Oikos **OIK-01074**

Zeilinger, A. and Daugherty, M. P. 2013. Vector preference and host defense against infection interact to determine disease dynamics. – Oikos 000: 000–000.

Appendix 1

Derivation of R_0 equations

We calculated the basic reproduction number, R_0, for the SEI and single-patch SEIC models (system 1 and 3 in main text) using the next generation method (Diekmann et al. 1990, van den Driessche and Watmough 2002). Given that the assumptions of Diekmann et al. (1990) and van den Driessche and Watmough (2002) are met, R_0 is defined as the spectral radius (dominant eigenvalue) of a "next generation operator," $\mathbf{FV}^{-1}$, where **F** is a matrix representing the rate of appearance of new infections among all model compartments, which we call the accumulation functions, and **V** is a matrix representing the leaving rates from all compartments, or the loss functions. Both **F** and **V** are *m* x *m* matrices, where *m* is the number of compartments in which new infections arise. The matrices are populated from the partial derivatives of the compartmental equations; following the notation of van den Driessche and Watmough (2002):

$$\boldsymbol{F} = \left[\frac{\partial F_i(x_0)}{\partial x_j}\right] \qquad \boldsymbol{V} = \left[\frac{\partial V_i(x_0)}{\partial x_j}\right] \tag{A1}$$

where x_j are the number of individuals in the *j*th compartment, with $i, j = 1,\ldots, n$ number of infected compartments, and x_0 is the disease free equilibrium (DFE).

For the SEI model, system 1, only the exposed host $\left(\frac{dE}{dt}\right)$ and infected vector $\left(\frac{dV}{dt}\right)$ equations contribute non-zero elements to the accumulation matrix, **F**, because these are the only compartments where new infections arise. Likewise, the exposed host, infected host $\left(\frac{dI}{dt}\right)$, and infective vector equations contribute non-zero elements to the loss matrix, **V**. Taking the partial derivatives of these equations, with respect to each infected compartment, produces the matrices

$$\boldsymbol{F} = \begin{pmatrix} 0 & 0 & \beta \\ 0 & 0 & 0 \\ 0 & \dfrac{\alpha p T}{N} & 0 \end{pmatrix} \qquad \boldsymbol{V} = \begin{pmatrix} \delta + \gamma & 0 & 0 \\ -\delta & \gamma & 0 \\ 0 & 0 & \mu \end{pmatrix} \tag{A2}$$

and the dominant eigenvalue of **FV**$^{-1}$ is the R_0 equation (Eq. 2 in text)

$$R_0 = \sqrt{\frac{\alpha\beta\delta p T}{\mu(\delta + \gamma)N}}. \tag{A3}$$

For the SEIC single-patch model, the carrier equation $\left(\frac{dC}{dt}\right)$ contributes to the loss matrix in addition to compartments described above. Then the **F** and **V** matrices are

$$\begin{gathered} \boldsymbol{F} = \begin{pmatrix} 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \dfrac{a\alpha T}{N} & \dfrac{a\alpha p T}{N} & 0 \end{pmatrix} \\ \boldsymbol{V} = \begin{pmatrix} \delta + \gamma & 0 & 0 & 0 \\ -\delta q & \gamma & 0 & 0 \\ -\delta(1-q) & 0 & \gamma & 0 \\ 0 & 0 & 0 & \mu \end{pmatrix} \end{gathered} \tag{A4}$$

and the dominant eigenvalue of **FV**$^{-1}$ is the R_0 equation (Eq. 4 in text)

$$R_0 = \sqrt{\frac{a\alpha\beta\delta(p + q - pq)T}{\gamma\mu(\delta + \gamma)N}}. \tag{A5}$$

As noted by van den Driessche and Watmough (2002), the square root arises in the R_0 equations because it takes two generations (one vector and one host generation) for either an infected vector or host to produce a newly infected vector or host. Also, as noted by Wonham et al. (2004), in vector-host models, R_0 is positively related to vector population size, T, but negatively related to host population size, N.

References

Diekmann, O. et al. 1990. On the definition and the computation of the basic reproduction ratio R_0 in models for infectious diseases in heterogeneous populations. – J. Math. Biol. 28: 365–

382.

Van den Driessche, P. and Watmough, J. 2002. Reproduction numbers and sub-threshold endemic equilibria for compartmental models of disease transmission. – Math. Biosci. 180: 29–48.

