

## 1 Derivatives, Procedures, Integrals, Taylor Series, Limits

Maple session	Comments
<pre>&gt; diff(cos(x),x);       - sin(x) ----- &gt; diff(cos(x),x,x);       - cos(x) ----- &gt; diff(log(sin(x)),x);       cos(x)       -----       sin(x)</pre>	<p>The command <code>diff</code> does differentiation. Repeating the variable gives a second derivative. Maple knows a wide variety of standard functions. The names are generally self-evident.</p>
<pre>&gt; r := sqrt(x^2+y^2+z^2);       2 2 2 1/2       r := (x + y + z ) ----- &gt; V := proc(x,y,z) 1/r end; V := proc(x,y,z) 1/r end ----- &gt; diff(V(x,y,z),x);       x       -----       2 2 2 3/2       (x + y + z )</pre>	<p>You can define your own functions. The first command assigns the expression <math>\sqrt{x^2 + y^2 + z^2}</math> to the name <math>r</math>. The second command defines a function <math>V(x, y, z)</math> in terms of <math>r</math>. Notice that it is necessary to specify the dependent variables by including them as arguments to the Maple <code>proc()</code> function. The actual statement(s) that define the function are placed between the <code>proc()</code> and <code>end</code>. The reason for the <code>end</code> flag is that function procedures can run on to several lines, and it is necessary to indicate which is the last line. Notice that the partial derivative is taken here.</p>
<pre>&gt; diff(V(x,y,z),x,y);       x y       3 -----       2 2 2 5/2       (x + y + z )</pre>	<p>This is the way to get <math>\partial^2 V / \partial x \partial y</math>.</p>

Maple session	Comments
<pre>&gt; int(x^3*cos(x),x);       3      2 x  sin(x) + 3 x  cos(x)  - 6 cos(x) - 6 x sin(x)</pre>	Indefinite integral $\int x^3 \cos x dx$
<pre>&gt; int(sin(t)/t,t=0..x);       Si(x)</pre>	Definite integral $\text{Si}(x) = \int_0^x \sin t dt/t$ . This integral can't be reduced to more standard functions, so it is just called the sine integral and given the name $\text{Si}(x)$ .
<pre>&gt; taylor("",x);       3      5      6 x - 1/18 x  + 1/600 x  + 0(x )</pre>	The Taylor series expansion of $\text{Si}(x)$ at $x = 0$ .
<pre>&gt; Order := 10;       Order := 10  ----- &gt; taylor("",x);       3      5 x - 1/18 x  + 1/600 x  -        7      9      10 1/35280 x  + 1/3265920 x  + 0(x )</pre>	You can get more terms in the Taylor series by changing the preassigned value of <code>Order</code> .
<pre>&gt; taylor(sin(x),x=Pi);       3 - (x - Pi) + 1/6 (x - Pi) -        5      6 1/120 (x - Pi) + 0((x - Pi) )</pre>	Use <code>x=a</code> to get the Taylor series expansion about $x = a$ .
<pre>&gt; f := proc(x,h)       (sin(x+h) - sin(x))/h end; f := proc(x,h) (sin(x+h)-sin(x))/h end  ----- &gt; limit(f(x,h),h=0);       cos(x)</pre>	This is how to find a limiting value. In this case the expression defines the derivative of $\sin x$ at $x$ .

Maple session	Comments
<pre>&gt; evalf(erf(1)); .8427007929 ----- &gt; evalf(erf(2)); .9953222650 ----- &gt; limit(erf(x),x=infinity); 1</pre>	<p>The error function, known to Maple, is defined as</p> $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt$

## 2 Newton-Raphson Method for Solving Nonlinear Equations

Suppose we wanted to find the values of  $x$  for which  $\tan x = x$ . The Maple `solve` command gives us just one solution:

```
> solve(x=tan(x));
```

0

To see where the other roots lie, we can plot the two functions and see where they intersect.

```
> plot({x,tan(x)},x=0..3*Pi,y=-1..10);
```

Notice that we specify the vertical range because of the singularities in the  $\tan(x)$  function. From the plot we can see solutions around  $x = 4.5$  and  $x = 7.5$  as well as  $x = 0$ .

To find the other zeros without the aid of Maple, we can try the Newton-Raphson method. The general method finds the zeros of  $f(x)$ . In our case we will be using  $f(x) = \tan x - x$ .

