n are non-negative

integers, either of the the following command sequences can be used:

```
A := array(1..m, 1..n);
```

or

```
with(linalg):
A := matrix(m, n);
```

Assigning values to the entries of a matrix can be done either when creating the matrix, or after the matrix has been created.

For example, to create the following 2×2 matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

either of the following command sequences can be used:

```
A := array(1..2, 1..2, [[1, 2], [3, 4]]);
```

Note: If the `matrix` function was used instead, the command sequence would be

```
with(linalg):
A := matrix(2, 2, [1, 2, 3, 4]);
```

or

```
A := array(1..2, 1..2);
A[1,1] := 1;
A[1,2] := 2;
A[2,1] := 3;
A[2,2] := 4;
```

To view a matrix A, the following command is used:

```
print(A);
```

Maple contains functions such as `diagonal`, `identity` and `symmetric`, among others, that allow for the creation of specific types of matrices.

For example, to create a 4×4 **identity** matrix, the following command can be used

```
array(1..4, 1..4, identity);
```

The function `evalm` is used for performing sums, products and integer powers of matrices.

For example, to calculate the sum of two matrices A and B, the following command can be used

```
evalm(A + B);
```

The operator `&*` is used to indicate **non-commutative** matrix multiplication.

For example, to calculate the matric product ABC, either of the following two commands can be used

```
evalm( A &* B &* C );
```

or

```
evalm( &*(A, B, C) );
```

Note: The `multiply` function, which is part of the `linalg` package, can instead be used to calculate a matrix product:

```
with(linalg):
multiply(A, B, C);
```

The functions `row` and `col`, contained in the `linalg` package, are used to extract the i'th row or column, respectively, from a given matrix. The results returned are vectors.

For example, consider the matrix A, defined as follows

$$\begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix}$$

To extract row 1, that is,

$(1, 3, 5)$

the following command is used

```
row(A, 1)
```

and to extract columns 2 and 3, that is,

$(3, 9), (5, 11)$

the following command is used

```
col(A, 2..3);
```

There are a large number of functions available in the `linalg` package for performing matrix calculations, including `det`, `inverse`, `transpose`, `augment`, `eigenvalues`, and `hermite`, which calculates the **Hermite Normal Form**, or **Reduced Row Echelon Form**, of an $m \times n$ rectangular matrix.

For example, to calculate the **determinant** of a **square** matrix A, the following command sequence is used

```
with(linalg):
```

```
det(A);
```

It is also possible to ask Maple to produce the general form of the determinant for a $n \times n$ matrix.

For example, to obtain the following general form of the determinant for a general 2×2 matrix A,

$A_{1,1}A_{2,2} - A_{1,2}A_{2,1}$

the following command sequence can be used

```
with(linalg):
A := matrix(2, 2);
det(A);
```

10.1.2 Vectors

A vector is represented as a one-dimensional array whose indices start at 1. There are two ways in which a vector can be defined, either via the `array` function or the `vector` function, which is part of the `linalg` package.

For Example, to create an empty vector of length m, where m is a non-negative integer, either of the the following command sequences can be used:

```
A := array(1..m);
```

or

```
with(linalg):
A := vector(m);
```

Assigning values to the entries of a vector can be done either when creating the vector, or after the vector has been created.

For example, to create the following vector A of length 4

$$(2, 4, 6, 8)$$

either of the following command sequences can be used:

```
A := array(1..4, [2, 4, 6, 8]);
```

Note: If the `vector` function was used instead, the command sequence would be

```
with(linalg):
A := vector(4, [2, 4, 6, 8]);
```

or

```
A := array(1..4);
A[1] := 2;
A[2] := 4;
A[3] := 6;
A[4] := 8;
```

To view a vector A, the following command is used:

```
print(A);
```

There are a large number of functions available in the `linalg` package for performing vector calculations, including `dotprod`, `crossprod` and `grad`.

For example, to calculate the **dot product** of two vectors A and B which are of

the same length, the following command sequence is used

```
with(linalg):
dotprod(A, B);
```

10.2 LinearAlgebra

The `LinearAlgebra` package contains an extensive set of functions for performing linear algebra calculations. It contains nearly all the functionality of the `linalg` package, but differs in the following ways:

- Contains well-defined data structure
- Contains additional commands for creating special types of matrices
- Improved matrix algebra
- More powerful and efficient for large numeric matrix calculations

The basic data structures used by the `LinearAlgebra` package are **Vectors** and **Matrices**, which are created by using the `Vector` and `Matrix` functions, respectively.

