

Introduction to Maple — A Symbolic Computation Package

Hemanshu Roy Pota*

February 9, 2006

Contents

1	Introduction	2
1.1	Start and Quit	2
1.2	An Example	2
2	Variables and Getting Started	3
3	Numbers and Strings	4
4	Expressions and Functions	5
4.1	Maple Functions	5
5	Sequences, Sets, Lists, Arrays, and Tables	6
6	Control Structures	6
7	Repetition - seq, \$, for	6
8	Manipulation and Simplification	7
9	Helpful Commands	7
10	Procedures	7
11	Input and Output	8
12	Miscellaneous	8
12.1	Can I do this?	8

*School of Electrical Engineering, University College, University of New South Wales, ADFA, Canberra ACT 2600 Australia, pota@adfa.oz.au

Abstract

This short note is meant to get you started in formulating and solving problems using the functional symbolic programming software Maple. The material in this document gives you both philosophy and actual workings of Maple. To be able to use Maple effectively you need to understand the philosophy. For an extensive and all comprehensive package like Maple one can never learn all the features it has but one can understand its basic philosophy and use that understanding to search for appropriate commands to do a particular job. Maple has an extensive help facility. This document is an attempt to get you to wander through the help facility intelligently.

1 Introduction

Maple is a functional programming language with symbolic manipulation capabilities. Most of the other programming languages like C are procedural programming languages. When using Maple, think in terms of formulating your problem as functions of unknown variables and then manipulate these functions to solve for the unknowns. This is a powerful paradigm. All our mathematical and scientific thinking is done in terms of functions so it makes sense to use the computer software which enables you to work in the same way.

1.1 Start and Quit

To start a Maple session on unix systems type `xmaple&` at the command prompt; on IBM PC and Macintosh starting a Maple session should be fairly obvious. On Macintosh, to execute a command, remember to use the `ENTER` key in the numeric keypad. In a Maple session the default prompt is `>`. In the following it's assumed that you will type in all the statements which follow `>`. Everything following the sharp key `#` is treated as a comment in Maple. There is no need to type the comments following the commands while going through this tutorial. Text can be cut-and-paste within a Maple session like in any other editor. You can also go to a previous statement, edit it, and then execute it by hitting `ENTER`. `quit` or the selection of `Exit` from the `File` menu will terminate the program.

1.2 An Example

Let us look at the following simple example.

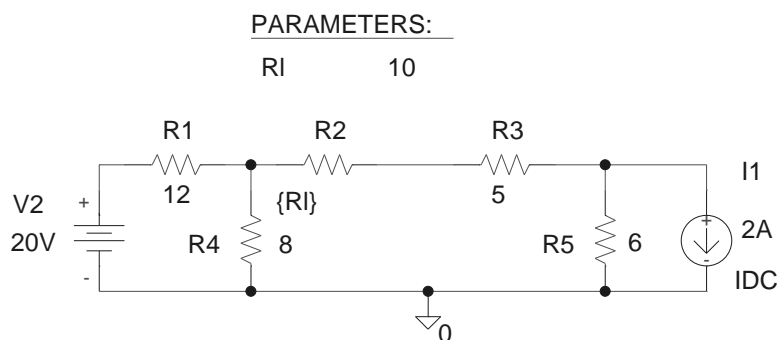


Figure 1: Maximum Power Transfer Example

For the circuit shown in Figure 1 we want to find that resistance R_L which will transfer maximum power to it from the network. The conceptual way to find that resistance is to get an expression for the power

dissipated across the resistor R_L , differentiate it with respect to R_L , equate the differential to zero, and solve for R_L . In Maple this solution can be implemented in its conceptual form itself. Enter the following statement sequence in a Maple worksheet to see how the solution is implemented.

```
> restart; #Start a fresh session.
> alias(I=I, j=sqrt(-1));
> eq1:=I[1]*12+(I[1]-I[2])*8=20;
> eq2:=(I[2]-I[1])*8+x*I[2]+5*I[2]+6*(I[2]-2)=0;
> sol:=solve({eq1,eq2},{I[1],I[2]});
> I[1]; #The value is not assigned yet.
> simplify(subs(sol,{eq1,eq2}));
> assign(sol);
> power:=(I[2]^2)*x;
> diff(power,x);
> R[L]:=solve("=0,x); #ditto (") copies the previous output
> evalf(R[L]);
> evalf(R[L],4);
```

