

ELECTRONIC SUPPLEMENTARY MATERIAL

SPATIAL EPIDEMICS DYNAMICS: SYNCHRONIZATION


MODEL STUDENTS

Nicole Dumont, Melody Fong, Carolina Weishaar

CONTENTS

1	Introduction	1
2	Bifurcation Diagram	1
3	Deterministic Solution	1
4	Stochastic Simulation	5
4.1	Gillespie Algorithm	5
4.2	Adaptive Tau-Leaping Algorithm	9
5	Coherence Dependence on Parameters	13
5.1	3D Plot with Deterministic Solution	13
5.2	Four Panel Plot with Adaptive Tau	15

1 INTRODUCTION

This is a supplementary document for the report "Spatial epidemics dynamics: Synchronization" created for reproducibility. This supplement is written using the  package `knitr`. Other packages used are `deSolve`, `tikzDevice`, and `adaptivetau`.

```
library("deSolve")
library("tikzDevice")
library("adaptivetau")
```

Output is currently suppressed during compile in the interest of speed - some code chunks take hours to run - but code can still be run using the uncompiled document `SupplementaryMaterial.Rnw`.

2 BIFURCATION DIAGRAM

3 DETERMINISTIC SOLUTION

The SIR meta-patch vector field function, `SIRmeta.vector.field`, returns the values of $\frac{dS_i}{dt}$, $\frac{dI_i}{dt}$, and $\frac{dR_i}{dt}$ for all patches (i.e. $1 \leq i \leq n$) at a given time, given the values of S_i , I_i , and R_i for all patches and the values of parameters \mathcal{R}_0 (the basic reproductive number), μ (the death/birth rate), γ (the recovery rate), α (the strength of seasonal forcing), m (the mixing parameter) and EC (if **TRUE** use equal coupling, otherwise use nearest neighbour coupling).

It creates the beta matrix:

$$\beta(t) = \langle \beta \rangle (1 + \alpha \cos(2\pi t))M \quad (1)$$

where M is the mixing matrix. If equal coupling is used then

$$M = \begin{bmatrix} 1-m & \frac{m}{n-1} & \frac{m}{n-1} & \frac{m}{n-1} \\ \frac{m}{n-1} & 1 & \frac{m}{n-1} & \frac{m}{n-1} \\ \frac{m}{n-1} & \frac{m}{n-1} & 1 & \frac{m}{n-1} \\ \frac{m}{n-1} & \frac{m}{n-1} & \frac{m}{n-1} & 1 \end{bmatrix}$$

Otherwise nearest neighbour coupling is used.

$$M = \begin{bmatrix} 1-m & \frac{m}{2} & 0 & 0 & \dots & \frac{m}{2} \\ \frac{m}{2} & 1-m & \frac{m}{2} & 0 & & \vdots \\ 0 & \frac{m}{2} & 1-m & & & \\ 0 & 0 & & \ddots & & \\ \vdots & & & & \ddots & \frac{m}{2} \\ \frac{m}{2} & & & \dots & \frac{m}{2} & 1-m \end{bmatrix}$$

The proportional dynamics of a single patch in the meta-patch SIR model are given by

$$\begin{aligned} \frac{dS_i}{dt} &= \mu - S_i \sum_{j=1}^n \beta_{ij}(t) I_j - \mu S_i \\ \frac{dI_i}{dt} &= S_i \sum_{j=1}^n \beta_{ij}(t) I_j - \gamma I_i - \mu I_i \\ \frac{dR_i}{dt} &= \gamma I_i - \mu R_i \end{aligned} \tag{2}$$

```
SIRmeta.vector.field <- function(t,vars,parms=NULL) {
  ##parms should be of form (R_0,gamma,mu,alpha,m,EC)
  ##vars should be of form (S,I,R)
  with(as.list(c(parms,vars)), { #So that it can call variables stored in parms
    ##vars contains the values of S, I, and R at time t
    n <- length(vars)/3
    S <- vars[1:n]
    I <- vars[(n+1):(2*n)]
    R <- vars[(2*n+1):(3*n)]

    #Equal Coupling
    if (EC==TRUE){
      ##mixing matrix
      M <- matrix(m/(n-1),n,n)+(1-m*(1+1/(n-1)))*diag(n)
      ##beta matrix
      betam <- R_0*(gamma+mu)*(1+alpha*cos(2*pi*t))*M
    }else{
      #Nearest Neighbors
      ##mixing matrix
      M <- (1-m)*diag(n)
      M[row(M)%n==(col(M)+1)%n] <- m/2
      M[row(M)%n==(col(M)-1)%n] <- m/2
      ##beta matrix
      betam <- R_0*(gamma+mu)*(1+alpha*cos(2*pi*t))*M
    }
  })
}
```

```

dS <- NULL
dI <- NULL
dR <- NULL
for (i in 1:n){
  dS[i] <- mu*(1-S[i]) - S[i]*sum(betam[i,]*I) ##dS_i/dt
  dI[i] <- S[i]*sum(betam[i,]*I) - (mu+gamma)*I[i] ## dI_i/dt
  dR[i] <- gamma*I[i] -mu*R[i]
}
vec.fld <- c(dS=dS,dI=dI,dR=dR)
return(list(vec.fld))
})
}

