Lesson 5: Newton's method

```
> restart;
```

Newton's method

Newton's method is a method of approximately solving an equation, say $f(x) = 0$. We start with an initial guess x_0, and Newton's method produces a sequence of numbers x_1, x_2, ... that converges very rapidly to a solution (when things are working well). The formula for Newton's method is

$$x_{n+1} = x_n - \frac{f(x_n)}{D(f)(x_n)}$$

I defined a function **newt** to do one iteration of Newton's method: $x_{n+1} = newt(x_n)$.

```
> newt := x -> evalf(x - f(x)/D(f)(x));
```

$$newt := x \to evalf\left(x - \frac{f(x)}{D(f)(x)}\right) \tag{1.1}$$

This was my function f

```
> f:= x -> sin(x) - x/Pi;
```

$$f := x \to \sin(x) - \frac{x}{\pi} \tag{1.2}$$

I rather arbitrarily chose my starting point

```
> x[0] := 3;
```

$$x_0 := 3 \tag{1.3}$$

I used a "for loop" to automate the iterations.

```
> for count from 0 to 6 do
    x[count + 1] := newt(x[count])
  end do;
```

$$x_1 := 2.377965170$$
$$x_2 := 2.315135129$$
$$x_3 := 2.313734857$$
$$x_4 := 2.313734132$$
$$x_5 := 2.313734132$$
$$x_6 := 2.313734132$$
$$x_7 := 2.313734132 \tag{1.4}$$

Some things to notice about the for loop:

• The whole loop (from "**for**" to "**end do**" constitutes one big Maple statement, and must go in one

execution group (delineated by the bracket in the left margin). It could all go on one line if it fits, but for better readability it's a good idea to use separate lines, indenting the body of the loop. You can use shift-Enter instead of Enter to make a new line while staying in the same group. Otherwise you get something like this:

```
> for count from 2 to 6 do


Warning, premature end of input, use <Shift> + <Enter> to
avoid this message.
```

- That body could consist of any number of Maple statements, separated by either colons or semicolons. Whether the results are printed depends only on whether there is a colon or semicolon after the final "end do".

- A "side effect" of the loop is that the loop variable **count** is left with a value, in this case 7. This can cause trouble if you try to use the same variable as a symbolic variable later. So I try to use names for loop variables that I won't be tempted to use as symbolic variables. Instead of **i, j, k** I often use **ii, jj, kk**.

In this case Maple seems to have settled on a value: x_5 was the same as x_4, and then all the later x_n would have to be the same too. Is this a solution?

```
> evalf(f(x[5]));
```

$$3.\ 10^{-10} \qquad \textbf{(1.5)}$$

Yes (to within roundoff error).

This was an example where Newton's method worked very well. But with other starting points we might not be so lucky.

To keep the x_n values we just calculated, I'll use a different name for the next table.

```
> w[0]:= 5;
  for count from 0 to 20 do
    w[count + 1] := newt(w[count])
  end do;
```

$$w_0 := 5$$

$$w_1 := -68.61162966$$

$$w_2 := -108.6418288$$

$$w_3 := -49.91370651$$

$$w_4 := -76.07621691$$

$$w_5 := -127.2974910$$

$$w_6 := -23.6349836$$

$$w_7 := 11.09525973$$

$$w_8 := -9.59479714$$

$$w_9 := -7.122720081$$

$$w_{10} := -11.48009568$$

$$w_{11} := -42.25904592$$

$$w_{12} := -11.54734007$$

$$w_{13} := -33.53680463$$

$$w_{14} := -21.85804805$$

$$w_{15} := -16.64605346$$

$$w_{16} := -9.934563679$$

$$w_{17} := -6.870102836$$

$$w_{18} := -10.04508217$$

$$w_{19} := -6.707014032$$

$$w_{20} := -9.612649594$$

$$w_{21} := -7.116658811 \quad \textbf{(1.6)}$$

It's showing no signs of settling down (eventually it does, though).

```
> for count from 21 while w[count] <> w[count-1] do
    w[count + 1] := newt(w[count])
  end do:
```

This is a slightly different form of "for loop". Instead of specifying an end value for count, I'm just telling Maple to start at count = 21 and keep going as long as $w_{count} \neq w_{count-1}$, then stop. Note that when it looks at this condition, Maple already knows w_{count} and $w_{count-1}$ from the previous iteration. This form of the loop could be a bit dangerous, though, since I don't know for sure that it will ever stop. Well, if necessary I should be able to interrupt Maple by clicking the stop sign.

```
> count;
```

$$510 \quad \textbf{(1.7)}$$

```
> w[508], w[509], w[510];
```

$$-2.313734259, -2.313734132, -2.313734132 \quad \textbf{(1.8)}$$

Data structures: Sequences, sets and lists

Data structures are ways of storing and organizing data. Maple has lots of data structures. We've seen and used some of them already, though I didn't always mention what they were. It's time I did so.

