Simulation of epidemics in space

Aaron A. King

June 5, 2008

Contents

1 Chain binomial model

We've seen that, in a simple SIR disease, it is sometimes useful to model transmission using a chain binomial formulation. In each generation, new infections are binomially distributed with size (i.e., number of trials) equal to the number of susceptibles, S_t, and probability of infection, $p = 1-\exp(-\beta\, I_t)$:

$$I_{t+1} \sim \text{binom}(S_t, 1 - \exp(-\beta\, I_t))$$

The susceptibles are then depleted by the number of these infections

$$S_{t+1} = S_t - I_{t+1}$$

Simulating this simple model is extremely easy in R. First, we write a little function that simulates a single generation.

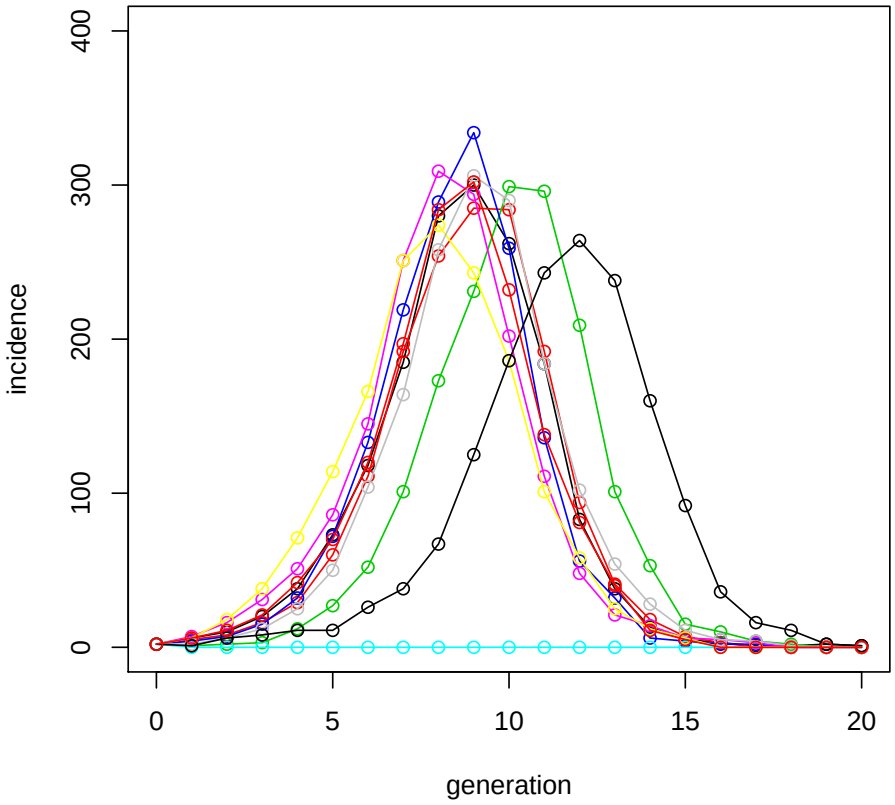
```
> chain.binomial.onestep <- function (x, params) {
+    S <- x[1]
+    I <- x[2]
+    beta <- params['beta']
+    new.I <- rbinom(n=1,size=S,prob=1-exp(-beta*I))
+    new.S <- S-new.I
+    c(S=new.S,I=new.I)
+ }
```

Then we put these together in sequence to simulate an entire epidemic:

```
> chain.binomial.model <- function (x, params, nstep) {
+   X <- array(dim=c(nstep+1,3))
+   colnames(X) <- c("time","S","I")
+   X[1,1] <- 0
+   X[1,-1] <- x
+   for (k in 1:nstep) {
+     X[k+1,1] <- k
+     X[k+1,-1] <- x <- chain.binomial.onestep(x,params)
+   }
+   X
+ }
```

We'll now specify some parameters, simulate the model a few times, and plot the results.

```
> set.seed(38499583)
> nsims <- 10
> nstep <- 20
> xstart <- c(S=2000,I=2)
> params <- c(beta=1e-3)
> x <- vector(mode='list',length=nsims)
> for (k in 1:nsims) {
+   x[[k]] <- as.data.frame(chain.binomial.model(xstart,params,nstep))
+ }
> plot(c(0,20),c(0,400),type='n',xlab='generation',ylab='incidence')
> for (k in 1:nsims) {
+   lines(I~time,data=x[[k]],col=k,type='o')
+ }
```



2 The SIR model

The chain binomial, like all models, is an approximation. One particularly big assumption that it makes is that the generations are perfectly synchronized. For some diseases, this may not be such a bad approximation; in others, it might very well be. Let's have a look at what can be done with models that don't make this assumption, i.e., when the dynamics play out in continuous time.

The simplest place to start is with the classical SIR model. This model divides the host population into three classes with respect to their infection status: individuals are either Susceptible, Infected (and Infectious), or Recovered. The model simply keeps track of how many individuals are in each class: individuals that leave one class must enter another. The only exceptions, of course, are births and deaths.

$$\begin{aligned}\frac{dS}{dt} &= \mu N - \lambda(I,t)\,S - \mu S\\ \frac{dI}{dt} &= \lambda(I,t)\,S - \gamma I - \mu I\\ \frac{dR}{dt} &= \gamma I - \mu R\end{aligned}$$

Here, $b(t)$ is the birth rate (which we have assumed is equal to the death rate), N is the host population size, and γ the recovery rate. The only interesting bit is the force of infection $\lambda(I,t)$. We'll assume that

it has the very simple, so-called *frequency dependent* form

$$\lambda(I,t) = \beta(t)\,\frac{I}{N}$$

so that the risk of infection a susceptible faces is proportional to the fraction of the population that is infectious. Notice that we allow for the possibility of a contact rate, β, that varies in time.

