Discrete Time Epidemic Models in R

Eric Lofgren
Eric.Lofgren@unc.edu
October 27, 2012

Section 1: Getting to Know R

What is R?:
R is an extremely powerful, flexible programming language originally designed for statistical computing and data visualization. It has been extended to other aspects of scientific computing including simulation and mathematical modeling.

Why R?:
R was chosen from several alternatives for today's tutorial on the mathematical modeling of epidemics for several reasons.

- Relative ease: R is a high-level, relatively approachable programming language with a robust support community as well as a number of books focusing on the language.
- Strong data visualization capabilities: While we will only scratch the surface of data visualization in R today, data visualization is an important component of the presentation and interpretation of mathematical modeling results.
- Open Source: R is a free, open source, cross-platform programming language available on Windows, OS X and Unix/Linux. It requires no commercial software licensing in order to use.
- Utility: Because of its strength as a statistical analysis language, students in Epidemiology and public health may find some experience with R more directly useful in their daily lives than other languages with less of a statistical focus.

Where to Get Help:
Support for learning R, solving programming problems and debugging code is available from a wide variety of Internet sources. Your first stop should be the R manuals, available at cran.r-project.org/manuals.html. Other useful resources include *The R Journal*, an open access, peer-reviewed journal concerning the R language (journal.r-project.org) and the R listservs - especially *R-help*, *R-sig-ecology* and *R-sig-dynamic-models*. Finally, the online question-and-answer site StackExchange has two excellent resources for getting R questions answered: www.stackoverflow.com for programming-specific questions and

stats.stackexchange.com for R questions requiring more specific statistical expertise.

Installing R:

Hopefully you have already installed an up-to-date copy of R on your computer prior to attending this Learning Institute. However, if you have not, head to the UC Berkeley mirror of the Comprehensive R Archive Network (http://cran.cnr.berkeley.edu) and follow the instructions there to download and install the appropriate version of R for your computer.

Section 2: Basic R Commands

Open R on your computer. On Windows or OS X machines, you should be confronted with a new window called "R Console" or something similar. There you should see some start up information about the version of R you are using and a prompt: >

We are in the console, an interactive terminal that allows us to input commands in R line by line. First, lets try some simple addition:

```
> 10+8
[1] 18
```

Next lets try assigning a value (such as "10") a specific name. In R, this is accomplished by typing *name* <- *value*. For example:

```
> a <- 10
> b <- 8
> a
[1] 10
> b
[1] 8
> a+b
[1] 18
```

Scripts:

It is often useful, when doing more complicated tasks like implementing a mathematical model in R, not to have to input commands line by line. To do this, we can writeout commands in a file called a *script*. Scripts can be written in any text editor by writing your commands in plain text and saving the file with the extension ".r". Commands can then be cut-and-pasted into the R console or run as a chunk using built-in functionality within R.

Alternately, R provides its own script editor, which we will be using for this course. Bring it up by selecting "New Document" (OS X) or "New script" (Windows) from the File menu. Note that you can either cut-and-paste commands into the R console from this window or selectively run chunks of code by highlighting the lines you want to submit, then selecting "Execute" (OS X) or "Run line or selection" (Windows) from the Edit menu.

Functions:

Functions are a special type of command that takes a number of inputs (called *arguments*), does something with those arguments, and then returns a value. Generally, they are written in a general form and then are used later with specific arguments. For example, consider the following function that takes three numbers and returns their mean:

```
my.mean <- function(x, y, z){
 total <- x + y + z
 avg <- total/3
 return(avg)
}
```

Now try running my.mean(5,9,15). We'll explore more complex functions later in the course, but for now, you should have a grasp of the basic syntax. Much of what we'll be doing in R is putting specific values into functions we write, or are already written and bundled into downloadable packages.