- An **expression sequence** consists of several expressions, separated by commas. For example, this is what solve gives us when it returns more than one solution.

```
> restart;
> E:= solve(x^2 - 4*x + 3);
```

$$E := 3, 1 \quad \textbf{(2.1)}$$

An expression sequence can be assigned as the value of a variable, as above, and we can refer to

individual items in the sequence using an index in square brackets, e.g.

```
> E[2];
```

$$1 \tag{2.2}$$

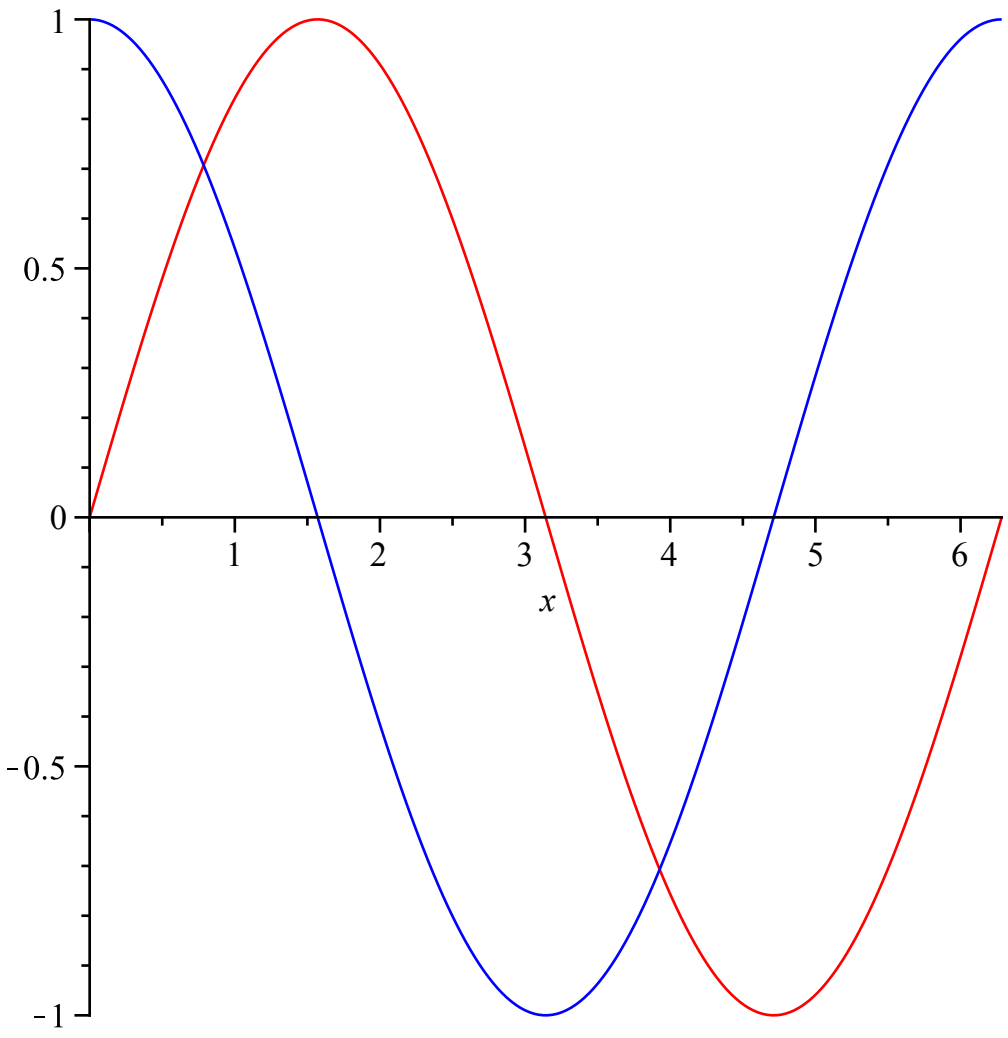
A **list** is enclosed in square brackets. If there is more than one item inside, the items are separated by commas.

```
> L := [a,b,c];
```