The bad news is that one can't solve the SIR equations explicitly. Rather, to simulate a continuous-time model such as the SIR, we must integrate those ordinary differential equations (ODE) numerically. In general, this can be a tricky business. The good news is that R has a very sophisticated ODE solver facility which for many problems will give highly accurate solutions. The bad news is that we can only obtain one (or a few) trajectories of the system at a time. To use the numerical integration package, we must load the package

```
> require(odesolve)
```

[If you get a warning that the package was not loaded, check to make sure it is installed on your computer.] The ODE solver needs to know the right-hand sides of the ODE. We give it this information as a function:

```
> sir.model <- function (t, x, params) {
+   S <- x[1]
+   I <- x[2]
+   R <- x[3]
+   with(
+        as.list(params),
+        {
+          dS <- mu*(N-S)-beta*S*I/N
+          dI <- beta*S*I/N-(mu+gamma)*I
+          dR <- gamma*I-mu*R
+          res <- c(dS,dI,dR)
+          list(res)
+        }
+        )
+ }
```

Notice that here, we've assumed β is constant. [In case the `with` function is unfamiliar, it serves here to make the parameters `params` available to the expressions in the brackets, *as if they were variables*. One could achieve the same effect by, for example, `dS <- params["mu"]*(params["N"]-S)-params["beta"]*S*I/params["N"]` and so on.]

We'll now define the times at which we want solutions, assign some values to the parameters, and specify the *initial conditions*, i.e., the values of the state variables S, I, and R at the beginning of the simulation:

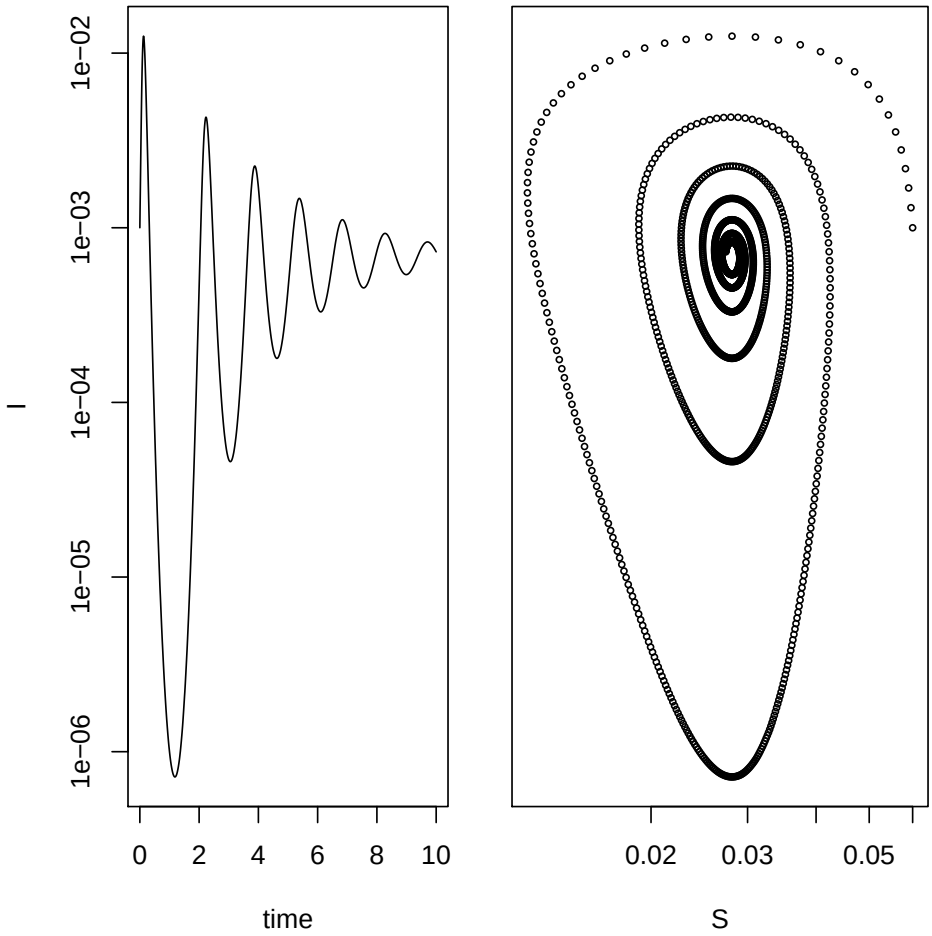
```
> times <- seq(0,10,by=1/120)
> params <- c(mu=1/50,N=1,beta=1000,gamma=365/13)
> xstart <- c(S=0.06,I=0.001,R=0.939)
```

Now we can simulate a model trajectory with the `lsoda` command:

```
> out <- as.data.frame(lsoda(xstart,times,sir.model,params))
```

and plot the results

```
> op <- par(fig=c(0,0.5,0,1),mar=c(4,4,1,1))
> plot(I~time,data=out,type='l',log='y')
> par(fig=c(0.5,1,0,1),mar=c(4,1,1,1),new=T)
> plot(I~S,data=out,type='p',log='xy',yaxt='n',xlab='S',cex=0.5)
> par(op)
```



Exercise 1. Explore the dynamics of the system for different values of the β and μ parameters by simulating and plotting trajectories as time series and in phase space (e.g., I vs. S).

