

2. Introduction to *Maple*

What is *Maple*? *Maple* is a computer algebra system (CAS). What is a CAS? As the name suggests, it is a computer software program that manipulates sophisticated algebraic symbols and expressions - - saving you the time, tedium, and frustration of doing these by hand. But as you shall see, it is much more. It can do some very advanced, difficult calculations, generate impressive graphs, and generate numerical and symbolic solutions to problems that may be intractable by hand. If used properly, a CAS allows the user to concentrate more on the concepts of mathematics rather than spending time doing endless, time-consuming computations.

A CAS is a software package that works directly with an expression as a basic data structure (compare this to most software packages that require numeric data and perform numeric operations on the data). *Maple* is one of several CASs; other major examples are Mathematica, Macsyma, Derive, and MathCad.

In this introductory section, the rudiments of using *Maple* are given. Discussed in this section are the following topics.

1. The *Maple* Environment
2. Essentials for Using *Maple*
3. Functions and Operators
4. Plotting with *Maple*
5. Using *Maple* Packages
6. Programming with *Maple*

1. The *Maple* Environment

When you use *Maple* on the computer, you work on a *Maple* worksheet. This document was written on a *Maple* worksheet. It is the means by which you input, output, save, and print information. Every *Maple* worksheet supports any combination of inert commentary text, active *Maple* input, *Maple* output, and graphics/animation.

The only means of computing in *Maple* is through an execution group on the worksheet denoted as a square bracket [and located on the left side of a worksheet. The following is an example of an execution group enclosing inert commentary text, two types of active *Maple* input and their respective *Maple* outputs. The results of the computation are at the end of the execution group.

[The syntax in *Maple* for computing the integral of a function f with respect to x is $\text{int}(f(x), x)$ or $\text{Int}(f(x), x)$. In the first case, without the capital "i", *Maple* will return the unevaluated integral of the function. In the second case, with the "i" capitalized, *Maple* will return the evaluated integral.
Consider the integral of the function $f(x) = x^2$.
> **int(x^2, x);**
> **Int(x^2, x);**

$$\frac{1}{3}x^3$$
$$\int x^2 dx$$

Maple is a case-sensitive language; lower-case letters and upper-case letters are distinct.

[>

When a *Maple* worksheet is executed by repeatedly pressing the enter key, the cursor will jump to the next prompt `>` within the next execution group. The empty execution group above is to prevent *Maple* from jumping too far on the worksheet and passing by the narrative.

There are basically three types of text on a worksheet: (1) inert text, (2) active *Maple* input, and (3) *Maple* output.

1. Inert text or inert in-line *Maple* input: The text you are reading now is an example of inert commentary text which can be inputted *within an execution group* by selecting *Insert/Text Input* or by clicking the icon



. To input non-executable mathematical notation within your text such as $\pi = 2 \arcsin(1)$ or an expression like $\int x^2 dx$, you must remove the command prompt `>` by placing the cursor just to the right of it



and then select *Insert/Text Input* or click the icon . Inert *Maple* input in standard mathematical



notation can then be inserted in-line by clicking the sigma icon or by selecting *Insert/Maple Input*.

Both commands create a box enclosing a bold question mark and an edit field near the top of the *Maple* window where you can enter *Maple* code, such as `Pi=2*arcsin(1)` or `int(x^2, x)`, which corresponds to the mathematical notation you seek to input. You can also remove the execution group altogether to insert inert text by clicking the execution group with the mouse and then hitting the keyboard delete button.

2. Active text or active in-line *Maple* input: Active input is a mathematical statement you want *Maple* to evaluate, and it may be inputted within an execution group by selecting *Insert/Math Input*. The input text

is bright red in color and *Maple* evaluates the command when you hit enter or the icon



3. *Maple* output: Output, by default, is blue in color and in typeset notation. You can change the display of output by selecting *Options/Output Display*.

[`>`

The icons displayed toward the top of the *Maple* window contain pre-programmed commands that can enhance and expedite the creation of a *Maple* worksheet. A description of each of the icon-buttons is provided. Other commands will be addressed when the need arises.



This creates a new, untitled *Maple* worksheet.



Clicking the open folder prompts *Maple* to open a file. This icon is equivalent to invoking *File/Open*.



Clicking the diskette saves the active *Maple* worksheet. The printer icon takes the user to the print menu where you can select which printer to be used, the print range, and the number of copies.



Cut, copy, and paste are frequently used commands. You can highlight any region of *Maple* with the mouse or shift-arrow keys combination, cut or copy the highlighted region, and then paste the highlighted region to the present cursor location.



Clicking the curved arrow will undo the last deletion.



There are two different types of inert commentary text. You can input plain commentary text, such as this sentence, by clicking the (T) icon-button, or you can input inert mathematical statements within pre-existing text by clicking the sigma (Σ) icon. Clicking this icon prompts a bold question mark surrounded by a box and an edit field at the top of the *Maple* window where you can then input text in standard mathematical notation.



This inserts a new *Maple* input prompt after the cursor. To remove a line, highlight the line you wish to delete, and then hit the delete button on your keyboard.



The right arrow introduces subsections into a section. The left arrow removes subsections. Subsections, as shown below, are useful for organizing a worksheet.

subsection one



It is easy for optimistic programmers to write something which their computer is unable to handle or their patience is unwilling to wait for. When a computation is activated, the stop button brightens from gray to red. Clicking the stop button halts the active computation.



These three buttons control the magnification of the entire *Maple* worksheet. By default, the smallest x is depressed which represents 100% or normal magnification. Depressing the middle-sized x enlarges the magnification to 150%, and the largest x magnifies the workspace to 200%.



There are invisible word processing characters hidden within a *Maple* worksheet. You can see these hidden codes by depressing the button shown here. To continue to hide these invisible characters, click the button a second time.



Clicking this four-arrow button pushes the active *Maple* worksheet to fill the entire *Maple* window. This command is only available for certain versions of *Maple*.



Clicking on the first icon toggles the display from inert or active text such as $\int e^x dx$ to the actual *Maple* code `int(exp(1)^x, x)`. Hitting the first icon a second time changes the display from *Maple* code to text format. The second icon, which looks like a maple leaf with an integral sign, toggles a command from inert to active or vice versa.



Inputting formulas often requires the use of parentheses. For example, inputting the expression

$$\frac{y(x - \sin(\pi x) + e^x)}{x - 1}$$

as an active *Maple* command would require the following use of parentheses.

```
[ > y*(x-sin(Pi*x)+exp(1)^x)/(x-1);
```

$$\frac{y(x - \sin(\pi x) + (e)^x)}{x - 1}$$

The total number of left parentheses used must equal the total number of right parentheses used. Hitting this button auto-corrects any imbalance in parentheses usage. Be careful when using this feature! What *Maple* adjusts may not be what you intended to input.



Clicking this button executes every active statement within an execution group in the order they are listed.



The bold, *italics*, and underline commands are only available when the cursor is in a section of inert text. To use these buttons, simply highlight the area of text you wish to change and then click any *combination* of the respective icons.



These buttons control the alignment of text. After highlighting a region of text, click the first button to align text to the left, the middle button to center text, and the third button to align text to the right.

