Thomas Gamsjäger

Separation of Variables

MAYTENSOR

Introduction

The fact that you are here leads me to the assumption that you already know what a differential equation is. In any case, here is the prime example:

$$\frac{dN}{dt} = kN \tag{1}$$

This differential equation describes a dynamical system (evolving over time t) where the rate of change of quantity N (denoted by the term on the left side, dN/dt) is proportional to the amount of N currently present (on the right side) with the parameter k being a proportionality constant.

It is the classical model of exponential growth. But before we can actually see this behaviour (e. g. in a diagram with quantity N plotted over time t) we need to solve the differential equation to get the desired function $N(t)$ that lets us calculate the values of N for any point of time t.[1]

Here is how an example looks like. We start with, say, 100 bacteria, and we happen to know that they happily procreate with a proportionality constant of $k = 0.2$.

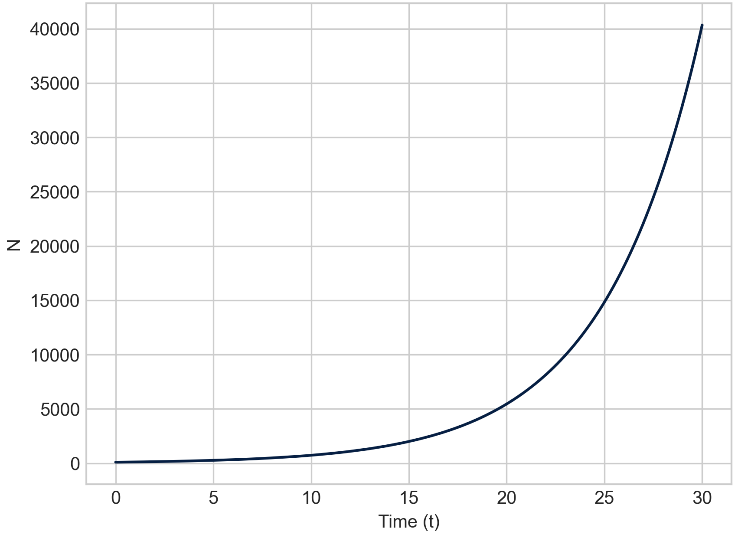

Figure 1: The typical looks of exponential growth. After only 30 time units our merry bunch of 100 bacteria have multiplied to more than 40,000 (on condition that they don't run out of supplies and space; such limiting factors can be modelled as logistic growth, but this is a matter for another opportunity).

[1] Recall that the solution of a differential equation is a function (as just stated), the solution of a function is a number, and the plot of a function is just a series of numbers.

Therefore, we have to solve the differential equation. There are two options. We can either aim for an analytical solution, which is mathematically exact, or content ourselves with a numerical approximation.

Separation of variables

Whereas the numerical approach pretty much always works, deriving an analytical solution of a differential equation necessitates a specific procedure, the separation of variables (the reason why we are here). This only works for comparatively simple differential equations like the one above.

Buckle up.

We start with the differential equation we already know:

$$\frac{dN}{dt} = kN \tag{2}$$

The first step is already crucial. Do you remember that they told you that what looks like a fraction (dN/dt) is not a fraction at all, and that you can never treat it as such.[2] Well, it depends. In fact, we will do just that: treat it as a fraction by multiplying the equation by dt. Let's see what happens.

$$dN = kNdt \tag{3}$$

Now we divide by N.

$$\frac{dN}{N} = kdt \tag{4}$$

With this move we have already made a major achievement: we have separated the variables. All the Ns are on the left, and the rest is on the right. Please note that dN is a symbol in its own right and not the product of d and N (just as dt). We <u>cannot</u> cancel the Ns in dN/N.

At its core, solving a differential equation means integration. Before we do that, I want to rearrange the left side a bit to make things clearer:

$$\frac{1}{N}dN = kdt \tag{5}$$

[2] After all, $\frac{dN}{dt}$ is just a type of notation, the one by Leibniz, whereas Lagrange preferred N' (prime notation) and Newton $\dot{N}$ (dot notation). (Note that N is just an arbitrary variable name.)