Working with and Creating Data:

In addition to assigning a single value a variable name in R, variables may refer to vectors, matrices or more complex data structures. For example:

```
> years <- c(0,1,2,3,4,5)
```

Creates a variable with five different values, which can then be used all at once, as in:

```
> mean(years)
[1] 2.5
```

Additionally, specific parts of a more complicated data structure can be referenced. For example,

```
> years[2]
[1] 1
```

References the 2nd element of the vector *years*. More complex referencing will be introduced later when we create and analyze some of our mathematical models.

Section 3: Basic SIR Model – Discrete Implementation

Below is the code for creating and running a basic deterministic SIR model in discrete time using R. You may enter the code line-by-line, type it into a script to run as a chunk, or download the source code from the course's GitHub repository (https://github.com/elofgren/zombies) - the filename is *basic_sir.r* . I recommend however that you do not rely too heavily on cut-and-paste from existing code, as it is quite difficult to get a feel for what you are doing that way.

Setting Up Initial Conditions:

First, we set up some initial conditions for the model, create a data array our model output will go into, and specify the initial population for each compartment as well as the values for the two parameters needed in a basic SIR model with no demography:

```
#Specify the length of time the model will be run for
t <- 100

#Create array to take model output, and name the columns
 appropriately
output <- array(dim=c(t+1,5))
colnames(output) <- c("S", "I", "R", "N", "Time")

#Specify initial populations in each compartment
output[1,"S"] <- 999
output[1,"I"] <- 1
output[1,"R"] <- 0
output[1,"N"] <- 1000
output[1,"Time"] <- 0

#Assign values to the two transmission parameters beta and gamma
beta <- 0.50
gamma <- 1/10
```

A few things to note: We have created a five column, 101 row array called *output* in order to store our model's output. We then give the columns of that array meaningful names – this is not necessary, but for more complex models being able to reference parts of the array by a name rather than a pair of numbers is extremely helpful. Finally, we assign the first row of *output* for each column a starting value. Note that you can reference any portion of the *output* array as *output[row, column]*.

Next, we run the model, based on the difference equations discussed earlier in the course. This is what is known as a *for-loop*. For each value of *i* between 2 and t+1 (we don't need to run the loop for t = 1 because we have already assigned those values) the computer assigns a value to each column in row *i* based on the values of the rows above it and the model parameters:

```
# Run the model for every time step except time 1 (which was set
 to fixed above)
for (i in 2:(t+1)){
  output[i,"S"] <- max(output[i-1,"S"]
- beta*output[i-1,"S"]*output[i-1,"I"]/output[i-1,"N"],0)
  output[i,"I"] <- max(output[i-1,"I"]
+ beta*output[i-1,"S"]*output[i-1,"I"]/output[i-1,"N"]
- gamma*output[i-1,"I"],0)
  output[i,"R"] <- max(output[i-1,"R"] + gamma*output[i-1,"I"],0)
  output[i,"N"] <-
 max(output[i,"S"]+output[i,"I"]+output[i,"R"],1)
  output[i,"Time"] <- output[i-1,"Time"]+1
}
```

Wrapping everything in the *max( )* function accomplishes something important with discrete time models – it ensures the system cannot drop below zero. While rare, it is possible to have a model that should never drop below zero when written as differential equations, but may drop below zero in the coarser time scales of a difference equation model. By taking the maximum of either the model result **or** zero, we ensure that the model never has negative people in any category, something that plays havoc with the model system.