Help. The on-line help feature in *Maple* is very good -- once you learn how to use it. The help pages describe the syntax of each command, include a brief description of the algorithm used to implement the command, and provide a few examples illustrating the use of the command. The examples can be very helpful. One can often cut and paste an example from a help page into the worksheet, confirm that the example works as advertised, then modify the arguments to answer the question at hand.

For example, to obtain help on the command **plot** type **?plot** at a *Maple* prompt **>** and then press enter. For example,

```
[ > ?plot;
```

Alternatively, you can go the Help menu on top of the screen and select one of the many options such as the topic search or information on using help.

```
[ >
```

2. Essentials for Using *Maple*

It is a good idea to begin EVERY worksheet with the command:

```
[ > restart;
```

This ensures that all of *Maple's* memory is cleared. Future operations can be performed without risk that previous definitions and assigned constants will interfere with the processing.

Maple is a computer language; it cannot read your mind. *Maple* does follow your instructions to the letter.

This means that you need to learn to communicate your needs to *Maple* in a way that it understands.

Here are some basic symbols and other fundamentals.

Assignments are made with `:=` (plain `=` has a different meaning). The `:=` gives the value of the right hand side (RHS) to the name on the LHS. Here is an example.

```
[ > number:=2^3;
                                     number := 8
[ >
```

Every active Maple statement must end with a **semicolon** (`;`) or a **colon** (`:`). Executing a *Maple* instruction with a semicolon activates *Maple* to perform the command and *display* the output. A colon activates *Maple* to perform the command and *hide* the output display.

The three most recently executed commands are temporarily stored and may be retrieved by the percentage operators. The single **percentage sign** (`%`) refers to the first previously executed command, the double percentage sign (`%%`) refers to the second previously executed command, and the triple percentage sign (`%%%`) refers to the third. The following demonstrates the use of percentage signs.

```
[ > 11;
                                     11
[ > 8 + %;
                                     19
[ > % * %%;
                                     209
[ > (%%% - %) / %%;
                                     -198
                                     19
[ > (% * 10) + 55;
                                     -935
                                     19
[ >
```

If we change the initial value of 11 to say 6 and execute, *Maple* does not automatically adjust other terms even if those statements contain a reference to the changed variable. The only means of re-evaluating terms is by re-executing them and the order in which statements are executed determines the final results.

Maple is **case sensitive**. For example, the names `x` and `X` are different, and `pi` and `Pi` are not the same.

`{ }` represents **set notation** (members of a set are unordered).

`..` (as in `a .. b`) is how *Maple* indicates the interval `[a, b]`, the real numbers from `a` to `b`.

`?` is a request to *Maple* for information

The **standard mathematical functions** are denoted by their standard names:

`+`(plus) , `-` (minus) , `*` (times) , `/` (divided by) , `^` or `**`(raised to the power) , **sin**, **cos**, **tan**, **abs**, and **sqrt**.

Common constants are predefined in *Maple*. Their names are: Pi, E, I, and infinity.
(Yes, they have to be capitalized exactly like this.)


Unless specified otherwise (with **assume**), all names are assumed to be complex-valued. This is not often a problem, but there are times when this can lead to surprising results.

Let's get the idea of how *Maple* works by letting it perform some arithmetic.

In the following lines, use paper and pencil to first predict what you think *Maple* will do. Then execute the command and see what actually happens!

```
[ > number := 4 * 6 + 12 / 6 - 1 ;
[ > power := (-3)^3 ;
[ > abs( % );
[ > Pi ;
[ > v:= sin( Pi / 4 ) :
[ > w:= v^2 ;
[ > v;
[ > tan( -Pi / 2 ) ;
[ > 3 / ( 5 - sqrt( number ) ) ;
[ >
```

Now enter some commands of your own! To produce the input prompt, either select Insert Execution

Group (After) from the Insert menu or click on the  icon . Each of these will produce a new input region below the current cursor position. Be generous with the space bar to make your commands easy to read and to edit (modify after the fact). Go back to the tangent calculation above and change it to compute the tangent of $\text{Pi} / 3$.

Now for some slightly more impressive computations.

```
[ > 2^100;
[ > evalf( Pi, 250 );
[ > 400!;
[ >
```

Now let's define a few functions and see how *Maple* operates with these. (The only difference is that in these examples the name x does not have a value.)

```
[ >
[ > f := sin( 2 * x ) ;
[                                     f:= sin(2 x)
[ > g := 4 * x^2 ;
[                                     g:= 4 x^2
[ > h := 0.25 * cos ( 8 * x ) ;
[                                     h:= .25 cos(8 x)
[ > p := f * g ;
[                                     p:= 4 sin(2 x) x^2
[ > u1 := f * h ;
[                                     u1:= .25 sin(2 x) cos(8 x)
[ > u2 := f + h ;
```

```
[ u2 := sin(2 x) + .25 cos(8 x)
[ >
```

3. Functions and Operators

```
[ > restart;
```

Built into *Maple* are the basic functions, relational operators, and logical operators. They are denoted as follows.

trigonometric: sin, arcsin, cos, arccos, tan, arctan, csc, arccsc, sec, arcsec, cot, arccot

relational: <, <=, >, >=, =, <> (not equal)

logical: and, or, not

transcendental: ln, log[b], exp

hyperbolic: sinh, cosh, tanh, sech, coth, csch

There are many other functions as well.

Functions can be composed using the composition operator @. For examples

```
[ > (cos@sin)(x);
[                                     cos(sin(x))
[ > (cos@sin)(Pi);
[                                     1
```

Repeated compositions can be performed by @@2, @@@3, ... or multiple use of the @ operator as shown,

```
[ > (cos@sin@cos)(x);
[                                     cos(sin(cos(x)))
[ > (exp((tan@sin)(x)));
[                                     etan(sin(x))
[ > (cos@@2)(x);
[                                     (cos(2))(x)
[ > (cos@cos)(x);
[                                     (cos(2))(x)
```

As mentioned in Section 2, assigning names to expressions is done by "colon equal" (:=). Once a name is assigned to an expression, that name can be used anywhere within the worksheet when you want to refer to the expression.

