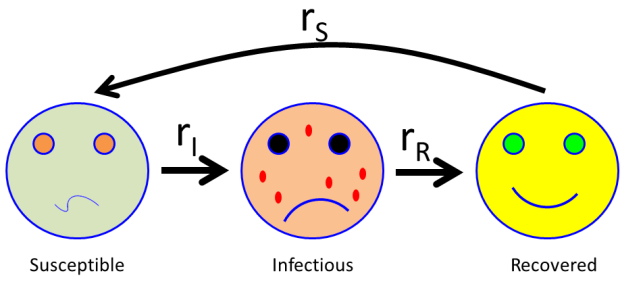
Tutorial Computational Modelling with ordinary differential equations

Modelling outbreak of an infection with SIR

ARCAID Bioinformatics Course

Version 9. March 2022



Antoine van Kampen
Rodrigo Garcia Valiente
Bioinformatics Laboratory
Amsterdam University Medical Centers
AMC

a.h.vankampen@amsterdamumc.nl
020-5667096
www.bioinformaticslaboratory.eu

Introduction

In this tutorial you will model the outbreak of an infection in a population using the SIR model. In **part I** of this tutorial we will introduce you to the 'deSolve' package, which you will use to solve differential equations. In **Part II** you will use the 'deSolve' and 'FME' package to implement and calculate the SIR infection model. In Appendix you find a nano-review about ordinary differential equations (ODEs). All documents can be found on our website.

Part I. Getting started with R and deSolve

We will use R Studio and the deSolve package for the modelling. Therefore, we will assume that you are familiar with R and R studio. See the course website for installation instructions and documentation for R/Rstudio. If you are not familiar with R then just proceed and try to use the provided code as a 'black box'.

Installation of deSolve and FME

For this computer lab you need two R packages: deSolve and FME. To install these packages, first start Rstudio. Next go to the menu 'Tools' → 'Install Packages'. From the popup menu you can install packages from CRAN. Search for the deSolve and FME package and install them on your computer.

Introduction to deSolve

Here we provide a short tutorial that explains how to setup and solve ordinary differential equations in R/deSolve. deSolve includes R functions that numerically solve (a) initial value problems of a system of first-order ordinary differential equations (ODEs), (b) partial differential equations (PDEs), (c) differential algebraic equations (DAEs; mixture of differential and algebraic equations) and (d) delay differential equations (DDEs).

In this tutorial we focus on ODEs. deSolve was developed by Karline Soetaert, Thomas Petzoldt, and R. Woodrow Setzer (Soetaert, 2012).

The R package deSolve

deSolve has several built-in functions for computing a numerical solution of initial value problems (IVP) for ODEs. A simplified form of the syntax for solving ODEs is

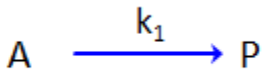
```
ode(y, times, func, parms, …….)
```

where *times* holds the times at which output is wanted, *y* holds the initial conditions, *func* is the name of the R function that describes the differential equations, and *parms* contains the parameter values (or is NULL).

Many additional inputs can be provided, e.g., the integration method, the absolute and relative error tolerances and the maximum number of steps of the integration method. We will not discuss these additional inputs in this tutorial but you can find more information in the deSolve manual pages.

1.1 Example 1. Conversion of A to P

The simplest possible reaction is the irreversible conversion of substance *A* to product *P* (e.g., radioactive decay; see also Appendix I):



The corresponding differential equation is given by:

$$\frac{dC_A(t)}{dt} = -kC_A(t)$$
$$C_A(t=0) = 0$$

Here C_A represents the concentration of A. To implement this initial value problem (IVP) in R we first define the parameter *k* and the initial condition *C(0)*. **What are the units of C and k?**

```
k=0.2     #proportionality constant
Cini=10   #initial value of concentration at t=0
```

The simple differential equation is implemented in an R function called '*derivs*' and takes as arguments the current time (*t*), the value of the dependent variable (*C*) and a parameter vector (*parms*), and returns the derivative as a R list. The parameter *k*, although defined outside of function *derivs* is also known within the derivative function:

```
derivs = function(t, C, parms) {
   dC = -k*C            #calculate the derivative
   return (list(dC))    #return the derivative
}
```

We require output at half day intervals for 20 days, which we specify in the vector '*times*' by using the R function 'seq':

```
times = seq(from =0, to=20, by=0.5)
```

The model is solved, using the R function 'ode'. The integrator ode is available from the package deSolve, which is loaded first:

```
library(deSolve)
out = ode(y=Cini, times=times, func=derivs, parms=NULL)
```

Note that we have set *parms=NULL* because we defined the parameter *k* as a global variable.