To solve the above problem in a procedural programming environment one would first write a procedure to calculate the power dissipated across R_L for a given value of R_L . The second stage is to guess a suitable numerical range within which the solution will lie. Finally power dissipation for all the values in that range is either tabulated or plotted to find the maximum power transfer. This is not the most convenient way to think about the problem but one is forced to think this way because of the limitations posed by the procedural programming. We will implement this procedural solution in PSpice.

Learning is considerably enhanced by a close link between the natural problem formulation and its implementation in functional symbolic programming software like Maple.

Next we look at a few immediately useful features of Maple. Please be reminded that all the things I touch upon in this document give you a very brief introduction to the facility. Every function or command has many more features and you are well advised to look for them using the help facility.

2 Variables and Getting Started

In numerical procedural programming languages variables are used to store numerical values which are either frequently referred to or operated upon. In a symbolic functional programming language like Maple variables normally stand for the unknowns in functions and other expressions. In this paradigm a function is evaluated directly at the points of interest rather than indirectly by assigning values to a variable. Of course both numerical and symbolic values can also be assigned to variables in Maple. The next sequence of commands illustrates the direct and the indirect approach.

```

> # direct evaluation --- a functional paradigm
> f:=x->x^3+3*x^2+2*x+7; #function definition
> map(f,[1,2,3,4,10,101]); # map the function to a list
> # indirect evaluation --- a procedural paradigm
> s:=[1,2,3,4,10,101]:
> for x in s do f(x) od;

```

Although Maple provides for both functional and procedural approach, the functional paradigm should always be preferred.

?name should tell you the syntactically valid variable names. ?command is a shorthand for help(command); try ?help. It's a good idea to find out the names Maple has reserved for predefined constants (?constants). Try (?operators) to find out the predefined mathematical operators in Maple.

A semicolon followed by a newline character is an indication to Maple to begin processing the input. Maple echos the result on the screen as it processes the input. If you don't want the result to be displayed on the screen, end the input with a colon. Maple statements can be continued over several lines, the parser waits for either a semicolon or a colon before processing the statement. Try the following command sequence:

```

> restart; #restart starts a fresh maple session.
> x:=2e-3; #echo the result on the screen.
> x:=2: #do not echo the result on the screen.
> x:='x'; #unassign the value
> x;

```

The variable left of ':=' is assigned the 'fully evaluated' value of the expression to its right (?assignment). Evaluation of an expression can be delayed by enclosing it in quotes. Try the following.

```

> y:=x^2+5; #assign an expression to the variable y.
> x:=3:
> y;
> x:='x';
> y;
> y:='y';
> y;

```

Maple will not complain when the predefined constants are used as variable names but it will not allow a value to be assigned to these names. Be careful with I. This is predefined in Maple as $\sqrt{-1}$. Try:

```

> x:=3*I;
> I:=3; #Maple should complain

```

The following statement will change $\sqrt{-1}$ to the familiar j and free up I to be used as a normal variable.

```

> alias(I=I, j=sqrt(-1));
> I:=2;

```

3 Numbers and Strings

Like all other programming languages Maple has integer and float representation of numbers; in addition numbers can be represented as fractions. Strings have to be quoted using back quotes. Many of the options

for Maple function are strings, remember to use back quotes to specify strings. Remember that you can copy-and-paste in Maple like in any other editor. Try:

```
> x:=1; type(x,integer); type(x,float); type(x,fraction); type(x,numeric);
> x:=1e-3; type(x,integer); type(x,float); type(x,fraction); type(x,numeric);
> x:=11.34; type(x,integer); type(x,float); type(x,fraction); type(x,numeric);
> x:=3/4; type(x,integer); type(x,float); type(x,fraction); type(x,numeric);
> x:='This is a string in back quotes'; type(x,numeric); type(x,string);
> ?type;
```