***Exercise 2.** Modify the codes given to study the dynamics of an SEIR model.

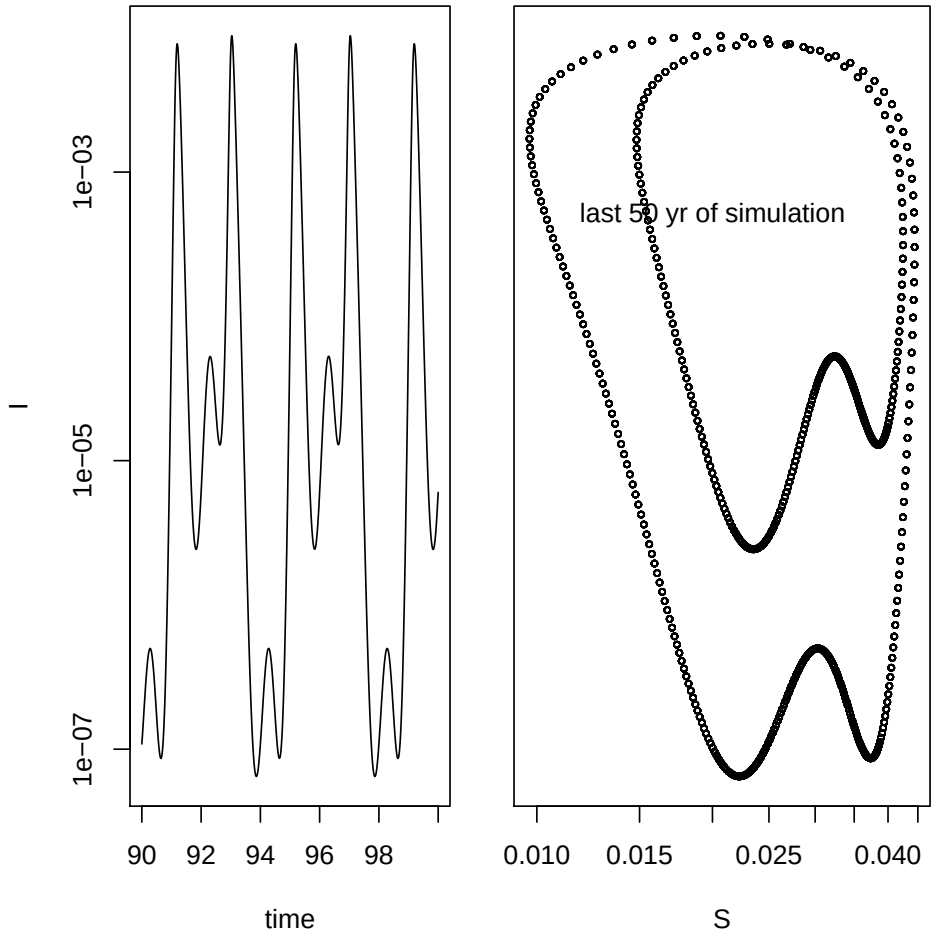
Seasonality

The simple SIR model always predicts damped oscillations towards an equilibrium (or pathogen extinction if R_0 is too small). This is at odds with the recurrent outbreaks seen in many real pathogens. Sustained oscillations require some additional drivers in the model; an important driver in childhood infections of humans (e.g., measles) is seasonality in contact rates because of aggregation of children the during school term. We can analyze the consequences of this by assuming sinusoidal forcing on β according to $\beta(t) = \beta_0 \left(1 + \beta_1 \cos(2\,\pi\,t)\right)$. Translating this into R:

```
> seasonal.sir.model <- function (t, x, params) {
+   with(
+        as.list(c(x,params)),
+        {
+          beta <- beta0*(1+beta1*cos(2*pi*t))
+          dS <- mu*(N-S)-beta*S*I/N
+          dI <- beta*S*I/N-(mu+gamma)*I
+          dR <- gamma*I-mu*R
+          res <- c(dS,dI,dR)
+          list(res)
+        }
+        )
+ }
```

We'll simulate as before, with the same mean contact rate, β_0 as before, but now with a fairly strong amplitude of seasonality, β_1.

```
> times <- seq(0,100,by=1/120)
> params <- c(mu=1/50,N=1,beta0=1000,beta1=0.4,gamma=365/13)
> xstart <- c(S=0.06,I=0.001,R=0.939)
> out <- as.data.frame(lsoda(xstart,times,seasonal.sir.model,params,rtol=1e-12,hmax=1/120))
> op <- par(fig=c(0,0.5,0,1),mar=c(4,4,1,1))
> plot(I~time,data=out,type='l',log='y',subset=time>=90)
> par(fig=c(0.5,1,0,1),mar=c(4,1,1,1),new=T)
> plot(I~S,data=out,type='p',log='xy',subset=time>=50,yaxt='n',xlab='S',cex=0.5)
> text(0.02,0.0005,"last 50 yr of simulation")
> par(op)
```



Exercise 3. Explore the effects of changing amplitude of seasonality, β_1 on the dynamics of this model. Be careful to distinguish between transient and asymptotic dynamics.

A great deal more on the dynamics of seasonal SIR-type models such as this can be found, for example, in Earn et al. (2000).

