Chapter 10

Programming with Maple

We have indicated in earlier chapters that Maple is a programming language and have already seen a simple `do` loop in applying Newton's Method in Chapter 4, Exercise 21. In fact, it is possible to use Maple to create highly structured programs that can perform a large number of useful tasks. Most importantly, any program written in Maple can call any of the high level commands available in Maple, or any program that you have previously written in Maple and included in the same session.

This chapter is intended as a short introduction to the Maple programming environment. This includes conditional and logical structures

`if...then...elif/else...end if, and, or, not, true, false`,

looping structures

`for/while...do...end do, next, break`

and procedural structures

`proc...end proc, args, nargs, return, module`.

10.1 Conditional and Logical structures

The first class of programming structures are conditional statements which tell Maple to perform certain commands provided a certain condition is satisfied.

EXAMPLE 1: Determine if the quadratic polynomial $3x^2 - 2x + 4$ has 2 distinct real roots.

SOLUTION: We know that ax^2+bx+c has 2 distinct real roots if the discriminant $b^2 - 4ac$ is positive. To compute this, we enter a polynomial:

```
> p:=3*x^2-2*x+4;
```

$$p := 3\,x^2 - 2\,x + 4$$

read off the coefficients:

```
> a,b,c:=coeff(p,x,2), coeff(p,x,1), coeff(p,x,0);
```

$$a,\, b,\, c := 3,\, -2,\, 4$$

and test the discriminant:

```
> if b^2-4*a*c > 0
> then "2 real roots"
> else "not 2 real roots"
> end if;
```

"not 2 real roots"

So the polynomial does not have 2 real roots.

Notice we entered the `if...then...else...end if` on multiple lines. You can do this by pressing ⟨SHIFT-ENTER⟩. If you just press ⟨ENTER⟩, the commands will work but you get a warning message that the input is incomplete. Putting the `then` and `else` clauses on separate lines is good style, but is not necessary.

In this example, we could have just computed the discriminant:

```
> b^2-4*a*c;
```

$$-44$$

to see that it is negative. However, if this code is part of a larger computation and Maple needs to do something else based on the result of this test, then there might not be a human to decide if the discriminant is positive or negative.

"If" statements may be nested or additional conditions may be included by adding `elif` clauses. (This is an abbreviation for "else if.")

EXAMPLE 2: Determine if the quadratic polynomial $x^2 - 4x + 4$ has 2 real roots, 1 real root or 2 complex roots:

SOLUTION: We add an `elif` clause to our `if` statement:

```
> q:=x^2-4*x+4;
```

$$q := x^2 - 4\,x + 4$$

```
> a,b,c:=coeff(q,x,2), coeff(q,x,1), coeff(q,x,0);
```

$$a,\, b,\, c := 1,\, -4,\, 4$$

```
> if b^2-4*a*c > 0
> then "2 real roots"
> elif b^2-4*a*c = 0
> then "1 real root"
> else "2 complex roots"
> end if;
```

"1 real root"

Repeat this with the polynomials `q:=x^2-3*x+4:` and `q:=x^2-5*x+4:` to see the difference in output.

In general, the syntax for an "if" statement is:

`if` *logical condition*
  `then` *statements*
  `elif` *logical condition*
    `then` *statements*

```
    else statements
end if;
```

Here, the `elif...then...` and `else...` clauses are optional and there may be an arbitrary number of `elif...then...` clauses. The *statements* following `then` or `else` can be any valid Maple commands, with each statement ending in either a colon or semicolon. The *logical conditions* following `if` or `elif` must evaluate to `true` or `false` and may be constructed using, for instance, the math and set relations:

Relation	**Math Symbol**	**Maple Symbol**
equal	$=$	`=`
not equal	$\neq$	`<>`
less than	$<$	`<`
less than or equal	$\leq$	`<=`
greater than	$>$	`>`
greater than or equal	$\geq$	`>=`
subset	$\subseteq$	`subset`
element of	$\in$	`in`

as well as the logical and set operators:

Operator	**Math Symbol**	**Maple Symbol**
and	$\wedge$	`and`
or	$\vee$	`or`
exclusive or		`xor`
not	$\neg$	`not`
implies	$\Rightarrow$	`implies`
union	$\cup$	`union`
intersection	$\cap$	`intersection`
set minus	$\setminus$	`minus`

The order of precedence for these operators may be found by typing `?precedence`, but to be safe *use parentheses* if you are unsure. You can also use an explicit `true` or `false`.

