Metcalf.TeachingE2M2

Jess Metcalf

10/24/2016

Population Growth

The simplest formulation for population growth (in **discrete time**) is $N_{t+1} = \lambda N_t$ where N_t is the size of the population at time t. The World Bank provides time-series of population size for all countries in the world, including for Madagascar; this is in the file "WorldBankPop.csv". Keep in mind that this is **not** data from direct observations - censuses have not occurred in all the years for which data is available!

Bring in this data by identifying the right path to the folder you are working in (use the command 'getwd' to find out where you are, and then use 'setwd' to get R to look in the right place) and have a look at its structure in R (just the first 10 columns):

```
setwd("/Users/jessicametcalf/Dropbox/E2M2.Metcalf.Lectures/Rcode/")
pop.data <- read.csv("WorldBankPop.csv")
head(pop.data[,1:10])
```

```
##   Country.Name Country.Code    X1960    X1961    X1962     X1963     X1964
## 1        Aruba          ABW    54208    55435    56226     56697     57029
## 2      Andorra          AND    13414    14376    15376     16410     17470
## 3  Afghanistan          AFG  8994793  9164945  9343772   9531555   9728645
## 4       Angola          AGO  5270844  5367287  5465905   5565808   5665701
## 5      Albania          ALB  1608800  1659800  1711319   1762621   1814135
## 6   Arab World          ARB 92540534 95077992 97711191 100439395 103263656
##       X1965     X1966     X1967
## 1     57360     57712     58049
## 2     18551     19646     20755
## 3   9935358  10148841  10368600
## 4   5765025   5863568   5962831
## 5   1864791   1914573   1965598
## 6 106184090 109210743 112342573
```

We can pull out the row corresponding to Madagascar:

```
chs <- which(pop.data[,1]=="Madagascar",arr.ind=TRUE)
mada.pop <- as.numeric(pop.data[chs,3:ncol(pop.data)])
```

The command 'which' finds the right row, which we baptize 'chs'. We plug that into the matrix by putting 'chs' into the row index (the first value after the "["; the second value is the column). To get the population of Madagascar from 1960 to 2016, we select the 3rd column to the end of the columns - the results from the command 'head' above shows why (look at the names of the 1st, 2nd and 3rd column to see why). We can get one other country for comparison:

```
chs <- which(pop.data[,1]=="France",arr.ind=TRUE)
france.pop <- as.numeric(pop.data[chs,3:ncol(pop.data)])
```

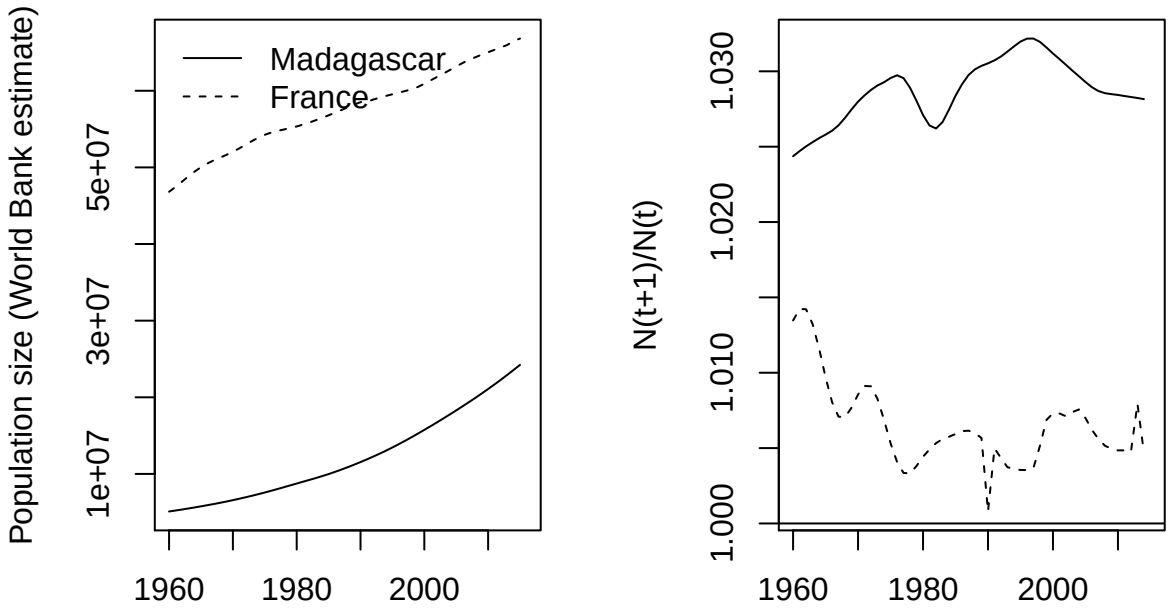
We can calculate the growth rate, N_{t+1}/N_t, by taking a vector that goes from the 2nd until the last column (this is years 1961 until 2016, and thus we call it N_{t+1}). We will divide this by the the vector that goes from

the first until the before-last column, which will correspond to 1960 to 2015. Relative to the first vector, this is N_t - i.e., the first number in the N_{t+1} vector is 1961; and the first number in the N_t vector is 1960, and so on.

```
growth.mada.pop <- mada.pop[2:length(mada.pop)]/mada.pop[1:(length(mada.pop)-1)]
growth.france.pop <- france.pop[2:length(france.pop)]/france.pop[1:(length(france.pop)-1)]
```

and we can then plot these two things:

```
par(mfrow=c(1,2)) #create a matrix of plots with 1 row and 2 cols
plot(1960:2016,mada.pop, type="l",xlab="",  #plot mada pop data
     ylab="Population size (World Bank estimate)",
     ylim=range(c(mada.pop,france.pop),na.rm=TRUE))  #define limits of the y axis
points(1960:2016,france.pop, type="l",lty=2) #plot France pop data
legend("topleft",legend=c("Madagascar","France"),lty=1:2,bty="n")

plot(1960:2015,growth.mada.pop, type="l",xlab="", ylab="N(t+1)/N(t)",
     ylim=range(c(growth.mada.pop,growth.france.pop),na.rm=TRUE))
points(1960:2015,growth.france.pop, type="l",lty=2)
abline(h=1)
```



The growth rates for Madagascar and France never drop below 1 (although France comes close), so these populations have been persistently growing. To predict the future, we could take the last value of population size that we have (2016) - and multiply it by the estimated most recent growth rate (which could be accessed by doing "growth.mada.pop[length(growth.mada.pop)]" as the command "length" tells us how long the vector is). This is, of course, **a very simple** model of population growth! We could add age-structure, spatial variation, variation in birth rates, mortality rates, etc, to try and make a more precise prediction.

Things to try:

- Change the colour of the plots and the legend;
- Experiment with plotting out different countries (Germany? Tanzania?)

- See if you can find which country in the entire data-base has the fastest growth, and when that occurred. You might want to use the 'which' command, with the argument 'arr.ind=TRUE' since this will index the matrix, i.e., tell you the row (which corresponds to the country) and the column (which corresponds to the year). Note that you will need to apply this to a matrix of growth rates!

Continuous vs. discrete time

The model of population growth introduced above is framed in discrete time, i.e., the population is projected from one year to the next, without considering time-steps in between. To understand discrete time models better, we are going to consider an example where the population growth is **constant**, i.e. is the same every year. If we assume that the population growth rate is constant, i.e., say, $N_{t+1}/N_t = \lambda = 1.1$, and the population starts with 10 individuals, we can project the population forward for 100 time-steps using the fact that $N_{t+1} = \lambda N_t$. This can be done using a **loop** in R. First, we set up an empty vector to contain the population size at each time-step, and we set the first value to 10, and define the growth rate, λ:

```
N <- rep(NA,100) #create a vector for the population
N[1] <- 10       #set N(0) to be 10, as described
lambda <- 1.1    #set lambda, as described
```

By using the command "for" we can loop over values of t from 2 to 100. For each value of t, we calculate the N_t for that time-step based on the last time-step, and then store it in our matrix:

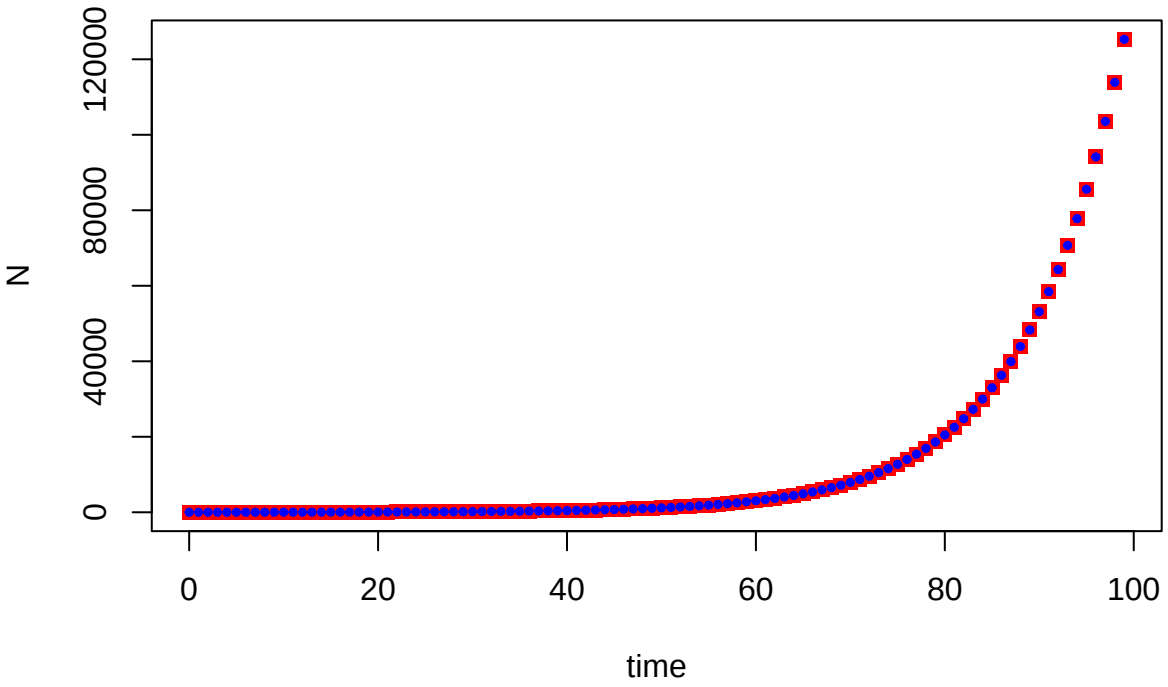
```
#loop t from 2 to 100, noting that t=1 corresponds to N(0)
for (t in 2:100) N[t] <- N[t-1]*lambda
```