For more information on the `LinearAlgebra` package select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Linear Algebra* → *Linear Algebra Package*

10.2.1 Matrices

A matrix is represented as a two-dimensional array with row and column indices indexed from 1. Matrices are created using the `Matrix` function.

For example, to create a matrix A of size $m \times n$, where m and n are non-negative integers and where all entries are initially 0, the following command is used

```
A := Matrix(m, n);
```

To create a matrix B of size $p \times q$, where p and q are non-negative integers and where all entries are initially set to a number k, the following command is used

```
B := Matrix(1..p, 1..q, k);
```

To create a matrix C of size $u \times v$, where u and v are non-negative integers, and where each entry is represented by a symbol of the form $c_{i,j}$, where i and j denote the row and column numbers of the entry, the following command is used

```
C := Matrix(u, v, symbol = 'c');
```

To create a matrix C of size 2×2, defined as follows

$$\begin{pmatrix} 2 & 6 \\ 4 & 8 \end{pmatrix}$$

either of the following commands can be used

```
C := Matrix(2, 2, [[2, 6] , [4, 8]]);
```

or

```
C := Matrix([ [2, 6], [4, 8] ]);
```

To create an $n \times n$ **identity** matrix, the following command is used

```
Matrix(n, n, shape=identity)
```

Other options for `shape` include `diagonal`, `symmetric` and `triangular`.

Assigning values to the entries of a matrix can be done either when creating the matrix, or after the matrix has been created.

For example, to create the following 2×2 matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

either of the following command sequences can be used:

```
A := Matrix(2, 2, [[1, 2], [3, 4]]);
```

or

```
A := Matrix([ [1, 2], [3, 4] ]);
```

or

```
A := Matrix(2, 2);

A[1,1] := 1;

A[1,2] := 2;

A[2,1] := 3;

A[2,2] := 4;
```

To view a matrix A, use the command

```
A;
```

For more information on options for constructing matrices select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Linear Algebra* →

Linear Algebra Package → *Data Structures* → *Matrix Construction*

The calculation of sums, products and powers of matrices is quite simple as it is done directly.

To calculate the **sum** of two matrices A and B, with equal dimensions, the following command is used

```
A + B;
```

To **multiply** two matrices C and D, where the row dimension of C equals the column dimension of D, the following command is used

```
C . D;
```

Note: The operator '.' is used, and **not** '*'.

The `MatrixMatrixMultiply` function, which is part of the `LinearAlgebra` package, can also be used to multiply matrices.

The sequence of commands

```
with(LinearAlgebra):
MatrixMatrixMultiply(A, B);
```

calculates the product

```
A . B
```

Other functions contained in the `LinearAlgebra` package related to matrix multiplication include `MatrixScalarMultiply` and `MatrixVectorMultiply`.

To raise a square matrix M to the **power** of c, where c is an integer, the following command is used

```
M^c;
```

If the integer c is negative, then the inverse is calculated.

For more information on the calculations of matrices

$Help \rightarrow Table\ of\ Contents$

from the toolbar, and under the *Contents* tab select

$Mathematics \rightarrow Linear\ Algebra \rightarrow Linear\ Algebra\ Package \rightarrow Algebra$

Note: Maple will only display **small** matrices inline. For the worksheet version of Maple, small is defined as a matrix with the number of rows or columns in the range 1..10. Any matrix whose dimensions are larger than this is displayed using a place-holder.

There are a large number of functions available in the `LinearAlgebra` package for performing matrix calculations. These include those mentioned for the `linalg` package, along with many others, including `Determinant`, `Eigenvalues`, `Eigenvectors`, `ReducedRowEchelonForm` and `LUDecomposition`.

For example, to calculate the **determinant** of a matrix M, the following sequence

of commands can be used

```
with(LinearAlgebra):

Determinant(M);
```

The `Determinant` function can also produce the general form of the determinant for a matrix.

For example, to obtain the general form of the determinant for a 6×6 matrix, the following sequence of commands can be used

```
G := Matrix(6, 6, symbol = 'g');

with(LinearAlgebra):

Determinant(G);
```

To calculate the **inverse** of a square non-singular matrix A, the following command sequence is used

```
with(LinearAlgebra):

MatrixInverse(A);
```

For a complete list of all the functions available in the `LinearAlgebra` package select

Help $\rightarrow$ *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics $\rightarrow$ *Linear Algebra* $\rightarrow$ *Linear Algebra Package* $\rightarrow$ *Overview*

10.2.2 Vectors

A vector is represented as a one-dimensional array indexed from 1. Vectors are created using the `Vector` function.