Fitting continuous-time models to data: trajectory matching

If we assume that the only source of variability in the data is measurement error, i.e., that there is no process noise, then *trajectory matching* is a statistically appropriate basis for inference. Let's fit the SIR model to the Niamey measles outbreak data.

```
> niamey <- read.csv("niamey_measles.csv")
```

In particular, if we assume normal measurement error, then maximizing the likelihood is equivalent to fitting by least-squares. First, we'll assume that demographic turnover over the course of this epidemic is negligible. This allows us to combine the β and N parameters; we'll take $b = \beta/N$.

```
> closed.sir.model <- function (t, x, params) {
+   S <- x[1]
```

```
+   I <- x[2]
+   with(
+        as.list(params),
+        {
+          dS <- -b*S*I
+          dI <- b*S*I-gamma*I
+          list(c(dS,dI))
+        }
+        )
+ }
```

Notice that we can ignore the recovered class, since recovered individuals can't influence the dynamics in any way.

Next, we'll need to have the log likelihood as a function of the parameters, (β, γ) and the initial conditions for S and I. Because all of these parameters must be positive to make sense, we'll log transform them. This will keep the optimizer (see below) from straying off into meaningless regions of parameter space.

```
> sir.NLL <- function (log.b, gamma, log.S.0, log.I.0, biweek, cases) {
+   params <- c(b=exp(log.b),gamma=gamma)
+   xstart <- c(S=exp(log.S.0),I=exp(log.I.0))
+   out <- as.data.frame(lsoda(xstart,times=biweek*14/365,closed.sir.model,params,hmax=1/120))
+   length(cases)*log(mean((out$I-cases)^2))/2
+ }
```

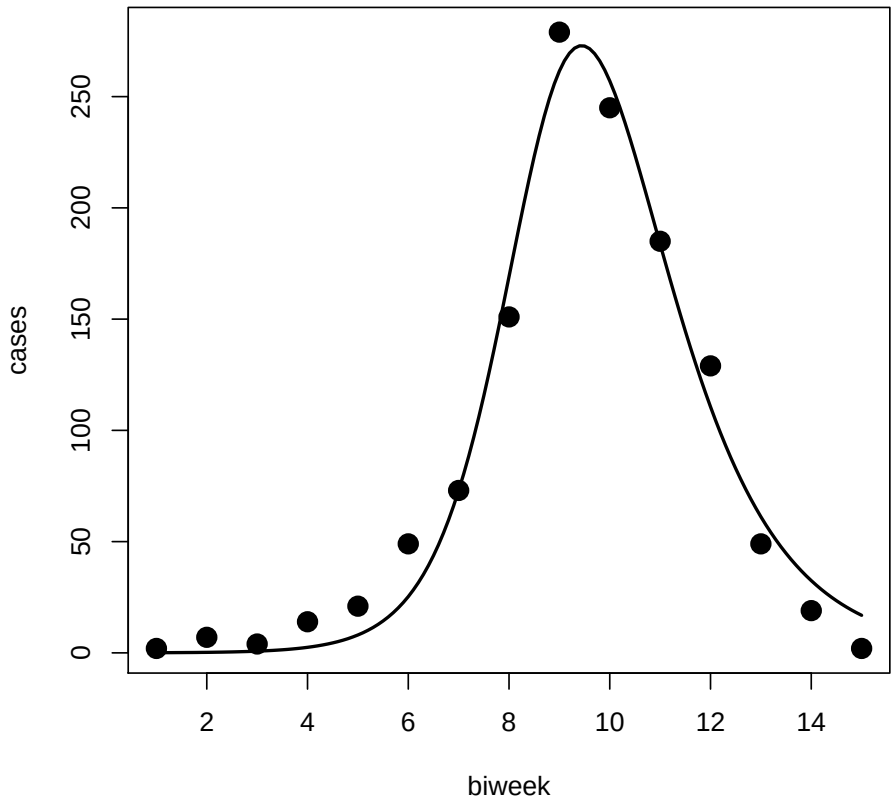
Now we'll use `mle2`—from the `bbmle` package—to maximize the likelihood.

```
> require(bbmle)
> fit <- mle2(
+             sir.NLL,
+             start=list(log.b=log(0.1),log.S.0=log(500),log.I.0=log(1)),
+             fixed=list(gamma=365/13),
+             data=as.list(subset(niamey,site==3)),
+             optimizer="optim",
+             method="Nelder-Mead"
+             )
> summary(fit)


Maximum likelihood estimation

Call:
mle2(minuslogl = sir.NLL, start = list(log.b = log(0.1), log.S.0 = log(500),
    log.I.0 = log(1)), method = "Nelder-Mead", optimizer = "optim",
    fixed = list(gamma = 365/13), data = as.list(subset(niamey,
        site == 3)))

Coefficients:
        Estimate Std. Error  z value     Pr(z)
log.b   -3.29826    0.07597 -43.4153 < 2.2e-16 ***
log.S.0  7.37605    0.04868 151.5200 < 2.2e-16 ***
log.I.0 -2.64453    0.53695  -4.9251 8.433e-07 ***
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

-2 log L: 77.28913
```

Trajectory matching gives us an estimate of $\hat{b} = 0.037$, $\hat{S}_0 = 1600$, and $\hat{R}_0 = \hat{b}\,\hat{S}_0/\gamma = 2.1$, which can be compared with the estimates obtained using the chain binomial. We can evaluate the fit graphically:

```
> dat <- subset(niamey,site==3)
> plot(cases~biweek,data=dat,type='n')
> lines(cases~biweek,data=dat,col='black',cex=1.5,pch=16,type='p',lwd=2)
> params <- with(as.list(coef(fit)),c(b=exp(log.b),gamma=gamma))
> xstart <- with(as.list(coef(fit)),c(S=exp(log.S.0),I=exp(log.I.0)))
> times <- seq(from=min(dat$biweek),to=max(dat$biweek),by=0.1)*14/365
> out <- as.data.frame(lsoda(xstart,times=times,closed.sir.model,params,hmax=1/120))
> lines(I~I(time*365/14),data=out,col='black',lwd=2)
```



Exercise 4. Fit the closed SIR epidemic model to the data from the other two sites. How consistent are the results?

