Numerical Integration in R (deSolve)

Rudolf P. Rohr

4/5/2020

Introduction

During this practical, you will have to solve numerically ordinary differential equations (ode). This can be done in R with the library "deSolve". As an example, we will use Malthus' model of population growth:

$$\frac{dN}{dt} = N \cdot r$$

See chapter 2 of the script for Malthus's model, and chapter 5 for a short introduction to numerical integration of ode.

Library "deSolve" and function "ode" for one-dimensional models (one state variable like Malthus' model)

First, install the library deSolve and load it. The function, we will use is "ode":

```
library(deSolve)
?ode
```

The function "ode" has 5 input arguments. The last argument is the "method". The standard method that we should first use is "ode45". If it did not work then it is better to ask an expert of numerical analysis. For this practical it will always work.

The 4th argument, "parms" is the list of parameters of the model. Here, it is simply the intrinsic growth rate r. The parameters have to be given as a list:

```
p <- list(r = 0.1)
```

The 3rd argument, "func", is the function that defines the model, i.e., the differential equation. This function has three arguments; the time "t", the state of the system "N", and the parameters "p":

```
f <- function(t,N,p){
  dN <- p$r * N
  return(list(c(dN)))
}
```

The time argument "t" will never be used during this practical. It offers to the possibility to have time dependant differential equations, but do not remove it when writing the head of your differential equation function. The state argument "N" is the population size at which the time derivative will be evaluated. In general it is a list, one element per dimension of the system. Malthus's model is one-diemnsional, therefore, it is simply a list of one element here. The parameters argument "p" allows using the parameter list in the function. Note, it is very important to return the value of the derivative "dN" as list.

The 2nd argument "times" is a vector of time steps at which we want to have the evaluation of the numerical solution. Here, I chose to go from time 0 to time 10, by steps of 0.01:

```
time_steps <- seq(0,10,0.01)
```

The 1st argument "y" is the initial condition, i.e., the population size at time 0. Here I chose $N(0) = 0.1$:

```
N0 <- 0.1
```

Finally we can run the "ode" function and store its output in a variable "out":

```
out <- ode(y = N0, times = time_steps, func = f, parms = p, method = c("ode45"))
```

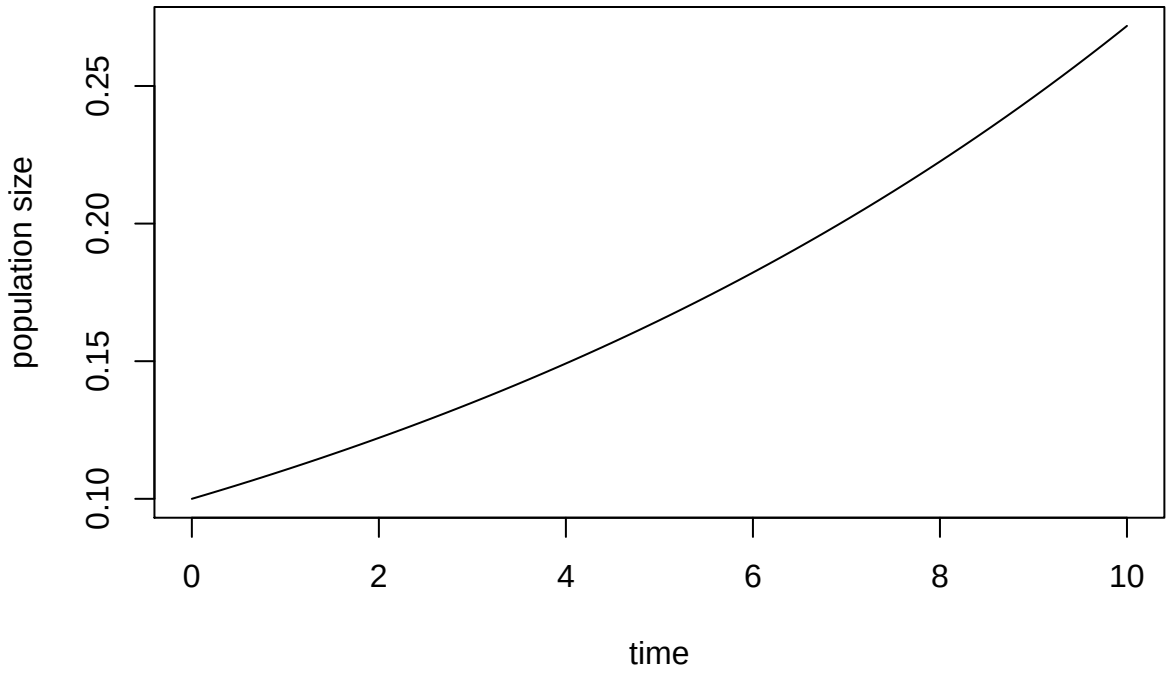
The output variable "out" is a matrix. Each row is a time step. The first column gives the time at which the solution was evaluated The other columns give the sate of the system. Here there is only one column for the state, as Malthus's model is one-dimensional.

```
head(out)
```

```
##      time         1
## [1,] 0.00 0.1000000
## [2,] 0.01 0.1001001
## [3,] 0.02 0.1002002
## [4,] 0.03 0.1003005
## [5,] 0.04 0.1004008
## [6,] 0.05 0.1005013
```