Now the (dreaded) integration:

$$\int \frac{1}{N} dN = \int k dt \tag{6}$$

Here the nature of the symbols dN and dt become clearer still: they indicate that we want to integrate with respect to N and t, respectively.

Integrating the right side is straightforward. Don't forget the integration constant (here denoted by C_2)!

$$\int \frac{1}{N} dN = kt + C_2 \tag{7}$$

But the left side with its $1/N$ looks somewhat unfamiliar in terms of integration. Perhaps we can deduce how to integrate it from how the right side worked... Well, don't try it. What we need is a very specific integration rule that is able to handle $1/N$.[3] Its derivation is something better left for the hardcore mathematicians because $1/N$ integrated is the natural logarithm of N (plus integration constant), $ln(N) + C_1$:

$$ln(N) + C_1 = kt + C_2 \tag{8}$$

We are moving briskly along. C_1 and C_2 are simply numbers. We can roll them into a single number C_3.

$$ln(N) = kt + C_3 \tag{9}$$

To extract N from $ln(N)$ we employ the antilogarithm:

$$N = e^{kt} e^{C_3} \tag{10}$$

Wait. Not so fast. The antilog is nice and well, but, one moment, we have a sum ($kt + C_3$), and suddenly there is a product ($e^{kt} e^{C_3}$)?

This is the practical application of logarithm properties. Let's do it stepwise, starting again with:

$$ln(N) = kt + C_3 \tag{11}$$

We can rewrite this eqivalently as:

$$ln(N) = ln(e^{kt}) + ln(e^{C_3}) \tag{12}$$

[3] A handy table of integrals is generously provided by the Department of Physics at the University of Maryland: https://www.physics.umd.edu/hep/drew/IntegralTable.pdf

By way of the product rule which says that the logarithm of a product is the sum of the logarithms of its factors we get:

$$ln(N) = ln(e^{kt}e^{C_3}) \tag{13}$$

And with the antilog we get:

$$N = e^{kt}e^{C_3} \tag{14}$$

(For additional support as to the veracity of this logarithm wrangling, I have put together a Python script using the symbolic mathematics library SymPy and a MATLAB script using the Symbolic Math Toolbox (to be found in Appendix 1).)

On close inspection, we recognise that e is only a number[4] (as is C_3), so why not do a similar number roll as above again by defining $e^{C_3} = C$, and put the new C in front of e^{kt}:

$$N = Ce^{kt} \tag{15}$$

Now that we have a proper function that depends on t, we can make this fact more explicit by sticking (t) onto N:

$$N(t) = Ce^{kt} \tag{16}$$

(Analogous to the logarithm procedures above, Appendix 2 reveals how this analytical solution can be derived symbolically using Python/SymPy and MATLAB, respectively.)

In any case, we are almost there. We just have to determine C. For this we need the initial condition, the quantity of N we start with (100 bacteria in the example above). We denote it by N_0. With this piece of information we know a whole lot, specifically that the amount of N present at $t = 0$ (i. e. $N(t = 0)$) must be N_0. Let's plug this into the equation above:

$$N(t = 0) = N_0 = Ce^{k \cdot 0} \tag{17}$$

$k \cdot 0 = 0$, and e^0 is 1. Therefore:

$$N_0 = C \tag{18}$$

Or flipped around:

$$C = N_0 \tag{19}$$

[4] $e = 2.718281828...$

By plugging this into the equation above, we finally have our result, the analytical solution of the differential equation $dN/dt = kN$, the exponential function:

$$N(t) = N_0 e^{kt} \tag{20}$$

Admittedly, many steps along the way felt like mathematical trickery. But the beauty of it is that it works.

