

# Lesson 4: Numerical Computations; Newton's method

```
> restart;
```

## Catastrophic cancellation in the quadratic formula

One case where roundoff error can be severe is if you subtract two numbers that are very close together: the relative error of the result can be much larger than the relative errors of the inputs. This is called "catastrophic cancellation".

Here is an example of catastrophic cancellation affecting the solution of a quadratic equation. This is the solution formula from high school.

```
> s:= solve(a*x^2+b*x+c=0, x);
```

$$s := \frac{1}{2} \frac{-b + \sqrt{b^2 - 4ac}}{a}, -\frac{1}{2} \frac{b + \sqrt{b^2 - 4ac}}{a} \quad (1.1)$$

Now suppose  $b$  is large compared to  $a$  and  $c$ .

```
> r:= solve(x^2 + 444*x + 1=0, x);
```

$$r := -222 + \sqrt{49283}, -222 - \sqrt{49283} \quad (1.2)$$

Use `evalf` to get a floating-point value, with `Digits = 3`.

```
> Digits:= 3;
```

$$\text{Digits} := 3 \quad (1.3)$$

```
> evalf(r[2]);
```

$$-444. \quad (1.4)$$

No problem here, even though we're using `Digits = 3`.

```
> evalf(r[2],20);
```

$$-443.99774773632276650 \quad (1.5)$$

But what about the first solution?

```
> evalf(r[1]);
```

$$0. \quad (1.6)$$

The relative error is 100%.

```
> evalf(r[1],20);
```

$$-0.00225226367723350 \quad (1.7)$$

Even at `Digits = 10` the relative error is not insignificant: we only get 5 significant digits.

```
> evalf(r[1],10);
```

$$-0.0022523 \quad (1.8)$$

But we can actually get better relative error at the same setting of "Digits" by using a different formula that won't involve this catastrophic cancellation. Recall that the product of the roots is the constant term  $c$  divided by the leading term  $a$ .

```
> expand(s[1]*s[2]);
```

$$\frac{c}{a}$$

So the first root should be

```
> s[1] = c/a/s[2];
```

$$\frac{1}{2} \frac{-b + \sqrt{b^2 - 4ac}}{a} = - \frac{2c}{b + \sqrt{b^2 - 4ac}}$$

which won't produce a catastrophic cancellation (when b is positive). In our case:

```
> evalf(1/r[2]);
```

-0.00225

## Trouble with sums

Another situation where roundoff error can be significant is in adding small numbers to large numbers.

What happens when you add 1.23 and 456 using Digits = 3?

```
> 1.23 + 456;
```

457.

(2.1)

Note that the last two digits of the smaller number were basically lost: they had almost no influence on the final result. In a more extreme case, where the two numbers differ by at least Digits orders of magnitude, the smaller number would be lost altogether.

```
> 0.123 + 456;
```

456.

(2.2)

This can have serious consequences if you are trying to add up many numbers that are each very small (e.g. in numerical integration). Suppose we start with 0 and add 1000 numbers, each equal to .00444. The correct answer would be 4.44. And it would be, if we used Digits = 10.

```
> Digits:= 10:
total := 0:
for count from 1 to 1000 do
  total := total + .00444
end do:
total;
```

4.44000

(2.3)

This is what's called a "for loop" (we'll talk about them in more detail later). It starts out with **total** = 0, and repeats 1000 times the command **total := total + .00444**, i.e. adding .00444 to the current total and making that the new value of **total**.

But now try it with Digits = 3.

```
> Digits:= 3:
total := 0:
for count from 1 to 1000 do
  total := total + .00444
```

```
end do;  
total;
```

```
1.00 (2.4)
```

What's happening? After a certain number of iterations **total** reaches the value 1.00. After that, adding .00444 doesn't change the result.

```
> 1.00+.00444;
```

```
1.00 (2.5)
```

Even with Digits = 10, you could still have a problem if the numbers being added were small enough.

There are some rather clever ways to get around the problem (which we won't go into), but I just wanted to point it out. The main moral of the story is that, especially in a situation where you can see that one of these problems may occur, you should use a large enough setting of Digits.