EXAMPLE 3: Given three numbers a, b and c, put them in ascending order and rename them x, y and z. Use two `if` statements: the first to put a and b in order and the second to put c in the correct position. Try it on $a = 7$, $b = -2$ and $c = 4$.

SOLUTION: Use intermediate variables m and n to denote a and b in ascending order:

```
> a,b,c:=7,-2,4;
```

$$a,\ b,\ c := 7,\ -2,\ 4$$

```
> if a<b
> then m,n:=a,b;
> else m,n:=b,a;
> end if;
```

$$m,\ n := -2,\ 7$$

```
> if c<m
> then x,y,z:=c,m,n;
> elif c<n
> then x,y,z:=m,c,n;
> else x,y,z:=m,n,c;
> end if;
```

$$x,\ y,\ z := -2,\ 4,\ 7$$

Now try it with other choices of a, b and c.

10.2 Looping Structures

The second class of programming structures is looping structures (`do` loops) which tell Maple to perform certain commands repeatedly, with specified changes, until a certain condition is satisfied. You have already seen some examples of `do` loops in Chapter 4, Exercise 21 and Section 8.4.

In general, the syntax for a "do" loop is:

`for` *index* `from` *start* `by` *step* `to` *finish*
  `while` *logical condition*
  `do`
    statements
`end do;`

Here, the *statements* between `do` and `end do` are executed repeatedly, with the variable *index* initially having the value *start* and incrementing each time by the amount *step* until either *index* becomes larger than the value *finish* or the *logical condition* in the `while` clause becomes false.

The `for...`, `from...`, `by...`, `to...` and `while...` clauses are all optional. If the `from...` or `by...` clauses are missing, then the values of *start* and *step* are both taken to have the default value `1`. If the `to` and `while` clauses are both missing, then the loop will be repeated forever or until a `break` command is executed (as described below).

The *logical condition* in the `while` clause must evaluate to `true` or `false` and is constructed exactly like those in an `if` statement. The *statements* between `do` and `end do` can be any valid Maple commands, with each statement ending in either a colon or semicolon.

EXAMPLE 1: Find the first 4 derivatives of $f(x) = \sin(x)\cos(x)$ and see if you can find a pattern.

SOLUTION: Define the function and use a `do` loop to compute the derivatives:

```
> f:=sin(x)*cos(x);
```

$$f := \sin(x)\cos(x)$$

```
> for n to 4 do
> diff(f,x$n)
> end do;
```

$$\cos(x)^2 - \sin(x)^2$$

$$-4\sin(x)\cos(x)$$
$$-4\cos(x)^2+4\sin(x)^2$$
$$16\sin(x)\cos(x)$$

NOTE: We entered multiple lines by pressing ⟨SHIFT-ENTER⟩. Further, we did not type a `from` or `by` clause; so by default, the index `n` starts at 1 and steps by 1 on each iteration.

Notice that $f^{(2)} = -4f$ and $f^{(3)} = -4f^{(1)}$. If n is even, so that $n = 2k$, then $f^{(2k)} = (-4)^k f$, and if n is odd, so that $n = 2k+1$ then $f^{(2k+1)} = (-4)^k f^{(1)}$.

Finally, there are two other commands which are used to control the flow in `do` loops: `next` and `break`. The `next` command causes Maple to skip the rest of that iteration and go on to the *next* iteration. The `break` command causes Maple to *break* out of the loop by skipping the rest of that iteration and all remaining iterations. Here is an example:

EXAMPLE 2: Find the first 10 derivatives of $f(x) = \sin(x)\cos(x)$. Evaluate each at $x = 0$. If a given derivative is positive at $x = 0$, also find its value at $x = \pi/2$. If the second evaluation is also zero, go on to the next derivative. If the second evaluation is negative, don't bother to compute any higher derivatives.