Finally, we can plot the time trajectory of the model:

```
plot(x = out[,1], y = out[,2], type = "l", xlab = "time", ylab="population size")
```



Function "ode" for two (and more) dimensional model

As an example of a two-dimensional system, we will use the following set of two ordinary differential equations:

$$\frac{dN_1}{dt} = N_1 \cdot r - \alpha N_1^2 - \beta \cdot N_1 \cdot N_2$$

$$\frac{dN_2}{dt} = +\beta N_1 \cdot N_2$$

The list of parameters:

```
p_2 <- list(r = 0.2, alpha = 0.1, beta = 0.2)
```

The differential equation function:

```
f_2 <- function(t,N,p){
  N1 <- N[1]
  N2 <- N[2]
  dN1 <- p$r * N1 - p$alpha * N1 * N1 - p$beta * N1 * N2
  dN2 <- + p$beta * N1 * N2
  return(list(c(dN1,dN2)))
}
```

The time steps

```
time_steps_2 <- seq(0,50,0.05)
```

The initial condition:

```
N0_2 <- c(0.1, 0.2)
```

Running the "ode" function:

```
out_2 <- ode(y = N0_2, times = time_steps_2, func = f_2, parms = p_2, method = c("ode45"))
```

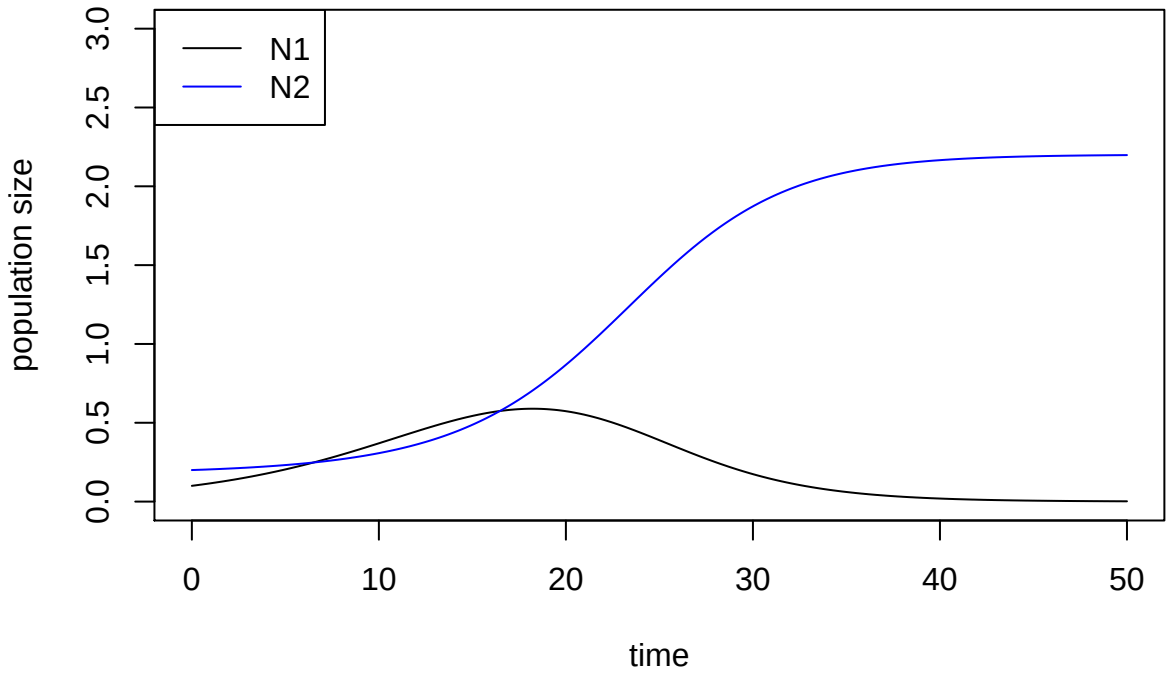
The output variable "out_2:

```
head(out_2)
```

```
##      time         1         2
## [1,] 0.00 0.1000000 0.2000000
## [2,] 0.05 0.1007525 0.2002009
## [3,] 0.10 0.1015101 0.2004034
## [4,] 0.15 0.1022728 0.2006077
## [5,] 0.20 0.1030407 0.2008138
## [6,] 0.25 0.1038136 0.2010216
```

Finally, the plot:

```
plot(x = out_2[,1], y = out_2[,2], type = "l", xlab = "time", ylab="population size", ylim=c(0,3))
lines(x = out_2[,1], y = out_2[,3], col='blue')
legend("topleft", legend = c("N1","N2"), col= c("black","blue"), lty = c(1,1))
```


N1
N2
population size
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
10
20
30
40
50
time