3 Simple epidemics in space

The simplest spatial model is one with two patches. Here, we'll see how to extend the SIR model to this spatial context. By extension, it will become clear how to extend to larger numbers of patches and other continuous-time models. The simplest way to model the coupling of two patches is to assume that hosts move back and forth between patches at some rate which is independent of their disease status. To illustrate this, let's consider the closed SIR epidemic model in two patches. We'll hang subscripts on the state variables to indicate to which patch they refer. Thus

$$\begin{aligned}
\frac{dS_1}{dt} &= -\beta S_1 I_1 + m(S_2 - S_1) \\
\frac{dI_1}{dt} &= \beta S_1 I_1 - \gamma I_1 + m(I_2 - I_1) \\
\frac{dS_2}{dt} &= -\beta S_2 I_2 + m(S_1 - S_2) \\
\frac{dI_2}{dt} &= \beta S_2 I_2 - \gamma I_2 + m(I_1 - I_2)
\end{aligned}$$

The number of ODE has now doubled, but the structure is not too much worse. Specifically, the transmission and recovery processes are localized to each patch; this means that the same function can be applied to each patch to take account of transmission and recovery. Secondly, migration has a particularly simple form: it is linear. These two facts will help us not only to write cleaner, faster code, but to understand the system better. This code introduces two tricks to take advantage of the two points mentioned above. First, we pass the variables as a vector `c(S1,I1,S2,I2)`. The ode solver computes the derivatives, but it operates on vectors, so no looping is required and the program runs faster. We apply the local model to each patch independently. Second, to deal with migration, we use matrix multiplication, with the symbol `%*%`.

```
> twopatch.sirs.model <- function (t, x, params) {
+   d1 <- closed.sir.model(t,x[c(1,2)],params)[[1]] # local dynamics, patch 1
+   d2 <- closed.sir.model(t,x[c(3,4)],params)[[1]] # local dynamics, patch 2
+   mig <- with(
+               as.list(params),
+               m*c(Psi%*%x[c(1,3)],Psi%*%x[c(2,4)])-m*x[c(1,3,2,4)]
+               )
+   list(c(d1,d2)+mig[c(1,3,2,4)])
+ }
```

Note that `closed.sir.model` returns a list with just one element in it; we don't want a list, so we'll extract that one element. The matrix Ψ is defined to be:

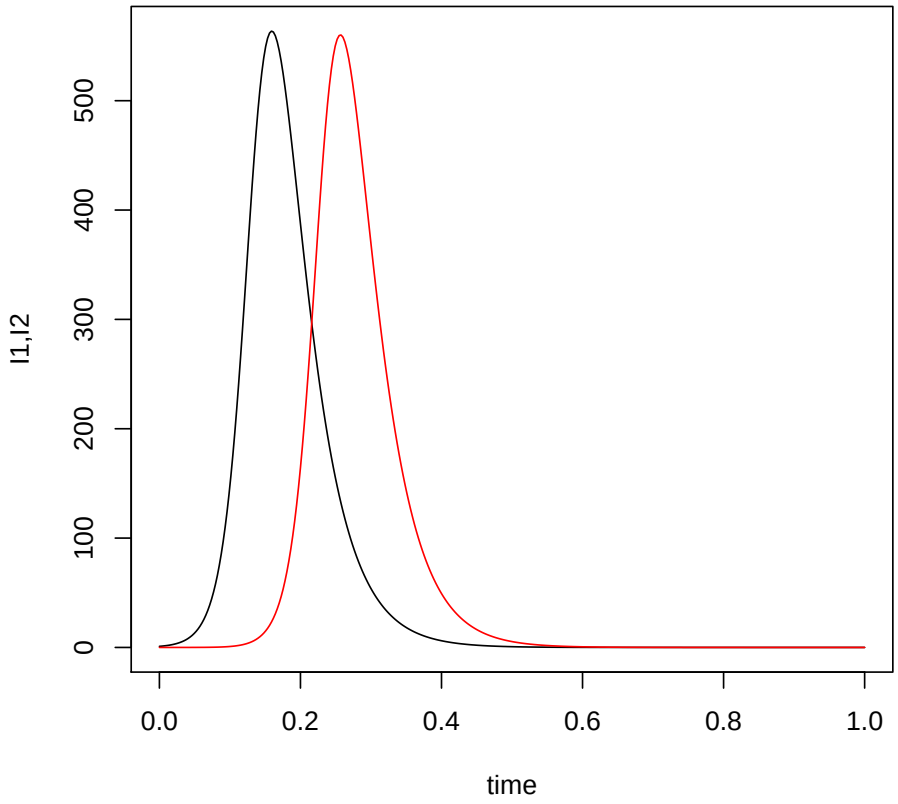
```
> Psi <- matrix(c(0,1,1,0),2,2)
> Psi
```

```
     [,1] [,2]
[1,]    0    1
[2,]    1    0
```

So, for example, `Psi %*% c(S1,S2)` returns the vector `c(S2,S1)`. Matrix multiplication can be very fast. This is a trick we'll extend when we simulate epidemics on more complicated spaces. Finally, to conserve individuals, i.e., to make sure that a migrant leaving patch 1 doesn't just vanish into thin air, we have to subtract `m*x[c(1,3,2,4)]`.