SOLUTION:

```
> for n to 10 do
> derf[n]:=diff(f,x$n);
> derf0[n]:=eval(derf[n],x=0);
> if derf0[n]<0
> then break
> elif derf0[n]=0
> then next
> end if;
> derfp[n]:=eval(derf[n],x=Pi/2)
> end do;
```

$$derf_1 := \cos(x)^2 - \sin(x)^2$$
$$derf0_1 := 1$$
$$derfp_1 := -1$$
$$derf_2 := -4\sin(x)\cos(x)$$
$$derf0_2 := 0$$
$$derf_3 := -4\cos(x)^2 + 4\sin(x)^2$$
$$derf0_3 := -4$$

Notice the first derivative is positive at $x = 0$, so Maple also computes the first derivative at $x = \pi/2$. The second derivative at $x = 0$ is zero, so Maple skips the second derivative at $x = \pi/2$. The third derivative at $x = 0$ is negative, so Maple skips the remaining derivatives.

Not all problems involve taking derivatives, and Maple can be applied to those problems as well. Here is an example.

EXAMPLE 3: **The Birthday Problem:** If n people are chosen at random, what is the probability that at least two of them have the same birthday? What is the smallest number of individuals necessary for the probability of a simultaneous birthday to be at least one half?

SOLUTION: We actually compute the probability that all the people have different birthdays and we gradually increase the number of people. If there are two people in a room, the likelihood that the second has a birthday different than the first is $\frac{364}{365}$. If neither of the previous individuals have the same birthday, then the likelihood that the third individual has still a different birthday is $\frac{364}{365} \times \frac{363}{365}$. It is clear how one would compute the probabilities for additional people, but the process of doing so is boring.

Such a problem is perfect for a Maple loop. We start with probability $p = 1.0$, (as a decimal to force all the results to be decimal) then repeatedly multiply by the respective factors while p is greater than one half. On the first iteration that the probability drops below the threshold, the loop ends. If the loop ends with a colon, the intermediate steps are not printed unless one uses a print statement.

```
> p:=1.0:
> for n from 2 while p>0.5 do
> p:=p*(366-n)/365:
> print(n,p);
> end do:
```

2, 0.9972602740
3, 0.9917958342
4, 0.9836440877
5, 0.9728644266
6, 0.9595375167
7, 0.9437642973
8, 0.9256647079
9, 0.9053761663
10, 0.8830518225
11, 0.8588586219
12, 0.8329752115
13, 0.8055897252
14, 0.7768974885
15, 0.7470986808
16, 0.7163959953
17, 0.6849923353
18, 0.6530885827
19, 0.6208814745
20, 0.5885616170

21, 0.5563116655
22, 0.5243046929
23, 0.4927027663

This says that if there are 23 people in a room, there is only a 49% chance they all have different birthdays.

10.3 Procedures

Once you have written some code, you may want to reuse it with different input data. You don't want to have to retype the code every time with slight changes to the variable names or values. So you can write your own Maple commands with variable input (called parameters or arguments). These are called Maple programs or procedures. For example a procedure to add a first number to 3 times a second number is defined by

```
> mysum:=proc(a,b)
> a+3*b;
> end proc;
```

$$mysum := \mathbf{proc}(a,\, b)\, a + 3 * b \;\; \mathbf{end\ proc}$$

It is called with `a=2` and `b=1` by executing

```
> mysum(2,1);
```

$$5$$

The arguments in the call can be expressions and are totally unrelated to the names of the parameters used in the definition of the procedure. For example:

```
> mysum(b,x+y);
```

$$b + 3\,x + 3\,y$$

Notice that parameter `a` in the definition has been replaced by the argument `b` in the call, and `b` has been replaced by `x+y`.

