An introduction to Mathematica

Levien van Zon, University of Utrecht

Systems Biology 2013, Mathemathics for Biologists

The Basics

In Mathematica you can enter *expressions*, which you can then *evaluate* by putting your cursor after the expression and pressing Shift + Enter. An expression is usually a calculation, or a *function* that tells Mathematica to do something. Evaluation simply means that Mathematica will do the calculation, or run the function.

For instance, the calculation below is an expression. You can get the result by putting your cursor behind it, and pressing Shift + Enter.
Try this.

```
15^2 + (10 * 56 + 18 - 100) / 29
```

If you want to change something in the calculation, just put your cursor there, change it, and press Shift + Enter again.

As you can see, Mathematica did not give you a number as the result of the above calculation. Instead, it gave you a fraction. To get a number, try the following :

```
N[%]
```

So what does this do? The function N[] will convert anything that is between the brackets to a number, if possible. The %-sign is used to repeat the outcome of the previous expression.

You can also store the outcome of an expression, by *assigning* it to a named *variable*. The following expression stores the result of our previous calculation in a variable named "number":

```
number = %
```

You can now use the word "number" instead of the number itself.
Note that variables need to start with a lowercase-letter, they may not start with a number and it is unwise to start with a capital letter (because those names may already be in use for a function or built-in variable).

Of course you can also assign more complicated expressions to a variable :

```
complicated = x^8 + x^5 / (x - a) + b
```

If you now want to know what the outcome of the above formula is for $x = 84$, $a = 18$, $b = -36$, you can temporarily "fill in" (substitute) these values into the formula, using the so-called *ReplaceAll* (or slash-dot) *operator* "/.":

```
complicated /. {x → 84, a → 18, b → -36}
```

You can just type -> to get an arrow. The arrow is called a *transformation operator*. It temporarily transforms every x into 84, every a into 18, etc.

Mathematica is a computer language for symbolic calculations, which means that it can work with numbers as well as with parameters. Very few other programs can do this, and it is very useful. To see it in action, just leave out the values for a and b in the above example:

```
complicated /. x → 84
```

Mathematica also knows about constants such as π, e and even abstract concepts such as i and ∞. Note that names of constants and functions always start with a capital letter:

```
Pi
```

```
N[Pi]
```

```
N[E]
```

```
I^2
```

```
Infinity * Infinity
```

The document you're using right now is called a Mathematica *notebook*. The text, expressions, results, pictures and other things in the notebook are organised into *cells*, which in turn can be divided into subcells. On the right side of this window you can see small blue/grey bars, these indicate where cells and subcells start and end. If you want to insert a new cell somewhere, just put the mouse below a cell, or in between two cells, and click. A horizontal line should appear. Anything you type now will go into a new cell. You can delete a cell by clicking on the blue/grey bar on the right, and pressing Delete.

Variable that you assign will keep their value until you either clear them, or exit from Ma
This can of course lead to problems, if you "recycle" names of variables and parameters in
To avoid this, either choose new names, or clear the variables you use.

A variable or parameter that is defined is shown in black, a variable/parameter that is not
To clear the variable *x*, you can do: "x=."
To list all variables, you can use: Names["Global`*"]

```
Names["Global`*"]
```

To clear all variables, use: Clear["Global`*"]

```
Clear["Global`*"]
```

Useful Operations

The way to write calculations in Mathematica is similar to what you're probably already used to from programs like Excel. For division you use a forward slash /, for exponentiation you use a caret ^, for multiplication you can either use an asterisk *, or just put a space (the latter is rather unique to Mathematica, a space is not accepted as multiplication by most other programs). For complicated expressions you may need a lot of open- and closing-brackets, (and). To make things easier, you can also use the Basic Math Assistant, which you can find in the Palettes menu.

Note: Take care when multiplying non-numerical things: a*x or a x is a multiplication. But if you forget a space, you may get ax, which Mathematica will ***not*** interpret as a multiplication. Instead, Mathematica will see this as the name of an unspecified variable called "ax"! If you want to be on the safe side, always use a * for multiplication (although we don't generally do it in the examples).