Alternatively, we can use the solution that $N(t) = \lambda^t N(0)$ (see the lecture):

```
Ndirect <- N[1]*lambda^(0:99)
```

Again, noting that the first value in the vector N[1] corresponds to t=0, we can plot the two estimates for comparison, here with red squares for N, and small blue points for Ndirect (the size of the points is set by the command 'cex', and making the 2^{nd} set of points smaller is necessary to be able to see both):

```
plot(0:99,N, pch=15,col="red", xlab="time", ylab="N")
points(0:99,Ndirect,pch=19,col="blue", cex=0.5)
```



These match perfectly, which is reassuring. If, instead, we choose to work with **continuous time**, we need to use a differential equation, $dP(t)/dt = rP$. This is equivalent to framing the growth of the population in continuous time, with an instantaneous rate of change r of P, the population size (here, I'm using P just to distinguish from the discrete time estimate, N; and again, we're assuming that the growth rate of the population is not changing through time). We can write out a function that defines this differential equation in R:

```
pgrow <- function(t,y,parms){
  # The with() function gives access to the named values of parms within the
  # local environment created by the function
  with(c(as.list(y),parms),{
    dPdt <- r*P
    list(c(dPdt))
  })
}
```

To solve this, we're going to need to use the R library called "deSolve". We also need to define the starting variables and parameters:

```
library(deSolve)              # Load libary to be used for numerical integration
```

```
##
## Attaching package: 'deSolve'
```

```
## The following object is masked from 'package:graphics':
##
##     matplot
```

```
Pop <- c(P=10)                # Define starting variables (P(0)=10, as above)
values <- c(r=log(lambda))    # Define parameters - use an approx of the lambda used in the discrete ti
time.out <- seq(0,100,by=0.1) # Set up time-steps for variables - here go for 100 years
pgrow(t=0,y=Pop,parms=values)  #try out the function for t=0
```

```
## [[1]]
## [1] 0.9531018
```

The function outputs the time derivative of P at the time t. This means that in order to get the (approximate) values of P at some time delta.t in the future, we need to calculate the equivalent of P(t+delta.t)=P(t)+rP(t)*delta.t. We can do this by hand:

```
delta.t <- 0.1                    # Set a small value for delta.t (0.1 day)
pop.next <- Pop + unlist(pgrow(t=time,y=Pop,parms=values)) * delta.t
pop.next
```

```
##        P
## 10.09531
```

We could iterate this process, updating our values of Pop and time with each iteration to get a discrete-time approximation of the values of P through time. However, differential equations describe the rates of change **in the limit as delta.t goes to zero**. It turns out that using the discrete time approximation of this process leads to rapid acummulation of error in our estimate of the state variables through time, even if we set our value of delta.t to be very small. Fortunately, a number of algorithms have been developed to come up with better (both more accurate and more computationally efficient) approximations to the values of the state variables through time. One such is "lsoda" which we loaded above, and which we can run, here storing the result in a variable called 'Ncontinuous':

```
Ncontinuous <- lsoda(
  y = Pop,                # Initial conditions for population
  times = time.out,       # Timepoints for evaluation
  func = pgrow,           # Function to evaluate
  parms = values          # Vector of parameters
  )
```

We can check the first few lines of the output:

```
head(Ncontinuous)
```

```
##      time        P
## [1,]  0.0 10.00000
## [2,]  0.1 10.09577
## [3,]  0.2 10.19245
## [4,]  0.3 10.29006
## [5,]  0.4 10.38860
## [6,]  0.5 10.48809
```

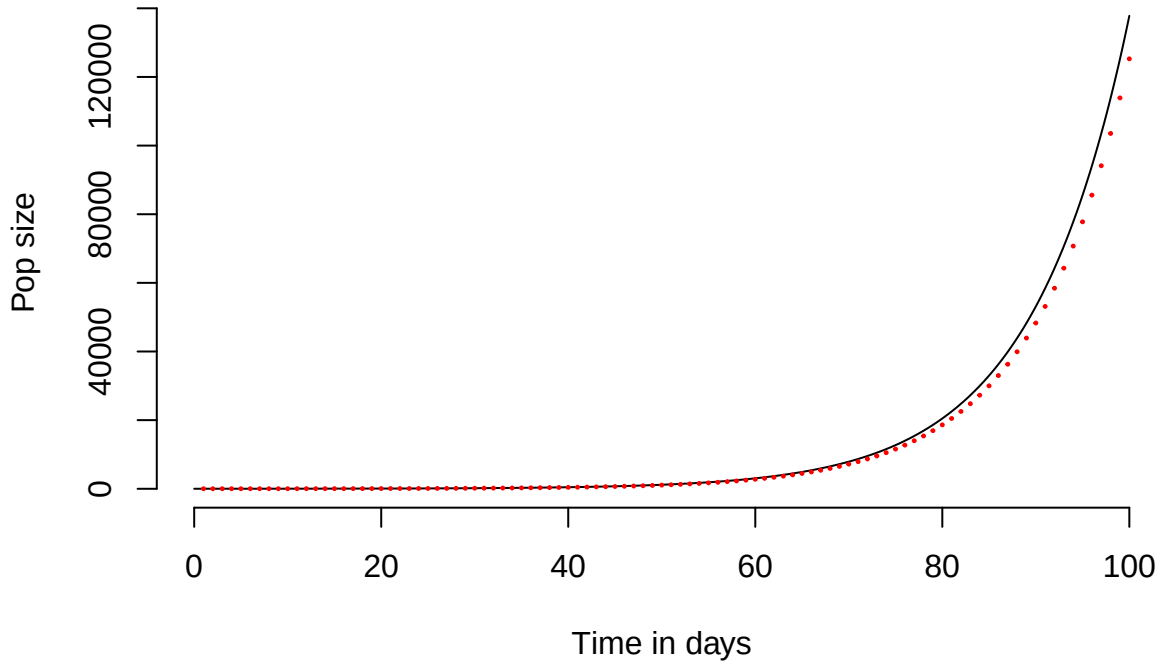
The data frame has 2 columns showing the values of time (labeled "time") and the state variable ("P", which is population size) through time. Let's see what has happened after 2 years, by printing it out:

```
subset(Ncontinuous,time==2)
```

```
##    P
## 12.1
```

We can also plot the full time-series:

```
plot(Ncontinuous[,"time"],                # Time on the x axis
     Ncontinuous[,"P"],                   # Number infected (I) on the y axis
     xlab = "Time in days",      # Label the x axis
     ylab = "Pop size",          # Label the y axis
     main = " ",                 # Plot title
     xlim = c(0,100),            #
     type = "l",                 # Use a line plot
     bty = "n")                  # Remove the box around the plot
points(1:100,N, pch=19,col="red",cex=0.2)  #compare with our discrete time
```



Note that we overlaid our discrete time model predictions (the variable N) for comparison. This is close, but not an exact match to our discrete time version, which makes sense, since these are different processes.

Things to try:

- See what happens to a population where $r < 0$
- Change the time resolution (set by "time.out"") and see if/where the results change

Determinstic vs. stochastic

We can also make a distinction between **deterministic** and **stochastic** models. Imagine that we can break down the λ above into a birth rate $b = 0.8$ and a probability of survival $s = 0.2$. For now, let's continue to assume that all individuals are identical, so that we have: $N_{t+1} = (s + b)N_t$, i.e., the number of individuals one time-step in the future is the sum of the proportion that survived, sN_t and the number of offspring they produced, bN_t (note that we are ignoring important biological facts like the fact that not everyone in the pouplation can reproduce!). As above, we can construct a loop to explore the growth of the population from a starting population of size $N(0) = 10$.

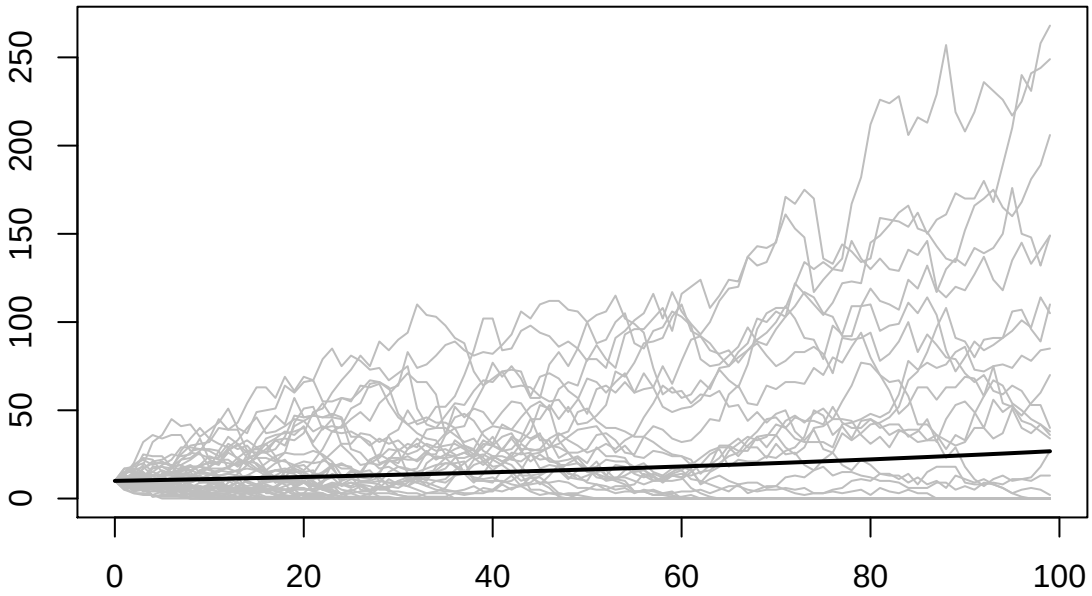
```
N <- rep(NA,100)                                  #create a vector for the population
N[1] <- 10                                        #set N(0) to be 10, as described
b <- 0.8; s=0.21                                  #set b and r as described
for (t in 2:100) N[t] <- N[t-1]*b+ N[t-1]*s #loop, noting that t=1 corresponds to N(0)
```