$$L := [a, b, c] \tag{2.3}$$

The main difference between an expression sequence and a list is that many commands can deal with a list as a single object, while the members of an expression sequence would be considered as separate inputs to the command. For example, in Lesson 3 when we wanted to plot several different expressions, we were using lists for the expressions to plot and the colours to use.

```
> plot([sin(x), cos(x)], x=0 .. 2*Pi, colour=[red,blue]);
```



Again, you can refer to the members of a list L with an index in square brackets.

```
> L[2];
```

$$b \tag{2.4}$$

You can also assign new values to the individual members of a list, thus changing the list. For example:

```
> L[2]:= c;
```

$$L_2 := c \tag{2.5}$$

```
> L;
```

$$[a, c, c] \tag{2.6}$$

(For technical reasons I won't get into, it can't be done this way if the list has more than 100 members)

A **set** is enclosed in curly brackets. Again, if there is more than one item inside, they are separated by commas.

```
> S:= {a,b,c};
```

$$S := \{a, b, c\} \tag{2.7}$$

The main differences between a set and an expresssion sequence or list are that the order of elements in a set is not significant (so {a,b,c} and {a,c,b} are the same set) and duplicate elements are removed (so a set can contain only one copy of any given expression). So this is the same set as *S*:

```
> {b,c,b,a} = S;
```

$$\{a, b, c\} = \{a, b, c\} \tag{2.8}$$

Once Maple's memory has a particular set in a particular order, it will keep that order. But one sometimes annoying thing about a set, as opposed to a list, is that you can't predict the order beforehand. Even though I entered *S* as {a,b,c}, it might have come out as {a,c,b}. You could extract a particular member of *S* with an index in square brackets, just as for an expression sequence or list:

```
> S[1];
```

$$a \tag{2.9}$$

In this case, having seen *S*, I know what this will be in this Maple session. But if I open the same worksheet again in a different Maple session, or even restart and re-execute the worksheet, I can't be sure what the result would be. In general, sets should be used in such a way that the order of their elements won't matter.

We used a set (containing one element) in the **solve** command in Lesson 2

```
> solve(a + 3*x = 5, {x});
```

$$\left\{x = \frac{5}{3} - \frac{1}{3}\, a\right\} \tag{2.10}$$

We could also solve several equations for several variables:

```
> S := solve({x + y = b, x - y = c}, {x,y});
```

$$S := \left\{x = \frac{1}{2}\, c + \frac{1}{2}\, b, y = -\frac{1}{2}\, c + \frac{1}{2}\, b\right\} \tag{2.11}$$

The result is returned as a set of equations, one for each variable we're solving for. But what if I want to extract the value of one of the variables, say *x*, without looking to see what order the set *S* is in? The trick is to use **eval**, which, as we saw in Lesson 3, can take a set as its second argument.

```
> eval(x,S); eval(y,S); eval([x,y],S);
```

$$\frac{1}{2}\, c + \frac{1}{2}\, b$$

$$-\frac{1}{2}\, c + \frac{1}{2}\, b$$

$$\left[\frac{1}{2}\,c+\frac{1}{2}\,b,\ -\frac{1}{2}\,c+\frac{1}{2}\,b\right] \tag{2.12}$$

How Newton's method works

Newton's method is based on the tangent line approximation. Near $x=x_0$ we can approximate the curve $y=f(x)$ by its tangent line at $x=x_0$, which has the equation $y=f(x_0)+\mathrm{D}(f)\,(x_0)\,(x-x_0)$. That tangent line hits the x axis at $x=x_0-\frac{f(x_0)}{\mathrm{D}(f)\,(x_0)}$, which is what we take for x_1. Then we repeat the process, using the tangent line at x_1 to get x_2, etc.