```

The `run.soln` function returns the values of S_i , I_i , and R_i for all patches at times given in `times`, given the initial conditions, the vector field function, and parameter to pass to the vector field function. It also returns the value of "coherence" at all times. Here coherence is the coefficient of variation, also called the relative standard deviation, of $\{I_i(t)\}_{i=1}^n$ at all times. It is the standard deviation of these values divided by the mean. It is a measure of dispersion.

$$\begin{aligned}
c_v(t) &= \frac{SD(I_i(t))}{\overline{I_i(t)}} \\
&= \frac{\sqrt{\frac{1}{n} \sum_{i=1}^n (I_i(t) - \overline{I_i(t)})^2}}{\overline{I_i(t)}}
\end{aligned} \tag{3}$$

```

## Get Solutions and coherence measure
run.soln <- function(n,ic=c(S=rep(0,n),I=rep(0,n),R=rep(1,n)),tmax=1,
                      times=seq(0,tmax,
                                by=tmax/1000),
                      func,parms) {
  soln <- ode(ic,times,func,parms)

  #Measure of coherence: the coefficient of variation/relative standard deviation of
  #all the patches at each time step
  coherence <- rep(0,length(times))
  for(tims in 1:length(times)){
    coherence[tims]<- sd(soln[tims,paste0('I',1:n)])/mean(soln[tims,paste0('I',1:n)])
    if(is.nan(coherence[tims])){
      coherence[tims]<- 0
    }
  }
  return(list("times"=times, "soln" = soln, "coherence"= coherence))
}

```

Function `plot.SIRmeta` uses `run.soln` to get the solution of the deterministic model with the given initial conditions and parameters, and plots the results. These functions are separate so that the solution can be found without plotting.

```

#Given ics and paramters, plot the det soln of the SIR meta model to pdf
plot.SIRmeta <- function(n,ic=c(S=rep(0,n),I=rep(0,n),R=rep(1,n)),tmax=1,
                          parms,...){
  ##parms should be of form (R_0,gamma,mu,alpha,m,EC)

```

```

with(as.list(c(parms)), { #So that it can call variables stored in parms
## get solutions
run <- run.soln(n=n,ic=ic,tmax=tmax,
               func=SIRmeta.vector.field,
               parms=parms)
times<-run$times
soln<-run$soln
coherence<-run$coherence

#Print to pdf using tikz
filename <- paste0("det",if(EC==TRUE) "EC" else "NN", "R0", R_0, "m", m, ".tex")
tikz(filename,standAlone=TRUE,width=6,height=6)

#leave room on right margin for coherence label
par(mar = c(5,5,2,5))
#Calculates I* to estimate how high the y-axis needs to be
IEE <- (1 - 1/R_0)*mu/gamma
## draw box for plot
plot(0,0,xlim=c(0,tmax),ylim=c(0,IEE*5),type="n",xlab="Time (years)",
     ylab="Prevalence (I)",las=1)

#Plot I for all patches with different colours and lty
for (i in 1:n){
  lines(times,soln[,paste0('I',i)],col=i,lty=i)
}

#Plot coherence over top
par(new = T)
plot(times,coherence,type='l',lwd=2,axes=F,xlab=NA, ylab=NA,ylim=c(0,1))
axis(side=4)
#Label left axis
mtext(side = 4, line = 3, 'Coherence')

#Make legend
labels <- c(1:n)
for(i in 1:n){
  labels[i]<-paste('$I_{',i,'}$',collapse=NULL)
}
legend("topright",legend=labels,col=1:n,lty=1,bty='n')
dev.off()
tools::texi2dvi(filename,pdf=T)
})
}

```

The following chunk of code uses `plot.SIRmeta` to plot solutions to the meta-patch model for different parameters and with initial conditions within $\pm 30\%$ of the endemic equilibrium solution values. This is so that the solutions converge quickly to a stable solution. The endemic equilibrium is