Functions start with a capital letter, and take one or more arguments separated by commas. For instance, the limit of $\frac{a\,x+q}{c^2+x^2}$ for *x* going to infinity (see **exercise** 2a, chapter 1, page 14) can be calculated by the function Limit[]:

```
Limit[(a x + q) / (c^2 + x^2), x → Infinity]
```

-> For the expression in chapter 1, page 14, exercise 2 b, calculate the limits when N goes infinity, and to zero. You can use the additional argument Direction -> 1 to find the limit from the left-side, and Direction -> -1 to find the limit from the right-side.

To take the derivative of a function, you can use the function D[]. As a second argument, you need to specify the variable you want to take the derivative to:

```
D[x^2 + 5 x, x]
```

-> Use the D[] function to make exercise 5 of chapter 1 (page 14).

If you want to find the minimum and maximum of a function, you can of course calculate the derivative, find where it is zero (we will see how to do that below), and then substitute the results back into the original function to find maxima and minima. However, if you're looking for just a single maximum or minimum, it may be faster to use the functions Maximize[] and Minimize[]:

```
Maximize[-x^2 + b x, x]
```

If you need to pass a *list* of things as argument to a function, you need to put the things between curly brackets, { and }, separated by commas. For instance, the function Plot[] requires two arguments. The first argument is the function to plot, or a list of functions to plot. The second argument is a list that contains the variable to put on the x-axis, and the minimum and maximum value to show on the axis:

```
Plot[x / (2 + x), {x, -2, 25}]
```

```
Plot[{x, x^2, x^3}, {x, -2, 2}]
```

-> Plot the parabola of the previous example for *b* = 2. Check whether the maximum value you would calculate for that value of *b* is correct.

-> Plot the function in exercise 4f of chapter 3 (page 45). (Here you need to plot dx/dt as function of x.)

-> Plot the function in exercise 5 of the same chapter, for s = 0. Can you find the local minimum and maximum (close to the y-axis) using Minimize[] and Maximize[]?

```
Plot[3 x^2 / (2 + x^2) - x, {x, -0.3, 3}]
```

You may have found that Minimize[] and Maximize[] sometimes give nonsense-values, if a functions goes to -∞ or ∞. In that case, you may need to specify a range for your variable. For instance, to find the local maximum of the third-degree function $-x^3+2x$, between $x \geq 0$ and $x \leq 2$, we can add conditions for x as follows:

```
Maximize[{-x^3 + 2 x, x >= 0, x <= 2}, x]
```

-> Now try again to find the local maximum and minimum of the function in the previous exercise.

Convieniently, the way to specify a list of things is also the way to specify a vector (as a vector is basically just a list of things). Just write the elements of the vector between curly braces, and separate them by commas: {1, 2, 3}. Similarly, a matrix is written as a list of vectors, one for each row: {{a, b},{c, d}}.
To see vectors and matrices in the more familiar matrix-notation, you can use the function MatrixForm[].
Some examples of working with vectors and matrices:

```
{a, b} + {c, d}
```

```
{a, b} . {c, d}
```

```
somematrix = {{a, b}, {c, d}}
```

```
MatrixForm[somematrix]

somematrix.{p, q}

MatrixForm[%]

Eigenvalues[{{1, 2}, {-2, -1}}]

Eigenvalues[somematrix]

Eigenvectors[somematrix]

Det[somematrix]
```

-> The function Tr[] computes the so-called *trace* of a matrix. Can you find out with Mathematica how the trace of a 2x2 matrix is defined?

Rewriting Expressions

A very useful application of Mathematica is the simplification of expressions. For instance, we can calculate the partial derivative ∂z/∂N of **exercise** 1h. (chapter 6, page 81) as follows:

```
dzdN = D[(b^2 N^2 T) / (1 + c N + b T N^2), N]
```

We can then simplify this expression using the function Simplify[]:

```
Simplify[dzdN]
```

-> Calculate and simplify the partial derivative to T (see chapter 6, **exercise** 1h, page 81).

Especially if you have long expressions that contain lots of braces or divisions, you can use Mathematica to simplify or rewrite these in different ways:

Simplify[*expression*] simplifies an expression, if possible.
FullSimplify[] will try to simplify it even further, if possible, using a bigger arsenal of mathematical trickery.
Expand[*expression*] will multiply out everything and get rid of braces.
Factor[*expression*] will do the opposite, factorizing the equation where possible.
Collect[*expression*, *variable*] will attempt to separate the various powers of the specified variable.
If an expression contains fractions, Together[] will try to write it as a single fraction.
Apart[] will do the oposite, splitting everything into separate fractions where possible.
Finally, Cancel[] will attempt to cancel out common factors in the numerator and the denominator.