Let's whip up a few lines of Python code to see how this exponential function looks like:

```python
# Bring in the libraries:
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-v0_8-whitegrid')

# Set the necessary parameters:
N0 = 100
k = 0.2
t_start = 0
t_end = 30
t = np.linspace(t_start, t_end, 1000)

# The exponential function:
N = N0*np.exp(k*t)

# Plotting:
plt.plot(t, N)
plt.xlabel('Time (t)')
plt.ylabel('N')
plt.show()
```

This is the plot:

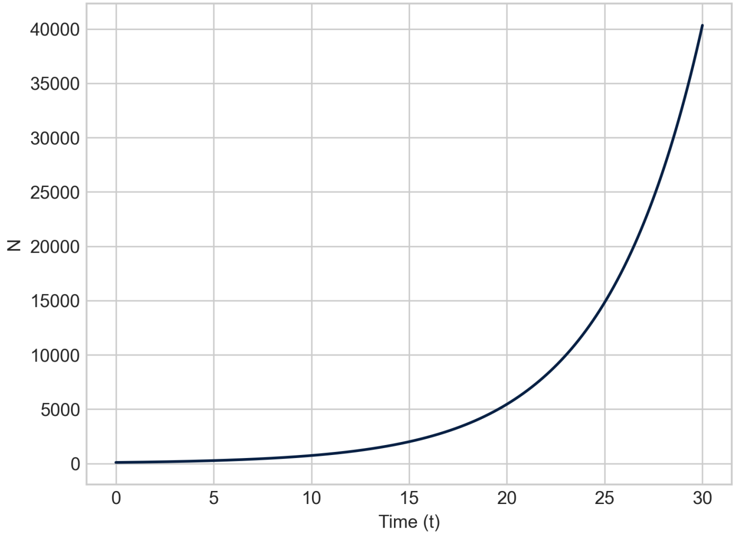

Figure 2: The plot of the analytically derived exponential function $N(t) = N_0 e^{kt}$ as the solution of the differential equation $dN/dt = kN$. Looks exactly like the plot in Figure 1.

Or if you are tired of having to import all the Python libraries again and again, here is the same script implemented in MATLAB. I find it much neater and tidier:

```
% Set the necessary parameters:
N0 = 100;
k = 0.2;
t_start = 0;
t_end = 30;
t = linspace(t_start, t_end, 1000);

% The exponential function:
N = N0*exp(k*t);

% Plotting:
plot(t, N)
grid on
xlabel('Time (t)')
ylabel('N')
```

Remarkable, how similar Python and MATLAB are (at least in this case here).

The MATLAB plot:

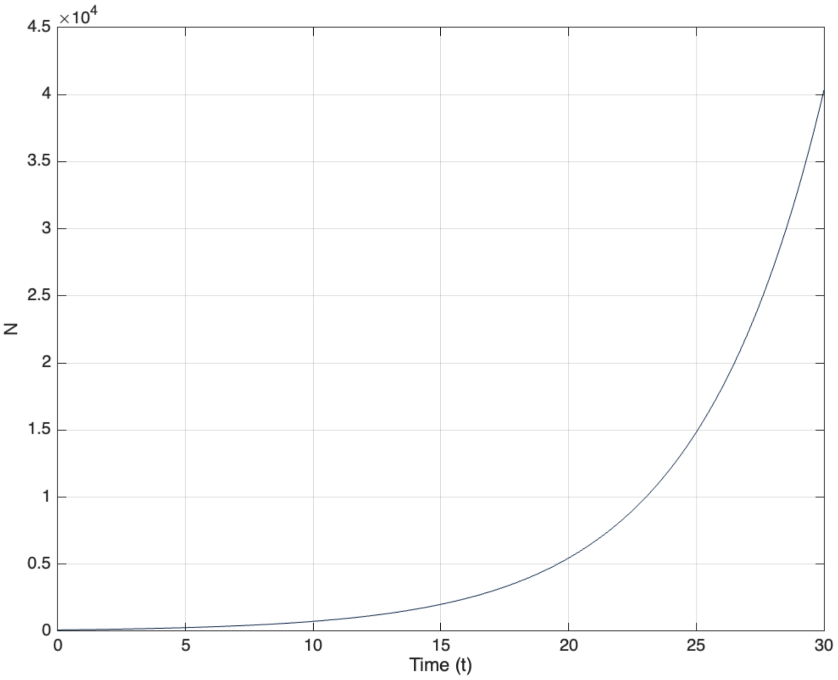

Figure 3: Even the plots look more sophisticated in MATLAB, but not necessarily nicer.