$$\begin{aligned}
 S^* &= \frac{1}{\mathcal{R}_0} \\
 I^* &= \frac{\mu}{\gamma} \left(1 - \frac{1}{\mathcal{R}_0} \right) \\
 R^* &= 1 - S^* - I^*
 \end{aligned} \tag{4}$$

```

#Parameters
R_0 <- 17    ##basic reproduction rate (if the system had no forcing)
gamma <- 365/13  ##inverse of mean infectious period
mu <- 1/50    ##death and birth rate
alpha <- 0.1  ##strength of seasonal forcing
n <- 10      ## Number of patches
m <- 0.2     ##Connectivity matrix parameter
EC <- FALSE  ##if true use equal coupling, if false use nearest neighbors

##Initial conditions
#Want ICs near EE for stability. Find EE values:
SEE <- 1/R_0
IEE <- (1 - 1/R_0)*mu/gamma
REE <- 1-SEE-IEE
#Use random initials conditions within +/-30% of EE values
set.seed(34) #For reproducibility
percent <- 0.3
S0 <- runif(SEE*(1-percent), SEE*(1+percent), n)
I0 <- runif(IEE*(1-percent), IEE*(1+percent), n)
R0 <- 1-S0-I0

tmax<-15

#### Plotting lots of solutions to pdf ####
plot.SIRmeta(n=n,ic=c(S=S0,I=I0,R=R0),tmax=tmax,
             parms=c(R_0=R_0,gamma=gamma,mu=mu,alpha=alpha,m=m,EC=EC))

EC<-TRUE
plot.SIRmeta(n=n,ic=c(S=S0,I=I0,R=R0),tmax=tmax,
             parms=c(R_0=R_0,gamma=gamma,mu=mu,alpha=alpha,m=m,EC=EC))

EC<-FALSE
m<-0.01
plot.SIRmeta(n=n,ic=c(S=S0,I=I0,R=R0),tmax=tmax,
             parms=c(R_0=R_0,gamma=gamma,mu=mu,alpha=alpha,m=m,EC=EC))

```

4 STOCHASTIC SIMULATION

Stochastic simulations of the model were also used.

4.1 Gillespie Algorithm

The Gillespie algorithm is a method of exactly simulating a stochastic process. Consider a process involving transitions between different states. The rate for a given transition i is labelled a_i . The total rate is

$$a_0 = \sum a_i \quad (5)$$

If we assume that the time spent in a given state is exponentially distributed then by sampling from an exponential distribution with rate a_0 (using `R`'s `rexp` function) we can simulate how much time was spent in a given state before the next transition.

The probability that transition i occurred is $\frac{a_i}{a_0}$. By sampling from a uniform distribution over $[0, a_0]$ we can simulate which transition occurred in the following way: If the sampled value is in the i^{th} interval from the list

$$[0, a_1), [a_1, a_1 + a_2), \dots [a_1 + a_2 + \dots + a_{i-1}, a_1 + a_2 + \dots + a_i), \dots \quad (6)$$

than transition i occurred.

For this problem, the states are the values of $\{(S_i, I_i, R_i)\}_{i=1}^n$. The state space is all possible set of n triplets of non-negative integers. Possible transitions are:

- A birth ($S_i \rightarrow S_i + 1$)
- A transmission ($S_i \rightarrow S_i - 1$ and $I_i \rightarrow I_i + 1$)
- A recovery ($I_i \rightarrow I_i - 1$ and $R_i \rightarrow R_i + 1$)
- A death of a susceptible ($S_i \rightarrow S_i - 1$)
- A death of an infected ($I_i \rightarrow I_i - 1$)
- A death of a recovered ($R_i \rightarrow R_i - 1$)

for $1 \leq i \leq n$. The transition rates are:

- The birth rate $\mu(S_i + I_i + R_i)$
- The transmission rate $\frac{S_i}{S_i + I_i + R_i} \sum_{j=1}^n \beta_{ij}(t) I_j$
- The recovery rate γI_i
- The susceptible death rate μS_i
- The infected death rate μI_i
- The recovered death rate μR_i

for $1 \leq i \leq n$.

The function `SIRmeta.Gillespie` implements a stochastic simulation of the meta-patch SIR model using the Gillespie algorithm. It returns a list of times (starting at zero and ending near the input variable `tmax` with non-uniform step sizes) and the values of S_i , I_i , and R_i for all patches at these times given initial conditions and parameter values.

```
SIRmeta.Gillespie <- function(tmax,ic=c(S0,IO,R0),
                             parms=c(R_0=2,gamma=0.25,mu=4e-5,
                                       alpha=0.1,m=0.2,EC=TRUE)){
  times <- 0

  n <- length(ic)/3
  #S, I, R matrices: column index is patch index and rows (added later in while loop)
  #are values at different time steps
  S <- matrix(0,1,n)
  I <- matrix(0,1,n)
  R <- matrix(0,1,n)
  #Initial Coniditions
  S[1,] <- ic[1:n]
  I[1,] <- ic[(n+1):(2*n)]
  R[1,] <- ic[(2*n+1):(3*n)]

  index <- 1
  while(times[index] < tmax){
    #A matrix of all rates
    rates <- matrix(0,n,6)
    #the first column is all birth rates
    rates[,1] <- mu*(S[index,]+I[index,]+R[index,])
```

```

#third column are recovery rates
rates[,3] <- gamma*I[index,]
#last three columns are death rates from S and I
rates[,4] <- mu*S[index,]
rates[,5] <- mu*I[index,]
rates[,6] <- mu*R[index,]
#Second column are transmission rates
#Equal Coupling
if(EC==TRUE){
  M <- matrix(m/(n-1),n,n)+(1-m*(1+1/(n-1)))*diag(n) ##connectivity matrix
  betam <- R_0*(gamma+mu)*(1+alpha*cos(2*pi*times[index]))*M ##beta matrix
}else{
  #Nearest Neighbors
  M <- (1-m)*diag(n)
  M[row(M)%%n==(col(M)+1)%%n] <- m/2
  M[row(M)%%n==(col(M)-1)%%n] <- m/2
  betam <- R_0*(gamma+mu)*(1+alpha*cos(2*pi*times[index]))*M ##beta matrix
}
rates[,2] <- S[index,]*colSums(t(betam[1:n,])*I[index,])/(S[index,]+I[index,]+R[index,])
totalrate <- sum(rates)

#Getting a timestep using R's exp distribution
timestep <- rexp(n=1, rate= totalrate)
times <- rbind(times, times[index]+timestep)

#Adding new row to S, I, R
S <- rbind(S, S[index,])
I <- rbind(I, I[index,])
R <- rbind(R, R[index,])

#Getting which compartment transistion occurred
#TO DO: Do a bisection search (or some other faster method) instead
#Get a random number from a uniform distribution
randomvar <- runif(n=1,min=0,max=totalrate)
interval <- 1
#Make the rates matrix into a vector for convinence
rateslisted <- c(0,as.vector(t(rates)))
#Find which interval randomval is in from list:
#[0,a_1), [a_1, a_1+a_2), etc
#If in interval [a_1 + .. + a_(i-1), a_1 + ... + a_i),
#transition given by rate a_i occurred
while(interval < (6*n)){
  if(randomvar >= sum(rateslisted[1:interval]) &
    randomvar < sum(rateslisted[1:(interval+1)])){
    break
  }else{
    interval<- interval+1
  }
}
}
#Get row, col indices of a_i in rates matrix
#row index says in which patch a transition occurred
#col index says what type of transition occurred
colInd <- interval%%6