This is the **deterministic** version of this model, i.e., if we know the number of individuals at t then we know how many survived, and how many offpsring were born at $t+1$, **exactly**. Note that this may mean having fractional results, i.e., we could end up with one and a half (1.5) individuals, which is, of course, not very realistic! We can also construct the **stochastic** version - assuming that survival follows a binomial distribution with probability s (remember normal distributions! this is the distribution that makes sense for data that comes in 0s and 1s); and reproduction reflects a poisson distribution with mean set to b (this is a natural choice for the distribution of an integer). Since this is stochastic, every time we run the model is likely to be different. Therefore, we're going to do 50 different runs, using a loop over time and runs, and storing them in a matrix:

```
Nstoch <- matrix(NA,50,100) #create a matrix to store the results
#                           # of the simulation, every row is a different run
Nstoch[,1] <- 10            #set N(0) to be 10, for every run.
for (j in 1:50){            #loop over sims
for (t in 2:100) {          #loop over time
    Nstoch[j,t] <- rpois(1,Nstoch[j,t-1]*b)+ rbinom(1,Nstoch[j,t-1],s)
}}
```

and now we now can plot them for comparison:

```
matplot(0:99,t(Nstoch), xlab="", ylab="",
        type="l",lty=1,col="grey")  #'matplot' plots matrices; and extension of plot
points(0:99,N, type="l",lwd=2)
```



The determinstic trajectory (solid black line) is in the middle of the range of stochastic trajectories. If you look at the stochastic trajectories, you will also see that these are all **integers** (no fractions) because the stochastic models we chose (binomial and poisson) work on integers. As mentioned, this is more realistic when we are thinking about individuals.

A lot of these stochastic trajectories go **extinct**; many end up much **larger** than the deterministic trajectory. This results because we allow for the fact that if, for example, survival probability is 0.5, sometimes, by chance more than 50 percent of the individuals will survive, and sometimes fewer; and the same principal for fertility. This result is very important in conservation biology, as well as life history evolution.

Things to try:

- Make the population MUCH bigger, see if you can still see the results of stochascticity
- Create a figure which shows what fraction of time-series go extinct for different values of b
- Plot the mean across all the stochastic trajectories (taking the mean at each time-point) and compare it to the determinstic trajectory. You might want to use the function 'colMeans'

Structured population models

So far, we have ignored structure in the population, and treated all individuals as if they were identical. We know this is not true. The simplest extension is to assume some form of **stage structure**, i.e., different *states* within the population. Let us assume that the population has two stage classes, 'juveniles' and 'adults'. We can describe the population using a vector of numbers of the two types of individuals $n_t = (n_j, n_a)$ where the j index indicates juveniles, and the a index indicates adults, n indicates 'numbers'. In discrete time, projection from one time-step to another requires defining **juvenile survival**, **juvenile probability of aging into the adult class in each time-step**, **adult survival** and **adult fertility**.

```
sj=0.2  #juvenile survival
aj=0.8  #juvenile aging
sa=0.8  #adult survival
f=1     #adult fertility
```

These parameters can be estimated by tracking individuals in the field and counting the number that survive between censuses (here, distinguishing 'juvenile' and 'adult' stages), what number moved between stages, and the number of offspring produced at each stage. From this, we obtain survival and aging probabilities, and fertility. We place these parameters in a **matrix** which has as many rows and columns as we have stages. The columns define the stage that individuals start in, and the rows the stage that individuals end up in:

```
A <- matrix(0,2,2)
A[1,1] <- sj*(1-aj)  #juvenile to juvenile = not aging and surving as a juvenile
A[2,1] <- sj*aj      #juvenile to adult =  aging and surving as a juvenile
A[2,2] <- sa         #adult to adult =  surviving as an adult
A[1,2] <- f          #adult to juvenile = survivng and reproducing
```

We can project the population forward by one time-step:

```
n.start <- c(5,10)    #start with pop of 5 juveniles and 10 adults
n.next <- A%*%n.start #get pop one time-step latter using matrix multiplication (%*%)
n.next                #more juveniles, but fewer adults...
```

```
##      [,1]
## [1,] 10.2
## [2,]  8.8
```

We could write a loop, and project the population into the future (by applying matrix multipicaton to A and n.next, and so on), and from that, figure out the growth rate of this population. There is also a shortcut, using some core results from **matrix population model theory** (Caswell, 2001): the population growth rate is defined by the dominant eigenvalue of the matrix. We can extract this with:

```
eigen(A)$value[1]      #extract the pop growth rate using the eigenvalue; <1 indicates pop shrinking
```

```
## [1] 0.9717246
```

```
A[1,2] <- f*2          #double the fertility
eigen(A)$value[1]      #extract the pop growth rate using the eigenvalue; now pop growing
```

```
## [1] 1.101469
```

There are many more powerful results that yield estimates of the stable population structure, the reproductive value, life expectancy, sensitivies, elasticities, etc, that have been used extensively in fields from conservation biology to life history evolution. You can also implement density dependence, etc.

Things to try:

- Iterate the population forwards in a loop and compare the growth rate obtained with the eigenvalue estimate of λ. You may find that the population gets unmanageably large very quickly (R can't handle very large numbers) - so one trick is to reset the popuation to sum to one after each iteration - it doesn't matter, since you're only interested in the change / increase from t to $t+1$.
- For some species, e.g., mosquitoes, the juvenile phase (or larval phase) could be sensitive to environmental drivers, like temperature, humidity, etc. Create a loop that changes survival of juveniles (or aging, which equates to development rate) in the matrix at each time-step based on such time-varying drivers (which you could build using sin to cpature seasonality, etc - note that this means that the time-step is < 1 year). Explore the consequences for total population size if temperature increases.
- You can download a large set of pre-digitised matrix population models from: http://www.compadre-db.org and then explore population growth, life expectancy, for 100s and 100s of species, etc. (see Salgeruo-Gomez et al. 2015)

```
# Code not run - if you have downloaded the Compadre data, then you can load it
#load("/Users/cmetcalf/Downloads/COMPADRE_v.4.0.1.RData")

# and then...

# names(compadre)           #check what it contains
# head(compadre$metadata)   #look at the metadata to see how its structured
                            #indexes here correspond to indexes in the list of mat
# compadre$mat[[1]]         #pull out the 1st set of matrices -
                            #in this list A is the full matrix;
#                           # U is only survival and growth,
                            # F is fertility, and C is clonal reproduction (if there is any)
# eigen(compadre$mat[[1]]$matA)$value[1]  #get the population growth rate for this species
#                                         #publication (defined by first row metadata)
# tmp <- which(compadre$metadata$Country=="MDG") #Find all the matrices relating to Madagascar
# compadre$metadata[tmp,]                        #Look at their metadata
# compadre$mat[[tmp[1]]]                         #If you want to manipulate in more depth - index...
```

Lotka-Volterra models of predator-prey dynamics

Moving beyond considering just one species, a classic dynamic is that of **predator-prey** cycles. To caricature the process, we can imagine that fossa eat lemurs, until there aren't enough lemurs. Then the fossa population crashes, which allows the lemur population to grow again, etc. We can frame this in continuous time with the following equation:

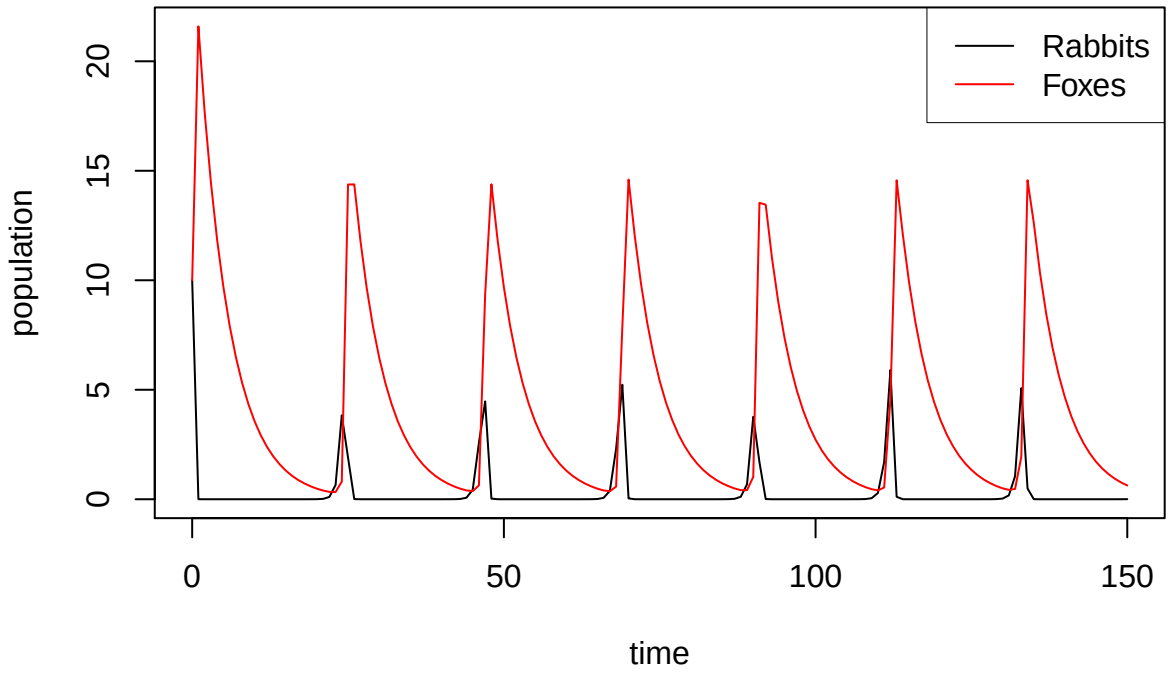
```
LotVmod <- function (Time, State, Pars) {
    with(as.list(c(State, Pars)), {
        dx = x*(alpha - beta*y)
        dy = -y*(gamma - delta*x)
        return(list(c(dx, dy)))
    })
}
```

where x reflects the prey population (hares, rabbits, lemurs, etc), and y reflects the predator population (wolves, foxes, fossa, etc). The prey population grows at a linear rate (α) and gets eaten by the predator population at the rate of (β) per predator. The predator gains a certain amount vitality by eating the prey at a rate (δ), while dying off at another rate (γ). Let's set some initial parameter values, and a starting population structure, and time-vector, as previously:

```
Pars <- c(alpha = 2, beta = .5, gamma = .2, delta = .6)
State <- c(x = 10, y = 10)
Time <- seq(0, 150, by = 1)
```

