Workshop: Modeling disease spread

Jesse Brunner

2022-08-31

Why model?

Models, broadly defined, are useful in several ways. They help us:

- **formalizing thinking about how a system works**, whether with boxes and arrows or a complex representation of processes and variables
- **identifying data gaps** as one tries to structure and parameterize the model
- **understand consequences of model structure / assumptions** by *running* the model, examining counterfactuals, identify influential parameters or processes (e.g., with sensitivity analyses, comparing models with different structures formally), sorting among models based on how they match reality or fit data, and, finally,
- **make quantitative predictions or projections** about likely dynamics, outcomes/etc.

Our goals today, however, are a bit simpler. We will:

1. Describe a simple "compartment" model as boxes & arrows and as differential equations
2. Implement the model in R with the `simecol` package

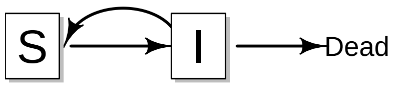
Part 1: Model structure

We often formulate mathematical models to help us understand what the forces that influence some quantity of interest. We call the things we want to change through time *"State" variables* and the numbers we determine (e.g., measure, or set from first principles or curiosity. . . or the "knobs" of the model) the *parameters*. The way the model is set up its *structure*, which is usually made up of *terms* that are the pieces, like words in a sentence.

Today, we are interested in describing the dynamics and outcome of an epidemic in a closed population. We will thus **focus on** the *fractions* of the hosts population that are in the susceptible class (S) or the infected class (I)[1] and track the flow of these fractions of the population between classes We will **assume** that individuals within

[1] the state variables

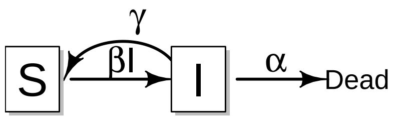
a class (S or I) are identical and transmission occurs by direct contact. For the sake of simplicity, we will **ignore** immunity (it does not seem to happen in tadpoles, anyway, and that is mostly what I think about. . .) and dynamics other than infection, recovery to susceptibility, and death due to infection. That is, no immigration, emigration, birth, or death not due to disease.



We can describe the overall forces or flows as follows:

Susceptible individuals become **infected** and infected animals either **die** or **recover** to S class (remember, no immunity?). Since we are ignoring many other forces (e.g., demographics, immunity), that is it! That is the whole description of the model. We can also draw this model as boxes (compartments) and arrows (flows). Indeed, this is usually a good place to start.

Now, let us think about the rates a bit. How does a susceptible individual become infected? It requires contact with an infected, so the term should include I in it. If infectious contacts occurs randomly at some rate, β, that increases with host density, then the per capita rate at which the susceptibles become infected is βI. Meanwhile, infected animals either die at some per capita rate, α, or recover to the susceptible class at per capita rate, γ.



Writing ODEs

We can now write this box-and-arrow model in terms of ordinary differential equations (ODEs). These equations describe the *rate* at which we gain/lose S and I through the processes we outlined above.

$$\frac{dS}{dt} = -\beta SI + \gamma I \tag{1}$$

$$\frac{dI}{dt} = \beta SI - \gamma I - \alpha I \tag{2}$$

Notice that we lose population from the S compartment through transmission, the overall rate of which is βSI, which is then added to the infected compartment. The reverse happens with recovery (rate γI) because those that recover are still susceptible, at least when thinking of tadpoles. Lastly, we lose I due to disease-induced mortality (=virulence). If we add up the two equations all terms cancels out except $-\alpha I$, which means that the population will shrink (and only shrink) because of the disease.

It is worth noting that the process of writing out ODEs is fairly simple. That is, if you can thin of the box and arrows, you can have at least a reasonable start on writing out equations. (Choose whatever letters you like!) However, while this is a fairly simple process, and ODEs are a very powerful set of tools that have been instrumental in disease ecology, public health, and so on, this relative easy means that you can easily make a model that you have no hope of understanding or parameterizing. This brings me to one of my favorite quotes:

> A model does nothing but expose the consequences of the assumptions upon which it is based. — Schauber & Ostfeld RS (2002) Ecol Appl 12:1142-1162

So think about what you want out of your model, what it can (and can't) do for you, what data you have, and so on before you get too far in the weeds. Also, start simple and *slowly* add complexity as needed.

Implementing our model in R

We will use a package called `simecol` to both organize our model and associated stuff (e.g., parameters, initial conditions) into a single "object" and solve the model for us. It follows a standard structure with clear "slots" for each bit and, so long as you get this initial stuff entered in correctly, you don't need to sweat most of the details. For more information, see the paper describing this package[2] and a "how to" guide[3].