```

```

rowInd <- ceiling(interval/6)
if(colInd ==1){ ##A birth occurred
  S[index+1,rowInd] <- S[index,rowInd]+1
}else if(colInd ==2){ ##An infection occurred
  S[index+1,rowInd] <- S[index,rowInd]-1
  I[index+1,rowInd] <- I[index,rowInd]+1
}else if(colInd ==3){ ##A recovery occurred
  I[index+1,rowInd] <- I[index,rowInd]-1
  R[index+1,rowInd] <- R[index,rowInd]+1
}else if(colInd ==4){ ##A death in S occurred
  S[index+1,rowInd] <- S[index,rowInd]-1
}else if(colInd ==5){ ##A death in I occurred
  I[index+1,rowInd] <- I[index,rowInd]-1
}else if(colInd ==0){ ##A death in R occurred
  R[index+1,rowInd] <- R[index,rowInd]-1
}

index <- index+1
}
return(cbind(times, S, I, R))
}

```

The following code uses the `SIRmeta.Gillespie` function to plot a stochastic simulation of the model.

```

#For reproducibility set the seed
set.seed(899)

tmax <- 10
n <- 10

#Parameters
R_0 <- 17 ##basic reproduction rate (if the system had no forcing)
gamma <- 365/13 ##inverse of mean infectious period
mu <- 1/50 ##death and birth rate
alpha <- 0.1 ##strength of seasonal forcing
n <- 10 ## Number of patches
m <- 0.5 ##Connectivity matrix parameter
EC <- TRUE ##if true use equal coupling, if false use nearest neighbors

##Initial conditions
#Want ICs near EE for stability. Find EE values:
SEE <- 1/R_0
IEE <- (1 - 1/R_0)*mu/gamma
REE <- 1-SEE-IEE
#Sample from initials conditions within +/-10% of EE values
#convert from proportions to numbers and convert to integers
percent <- 0.1
population <- 3000
possibleS0 <- seq(as.integer(population*SEE*(1-percent)), as.integer(population*SEE*(1+percent)),1)
possibleI0 <- seq(as.integer(population*IEE*(1-percent)), as.integer(population*IEE*(1+percent)),1)
S0 <- sample(possibleS0, n,replace=TRUE)
I0 <- sample(possibleI0, n,replace=TRUE)
R0 <- population-S0-I0

```



```

#Get simulation results
results <- SIRmeta.Gillespie(tmax,ic=c(S0,I0,R0),
                             parms=c(R_0,gamma,mu,alpha,m,EC))

times<-results[,1]
I <- results[, (n+2):(2*n+2)]

#Save data for later use and plot to pdf and png
write.table(results,file="GillECR017m0.2.pdf")
pdf(file="images/GillECR017m0.2.pdf")
copy <- dev.cur()
png(file="images/GillECR017m0.2.png")
dev.control("enable")

## draw box for plot
plot(0,0,xlim=c(0,tmax),ylim=c(0,population*IEE*2),xlab="Time (years)",
     ylab="Prevalence (I)")
#Plot all I_i using a differnt line colour and lty for each patch
for (i in 1:n) {
  lines(times, I[,i],col=i,lwd=0.2)
}
#Make legend
legend("topright",legend=1:n,fill="white",title="Patch",col=1:n,lty=1,cex=0.5)

dev.copy(which=copy)
dev.off
dev.off

```

This code runs very slowly for large populations. For a 250,000 population it took 3 hours. This method for simulating the stochastic realization of the model is not practical.