The output of this model is a matrix consisting of two columns, first time then the state variable C. We print the first five lines of this matrix and plot the results of the model:

```
head(out,n=5)
plot(out, type='l', col='blue', xlab='time',
ylab='C',main="First order reaction")
```

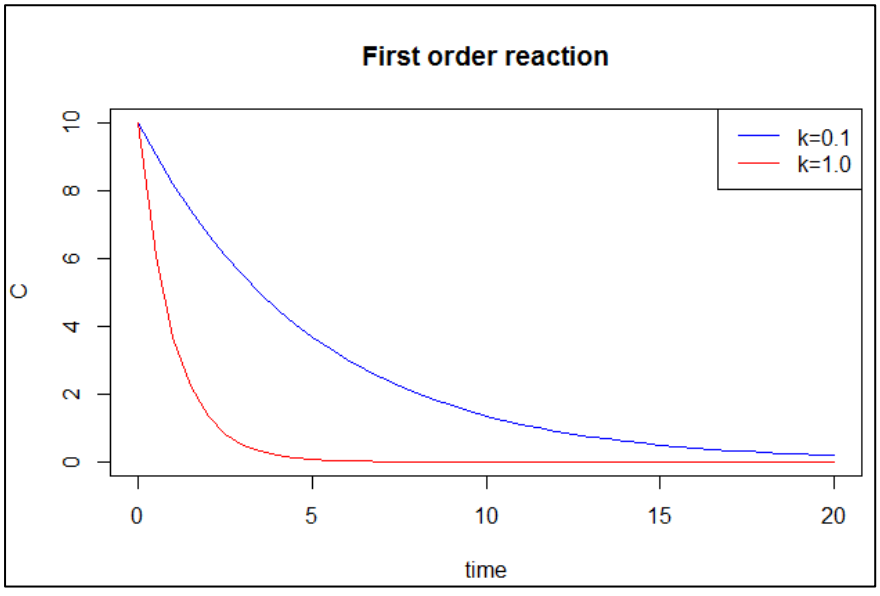
Exercise 1.1. Implement your first model

Now you have all the R code to implement your first model in R. In R Studio start a new script to implement this code ('+' icon in top-left corner). Copy/paste the R code in this script. Subsequently, you can select one or more lines with your mouse and execute the selected part of the script but clicking 'run' (from command bar on top of your script).

Execute the script two times with different values for *k*. What is the effect of *k*? Hint: run the model twice and collect the output in **out1** and **out2**. Then use the following code to plot the results:

```
plot(out1,out2, type='l', col=c('blue','red'),lty=c(1,1),
        xlab='time', ylab='C',main="First order reaction")

legend("topright",legend=c("k=0.1","k=1.0"),lty=c(1,1),col=c
('blue','red'))
```



1.2 Example 2: Pharmacokinetics

In order to be effective, the concentration of a drug taken by a patient must be large enough, yet too high concentrations may have serious side effects. Pharmacokinetic models are tools

to test the optimal frequency and dosing of drug intake. They represent absorption, distribution, decay and excretion of a drug. In the model below we assume that the drug is dosed orally (pills) and affects the absorption through the gut. The model calculates the drug concentration in the intestine and blood.

The system is defined by a system of two differential equations and, consequently, two initial conditions:

$$\frac{dI}{dx} = -aI + u(t) \quad \text{intestine}$$

$$\frac{dB}{dx} = aI - bB \quad \text{blood}$$

$$I(0) = 0$$

$$B(0) = 0$$

where *I* and *B* represent the drug concentration in the intestine and blood respectively. Both *I* and *B* are time dependent and should actually be written as I(t) and *B(t)* but this has been omitted for sake of brevity. The constant *a* is the absorption rate, and *b* is the removal rate from the blood. We assume that at *t=0* the drug concentration is zero. Note that the drug that disappears from the intestine (*-aI*), appears in the blood (*+aI*) (mass action kinetics).

What are the units of I, B, u, a and b?
Draw a cartoon (figure) on paper to represent this biological system.

Definition mass action kinetics: a kinetic scheme for chemical reaction networks which says that the rate of a chemical reaction is proportional to the product of the concentrations of the reacting chemical species. It remains one of the most common kinetic assumptions used by chemists, biologists, and mathematicians.

Since we have two initial values we use the vector y_{ini} to provide these initial values to 'ode'.

