Lab 06: Solving Differential Equations

We have already learned how to use Mathematica to iterate a recursion equation, but how do we solve a differential equation?

In this lab, we will investigate models of population growth that incorporate the Allee effect. The Allee effect describes the fact that survival and reproduction in some species are less successful when there aren’t enough individuals around. Low densities might, for example, make it difficult to find a mate or difficult to form a social coalition to defend territories.

A model of the Allee effect

To model the Allee effect, we want an equation for the growth of the population that is 0 when n=0, is negative when n is below a threshold (a), and is positive above this threshold as long as the population is below carrying capacity (k). One such equation is:

\[ \frac{d n}{d t} = r n (n-a) (1-n/k) \]

(e.g., Hastings 1996).

With this equation, the rate at which the population grows as a function of the current population size (plot of \(dn/dt\) vs n) looks like:

\[ \text{Plot}[r n (n-a) (1-n/k), \{ n, 1000 \}, \{ n, 100 \}, \{ r, 0.01 \}, (n, 0, 1005)] \]

In this lab, we’ll learn how to graph the variable over time (plot of n vs t) given a differential equation.

Numerically solving differential equations

First determine the parameters needed to solve the equation numerically:

- \(r\): a measure of growth rate
- \(k\): carrying capacity
- a: threshold population size, below which the population is unable to grow
- \(n0\): the initial population size at time 0

Mathematica numerically solves differential equations by a series of approximations. The exact algorithm it uses is a trade secret (and is said to be a complicated piece of code), but conceptually we can imagine starting at the initial condition and adding a small line through this point that has the right slope (\(dn/dt\) from the differential equation), we could then move along the line a little bit and repeat, piecing together a series of line fragments. Mathematica does something like this, smoothing the lines, and also spending less time when the variable changes little to focus on times when it changes a lot.

To apply this procedure, you need to tell Mathematica what range of time you want to focus on (because it can’t use the above procedure of guessing for all times). That is, we’ll ask Mathematica to solve for n between time points tmin and tmax (the first and last time points that you want to know about).

To numerically solve a differential equation, we use NDSolve:

\[ \text{NDSolve} \{\text{equations you want to hold true}, \{\text{variables you want to solve for}, (t, tmin, tmax) \}} \]

In the case of the Allee effect model, we have two equations we want to hold true:

\[ D[n(t), t] = r*n(t)*(n(t)-a)*(1-n(t)/k) \]

and

\[ n(0) = n0 \]

and we want to solve for one variable (n[t]).

[RECALL: The \(==\) is used to denote an equation that we want to be true. We’re not setting the right hand side to the left hand side.]

For example, for the standard logistic model (not a model of the Allee effect), the NDSolve command is:

\[ \text{NDSolve} \{\text{equations you want to hold true}, \{\text{variables you want to solve for}, (t, tmin, tmax) \}} \]

Mathematica won’t solve this yet, since it doesn’t know what the parameters are (you can try entering the above equation and seeing what error appears). What we’ll do is define a new function specifying which parameters are needed to obtain the solution. For the logistic model, this would be:

\[ \text{Clear} \{\text{logistic}, n\}; \]

\[ \text{logistic}[r, k, n0, \text{tmin}, \text{tmax}] := \text{logistic}[r, k, n0, \text{tmin}, \text{tmax}] = \text{NDSolve} \{\{D[n[t], t] = r*n[t]*(1-n[t]/k), n(0) = n0\}, \{n[t]\}, (t, \text{tmin}, \text{tmax})\} \]

[NOTE: \(\text{logistic}[r, k, n0, \text{tmin}, \text{tmax}] := \text{logistic}[r, k, n0, \text{tmin}, \text{tmax}] = \) notation ensures that Mathematica remembers the numerical solution that it finds.]

When we call logistic with values for all of the parameters, Mathematica will return its best numerical guess for the function, \(n(t)\), between tmin and tmax. It calls this approximation an "Interpolating function."

For example, with \(r=0.01, k=500, n0=20, \text{tmin}=0, \text{tmax}=100\):

We can then plot the variable n[t] by substituting in this interpolating function:

\[ \text{Plot}[n[t], \{\text{logistic} \{0.01, 500, 0, 100\}, (t, 0, 100)\}, \{n, 0, 300\}] \]

The above is the logistic with slow growth at rate \(r = 0.01\). The next two graphs show \(r = 0.1\) and \(r = 0.5\):

\[ \text{Plot}[n[t], \{\text{logistic} \{0.1, 500, 0, 100\}, (t, 0, 100)\}, \{n, 0, 300\}] \]

\[ \text{Plot}[n[t], \{\text{logistic} \{0.5, 500, 0, 100\}, (t, 0, 100)\}, \{n, 0, 300\}] \]
Question 1: Modify the code to include Allee effects

Write a new function similar to the logistic[] function described above that numerically solves the Allee model:

\[
\frac{dn}{dt} = r \cdot n \cdot (n-a) \cdot (1-n/k)
\]

Question 2: Generate plots

Using this new function, plot n[t] according to the Allee model for the following values of r: 0.01, 0.1, 1. Again use k=500, n0=20, tmin=0, and tmax=100.

First use a threshold population size of a = 25, so that the population size starts just below the threshold.

Then use a threshold population size of a = 15, so that the population size starts just above the threshold.

You might need to adjust the x-axis and/or y-axis plot ranges to see what is going on.

Question 3: Interpret the results

Compare the plots obtained from the logistic and Allee models.

Even using the same values for r, k, n0, tmin and tmax, which model produces faster growth when the population size is between the threshold, a, and the carrying capacity, k?

Think about why this particular model of the Allee effect grows so much faster when the (initial) population size is between the threshold, a, and the carrying capacity, k.

[Hint: Notice that the term (n-a) that is present in the Allee model is effectively equivalent to one in the logistic model.]

General solutions [Read if there is time]