

UQAR – Chaire de Recherche EEC

## Ordinary differential equations

Numerical analysis

**Dominique Gravel**

<http://www.chaire-eec.uqar.ca/>

August 27, 2013

- 1 Introduction
- 2 Objectives
- 3 Euler method
- 4 Runge Kutta
- 5 Libraries

**Objective:** If  $f(N, t)$  is a derivative for the rate of change of the population  $N$ , we are interested by the solution giving its dynamics, i.-e. the function  $N(t)$ .

To obtain that information, we need to 'solve' the differential equations, that is to derive the explicit dynamics in time of the state variables.

This procedure is based on *integration* of the differential equations. Integrating the model requires that *initial* conditions are specified.

Integration is based on the fact that:

$$\int d(f(t)) = f(t) + A$$

The integration is the 'reverse' of differentiation. Finding an analytical solution then proceeds in two steps:

1. Finding a *general solution* of the differential equation through integration.
2. The *particular solution* of the model is then found by considering the initial conditions

Take the exponential growth model:

$$\frac{dN}{dt} = rN$$

Which could be re-arranged as:  $\frac{dN}{N} = rdt$

Integrating over time:  $\int \frac{dN}{N} = \int rdt$

$$\int d \log N = \int rdt \quad \log N = rt + A'$$

$$N(t) = Ae^{rt} \text{ where } A = e^{A'}$$

The value of  $A$  can be calculated with our knowledge of  $N_0$ ,  $t_0$  and  $r$ :

$$N(t_0) = N_{t_0} = Ae^{rt_0}$$

Which allows writing the constant  $A$  as a function of the known initial condition,  $N_{t_0}$ :

$$A = N_{t_0} e^{-rt_0}$$

Putting back this solution in the solution of the previous slide we get:

$$N(t) = N_{t_0} e^{-rt_0} e^{rt}$$

Which simplifies to:

$$N(t) = N_{t_0} e^{rt}$$

The **BIG** problem:

most biological systems have non-linear equations describing dynamics and therefore are impossible to integrate. We must therefore rely on numerical methods to perform the integration.

1. Chaos
2. Paradox of enrichment



1. Euler method
2. Runge Kutta method
3. packages rootSolve and deSolve

1. loops (for, while)
2. indexing vectors and matrices
3. conditional statements
4. functions
5. rootSolve and deSolve packages

- ▶ Calculate the time series of the geometric growth model between  $t_0$  and  $t_{10}$  and with  $N_0 = 10$  and  $\lambda = 2$ ;
- ▶ Plot on the same figure the analytical solution
- ▶ Find the equivalent growth rate  $r$  for the exponential growth model
- ▶ Perform the numerical integration of the exponential growth model
- ▶ Plot on the same figure the analytical solution

The numerical integration of differential equations consist of making *discrete* jumps or steps in time:

$$t_0 \Rightarrow t_1 = t_0 + \Delta t \Rightarrow t_2 = t_1 + \Delta t \Rightarrow \dots t_n$$

where  $\Delta t$  is the time step. Note that this time step is not necessarily constant, some methods allow it to vary through time.

The simplest updating formula is to use Taylor expansion:

$$C^{t+\Delta t} = C^t + \Delta t \frac{dC^t}{dt} + O(\Delta t)$$

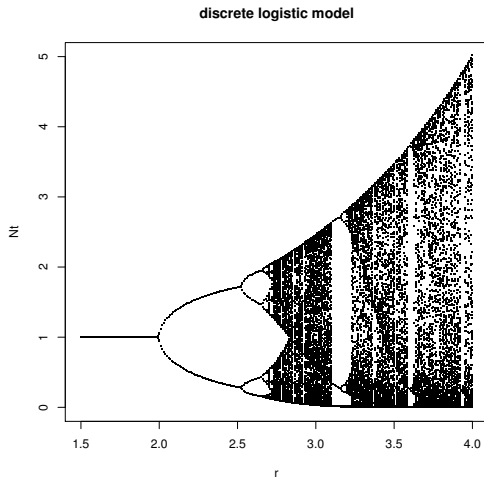
$$C^{t+\Delta t} \cong C^t + \Delta t \frac{dC^t}{dt}$$

where  $O(t)$  is the error made by the Taylor approximation. The method is called *forward differencing* or Euler integration and said to be first-order accurate, as all higher order terms are ignored.

An important assumption of this method is that the rate of the change (the derivative function) is constant in the interval  $\Delta t$ .