What makes this model a little more complex is the dosing of the drug to the intestine. We assume that dosing *u(t)* assumes a constant value for 1 hour, after which it is 0 for the rest of the day. Since the uptake is periodic we can use the modulo function (%%) to represent the uptake of the drug. Recall that the modulo operation finds the remainder of division of one number by another. Thus 6 %% 2 = 0 and 6 %% 4 = 2.

We calculate the model for 10 days and require output every hour.

The R code of this model is give below. Note that the drug concentrations in the intestine and blood are defined as *y1* and *y2*, and *u(t)* is defined as *uptake*.

```
library(deSolve)

a = 6
b = 0.6
yini = c(intestine = 0, blood = 0)

pharmacokinetics = function(t, y, p) {
  if ( (24*t) %% 24 <= 1)
    uptake = 2
  else
    uptake = 0
  dy1 = - a* y[1] + uptake
  dy2 =  a* y[1] - b *y[2]
  list(c(dy1, dy2))
}

times = seq(from = 0, to = 10, by = 1/24)
out = ode(func = pharmacokinetics, times = times, y =
yini)

head(out) #inspect the output

#plot the concentrations is separate plots
plot(out, lwd = 2, xlab = "day")

#plot both concentrations in a single plot
matplot(out[,-1],type='l')
```

Exercise 1.2. Run the pharmacokinetic model

A. Without running the model, make a sketch of the drug concentrations in the intestine and in blood.
B. Now run the model in R Studio and make sure you understand the code. Does this agree with your expectations?

This concludes the second part. You now should have a basic understanding of the deSolve package and how this can be used to solve differential equations.

However, there is one more thing to explain. We can slightly modify the code give above to implement the model by using the R construct 'with':

```
parameters = c(a = 6, b = 0.6) #vector of parameters
yini = c(intestine = 0, blood = 0) #vector of initial
conditions

pharmacokinetics <- function(t, y, parms) {
  with(as.list(c(y,parms)), {
     if ( (24*t) %% 24 <= 1)
       uptake = 2
     else
       uptake = 0
     dy1 = - a* intestine + uptake
     dy2 =  a* intestine - b*blood
     list(c(dy1, dy2))
  })
}

times <- seq(from = 0, to = 10, by = 1/24)
out <- ode(func = pharmacokinetics, times = times,
           y = yini, parms = parameters)
```

Instead of defining *a* and *b* as global variables we have now made them part of a vector 'parameters' that is provided to the function 'ode'. The use of 'with' allows to use the names of the variables in the vectors 'parameters' and 'yini'. Thus instead of using *y[1]* and *y[2]* we use now 'intestine' and 'blood' in the function for the derivatives. This improves the readability of the code and avoids problems with the use of global variables.

Note: In the answers of the exercises of Part II, we have used the 'with' construct. Thus, we suggest that you use it also when implementing the SIR model.

Part II. Implementation and analysis of the SIR model

In the exercises below we will model the outbreak of an infection in a population. The model we use is based on the SIR model of Kermack and McKendrick (1927). SIR refers to susceptible to disease (*S*), infected individuals (*I*) and Removed (Recovered) individuals (*R*) (because of acquired immunity). The SIR model is shown in Figure 1

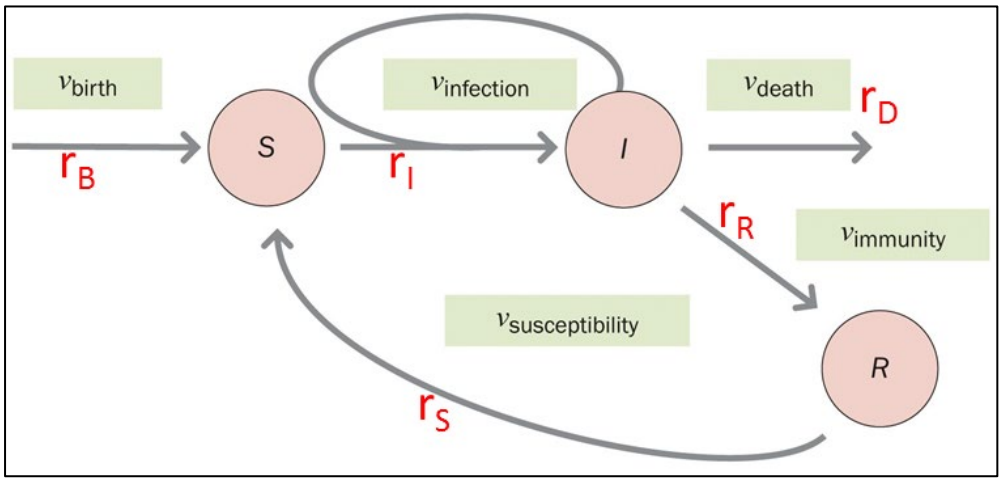


Figure 1. The SIR model according to Kermack and McKendrick (1927). Birth can also be interpreted as immigration.