For example, to create a vector A of size m, where m is a non-negative integer and where all entries are initially 0, the following command is used

```
A := Vector(m);
```

The `Vector` function has an option `shape`, that allows for special types of vectors to be constructed. The options are `constant`, `scalar`, `unit` or `zero`,

Note: Maple will only display **small** vectors inline. For the worksheet version of Maple, small is defined as a vector with the number entries in the range 1..10. Any vector whose dimensions are larger than this is displayed using a place-holder.

By default, Maple will display a vector vertically. To change this to the horizontal, the following command is used when creating a vector

```
A := Vector[row](m);
```

To create a vector B of size n, where n is a non-negative integer, and where all the entries are initially set to some number k, the following command is used

```
B := Vector(1..n, k);
```

Assigning values to the entries of a vector can be done either when creating the vector, or after the vector has been created.

For example, to create the following vector A of length 4

$$(2, 4, 6, 8)$$

either of the following command sequences can be used:

```
A := Vector(4, [2, 4, 6, 8]);
```

or

```
A := Vector([2, 4, 6, 8]);
```

or

```
A := Vector(4);

A[1] := 2;

A[2] := 4;

A[3] := 6;

A[4] := 8;
```

To view a vector A, use the command

```
A;
```

For more information on constructing vectors select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Mathematics → *Linear Algebra* → *Linear Algebra Package* →

Data Structures → *Vector Construction*

The calculation of sums and products of vectors is quite simple as it is done directly.

To calculate the **sum** of two vectors A and B of equal dimension, the following command is used

```
A + B;
```

To **multiply** two vectors C and D of equal dimension, the following command is used

```
C . D;
```

Note: The operator '.' is used, and **not** '*'.

There are a large number of functions available in the `LinearAlgebra` package for performing vector calculations. These include those mentioned for the `linalg` package. These include `DotProduct`, `CrossProduct`, `VectorScalarMultiply`, `VectorAngle` and `UnitVector`.

For example, to calculate the **angle** between two vector $V1$ and $V2$, the following command sequence is used

```
with(LinearAlgebra):
VectorAngle(V1, V2);
```

For a complete list of all the functions available in the `LinearAlgebra` package select

Help → Table of Contents

from the toolbar, and under the *Contents* tab select

Mathematics → Linear Algebra → Linear Algebra Package → Overview

11 Programming

Maple provides advanced functionality for programming. Programs are written in Maple via **procedures**.

11.1 Procedures

The general form of the structure of a procedure contains the procedure name, the set of arguments, the body of the procedure, or the code, and the end of the procedure. The command structure is as follows

```
proc_name := proc( list of arguments ) body end proc;
```

Note: All procedure must contain the words `proc` and `end proc;`.

For example, to write a procedure which simply forms a linear combination of the arguments, the following command structure can be used

```
lc := proc(a,b,c,d) a*b + c*d end proc;
```

To invoke a procedure, the procedure name and arguments must be used.

For example, to use the procedure `lc` to form a linear combination of the arguments `u, v, x, y`, the following command is used

```
lc(u, v, x, y);
```

to which, the following is returned

```
uv + xy
```

For more information on writing procedures in Maple select

Help → Table of Contents

from the toolbar, and under the *Contents* tab select

Programming → General Information → procedure

In general, procedures require the use of **statements**, such as

```
if...then...else...
```

```
for...do...
```

```
while...do...
```

but these are also used on their own.

For more information on statements select

Help → Table of Contents

from the toolbar, and under the *Contents* tab select

Programming → General Information → statements

11.1.1 The IF Statement

The `if` statement is a type of **conditional** statement, as a condition must be met for a command to be executed.

The general form of the `if` statement is as follows

```
if (condition) then statement

else statement

end if;
```

The `condition` is any **Boolean** expression formed using the relational operators (<, >, <=, >=, =, <>), the logical operators (and, or, not) and the logical names (true, false, FAIL).

When there are more conditions, the following general form is used

```
if (condition) then statement

elif (condition) then statement

elif (condition) then statement

else statement

end if;
```

Note: `elif` represents **else if**.