Wonham, M. J. et al. 2004. An epidemiological model for West Nile Virus: invasion analysis and control applications. – Proc. R. Soc. B 271: 501–507.

Appendix 2

R programming script to calculate and display R_0 values and equilibrial host and vector densities

```
#### Preliminaries
library(lattice); library(deSolve); library(akima); library(reshape)

###################################################################
#### 1. Host resistance and vector preference: SEI model
###################################################################
## Derivation of R0 for SEI model
## Model from Matt's powerpoint without Carrier state
## S=susceptible hosts, E=exposed hosts, I=infected hosts,
## U=non-infectious hosts, V=infectious hosts
## N=total # hosts (S+E+I), T=total # of vectors (U+V), gamma = recovery
## rate of infected and exposed hosts, alpha = acquisition efficiency,
## beta = inoculation efficiency, delta = incubation rate, mu = vector turnover rate
## p = vector preference for disease status

library(Ryacas)
EigenValues <- function(x) Sym("EigenValues(", x, ")")

# Define variables in yacas
S <- Sym("S"); E <- Sym("E"); In <- Sym("In"); N <- Sym("N")
U <- Sym("U"); Vec <- Sym("Vec"); T <- Sym("T")
gamma <- Sym("gamma"); alpha <- Sym("alpha"); beta <- Sym("beta")
delta <- Sym("delta"); mu <- Sym("mu"); p <- Sym("p")

# Define the model equations
dS <- gamma*(E + In) - (beta*S*Vec)/(p*In + S + E)
dE <- (beta*S*Vec)/(p*In + S + E) - (delta + gamma)*E
dIn <- delta*E - gamma*In
dU <- mu*(U + Vec) - (alpha*p*In*U)/(p*In + S + E) - mu*U
dVec <- (alpha*p*In*U)/(p*In + S + E) - mu*Vec

# Accumulation functions
FE <- (beta*S*Vec)/(p*In + S + E)
FI <- 0
FV <- (alpha*p*In*U)/(p*In + S + E)
# Loss functions
VE <- (delta + gamma)*E
VI <- gamma*In - delta*E
VV <- mu*Vec

# Partial Derivatives
# F matrix elements
f11 <- deriv(FE, E)
f12 <- deriv(FE, In)
f13 <- deriv(FE, Vec)
f21 <- 0
f22 <- 0
f23 <- 0
f31 <- deriv(FV, E)
f32 <- deriv(FV, In)
f33 <- deriv(FV, Vec)
# V matrix elements
v11 <- deriv(VE, E)
v12 <- deriv(VE, In)
v13 <- deriv(VE, Vec)
v21 <- deriv(VI, E)
v22 <- deriv(VI, In)
v23 <- deriv(VI, Vec)
v31 <- deriv(VV, E)
v32 <- deriv(VV, In)
```

```
v33 <- deriv(VV, Vec)

# Construct F and V matrices
F <- List(List(f11, f12, f13), List(f21, f22, f23), List(f31, f32, f33))
V <- List(List(v11, v12, v13), List(v21, v22, v23), List(v31, v32, v33))
# Evaluate F and V at disease-free equilibrium, where S = N, E = In = Vec = 0, and U = T
F0 <- Simplify(Subst(Subst(Subst(Subst(Subst(F, E, 0), In, 0), Vec, 0), S, N), U, T))
V0 <- Simplify(Subst(Subst(Subst(Subst(Subst(V, E, 0), In, 0), Vec, 0), S, N), U, T))
# Eigenvalues
K <- EigenValues(F0 * Inverse(V0))

#### Plotting R0 vs. delta
# Constants
alpha = 0.6; beta = 0.6; gamma = 0.3; mu = 0.5
N = 100; T = 200
S = 100; E = 0; In = 0; U = 199; Vec = 1
# Sequences of delta and p values
dv <- seq(0, 2, by = 0.1) # Values for delta
pv <- c(0.1, 1, 10) # Values for p

# R0 equation as evaluatable expression
R0.sei <- expression(sqrt((alpha*beta*delta*p*T)/(N*(delta + gamma)*gamma*mu)))

## Calculate R0 using 'outer' function for a range of delta and p values
r0.sim <- function(x, y) {
            eval(R0.sei, list(alpha = alpha, beta = beta, delta = x, p = y, T = T, N =