4.2 Adaptive Tau-Leaping Algorithm

To simulate a stochastic process efficiently, an approximate method can be used. The adaptive tau-leaping algorithm works by

The `runadapttau` function contains a list of all possible transitions and `rateFct`, a function that returns the values of all transition rates given the time and current state. `runadapttau` passes these, along with the input initial conditions and parameters, to the `ssa.adaptivetau` function. It returns a list of times (starting at zero and ending near input `tmax` with non-uniform step size) and the values of the state at all these times.

```

runadapttau <-function(tmax,ic=c(S0,I0,R0),
                      params=c(R_0,gamma,mu,alpha,m,EC)){
  #Get the # of patches from the ics
  n <- length(ic)/3

  #TO DO: This should be changed so that it works with different n values
  #and is less ugly
  #A list of all possible transitions
  transitions <- list(c(S1 = +1), c(S1 = -1, I1= +1), c(I1 = -1, R1= +1), c(S1 = -1),
                     c(I1 = -1), c(R1 = -1),
                     c(S2 = +1), c(S2 = -1, I2= +1), c(I2 = -1, R2= +1), c(S2 = -1),
                     c(I2 = -1), c(R2 = -1),
                     c(S3 = +1), c(S3 = -1, I3= +1), c(I3 = -1, R3= +1), c(S3 = -1),

```

```

c(I3 = -1), c(R3 = -1),
c(S4 = +1), c(S4 = -1, I4= +1), c(I4 = -1, R4= +1), c(S4 = -1),
c(I4 = -1), c(R4 = -1),
c(S5 = +1), c(S5 = -1, I5= +1), c(I5 = -1, R5= +1), c(S5 = -1),
c(I5 = -1), c(R5 = -1),
c(S6 = +1), c(S6 = -1, I6= +1), c(I6 = -1, R6= +1), c(S6 = -1),
c(I6 = -1), c(R6 = -1),
c(S7 = +1), c(S7 = -1, I7= +1), c(I7 = -1, R7= +1), c(S7 = -1),
c(I7 = -1), c(R7 = -1),
c(S8 = +1), c(S8 = -1, I8= +1), c(I8 = -1, R8= +1), c(S8 = -1),
c(I8 = -1), c(R8 = -1),
c(S9 = +1), c(S9 = -1, I9= +1), c(I9 = -1, R9= +1), c(S9 = -1),
c(I9 = -1), c(R9 = -1),
c(S10 = +1), c(S10 = -1, I10= +1), c(I10 = -1, R10= +1), c(S10 = -1),
c(I10 = -1), c(R10 = -1))

#A function that outputs the transition rates given parameters, S,I,R, and t
rateFct <- function(x,params,time){
  #x should be of form: x=c(S,I,R)
  #parms should be of form: paarms=c(R_0,gamma,mu,alpha,m,EC)

  with(
    as.list(c(params)),
    {
      n <- length(x)/3
      S <- as.vector(x[1:n])
      I <- as.vector(x[(n+1):(2*n)])
      R <- as.vector(x[(2*n+1):(3*n)])

      #A matrix of all rates
      rates <- matrix(0,n,6)
      #the first column is all birth rates
      rates[,1] <- mu*(S+I+R)
      #thrid column are recovery rates
      rates[,3] <- gamma*I
      #last three columns are death rates from S and I
      rates[,4] <- mu*S
      rates[,5] <- mu*I
      rates[,6] <- mu*R
      #Second column are transmission rates
      #Equal Coupling
      if (EC==TRUE){
        M <- matrix(m/(n-1),n,n)+(1-m*(1+1/(n-1)))*diag(n) ##connectivity matrix
        betam <- R_0*(gamma+mu)*(1+alpha*cos(2*pi*time))*M ##beta matrix
      }else{
        #Nearest Neighbors
        M <- (1-m)*diag(n)
        M[row(M)%n==(col(M)+1)%n] <- m/2
        M[row(M)%n==(col(M)-1)%n] <- m/2
        betam <- R_0*(gamma+mu)*(1+alpha*cos(2*pi*time))*M ##beta matrix
      }
      rates[,2] <- S*colSums(t(betam[1:n,])*I)/(S+I+R)
    }
  )
}

```

```

    return(as.vector(t(rates)))
  }
)
}

init.values <- c(S=S0,I=I0,R=R0)
#Get approximate simulation
simResults <- ssa.adaptivetau(init.values,transitions, rateFct, params, tf=tmax)
I <- simResults[, (12:21)]

#Measure of coherence: the coefficient of variation/relative
#standard deviation of all the patches at each time step
coherence <- rep(0,length(simResults[,1]))
for(tims in 1:length(simResults[,1])){
  coherence[tims]<- sd(I[tims,])/mean(I[tims,])
  #In case the mean is zero (and thus the coefficient of variation becomes nan or inf)
  if(is.nan(coherence[tims]) | is.infinite(coherence[tims])){
    coherence[tims]<- 0
  }
}

#Is the disease extinct in any of the patches? Store 1s for yes, 0 for nos
localextinction<-c(1:n)
for(i in 1:ncol(I)){
  if(mean(I[(nrow(I)-50):nrow(I),i])<= IEE*population*0.3){
    localextinction[i]<-1
  }else{
    localextinction[i]<-0
  }
}
localext<-mean(localextinction)

#If the disease is extinct in all patches there is global extinction
if(all(localextinction==1)){
  globalextinction<-1
}else{
  globalextinction<-0
}

return(list("sim" = simResults, "coherence"= coherence,"localext"=localext,"globalext"=globalextinction))
}