As an aside, you may be surprised to realize that you have already been writing simple Maple procedures in earlier chapters. These are the functions defined using arrow notation. You may be puzzled because they do not seem to have the form given above. However, the function defined by

```
> f:=(x,y)->x+3*y;
```

$$f := (x,\, y) \rightarrow x + 3\,y$$

is actually a shorthand for the procedure

```
> proc(x,y) x+3*y end proc;
```

$$\mathbf{proc}(x,\, y)\, x + 3 * y\, \mathbf{end\ proc}$$

which is equivalent to the procedure `mysum` above. The arrow symbol tells Maple to create a procedure with parameters x and y and output $x + 3y$. They both produce the exact same output, e.g.

```
> f(2,1), f(b,x+y);
```

$$5,\ b + 3\,x + 3\,y$$

More generally, the syntax for a procedure is:

```
procname:=proc(parameters)
   local variables;
   global variables;
   statements
end proc;
```

and it is called by executing:

```
procname(arguments);
```

Here *procname* is the name you assign to the procedure. When you call the procedure, the values of the *arguments* in the parentheses become the values of the *parameters* in the definition; and then the *statements* are executed in order. The value returned by the procedure is *normally* the result of the last statement in the definition. (However, see the discussion of the `return` command below.) This returned value is the only thing printed out by the procedure, unless there are explicit `print` statements.

The `local` and `global` statements are optional. The *variables* in the `local` statement are only known within the procedure while the *variables* in the `global` statement are known both inside and outside the procedure. If you don't include these statements, Maple has complicated default rules for determining if the variables are local or global, and the result may not be what you intended. So it is best to include the statements, so that you are sure Maple is doing what you expect. The remaining *statements* in the procedure definition can be any valid Maple commands, with each statement ending in either a colon or semicolon.

Now let's create some procedures that are slightly more complex than the simple example given above. The first of these illustrates a use of the `for/do` looping structure.

EXAMPLE 1: In the previous section, we wrote a `do` loop to compute 4 derivatives of a function. Convert this into a procedure which will work for any function (written as an expression) and any number of derivatives. Apply the procedure to find the first 3 derivatives of the function $e^x \sin(x)$.

SOLUTION: First define the procedure

```
> derivs:=proc(f,k)
> local n;
> for n to k do
> print(diff(f,x$n))
> end do;
> end proc;
```

$$derivs := \mathbf{proc}(f,\ k)\ \mathbf{local}\ n;\ \mathbf{for}\ n\ \mathbf{to}\ k\ \mathbf{do}\ \mathrm{print}(\mathrm{diff}(f,\ x\,\$\,n))\ \mathbf{end\ do}\ \mathbf{end\ proc}$$

NOTES: : We used a local variable `n` in the procedure so that executing the procedure does not change the value of any variable `n` used outside the procedure. Also, we explicitly typed `print`, since otherwise Maple would not print out intermediate results.

Now apply this program to the function $e^x \sin(x)$ and ask for 3 derivatives:

```
> exp(x)*sin(x); derivs(%,3);
```

$$e^x \sin(x)$$

$$e^x \sin(x) + e^x \cos(x)$$

$$2\, e^x \cos(x)$$

$$2\, e^x \cos(x) - 2\, e^x \sin(x)$$

NOTE: If the procedure ends with a semicolon, Maple will print the code so that one can check for errors. Once the procedure is working properly, there is seldom any reason not to suppress that output, and one can use a colon to clean up the worksheet as done below.

EXAMPLE 2: Write a procedure that will plot a function along with its tangent lines at several points. The inputs to the procedure should be the function, the two endpoints for the plot interval, and a list of the x-coordinates of the points of tangency. Use this program to plot $4\sin(x) + \cos(2x)$ along with its tangent lines at $x = -1.75,\ 2,\ 4$ on the interval $[-3, 6]$.

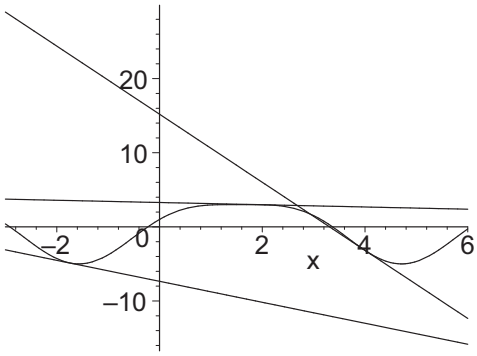
SOLUTION: The procedure (discussed below) is

```
> tan_plot:=proc(f,a,b,xs)
> local plist,x,i,xi;
> plist:=f(x);
> for i from 1 to nops(xs) do
> xi:=xs[i];
> plist:=plist,D(f)(xi)*(x-xi)+f(xi);
> end do;
> plot({plist},x=a..b);
> end proc:
```

and we apply it as follows:

```
> f:=x-> 4*sin(x)+cos(2*x):

> tan_plot(f,-3,6,[-1.75,2,4]);
```