The method works from a Taylor series and makes an iterative approximation to the true zero. The iterative approximation starts from a guess at the solution and generates a new guess that is supposed to be closer to the true solution. Suppose  $x_0$  is where the zero occurs. Suppose  $x$  is our guess. Then we may make a Taylor series expansion of  $f(x)$  around  $x_0$  and evaluate it at  $x_0$ :

$$f(x_0) = f(x) + (x_0 - x)f'(x) + O(x_0 - x)^2$$

We may substitute  $f(x_0) = 0$ . Then, keeping only the first two terms, we solve for  $x_0$ :

$$x_0 = x - f(x)/f'(x)$$

Because we kept only the first two terms in the Taylor series, the value of  $x_0$  is only approximate, but it gets better as we approach the true solution. The algorithm is then simply to plug our guess into the right side of the equation and get the new guess. Then repeat until our answer has converged.

Here is a Maple procedure that does the job, finding the zero near 4.5. We also introduce some new Maple techniques.

```
> diff(tan(x)-x,x);
                                     2
                                tan(x)
-----
> z0 := 4.5;
                                z0 := 4.5
-----
> newton := z0;
                                newton := 4.5
-----
> for j from 0 to 4 do z.(j+1) := evalf(z.j - (tan(z.j)-z.j)/(tan(z.j))^2);
>   newton := newton, z.(j+1);
>   od;
-----
> newton;
    4.5, 4.493613903, 4.493409655, 4.493409458, 4.493409458, 4.493409458
-----
> z5;
                                4.493409458
-----
> tan(z5);
                                4.493409460
```

What this procedure does is to generate a list of successive approximations using the Newton-Raphson technique, and to name the list `newton`. The list of approximations is a Maple object, called an “expression sequence”. An expression sequence is merely a list of expressions, separated by commas. The expressions in this case are just single numbers. The first member of this sequence is our first guess, which is called `z0`. The second member is called `z1` and represents the result of carrying out

the Newton-Raphson technique once with `z0` as input, and so on. The command `newton`; dumps the entire list.

The sequence is generated using a Maple `for` loop. The syntax is a bit different from other programming languages, but the ingredients are all there. The counter is `j`; the loop assigns the values 0, 1, 2, 3, 4. The statements in the `for` block are sandwiched between `do` and `od`. Each statement ends with a semicolon.

Two more details of note: The period `.` in `z.j` and `z.(j+1)` is one convenient way to create and refer to an indexed name. The period isn't really part of the name. The actual names are just `z0`, `z1`, etc. The purpose of the period is to cause the two parts of the name to be joined, or "concatenated".

Finally, notice that the `od`: statement ends with a colon, rather than a semicolon. With a semicolon, we would get a full report of the evaluation of the loop. Try it and see! A colon at the end of a Maple command tells Maple to do its work silently. Which punctuation is appropriate depends on how you design the computation. In this case we saved the results of the iteration in the list, so we didn't need a blow-by-blow account of the iteration. We just asked Maple to display the list variable "newton" after the computation was finished. In other applications, we may want to use the semicolon.

Now for the details of the program. The first command gets the derivative, which is to be used in the algorithm. The next two commands initialize the expression sequence `newton` and the first approximation. The next statements define the loop, in which the next term in the sequence `z.(j+1)` is calculated in terms of `z.j` according to the Newton-Raphson algorithm. We must use `evalf` to get a number. The loop also includes a statement that appends the result to the list `newton`.

### 3 Writing Maple input files

We see that Maple has all the power of a programming language, like Fortran and C. Although the examples given here are simple enough that they can be implemented in a couple of lines, you will soon encounter cases where you are writing Maple programs several lines long, and want to be able to keep them and edit them the same way you do Fortran and C programs. Doing that is quite simple. Use a standard text editor, such as emacs, and create an ASCII file. Suppose you called this file `maple_in`. Then run Maple as usual and type

```
read maple_in;
```

If your file name has special characters, such as `/` or `.`, you must enclose the name in back-quotes: e.g. `'asst01/maple.txt'`. Your file will be processed just as if you

had typed in the lines (except the lines are not echoed to the screen). Maple can also be invoked from the Unix command line (in “dumb terminal” mode) via

```
maple < maple_in
```

and the output will appear on your screen. If your input file ends with `quit`, then Maple will exit. Otherwise it will pause for further input from the keyboard.

## 4 Saving your work

If you must interrupt a Maple session and want to save the definitions and assignments you have accumulated so far, you may do so with the command

```
save 'my_session.m';
```

You are free to choose any name for your file. The `.m` extension is peculiar to Maple and requests that the file be in Maple “internal” format. Without the `.m` extension, the file will be in human readable form.

To resume Maple operations, start Maple and type

```
read 'my_session.m';
```

to restore your assignments.