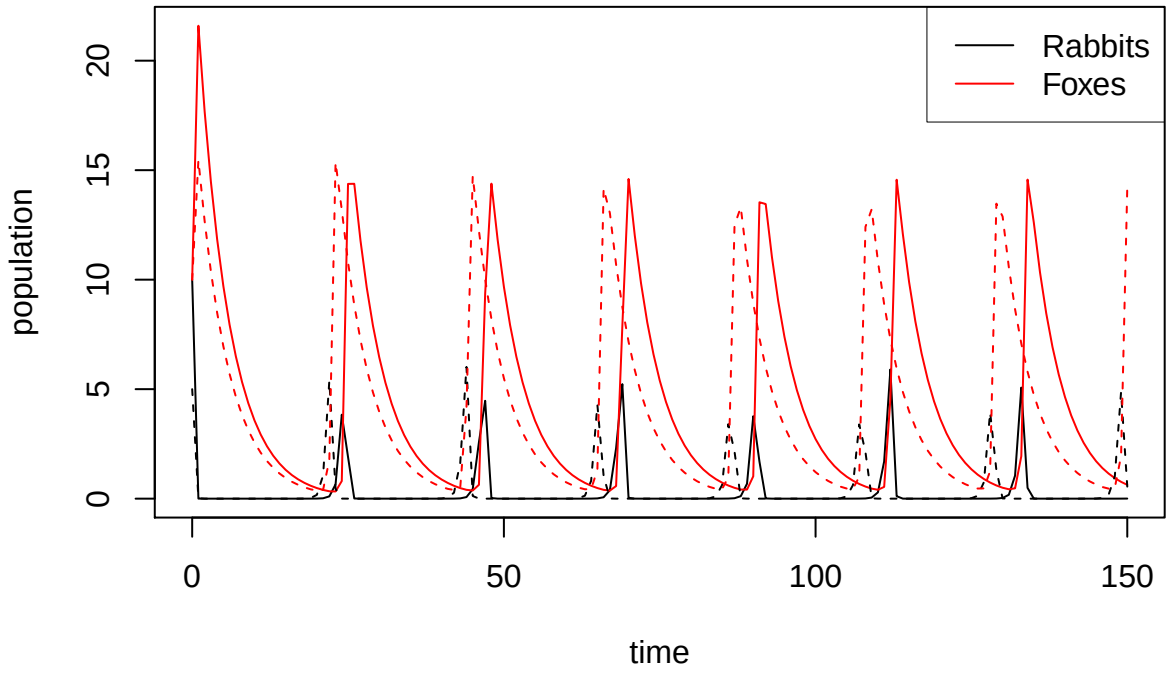
We run and plot this:

```
out <- as.data.frame(ode(func = LotVmod, y = State, parms = Pars, times = Time)) #run

matplot(out[,1],out[,-1], type = "l",     #separate first column (time), and plot against other two
        xlab = "time", ylab = "population", lty=1)
legend("topright", c("Rabbits", "Foxes"), lty = c(1,1), col = c(1,2), box.lwd = 0)
```



We can start with different starting numbers of rabbits and lynx, and overlay the two plots (using the command 'add=TRUE' in matplot). To distinguish the two runs, we use dashed lines for the second set of starting values using lty=2 (colours still represent predator and prey, as in the legend):

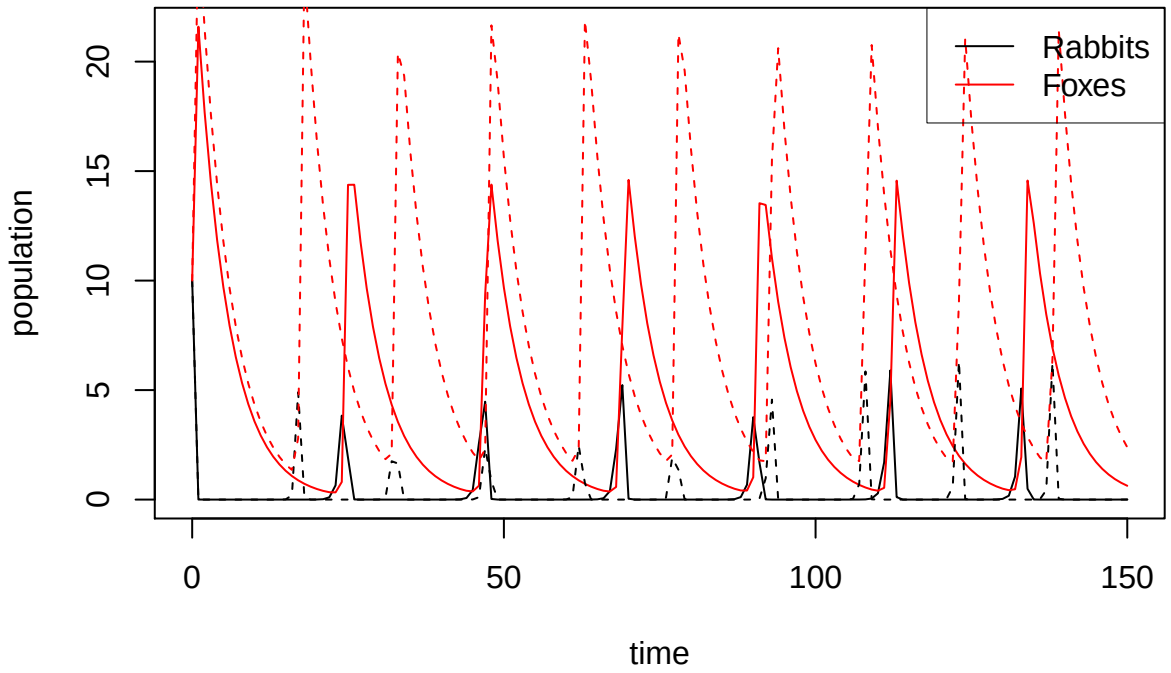
```
State2 <- c(x = 5, y = 10)
out2 <- as.data.frame(ode(func = LotVmod, y = State2, parms = Pars, times = Time))
matplot(out[,1],out[,-1], type = "l",
        xlab = "time", ylab = "population", lty=1)  #plot out our previous results
matplot(out2[,1],out2[,-1], type = "l", lty=2,add=TRUE)  #add the new runs
legend("topright", c("Rabbits", "Foxes"), lty = c(1,1), col = c(1,2), box.lwd = 0)
```



This shows the same pattern, just slightly lagged in time. After the initial **transient** (detectable by slightly higher or lower peaks for the first few cycles), the two simulations start **repeating the same pattern** over and over.

We can try the same with different parameters, to understand what the parameters do, again overlaying the new results for comparison. Here, we double the parameter α, which determines prey reproduction:

```
Pars2 <- c(alpha = 4, beta = .5, gamma = .2, delta = .6)
out3 <- as.data.frame(ode(func = LotVmod, y = State, parms = Pars2, times = Time))
matplot(out[,1],out[,-1], type = "l",
         xlab = "time", ylab = "population", lty=1)
matplot(out3[,1],out3[,-1], type = "l", lty=2,add=TRUE)
legend("topright", c("Rabbits", "Foxes"), lty = c(1,1), col = c(1,2), box.lwd = 0)
```



The two time-series settle down to different patterns. Interestingly, if *more* rabbits are being produced, you end up with *less* rabbits, because you are getting *more* predators. Sometimes, this is referred to as the 'paradox of enrichment'.

Things to try:

- Change the parameter values, see if there are combinations for which coexistence of rabbits and lynxes are impossible. Try and identify these parameters. Use some of the equilibria calculations that we did in class and add them to your plots.