I'd like to draw a picture of this process.

```
> f:= x -> sin(x) - x/Pi;
  newt:= x -> evalf(x - f(x)/D(f)(x));
```

$$f:=x\to \sin(x)-\frac{x}{\pi}$$

$$newt:=x\to evalf\left(x-\frac{f(x)}{\mathrm{D}(f)\,(x)}\right) \tag{3.1}$$

```
> x[0]:= 3;
  for count from 0 to 6 do
    x[count+1]:= newt(x[count])
  end do;
```

$$x_0:=3$$

$$x_1:=2.377965170$$

$$x_2:=2.315135129$$

$$x_3:=2.313734857$$

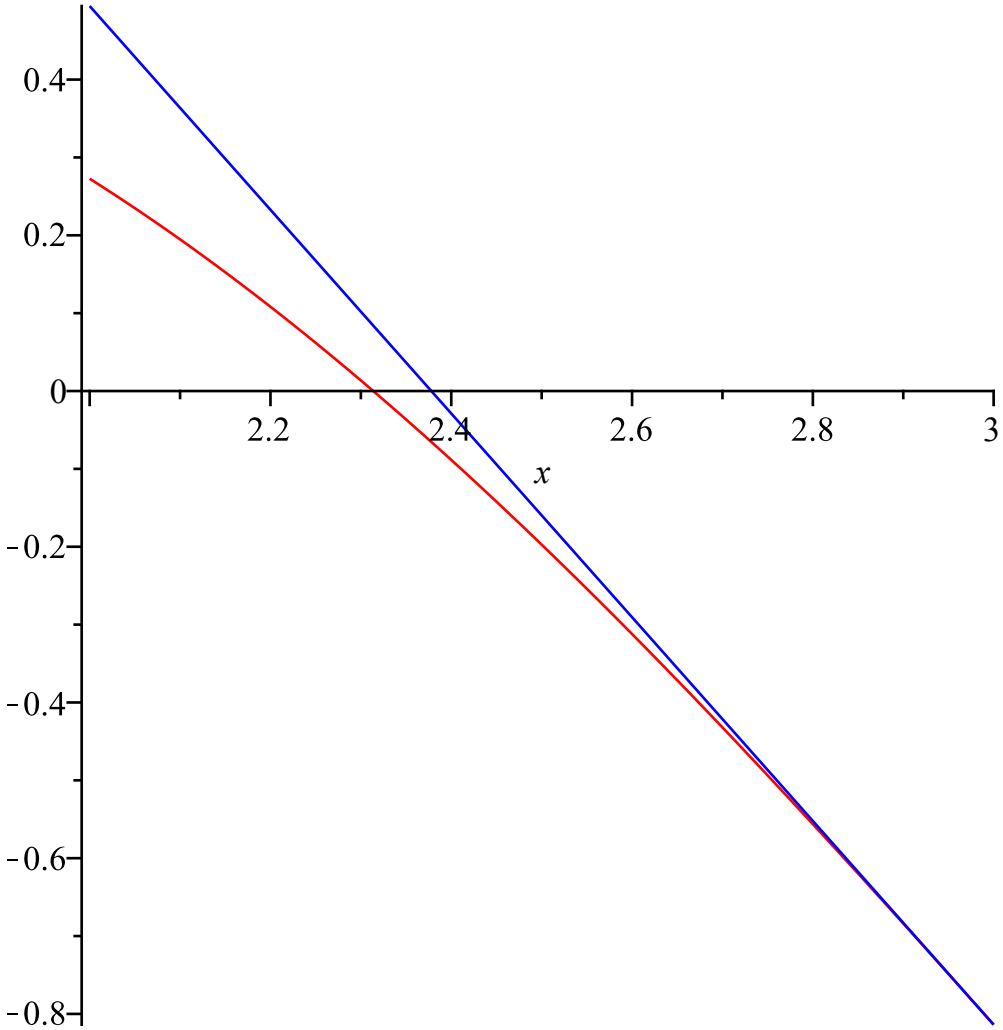
$$x_4:=2.313734132$$

$$x_5:=2.313734132$$

$$x_6:=2.313734132$$

$$x_7:=2.313734132 \tag{3.2}$$

```
> plot([f(x), f(x[0]) + D(f)(x[0])*(x-x[0])], x=2 .. 3, colour=
  [red,blue]);
```



Text and math in Maple

The Maple worksheet you submit as homework should contain text explaining what you're doing, as well as Maple input and output. Here's how to do that.
Sections help to organize your Maple worksheet. To get a new section, go to the Insert menu and choose Section.
To get a new text region, click on the T in the tool bar.