Consider the following assignment.

```
[ > f:=x^2;
[                                     f:=x2
```

To evaluate f at the value of x = 3 we would need to first assign x to be 3 and then compute the expression

for f.

```
[ > x:=4; f;
                                     x := 4
                                     16
```

Or we can use the substitute command to substitute the value of $x = 4$ into the expression for f.

```
[ > subs(x=4, f);
                                     16
```

We can declare f to be a function by using the `:=` command in conjunction with the arrow operator `->`. In Maple, the function $f(x) = x^2$ would be declared as

```
[ > f:=x -> x^2;
                                     f := x → x2
```

Now we can easily evaluate as we would any function. Consider for example,

```
[ > f(3), f(10), f(x), f(y), f(x+y);
                                     9, 100, x2, y2, (x + y)2
```

As indicated, to invoke the same arguments using assignments requires one to make the assignment $x := 3$, then evaluate that assignment in-context of $f := x^2$ and to continue repeating assignments and evaluations. This can become quite cumbersome! Also, with functions, we can easily consider more variables as shown here.

```
[ > g:=(x,y) -> x*sqrt(y);
                                     g := (x, y) → x√y
[ > h:=(x,y,z) -> abs(ln(x)*cos(y)*exp(z));
                                     h := (x, y, z) → |ln(x) cos(y) ez|
```

We then invoke a function by name and respective input variables.

```
[ > g(5, 17);
                                     5√17
[ > g(5.0, 17);
                                     5.0√17
[ > g(5, 17.0);
                                     20.61552813
[ > evalf(g(5, 17));
                                     20.61552813
```

where the data type of the input determines the form of the computed result. For example, the function $g(5, 17)$; has two integer inputs, 5 and 17, and therefore performs exact computations. Evaluation of $g(5.0, 17)$; and $g(5, 17.0)$; demonstrates that floating-point data types 5.0 and 17.0 sometimes activate

Maple to perform exact computations and sometimes activate *Maple* to perform approximate calculations. However, *evalf* (evaluate as floating-point) always results in approximate calculations. By default, floating-point results are approximated to 10 digits (not decimal places!) but can be adjusted to any positive integer by the command **Digits** as shown.

```
[ > Digits:=4;
                                     Digits := 4
[ > evalf(g(5, 17));
                                     20.62
```

A method of converting an assignment into a function is by means of the *unapply* command. For example, if *h* is the expression $x^3 + 2$, we can convert it into a function of *x* as follows.

```
[ > h:=x^3+2;
                                     h := x3 + 2
[ > h:=unapply(h,x);#This makes h into a function of x.
                                     h := x → x3 + 2
[ > h(4);
                                     66
```

Unassigning is the assigning of names to their original names. We can unassign $f := x^2$ by assigning *f* to *f* as

```
[ > f:=`f`;
                                     f:=f
```

In addition, executing the **restart**; command will unassign all previously assigned variables.

```
[ >
```

Operations can also be performed by *Maple* on **sets** and **lists**. In *Maple*, a set is an unordered collection of elements enclosed by $\{\}$. Consider the following sets,

```
[ > set_A:={a,b,b,c};
                                     set_A := { a, b, c }
[ > set_B:={d,a};
                                     set_B := { a, d }
[ >
```

Duplicates are automatically removed from sets and the elements of sets are re-ordered in output to intentionally show that there is no order among the elements of a set. The set of elements that are in *set_A* or *set_B* can be obtained by the *union* operator, the elements in common with two sets can be derived by the *intersect* operator, and set difference can be employed by the *minus* operator. The following demonstrates the use of these three operators.

```
[ > set_A union set_B;
                                     { a, b, c, d }
[ > set_A intersect set_B;
                                     { a }
[ > set_A minus set_A;
```

```
[
    { }
]
> set_A minus set_B;
{ b, c }
```

A list or an array is a user-specified ordered collection of elements enclosed by []. The following is a list of numbers.

```
[
> list_A:=[3,13,6,6,11];
list_A := [ 3, 13, 6, 6, 11 ]
```

Any element can be taken from the list by using the list name and element position as shown.

```
[
> list_A[1], list_A[2], list_A[3], list_A[4], list_A[5];
3, 13, 6, 6, 11
```

The *map* operator can be used to apply a command to every element of a list. The following demonstrates this feature.

```
[
> map(sqrt, list_A);
[ $\sqrt{3}, \sqrt{13}, \sqrt{6}, \sqrt{6}, \sqrt{11}$ ]
> evalf(%);
[1.732050808, 3.605551275, 2.449489743, 2.449489743, 3.316624790]
> f:=x->x^2;
> map(f,%%);
[3, 13, 6, 6, 11]
>
```

4. Plotting with Maple

Graphs of functions are produced by the plot command. In its simplest form, the plot command needs to know the function to be plotted and the range of values for the independent variable. Note that a..b is *Maple's* way of describing the interval [a,b]. Observe that signed titles are placed on some of these graphs. Again, try to predict the output before tapping the return key!

```
[
> restart;

> plot( 3 * t - 2 , t = -3 .. 10, title=`My First Maple Plot` ) ;

> f:=x->sin(2*x);
f:= x → sin(2 x)

> plot( f(x) , x = -Pi .. 2 * Pi ) ;#NOTE THAT SINCE f(x) IS NOT AN
ASSIGNED VARIABLE BUT A DECLARED FUNCTION. THUS WE MUST REFER TO IT
AS f(x) AND NOT f.
```

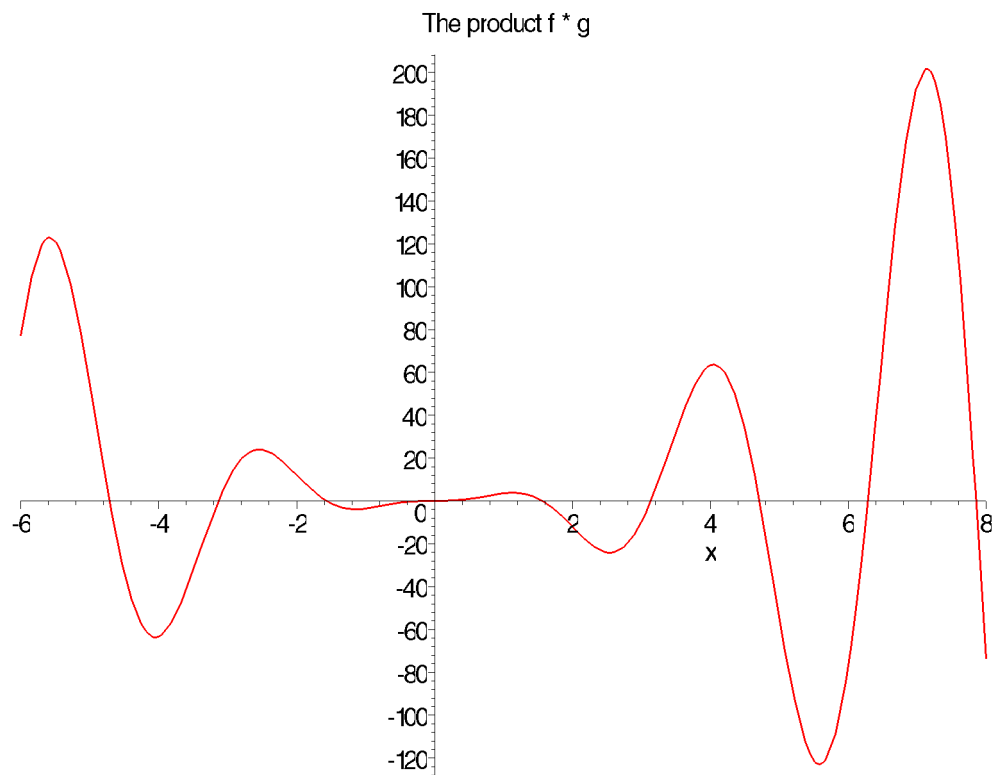
The preceding command has a remark attached. Everything after the # sign is non-executable.

```
> g:=x->4*x^2; p:=x->f(x)*g(x);
```

$$g := x \rightarrow 4x^2$$

$$p := x \rightarrow f(x)g(x)$$

```
> plot( p(x) , x = -6 .. 8 , title = ` The product f * g ` );
```



```
>
```