[2] See the vignette: `vignette("a-simecol-introduction")`

[3] `vignette("b-simecol-howtos")`

We will also use some functions in the `tidyverse` to deal with the data output from model runs.

```
library(simecol) # for running model
library(tidyverse) # for data wrangling & plotting
```

Building up the model

The basic structure of a `simecol` model of ODEs is:

```
SI <- odeModel(
          main = function (time, init, parms, ...) {
            ### The actual model will go here
          },
          parms = c(),
          times = c(),
          init = c(),
          solver = "rk4"
)
```

Notice that we have assigned this basic model to a variable called `SI` (i.e., named it `SI`).

Next, we need to write out the actual model. This will be the differential equations we wrote out, above. However, we need the equations/code to "know" what the variables and parameters are by name, so we need to (1) "unpack" the variables found in the `init` vector and (2) wrap the model equations in the `with()` function.

```
SI <- odeModel(
          main = function (time, init, parms, ...) {
            # (1) unpack variables
            S <- init["S"]
            I <- init["I"]

            # (2) everything in the with() function will
            # know what the names are in parms
            with(as.list(parms),  {
              ### The actual model will go here
            })
          },
          parms = c(),
          times = c(),
          init = c(),
          solver = "rk4"
)
```

Next we can (3) write out those differential equations and (4) specify that we want the model to return the values of the calculated equations.

```
SI <- odeModel(
          main = function (time, init, parms, ...) {
            # (1) unpack variables
            S <- init["S"]
            I <- init["I"]

            # (2) everything in the with() function will
            # know what the names are in parms
            with(as.list(parms),  {
              # (3) transmission  #recovery   #virulence
              dS <- -beta*S*I       + gamma*I
              dI <-  beta*S*I       - gamma*I - alpha*I
              #(4) return the calculated variables
              list(c(dS, dI))
            })
          },
          parms = c(),
          times = c(),
          init = c(),
          solver = "rk4"
)
```

Lastly, we want to (5) define the parameters, (6) define the time series over which to solve the equations, and (7) provide some initial values for the variables.

Notice that dS and dI are just the results of the calculations on the right hand sides of the <- given the values of the parameters beta, gamma, and alpha, and the current values of the variables S and I. What simecol does is use some (hidden) functions to do the calculations for small chunks of time to get the changes in the variables *S* and *I*, and then add those changes to the prior values of *S* and *I*. This is Euler's method. (Or you can just ignore it for the time being...)

```
SI <- odeModel(
          main = function (time, init, parms, ...) {
            # (1) unpack variables
            S <- init["S"]
            I <- init["I"]

            # (2) everything in the with() function will
            # know what the names are in parms
            with(as.list(parms),  {
              # (3) transmission  #recovery   #virulence
              dS <- -beta*S*I       + gamma*I
              dI <-  beta*S*I       - gamma*I - alpha*I
              #(4) return the calculated variables
              list(c(dS, dI))
            })
          },
          parms = c(beta=0.0005, gamma=0.02, alpha=0.05),
          times = c(from=1, to=90, by=1),
          init = c(S=500, I=1),
          solver = "rk4"
)
```

Thats the whole model!

Now to run the model using the `sim()` function. It's a bit odd, but we can just assign this back to the same model.

What is actually happening is that one of the many "slots" of the odeModel object is being updated when we simulate the results. Try typing str(SI) to see these slots. The one we will be updating is out.

```
# simulate or "run" the model
SI <- sim(SI)
```

The results are held in a slot called `out`. There are “accessor” functions with the same names as the slots. So we can see the output using the `out()` accessor function.

```
head( out(SI) )
```

```
##   time        S        I
## 1     1 500.0000 1.000000
## 2     2 499.7481 1.197143
## 3     3 499.4467 1.432953
## 4     4 499.0862 1.714929
## 5     5 498.6552 2.051988
## 6     6 498.1402 2.454714
```

```
tail( out(SI) )
```

```
##    time        S        I
## 85   85 47.76768 45.10660
## 86   86 47.59834 43.07185
## 87   87 47.44010 41.12552
## 88   88 47.29209 39.26413
## 89   89 47.15354 37.48431
## 90   90 47.02372 35.78277
```