We need to discuss the details of the procedure. First, since the procedure refers to `f(x)` and `D(f)`, we must define the function using arrow notation, not as an expression. Second, we used a new command `nops` (which stands for "number of operands"). This command returns the number of entries in a list (or, more generally,the number of operands in any operation). Consequently, the loop index `i` runs from 1 to the number of x's. The variable `xi` is defined as the i^{th} x so we don't have to recompute it three times. (As a rule of thumb, if a quantity is recomputed more than twice, give it a name.)

The variable `plist` stores the functions to be plotted. Initially `plist` is just the original function. The loop then appends each of the tangent line functions, and the `plot` command encloses it in braces. Further, we defined all the variables to be local except for the arguments of the procedure.

Finally, note that, although there are several executable statements with semicolons, we only see the plot output from the last one: a Maple procedure only returns the value of the last executable statement.

One other feature of Maple procedures is that you do not need to know in advance of the procedure call the number of arguments that will be included. This is demonstrated in the next example.

EXAMPLE 3: Rewrite the procedure from example 2 so that the inputs are the function, the x-range and y-range for the plot, and the x-coordinates of the points of tangency. Use it to plot $x^3 - 3x^2$ along with its tangent lines at $x = 0,\ 1,\ 2,\ 3$ on the interval $[-1, 4]$.

NOTE: The x-coordinates are *not* put in a list; so you don't know in advance how many arguments there will be. Maple handles this with two new commands: `args` and `nargs`. Within the procedure, `nargs` is the number of arguments and `args` is the list of arguments.

SOLUTION: The procedure (discussed below) is

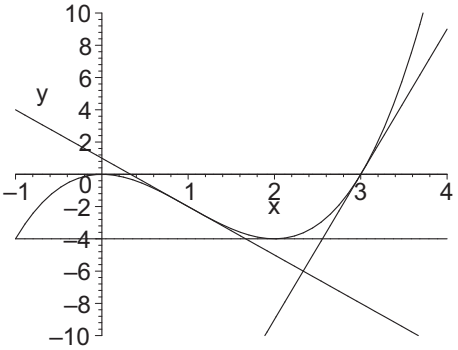
```
> tan_plot:=proc(f,xrange,yrange)
> local plist,x,i,xi;
> plist:=f(x);
> for i from 4 to nargs do
> xi:=args[i];
> plist:=plist,D(f)(xi)*(x-xi)+f(xi);
> end do;
> plot({plist},x=xrange,y=yrange);
> end proc:
```

and we apply it as follows:

```
> g:=x->x^3-3*x^2;
```

$$g := x \to x^3 - 3\,x^2$$

```
> tan_plot(g,-1..4,-10..10,0,1,2,3);
```



Again we need to discuss the details of the procedure. First, notice that when there are an indeterminate number of arguments, the first few are named in the procedure definition and the others remain unnamed. Second, the plot intervals are now given as ranges; so they must be entered in the form `a..b` instead of `a,b` which would count as two separate arguments. Third, instead of using `nops(xs)` we use `nargs` and instead of using `xs[i]` we use `args[i]`. Finally, the index `i` starts at 4, beginning after the three named arguments `f`, `xrange` and `yrange`.

A generalization of this example demonstrates the use of the `return` command to exit from the middle of a procedure and *return* a value which is not the result of the last statement in the procedure definition.

EXAMPLE 4: Rewrite the procedure from example 3 to check to see if any of the x-coordinates yield a vertical asymptote or vertical tangent. (A point $x = a$ is a vertical asymptote or tangent if $\lim_{x=a} f'(x) = \pm\infty$.) If there is a vertical asymptote or tangent, print out an error message and return that value of x. Use the procedure to plot $\dfrac{x}{x-2}$, along with its tangent lines at $x = 1$, 2 and 3, on the interval $[0, 4]$. If there is a vertical asymptote or vertical tangent at one of the indicated points, drop that x-coordinate from the list and replot.