These plots are nice, but what information do they convey? Let's concentrate on the plot of $f \cdot g$, that is, of p . Position the cursor on a point on the graph and click the left button. The numbers that appear on the upper-left corner are the coordinates of the current location of the cursor. Use this technique to identify the global maximum and minimum values of $f \cdot g$ on the interval $[-6, 8]$ and the x -values at which these are found. Where do other (local) minima and maxima occur? Can you guess the exact values?

```
>
```

```
> plot( 2 * u^3 + 4 * u - 5 , u = -10 .. 7 ,
        title = ` cubic function` );
```

```
>
```

Where does the cubic function cross the u -axis? Maybe it would be helpful to cut down the plotting interval from $[-10, 7]$ to $[-1, 3]$ or even narrower, say to $[0, 1.5]$. Try it. Can you find the u -intercept to 2-decimal point accuracy by this process? It is a powerful method which we call ZOOMING IN.

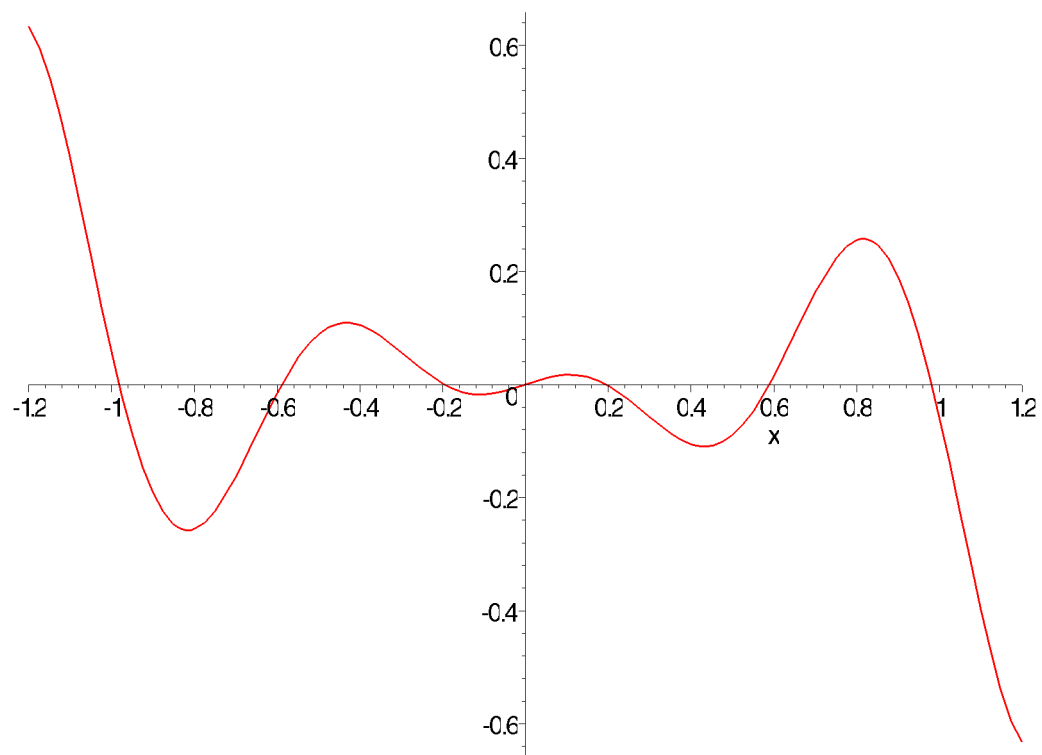
```
> h:=x->0.25*cos(8*x); u1:=x->f(x)*h(x); u2:=x->f(x)+h(x);
```

$$h := x \rightarrow .25 \cos(8x)$$

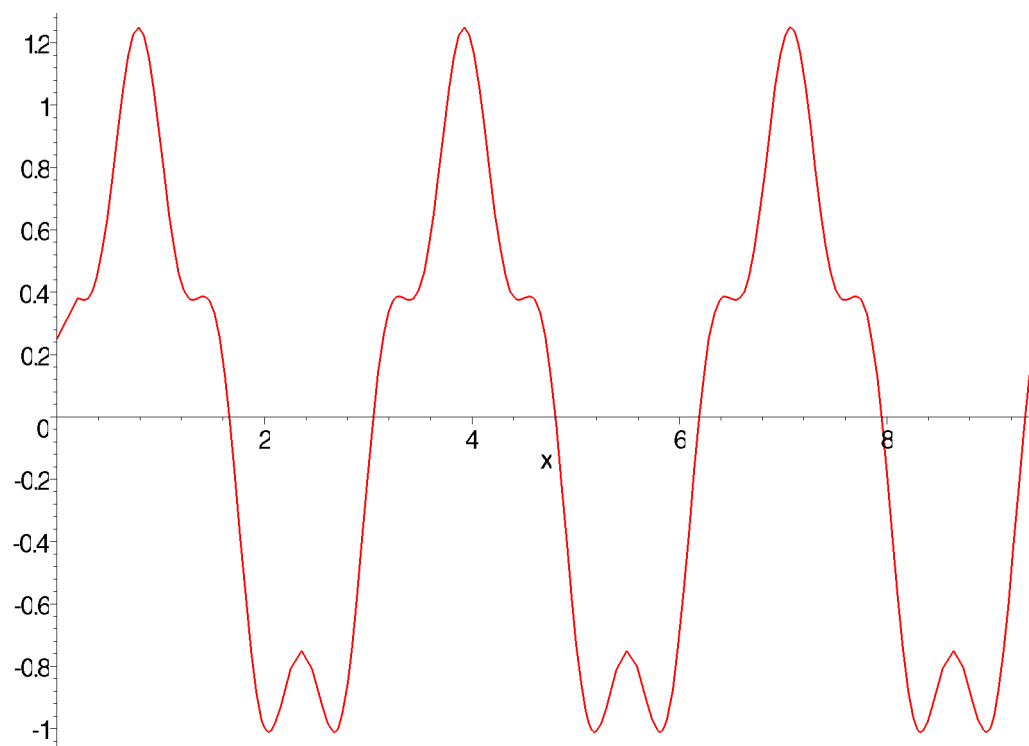
$$u1 := x \rightarrow f(x)h(x)$$

$$u2 := x \rightarrow f(x) + h(x)$$

```
> plot( u1(x) , x = -1.2 .. 1.2 );
```



```
> plot( u2(x) , x = 0 .. 3*Pi );
```