```

The following code uses `runadapttau` to approximately simulate the meta-patch SIR model and plot the results.

```

#Parameters
R_0 <- 17 ##basic reproduction rate (if the system had no forcing)
gamma <- 365/13 ##inverse of mean infectious period
mu <- 1/50 ##death and birth rate
alpha <- 0.1 ##strength of seasonal forcing
n <- 10 ## Number of patches
m <- 0.2 ##Connectivity matrix parameter
EC <- TRUE ##if true use equal coupling, if false use nearest neighbors
params<-c(R_0,gamma,mu,alpha,m,EC)

```

```

##Initial conditions
#Want ICs near EE for stability. Find EE values:
SEE <- 1/R_0
IEE <- (1 - 1/R_0)*mu/gamma
REE <- 1-SEE-IEE
#Sample ics from points +/-10% from the EE values
percent <- 0.1
population <- 5000000
possibleS0 <- seq(as.integer(population*SEE*(1-percent)),
                  as.integer(population*SEE*(1+percent)),1)
possibleI0 <- seq(as.integer(population*IEE*(1-percent)),
                  as.integer(population*IEE*(1+percent)),1)
S0 <- sample(possibleS0, n,replace=TRUE)
I0 <- sample(possibleI0, n,replace=TRUE)
R0 <- population-S0-I0
init.values <- c(S=S0,I=I0,R=R0)

tmax <- 15

run <- runadaptau(tmax,init.values,params)

simResults <- run$sim
coherence <- run$coherence

##Saving the data and the print plot to pdf
write.table(run,file="adaptauECR017m0.01j.csv")
pdf(file="images/adaptauECR017m0.01j.pdf")

## draw box for plot
#leave room on right margin for Coherence label
par(mar = c(5,5,2,5))
#Solutions should oscillate around IEE so ylim should go to around twice
#that so solns are in the middle of plot
plot(0,0,xlim=c(0,tmax),ylim=c(0,population*IEE*4),xlab="Time (years)",
     ylab="Prevalence (I)")
##plot I_i for each patch
j<-1:n
for (i in 1:n) {
  if(i%%8==1) j[i] <- i+3 #indices so that black and solid lines are
  #skipped; saved for legend
  lines(simResults[,1], simResults[,11+i],col=j[i],lty=j[i],lwd=0.5)
  # use a different line colour and line type for each patch
}

#Currently unused code for plotting the det soln over this sim
# draw.single.soln(n,ic=c(S=SEE,I=IEE,R=(1-SEE-IEE)),tmax=tmax,
#                  func=SIR.vector.field,
#                  parms=c(R_0,gamma,mu,alpha,population))

#Plot coherence onto of pre-existing plot
par(new = T)
plot(simResults[,1],coherence,type='l',axes=F,xlab=NA, ylab=NA, ylim=c(0,1))
#y axis and coherence label on right axis

```

```

axis(side=4)
mtext(side = 4, line = 3, 'Coherence')

#Make legends
legend("topright",legend=1:n,title="Patch",bg='white',col=j,lty=j,lwd=0.5,cex=0.8)
legend("bottomright", legend="Relative Standard Deviation",bg='white',
      col=1,lty=1, cex=0.8)

dev.off()

```

5 COHERENCE DEPENDENCE ON PARAMETERS

The dependence of a meta-patch becoming "coherent" on the parameters used in the model was explored. A system is considered coherent when the mean value of $c_v(t)$ from $t = 9$ years to $t = 10$ years is below a threshold value. We will refer to this value as the coherence threshold.

A list of \mathcal{R}_0 (the basic reproductive number) values from 1 to 30 was made, along with a list of m (the mixing parameter) values. For each possible set of \mathcal{R}_0 and m values from the lists, a number of different initial conditions near the EE solution were generated. A solution or simulation of the meta-patch SIR model with 10 patches was gotten for each set of initial conditions. It was stored whether or not each solution or simulation became coherent; a one was saved if it did and a zero was saved otherwise. By taking the mean of these values for all runs, the probability of becoming coherent for a given \mathcal{R}_0 and m value was found.