Exercise 2.1. Define the SIR model

You have seen that the ordinary differential equation (ODE) for the susceptible individuals can be defined as:

$$\frac{dS}{dt} = r_B + r_S R - r_I SI \qquad\qquad S(t_0) = S_0$$

S_0 denotes the initial condition. Note that these equations are symbolic. We specify values for the parameters (*r*) and initial conditions later.

A. What are the units for r_B, r_S, and r_I?
B. As a first step write down the ODEs and initial conditions for *I* and *R*.
C. This model is a simplification of the biological reality. Mention one simplification.
D. Make a sketch on paper of the expected dynamics of this model. That is, how do *S*, *I* and *R* change over time?

Exercise 2.2. Implementation and calculation of SIR model

A. Implement the SIR model in R and use the package deSolve to solve the ODEs. Perform a simulation for 365 days and request output for every day.

Assume that the parameters of this model are known from literature. Use the following values:

$$r_B = 3$$
$$r_S = 0.01$$
$$r_I = 0.0005$$
$$r_R = 0.05$$
$$r_D = 0.02$$

Use the following initial conditions:

$S_0 = 990$

$I_0 = 10$

$R_0 = 0$

Thus, 99% of the individuals in a population of 1000 are healthy, yet susceptible to disease (S=990). 1% of the population is infected for unknown reasons. Nobody is initially immune.

B. How do you interpret the results?

C. How many individuals have died? Can you change to model to explicitly show this?

Exercise 2.3: Diagnosis: determining the steady state of the system

In a steady state none of the variables (*S, I* and *R*) change anymore in time. That is, *dS/dt*, *dI/dt* and *dR/dT* are zero.

A. For this simple model we can easily determine the steady state without a computational method just by solving the resulting linear equations. Derive the steady state values for *S, I* and *R*. What is the total population size at steady state? What do you conclude from this? Use the *R* model to verify your steady state values.

B. What will happen to the steady state values of *S*, *I* and *R* if we start with an initial population size of *S*=1500? What happens if we start with *S*=700?

Exercise 2.4. Diagnosis: robustness of SIR model

Next we will make a (small) change to one of the parameters to investigate the effect the outcome.

A. Reduce the infection rate to 20% of its present value (change r_I = 0.0005 to r_I = 0.0001) and run the SIR model for 730 days. Is there any change? What do you conclude?

B. Make r_i 100 times large and run the model again to compare.

C. Assume that individuals obtain permanent immunity. Change and run the model again. Will this system reach a steady state?

Exercise 2.5. Exploration: asymptomatics

Suppose it becomes clear that many infected individuals do not show any serious symptoms and often do not even know that they carry the disease. Such individuals are called asymptomatics (*A*). To account for these individuals, we can make a modification to the model as depicted in Figure 2.

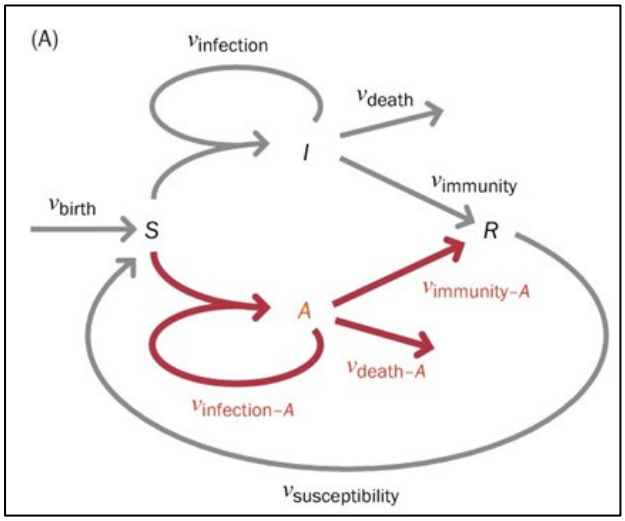


Figure 2. Extension of the SIR model to include asymptomatics (A)

Here we assume that contact between *S* and *I* cannot lead to *A* and, vice versa, contact between *S* and *A* cannot lead to *I*. Whether or not this is a valid assumption should be based on our prior biological and/or medical knowledge of the disease process.

A. Change the equations of the SIR model to account for asymptomatic individuals (A).
B. Implement and calculate the model in R. Assume that the infection of asymptomatic individuals is less severe.