- ▶ Program a function to integrate the logistic growth model with the Euler method between  $t_0$  and  $t_1$  00 and with  $N_0 = 0.1$ ,  $r = 1$   $K = 1$   

$$\frac{dN}{dt} = rN(1 - \frac{N}{K})$$
- ▶ Run it with  $r = 1, 2, 2.5, 2.6, 2.8, 3.5, 4.0$
- ▶ Run it also with  $r = 4.0$ , but for  $N_0 = 0.11$ . Compare the solution with  $N_0 = 0.1$
- ▶ Advanced exercise: compute the bifurcation map



```
1 ricker = function(N,r) N*exp(r*(1-N))
2 rseq = seq(1.5,4,0.01) # sequence of r-values
3
4 plot(0,0,xlim=range(rseq),ylim=c(0,5),type="n",
5      xlab="r",ylab="Nt",main="discrete logistic model")
6
7 for ( r in rseq)
8 {
9   N = runif(1)
10  for (i in 1:200) N = ricker(N,r) # spinup
11  for (i in 1:200) {
12    N = ricker(N,r)
13    points(r,N,pch=".",cex=1.5)}
14 }
```

# Numerical approximation and numerical errors

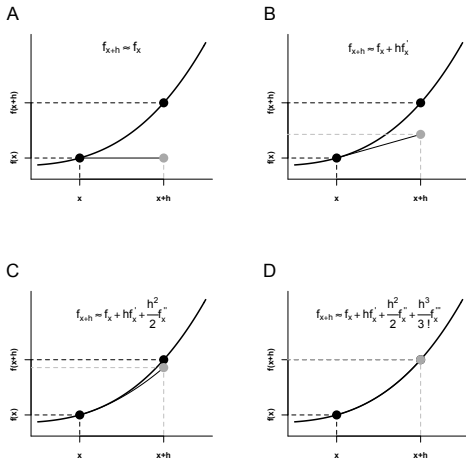
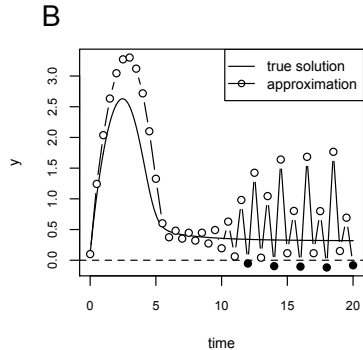
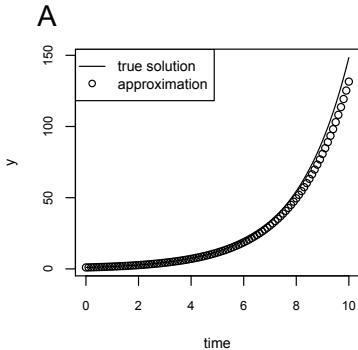


Fig. 6.2

Soetaert and Herman - chapter 6



# Numerical approximation and numerical errors



```
1 library(ecolMod)
2 demo(chap6)
```

**Accuracy** is a measure for the correctness of the solution. Relates to the approximation of the solution. Generally higher order methods are more accurate.

**Stability** refers to the potential of the method to lead to increasing oscillations between consecutive solution points. A consequence of the approximations. Some models are more sensitive than others.

**Speed** is the inverse of time required to compute the numerical integration. It does not matter nowadays for simple systems of differential equations, but it could be significant for large systems such as food webs or metacommunities.

**Memory** increases with the complexity of the integration routine (basically the high order derivatives in the Taylor expansion)

**Principle:** the method use extra evaluations of the differential equations at various positions in time and interpolate between them. The most common method is the 4th order:

$$N_{t+\Delta t} = N_t + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where:

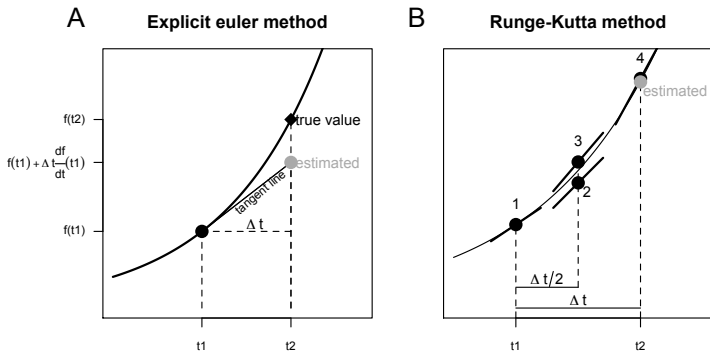
$$k_1 = f(t, N_t)$$

$$k_2 = f(t + \frac{\Delta t}{2}, N_t + \frac{\Delta t}{2} k_1)$$

$$k_3 = f(t + \frac{\Delta t}{2}, N_t + \frac{\Delta t}{2} k_2)$$

$$k_4 = f(t + \Delta, N_t + \Delta t k_3)$$

## Numerical approximation and numerical errors



```

1  # The function to integrate
2  dNdt = function(N,lambda) lambda*N
3
4  # Calculate the amount of change:
5  k1 = dNdt(N0,lambda = 1)
6  k2 = dNdt(N0 + 0.5*k1,lambda = 1)
7  k3 = dNdt(N0 + 0.5*k2,lambda = 1)
8  k4 = dNdt(N0 + k3,lambda = 1)
9  delta = (k1 + 2*k2 + 2*k3 + k4)/6
10
11 # Calculate the next density
12 N1 = N0 + delta