SOLUTION: The procedure (discussed below) is

```
> tan_plot:=proc(f,xrange,yrange)
> local plist,x,i,xi;
> plist:=f(x);
> for i from 4 to nargs do
> xi:=args[i];
> if limit(abs(D(f)(x)),x=xi)=infinity
> then print("Error:  The following point is a vertical
> asymptote or vertical tangent.");
> return x=xi;
> end if;
> plist:=plist,D(f)(xi)*(x-xi)+f(xi);
> end do;
> plot({plist}, x=xrange, y=yrange, discont=true);
> end proc:
```

and we apply it as follows:

```
> h:=x->x/(x-2);
```

$$h := x \to \frac{x}{x-2}$$

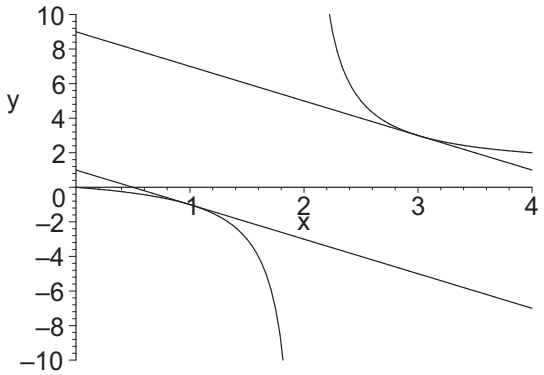
```
> tan_plot(h,0..4,-10..10,1,2,3);
```

"Error: The following point is a vertical asymptote or vertical tangent."

$$x = 2$$

Notice how Maple exits without completing the `do` loop or the remaining statements in the procedure. It did not print the result of the last statement, i.e., the `plot`. Rather it returned the value in the `return` statement, $x = 2$. Deleting 2 from the list of points, we get:

```
> tan_plot(h,0..4,-10..10,1,3);
```



Here is another situation in which programming constructs are useful. Consider the Folium of Descartes, discussed in Section 4.4. Since y is not isolated in the equation, we must use implicit differentiation to compute the slope. We first write y as $y(x)$ to tell Maple that y is a function of x.

```
> eq:=3*x*y=x^3+y^3:
> eq0:=eval(eq,y=y(x));
```

$$eq0 := 3\,x\,\mathrm{y}(x) = x^3 + \mathrm{y}(x)^3$$

Then we take the derivative of both sides, and solve for y'.

```
> diff(eq0,x);
> eq1:=diff(y(x),x)=solve(%,diff(y(x),x));
```

$$3\,\mathrm{y}(x) + 3\,x\,(\tfrac{d}{dx}\,\mathrm{y}(x)) = 3\,x^2 + 3\,\mathrm{y}(x)^2\,(\tfrac{d}{dx}\,\mathrm{y}(x))$$

$$eq1 := \tfrac{d}{dx}\,\mathrm{y}(x) = \frac{\mathrm{y}(x) - x^2}{-x + \mathrm{y}(x)^2}$$

Now, suppose that we also want the second derivative or a higher derivative. To get y'' we differentiate the first derivative equation:

```
> eq2:=diff(eq1,x);
```

$$eq2 := \frac{d^2}{dx^2}\,\mathrm{y}(x) = \frac{(\frac{d}{dx}\,\mathrm{y}(x)) - 2\,x}{-x + \mathrm{y}(x)^2} - \frac{(\mathrm{y}(x) - x^2)\,(-1 + 2\,\mathrm{y}(x)\,(\frac{d}{dx}\,\mathrm{y}(x)))}{(-x + \mathrm{y}(x)^2)^2}$$

However, we notice that the second derivative contains a y'. So we substitute the value of y' that we computed above and simplify.

```
> lhs(eq2)=simplify(subs(eq1,rhs(eq2)));
```

$$\frac{d^2}{dx^2}\,\mathrm{y}(x) = -\frac{2\,\mathrm{y}(x)\,x\,(1 - 3\,x\,\mathrm{y}(x) + \mathrm{y}(x)^3 + x^3)}{(-x + \mathrm{y}(x)^2)^3}$$

To compute even higher derivatives, it is clear what needs to be done: at each stage, we differentiate the previous derivative and then replace y' by its value in terms of x and y. However, this manual process would quickly become onerous. Such a situation is exactly where one should consider a loop structure in a Maple program.

