Session 7: ODE Simulation + Noise

Foundations of Quantitative Ecology (EEOB 8896.11)

Paul J. Hurtado
(hurtado.10@mbi.osu.edu)

Mathematical Biosciences Institute (MBI)
The Ohio State University

Last compile: October 3, 2013

The Rosenzweig-MacArthur Model

Classical predator-prey (consumer-resource) model. Assumptions? Logistic prey (R) growth, type-II predation, consumption drives predators (C) reproduction, and age-independent mortality:

$$\begin{aligned}\frac{dR}{dt} &= rR(1-R/K) - \frac{aRC}{1+aT_hR}\\ \frac{dC}{dt} &= \chi\frac{aRC}{1+aT_hR} - \mu C\end{aligned}$$

Dynamics: Steady-state coexistence at

$$R_* = \frac{\mu}{a(\chi-\mu T_h)}; \quad C_* = \frac{r}{a}\left(1-\frac{R_*}{K}\right)(1+aT_hR_*)$$

if

$$K < \frac{1}{aT_h}\left(\frac{\chi+\mu T_h}{\chi-\mu T_h}\right)$$

Simulating ODEs

Simulate ODEs using the `deSolve` package:

```
library(deSolve)  #install.packages('deSolve') if needed
dydt <- function(ts, y, params) {
    r = params[1]
    K = params[2]
    a = params[3]
    Th = params[4]
    conv = params[5]
    mu = params[6]
    dy1dt = r * y[1] * (1 - y[1]/K) - a * y[1] * y[2]/(1 + a * Th * y[1])
    dy2dt = conv * a * y[1] * y[2]/(1 + a * Th * y[1]) - mu * y[2]
    return(list(c(dy1dt, dy2dt)))
}
y0 = c(50, 5)
times = 0:500
prms = c(r = 0.2, K = 100, a = 0.02, Th = 1, conv = 1, mu = 0.4)
out1 = ode(y0, times, dydt, prms, method = "lsoda")
```

Simulating ODEs

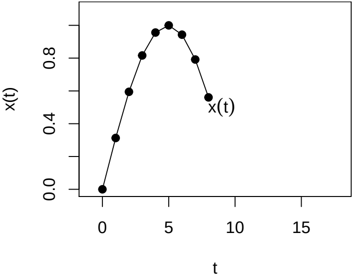
How does `ode()` work? A differential equation (e.g. $\frac{dx}{dt} = f(x)$) models *rates of change in x*. Since the derivative is approximately

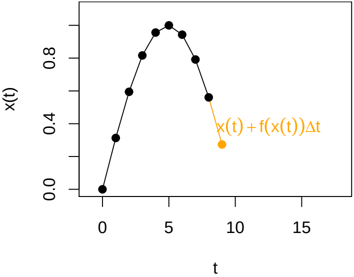
$$\frac{x(t+\Delta t) - x(t)}{\Delta t} \approx f(x(t))$$

we can rearrange things to approximate a step forward in time by

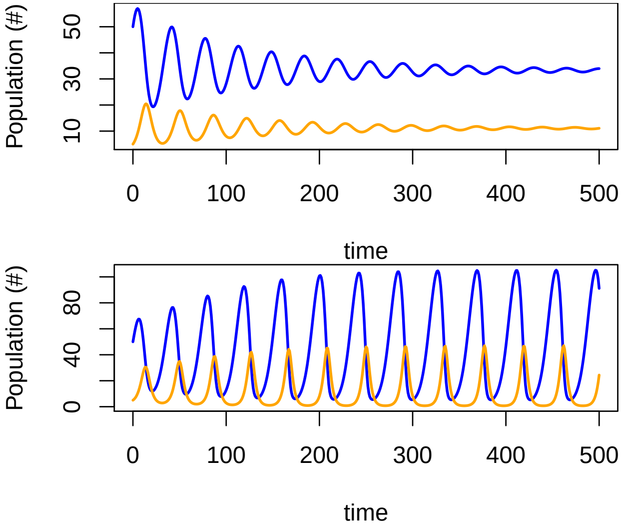
$$x(t+\Delta t) \approx x(t) + f(x(t)) \cdot \Delta t$$

That is, we add a change given by rate $f(x)$ times time step, Δt. The function `ode()` does something similar.