People sometimes say "I only need this result to 3 significant digits" or "my data come from measurements that are only correct to 3 significant digits, so the results should only be meaningful to 3 significant digits" and are tempted to reduce Digits to 3. That's usually a bad idea. Never use fewer than the default Digits = 10. When you come to the end of your calculation, you can always round the final answer to 3 significant digits if you want. If you wish you can also go to Options, Precision, and tell Maple to "Round screen display to" a certain number of digits. This will affect the numbers you see in Maple output, without changing the numbers that Maple uses internally. For serious numerical work, use Digits = 14 or 15, or even more if roundoff error appears to be a problem. The only real problem with large values of Digits is that the computations will require more time and the data may take up more memory.

```
> evalf(2.3456787, 3);
```

```
2.35 (2.6)
```

```
> Digits:= 10;
```

```
Digits := 10 (2.7)
```

## Functions and Expressions

Up to now we have been working with expressions in Maple. A mathematical expression is something that might be constructed from constants and variables using the algebraic operators +-\* /^ and mathematical functions. For example, here we assign an expression as the value of **A**:

```
> A := sin(x) + x^3 * y/Pi;
```

$$A := \sin(x) + \frac{x^3 y}{\pi} \quad (3.1)$$

If we want to substitute values for the variables  $x$  and  $y$ , we can do so using **eval**.

```
> eval(A, {x = 2, y = 1/4});
```

$$\sin(2) + \frac{2}{\pi} \quad (3.2)$$

And this expression, which has no variables left in it, represents a number. We can use **evalf** to get its approximate floating-point value.

```
> evalf(%);
```

```
1.545917199 (3.3)
```

A function, on the other hand, can be thought of as a procedure that takes, let's say, a number (or an expression that might represent a number), does something to it, and returns another number (or expression representing a number). Maple's built-in mathematical functions include, for example, **sin** and **exp**. But you can also define your own functions. The simplest way to do so is with `->` (that's "minus" and "greater than", which together make something that looks like an arrow pointing to the right). Thus:

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

$$f := x \rightarrow \sin(x) + \frac{x}{\pi} \quad (3.4)$$

You can then evaluate the function at a particular value of  $x$  by putting the value in parentheses after the name  $f$ , just as you would for  $\sin$  or  $\exp$ .

```
> f(3);
```

$$\sin(3) + \frac{3}{\pi} \quad (3.5)$$

```
> f(x);
```

$$\sin(x) + \frac{x}{\pi} \quad (3.6)$$

```
> f(t);
```

$$\sin(t) + \frac{t}{\pi} \quad (3.7)$$

You can also have a function of several variables:

```
> g := (x,y) -> sin(x) + x^3 * y/Pi;
```

$$g := (x, y) \rightarrow \sin(x) + \frac{x^3 y}{\pi} \quad (3.8)$$

```
> g(2, 1/4);
```

$$\sin(2) + \frac{2}{\pi} \quad (3.9)$$

```
> g(u,v-t);
```

$$\sin(u) + \frac{u^3 (v-t)}{\pi} \quad (3.10)$$

An expression is tied to the particular names of the variables in it. If you assigned a value to  $x$  or  $y$ , that will affect the value of  $A$ .

```
> x := 5;
```

$$x := 5 \quad (3.11)$$

```
> A;
```

$$\sin(5) + \frac{125 y}{\pi} \quad (3.12)$$

But the  $x$  in the definition of the function  $f$  or  $g$  is just a "dummy variable", and these functions are not affected.

```
> f(t); g(u,v);
```

$$\sin(t) + \frac{t}{\pi}$$

$$\sin(u) + \frac{u^3 v}{\pi} \quad (3.13)$$

If you just look at the function `f` itself, you won't see the definition.

```
> f;
```

$$f \quad (3.14)$$

In order to see what the definition is, you can use `eval`.

```
> eval(f);
```

$$x \rightarrow \sin(x) + \frac{x}{\pi} \quad (3.15)$$

It's often more convenient to use functions rather than expressions, especially when you might want to look at the function at many different values of the variables.