Now, we need to set up the parameter values, run a simulation, and plot the epidemic curves in the two patches.

```
> times <- seq(0,1,by=1/365)
> xstart <- c(S1=2000,I1=1,S2=2000,I2=0)
> params <- c(b=0.04,gamma=365/13,m=0.05)
> out <- as.data.frame(lsoda(xstart,times,twopatch.sirs.model,params,reltol=1e-12,hmax=1/365))
> plot(I1~time,data=out,type='l',col='black',ylab='I1,I2')
> lines(I2~time,data=out,type='l',col='red')
```



Exercise 5. Play a bit with the two-patch SIR model. How does the migration parameter m affect the dynamics?

4 More complicated spaces

We want to develop some tools for dealing with more complicated spaces. Sometimes, space has a natural patch structure. At other times, we make continuous space more computationally and operationally managable by chopping it up into a set of patches. Complicated spaces can be dealt with using the same tricks as we used in the two-patch case—we simply need a systematic way to index patches and to write down the adjacency matrix Ψ. Conceptually, Ψ is the matrix where the (i, j)-th entry quantifies

the strength of the coupling between the i-th and j-th patches. Here, we'll develop some methods for defining the nearest-neighbor matrix on one-dimensional arrays, rectangular grids, and arbitrary sets of points.

One-dimensional array of patches. The adjacency matrix for a one-dimensional array is relatively simple. The patches are indexed in the natural way, so that the i-th patch's neighbors are $i \pm 1$. The adjacency matrix thus has positive entries on the off-diagonals. For example, the adjacency matrix for an array of 5 patches has the form:

```
> Psi <- matrix(0,5,5)
> Psi[abs(row(Psi)-col(Psi))==1] <- 1
> Psi
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    0    0    0
[2,]    1    0    1    0    0
[3,]    0    1    0    1    0
[4,]    0    0    1    0    1
[5,]    0    0    0    1    0
```

One small detail has been swept under the rug here. In the above, we assumed that the patches at the boundaries had no neighbors and so no dispersal occured beyond the array. This is the assumption of *reflecting boundary conditions*. There are several other standard ways to deal with boundaries. If individuals dispersing outside of the array disappear or die, the boundaries are said to be *absorbing*. Finally, as would be the case if the linear space is the shoreline of an island or a lake, the patches with the highest and lowest index are each other's neighbors, so the adjacency matrix is the following:

```
> Psi[5,1] <- Psi[1,5] <- 1
> Psi
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    0    0    1
[2,]    1    0    1    0    0
[3,]    0    1    0    1    0
[4,]    0    0    1    0    1
[5,]    1    0    0    1    0
```

These correspond to so-called *periodic boundary conditions*. More generally, it is worthwhile to think carefully about what happens at the edges. Do would-be emigrants stay put, as if the edge were a barrier, or do they go where they can? Is the rate of migration at the edge higher or lower than in the interior? These are questions to be considered carefully in real problems.

Rectangular grid. A rectangular grid is just a bit more complicated. A quick way of getting the adjacency matrix for a 5×5 grid in which only nearest neighbors are coupled and the boundaries are reflecting is, for example:

```
> L <- 5
> xy <- expand.grid(1:L,1:L)
> d <- dist(xy)
```

```
> Psi <- matrix(0,L^2,L^2)
> Psi[(as.matrix(d)>0)&(as.matrix(d)<1.5)] <- 1
> Psi[1:9,1:9]
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    0    1    0    0    0    1    1    0    0
 [2,]    1    0    1    0    0    1    1    1    0
 [3,]    0    1    0    1    0    0    1    1    1
 [4,]    0    0    1    0    1    0    0    1    1
 [5,]    0    0    0    1    0    0    0    0    1
 [6,]    1    1    0    0    0    0    1    0    0
 [7,]    1    1    1    0    0    1    0    1    0
 [8,]    0    1    1    1    0    0    1    0    1
 [9,]    0    0    1    1    1    0    0    1    0
```

The command `expand.grid` gives the x-y coordinates of all the 25 points on the grid. The distances between all pairs of points can then be computed using `dist`. Note that only the first 9 rows and columns of this 25×25 matrix are displayed.

Exercise 6. Set up the adjacency matrix for a 10×5 rectangular grid of patches with (a) absorbing boundary conditions, (b) reflecting boundary conditions, and (*c) periodic boundary conditions. Use nearest-neighbor connectivity.