Exercise 2.6. Model assessment: global sensitivity analysis

The parameters of the SIR model have certain variability (**why?**) and consequently can vary within a certain range. Changes in one or more of the parameters affect the outcome (*S*, *I*, and or *R*). Here we will evaluate the effect of parameter changes.

We will first perform a global sensitivity analysis that assesses the effects of parameters on model outcome over a large region in the space of biologically reasonable parameter values. A global sensitivity analysis can be performed with the R function 'sensRange' from the R 'FME' package.

In the previous exercises we have used the values shown below:

$$r_B = 3$$
$$r_S = 0.01$$
$$r_I = 0.0005$$
$$r_R = 0.05$$
$$r_D = 0.02$$

We will now investigate the effect of changes in these parameters if we vary them within a certain range:

$$r_B \in [0,5]$$
$$r_S \in [0,0.2]$$
$$r_I \in [0,0.001]$$
$$r_R \in [0,0.1]$$
$$r_D \in [0,0.1]$$

We will use the same initial conditions: *S*=990, *I*=10, *R*=0.

To define the parameter ranges use the following R constructs:

```
parRanges <- data.frame(min = c(0,0,0,0,0), max = c(5, 0.2, 0.001,0.1,0.1))
rownames(parRanges) <- c("rb", "rs", "ri","rr", "rd")
parRanges
```

To be compatible with the FME sensRange function, you must embed the 'ode' function in another, self-defined, function:

```
solveSIR = function(parms, yini, times) {
   out <- ode(y = yini, times = times, func = SIR, parms = parms,
              verbose=FALSE, method = "lsoda")
   return(out)
}
```

Now you are ready to perform the analysis using (determine the parameters for sensRange yourself):

```
Library(FME)
sum.sR = sensRange(……)
```

The results of the analysis can be plotted with:

```
par(mfrow=c(2,3))
```

```
plot(summ.sR, xlab = "time, days", ylab = "#individuals",
legpos = "topleft", mfrow = NULL)

plot(summ.sR, xlab = "time, days", ylab = "#individuals", mfrow = NULL,
quant = TRUE, col = c("lightblue", "darkblue"), legpos = "topleft")
mtext(outer = TRUE, line = -1.5, side = 3, "Sensitivity to rB", cex = 1.25)
par(mfrow = c(1, 1))
```

A. Perform a global sensitivity analysis using the sensRange function (use ?sensRange to view the documentation of this function). Only consider the parameter r_B and keep the others fixed. Evaluate the parameter on a grid (dist=grid). Use num=50 to obtain 50 evaluations for each parameter. What is the exact meaning of 'grid'? How do you interpret the results of sensRange?

B. Instead of using the 'grid' approach we can also use the Monte Carlo approach. Use grid=unif (what is the effect of unif?). Perform a Monte Carlo analysis for all parameters simultaneously. What is your conclusion?

Exercise 2.7. Model assessment: local sensitivity analysis

Instead of varying the parameters over a large range we can also investigate the effect of small changes at specific values for the parameters (which may have been obtained from literature or from curve fitting).

The sensFun function of the FME package performs such local sensitivity analysis by considering the (partial) derivatives of the model output with respect to the parameters (not with respect to time):

$$V = \left.\frac{\delta Y(p)}{\delta p}\right|_{p=p_0}$$

Where *Y* is the model output (*S, I* or *R*) and *p* is a parameter. This derivative is evaluated at a value p_0. To obtain dimension-free sensitivity information that enables comparison, sensFun considers the scaled sensitivity:

$$S = V * \frac{\Delta p}{SC}$$

where Δp is an a priori measure of the reasonable range of *p*, which can be chosen as p_0 if little prior knowledge is available and at least accounts for different scales of the parameters. *SC* is a scale factor with the same dimensions of the observation, accounting mainly for

different scales of different output signals (in our case different scale at different time points). *SC* can be chosen as the mean value of the observations. See Brun (2001) for more information.

sensFun results in so-called sensitivity functions for each parameter that denote the scaled partial derivatives over all observations for each model output variable.

sensFun is used as follows:

```
sns = sensFun(func=solveSIR, parms=parameters, yini=yini, times=times365,
      sensvar="S",
      varscale=1, parscale=NULL)
head(sns)
plot(sns,legpos="bottomright")
```

This calculates the sensitivity functions for all parameters for the model output *S*. Here we set *SC* to 1 (varscale) and Δp to p_0 (i.e., parscale=NULL).