Of course looking at raw data is a little unsatisfying. Let us plot the results. If you are used to the tidyverse, this code will be meaningful. If not, you may want to just copy-paste it in and not worry about it for the moment.

```
# get the output from the model
out(SI) %>%
  # turn it from wide to long
  gather(key="Box", value="Number", S, I) %>%
  # construct the plot
  ggplot(., aes(x=time, y=Number, color=Box)) +
  geom_line()
```

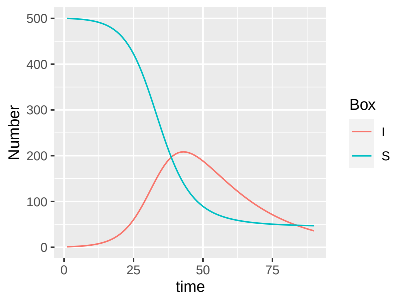


Figure 1: Plot of the the epidemic dynamics in our SI model.

Results! It looks like an epidemic happened! Now it is time to start tweaking and seeing what happens.

Tweaking parameters, variables, and time series

The combination of modular `odeModel` and accessor functions makes it fairly easy to tweak parameters, parameters, and the time series. For instance, let us update the parameter `beta` and see how the dynamics look.

While the core of the model is available with main(SI), I would recommend just creating a new model if you want to change the model structure.

```
parms(SI)
```

```
##  beta gamma alpha
## 5e-04 2e-02 5e-02
```

```
parms(SI)["beta"] <- 0.001 # twice as high
```

```
out(sim(SI)) %>%
  gather(key="Box", value="Number", S, I) %>%
  ggplot(., aes(x=time, y=Number, color=Box)) +
  geom_line()
```

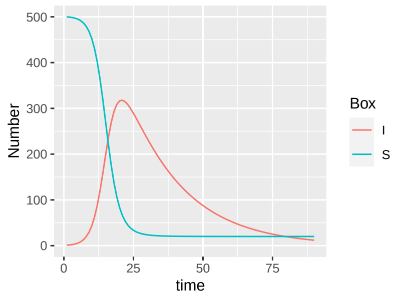


Figure 2: Updated dynamics with $\beta = 0.001$.

Or, similarly, if we start the model with a larger initial number of infecteds and for a longer time

```
init(SI)
```

```
##   S   I
## 500   1
```

```
init(SI)["I"] <- 10
times(SI)
```

```
## from   to   by
##    1   90    1
```

```
times(SI)["to"] <- 150
```

```
out(sim(SI)) %>%
  gather(key="Box", value="Number", S, I) %>%
  ggplot(., aes(x=time, y=Number, color=Box)) +
  geom_line()
```

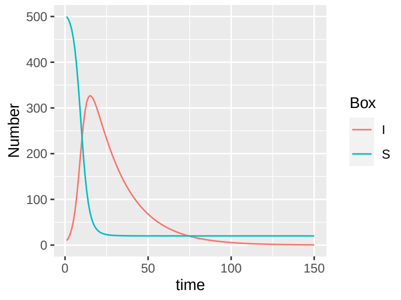


Figure 3: Updated dynamics with $I_0 = 10$ run through $t = 150$.

Creating composite variables

It can also be useful to calculate and plot other variables such as the total population size ($N = S + I$) or prevalence (I/N). The easiest way to do this is to create variables with the `mutate()` function after the fact.

```
out(sim(SI)) %>%
  # create a new variable for N
  mutate(N = S+I) %>%
  gather(key="Box", value="Number", S, I, N) %>%
  ggplot(., aes(x=time, y=Number, color=Box)) +
  geom_line()
```

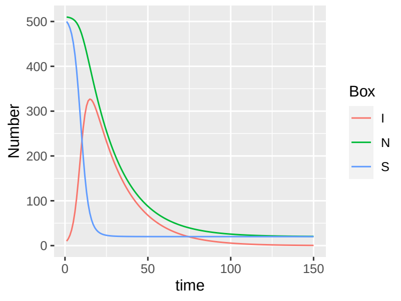


Figure 4: Updated model showing the dynamics of the whole population, $N = S + I$.

```
out(sim(SI)) %>%
  # create a new variable for N
  # then use it to calculate prevalence
  mutate(N = S+I,
         Prevalence = I/N) %>%
  ggplot(., aes(x=time, y=Prevalence)) +
  geom_line()
```

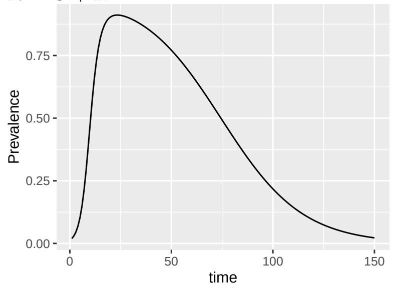


Figure 5: Updated model showing prevalence (=I/N) instead of I and S.

