UQAR – Chaire de Recherche EEC

Ordinary differential equations

Numerical analysis

Dominique Gravel

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

August 27, 2013

UQAR

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: $\int \frac{dN}{N} = \int rdt$

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

$N(t) = Ae^{rt}$ 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

Technical objectives

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:
$t_0 \Rightarrow t_1 = t_0 + \Delta t \Rightarrow t_2 = t_1 + \Delta_t \Rightarrow .....t_n$

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

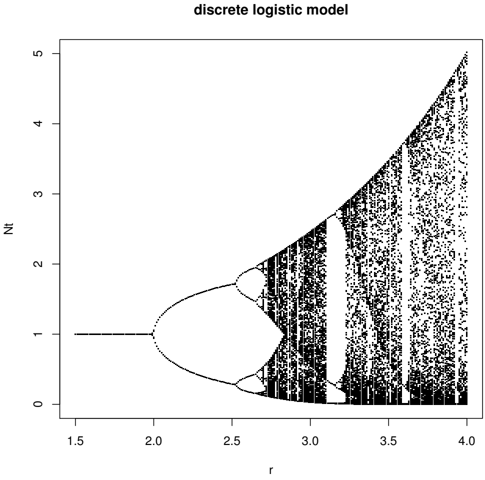
The simplest updating formula is to use Taylor expansion:
$C^{t+\Delta t} = C^t + \Delta t \frac{dC^t}{dt} + O(\Delta t)$
$C^{t+\Delta t} \approxeq C^t + \Delta t \frac{dC^t}{dt}$

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 Δt.

Exercise 2

- Program a function to integrate the logistic growth model with the Euler method between t_0 and t_100 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


discrete logistic model
Nt
0
1
2
3
4
5
r
1.5
2.0
2.5
3.0
3.5
4.0

The R script

```
ricker = function(N,r) N*exp(r*(1-N))
rseq = seq(1.5,4,0.01) # sequence of r-values

plot(0,0,xlim=range(rseq),ylim=c(0,5),type="n",
xlab="r",ylab="Nt",main="discrete logistic model")

for ( r in rseq)
  {
N = runif(1)
  for (i in 1:200) N = ricker(N,r)   # spinup
for (i in 1:200) {
N = ricker(N,r)
points(r,N,pch=".",cex=1.5)}
}
```

Numerical approximation and numerical errors

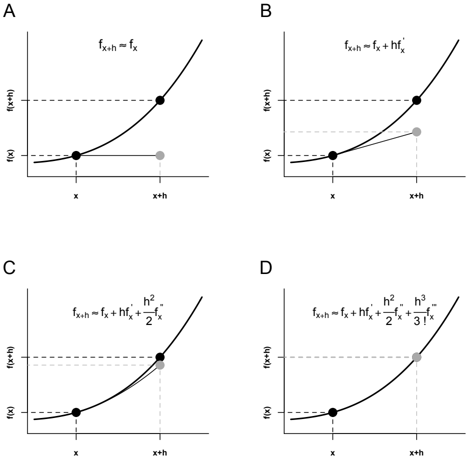
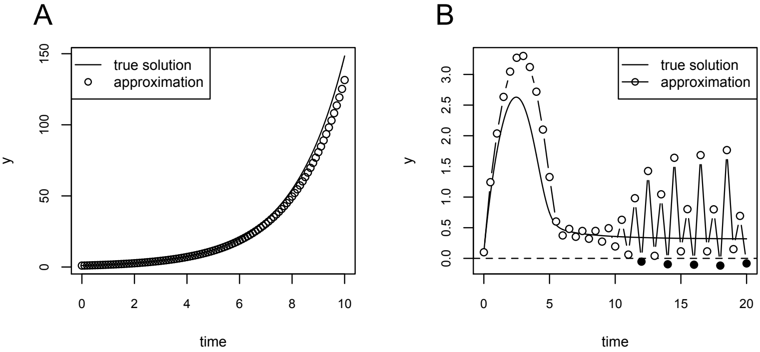


Fig. 6.2

Soetaert and Herman - chapter 6

Numerical approximation and numerical errors



The R script

```
library(ecolMod)
demo(chap6)
```

Accurracy is a measure for the correctness of the solution. Relates to the approximation of the solution. Generally higher order methods are more accurante.

