Metapoblaciones

Integración con `deSolve`

Gerardo Martín

28-07-2023

Integración con paquete `deSolve`

Necesidad real

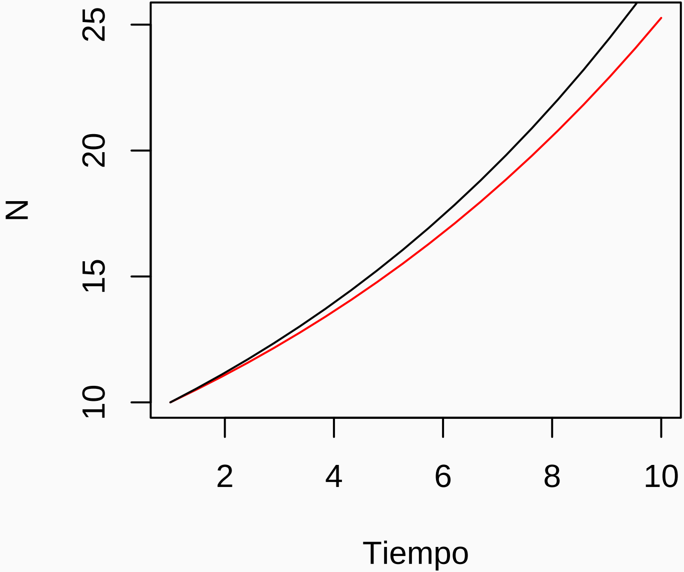


Figure 1: Línea negra es la solución analítica. Roja es solución de Euler

Necesidad real

- El método de Euler es muy inexacto
- El error de integración se acumula
- Se puede controlar, disminuyendo h, pero se vuelve leeento
- Métodos como Runge-Kutta de 2 y 4 pasos tienen menos error
    - Adams-Bashford son más sofisticados y rápidos
- Están implementados en paquete `deSolve` de `R`

Uso de `deSolve`

1. Crear función del modelo
2. Crear objeto con valores de parámetros
3. Establecer condiciones iniciales
4. Correr simulación confunción `lsoda`

La función del modelo

Crear funciones con R

- Funciones: código que contiene órdenes para R
- Se suelen crear cuando se necesita repetir una operación
- Sintaxis:

```r
f <- function(x){print(x)}
```

- Para especificar una función se crea un objeto que contentrá la órdenes
- El objeto se llama, y entre `( )` se especifican los argumentos

- La función `f` requiere un sólo argumento de nombre `x`
- Una vez que llamamos `a` tenemos que especificar el valor de `x`, y `R` imprimirá el resultado:

```r
f(1)
```

```
## [1] 1
```

Funciones con más argumentos

Las funciones pueden tomar más de un argumento:

```r
g <- function(x, y){print(x + y)}
g(1, 3)
```

```
## [1] 4
```

Ó utilizar argumentos de más de un tipo (números y caracteres)

```r
h <- function(x, y, z = "a"){print(paste0(x + y, "=", z))}
h(1, 2, "b")
```

```
## [1] "3=b"
```

Especificando la función del modelo exponencial

La función necesita tres argumentos, el tiempo `t`, los valores `y` y los parámetros del modelo:

```
expon <- function(t, y, parms){


}
```

Entre los corchetes `{}`, especificamos las posiciones de `y` que contienen las variables de estado (`N`)

```
N <- y[1]
```

las operaciones de que consiste el modelo, el exponencial:

```
dN <- r * N
```

Función completa del modelo

```r
expon <- function(t, y, parms){
  N <- y[1]
  with(parametros,{
    dN <- r * N
    return(list(dN))
  })
}
```

los argumentos `t` y `parms` los veremos a continuación

Argumentos de la función

`t` es una secuencia de valores del tiempo:

```
t <- seq(0, 10, by = 0.1)
```

`y` es un objeto que sólo contiene las condiciones iniciales:

```
y <- 10
```

`parms` es una lista que contiene los valores que cada parámetro:

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

Llamando `deSolve` para correr simulación

```r
library(deSolve)

sim <- lsoda(y = y, times = t, parms = parametros, func = exp
```