A. Perform a local sensitivity analysis for output *S* and all parameters. What is your conclusion?

Acknowledgements

Part of the material in Appendix 1 is taken from 'Chris de Koster, Huub Hoefsloot, Gooitzen Zwanenburg (2012) Modellen van Gezondheid en Ziekte (computer exercises for biomedical students at the University of Amsterdam). The exercise in Appendix 1 is taken from Bittinger (1993). The exercises in Part I are taken from Soetaert (2012). The exercises in Part II are based on Voit (2013). I thank Marcel Willemsen for proofreading this document.

R and R studio

The Amsterdam Doctorate School also provides the course 'Computing in R'. You can find this course on our website https://bioinformaticslaboratory.eu/2021/08/gs-computing-in-r/.

References

Blanchard, P., Devaney, R.L., and Hall, G.R. (2011). Differential Equations, 4th Edition Edition, (Canada: Brooks Cole, Cengage Learning).

Bittinger, M.L., Morrel, B.B. (1993) Applied Calculus (3rd edition). Reading: Addiston-Wesley publishing company.

Brun, R., Reichert, P., & Kfinsch, H. R. (2001). Practical identifiability analysis of large environmental simulation. Water Resources Research, 37(4), 1015–1030.

Kermack WO, McKendrick AG (1927) Contributions to the mathematical theory of epidemics. Proc. R. Soc. Lond. A, 155, 700-721.

Soetaert, K., Cash, J., & Mazzia, F. (2012). *Solving Differential Equations in R*. (R. Gentleman, G. Parmigiani, & K. Hornik, Eds.). Berlin: Springer-Verlag.

Voit, E. O. (2013). *A first course in systems biology*. New York: Garland Science, Taylor & Francis Group, LLC.

Appendix 1. nano-tutorial 'differential equations'

Note: if you are familiar with differential equations then you can skip this part.

1.1 Differential equations

A differential equation is any equation which contains derivatives, either ordinary derivatives (ODE; the dependent variable is a function of only one independent variable) or partial derivatives (PDF; more than one independent variable; e.g., time and position). Below we only consider ODEs.

There is one ordinary differential equation that everybody probably knows, that is Newton's Second Law of Motion. If an object of mass *m* is moving with acceleration *a* and being acted on with force *F* then Newton's Second Law tells us:

$$F = m \times a$$

To see that this is in fact a differential equation we need to rewrite it a little. First, remember that we can rewrite the acceleration, a, in one of two ways:

$$a = \frac{dv}{dt} \quad \text{or} \quad a = \frac{d^2u}{dt^2}$$

where *v*=velocity, *u*=distance and *t*=time. Acceleration is the change of velocity in time, or the change of change of distance in time.

Consequently, Newton's second law can be rewritten as one of the following two differential equations:

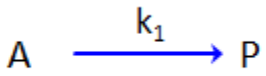
$$1: \quad m\frac{dv}{dt} = \mathrm{F(t,v(t))}$$

$$2: \quad m\frac{d^2u}{dt^2} = F(t, u(t), \frac{du(t)}{dt})$$

The first differential equation is a first order differential equation. The second equation is a second order differential equation. Note that a 'normal' equation (algebraic equation) such as $y=x^2$ has a number as solution. In contrast, a differential equation has not a number but a function as solution. A differential equation gives a relation between a function and its derivatives.

1.2 Example: First-Order irreversible reaction

The simplest possible reaction is the irreversible conversion of substance *A* to product *P* (e.g., radioactive decay):



We assume that this reaction is 'first order', i.e., the amount of *A* that is converted to *P* is proportional with *A*. Thus, the more we have of *A* the more of *P* will be formed per second. If the concentration of *A* doubles, then twice as much of *A* will be converted to *P*. This can be described in a differential equation:

$$\frac{dC_A(t)}{dt} = -k_1 C_A(t)$$

where C_A is the concentration of *A* and k_1 is a proportionality constant with dimension 'per second'. This equations shows that the change of concentration *A* in time, $dC_A(t)/dt$, is proportional with the concentration of *A*. The minus sign indicates that the concentration of *A* reduces in time. This reaction represents a first order reaction since the change in concentration is proportional with the concentration of *A* to the power of 1. Don't confuse this with first order differential equations.