N, gamma = gamma, mu = mu))
}
r0.dat <- expand.grid(x = dv, y = pv)
r0.dat$z <- as.vector(outer(dv, pv, FUN = r0.sim)) # calculate R0 for each combination
of delta and p values
names(r0.dat) <- c("delta", "p", "R0")

# Line plot of R0 and delta
xyplot(R0 ~ delta, groups = p,
       data = r0.dat,
       scales = list(alternating = 1, tck = c(1,0), cex = 1.3),
       layout = c(1,1), aspect = 1, type = "l", strip = FALSE,
       ylab = list(expression("R"[0]), cex = 1.5),
       xlab = list(expression("Incubation Rate (" * delta * ")"), cex = 1.4),
       cex = 1.2, lwd = 2, col = "black", lty = c(3, 1, 2),
       key = list(x = 0.025, y = 0.85, corner = c(0,0),
                  lines = list(lwd = 2, col = "black", lty = c(3, 1, 2)),
                  text = list(lab = c("avoid I", "no preference", "prefer I"), cex =
1.1),
                  fontfamily = "serif")
       )
       trellis.focus("panel", 1, 1); panel.abline(h = 1, lty = 2, col = "grey50");
trellis.unfocus()


################################################################################
#### SINGLE PATCH MODEL WITH CARRIER STATE
################################################################################
## Deriving R0
# Define variables in yacas
S <- Sym("S"); E <- Sym("E"); In <- Sym("In"); N <- Sym("N")
U <- Sym("U"); Vec <- Sym("Vec"); T <- Sym("T")
gamma <- Sym("gamma"); alpha <- Sym("alpha"); beta <- Sym("beta")
delta <- Sym("delta"); mu <- Sym("mu"); p <- Sym("p")
Ca <- Sym("Ca"); a <- Sym("a"); q <- Sym("q")  # Carrier State Model Variables

## Define model equations
dS <- gamma*(E + Ca + In) - (beta*S*Vec)/(p*In + S + E + Ca)
dE <- (beta*S*Vec)/(p*In + S + E + Ca) - (delta + gamma)*E
dCa <- q*delta*E - gamma*Ca
dIn <- (1 - q)*delta*E - gamma*In
dU <- mu*(U + Vec) - (alpha*p*In*U)/(p*In + S + E + Ca) - (a*alpha*Ca*U)/(p*In + S + E +
Ca) - mu*U
dVec <- (alpha*p*In*U)/(p*In + S + E + Ca) + (a*alpha*Ca*U)/(p*In + S + E + Ca) - mu*Vec
```

```
# Accumulation Functions
FE <- (beta*S*Vec)/(p*In + S + E + Ca)
FC <- 0
FI <- 0
FV <- (alpha*p*In*U)/(p*In + S + E + Ca) + (a*alpha*Ca*U)/(p*In + S + E + Ca)
# Loss Functions
VE <- (delta + gamma)*E
VC <- -q*delta*E + gamma*Ca
VI <- -(1-q)*delta*E + gamma*In
VV <- mu*Vec

# Derivatives
# F matrix elements
f11 <- deriv(FE, E)
f12 <- deriv(FE, Ca)
f13 <- deriv(FE, In)
f14 <- deriv(FE, Vec)
f21 <- 0
f22 <- 0
f23 <- 0
f24 <- 0
f31 <- 0
f32 <- 0
f33 <- 0
f34 <- 0
f41 <- deriv(FV, E)
f42 <- deriv(FV, Ca)
f43 <- deriv(FV, In)
f44 <- deriv(FV, Vec)
# V matrix elements
v11 <- deriv(VE, E)
v12 <- deriv(VE, In)
v13 <- deriv(VE, Ca)
v14 <- deriv(VE, Vec)
v21 <- deriv(VC, E)
v22 <- deriv(VC, Ca)
v23 <- deriv(VC, In)
v24 <- deriv(VC, Vec)
v31 <- deriv(VI, E)
v32 <- deriv(VI, Ca)
v33 <- deriv(VI, In)
v34 <- deriv(VI, Vec)
v41 <- deriv(VV, E)
v42 <- deriv(VV, Ca)
v43 <- deriv(VV, In)
v44 <- deriv(VV, Vec)

# Construct F and V matrices
F <- List(List(f11, f12, f13, f14), List(f21, f22, f23, f24), List(f31, f32, f33, f34),
List(f41, f42, f43, f44))
V <- List(List(v11, v12, v13, v14), List(v21, v22, v23, v24), List(v31, v32, v33, v34),
List(v41, v42, v43, v44))
# Evaluate F and V at disease-free equilibrium, where S = N, E = In = Ca = Vec = 0, and
U = T
F0 <- Simplify(Subst(Subst(Subst(Subst(Subst(Subst(F, E, 0), Ca, 0), In, 0), Vec, 0), S,