`lsoda` es la función de `deSolve` que hará la simulación

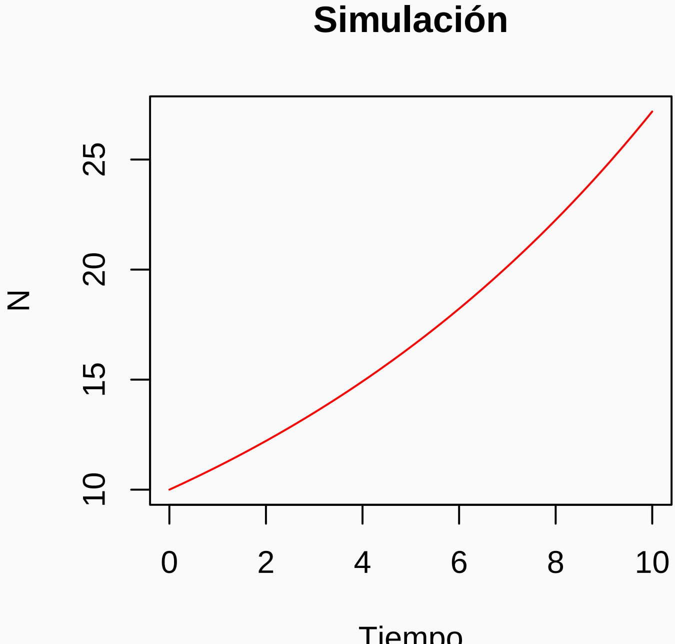
Los argumentos, ¿se explican solos?

Explorando la salida de `lsoda`

Podemos imprimir las primeras filas

```
head(sim)
```

```
##      time        1
## [1,]  0.0 10.00000
## [2,]  0.1 10.10050
## [3,]  0.2 10.20202
## [4,]  0.3 10.30455
## [5,]  0.4 10.40811
## [6,]  0.5 10.51271
```


Simulación
N
Tiempo
10
15
20
25
0
2
4
6
8
10

Simulación de un modelo con más de un parámetro

Función del modelo de levins

```r
levins <- function(t, y, parms){
  p <- y[1]
  with(parms, {
    dp <- c*p*(1-p) - e*p
    return(list(dp))
  })
}
```

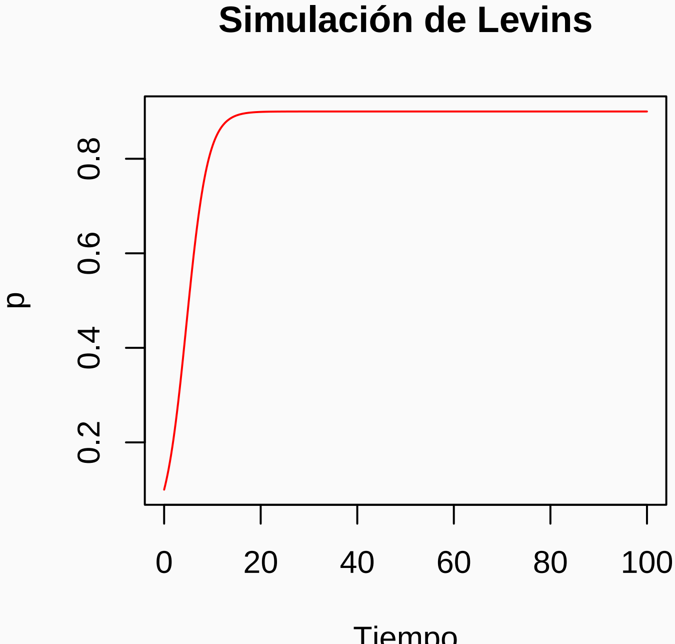
t, y, parms

```r
t <- seq(0, 100, by = 0.1)
y <- 0.1
parms <- list(c = 0.5, e = 0.05)
```

Simulando Levins

```r
sim.lev <- lsoda(y = y, times = t,
                 parms = parms,
                 func = levins)
head(sim.lev)
```

```
##      time         1
## [1,]  0.0 0.1000000
## [2,]  0.1 0.1040708
## [3,]  0.2 0.1082849
## [4,]  0.3 0.1126455
## [5,]  0.4 0.1171553
## [6,]  0.5 0.1218178
```