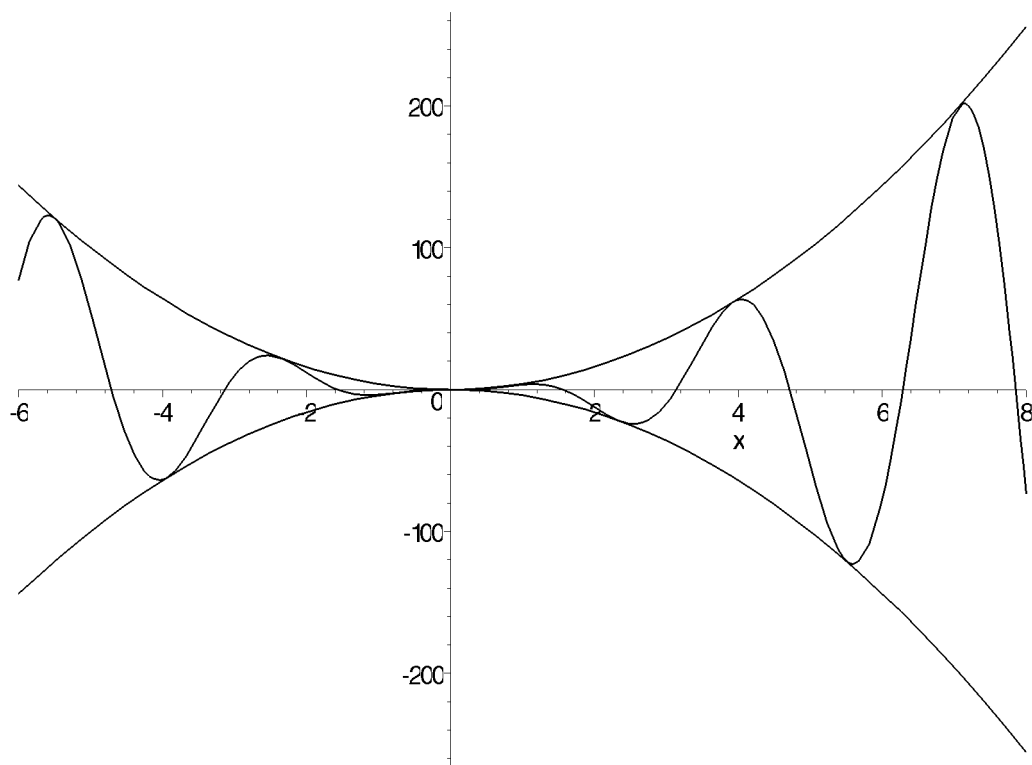


Can you explain why the graphs of u_1 and u_2 look the way they do?

You probably now have a big pile of plots cluttering up your screen. Plots can be reduced in size by putting the cursor on the plot, depressing the mouse key, putting the cursor on one of the dots on the edge around the graph, and dragging the cursor on the arrow that appears. To permanently get rid of plots that you no longer need, position the cursor inside the plot window, select with the cursor, and depress the delete button. Output can also be deleted by highlighting it and selecting the Remove Output under the Edit menu bar.

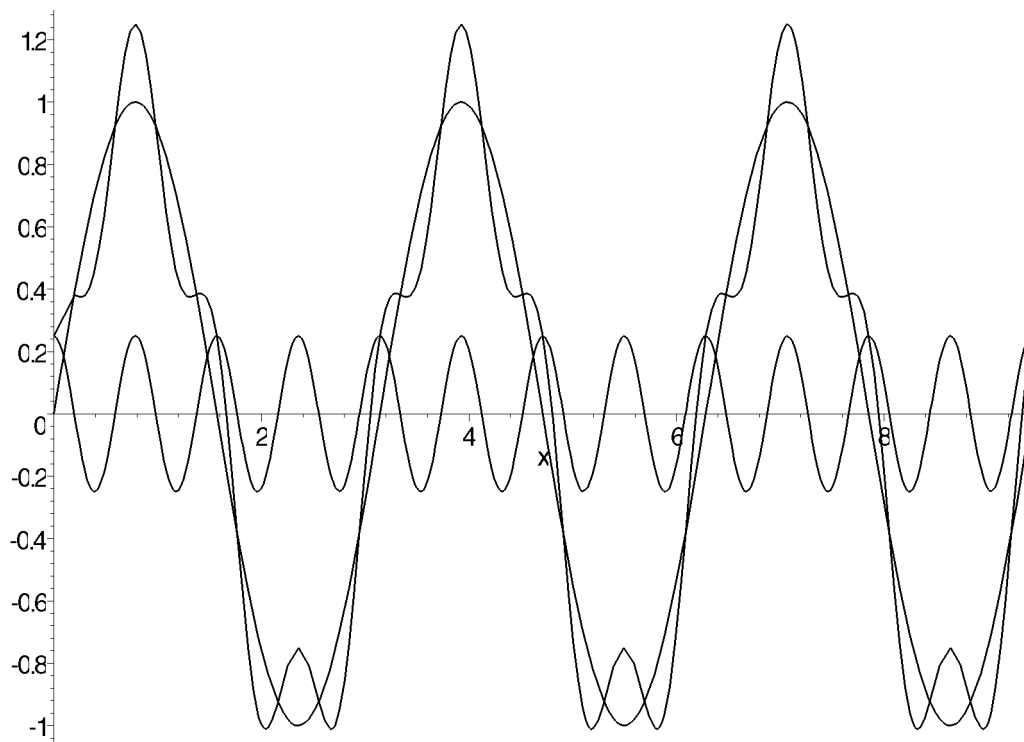
We can also plot many curves simultaneously.

```
> plot( { p(x) , g(x) , -g(x) } , x= -6 .. 8,color=black );
```



Can you identify the three different plots? What about the next one? Can you explain the little bumps that appear inside the large dips in the graph of s ?

```
> plot( { f(x) , h(x) , u2(x) }, x=0 .. 3*Pi);
```



```
[ >
```

The tangent function is well-known to all of us.

Note: This definition completely replaces the previous definition of f.

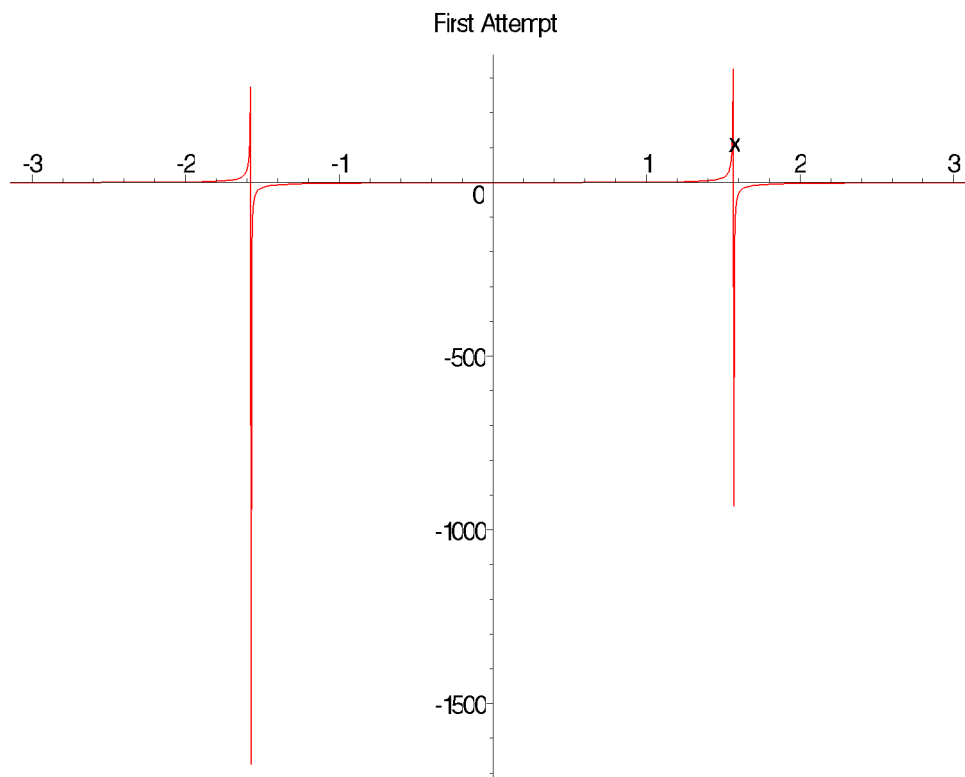
```
[ > f := tan(x);
```

```
f:=tan(x)
```

```
[ >
```