In the next step we solve the differential equation to determine the concentration *A* as function of time. For this ODE the solution is very simple. We are searching for a function of which the derivative is the function itself multiplied with a constant. In this case we can use an exponential function as solution. We try the following function:

$$C_A(t) = \alpha e^{\beta t}$$

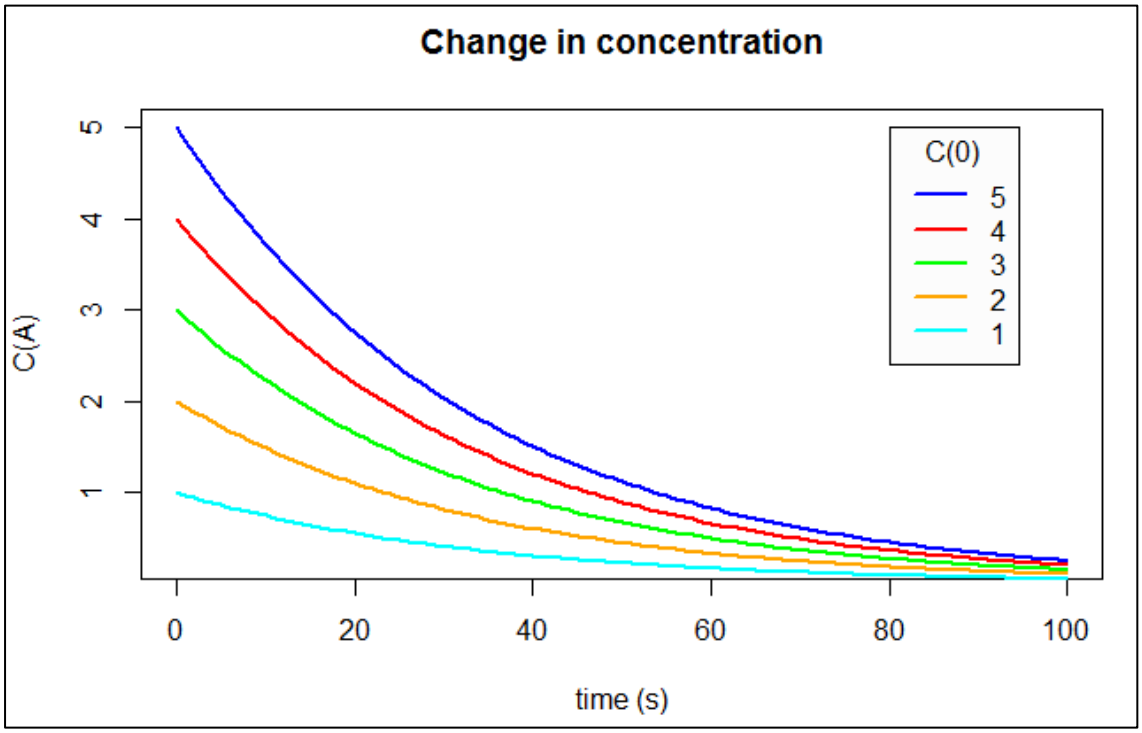
Substituting in the differential equation gives:

$$\frac{dC_A(t)}{dt} = -k_1 C_A(t)$$
$$\frac{d}{dt}\alpha e^{\beta t} = -k_1 \alpha e^{\beta t}$$
$$\alpha\beta e^{\beta t} = -k_1 \alpha e^{\beta t}$$
$$\beta = -k_1$$

We see that a solution of this differential equation is β = $-k_1$, giving

$$C_A(t) = \alpha e^{-k_1 t}$$

However, we cannot determine the value of α. This is a common feature of differential equations. In general, there is a family of functions to provide solutions to the differential equation. In Figure 1 below five solutions for the differential equation are drawn; each with a different value of α (corresponding to different initial conditions *C(0)*; see below).



Figuur 1. Family of functions that provide solutions to the ordinary differential equation. The lines represent solutions given various initial conditions, C(0).

To obtain a unique solution we also have to provide an initial value for the concentration. Hence, this type of problem is called an initial value problem (IVP). This is the value that the function assumes at *t=0*. In our example this is the starting concentration of *A*. Let's call this concentration *C*. Now we can determine the solution from the family of solutions that provide the correct initial value:

$$C_A(t) = \alpha e^{-k_1 t}$$
$$C_A(0) = C = \alpha e^{-k_1 0}$$
$$\alpha = C$$

Thus, the solution of this differential equation with initial condition *C(0)=C* is

$$C_A(t) = Ce^{-k_1 t}$$

Thus, in this example, we now solved the differential equation for the first-order reaction by using a *trial function* of which we determined the unknown parameters (α, β) by substituting the trial function in the differential equation and using the initial conditions. A more general and formal approach to solve differential equations is through integration.