Simulación de Levins
p
0.2
0.4
0.6
0.8
0
20
40
60
80
100
Tiempo

Simulación de un modelo con dos variables de estado

El modelo

$$\frac{N_1}{dt} = rN_1(1 - N_1/K) + iN_2 - eN_1 \quad (1)$$

$$\frac{N_2}{dt} = rN_2(1 - N_2/K) + iN_1 - eN_2 \quad (2)$$

La función

```
mig <- function(t, y, parms){
  with(parms, {
      N1 <- y[1]
      N2 <- y[2]

    dN1 <- r * N1 * (1 - N1/K) + m2 * N2 - m1 * N1
    dN2 <- r * N2 * (1 - N2/K) + m1 * N1 - m2 * N2

    return(list(c(dN1, dN2)))
  })
}
```

t, y, parms

```r
t <- seq(0, 10, by = 0.1)
y <- c(N1 = 10, N2 = 0)
parms <- list(r = 0.5, K = 25,
              m1 = 0.1, m2 = 0.2)
```

La simulación

```
sim.mig <- lsoda(y = y, times = t,
                 parms = parms,
                 func = mig)
head(sim.mig, 3)
```

```
##      time       N1        N2
## [1,]  0.0 10.00000 0.0000000
## [2,]  0.1 10.20099 0.1025226
## [3,]  0.2 10.40392 0.2101706
```

1. Necesitamos transformar la tabla generada a formato largo

```
sim.mig.df <- data.frame(sim.mig)
sim.mig.largo <- reshape2::melt(sim.mig.df, id.vars = "time")
head(sim.mig.largo, 3)
```

```
##   time variable    value
## 1  0.0       N1 10.00000
## 2  0.1       N1 10.20099
## 3  0.2       N1 10.40392
```

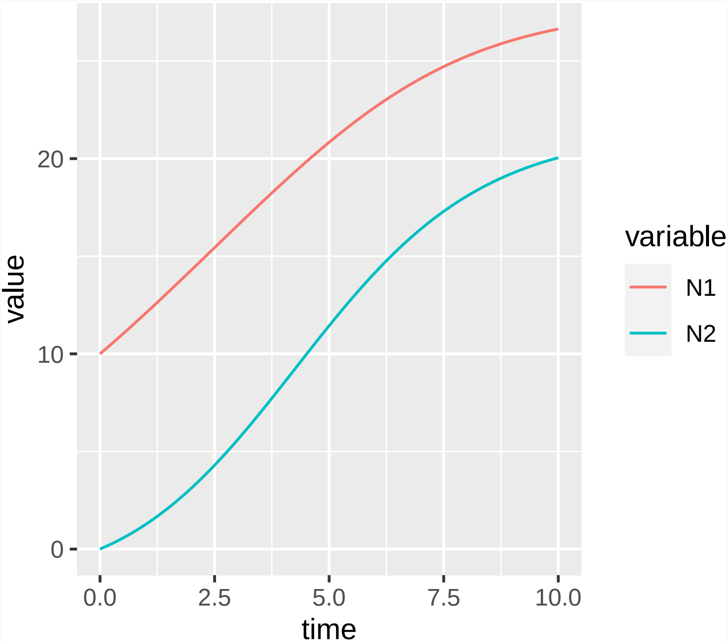
2. Necesitamos cargar el paquete `ggplot2`

```
library(ggplot2)
```

Graficando con ggplot2

```
ggplot(sim.mig.largo) + geom_line(aes(x = time,
                                      y = value,
                                      colour = variable))
```

1. En `ggplot`, llamamos la tabla que contiene los datos
2. A `ggplot` agregamos elementos geométricos, para líneas: `geom_line`
3. A los elementos geométricos especificamos los elementos "estéticos" con `aes`
4. En `aes` indicamos las coordenadas `x, y` y la variable que indica el color de las líneas



0
10
20
value
0.0
2.5
5.0
7.5
10.0
time
variable
N1
N2

Una representación alternativa de la trayectoria