Exercise 7. The preceding example assumed that only nearest neighbors are connected. Suppose instead that all patches are potentially coupled, but that the strength of coupling diminishes with distance. Specifically, assume that the coupling between patches i and j is described by the *kernel*

$$\Psi_{i,j} = \exp(-d_{i,j}),$$

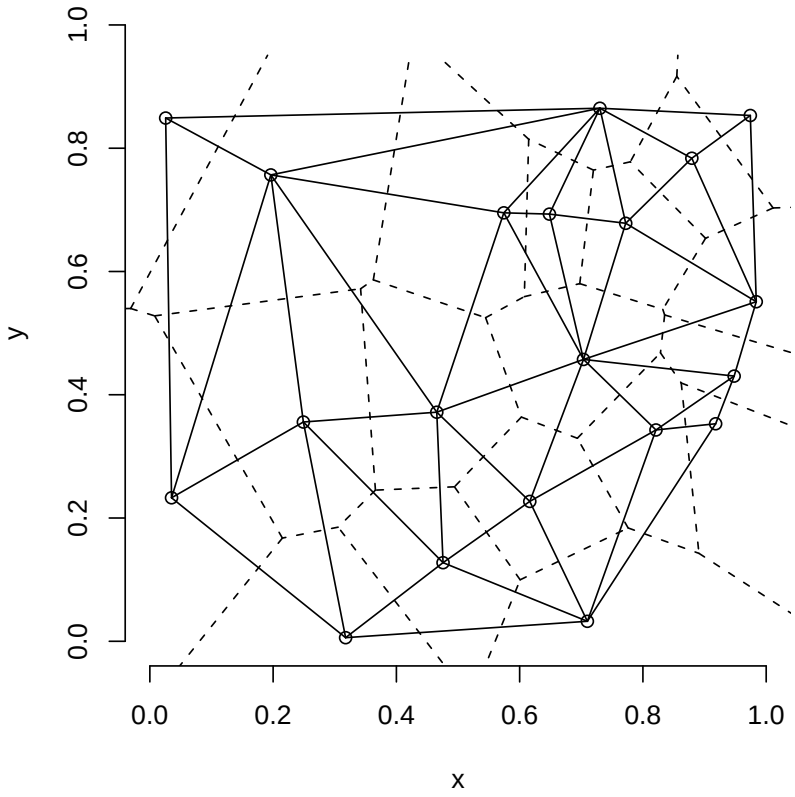
where $d_{i,j}$ is the distance between i and j. Compute the adjacency matrix using whichever boundary conditions you like.

More general spaces: tesselations of the plane. A tesselation is simply a subdivision of the plane into small regions, called tiles. We can use the R package `deldir` to do Delaunay triangulation and the Dirichlet tesselation. We'll illustrate the output of the algorithm for a random set of points:

```
> L <- 20
> x <- runif(L,0,1)
> y <- runif(L,0,1)
> require(deldir)
```

```
deldir 0.0-7
```

```
> dd <- deldir(x,y)
> plot(dd,xlim=c(0,1),ylim=c(0,1))
```



The algorithm subdivides space into a set of tiles such that every point inside the tile is closest to one of the finite set of specified points. The dotted lines show the boundaries of the tiles. The solid lines show the nearest neighbors. The last two columns of `dd$delsgs` are the indices of the adjacent pairs, so we can use these to construct the adjacency matrix:

```
> Psi <- matrix(0,L,L)
> for (k in 1:nrow(dd$delsgs)) {
+   i <- dd$delsgs[k,5]
+   j <- dd$delsgs[k,6]
+   Psi[i,j] <- Psi[j,i] <- 1
+ }
```

Note that each pair is ordered and listed once, so we have to worry about `Psi[i,j]` and `Psi[j,i]`. To create the adjacency matrix for a grid, we need only specify the distribution of points and `deldir` will do the work for us.

***Exercise 8.** Construct an adjacency matrix for the example above where you assume that the strength of coupling decreases exponentially with the distance between connected patches. As always, be careful with the boundaries. [Hint: have a look at the documentation for `deldir`: the information you need is to be found in the list it returns.]

5 Complex spatiotemporal dynamics: coupled map lattices

When both space and time as treated as discrete, dynamical models are known as *coupled map lattices*. In the following, we explore a simple coupled map lattice based on the chain binomial model. In particular, we'll construct the model so that, within each patch, the number of new infections is equal to what we'd expect it to be under the chain binomial model. To spice things up a bit, we'll also assume that the contact rate varies seasonally.

```
> local.dyn <- function(t, S, I, b0, b1, mu, N) {
+   beta <- b0*(1+b1*cos(2*pi*t/26))
+   I <- S*(1-exp(-beta*I))
+   S <- (1-mu)*S+mu*N-I
+   list(S=S,I=I)
+ }
```

We'll model space as a rectangular grid of width `xlen` and height `ylen`.

```
>   xlen <- 30
>   ylen <- 30
>   npatch <- xlen*ylen
```

We generate spatial coordinates as before and generate the adjacency matrix. Recall that the first column of `xy` the x-coordinates, the second is the y-coordinates, of the grid points.

```
>   xy <- expand.grid(1:xlen,1:ylen)
>   d <- dist(xy)
>   Psi <- matrix(0,npatch,npatch)
>   Psi[(as.matrix(d)>0)&(as.matrix(d)<1.5)] <- 1
```