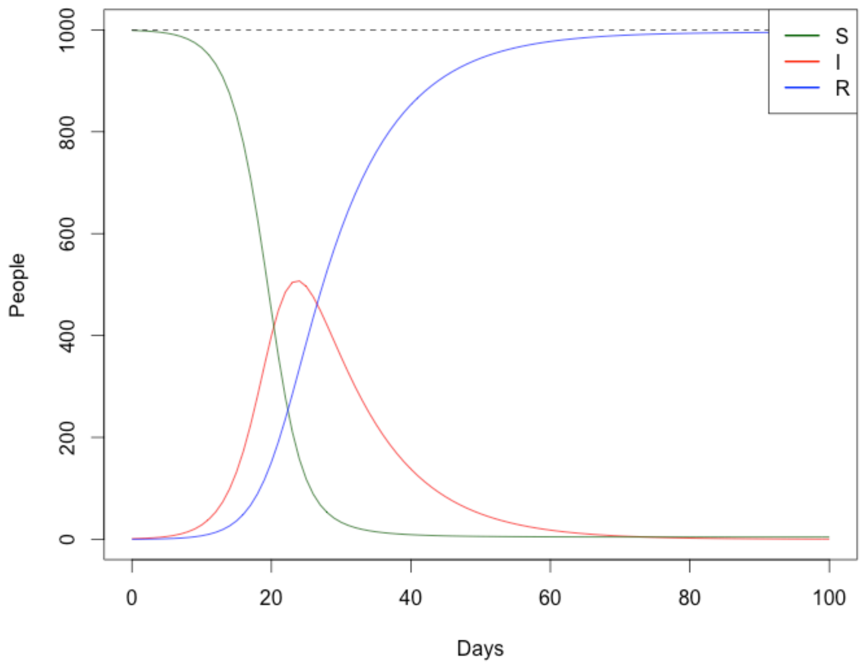
Now that we have some model results, let's do some data visualization. Plotting in R is done by creating an initial plot with some basic bounds such as the axis labels, minimum values of the x and y axis, etc. and then adding additional lines, points, annotations, legends, etc. to that basic canvas.

```
#Plot I against Time, then add lines for other categories and a
legend
plot(I~Time, data=output, type="l", col="Red", ylim=c(0,1000),
 xlab="Days", ylab="People")
lines(S~Time, data=output, type="l", col="Darkgreen")
lines(R~Time, data=output, type="l", col="Blue")
lines(N~Time, data=output, type="l", col="Black", lty=2)
legend("topright", c("S","I","R"), lwd=2,
col=c("darkgreen","red","blue"), lty=c(1,1,1), bty='y')
```

Here we plotted I against Time along with specifying that the upper and lower bounds of the y-axis should be between 0 and 1000, and then added additional lines for S, R and the total population (a useful model diagnostic). When drawing the lines, *type*="l" specifies a line (p specifies points, s specifies a step-function, etc.), *col="Colorname"* specifies what color the line should be, *lty*=2 specifies a dashed line (the others use the default of 1 to draw solid lines) and *lwd=2* specifies a line thickness of 2.

Finally, we added a label in the top right of the plot. Note that the label doesn't automatically reflect what we've already plotted – we had to tell it what to name the lines, what color they should be, and what style they should be in.

You should have something like this:

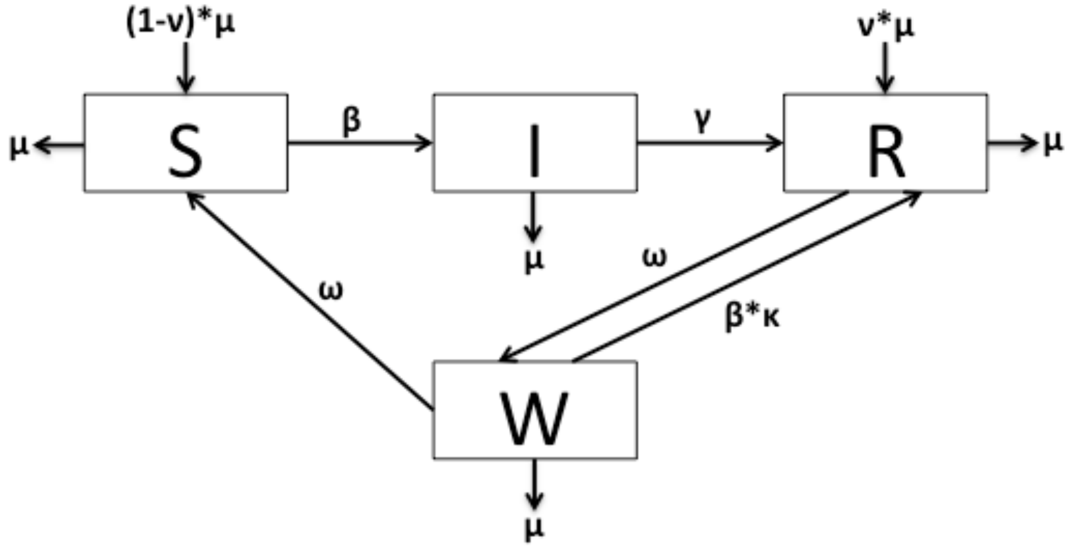


We've now implemented our first compartmental model.

Section 4: Worked Example – Pertussis

Let us now turn out attention to a slightly more complicated worked example of pertussis. Here we consider the effect of vaccination and waning natural immunity to pertussis based on a simplified version of the model presented in J.S. Lavine, A.A. King and O.N. Bjørnstad. Natural immune boosting in pertussis dynamics and the potential for long-term vaccine failure. *PNAS* (2011) vol. 108, no. 17: 7259-7264.

Specifically, we assume that individuals may be in one of four compartments – Susceptible, Infectious, Recovered and Waning. In this model, Recovered individuals have complete immunity to Pertussis from a recent infection or vaccination. Individuals in the Waning category still have immunity to the disease but may lose it over time, returning to Susceptible unless their immune system is challenged by a repeated exposure to the disease, whereby they move back to Recovered. The flow-diagram of this model is below, accompanied by a table of the parameters, their meanings, and values:



Parameter:	Value:	Meaning:
β	0.714	R_0 of ~ 15
γ	1/21	1 / Infectious Period of 21 days
ν	0.80	Infant Vaccine Coverage (%)
ω	0.000548	Waning period (1/2 of a 10-year duration of immunity)
κ	1	Relative efficacy of an infectious contact producing immunity (vs. producing infection)
μ	0.0000399	Birth and Death Rate (70 year life expectancy)

The corresponding difference equations for this system are:

$$S_t = S_{t-1} - \beta S_{t-1} I_{t-1} / N_{t-1} + \omega W_{t-1} + \left[\mu(1-\nu)\right] N_{t-1} - \mu S_{t-1}$$
$$I_t = I_{t-1} + \beta S_{t-1} I_{t-1} / N_{t-1} - \gamma I_{t-1} - \mu I_{t-1}$$
$$R_t = R_{t-1} + \gamma I_{t-1} + \beta \kappa W_{t-1} I_{t-1} / N_{t-1} - \omega R_{t-1} + \mu \nu N_{t-1} - \mu R_{t-1}$$
$$W_t = W_{t-1} + \omega R_{t-1} - \beta \kappa W_{t-1} I_{t-1} / N_{t-1} - \omega W_{t-1} - \mu W_{t-1}$$

Implementation:

The code for this model is available from the course's GitHub repository as *discrete_pertussis.r* or from the instructor. Once you have run the model and gotten results, consider the following scenarios by adjusting parameters one way or the other:

- What happens if you increase κ to 4? Decrease it to 0.25? What happens if you increase κ to 50 – and why is it happening?
- The proportion of susceptibles (P) who need to be vaccinated to eradicate disease in a standard SIR model is given P =1/1-R_0. For an R_0 of 15 that number is 0.933. Try setting ν to 0.933 – do we eradicate the disease? Is it possible to eradicate disease in this system? Is that answer true for different values for κ?
- Instead of plotting a compartment against time, try plotting two compartments (for example, I and S) against each other. What might this plot tell you?

Next Steps:

Though they are beyond the scope of this Learning Institute, for your convenience examples of the pertussis model we have been discussing implemented as a continuous series of differential equations, as well as a stochastic version of the model, have been uploaded to the GitHub site as a basis for further exploration. Additionally, versions of all the zombie apocalypse models have also been uploaded.