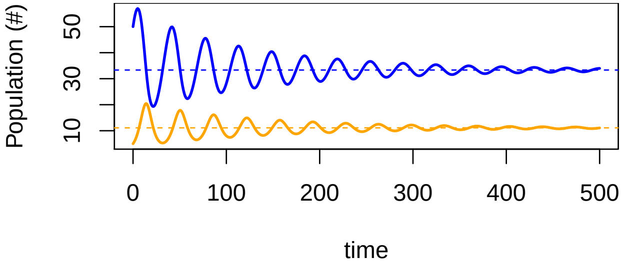


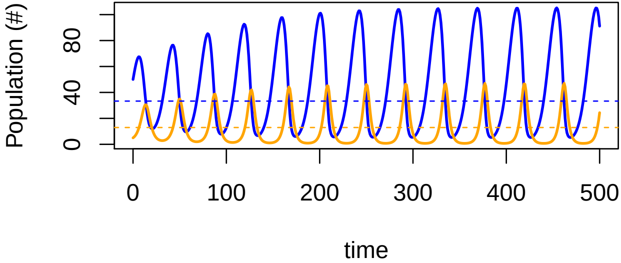


Simulating ODEs



Simulating ODEs





Simulating ODEs

Code for the plots on the previous slide:

```
matplot(out1[,1], out1[,-1], type="l", lwd=2, col=c("blue","orange"),
        lty=1, xlab="time",ylab="Population (#)")
with(as.list(prms),{
  abline(h = mu/(a*(conv-mu*Th)),col="blue",lty=2)
  abline(h = r/a*(1-(mu/(a*(conv-mu*Th)))/K)*
             (1+a*Th*(mu/(a*(conv-mu*Th)))), col="orange",lty=2)
  })
prms2 <- prms; prms2["K"] <- 150;
out2=ode(y0, times, dydt, prms2, method="lsoda");
matplot(out2[,1], out2[,-1], type="l", lwd=2, col=c("blue","orange"),
        lty=1,xlab="time",ylab="Population (#)")
with(as.list(prms2),{
  abline(h = mu/(a*(conv-mu*Th)),col="blue",lty=2)
  abline(h = r/a*(1-(mu/(a*(conv-mu*Th)))/K)*
             (1+a*Th*(mu/(a*(conv-mu*Th)))), col="orange",lty=2)
  })
```

Exercise: Recreate these plots using the `ggplot2` package instead of `matplot()`. *Hint: start by googling "matplot ggplot2 -python".*

Simulating ODEs + 'Intrinsic Noise'

Exercise #1: Implement the Stochastic Simulation Algorithm (SSA; aka Gillespie Algorighm) for the above model with demographic stochasticity.

1. Compute a rate for some event (birth, death, predation)
2. Draw a random exponential time step to the next event
3. Use a random uniform to decide which event happened
4. Update the state variables, then repeat.

Simulating ODEs + 'Intrinsic Noise'

Pseudocode for Exercise #1:

```
## Use the parameter values from the ODE script
#
## Setup for iterating forward in time...
times = c(0) # initialize
y     = y0   # initial conditions from above
i     = 0    # initialize indexing/count variable
#
## Iterate forward in time until we hit t=500...
  # recompute rates, total event rate, event probs
  # Pick a random time step and update times
  # Pick which event happened and update y values
  # Repeat
#
### Plot and compare with ODE output from above
```

Simulating ODEs + 'Intrinsic Noise'

Components we'll need to put together:

```
event_rates = c(
  r*y[1],             # Increment y[1] only (birth)
  r*y[1]^2/K,         # Decrement y[1] only (natural death)
  a*y[1]*y[2]/(1+a*Th*y[1]), # Decrement y[1] (predation)
  conv*a*y[1]*y[2]/(1+a*Th*y[1]), # Increment y[2] (birth)
  mu*y[2] ## Decrement y[2] (predator death)
)
# Total event rate
S = sum(event_rates)  # rate of "something" happening
# Proportions for picking event type
event_probs = event_rates/S
# Check cases with if-else statments, e.g.
if(event1) { y[,i+1] <- update_accordingly(y[,i]) }
else if(event2) { ... }
...
else if(lastevent) {...}
# OR use something like switch(). Ex:
switch(sample(1:5),"one","two","three","four","five")
switch(which(runif(1) < cumsum(event_probs))[1],
       c(1,0), c(-1,0), c(-1,0), c(0,1),c(0,-1))
```