4 Expressions and Functions

Expressions and functions are the basic entities and form the heart of Maple. Numeric data types are constant expressions. An expression or a function can also be a ratio of two expressions. Expressions can be added, multiplied, and divided just like ordinary variables to form other expressions.

```
> restart;
> f:=x^2+3*y+sin(x); #an expression
> subs({x=Pi/2,y=1},f); #expression evaluation by substitution
> evalf(",4); #floating point evaluation of the previous expression
> subs({x=a,y=b},f); #expression evaluation
> x; #value of x outside of the subs command is unchanged
> g:=x->x^2+3*sin(x); #a function of one variable
> g(Pi/2); #function evaluation
> h:=(x,y)->x^2+y^2; #a function of two variables
> h(1,2); #function evaluation
> f:=unapply(f,(x,y)); #make a function from an expression
> f(Pi/2,1);
> g:=f(x,y)/h(x,z); g*x^2+2*x; f(x,y)+h(x,y);
```

4.1 Maple Functions

Maple has functions to take limits (?limit), differentiate (?diff, ?D), integrate (?int), solve algebraic (?solve, ?fsolve) and differential (?dsolve) equations, perform Laplace transformations (?laplace), and also perform most of the standard mathematical operations. Try:

```
> restart;
> f:=x+2*x^2;
> g:=z->2*z+3/z;
> diff(f,x); #partial differential of f wrt x.
> diff(g(x),x);
> D(g);
> int(f,x);
> int(f,x=2..3); #a range is specified by var=lower..upper
> int(g(x),x);
> int(g(x),x=a..b);
> limit(g(x),x=infinity);
```

To find out about other predefined functions try:

```
> ?inifcns
> ?ininame
> ?index[packages]
```

5 Sequences, Sets, Lists, Arrays, and Tables

Expressions and functions can be organised in several ways. A sequence (`?sequence`) is formed using the comma operator. Sequences can be used to form other data types. Sets (`?set`) are formed by enclosing a sequence in curly brackets. They have all the properties of a conventional set. Order of elements in a set can change because Maple orders elements of a set convenient for its implementation. Lists (`?list`) are formed by enclosing sequences in square brackets. The ordering of the elements is preserved in a list. Arrays (`?array`) implement the conventional vectors and matrices. Arrays are built from ranges (`?range`) and lists. Tables (`?table`) are associative arrays. This means that tables can be indexed by data types other than integers.

```
> restart;
> f:=x->x^3+5; #a function
> g:=y^3+7+3*z^2; #an expression
> seq1:=x, 3, x^2, y^3, cos(x), f(r); #a sequence of expressions
> 3, 4, 4.05, f(n), x/y^2, sin(x)/cos(x); #another sequence
> s1:={seq1}; #a set can be formed from a sequence
> set1:={1, 1, 3, 6, 8, 8}; #a set
> {x, x^2, y, sqrt(x), g, 'A string'}; #another set
> l1:=[seq1]; #a list can be formed from a sequence
> list1:=[1, 5, 17, 100]; #a list
> list1[1]; op(1,list1); list1[4]; op(4,list1); #selecting elements of a list
> [f(7), f(x), 3, k^2, g, 'Maple Programming']; #another list
> op(6,"");
> array1:=array(1..2,1..2,[[1,2],[3,4]]); #an array
> array1[1,2]; array1[2,2]; #elements of the matrix
> array(1..2,1..2,[[x,x^2],[3*x,4*y]]); #an array
> table1:=table([name=John, surname=Citizen]); #a table (an associative array)
> table1[name];
```

6 Control Structures

Decision making can be done by `if/then/else` (`?if`) and other similar constructs.