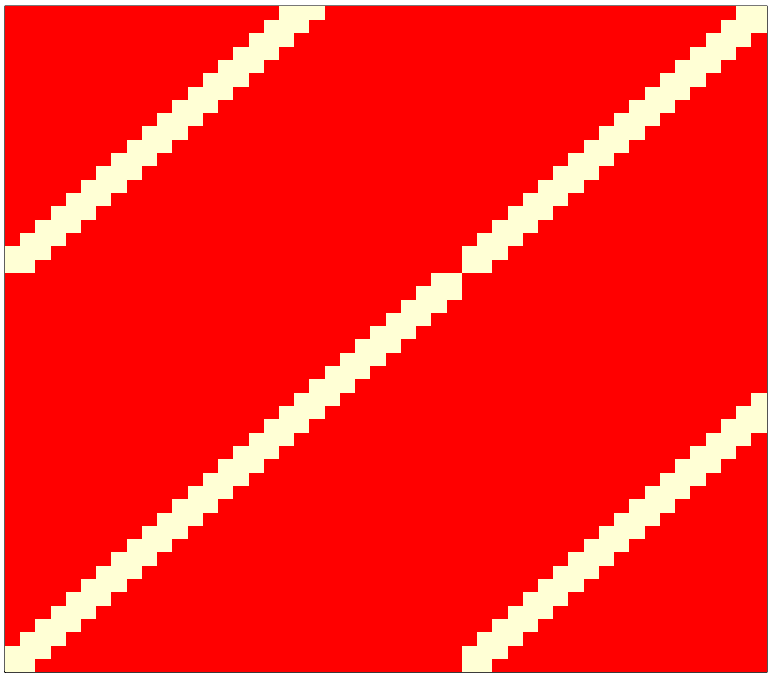
To get the maximum computational advantage, we'll combine all movement into a single dispersal matrix. As in the earlier example, to conserve individuals, we have to subtract some quantity from each diagonal so that the columns sum to one. In the following `m` is the proportion of hosts that disperse to neighboring patches.

```
> m <- 0.5
> D <- (m/8)*Psi+diag(1-m,nrow=npatch)
```

Exercise 9. Why do we divide by 8? Construct a dispersal matrix in which only the four neighbors (N, E, S, W) are coupled.

We can get a sense of the pattern of zeros and ones in the dispersal matrix D using `image`:

```
> image(D[1:50,1:50]>0,xaxt='n',yaxt='n')
```



Now we'll choose some parameters, initialize the system, and run a simulation.

```
> niter <- 260
> b0 <- 0.04
> b1 <- 0.2
> mu <- 0.02/26
> N <- 1000
> S.0 <- 100
```

In the above, `niter` is the number of generations we'll simulate. We initialize the system with infectives in lower left-hand corner only:

```
> S <- array(0,dim=c(npatch,niter))
> I <- array(0,dim=c(npatch,niter))
> I[(xy[,1]<5)&(xy[,2]<5),1] <- 10
> S[,1] <- S.0-I[,1]
```

Now we are ready to run the spatial simulation.

```
> for (t in 2:niter) {
+    tmp <- local.dyn(t,S=S[,t-1],I=I[,t-1],b0=b0,b1=b1,mu=mu,N=N)
```

```
+    S[,t] <- D%*%tmp$S
+    I[,t] <- D%*%tmp$I
+ }
> save(list=c("S","I","b0","b1","mu","N","S.0"),file="cml.rda")
```

The first line in the loop advances the system's state within each patch independently. The next two lines disperse individuals between patches. Note that the results are saved to disk. You can retrieve them using

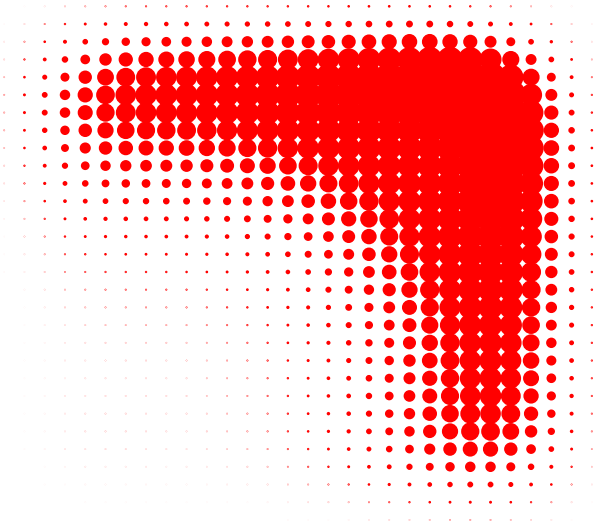
```
> load("cm1.rda")
```

To make a movie out of the results, try:

```
> x <- xy[,1]
> y <- xy[,2]
> for (k in 1:niter) {
+   symbols(x,y,fg=2,circles=I[,k],inches=0.1,bg=2,xlab="",ylab="")
+   Sys.sleep(.1)
+ }
```

Here is a snapshot from the end of the simulation. The size of the circles is proportional to the relative density of infected hosts in each patch.

```
> x <- xy[,1]
> y <- xy[,2]
> symbols(x,y,fg=2,circles=I[,niter],inches=0.1,bg=2,ann=F,xaxt='n',yaxt='n',bty='n')
```



Exercise 10. Have a look at the time series of infecteds at individual patches. How do they compare with the dynamics of a single patch?

Exercise 11. Explore the effects of changing the size of the arena from 30 by 30 to smaller sizes. How does this affect the dynamics? How does it affect persistence of the pathogen?

Exercise 12. Explore the effects of changing the dispersal fraction. How does it change the persistence and dynamics of the disease?

***Exercise 13.** The SIR coupled map lattice above is deterministic, but based on the chain binomial model: the local dynamics are the conditional expectation of the chain binomial model. Modify the map lattice to take account of stochasticity. Specifically, replace `local.dyn` with a stochastic map. Explore the dynamics and persistence of the disease with respect to (a) lattice size, (b) migration rate, and (c) contact rate, b.

Acknowledgements. Thanks to Ben Bolker, Ottar Bjørnstad, and Dave Smith for the use of source materials for this exercise.

References

D. J. D. Earn, P. Rohani, B. M. Bolker, & B. T. Grenfell (2000). 'A simple model for complex dynamical transitions in epidemics.' *Science* **287**:667–670.