EXAMPLE 5: Write a Maple procedure which takes an equation in x and y, an x-range and y-range for a plot and an integer n, plots the equation and returns the n^{th} derivative of y with respect to x. Use it to find y''' for the Folium of Descartes.

SOLUTION: Before writing the procedure, it is a good idea to first develop and test the central loop. Here it is for the third derivative. (Notice the use of embedded comments to help document the code. Anything following a sharp, `#`, is a Maple comment and is ignored when the line is executed.)

```
> # Initialize eqi as the first implicit derivative.
> eqi:=eq1:
> # Compute the 3rd implicit derivative
> for i from 2 to 3 do
> diff(eqi,x):
> eqi:=lhs(%)=simplify(subs(eq1,rhs(%))):
> end do:
> print(eqi);
```

$$\frac{d^3}{dx^3}\,\mathrm{y}(x) =$$

$$\frac{2\,(1 - 3\,x\,\mathrm{y}(x) + \mathrm{y}(x)^3 + x^3)\,(-3\,\mathrm{y}(x)\,x^2 - 4\,x\,\mathrm{y}(x)^3 + x^4 + 5\,x^3\,\mathrm{y}(x)^2 + \mathrm{y}(x)^5)}{(x - \mathrm{y}(x)^2)^5}$$

We are now ready to embed this loop in a procedure. Notice the method used to obtain the first implicit derivative differs from the rest. So an `if` statement is used to determine which derivative is requested. When it is the first derivative, a `return` command is used to skip the rest of the procedure.

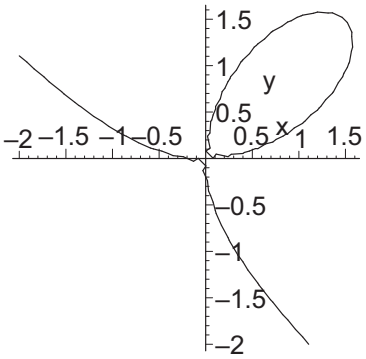
```
> implicitder:=proc(eq,xrange,yrange,n::posint)
> # Define local variables so any preexisting values
> # of i,eq0,eq1,eqi are unchanged
> local i,eq0,eq1,eqi;
> print(plots[implicitplot](eq, x=xrange, y=yrange,
> scaling=constrained, color=black));
> # replace y by y(x)
> eq0:=eval(eq,y=y(x));
> # Compute the first implicit derivative
> diff(eq0,x);
> eq1:=diff(y(x),x)=solve(%,diff(y(x),x));
> if n=1 then return simplify(eq1) end if;
> # Initialize eqi as the first implicit derivative
> eqi:=eq1;
> # Compute higher order implicit derivatives
> for i from 2 to n do
> diff(eqi,x):
> eqi:=lhs(%)=simplify(subs(eq1,rhs(%)));
> end do:
> eqi;
> end proc:
```

NOTE: One of the parameters is `n::posint`. This notation means that Maple should check the argument `n` to make sure it is a positive integer. If it is not, then an error message is generated. This powerful technique is called *type checking*. See `?type` for a list of possible types.

Here is an example of the procedure at work:

```
> eq:=3*x*y=x^3+y^3;
> implicitder(eq,-2..2,-2..2,3);
```

$$eq := 3\,x\,y = x^3 + y^3$$



$$\frac{d^3}{dx^3}\,\mathrm{y}(x) =$$

$$\frac{2\,(1 - 3\,x\,\mathrm{y}(x) + \mathrm{y}(x)^3 + x^3)\,(-3\,\mathrm{y}(x)\,x^2 - 4\,x\,\mathrm{y}(x)^3 + x^4 + 5\,x^3\,\mathrm{y}(x)^2 + \mathrm{y}(x)^5)}{(x - \mathrm{y}(x)^2)^5}$$

NOTE: In the preceding procedure, we used the `implicitplot` command from the `plots` package. If we use the `with(plots):` command, then the whole package is read in and can affect things outside the procedure. So instead, we use the full name of the `implicitplot` command which is `plots[implicitplot]`.

Everyone makes errors in writing computer programs. For help with debugging your Maple procedures, see Section 11.8.