Within a text region, in Windows you can switch to 2 D *Math mode* by pressing Ctrl-R (i.e. the Ctrl and R keys together) or **1D Maple input mode** with Ctrl-M, and back to text with Ctrl-T. `Error, missing operator or ;`On a Mac, Ctrl is replaced by Command, the key that looks like this:`Error, missing operator or ;`

Yes, you can insert images too (Insert, Image...).
Alternatively, you can type the math in text mode and then convert it to 2D math by highlighting it with the mouse and choosing Format, Convert To, 2D Math.
The next section shows, among other things, how Maple can be used as a "mathematical assistant" in writing up a proof. If you want non-Maple people to read it, you can hide the Maple input: choose View, Show/Hide Contents, and remove the checkmark from Input.

An error estimate for Newton's method

Suppose the function $f(x)$ is twice differentiable and has a zero at $x = r$. If we start Newton's method at x_0, the first step takes us to $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$. I want to estimate how the "error" $x_1 - r$ compares to $x_0 - r$.

Taylor's theorem says

```
> restart;
  f(r) = f(x[0]) + `f'`(x[0])*(r-x[0]) + `f''`(c)*(r-x[0])^2/2;
```

$$f(r) = f(x_0) + f'(x_0)\,(r - x_0) + \frac{1}{2} f''(c)\,(r - x_0)^2 \qquad \textbf{(5.1)}$$

for some c between r and x_0. But our assumption is that $f(r) = 0$. So we can turn (4.1) around to say

```
> isolate(eval(%,f(r)=0),f(x[0]));
```

$$f(x_0) = -f'(x_0)\,(r - x_0) - \frac{1}{2} f''(c)\,(r - x_0)^2 \qquad \textbf{(5.2)}$$

and substitute that in to the expression for x_1:

```
> eval(x[1] = x[0] - f(x[0])/`f'`(x[0]), %);
```

$$x_1 = x_0 - \frac{-f'(x_0)\,(r-x_0) - \frac{1}{2} f''(c)\,(r-x_0)^2}{f'(x_0)} \tag{5.3}$$

Now subtract both sides from r and simplify:

```
> factor(r-lhs(%) = r-rhs(%));
```

$$r - x_1 = -\frac{1}{2}\,\frac{f''(c)\,(r-x_0)^2}{f'(x_0)} \tag{5.4}$$

If f is a smooth function and $f'(r) \neq 0$, $\frac{f''(c)}{f'(x_0)}$ will be close to the constant $\frac{f''(r)}{f'(r)}$ when x_0 is close to r. This says that for x_0 in some interval around r the error in x_1 is approximately some constant times the square of the error in x_0. Since the square of something small is extremely small, this tells us that if the initial guess x_0 is close enough to the solution r, the result of the first step of Newton's method will be much closer to r. This will again be true for subsequent steps, so we get very rapid convergence to the solution.

On the other hand, if $f'(r) = 0$, this equation is not so useful because the denominator could be very close to 0. As you'll see in a later homework question, the convergence could be quite slow in this case.

Maple aspects of that section

- I used **restart** because I didn't want to keep the definition of f from previous sections.
- To get f' or f'' as a Maple name, I enclosed it in back-quotes (to the left of 1 on the keyboard). You can make basically anything into a name by enclosing it in back-quotes.
- I used **isolate** rather than **solve**, because the thing I wanted to "solve" for was the expression $f(x_0)$, not just a variable. This will usually work when you want to put a certain subexpression on the left side of the equation by simple algebraic operations.
- After subtracting the left and right sides of the equation from r, some algebraic simplification was needed. There are several commands that could be used, including **expand**, **simplify** and **factor**. I used **factor** because the others would leave me with something involving $r^2 - 2\,r\,x_0 + x_0^2$ instead of $(r-x_0)^2$.

```
> expand((x+a)*(x+b));
```

$$x^2 + x\,b + a\,x + a\,b \tag{6.1}$$

```
> factor(%);
```

$$(x+a)\,(x+b) \tag{6.2}$$

- To get the $\neq$ sign, I used <>.

Maple objects introduced in this lesson

- `for ... from ... to ... do ... end do`
- `->`
- `unapply`
- `diff`
- `D`