```

Redo the steps of exercise 2, but with the Runge Kutta 4 integration method. Compare the results. Try different  $\Delta t$ .

Compute the numerical integration of the Rosensweig-MacArthur model of predator prey interactions:

$$\begin{aligned}\frac{dR}{dt} &= rR\left(1 - \frac{R}{K}\right) - \frac{aRC}{1+bR} \\ \frac{dC}{dt} &= \frac{aRC}{1+bR} - dC\end{aligned}$$

Consider the parameters  $r = 1$ ,  $a = 1$ ,  $b = 5$ ,  $d = 0.1$ . Try different values of  $K$  within the range  $[0.4, 0.8]$ . What happens?



```
1 # Load the library
2 library(deSolve)
3
4 # Define the mathematical model
5 model = function(Time, State, Pars) {
6   with(as.list(c(State, Pars)), {
7     dR = r*R*(1-R/K) - a*R*C/(1+b*R)
8     dC = a*R*C/(1+b*R) - d*C
9     list(c(dR, dC))
10   })
11 }
```

```
1 # Define the parameters
2 r = 1
3 a = 1
4 b = 5
5 K = 0.6
6
7 # Collect them in a vector
8 pars = c(r = r, a = a, b = b, d = d, K = K)
9
10 # Set the initial conditions
11 T0 = c(R = 1, C = 0.1)
12
13 # Set the conditions for the simulation
14 times = seq(0, 1000, by = 0.1) # THE BY ARGUMENT SPECIFIES THE INTEGRATION STEP
```

```
1 # Run the simulation
2 out = ode(func=model, y = T0, parms = pars, times = times)
3
4 # Plot the results
5 par(mar = c(5,6,2,1))
6 plot(out[,1],out[,2],type="l",xlab = "Time",ylab = "Density",cex.lab = 1.75,
7      cex.axis = 1.5,ylim=range(out[,2:3]))
8 lines(out[,1],out[,3],col = "blue")
```

```
1 library(rootSolve)
2
3 # Specify the model
4 model = function(t,y,pars) {
5   with(as.list(c(y,pars)), {
6     dR = r*R*(1-R/K) - a*R*C/(1+b*C)
7     dC = a*R*C/(1+b*R) - d*C
8     list(c(dR,dC))
9   })
10 }
11
12 # Parameters
13 pars = c(r = 1, K = 0.7, a = 1, b = 5, d = 0.1)
14
15 # Initial conditions
16 T0 = c(R = 0.1, C = 1.1)
17
18 # Solve the model using stode (iterative state solver)
19 eq = stode(y=T0, func=model, parms=pars, pos=TRUE)
```

```
1 # Compute the jacobian
2 J = jacobian.full(y=eq, func=model, parms=pars)
3
4 # Compute the eigen values
5 eigen(J)$values
```

## Example: boundary conditions with the R-M model

```

1 # Vecteur of K values
2 Ks = seq(0.5,1,0.01)
3
4 # Vector to store the results
5 res = numeric(length(Ks))
6
7 # Loop to calculate eigen values for each K value
8 for(i in 1:length(Ks)) {
9   pars = c(r = 1, K = Ks[i], a = 1, b = 5, d = 0.1)
10  eq = stode(y=eq, func=model, parms=pars, pos=TRUE)[[1]]
11  J = jacobian.full(y=eq, func=model, parms=pars)
12  res[i] = max(as.real(eigen(J)$values))
13 }
14
15 # Plot the results
16 par(mar = c(5,6,2,1),mfcol=c(1,3))
17 plot(Ks,res, type = "l", xlab = "K", ylab = "Maximal eigen value", cex.axis = 1.25, cex.lab = 1.5)
18 abline(h = 0, lty = 3)
19
20 # Analytical criteria
21 a = 1
22 b = 5
23 d = 0.1
24 abline(v = (1+d*b)/(a*b*(1-d*b)), lty = 3, col = "red")

```

## Other useful functions to explore....

1. `runsteady`
2. `uniroot.all`
3. `multiroot`
4. `steady.2D` (advanced, for diffusion models)