Recall that the graph of $y=\tan(x)$ has vertical asymptotes at all odd multiples of $\pi/2$. Let's see how *Maple* handles this.

```
> plot( f , x = -Pi .. Pi, title=`First Attempt` );#NOTE THAT SINCE f
WAS NOT A DECLARED FUNCTION WE USE f INSTEAD OF f(x) IN THE PLOT
COMMAND.
```

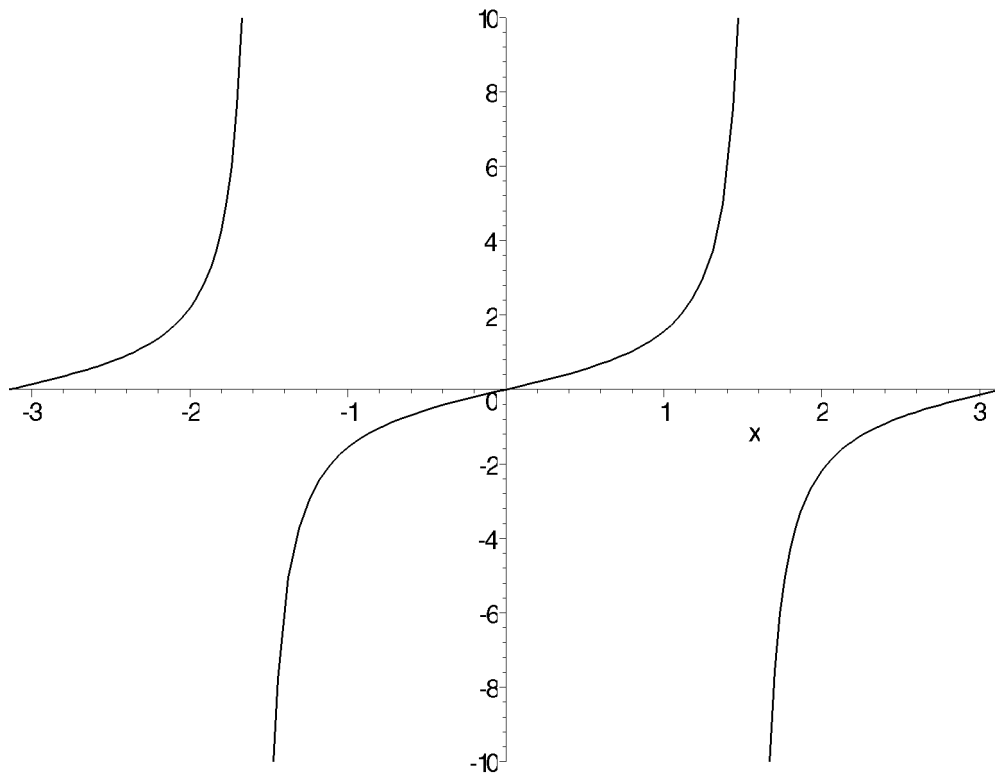


```
[ >
```

Is this what you expected to see? Why does the graph look like this? What can be done to improve the appearance of the plot?

There are two problems. First, we have to tell *Maple* that this function is discontinuous. Next, look at the labels on the vertical axis. The units are VERY LARGE on the y axis; we want to see the details on a much finer scale. Let's limit the vertical scale to the range from -10 to 10. Each of these pieces of information is an optional argument to plot.

```
> plot( f, x = -Pi .. Pi , -10 .. 10 , scont=true ,color=black);
```



```
[ >
```

This technique is very useful for any function that has vertical asymptotes.

OK. Now it's your turn. Add appropriate optional arguments to the following command to produce a graph from which you can identify the local minimum and local maximum. (Don't forget the title.)

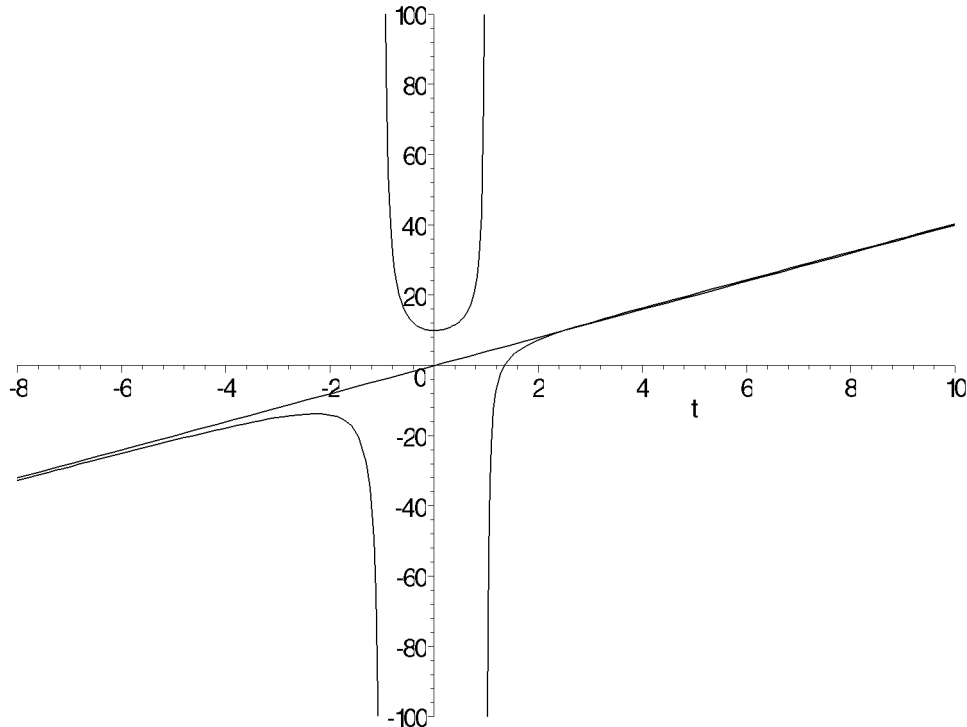
```
[ > plot ( ( 10 - 4*t^3 ) / ( 1 - t^2 ) , t = -3 .. 5 );
```

```
[ >
```