Next steps and advice

We have just begun to scratch the surface of epidemic modeling. As you progress you will note that there is a great deal of "art" to modeling, and experience and context are both crucial to doing it well. But to get you started thinking about these things, let me offer some interesting issues to consider.

Transmission is at the heart of epidemics, but getting this right for a particular system is very difficult. One key issue is which functional form to use. We have used density-dependent term, βIS, where transmission rates increase with the density of infected and susceptible hosts. With this form of transmission, there can be a threshold host density below which transmission is so slow that infections do not replace themselves. However, if we assume that hosts make contacts with other hosts at a constant rate (e.g., X/day), then we need only worry about the fraction of those contacts that are with infected hosts. The transmission term changes to $\beta IS/N$ (often called "frequency-dependent", but maybe density-independent is better) and the threshold host disappears. Empirical estimates of transmission usually suggest some hybrid where transmission scales with density at low densities and then levels off. Anyway, if you try different functional forms you will get different results... at least when host density changes. For more, see Begon et al. 2002[4], McCallum et al. 2001[5], and McCallum et al. 2017[6].

[4] Begon, M., M. Bennett, R. G. Bowers, N. P. French, S. M. Hazel, and J. Turner. 2002. A clarification of transmission terms in host-microparasite models: Numbers, densities and areas. Epidemiology and Infection 129:147-153.

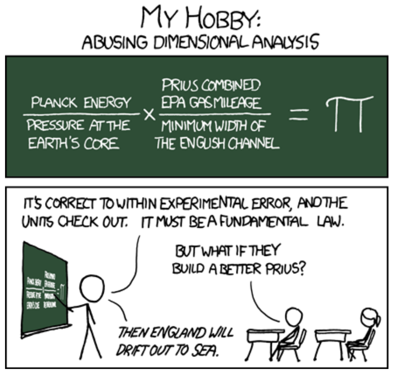
[5] McCallum, H., N. Barlow, and J. Hone. 2001. How should pathogen transmission be modelled? Trends in Ecology & Evolution 16:295-300.

[6] McCallum, H., A. Fenton, P. J. Hudson, B. Lee, B. Levick, R. Norman, S. E. Perkins, M. Viney, A. J. Wilson, and J. Lello. 2017. Breaking beta: deconstructing the parasite transmission function. Philos Trans R Soc Lond B Biol Sci 372

We have been sweeping this under the rug, but are we modeling the number of hosts or the density of hosts? It is important to think through this when trying to fit your model to actual data. The choice will also influence how you interpret your transmission terms. See Begon et al. 2002 for more.

When trying to add complexity to your model or connecting it with real-world parameters, it can be very helpful to write out the units of your terms. We are using *equations* and so the units on the left and right sides of the equal signs must be equal if everything is correct. For instance, if S and I are in units of numbers of hosts, then β must have units of host^{-1}time^{-1} in order for βSI to produce number $\times$ time^{-1}, which is what is on the left-hand side.

But do beware: just because your units work out does not mean you have the right answer!


MY HOBBY:
ABUSING DIMENSIONAL ANALYSIS
PLANCK ENERGY
PRESSURE AT THE EARTH'S CORE
PRIUS COMBINED EPA GAS MILEAGE
MINIMUM WIDTH OF THE ENGLISH CHANNEL
= π
IT'S CORRECT TO WITHIN EXPERIMENTAL ERROR, AND THE UNITS CHECK OUT. IT MUST BE A FUNDAMENTAL LAW.
BUT WHAT IF THEY BUILD A BETTER PRIUS?
THEN ENGLAND WILL DRIFT OUT TO SEA.