Susceptible-Infected-Recovered models

Let's think about another two species model: dynamics for a directly transmitted immunizing infection, with a short generation time (e.g., ~2 weeks), like measles. We need to keep track of 3 **states**, or types of individuals, i.e., susceptible individuals ('S'), infected individuals ('I') and recovered individuals ('R'). We can write the function that defines this process, assuming that the total population size is constant, which means we don't need to keep track of the ('R') compartment. This is:

```
sir <- function(t,y,parms){
  with(c(as.list(y),parms),{
    dSdt <- -beta*S*I/N
    dIdt <- beta*S*I/N - gamma*I
    list(c(dSdt,dIdt))
  })
}
```

We define a starting population with $500,000$ susceptible individuals and 20 infected individuals; and a list of parameters that define the processes, β, the transmission rate, and γ, the recovery rate (defined as 1/infectious period):

```
pop.SI <- c(S = 500000,I = 20)
values <- c(beta = 3.6,           # Transmission coefficient
            gamma = 1/5,          #recovery rate
            N=800000)             # population size (constant)
```

We can calculate the value of the basic reproduction number $R_0 = \beta/\gamma$, as follows:

```
values["beta"]/values["gamma"] # value of R0
```

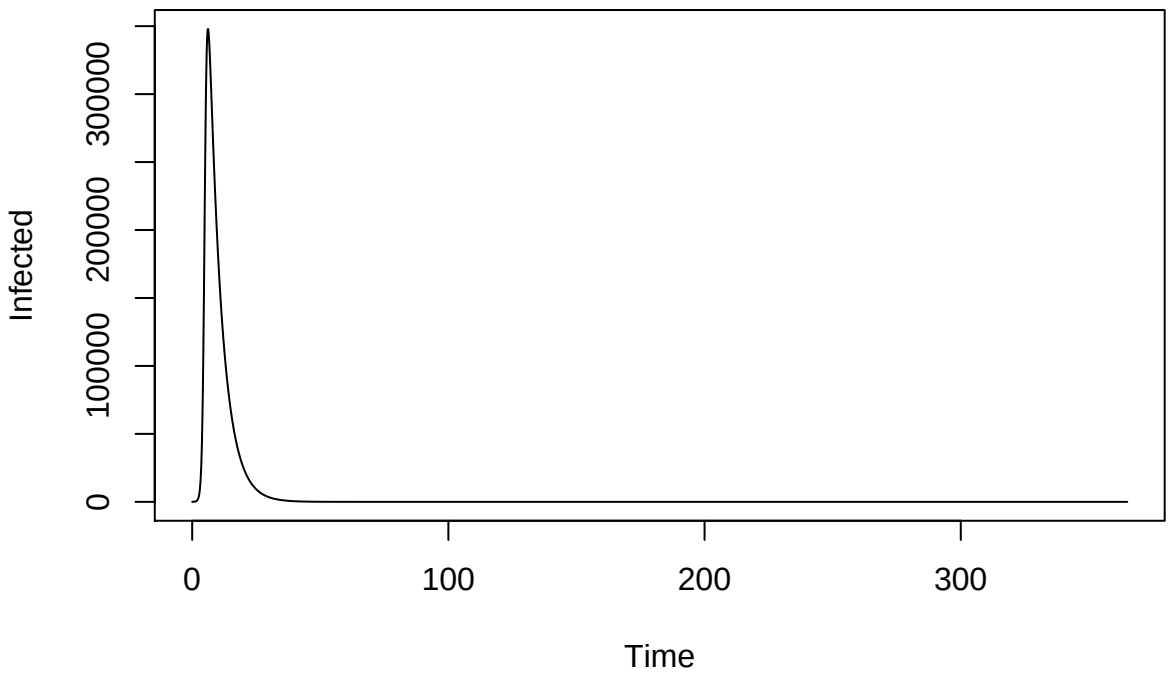
```
## beta
##   18
```

As in the population example, we can use the function lsoda, here taking as our time-unit 1 day, setting a small time-step, and running it out across 1 year (365 days):

```
ts.sir <- data.frame(lsoda(
  y = pop.SI,                    # Initial conditions for population
  times = seq(0,365,by=0.1),    # Timepoints for evaluation
  func = sir,                    # Function to evaluate
  parms = values                 # Vector of parameters
  ))
```

and then we can plot this:

```
plot(ts.sir[,"time"],ts.sir[,"I"], xlab="Time", ylab="Infected", type="l")
```



The epidemic burns itself out (i.e., at the end, the number of infected individuals is zero), but, interestingly, not all susceptible individuals are infected (i.e., the number of susceptibles at the end is > 0). This is the phenomenon of **herd immunity**.

Things to try:

- This model reflects a 'closed' population. There are **no new births** entering the population. Can you imagine how you might change the model to include this? What are the main dynamical consequences?

Discrete time SIR

We can also frame the question via a **discrete-time SIR**. An advantage (discussed below) is that this allows us to directly fit data and estimate parameters such as transmission and its seasonal fluctuations (we can fit continuous models to data too; its just a bit more complicated).

We use a function similar to the ODE one above, but in discrete time. We assume that each year contains 26 time steps, and that one time-step is approximately the generation time of the pathogen, i.e., around 2 weeks (like measles). Accordingly, the transmission parameter β is approximately equal to R_0.

We also introduce seasonality in transmission, using $\beta = R_0(1 + \gamma(2\pi t))$ where γ describes the strength of seasonality. We use S/N in the transmission process, since many lines of evidence suggest frequency dependent transmission for these types of infections.

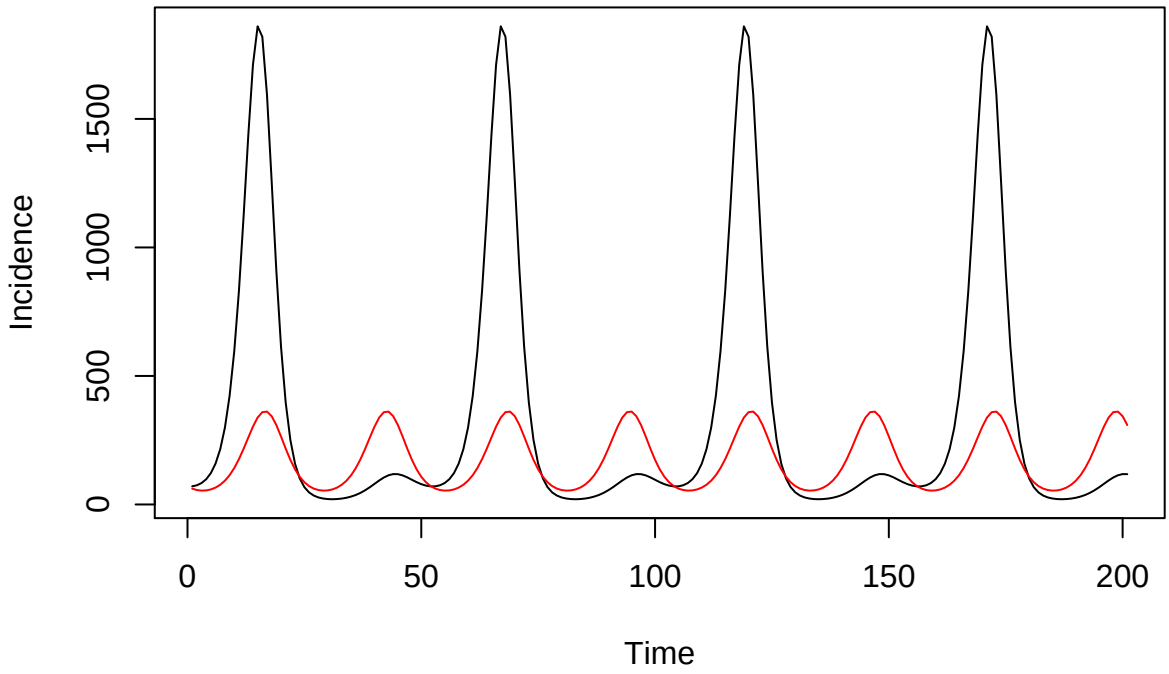
Since discretizing the process (which in reality is continuous) can result in very explosive dynamics which are not very realistic, we frame the process with I^α where $\alpha = 0.97$. This essentially attenuates the large peaks, and lessens the effect of the troughs. To make this more interesting, we are going to allow for seasonal fluctuations in births and vaccination in the way we frame our function, which includes some default arguments (set by "=" in the function arguments):

```
simTSIR <- function(I0=10,S0=1000,N=300000,      #starting no infected, susceptible, total pop size
                    beta=15,alpha=0.97,gamma=0.2,#transmission, alpha, and recover
                    vacc.cover=rep(0,26),        #one level of vaccination for every week of the year
                    births=rep((30*300/26),26),  #one level of birth for every week of the year
                    Tmax=2600){                  #maximum desired time-span.
  #set up storage for the population; vectors of length Tmax
  Istore <- Sstore <- rep(NA,Tmax)
  Istore[1] <- I0
  Sstore[1] <- S0

  #set up the seasonality (allowing transmission to fluctuate over the course of the year)
  beta.seas <- beta*(1+gamma*cos(2*pi*(1:26)/26))
  #create an index, so for each of the Tmax time-steps, we know what season it is
  seas.index <- rep(1:26,Tmax/26)

  #loop over the time-series calculating new infecteds and susceptibles.
  for (t in 2:Tmax){
    Istore[t] <-  beta.seas[seas.index[t]]*(Istore[t-1]^alpha)*Sstore[t-1]/N
    Sstore[t] <- Sstore[t-1]+(births[seas.index[t]]*(1-vacc.cover[seas.index[t]]))-Istore[t]
  }
  return(list(Istore=Istore,Sstore=Sstore))
}
```

In human populations, births range from 12 to 45 per 1000 per year; and values of γ for measles range from around 0.2 to 0.6. An example run is:

```
tmp <- simTSIR() ## by just leaving the arguments blank, R will use all the defaults.
plot(tmp$Istore[2200:2400], type="l", xlab="Time", ylab="Incidence")
        #(leave out the first 2200 time-steps to get rid of transients)
```

```
tmp1 <- simTSIR(births=rep((15*300/26),26)) ##half the birth rate, and rerun
points(tmp1$Istore[2200:2400], type="l",col=2)  #add this to the previous plot
```