Finally, we point out that once you have created a collection of related procedures, you can group them in a package by using the module command. See `?module` for more details.

10.4 Additional Programming Constructs

We have given only a brief glimpse of the programming capability of Maple. We encourage you to explore all of Maple's programming capability by clicking on the HELP menu. In the Standard Interface, select MAPLE HELP and then from the TABLE OF CONTENTS select PROGRAMMING. In the Classic Interface, select INTRODUCTION and then from the first column of the Help Browser select PROGRAMMING. In either case, you can the explore the help pages on GENERAL INFORMATION, FLOW CONTROL, LOGIC, DATA TYPES, NAMES AND STRINGS, OPERATIONS, NOTATION, EXPRESSIONS, PROCEDURES AND FUNCTIONS, RANDOM OBJECTS, and INPUT AND OUTPUT.

10.5 Exercises

1. Enter the expressions $f = \dfrac{x^3}{x^2+\sin(x)}$ and $g = \dfrac{x-\sin(x)}{x^2}$. Use an `if` statement to determine whether $f(3) > g(3)$, $f(3) = g(3)$ or $f(3) < g(3)$.

2. Write a `do` loop which will compute $\cos(1.0)$, $\cos(\cos(1.0))$, ..., $\cos(\cos(\cdots(\cos(1.0))\cdots))$ with 20 applications of cos. What do you think is the limit as the number of cos's becomes infinite?
   HINTS: Before the loop, execute `x:=1.0`. Then on each iteration execute `x:=cos(%)`. Plot the curves $y = x$ and $y = f(x)$ on the interval `0..1` and find the intersection using `fsolve`. With regard to the intersection point, what do you observe happening on alternate iterations of the loop? What method do you think Maple is using to find the intersection?
   NOTES: The final answer can also be obtained from `(cos@@20)(1.0)`. Further, the process of finding the limit is known as finding a fixed point of the function.

3. In Section 10.1, Example 2 tests the sign of the discriminant of a quadratic polynomial to see if the polynomial has 2 real roots, 1 real root or 2 complex roots. A similar test can be done on cubic polynomials.

   For a general cubic polynomial $ax^3 + bx^2 + cx + d$, show that the change of coordinates from x to y where $x = y - \frac{b}{3a}$ will enable us to write the

polynomial in a simplified form with no y^2 term. Moreover, by dividing by a, we can make the polynomial have lead coefficient 1, so that it looks like $y^3 + py + q$.

The discriminant of $y^3 + py + q$ is $-4p^3 - 27q^2$. If the discriminant is positive, then there are three distinct real roots. If the discriminant is negative, there is one real root and two distinct complex roots. And if the discriminant is zero, then there are repeated roots.

Write a procedure whose argument is a cubic polynomial and which applies this test to its argument. Your procedure should print the original polynomial, the simplified polynomial, the value of the discriminant, the result of the test and the roots of the original cubic equation. Test it on the polynomials $x^3 - 6x^2 + 11x - 6$, $x^3 - 5x^2 + 8x - 4$ and $x^3 - 1$.

4. The first 10 Fibonacci numbers are $1, 1, 2, 3, 5, 8, 13, 21, 34, 55$. In general,

$$F_1 = 1, \qquad F_2 = 1, \qquad F_n = F_{n-2} + F_{n-1}$$

Write a procedure which returns the n^{th} Fibonacci number.

5. Write a procedure which takes 2 numbers written in a Maple list and returns a list in either increasing order or decreasing order, depending on the second argument in the procedure.

6. Write a procedure which computes and simplifies the first n derivatives of an expression. Apply the procedure to find the first 4 derivatives of $\dfrac{e^x + e^{-x}}{e^x - e^{-x}}$.

7. Write a procedure which computes the first n derivatives of a given function at a given number $x = a$. However, if one of these derivatives is zero at $x = a$, the procedure should quit after that derivative and print out an error message. Use either functions or expressions. If you use expressions, remember that the result of a `subs` command is not automatically simplified. So `simplify` the value of the derivative. Apply the procedure to compute the first 6 derivatives of

$$\frac{\sin(x)}{\cot(x) + \tan(x)}$$

at $x = \pi/3$ unless it is zero there. Then repeat the process with the same function but at $x = \pi/2$.