We integrate both sides of the differential equation from *t'=0* to *t'=t*:

$$\int_{t'=0}^{t'=t} \frac{dC_A(t')}{dt'} dt' = -k_1 \int_{t'=0}^{t'=t} C_A(t') dt'$$

$$\Rightarrow \int_0^t dC_A(t') = -k_1 \int_0^t C_A(t') dt'$$

$$\Rightarrow \int_0^t \frac{dC_A(t')}{C_A(t)} = -k_1 \int_0^t dt'$$

t' is a dummy variable for integration without any further meaning. *t* denotes the time. Next you need some knowledge about integration and primitive functions. The derivative of *ln(x)* is *f(x) = 1/x* thus the left integral is equal to $ln(C_A(t'))$. The right integral results in *t'*:

$$\ln C_A(t') \Big|_0^t = -k_1 t' \Big|_0^t$$

$$\Rightarrow \ln C_A(t) - \ln C_A(0) = -k_1 (t - 0)$$

$$\Rightarrow \ln \frac{C_A(t)}{C_A(0)} = -k_1 t$$

$$\Rightarrow \frac{C_A(t)}{C_A(0)} = e^{-k_1 t}$$

$$\Rightarrow C_A(t) = C_A(0) e^{-k_1 t}$$

$$\Rightarrow C_A(t) = C e^{-k_1 t}$$

The initial value of $C_A(t)$ (i.e., *C*) enters the solution correctly because we integrated from *t=0* to *t=t*.

Note: only do the next exercise if you are interested in practicing your integration skills. In Part II and III we will numerically solve differential equation models and will not search for analytical solutions like we have done in this part. In fact, many differential equations are impossible to solve analytically.

Exercise 1.1. The Weber-Fechner law

In psychology, one model of stimulus-response asserts that the rate of change *dR/dS* of the response *R* (of an individual) with respect to a stimulus *S* is inversely proportional to the stimulus:

$$\frac{dR}{dS} = \frac{k}{S}$$

where *k* is a positive constant.

Assume that S_0 is the lowest level of stimulus that can be detected and that the for this stimulus the response $R(S_0)$ of the individual is 0.

Solve this equation through integration to obtain the function *R*.

Answer

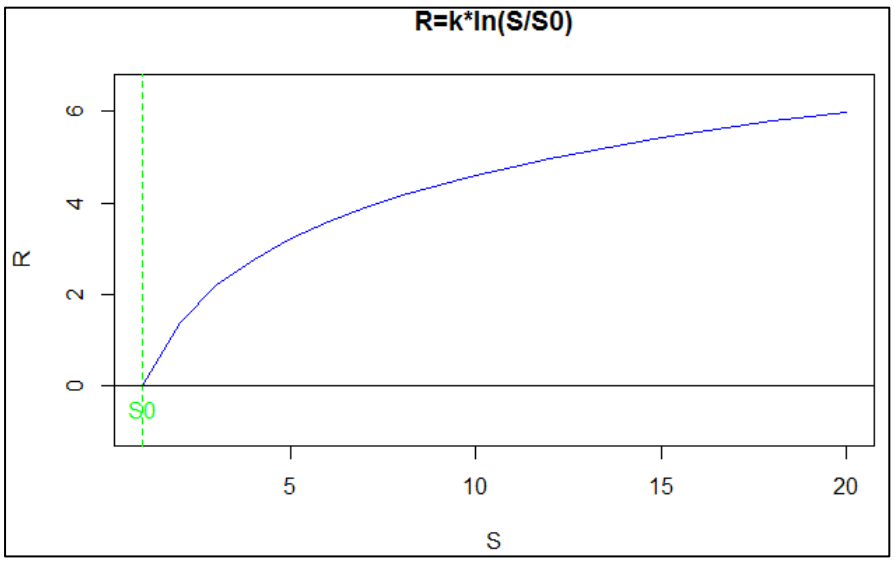
$$\frac{dR}{dS} = \frac{k}{S}$$

$$dR = \frac{k}{S} dS$$

$$\int_{S'=S_0}^{S'=S} dR = \int_{S'=S_0}^{S'=S} \frac{k}{S'} dS'$$

$$R(S) - R(S_0) = k\left[\ln(S) - \ln(S_0)\right]$$

$$R(S) = k \ln\left[\frac{S}{S_0}\right]$$



R code to make this plot:

```
x=(1:20)
r=2*log(x)/1
plot(x,r,type='l',col='blue',xlab='S', ylab='R', main='R=k*ln(S/S0)',ylim=c(-1,6.5))
abline(v=1,col='green',lty=2)
abline(h=0,col='black',lty=1)
text(1,-0.5,"S0",col='green')
```