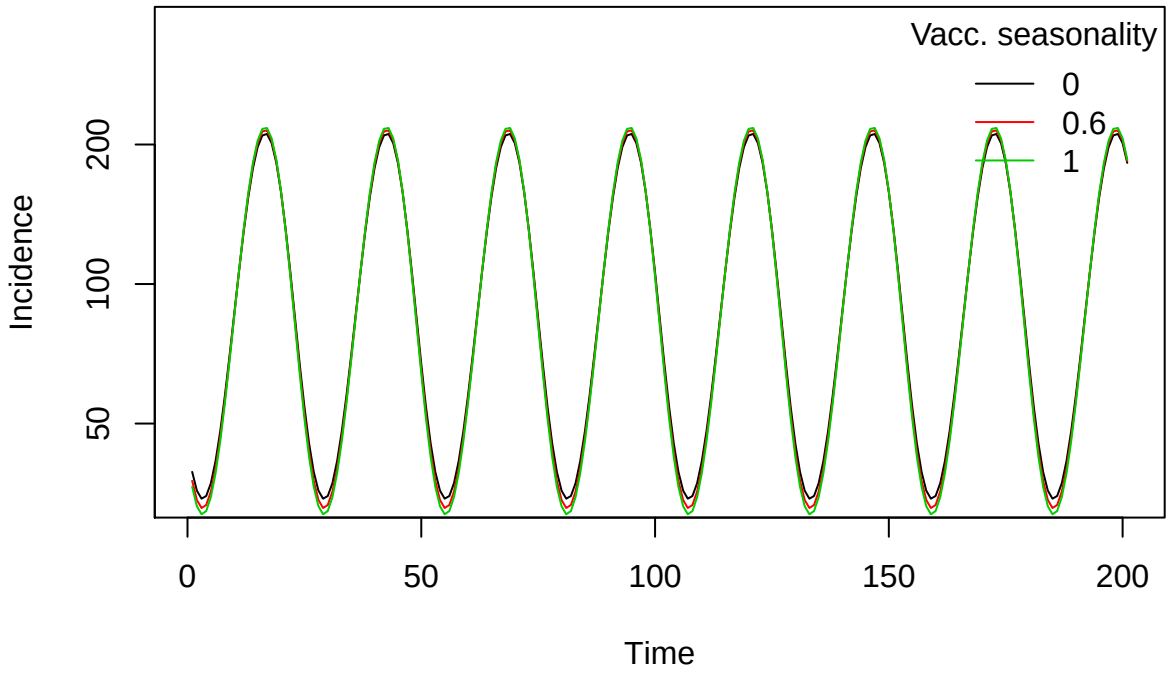


Shifting the birth rates moves us from biennial dynamics (at high birth rates) to annual dynamics (at low birth rates). This is a general feature of the non-linear feedbacks linked to infectious disease dynamics.

In places where road access may fluctuate over the course of the year, vaccination coverage may also fluctuate seasonally. This could affect dynamics. We can explore the impact of seasonality in vaccination, assuming vaccination coverage of 0.50 and increasing the amplitude of seasonality, using a cosine as above. We implement an offset in the timing of when vaccination peaks, so that seasonality in transmission and coverage don't exactly echo each other.

```
mean.coverage <- 0.4
time.offset.cover <- 0.8

# get coverage with three increasing levels of seasonality  (0,0.6 and 1);
### using pmin and pmax to make sure values stay between 0 and 1
cover1 <- pmax(pmin(mean.coverage*(1+0*cos(2*pi*(time.offset.cover+(1:26)/26))),1),0)
cover2 <- pmax(pmin(mean.coverage*(1+0.6*cos(2*pi*(time.offset.cover+(1:26)/26))),1),0)
cover3 <- pmax(pmin(mean.coverage*(1+1.0*cos(2*pi*(time.offset.cover+(1:26)/26))),1),0)
#[plot these out for comparison to see how they differ; and to see how the offset works!]

#Now run time-series that differ only in seasonality in vaccination:
tmp0 <- simTSIR(vacc.cover=cover1,
                births=rep((15*300/26),26))

tmp0.6 <- simTSIR(vacc.cover=cover2,
                births=rep((15*300/26),26))
tmp1 <- simTSIR(vacc.cover=cover3,
                births=rep((15*300/26),26))

#plot them out to see what's going out
```

```
plot(tmp0$Istore[2200:2400], type="l", xlab="Time", ylab="Incidence",
     ylim=range(min(tmp0$Istore[2200:2400]),max(tmp0$Istore[2200:2400])+150),
     log="y")  #log it, because this makes differences more obvious.
points(tmp0.6$Istore[2200:2400], type="l", col=2)
points(tmp1$Istore[2200:2400], type="l", col=3)

legend("topright", legend=c(0,0.6,1), col=1:3,title="Vacc. seasonality", lty=1,bty="n")
```



The plot suggests that there is an effect - but it is small! The largest magnitude of seasonal fluctuations in vaccination coverage (with variability set at 1) corresponds to slightly higher peaks, and slightly deeper troughs than the others (green line). To explore this thoroughly, we would need to test i) a range of vaccination coverage levels; ii) a range of birth rates; iii) a range of timings of the peak of vaccination coverage, birth rate and transmission; iv) a range of shapes of these relationships. But, we would want to map this onto a realistic situation where we thought seasonality in vaccination might be a problem.

Things to try:

- Plot out different magnitudes of seasonality and offsets to identify the different patterns
- Explore the effect of seasonality in birth in an unvaccinated population.

Fitting a TSIR model: a mechanistic model framed statistically

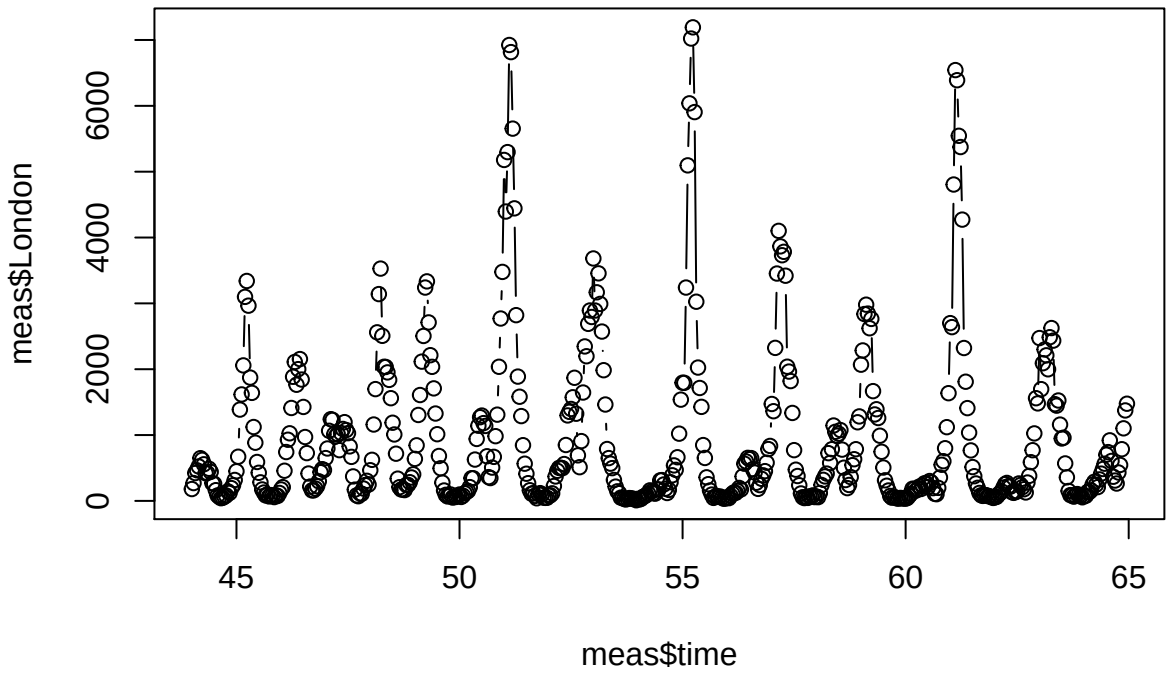
The biweekly incidence (number of cases for each two-week period) of measles has a long history in the study of infectious disease dynamics. The data set meas.csv contains the records from London between 1944 and 1966:

```
meas = read.table("meas.csv", sep = ",", header = TRUE)
names(meas)
```

```
## [1] "year"   "week"   "time"   "London" "B"
```

The data set contains columns reporting time indexes, *meas*year$, and *meas*month$, the two combined into *meas*time, *theincidence*(meas*London*), and biweekly number of births (*meas*B$). We can plot out these time-series:

```
plot(meas$time, meas$London, type = "b")
```



Since the time-step is approximately equal to the generation time of measles, the mechanistic model for this process is relatively straightforward.

$I_{t+1} = \beta I_t S_t$

With time series of I and S the candidate for estimation of core parameters is quite clear. We can just take logs on both sides, and end up with something that looks a lot like a linear-regression:

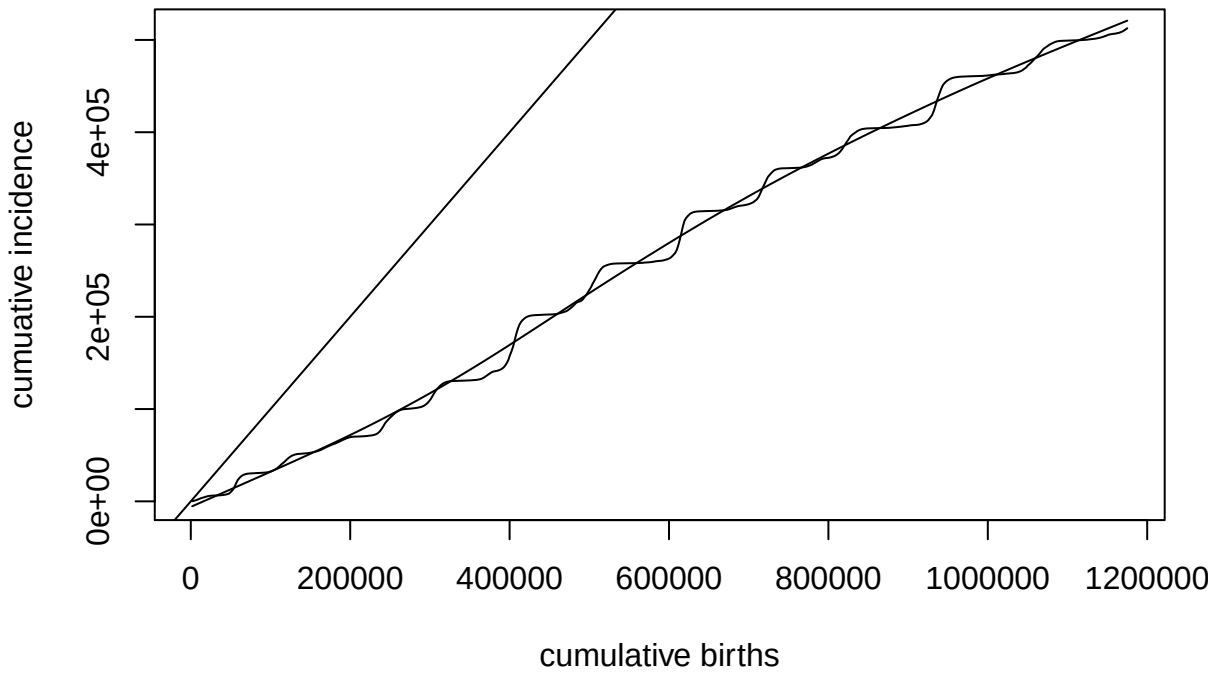
$log(I_{t+1}) = log(\beta) + log(S_t) + \alpha log(I_t)$