We have used constant rates for the transitions from one stage to another. This is simple and mathematically tractable. However, this also implies an exponential distribution of times in each category

(=waiting time or residence time). Let us illustrate this by tracking a cohort of 100 I as they die. We will set $I_0 = 100$, $\alpha = 0.05$, and $S_0 = 0$.

```
rates1 <- odeModel(
  main = function (time, init, parms, ...) {
            # unpack variables
            S <- init["S"]
            I <- init["I"]

            with(as.list(parms),  {
              #transmission      #virulence
              dS <- -beta*S*I
              dI <-  beta*S*I - alpha*I

              list(c(dS, dI))
            })
          },
  parms = c(beta= 0.0005, alpha= 0.05),
  times = c(from=1, to=90, by=1),
  init = c(S=0, I=100),
  solver = "rk4"
)

out(sim(rates1)) %>%
  gather(key="Box", value="Number", S, I) %>%
  ggplot(., aes(x=time, y=Number, color=Box)) +
  geom_line() +
  geom_vline(xintercept = 1/0.05)
```

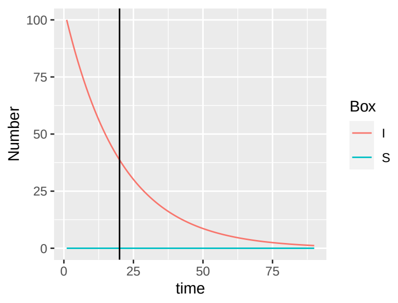


Figure 6: Exponential waiting times in the infected class

First, notice that the line for I follows an exponential decline. While most infecteds die rapidly, in fact they start dying *instantly*, some survive out to the end of our time series. The average time to death is $1/\alpha = 1/0.05 = 20$, although since there is such a long right tail, the median time to death is less than 20.

Constant rates are nice and simple mathematically, the resulting assumption of exponentially distributed waiting times in a compartment often does not fit biological reality. For instance, individuals do not become instantly infectious after becoming infected themselves. If this delay is short relative to the dynamics of the infection it would be quite reasonable to ignore the delay in favor of simplicity, but sometimes these details matter a great deal. In this case, we might add an Exposed-but-not-yet-infectious class (for an SEI model) to provide a slight delay, on average, before individuals become infectious.

We can take this approach even further. For instance, to get our survival time curve to look a bit more realistic, we can break up the I class into multiple compartments and have them flow one into the next (i.e., $I_1 \rightarrow I_2, I_2 \rightarrow I_3, \ldots, I_{n-1} \rightarrow I_n, I_n \rightarrow$ death). Let us try $n = 5$. Since individuals must now go through five compartments to get to death, we need to increase the rate at which they pass through them to get the same average time to death. So $\alpha_{new} = n \times \alpha = 5 \times 0.05$.

```
rates2 <- odeModel(
  main = function (time, init, parms, ...) {
            # unpack variables
            S <- init["S"]
            I1 <- init["I1"]
            I2 <- init["I2"]
            I3 <- init["I3"]
            I4 <- init["I4"]
            I5 <- init["I5"]

            with(as.list(parms),  {
              #transmission           #virulence
              dS  <- -beta*S*(I1+I2+I3+I4+I5)
              dI1 <-  beta*S*(I1+I2+I3+I4+I5) - alpha*I1
              dI2 <-  alpha*I1      - alpha*I2
              dI3 <-  alpha*I2      - alpha*I3
              dI4 <-  alpha*I3      - alpha*I4
              dI5 <-  alpha*I4      - alpha*I5

              list(c(dS, dI1, dI2, dI3, dI4, dI5))
            })
          },
  parms = c(beta= 0.0005, alpha= 0.05*5),
  times = c(from=1, to=90, by=1),
  init = c(S=0, I1=100, I2=0, I3=0, I4=0, I5=0),
  solver = "rk4"
)

out(sim(rates2)) %>%
  # make new variable with the sume of all of the Ix columns
  mutate(I=rowSums(select(., contains("I", ignore.case = FALSE))
  gather(key="Box", value="Number", -time) %>%
  ggplot(., aes(x=time, y=Number, color=Box)) +
  geom_line() +
  geom_vline(xintercept = 1/0.05)
```

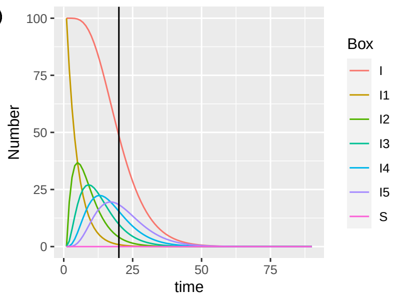


Figure 7: Using five sequential infected classes to create a gamma distribution of waiting times in the infected class.

You can see the way the population flows through the five I_x compartments and the whole lot of them in the $I = \sum I_x$ class (red line). Notice that this gives a more realistic distribution of times to death; they do not start dying in earnest until $t \approx 5$ and everyone is dead by $t = 50$. If you add more compartments, the variance in residence time will decrease. Indeed, what we have created is a gamma distribution of residence times with a mean of α and a variance of $([1/\alpha]^2)/n$.