Below are some examples. First, some complex trigonometry:

```
FullSimplify[Cos[x] + I * Sin[x]]
```

Then, using the equation from chapter 1, **exercise** 6b (page 14):

```
ch1ex6b = (r / a) (h + R) (1 - R / K)

Expand[ch1ex6b]

Collect[ch1ex6b, R]

Together[ch1ex6b]

Apart[ch1ex6b]
```

Solving Equations

Another very useful applications of Mathematica is that it can solve equations, even very complicated ones with parameters.

Equations are written in the following general form :
left-side == right-side
The double equal-signs '==' mean "is equal to", whereas a single '=' means "assign the expression on the right to the variable on the left" (you can interpret '=' and '->' as "becomes").
Be careful not to confuse those two, or you may get strange results!

In addition to ==, you also have > and < for "greater than" and "smaller than", >= for "is greater than or equal to" and <= for "is smaller than or equal to".

To solve a single equation for a given variable, you can use the function Solve[*equation*, *variable*]. For example:

```
Solve[a x^2 + b x + c == 0, x]
```

Or we can do **exercise** 3d from chapter 1 (page 14) :

```
Solve[(b - d (1 + N / K)) N == 0, N]
```

Or something a bit more complicated:

```
Solve[x^4 - 15 x^2 + 30 x - 86 == 192, x]
```

-> Can you get a simpler, numeric solution?

-> Now solve the equation in chapter 1, exercise 3e, and find the intercepts with both axes in exercise 6a (both page 14).

If you want to solve systems of equations with multiple variables, just put the equations and the variables in a list. As an example, we will do **exercise** 4d (chapter 1, page 14):

```
system4d = {4 x - x y - x^2 == 0, 9 y - 3 x y - y^2 == 0}
Solve[system4d, {x, y}]
```

If you really want to find *all* possible solutions, including all possible combinations of parameters and variables, you can use Reduce[] instead of Solve[].
In the result you may encounter the symbols || (which means "or"), and && (which means "and").

-> Now solve the equation in exercise 4e (chapter 1, page 14) yourself. Try with both Solve[] and Reduce[].

Solving differential equations

Mathematica can also solve differential equations, at least up to a point, using the function DSolve[].
For example, we can easily find the general solution for exponential growth:

```
expgrowth = p'[t] == r p[t]
DSolve[expgrowth, p[t], t]
```

To find a specific solution, we need to specify the initial value of our variable:

```
solexp = DSolve[{expgrowth, p[0] == 30}, p[t], t]
```

Remember that /. can be used to fill in (substitute) the values (or even functions) on the right-hand side into the left-hand side.
For r = 0.5, we can use this to plot our solution as follows:

```
Plot[p[t] /. solexp /. r → 0.5, {t, 0, 10}]
```

A more complicated example, logistic growth:

```
loggrowth = q'[t] == r (1 - q[t] / K) q[t]
DSolve[loggrowth, q[t], t]
```

-> Can you find the specific solution for an initial population of q = 5?

-> What happens to the general solution if you make the death-rate density independent? Can you still find the specific solution?

You are familiar with the following model for the production and decay of bloodcells:

```
production = y'[t] == c - d y[t]
solpr = DSolve[{production, y[t] == 10}, y[t], t]
```

Above we have calculated the specific solution for y[0] == 10. We could now try to use the limit t→∞ to find the equilibrium:

```
Limit[y[t] /. solpr, t → Infinity]
```

You will notice that this does not give us a useful anwer. The reason is that Mathematica needs to have more information on the parameters, before it can calculate the limit. If *c* and *d* are negative, the result will be different from when they are positive. We can use the function Assuming[*conditions*, *expression*] to specify conditions for the parameters:

```
Assuming[{c > 0, d > 0}, Limit[y[t] /. solpr, t → Infinity]]
```

In addition to differential equations with one variable, we can also solve systems of differential equations:

```
system = {x'[t] == a x[t] + b y[t], y'[t] == c x[t] + d y[t]}
solsys = DSolve[system, {x[t], y[t]}, t]
```

-> The result may surprise you. Can you make some sense of it?