N), U, T))
V0 <- Simplify(Subst(Subst(Subst(Subst(Subst(Subst(V, E, 0), Ca, 0), In, 0), Vec, 0), S,
N), U, T))
# Eigenvalues
K <- EigenValues(F0 * Inverse(V0))
PrettyForm(K)

#### Numerical simulation
seic.mod <- function(Time, State, Pars){
  with(as.list(c(State, Pars)), {
    dS <- gamma*(E + Ca +In) - (beta*S*Vec)/(p*In + S + E + Ca)
    dE <- (beta*S*Vec)/(p*In + S + E + Ca) - (delta + gamma)*E
    dCa <- q*delta*E - gamma*Ca
    dIn <- (1-q)*delta*E - gamma*In
    dU <- mu*Vec - (alpha*p*In*U)/(p*In + S + E + Ca) - (a*alpha*Ca*U)/(p*In + S + E +
Ca)
    dVec <- (alpha*p*In*U)/(p*In + S + E + Ca) + (a*alpha*Ca*U)/(p*In + S + E + Ca) -
mu*Vec
    return(list(c(dS, dE, dCa, dIn, dU, dVec)))
  })
}

Pars <- c(alpha=0.6, beta=0.6, delta=2, gamma=0.2, mu=0.5, p=1, q=0.4, a=1)
State <- c(S=100, E=0, Ca=0, In=0, U=199, Vec=1)
Time <- seq(0, 100, by = 1)
mod.out <- as.data.frame(ode(func = seic.mod, y = State, parms = Pars, times = Time))

#### Time series figure
matplot(mod.out[,-1], type = "l",
        xlab = "Time step", ylab = "Population size",
        col = grey(seq(0,1,by=1/6)), lty = c(1:6), lwd = 2,
        cex.axis = 1.3, cex.lab = 1.6)
legend("topright", c("S", "E", "C", "I", "U", "V"), col = grey(seq(0,1,by=1/6)), lty =
c(1:6), lwd = 2)

#### Contour plots for single-patch SEIC model; combining R0, I*, V*
## Plot-wide constants
alpha = 0.6; beta = 0.6; gamma = 0.3; mu = 0.5; a = 1
N = 100; T = 200
S = 100; E = 0; Ca = 0; In = 0; U = 199; Vec = 1
# Sequences of delta, p, and q values
dv <- seq(0, 2, by = 0.1) # Values for delta
qv <- seq(0, 1, by = 0.05) # Values for q
pv <- c(0.1, 1, 10) # Values for p
dpq.grid <- expand.grid(dv, pv, qv)
names(dpq.grid) <- NULL

# R0.C calculation over ranges of delta, q, and p
R0.C <- expression(sqrt((alpha*beta*delta*(p + a*q - q*p)*T)/(gamma*(delta +
gamma)*mu*N)))
r0.dpq <- function(x) {
  R0.calc <- eval(R0.C, list(alpha = alpha, beta = beta, delta = x[1], p = x[2], T = T,
N = N, gamma = gamma, mu = mu, q = x[3], a = a))
  dat <- c(x, R0.calc)
  return(dat)
}
rdat <- apply(dpq.grid, 1, r0.dpq)
Rdat <- data.frame(t(rdat))
Rdat <- cbind(Rdat[,1:3], rep("R0", nrow(Rdat)), Rdat[,4])
names(Rdat) <- c("delta", "p", "q", "variable", "response")

# Equilibrium Simulation
seic.sim <- function(x){
  Pars <- c(alpha = alpha, beta = beta, delta = x[1], gamma = gamma, mu = mu, p = x[2],
q = x[3], a = a)
  State <- c(S = S, E = E, Ca = Ca, In = In, U = U, Vec = Vec)
  Time <- seq(0, 1000, by = 1)
  model.out <- as.data.frame(ode(func = seic.mod,
                               y = State,
                               parms = Pars,
                               times = Time))
  dat <- c(x, model.out[nrow(model.out),]$In, model.out[nrow(model.out),]$Vec)
  return(dat)
}
seicdat <- apply(dpq.grid, 1, seic.sim)
seicdat2 <- data.frame(t(seicdat))
names(seicdat2) <- c("delta", "p", "q", "In*", "Vec*")
SEICdat <- reshape(seicdat2, direction = "long", timevar = "variable",
          times = names(seicdat2[,4:5]), varying = 4:5, v.names = "response")

# Combining and restructuring datasets for plotting
plotdat <- rbind(Rdat, SEICdat[,-6])
plotdat$pf <- factor(ifelse(plotdat$p == 0.1, "p = 0.1",
                    ifelse(plotdat$p == 1, "p = 1", "p = 10")))