Now, one last challenge. Use your graph to estimate the (oblique) asymptote for this function. Check your estimate by graphing the function and the asymptote in the same plot. (Here's an example to demonstrate what we are seeking and to show how plots look when inserted into a worksheet.)

```
> plot({(10-4*t^3)/(1-t^2),4*t},t=-8..10,-100..100,scont=true,
color=black,title=`Discontinuous Function with Oblique Asymptote`);
```


Discontinuous Function with Oblique Asymptote



These examples only scratch the surface of *Maple's* abilities when it comes to plotting. *Maple* can also produce 3-D plots, plots of parametric curves and surfaces, and animated sequences. We will examine some of these examples in the next section about *Maple* packages. The best general source of information is the on-line help for the plots package.

```
[ > ?plots
```

```
[ >
```

```
[
```

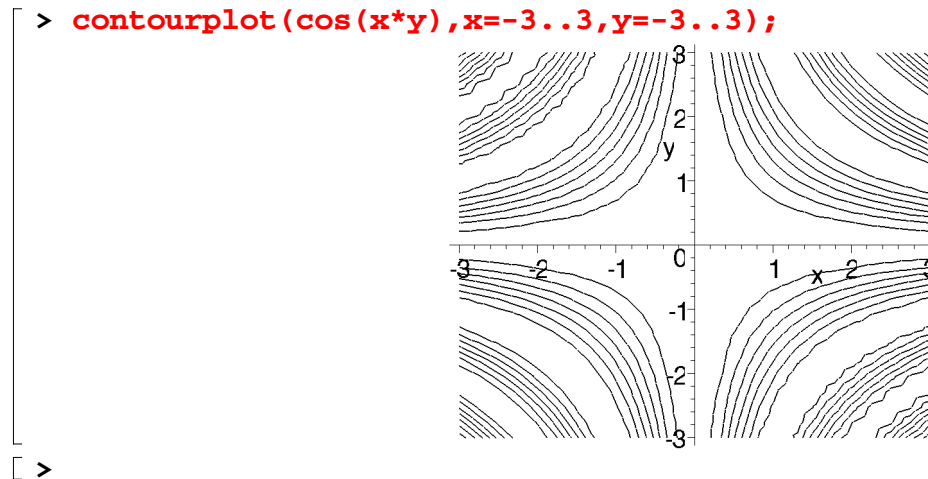
5. Using *Maple* Packages

Many of *Maple's* commands are stored in libraries that must be explicitly loaded by syntax *with(package_name)*. Libraries include the *plots* library, the *DEtools* library, the *linalg* library, to name a few. For example, the plotting library can be accessed by inputting,

```
[ > with(plots);
```

```
[animate, animate3d, animatecurve, changecoords, complexplot, complexplot3d, conformal,
  contourplot, contourplot3d, coordplot, coordplot3d, cylinderplot, densityplot, display, display3d,
  fieldplot, fieldplot3d, gradplot, gradplot3d, implicitplot, implicitplot3d, inequal, listcontplot,
  listcontplot3d, listdensityplot, listplot, listplot3d, loglogplot, logplot, matrixplot, odeplot, pareto,
  pointplot, pointplot3d, polarplot, polygonplot, polygonplot3d, polyhedra_supported, polyhedraplot,
  replot, rootlocus, semilogplot, setoptions, setoptions3d, spacecurve, sparsematrixplot, sphereplot,
  surfdata, textplot, textplot3d, tubeplot]
```

We can now access any of these commands, such as the *contourplot* command shown here.



Clicking on the image with the mouse reveals on the top of the worksheet nine new icons for two-dimensional plots. Each icon is discussed below.



There are three types of two-dimensional plotting styles: line, point and patch. The first and second icon redraw the two-dimensional plot in terms of lines and points, respectively. The third icon uses the polygon patch style with gridlines, and the fourth icon redraws the plot using the patch style without gridlines.



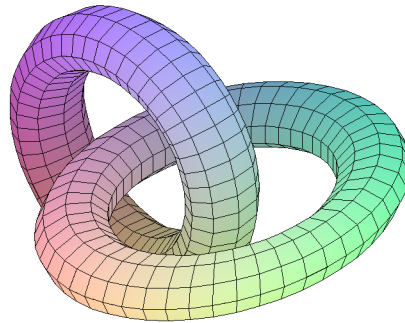
A two-dimensional plot can be drawn with the axes boxed, framed, normal, or hidden. The first icon redraws the plot with the axes boxed, the second with the axes framed, the third as normal, and the fourth icon hides the axes from the drawing.










Clicking this icon maintains a one-to-one proportion of the axes.

Another feature of the *plots* library is the three-dimensional graphics package. Perhaps the most useful feature from this utility is the opportunity to rotate the orientation of an image by mouse or by adjusting the theta and phi angle parameters located with the icons for three-dimensional plots. Try using the mouse to rotate the orientation of the tubeplot shown below, and then redraw the image by clicking the redraw icon discussed below.

```
> tubeplot([cos(t),sin(t),0],[0,sin(t)-1,cos(t)]), t=0..2*Pi,
radius=1/4);
```



[>

Clicking on the image reveals on top of the worksheet more icons that represent seven types of three-dimensional plotting styles. Each style is associated with a respective icon as follows: patch with gridlines , patch , patch with contour lines , hidden , contour , wireframe (or line) , and point .

Other icons within the three-dimensional graphics package are discussed below.



Similar to two-dimensional plots, there are four ways to control the display of the coordinate axes. The first icon redraws the three-dimensional plot with a boxed axes, the second with a framed axes, the third with a normal coordinate axes, and the fourth icon hides the axes from the drawing.



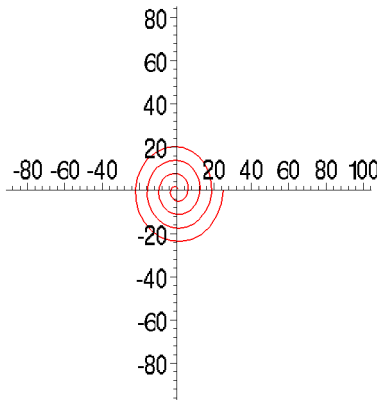
Clicking this icon maintains a one-to-one proportion of the axes.



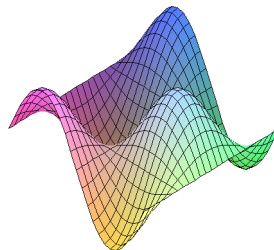
Clicking this icon redraws the three-dimensional figure.

We can also animate in two- and three-dimensions as demonstrated here.

```
> animate( [u*t,t,t=1..8*Pi],  
u=1..4,coords=polar,frames=60,numpoints=100);
```



```
> animate3d(cos(t*x)*sin(t*y),x=-Pi..Pi, y=-Pi..Pi,t=1..2);
```



```
>
```

Clicking either animation image reveals icons for two- and three-dimensional animations. The play-icon



executes the animation while the stop-icon



halts the animation.

Other animation icons are as follows.



Animations are composed of frames and hitting this icon moves the animation forward by 1 frame.



The left arrow plays the animation in the backward direction, while the right arrow executes the animation in the forward direction.



The left double-arrow decreases the speed of the animation, while the right double-arrow increases the speed of the animation.