In conclusion

Above we have seen how the method of separating the variables in order to solve a simple differential equation works. The result is a mathematically exact function as the solution of the differential equation (aka analytical solution).

For more involved differential equations finding such an analytical solution is either tedious or outright impossible. In such cases, a solution can still be obtained by way of numerical methods where we let a computer crunch the numbers. How this approach works is best discussed on a separate occasion...

Appendix 1

This is a quick Python script using the symbolic mathematics library SymPy to further illustrate the logarithm procedures.

Please note specifically that `log()` signifies the natural logarithm (and not the logarithm base 10). And the number e is expressed as `exp()`.

```python
# Import the library:
from sympy import *

# Define the symbolic variables:
k, t, C3 = symbols('k t C3')
```

Let's see if we can define $ln(e^{kt}) + ln(e^{C_3})$ correctly in SymPy lingo:

```python
log(exp(k*t)) + log(exp(C3))
```

$> log(e^{kt}) + log(e^{C3})$

Looks good. (Remember: in SymPy, $log()$ means the natural logarithm.

Now we combine the sum into the product according to the product rule:

```python
p = logcombine(log(exp(k*t)) + log(exp(C3)), force=True)
p
```

$> log(e^{kt}e^{C3})$

And now the antilog:

```python
exp(p)
```

$> e^{kt}e^{C3}$

Using MATLAB (and its Symbolic Math Toolbox):

```matlab
syms k t C3
log(exp(k*t)) + log(exp(C3))
p = combine(log(exp(k*t)) + log(exp(C3)), 'log',
        'IgnoreAnalyticConstraints', true)
exp(p)
```

The result is the same as above.

Appendix 2

This is the procedure to derive the analytical solution of the differential equation

$$\frac{dN}{dt} = kN \tag{21}$$

using Python in conjunction with the SymPy library:

```
# Import the library:
from sympy import *

# Define the symbolic variables:
k, t = symbols('k t')
```

For the system variable N we need to be more specific. We have to define it explicitely as a function.

```
N = symbols('N', cls=Function)
N
```

$> N$

Now the part that looks a bit complicated: setting up the differential equation (with DE being just an arbitrary variable name):

```
DE = Eq(N(t).diff(t), k * N(t))
DE
```

$> \frac{d}{dt}N(t) = kN(t)$

Apparently, Python has correctly interpreted our intentions.

Ready to solve the differential equation (with remarkably straightforward syntax compared to the lenghty setup above):

```
sol = dsolve(DE)
sol
```

$> N(t) = C_1 e^{kt}$

Done.

Now the same with MATLAB and its Symbolic Math Toolbox.

```
syms N(t) k
DE = diff(N, t) == k*N
```

$> \frac{\partial}{\partial t} N(t) = k N(t)$

For no particular reason, MATLAB uses the symbol of the partial derivative (∂) here instead of the ordinary derivative (d).

Now we can solve the equation:

```
sol = dsolve(DE)
```

$> sol = C_1 e^{kt}$

Done.

Further reading

Barnes B, Fulford GR. Mathematical modelling with case studies. CRC Press 2015

Gustafson GB. Differential equations and linear algebra, Volume I. Gustafson 2022

Kline M. Calculus. Dover 1998

Back matter

Thomas Gamsjäger

Separation of variables

MAYTENSOR

Hallein, Austria

First edition, January 2025

Published on www.maytensor.org

Cite as: Gamsjäger T. Separation of variables. Maytensor 2025

This article is part of a series in which the aim is to provide explanation, insights and perhaps (or hopefully) some fresh perspectives on selected topics in science and engineering. As standard textbook knowledge is the basis, typically no specific references are given. But selected sources are listed in the 'Further reading' section.

Cover image: Colonies of the bacterium Legionella on a Petri dish. (Image credit: James Gathany. Centers for Disease Control and Prevention 2005. Public domain. https://phil.cdc.gov/Details.aspx?pid=7925, retrieved 6 January 2025)

Typeset with LaTeX.