5.1 3D Plot with Deterministic Solution

A 3D plot showing the probability of coherence vs \mathcal{R}_0 and m is created in the function `coherence3DPlot`. If parameter `DetOrAdap` is TRUE this is done using deterministic solutions to the model, created by the `run.soln` function. Otherwise, this is done using adaptive tau simulations in the `runadapttau` function.

```

coherence3DPlot <- function(params=c(gamma,mu,alpha,EC),
                                R_0list,mlist,nsample,DetOrAdap,population=1){
  with(
    as.list(c(params)),
    {
      n <- 10 ## Number of patches
      ##Param for Initial conditions
      percent <- 0.1

      tmax <- 10
      gridsize<-length(mlist)
      gridsizeR_0 <- length(R_0list)
      avgCoherence<- matrix(0, gridsize,gridsizeR_0)

      index2 <- 1
      for (R_0 in R_0list){
        index1 <- 1
        #Want ICs near EE for stability. Find EE values:
        #Note: They depend on R_0
        SEE <-1/R_0
        IEE <-(1 - 1/R_0)*mu/gamma

        if(!DetOrAdap){ #Only need these for adapttau sim
          possibleS0 <- seq(as.integer(population*SEE*(1-percent)),
                            as.integer(population*SEE*(1+percent)),1)

```

```

possibleIO <- seq(as.integer(population*IEE*(1-percent)),
                  as.integer(population*IEE*(1+percent)),1)
}

for (m in mlist){
  coherenceProb <- rep(0, nsample)
  for (sampler in 1:nsample){
    #Get det soln nsample times with different ICs
    #Take mean of coherence near end of soln
     #(since coherence can be somewhat oscillatory)

    if(DetOrAdap){
      S0 <- runif(n,SEE*(1-percent),SEE*(1+percent))
      I0 <- runif(n,IEE*(1-percent),IEE*(1+percent))
      R0 <- 1-S0-I0
      coherenceRun <- run.soln(n,ic=c(S=S0,I=I0,R=R0),tmax=tmax,
                               func=SIRmeta.vector.field,
                               parms=c(R_0,gamma,mu,alpha,m,EC))$coherence
    }else{
      S0 <- sample(possibleS0, n,replace=TRUE)
      I0 <- sample(possibleIO, n,replace=TRUE)
      R0 <- population-S0-I0
      coherenceRun <- runadapttau(tmax,init.values=c(S=S0,I=I0,R=R0),
                                  params=c(R_0,gamma,mu,alpha,m,EC))$coherence
    }

    mean <- mean(coherenceRun[length(coherenceRun)-(100:0)])
    if(mean<threshold){
      coherenceProb[sampler] <- 1
    }else{
      coherenceProb[sampler] <- 0
    }
  }
  #Take mean of all coherence for all runs with these parameters
  avgCoherence[index1,index2]<- mean(coherenceProb)
  index1<-index1+1
}
index2<-index2+1
}

det3D <- cbind.data.frame(R_0list,mlist,avgCoherence)
write.table(det3D,file="det3DNN.csv")

#3D plot:
pdf(file="images/det3DNN.pdf")
#Making a colour gradient for the plot
nbcol <- 100
#You can change start and end to get different parts of the rainbow
color <- rev(rainbow(nbcol, start = 3/6, end = 3.5/6))
#The colour gradient will correspond to the 'height' of the plot
zcol <- cut(t(avgCoherence), nbcol)
persp(R_0list,mlist, t(avgCoherence), theta = 30, phi = 30, expand = 0.5,
      col = color[zcol],ltheta = 120, shade = 0.75, ticktype = "detailed",

```

```

      xlab = "Basic Reproduction Number", ylab = "Mixing Parameter",
      zlab = "Probability of Coherence")
    dev.off()
  })
}

```

The following code calls `coherence3DPlot` to make a plot.

```

#Use the det soln if TRUE or adapttau sim if FALSE
DetOrAdap <- TRUE

#The value at which things are considered coherent
threshold <- 0.05
#Parameter values for grid
gridsizeR_0 <- 10
gridsizeM <- 10
nsample <- 1 #Number of runs per grid pt
mlist <- seq(0.0001, 0.15, (0.15-0)/(gridsizeM-1))
R_0list <- seq(1,30,(30-1)/(gridsizeR_0-1))

#Parameters
gamma <- 365/13 ##inverse of mean infectious period
mu <- 1/50 ##death and birth rate
alpha <- 0.1 ##strength of seasonal forcing
EC <- FALSE ##if true use equal coupling, if false use nearest neighbors

coherence3DPlot(params=c(gamma=gamma,mu=mu,alpha=alpha,EC=EC),
                R_0list,mlist,nsample,DetOrAdap,population=1)

```

5.2 Four Panel Plot with Adaptive Tau

Making grids of coherence (needs `runadapttau` fct from earlier chunk to run):

```

#The value at which things are considered coherent
threshold <- 0.15
#Parameter values for grid
gridsizeR_0 <- 100
gridsizeM <- 4
nsample <- 15
mlist <- c(0.2,0.4,0.6,0.8)
R_0list <- seq(1.1,30,(30-1)/(gridsizeR_0-1))

#Parameters
gamma <- 365/13 ##inverse of mean infectious period
mu <- 1/50 ##death and birth rate
alpha <- 0.1 ##strength of seasonal forcing
n <- 10 ## Number of patches
EC <- FALSE ##if true use equal coupling, if false use nearest neighbors

##Params for Initial conditions
percent <- 0.1
population <- 250000