For example, the following statement sequence determines if a given number x is equal to, greater than or less than the number 2, and then prints the result to screen

```
if (x = 2) then print(equal)
```

```
elif (x > 2) then print(greater)

else print(less)

end if;
```

For more information on the `if` statement select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Programming → *Flow Control* → *if*

11.1.2 The Repetition Statement

The **repetition** statement `for...while...do...` allows for a statement sequence to be executed repeatedly for one of two reasons:

1. For a given amount of times, using the `for...to...` clause, or

2. Until a given condition is satisfied, using the `while...` clause.

The general form of the `for...to...` clause is

```
for name from expr by expr to expr while

do statement

end do;
```

where `from` indicates the value that the counter is to begin, and `by` specifies the increment step of the counter. If `from` or `by` are omitted, the default values of

`from 1` and `by 1`, respectively, are used by default.

For example, the following statement sequence prints all the even numbers between 0 and 10

```
for i from 0 by 2 to 10
do print(i)
end do;
```

The general form of the `while...` clause is

```
for name in expr while expr
do statement
end do;
```

For example, the following statement sequence calculates the sum of consecutive integers, starting at 1, until the sum exceeds 10

```
tot_sum := 0:
for i while tot_sum <= 10 do
tot_sum := tot_sum + i
end do;
```

For more information on repetition statements select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Programming → Flow Control → Iteration or Looping → for

11.2 Functional Operators

A functional operator is a special form of a procedure. They are used for defining functions, and are an alternative to defining the function as a procedure explicitly. Functional operators are written using **arrow** notation.

The general form of a functional operator is

```
vars -> result
```

where `vars` is a variable, or sequence of variable names, and `result` is the rule of the function acting on `vars`.

The functional operator is equivalent to

```
proc(vars) result end proc;
```

For example, to define the function $f(x) = 2x + 7$, the following command can be used

```
f := x -> 2*x + 7;
```

To evaluate the function at $x = 4$, for instance, the following command is used

```
f(4);
```

Note: This function can be defined explicitly as a procedure in the following way

```
f := proc(x) 2*x + 7 end proc;
```

For example, to define the function $g(x, y) = x^2 + y^2$, the following command is used

```
g := (x,y) -> x^2 + y^2;
```

For more information on functional operators select

Help → *Table of Contents*

from the toolbar, and under the *Contents* tab select

Programming → *Operations* → *Operators* → `->`

Alternatively, functional operators can be defined by using the `unapply` function. This function converts an expression, `expr`, and its arguments, `x, y, ...` into a function `f`, via the following command

```
f := unapply(expr, x, y, ...);
```

For example, to convert the expression

```
g := x^2 + cos(x);
```

into the function $G(x) = x^2 + cos(x)$, the following command is used

```
G := unapply(g, x);
```

To **substitute** one expression for another in a function, or any expression, the `subs` function is used.

The general call to the function is of the general form

```
subs(x = a, expr);
```

which substitutes `a` for `x` in the expression `expr`.

For example, to substitute $x = (a + b)$ into a function $f(x)$, the following command is used

```
subs(x = (a + b), f(x));
```

12 Help

In each of the sections discussed in this introductory manual, relevant instructions have been given to access more information via Maple's extensive *Help* menu, located on the menu bar. Alternatively, help topics can be accessed via the following command from within a worksheet

```
?topic;
```

Upon execution of this command, a new window will be generated within which the relevant help topic will appear.

13 Troubleshooting

PROBLEM	POSSIBLE SOLUTIONS
No output	Using : instead of ;
Function call won't work	Check case in function name
	Haven't imported the relevant package
	Using inert form of function, eg, `Limit` instead of `limit`
Plot doesn't appear	non-numeric value present
Matrix is not displayed	exceeded dimension

Additional Material?

Algorithms and code: examples

Files: Input and Output, eg read/write and word, pdf, ps

Differences between Maple versions and platforms

Cut and paste commands/output

Modification of plots, for example, via the menu bar

Further discussion regarding the online Help menu

List of websites, books, manuals, etc... related to Maple

References

[1] Rakesh, Maple Tutorial for MATH 243, *University of Dellaware*, August 20, 2000.

[2] Worthy, A., An Introduction to Mathematica 2.0, *School of Mathematics and Applied Statistics, University of Wollongong*, 1997.

[3] Edwards, M., MATH321 Nuncerial Analysis: Introductory Lab, Maple Crash Course, *School of Mathematics and Applied Statistics, University of Wollongong*, 2003.

[4] Maple 9.01: Copyright©Maplesoft, a division of Waterloo Maple Inc.1981 - 2003.