```

```
plotdat$variable <- factor(plotdat$variable)

## Contourplots
# Iterative function for interpolation
lplotfun <- function(x){
  subdat <- subset(plotdat, variable == x[1] & pf == x[2])
  ca.interp <- interp(x = subdat$q, y = subdat$delta, z = subdat$response,
                      xo = seq(min(subdat$q), max(subdat$q), length = 200),
                      yo = seq(min(subdat$delta), max(subdat$delta), length = 200))
  ca.eg <- expand.grid(ca.interp$y, ca.interp$x)
  zzrc <- recast(as.data.frame(ca.interp), formula = ... ~ ., id.var = c("x", "y"))
  as.data.frame(cbind(ca.eg, zzrc[,4]))
}
eg <- expand.grid(levels(plotdat$variable), levels(plotdat$pf))
lplotdat <- apply(eg, 1, lplotfun)
lpd.casim <- do.call("rbind", lplotdat)
names(lpd.casim) <- c("delta", "q", "response")
nr <- nrow(as.data.frame(lplotdat))
lpd.casim$variable <- factor(rep(c(rep("R0", nr),
                                   rep("In*", nr),
                                   rep("Vec*", nr)), 3))
lpd.casim$pf <- factor(c(rep("p = 0.1", nr*3), rep("p = 1", nr*3), rep("p = 10", nr*3)))
lpd.casim$response.perc <- ifelse(lpd.casim$variable == "Vec*",
                            (lpd.casim$response/T)*100, lpd.casim$response) # Population
equilibria as percent
summary(lpd.casim)

# levelplot functions
# In* and Vec* contour plots
levelplot(response.perc ~ delta * q|variable*pf, data = lpd.casim[!(lpd.casim$variable
== "R0"),],
          col.regions = grey(seq(0,1,by=0.01)),
          colorkey = list(labels = list(cex = 1.3)),
          layout = c(2,3), aspect = 1, as.table = TRUE, strip = TRUE,
                    scales = list(alternating = 1, tck = c(1,0), relation = "free", axs
= "i", cex = 1.3,
                        x = list(limits = c(0,2), at = c(0, 1, 2),
                                 labels = list(rep("",3), rep("",3),
                                               rep("",3), rep("",3),
                                               c(0, 1, 2), c(0, 1, 2))),
                        y = list(limits = c(0,1), at = c(0, 0.5, 1),
                                 labels = list(c(0, 0.5, 1), rep("",3),
                                               c(0, 0.5, 1), rep("",3),
                                               c(0, 0.5, 1), rep("",3)))),
          ylab = list("Proportion Carriers (q)", cex = 1.6),
          xlab = list(expression("Incubation rate" (delta)), cex = 1.6))

# Just R0 contour plots
levelplot(response ~ delta * q|variable*pf, data = lpd.casim[lpd.casim$variable ==
"R0",],
          col.regions = grey(seq(0,1,by=0.01)),
          colorkey = list(labels = list(cex = 1.3)),
          layout = c(1,3), aspect = 1, as.table = TRUE, strip = TRUE,
          scales = list(alternating = 1, tck = c(1,0), relation = "free", axs = "i", cex
= 1.3,
                        x = list(limits = c(0,2), at = c(0, 1, 2),
                                 labels = list(rep("",3),
                                               rep("",3),
                                               c(0, 1, 2))),
                        y = list(limits = c(0,1), at = c(0, 0.5, 1),
                                 labels = list(c(0, 0.5, 1),
                                               c(0, 0.5, 1),
                                               c(0, 0.5, 1)))),