The frames of an animation compose the animation cycle. If the first icon is pressed, the animation will run a single, complete cycle. Pressing the second icon will play the animation cycle continuously.

6. Programming with *Maple*

There are basically three aspects of programming in *Maple* that we will be using in this manual. They are the use of *repetitive loops* (for loops), the use of *conditionals* (*if - then*), and *procedures*. Let's begin with repetitive loops since they may be used in procedures.

The general syntax for a *repetitive* loop is as follows.

```
for <name> from <initial_value> by <value> to <end_value>
do
    <statement_sequence>
od;
or,
for <name> from <initial_value> by <value> while <relation>
do
    <statement_sequence>
od;
```

where the loop must be terminated by `od`; or equivalently `od:`. The expression "by <value>" is optional and *Maple* increments by +1 if this is omitted. Consider for example a repetitive loop with outputs 1, 2, and 3 all raised to the 4th power.

The repetitive loop would begin as,

```
[ > for n from 1 to 3
>
>
Warning, premature end of input
```

Since the first line of this program does not merit a semi(colon), *Maple* responds with a warning and a new line of input. The warning statement(s) disappear when the program is valid, complete and closed by a semi(colon). Our repetitive loop and output could be in either of the following forms.

```
[ > for n from 1 to 3
>     do
>         print (n^4)
>     od;
                                     1
                                     16
                                     81
> for n from 1 by 1 while n < 4
>     do
>         print(n^4)
>     od;
                                     1
                                     16
                                     81
```

Additional lines can be inserted within an execution group by placing the cursor just left of the new line operator > and hitting enter. Or, the execution group can be split at the present cursor location by selecting Split or Join from the Edit pull-down menu. Separated execution groups can be joined two at a time by highlighting neighbor execution groups and selecting Split or Join from the Edit menu and then selecting **Join Execution Groups**.

The general syntax for a *selection* statement is as follows.

```

if <relation>
    then <statement_sequence>
elif <relation>
    then <statement_sequence>
    ....
elif <relation>
    then <statement_sequence>
else <statement_sequence>
fi;

```

where the if-statement must be terminated by either fi; or fi:, elif ("else if") and else are optional, and there can be as many "else if" statements as desired.

As an example of the use of a selection statement, let's pick 1000 random numbers between 0 and 2. We will say the number is good if it is less than one. We will multiply all those between 0 and 1 together and keep a record of how many between 0 and 1 are selected. We will use the random number generator of *Maple* to select the random number.

```

[ > restart;
[ > ran_num:=rand(0..1000)/500.0:

```

Let's see some of the random numbers generated by this procedure.

```

[ > for n from 1 to 5
[ > do
[ >     lprint(ran_num())
[ > od;
.8100000000
.7340000000
1.8280000000
.4040000000
.3160000000
[ > good:=0: bad:=0:prod:=1:
[ >
[ > for n from 1 to 100
[ >     do
[ >         x:=ran_num():
[ >         if (x < 1)
[ >             then good:=good+1:
[ >                 prod:=prod*x:
[ >             else bad:=bad+1:
[ >         fi:
[ >     od:
[ > lprint(`less_than_one=`,good,`product=`,prod);
[ less_than_one= 56 product= .1351233333e-27

```

Note that the product is very small, as it should be. You can try 1000 numbers and see what product results and see if half of the numbers chosen are between 0 and 1.

[> **restart;**

Finally let's examine *procedures*. We can combine several activities together and call it a procedure. Then whenever we wish to use it we just have to call for it. The general syntax of a Maple procedure is as follows,

```
procedure_name:=proc(input_variables)
    <statement_sequence>
    <statement_sequence>
    ....
    <statement_sequence>
end;
```

where all procedures must be terminated by end; or end:. A function is an example of a procedure. We can program the function $f(x) = x^2$ as a procedure as follows,

```
[ > f:=proc(x)
>     x^2
> end;

                                f:= proc(x) x^2 end
```

If the procedure is valid and closed by a semicolon, then pressing enter activates *Maple* to display the program in a blue color. Ending the procedure with a colon hides a re-display of the program code. A procedure can then be invoked by name, in this case f , as demonstrated below.

```
[ > f(2), f(4), f(a), f(b), f(a+b);
                                4, 16, a^2, b^2, (a + b)^2
```

In constructing more complicated procedures, it is often useful to declare the scope of the variables used. A local variable is active only within the execution group where it is declared; a variable declared global retains its value for the entire worksheet. Consider this distinction within the following procedure which computes the sum of square integers from 1 to 3.

```
[ > sum_of_square:=proc()
>     global total;local n;
>     total:=0;
>     for n from 1 to 3
>     do
>         total:=total + n^2
>     od;
> lprint(`Procedure Complete`);
> end;
```

We invoke this procedure by name.

```
[ > sum_of_square();
    Procedure Complete
```

The incremental variable n was declared local by the programmer. In comparison, the variable *total* was declared global. Because of this distinction, **total** retains its value outside of this procedure while variable n does not.

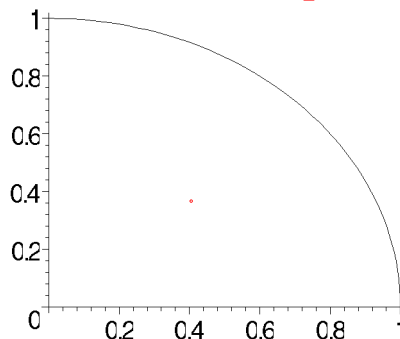
```
[ > lprint(total, n);
  14      n
```

Another application of procedures are animations. Imagine throwing darts at random onto a unit square. Inside the square is one fourth of the unit circle with the radius on two sides of the square. We can animate this process by plotting the random data values within the unit square in the first quadrant. Some of the plots will land within the unit circle (where *arc()* is located within the *plottools* library), and plots will be stored in sequence as single frames of an animation via the dot (.) operator. The frames are implicitly declared global; all other variables used within this procedure can be declared local.

```
[ > with(plottools): with(plots):
[ > ran_num:=rand(0..1000)/1000.0:
[ > Darts:=proc()
[ >     local x, y, n, m, table, data;
[ >
[ >     for n from 1 to 75
[ >     do
[ >         x[n]:=ran_num():
[ >         y[n]:=ran_num():
[ >
[ >         table:=[seq([x[m], y[m]], m=1..n)]:
[ >         data:=plot(table, style=point):
[ >
[ >         frames.n:=plots[display]({ arc([0,0], 1, 0..(Pi/2),
[ >             color=black), data });
[ >     od;
[ >     print(`Procedure Complete`);
[ > end:
```

This procedure is run by name, and the frames of the animation are compiled by the *display* command, where *insequence=false* places all frames of an animation on top of one another, while *insequence=true* retains frame sequencing.

```
[ > Darts();
[
[ > display([seq(frames.n, n=1..75)], insequence=true);
```



[>

[To play the animation, click on the graph and click on the icon on top of the screen to play it.

7. Exercises

Now that you have been through the essential elements of *Maple* that will be used in this text, a good exercise is to practice some of the commands and change parameters, functions, and so forth to become more familiar with the commands and procedures given.