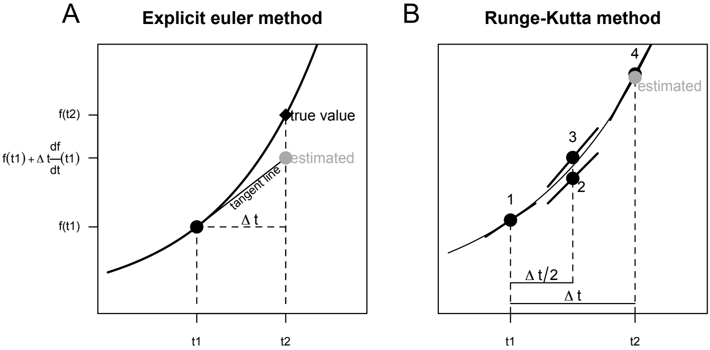
Stability refers to the potential of the method to lead to increasing oscillations between consecutive solution points. A consequence of the approximations. Some models are more sensitive than others.

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:
$k_1 = f(t, N_t)$
$k_2 = f(t + \frac{\Delta t}{2}, N_t + \frac{\Delta t}{2}k_1)$
$k_3 = f(t + \frac{\Delta t}{2}, N_t + \frac{\Delta t}{2}k_2)$
$k_4 = f(t + \Delta, N_t + \Delta t k_3)$


A
Explicit euler method
f(t2)
$f(t1) + \Delta t \frac{df}{dt}(t1)$
f(t1)
true value
estimated
tangent line
Δ t
t1
t2
B
Runge-Kutta method
4
estimated
3
2
1
Δ t/2
Δ t
t1
t2

```
# The function to integrate
dNdt = function(N,lambda) lambda*N

# Calculate the amount of change:
k1 = dNdt(N0,lambda = 1)
k2 = dNdt(N0 + 0.5*k1,lambda = 1)
k3 = dNdt(N0 + 0.5*k2,lambda = 1)
k4 = dNdt(N0 + k3,lambda = 1)
delta = (k1 + 2*k2 + 2*k3 + k4)/6

# Calculate the next density
N1 = N0 + delta
```

Exercise 3

Redo the steps of exercise 2, but with the Runge Kutta 4 integration method. Compare the results. Try different Δt.

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

$$\frac{dR}{dt} = rR(1 - \frac{R}{K}) - \frac{aRC}{1+bR}$$

$$\frac{dC}{dt} = \frac{aRC}{1+bR} - dC$$

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?

```
# Load the library
library(deSolve)

# Define the mathematical model
model = function(Time, State, Pars) {
with(as.list(c(State,Pars)), {
dR = r*R*(1-R/K) - a*R*C/(1+b*R)
dC = a*R*C/(1+b*R) - d*C
list(c(dR,dC))
})
}
```

```r
# Define the parameters
r = 1
a = 1
b = 5
K = 0.6

# Collect them in a vector
pars = c(r = r, a = a, b = b, d = d, K = K)

# Set the initial conditions
T0 = c(R = 1, C = 0.1)

# Set the conditions for the simulation
times = seq(0, 1000, by = 0.1) # THE BY ARGUMENT SPECIFIES THE INTEGRATION STEP
```

```
# Run the simulation
out = ode(func=model, y = T0, parms = pars, times = times)

# Plot the results
par(mar = c(5,6,2,1))
plot(out[,1],out[,2],type="l",xlab = "Time",ylab = "Density",cex.lab = 1.75,
cex.axis = 1.5,ylim=range(out[,2:3]))
lines(out[,1],out[,3],col = "blue")
```

```
library(rootSolve)

# Specify the model
model = function(t,y,pars) {
with(as.list(c(y,pars)), {
dR = r*R*(1-R/K) - a*R*C/(1+b*C)
dC = a*R*C/(1+b*R) - d*C
list(c(dR,dC))
})
}

# Parameters
pars = c(r = 1, K = 0.7, a = 1, b = 5, d = 0.1)

# Initial conditions
T0 = c(R = 0.1, C = 1.1)

# Solve the model using stode (iterative state solver)
eq = stode(y=T0, func=model, parms=pars, pos=TRUE)
```

```
# Compute the jacobian
J = jacobian.full(y=eq,func=model,parms=pars)

# Compute the eigen values
eigen(J)$values
```

Example: boundary conditions with the R-M model

```r
# Vecteur of K values
Ks = seq(0.5,1,0.01)

# Vector to store the results
res = numeric(length(Ks))

# Loop to calculate eigen values for each K value
for(i in 1:length(Ks)) {
pars = c(r = 1, K = Ks[i], a = 1, b = 5, d = 0.1)
eq = stode(y=eq, func=model, parms=pars, pos=TRUE)[[1]]
J = jacobian.full(y=eq,func=model,parms=pars)
res[i] = max(as.real(eigen(J)$values))
}

# Plot the results
par(mar = c(5,6,2,1),mfcol=c(1,3))
plot(Ks,res, type = "l", xlab = "K", ylab = "Maximal eigen value", cex.axis = 1.25, cex.lab = 1.5)
abline(h = 0, lty = 3)

# Analytical criteria
a = 1
b = 5
d = 0.1
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)