Note that we also put a parameter on α (i.e., we're assuming that the initial framing is: $I_{t+1} = \beta I_t^{\alpha} S_t$) - we won't go into this too much here, but, as mentioned above, this parameter accounts for the discretization, and factors not formally addressed in this simplistic framework. We can estimate the unknown parameters β and α by a regression of $\log(I_{t+1})$ on $\log(I_t)$ with $\log(S_t)$ as an offset - this means the slope for the variable is fixed at unity. The intercept of this regression would be the estimate of $\log(\beta)$ and the slope against $\log(I_t)$ would be the estimate α.

One major challenge is that most real data sets do not contain perfect records on all state variables. For example, the "meas"" data does not contain information on S, and I is under-reported. However, we do have information about births. Another challenge is the strong seasonality in transmission rates that result from aggregation of children during school term.

To tackle the first question, we can use the fact that, pre-vaccination, almost everyone become infected. As a result, the cumulative number of infected individuals will be equal to the cumulative number of births. This relationship also gives us a chance to get at the under-reporting. As it turns out, reporting rates sometimes varies subtly through time so it is good to use a slightly more flexible regression than linear regression – a smoothing spline (with 2.5 degrees-of-freedom) for example.

```
cum.reg <- smooth.spline(cumsum(meas$B), cumsum(meas$London), df=5)
D <- - resid(cum.reg) #The residuals
plot(cumsum(meas$B), cumsum(meas$London), type="l",
        xlab="cumulative births", ylab="cumuative incidence")
lines(cum.reg)
abline(a=0,b=1)
```



The 1-to-1 line generated by the abline-command shows that the cumulative number of cases are less than the cumulative number of births. The slope of the cumulative regression, therefore, is an estimate of the under-reporting rate in this system. We can get these estimated under-reporting rates for each time step from the cumulative regression as follows:

```
rr <- predict(cum.reg, deriv=1)$y
summary(rr)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3485  0.3841  0.4424  0.4522  0.5214  0.5635
```

The reporting rate is clearly almost constant across the 20-years at around 52%. As a simple hack, we can correct the time series data for the under-reporting; as well as the residuals:

```
Ic <- meas$London/rr
Dc <- D/rr
```

To estimate parameters, we rewrite the model in terms of the data and unknown parameters on a log-scale as:

$\lambda_{t+1} = log(\beta_u) + log(D_t + S) + \alpha log(I_t)$

where λ_{t+1} is the expected number of cases in time $t+1$. This is almost (but not quite) a linear regression with unknown parameters β_u, α and S.

Before we are ready to estimate the parameters, however, we need to consider the second complication, i.e., the fact that β varies seasonally; this is why we introduce the subscript u. The most flexible model is to

assume the each of the 26 biweeks of the year has its own transmission rate. Under that assumption we have 28 parameters to estimate. Let us define a vector that flags these across the 21 years (seas), and create the three vectors of current (lInew) and lagged infecteds (lIold) and lagged 'residual susceptibles' (Dold):

```
seas = rep(1:26, 21)[1:545]
lInew = log(Ic[2:546])
lIold = log(Ic[1:545])
Dold = Dc[1:545]
```

A simple trick is to realize that given a vaule for S, the models falls neatly within the linear regression framework (thought had it not, we can always write out the likelihood, and use optim to find the MLEs). We can therefore use glm to find a profile likelihood estimate of S. We know from serology that the average proportion of susceptibles in measles is somewhere in the 5%-10% range and – given the size of London at the time (3.3M) – we can postulate a reasonable range of candidate values:

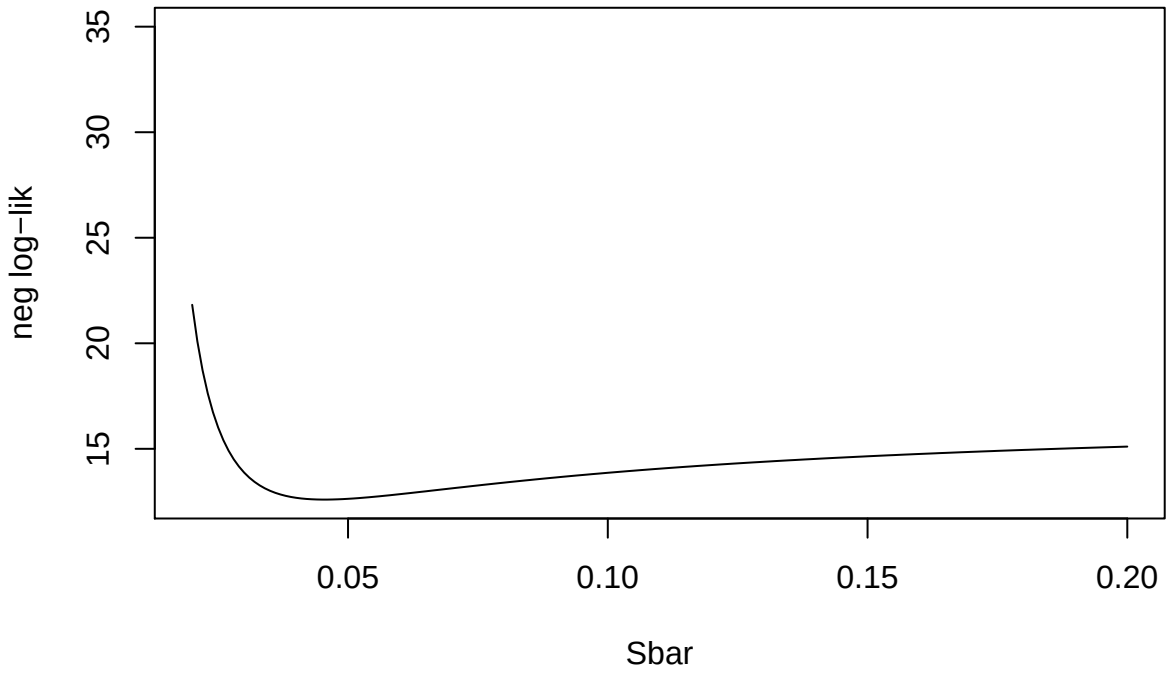
```
N <- 3.3E6
Smean <-  seq(0.02, 0.2, by=0.001)*N
offsetN <- rep(-log(N), 545)
```

We then set up a vector to store the log-likelihood values corresponding to each candidate:

```
llik <- rep(NA, length(Smean))
```

We then loop over all the values. Note the -1 in the regression formula removes the intercept, so that as.factor(seas) becomes the estimates of the log-β's (i.e., we're not estimating an overall intercept). Note further that glmfit$deviance holds -2*log likelihood.

```
for(i in 1:length(Smean)){
  lSold = log(Smean[i] + Dold)
  glmfit = glm(lInew ~ -1 +as.factor(seas) + lIold + offset(lSold+offsetN))
    llik[i] = glmfit$deviance / 2
  }
par(mfrow=c(1,1))
plot(Smean/3.3E6, llik, ylim=c(min(llik),35), xlab="Sbar", ylab="neg log-lik", type="l")
```



Our best estimates are:

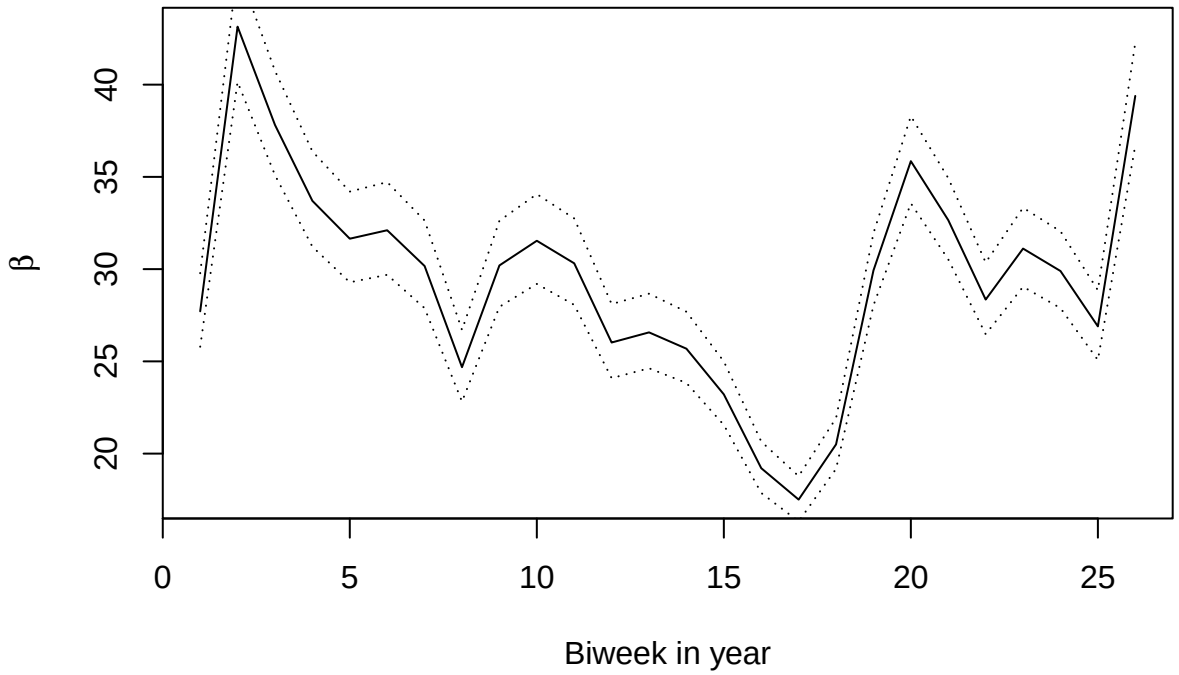
```
lSold = log(Smean[which(llik==min(llik))] + Dold)
glmfit = glm(lInew ~ -1 +as.factor(seas) + lIold + offset(lSold+offsetN))
summary(glmfit)
```

```
##
## Call:
## glm(formula = lInew ~ -1 + as.factor(seas) + lIold + offset(lSold +
##     offsetN))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.05715  -0.11754   0.00166   0.11648   0.76149
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## as.factor(seas)1  3.321905   0.071838   46.24   <2e-16 ***
## as.factor(seas)2  3.764484   0.071933   52.33   <2e-16 ***
## as.factor(seas)3  3.632712   0.074706   48.63   <2e-16 ***
## as.factor(seas)4  3.517494   0.076589   45.93   <2e-16 ***
## as.factor(seas)5  3.454708   0.077628   44.50   <2e-16 ***
## as.factor(seas)6  3.469083   0.078144   44.39   <2e-16 ***
## as.factor(seas)7  3.406754   0.078632   43.33   <2e-16 ***
## as.factor(seas)8  3.206193   0.078593   40.80   <2e-16 ***
## as.factor(seas)9  3.407831   0.077199   44.14   <2e-16 ***
## as.factor(seas)10 3.451208   0.077083   44.77   <2e-16 ***
## as.factor(seas)11 3.411638   0.077221   44.18   <2e-16 ***
## as.factor(seas)12 3.258933   0.077088   42.27   <2e-16 ***
## as.factor(seas)13 3.279828   0.075999   43.16   <2e-16 ***
## as.factor(seas)14 3.246008   0.075097   43.22   <2e-16 ***
## as.factor(seas)15 3.144629   0.074052   42.47   <2e-16 ***
```

```
## as.factor(seas)16 2.955443   0.072480   40.78   <2e-16 ***
## as.factor(seas)17 2.862731   0.069926   40.94   <2e-16 ***
## as.factor(seas)18 3.020490   0.067078   45.03   <2e-16 ***
## as.factor(seas)19 3.398819   0.065363   52.00   <2e-16 ***
## as.factor(seas)20 3.579410   0.065859   54.35   <2e-16 ***
## as.factor(seas)21 3.486030   0.067413   51.71   <2e-16 ***
## as.factor(seas)22 3.344608   0.068474   48.84   <2e-16 ***
## as.factor(seas)23 3.437610   0.068750   50.00   <2e-16 ***
## as.factor(seas)24 3.397778   0.069605   48.81   <2e-16 ***
## as.factor(seas)25 3.292005   0.070244   46.87   <2e-16 ***
## as.factor(seas)26 3.673480   0.070525   52.09   <2e-16 ***
## lIold             0.963691   0.008019  120.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.04862894)
##
##     Null deviance: 54814.13  on 545  degrees of freedom
## Residual deviance:    25.19  on 518  degrees of freedom
## AIC: -72.876
##
## Number of Fisher Scoring iterations: 2
```

and we can plot out our estimate of the varying tranmission rates over the year, also showing the upper and lower confidence intervals:

```
beta=exp(glmfit$coef[1:26])
ubeta=exp(glmfit$coef[1:26]+summary(glmfit)$coef[1:26, 2])
lbeta=exp(glmfit$coef[1:26]-summary(glmfit)$coef[1:26, 2])

plot(beta, type="l", xlab="Biweek in year", ylab=expression(beta))
points(ubeta, type="l",lty=3)
points(lbeta, type="l",lty=3)
```



A famous result of this analysis is the fact that low transmission maps onto times of the year when children are on holiday - which tells us that children being in school is a key driver of fluctuations in measles transmission. So the model has alreayd been using - the phenomenological framing, or pattern over time, is telling us what the mechanism is - i.e. transmission being low when children are out of school.

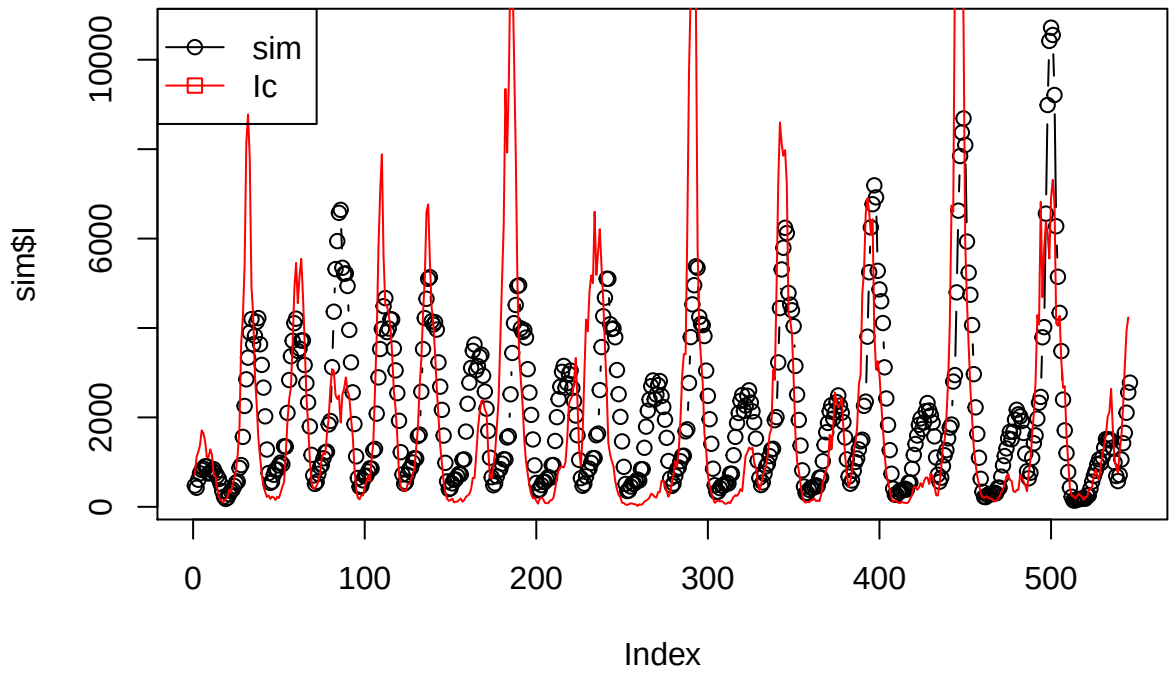
But is the model good enough to simulate from? Can we predict the future using this framework? We can write a general function SimTsir2 to simulate the seasonally-forced TSIR from the estimated parameters - similar but slightly different to the one we used above. We also write it to either perform a deterministic (type="det") or stochastic (assuming demographic stochasticity type="stoc") so as we might be able to use it in different settings.

```
SimTsir2=function(beta, alpha, B, N,
                  inits = list(Snull = 0, Inull = 0), type = "det"){
    type <- charmatch(type, c("det", "stoc"), nomatch = NA)
    if(is.na(type))
        stop("method should be \"det\", \"stoc\"")
    IT <- length(B)
    s <- length(beta)
    lambda <- rep(NA, IT)
    I <- rep(NA, IT)
    S <- rep(NA, IT)
    I[1] <- inits$Inull
    lambda[1] <- inits$Inull
    S[1] <- inits$Snull
    for(i in 2:IT) {
        lambda[i] = beta[((i - 2) %% s) + 1] * S[i - 1] * (I[i - 1]^alpha)/N
        if(type == 2) {
                I[i] = rpois(1, lambda[i])
            }
        if(type == 1) {
            I[i] = lambda[i]
}
        S[i] =S[i - 1] + B[i] - I[i]
```

```
    }
    return(list(I = I, S = S))

}
```

and to simulate from this and plot:

```
sim <- SimTsir2(beta=exp(glmfit$coef[1:26]), alpha=glmfit$coef[27],
                B=meas$B, N=N,
                inits=list(Snull=Dc[1]+Smean[which(llik==min(llik))], Inull=Ic[1]))
plot(sim$I, type="b")
lines(exp(lInew), col="red")
legend("topleft",
       legend=c("sim", "Ic"),
       lty=c(1,1),
       pch=c(1,0),
        col=c("black", "red"))
```



So it works quite well! and we might be able to predict the future for measles. Note that this is based on just how simple the measles life cycle is (infection is completely immunizing and there is no recovery; also the generation time is short, and can be approximated by two weeks). Trying to do this for something like malaria is likely to be much more challenging.

Things to try:

- You can download time-series for a number of infections including measles from https://www.tycho.pitt.edu; you could explore seasonality for different pathogens, or patterns through space and time, and across different city sizes

Final notes

Some of this material is extended from DAIDD and MMED courses; the TSIR section is from Ottar Bjornstad.

References

Bjornstad, O.N., B. Finkenstadt, and B.T. Grenfell, Endemic and epidemic dynamics of measles: Estimating epidemiological scaling with a time series SIR model. Ecological Monographs, 2002. 72: p. 169-184.

Cawell, H. Matrix Population Models. 2001. Sinauer

Salguero-Gomez, R., et al. The COMPADRE Plant Matrix Database: an open online repository for plant demography. Journal of Ecology, 2015. 103(1): p. 202-218.