## Functions from expressions

```
> restart;
```

Suppose you have an expression such as

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

$$B := \sin(x) - \frac{x}{\pi} \quad (4.1)$$

and you would like to make it into a function. If it's a complicated expression, you probably don't want to type it in again. It's tempting to try

```
> f := x -> B;
```

$$f := x \rightarrow B \quad (4.2)$$

But that doesn't work.

```
> f(3);
```

$$\sin(x) - \frac{x}{\pi} \quad (4.3)$$

Why doesn't it work? Because there are really two different variables called `x` here: the one in `B`, which is an ordinary variable (called a **global** variable in Maple terminology), that you can assign values to, while the one in the definition of `f` is a dummy variable or "formal parameter" that would be unaffected by assigning values to the global `x`. When you enter the command `f(3)`; Maple substitutes 3 for the formal parameter `x` in the definition of `f`, but doesn't do anything to the global variable `x`.

The right way to make an expression into a function, is using the `unapply` command. Thus:

```
> f := unapply(B, x);
```

$$f := x \rightarrow \sin(x) - \frac{x}{\pi} \quad (4.4)$$

```
> f(3);
```

$$\sin(3) - \frac{3}{\pi} \quad (4.5)$$

You can also use `unapply` to make a function of several variables.

```
> A := sin(x+y) - exp(y+z);
```

$$A := \sin(x + y) - e^{y+z} \quad (4.6)$$

```
> g := unapply(A, x, y, z);
```

$$g := (x, y, z) \rightarrow \sin(x + y) - e^{y+z} \quad (4.7)$$

Caution: you want to define `f`, not `f(x)`. Here is a common mistake:

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

$$h(x) := \sin(x) - \frac{x}{\pi} \quad (4.8)$$

It looks OK, Maple doesn't complain, and you can even enter

```
> h(x);
```

$$\sin(x) - \frac{x}{\pi} \quad (4.9)$$

But what this definition did was literally to define `h(x)`, not `h(anything else)`.

```
> h(y);
```

$$h(y) \quad (4.10)$$

```
> h(3);
```

$$h(3) \quad (4.11)$$

Maple doesn't know what `h(y)` is supposed to be, it just knows `h(x)`. What you should have done was

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

$$h := x \rightarrow \sin(x) - \frac{x}{\pi} \quad (4.12)$$

```
> h(y);
```

$$\sin(y) - \frac{y}{\pi} \quad (4.13)$$

## Derivatives

There are two methods of differentiation in Maple: **D**, which applies to functions, and **diff**, which applies to expressions. Suppose we have an expression, such as `B`, containing the variable `x`:

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

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

$$f := x \rightarrow \sin(x) - \frac{x}{\pi} \quad (5.1)$$

We can use **diff** to take its derivative with respect to `x`.

```
> diff(B,x);
```

$$\cos(x) - \frac{1}{\pi} \quad (5.2)$$

If the expression contains more than one variable, what diff gives you is the partial derivative with respect to this particular variable.

```
> A:= sin(x+y) - exp(y+z); g:= unapply(A,x,y,z);
```

$$\begin{aligned} A &:= \sin(x+y) - e^{y+z} \\ g &:= (x,y,z) \rightarrow \sin(x+y) - e^{y+z} \end{aligned} \quad (5.3)$$

```
> diff(A, x);
```

$$\cos(x+y) \quad (5.4)$$

```
> diff(A, y);
```

$$\cos(x+y) - e^{y+z} \quad (5.5)$$

On the other hand, D takes a function of one variable and gives you another function of one variable, its derivative:

```
> D(f);
```

$$x \rightarrow \cos(x) - \frac{1}{\pi} \quad (5.6)$$

You can treat D(f) like any other function. For example, you can evaluate it at a point.

```
> D(f)(Pi);
```

$$-1 - \frac{1}{\pi} \quad (5.7)$$

Here I'm going to look at both types of derivative. On the left I'm going to take the function D(f) and evaluate it at x. On the right I'll take the derivative of the expression f(x) with respect to x. The results should be the same.

```
> D(f)(x) = diff(f(x),x);
```

$$\cos(x) - \frac{1}{\pi} = \cos(x) - \frac{1}{\pi} \quad (5.8)$$

To evaluate a **diff** expression at a certain value of x, you need to use **eval**:

```
> D(f)(Pi) = eval(diff(f(x),x), x=Pi);
```