```

```

tmax <- 10

avgCoherence<- matrix(0, gridsizem,gridsizemR_0)
avgLE<- matrix(0, gridsizem,gridsizemR_0)
avgGE<- matrix(0, gridsizem,gridsizemR_0)

index2 <- 1
for (R_0 in R_0list){
  index1 <- 1
  #Want ICs near EE for stability. Find EE values:
  #Note: They depend on R_0
  SEE <-1/R_0
  IEE <-(1 - 1/R_0)*mu/gamma
  possibleS0 <- seq(as.integer(population*SEE*(1-percent)),
                    as.integer(population*SEE*(1+percent)),1)
  possibleI0 <- seq(as.integer(population*IEE*(1-percent)),
                    as.integer(population*IEE*(1+percent)),1)

  for (m in mlist){
    params<-c(R_0=R_0,gamma=gamma,mu=mu,alpha=alpha,m=m,EC=EC)
    coherenceProb <- rep(0, nsample)
    localextProb <- rep(0, nsample)
    globalextProb <- rep(0, nsample)
    for (sampler in 1:nsample){
      #Run stochastic sim nsample times with different ICs
      S0 <- sample(possibleS0, n,replace=TRUE)
      I0 <- sample(possibleI0, n,replace=TRUE)
      R0 <- population-S0-I0
      init.values <- c(S=S0,I=I0,R=R0)

      #Get results
      run <- runadaptau(tmax,init.values,params)
      localextProb[sampler] <- run$localext
      globalextProb[sampler] <- run$globalext

      coherenceRun <- run$coherence
      #Take mean of coherence near end of sim
       #(since coherence is very oscillatory)
      mean <- mean(coherenceRun[length(coherenceRun)-(100:0)])
      if(mean<threshold){
        coherenceProb[sampler] <- 1
      }else{
        coherenceProb[sampler] <- 0
      }
    }
  }
  #Take mean of all coherence for all runs with these parameters
  #And mean of local & global ext
  avgCoherence[index1,index2]<- mean(coherenceProb)
  avgLE[index1,index2]<- mean(localextProb)
  avgGE[index1,index2]<- mean(globalextProb)
  index1<-index1+1
}
index2<-index2+1

```



```

}

#Save data
#Adaptau4pan <- cbind.data.frame(R_Olist,mlist,avgCoherence)
write.table(avgCoherence,file="Adaptau4pan.csv")

#Save plot to pdf
if(EC==TRUE){pdf(file="images/FourPanelEC.pdf")}
}else{pdf(file="images/FourPanelNN.pdf")}}

#Get bifur data for backgrounds
bfd <- read.table("bifurcation.dat",
                  col.names=c("time","S","I","R0","log10S","log10I"))

#Get max probs for y lim
maxExt <- max(avgLE,avgGE)
maxCoh <- max(avgCoherence)

#To fit bifur data to probability plot
bfdI <- maxExt*(1.1/max(bfd$I))*bfd$I

#leave room on outside margins for labels on righth, left and bottom
#par(mar = c(5,5,2,5))
par(oma=c(4,4,2,4))
#A four panel plot
par(mfrow=c(2,2))
par(mar = c(2, 3, 1, 2))
par(tcl = -0.25)
for(ind in 1:4){
  #Make an empty plot
  plot(0,0,xlim=c(1,30),
        ylim=c(0,maxExt*1.1),xlab=NA, ylab=NA,type='n')

  #Plot the bifur background
  points(bfd$R0,bfdI, pch=20, col="grey90", xpd=FALSE)
  #Plot LE, GE probs
  lines(R_Olist, avgLE[ind,],col=4)
  lines(R_Olist, avgGE[ind,],col=2)
  #Plot coherence prob overtop of current panel
  par(mfg = c(ceiling(ind/2),(ind-1)%2 +1))
  plot(R_Olist, avgCoherence[ind,],type='l',col=3,axes=F,xlab=NA, ylab=NA,ylim=c(0,maxCoh*1.1))
  axis(side=4)
  #Label m values on each plot
  mtext(paste0("m=",mlist[ind]), side = 3, line = -2, adj = 0.1, cex = 0.8, col = 1)
}

#Axes labels:
mtext("Basic Reproduction Number", side = 1, outer = TRUE, line = 1)

#Left axis label has multiple colors
mtext(expression("Probability of " * phantom("Local/ Global Extinction")),
        side = 2, outer = TRUE, line = 1, col = 1)
mtext(expression(phantom("Probability of ") * "Local" * phantom("/ Global Extinction")),

```

```

    side = 2, outer = TRUE, line = 1, col = 4)
mtext(expression(phantom("Probability of Local")*"/"*phantom(" Global Extinction")),
    side = 2, outer = TRUE, line = 1, col = 1)
mtext(expression(phantom("Probability of Local/")*" Global"*phantom(" Extinction")),
    side = 2, outer = TRUE, line = 1, col = 2)
mtext(expression(phantom("Probability of Local/ Global")*" Extinction"),
    side = 2, outer = TRUE, line = 1, col = 1)

mtext("Probability of Coherence", side = 4, outer = TRUE, line = 1, col = 3)

dev.off()

```