Simulating ODEs + 'Intrinsic Noise'

Solution:

```
prms = c(r=0.2,K=100,a=0.02,Th=1,conv=1,mu=0.4)
y0 = c(50, 5) # initial conditions
## Unpack our parameter values
r = prms[1]
K = prms[2]
a = prms[3]
Th = prms[4]
conv = prms[5]
mu = prms[6]
## Set up the while loop to iterate steps..
i = 1
times = c(0)
y = cbind(P=y0[1],C=y0[2])
```

Simulating ODEs + 'Intrinsic Noise'

Solution (continued):

```
## Set up the while loop to iterate steps..
i = 1 # index for while loop
times = c(0) # store time values here
y = cbind(P=y0[1],C=y0[2]) # store state values
while(times[i] < 500) {
  # Update which step we're on
  # Recompute our rates
  # ... and total event rate
  # Recompute proportions & pick which event occcured
  # Update y[i,] accordingly
  # update time
}
# plot output
```

Simulating ODEs + ‘Intrinsic Noise’

```
while(sum(y[i,]) > 0 & times[i] < 500) { # Stop if both extinct, t>500
  i <- i+1  # Update which step we're on
  event_rates = c( # Recompute our rates. Events are:
    r*y[i-1,1],             # Increment y[1] only (birth)
    r*y[i-1,1]^2/K,         # Decrement y[1] only (natural death)
    a*y[i-1,1]*y[i-1,2]/(1+a*Th*y[i-1,1]), # Decrement y[1] (predation)
    conv*a*y[i-1,1]*y[i-1,2]/(1+a*Th*y[i-1,1]), # Increment y[2] (birth)
    mu*y[i-1,2] ## Decrement y[2] (predator death)
  )
  # Total event rate
  S = sum(event_rates)  ## rate of "something" happening
  # Proportions for picking which event type occcured
  event_probs = event_rates/S
  # Pick an event type and update y[i,] accordingly
  y <- rbind(y, y[i-1,]+switch(which(runif(1) < cumsum(event_probs))[1],
                       c(1,0), c(-1,0), c(-1,0), c(0,1),c(0,-1)) )
  times[i] = times[i-1] + rexp(1,rate=S) # update time
}
matplot(times,y,type="l",col=c("blue","orange"),lty=1,lwd=2,
        xlab="time",ylab="Population (#)",main="Stochastic")
```

Simulating ODEs

Exercise #1: Modify the above solution so that it all takes place inside a function `SimModel()` that takes a parameter list (like `prms`) and returns `cbind(times,y)`.

Exercise #2: Use that function to create a 2x2 figure with the above ODE model output in the first column, and the stochastic model output for the corresponding parameter values in the second column.

Exercise #3: Modify the solution to **#2** to plot multiple replicates of the stochastic simulations in the second column.

Simulating ODEs + 'Intrinsic Noise'

Q: So what did we just do?

Simulating ODEs + 'Intrinsic Noise'

Q: So what did we just do?
A: We used our ODE terms to parameterize and then simulate a *marked poisson process*.

Simulating ODEs + 'Intrinsic Noise'

Q: So what did we just do?
A: We used our ODE terms to parameterize and then simulate a *marked poisson process*.

Basic Poisson Process Assumptions:

1. Event probabilities are *memoryless*. The probability of an event in a small time interval of length Δt is $\lambda \Delta t$.
2. For N individuals, we can pick Δt small enough so almost always get at most 1 event per time interval. In the limit as $\Delta t \to 0$, we get a continuous time process.

Poisson Processes

Homogeneous Poisson Process:

1. Times between events are exponentially distributed with rate λ.
2. The number of events in a time interval of length T is Poisson distributed with rate λT.

Inhomogeneous Poisson Process:

1. Events happen at time-dependent rate $\lambda(t)$.
2. If there is some upper bound λ on $\lambda(t)$ we can simulate our IPP by *thinning* the HPP with rate λ by throwing out events with probability $\lambda(t)/\lambda$.

Events can be *marked*, according to some other process, to specify event types.

Poisson Processes

Exercise 1: Simulate a poisson process using *rexp* and confirm it has the appropriate poisson distribution.

Exercise 2: Simulate a poisson process with a periodic rate $\lambda(t)$ by thinning the appropriate homogeneous poisson process.