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

Have a look at the binary operators which can be used in the conditional statements

`?operators[binary]`.

7 Repetition - seq, \$, for

Repeated evaluation of an expression can be achieved in Maple in several ways. Sequences can be built using `seq` (`?seq`) construct. Dollar (`?dollar`) is a shorthand way to build sequences. Loops with more

than one action statement can be built using for/while/do (?for) construct.

```
> restart;
> seq( i^2, i=1..5 ); #a sequence
> {seq( i^2, i=1..5 )}; #a set
> [seq( i^2, i=1..5 )]; #a list
> x$4; # $ is an iterator
> x[i] $i = 1..3;
> i:=2;
> x[i] $i = 1..3;
> 'x[i]' $'i' = 1..3;
> x:=3;
> 'x[i]' $'i' = 1..3;
> ''x[i]'' $'i' = 1..3;
> unassign('x','i');
> for i from 1 by 2 to 10 do i^3 od;
> seq1:=1,2,y; set1:={1,2,y}; list1:=[1,2,y];
> for x in seq1 do x^2; sin(x); od;
> for x in set1 do x^2; sin(x); od;
> for x in list1 do x^2; sin(x); od;
> for x in seq1 while (x <> 2) do x^2; sin(x); od;
```

8 Manipulation and Simplification

Maple has very powerful in-built commands to manipulate expressions. Try:

```
> restart;
> p:=2*x+3*x^2+4*y+5*y^2+y^3+x*y;
> simplify(sin(x)^2+cos(x)^2);
> factor(x^2+2*x*y+y^2);
> expand((x+a)^7);
> collect(p,x); collect(p,y);
> normal(2/(x+1) + 3/(x^2+3));
> coeff(p,x,2); coeff(p,y,3);
```

9 Helpful Commands

```
??; ?assign; ?atsign; ?ditto; ?dot; ?eval; ?evalc; ?evalf; ?example;
?help; ?info; ?map; ?op; ?plot; ?plot[options]; ?quotes; ?remove; ?select;
?LargeExpressions[Veil]; ?LargeExpressions[Uneil];
?unapply; ?uneval; ?remove; ?unassign; ?zip
```

10 Procedures

Here is a simple procedure (?proc) to convert temperature in degree Celcius to degree Fahrenheit.

```

> c2f:=proc(c) local f;
> 'A procedure to convert temperature in degree Celsius
> to degree Fahrenheit';
> f:=1.8*c+32;
> RETURN(f);
> end;
> c2f(100);

```

The following sequence of commands plots (?plots) the temperature in degree Celsius versus degree Fahrenheit.

```

> clist:=[0,20,38,50,60,80,100];
> flist:=map(c2f,clist);
> plot(zip((x,y)->[x,y],clist,flist),title='C to Fahr',
labels=['degree C','degree Fahr']);

```

11 Input and Output

Maple is an interactive language and all the work can be done from within a worksheet but many times it's a good idea to have a separate text file with all the commands and then read in that file. To read in a file use the read (?read) command:

Example: `> read('c:/assignments/maplefiles/circuits/a1.ma');`

Remember that the complete file name needs to be specified within two back quotes.

Also see ?write, ?save, ?load.

12 Miscellaneous

All the predefined Maple functions and commands start with lowercase letters. There is also another form of most of the mathematical functions which starts with an uppercase letter. This form is called the inert form. It is called inert because it doesn't perform any mathematical function but is simply used to provide a literal description of the function in mathematical equations. Try:

```

> Limit(sin(x)/x,x=infinity)=limit(sin(x)/x,x=infinity);
> Int(x^2,x=a..b)=int(x^2,x=a..b);

```

12.1 Can I do this?

As mentioned in the beginning Maple is a functional programming environment. You can expect to find an equivalent function or a sequence of functions in Maple for anything you want to do with and to functions, so look for it in the Contents under the help menu item in the worksheet or try ?contents.