-> Find the general solutions for the systems in chapter 5, exercises 1a and 1e (page 73).
(Hint: You can do this really quickly if you use /. and the variable containing the general solution...)

```
solsys /. {a → 0, b → -2, c → 1, d → -2}
```

-> Below you see the system from chapter 5, exercise 2a (page 74). Can you find a solution for this one? Why?

```
sys2a = {x'[t] == x[t] (1 - x[t] - y[t]), y'[t] == y[t] (1 - 2 x[t])}
```

```
DSolve[sys2a, {x[t], y[t]}, t]
```

The Jacobian

Our last set of examples features the Jacobian Matrix (see chapter 6, section 6.3, page 80).
Consider a general system of two differential equations, in which $\frac{dx}{dt}=f(x,y)$ and $\frac{dy}{dt}=g(x,y)$.
If the system is linear, we can write $f(x,y)$ and $g(x,y)$ as follows:

```
f = a x + b y
g = c x + d y
```

We can then calculate a Jacobian matrix by first taking the derivative of function *f* to *x*, then to *y*, and then taking the derivative of function *g* to *x* and *y*, and finally putting it all in a 2x2 matrix:

```
jacobian = {{D[f, x], D[f, y]}, {D[g, x], D[g, y]}}
```

This Jacobian is of course the matrix that you would expect for a linear system.
We can again display it in a more familiar 2x2 matrix form using MatrixForm[]:

```
MatrixForm[jacobian]
```

There is actually a faster "trick" to make a Jacobian matrix from the functions *f* and *g*:

```
jacobian = D[{f, g}, {{x, y}}]
```

We can now quickly determine the stability of the system in chapter 6, **exercise** 5a (page 81):

```
Eigenvalues[jacobian /. {a → -2, b → 1, c → 1, d → -2}]
```

```
-> What is the stability of the equilibrium in this system?
```

Of course in this case it's a linear system, so we could have constructed the matrix directly.
But if we specify different functions for f and g, we can also find the Jacobian for a non-linear system.

But first we can do another useful trick. Instead of having to repeat jacobian = D[{f,g},{{x,y}}] every time we change the functions *f* and *g*, we can store the derivative in an *unevaluated form*. We can do this by useing ':=' instead of '='. Now, every time we use the variable "jacobian", Mathematica will look up the current values of *f* and *g*, and use those for the derivative.

```
jacobian := D[{f, g}, {{x, y}}]
```

We can now easily find the jacobian and its eigenvalues for the system in **exercise** 3a (page

```
f = -4 y
g = 4 x - x^2 - 0.5 y
MatrixForm[jacobian]
eigen = Eigenvalues[jacobian]
```

This gives us general expressions for the Jacobian and its eigenvalues, which are valid for any value of *x* and *y*.
To get the eigenvalues in an equilibrium point, we first need the *x* and *y* values of the equilibria.
We can get these by solving *x* and *y* for *f* == 0 AND (&&) *g* == 0:

```
equilibria = Solve[f == 0 && g == 0]
```

You can now just substitute the solutions above into the Jacobian, and you will get a Jacobian matrix for each equilibrium point:

```
MatrixForm[jacobian] /. equilibria
```

And of course you can do the same for the eigenvalues:

eigen /. equilibria

-> What types of equilibria do we have here?

-> Now try to find the equilibria of exercise 3d (page 81), and determine their type.

Extra Exercises (for interested students)

1a. Solve the equation 40-10 x+x^2 = 0
1b. Multiply the solutions, and simplify.
1c. Explain the result of 1b.

2a. Given two numbers, *a* = 1 and *b* = 0.5 + I, multiply *a* and *b*, and store the result in *c*.
2b. You can draw *c* on the complex plane, with:
Graphics[Arrow[{{0,0},{Re[a], Im[a]}}], Axes→True]
On paper, sketch a, b and c on the complex plane (in one figure).
2c. Multiply *c* again with *b*, and store the result in *c*. A fast way to do this is: c *= b
Sketch the result again on the complex plane.
2d. Repeat this a number of times. Describe what happens.

3. On page 42 of the Theoretical Biology reader, you can find a simple model of virus infections and the immune response (system 6.1).
Here, T is the number of uninfected cells, I is the number of infected cells, V is the number of virus particles, and E is the immune response.
3a. Enter the four equations of the model into four variables. It's best to avoid using capital letters, and parameter-names that you have used before.
3b. Find the equilibria of this model. Compare with 6.2 and 6.3