$$-1 - \frac{1}{\pi} = -1 - \frac{1}{\pi} \quad (5.9)$$

D on a function of more than one variable gives an error.

```
> D(g);
```

Error, (in D/procedure) function must be unary

Instead, use D[i] for the partial derivative with respect to the i'th variable.

```
> D[1](g)(x,y,z) = diff(g(x,y,z), x);
```

$$\cos(x+y) = \cos(x+y) \quad (5.10)$$

```
> D[2](g)(x,y,z) = diff(g(x,y,z), y);
```

$$\cos(x+y) - e^{y+z} = \cos(x+y) - e^{y+z} \quad (5.11)$$

```
> D[4](g)(x,y,z);
```

Error, (in D/procedure) index out of range: function takes only 3 arguments

## Newton's method

When **fsolve** finds an approximate solution to an equation, it's using a numerical method. The prototype of these methods for solving equations is Newton's method. The methods **fsolve** uses are sometimes slightly more sophisticated, but still based on Newton's method, so it's worthwhile trying to understand 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  $x_1, x_2, \dots$  that converges very rapidly to a solution (when things are working well). As we'll see, it doesn't always work well, and we'll investigate what can happen. The formula for Newton's method is

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

It's convenient to define a function I'll call **newt** to do this, so that this is

$$x_{n+1} = \text{newt}(x_n).$$

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

$$\text{newt} := x \rightarrow \text{evalf}\left(x - \frac{f(x)}{D(f)(x)}\right) \quad (6.1)$$

There are a couple of things to notice here.

- I used **evalf**, because I will want decimal numbers here. I don't want complicated exact expressions.

```
> 3 - f(3)/D(f)(3);
```

$$3 - \frac{\sin(3) - \frac{3}{\pi}}{\cos(3) - \frac{1}{\pi}} \quad (6.2)$$

```
> % - f(%) / D(f)(%);
```

$$3 - \frac{\sin(3) - \frac{3}{\pi}}{\cos(3) - \frac{1}{\pi}} - \frac{\sin\left(3 - \frac{\sin(3) - \frac{3}{\pi}}{\cos(3) - \frac{1}{\pi}}\right) - \frac{3 - \frac{\sin(3) - \frac{3}{\pi}}{\cos(3) - \frac{1}{\pi}}}{\pi}}{\cos\left(3 - \frac{\sin(3) - \frac{3}{\pi}}{\cos(3) - \frac{1}{\pi}}\right) - \frac{1}{\pi}} \quad (6.3)$$

- The definition of **newt** uses **f** itself, not the definition of **f**, so in the definition, even if **f** has already been defined as, say,

$x \mapsto \sin(x) - \frac{x}{\pi}$ , you see  $f(x)$  rather than  $\sin(x) - \frac{x}{\pi}$ . When you use

**newt**, it will use whatever is the current definition of  $f$ . If you changed  $f$ , **newt** would still be good for doing Newton's method with the new version of  $f$ .

Let's start with a typical initial guess, and see where Newton's method takes us.

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

$$f := x \rightarrow \sin(x) - \frac{x}{\pi} \quad (6.4)$$

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

$$x_0 := 3 \quad (6.5)$$

This is actually an entry in what Maple calls a "table". If you've done some computer programming, you may be pleasantly surprised to see that you don't need to tell Maple that you're making a table, or how big it will be: the table is created automatically whenever you first use " $x[\dots]$ ", and will hold whatever entries you put in it. The index inside the brackets is shown as a subscript in output.

```
> x[1] := newt(x[0]);
```

$$x_1 := 2.377965170 \quad (6.6)$$

```
> x[2] := newt(x[1]);
```

$$x_2 := 2.315135129 \quad (6.7)$$

This could take a while if I had to type this in each time I want to get another  $x_n$ . Fortunately, we can automate the process using a "for loop".

```
> for count from 2 to 6 do  
  x[count + 1] := newt(x[count])  
end do;
```

$$x_3 := 2.313734857$$

$$x_4 := 2.313734132$$

$$x_5 := 2.313734132$$

$$x_6 := 2.313734132$$

$$x_7 := 2.313734132$$

(6.8)

## ▼ Maple objects introduced in this lesson

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