          ylab = list("Proportion Carriers (q)", cex = 1.6),
          xlab = list(expression("Incubation rate" (delta)), cex = 1.6))


###############################################################################
#### Varying acquisition, a, and q for SEIC model
## Constants
```

```
alpha = 0.3; beta = 0.6; gamma = 0.3; mu = 0.8; delta = 1; p = 1
N = 100; T = 200
S = 100; E = 0; Ca = 0; In = 0; U = 199; Vec = 1
# Sequences of a, p, and q values
av <- c(0.5, 1, 2) # Values for a
qv <- seq(0, 1, by = 0.05) # Values for q
aq.grid <- expand.grid(av, qv)
names(aq.grid) <- NULL

# R0 Simulation
R0.aq <- function(x) {
  R0.res <- eval(R0.C, list(alpha = alpha, beta = beta, delta = delta, p = p, T = T, N =
N, gamma = gamma, mu = mu, q = x[2], a = x[1]))
  dat <- c(x, R0.res)
  return(dat)
}
radat <- apply(aq.grid, 1, R0.aq)
Radat <- data.frame(t(radat))
names(Radat) <- c("a", "q", "R0")
Radat$anice <- factor(paste("a", Radat$a, sep = " = "))

# xyplot function
xyplot(R0 ~ q, data = Radat, groups = anice,
       type = "l", pch = 16, col = "black", lty = c(2,1,3), lwd = 2,
       as.table = TRUE, strip = FALSE,
       scales = list(alternating = 1, tck = c(1,0), cex = 1.3),
       xlab = list("Host tolerance (q)", cex = 1.6),
       ylab = list(expression("R"[0]), cex = 1.6),
       key = list(x = 0.05, y = 0.95,
                  text = list(labels = levels(Radat$anice), cex = 1.2),
                  lines = list(col = "black", lty = c(2,1,3), lwd = 2)))


################################################################################
### 2 patch SEIC model with constant movement
################################################################################
## Constants
alpha = 0.6; beta = 0.6; gamma = 0.3; mu = 0.5; a = 1
qs = 0; deltas = 2
md = 0.5; ms = 0.5
Sd = 100; Ss = 100; Ed = 0; Es = 0
Cd = 0; Cs = 0; Id = 0; Is = 0
Ud = 200; Us = 199; Vd = 0; Vs = 1
Nd = Sd + Ed + Cd + Id; Ns = Ss + Es + Cs + Is

# Sequences of delta, p, and q values
ddv <- seq(0, 2, by = 0.1) # Values for deltad
qdv <- seq(0, 1, by = 0.05) # Values for qd
pv <- c(0.1, 1, 10) # Values for p
patch.grid <- expand.grid(ddv, pv, qdv)
names(patch.grid) <- NULL

#### Numerical simulation
# Equations for 2 patch SEIC model
ca.patch <- function(Time, State, Pars){
  with(as.list(c(State, Pars)), {
    # Defended patch
    dSd <- gamma*(Ed + Cd + Id) - (beta*Sd*Vd)/(p*Id + Sd + Ed + Cd)
    dEd <- (beta*Sd*Vd)/(p*Id + Sd + Ed + Cd) - (deltad + gamma)*Ed
    dCd <- qd*deltad*Ed - gamma*Cd
    dId <- (1-qd)*deltad*Ed - gamma*Id
    dUd <- mu*Vd - (alpha*p*Id*Ud)/(p*Id + Sd + Ed + Cd) - (a*alpha*Cd*Ud)/(p*Id + Sd +
Ed + Cd) - md*Ud + ms*Us
    dVd <- (alpha*p*Id*Ud)/(p*Id + Sd + Ed + Cd) + (a*alpha*Cd*Ud)/(p*Id + Sd + Ed + Cd)
- mu*Vd - md*Vd + ms*Vs
    # Susceptible patch
    dSs <- gamma*(Es + Cs + Is) - (beta*Ss*Vs)/(p*Is + Ss + Es + Cs)
    dEs <- (beta*Ss*Vs)/(p*Is + Ss + Es + Cs) - (deltas + gamma)*Es
    dCs <- qs*deltas*Es - gamma*Cs
    dIs <- (1-qs)*deltas*Es - gamma*Is
    dUs <- mu*Vs - (alpha*p*Is*Us)/(p*Is + Ss + Es + Cs) - (a*alpha*Cs*Us)/(p*Is + Ss +
Es + Cs) - ms*Us + md*Ud
    dVs <- (alpha*p*Is*Us)/(p*Is + Ss + Es + Cs) + (a*alpha*Cs*Us)/(p*Is + Ss + Es + Cs)
- mu*Vs - ms*Vs + md*Vd
    return(list(c(dSd, dEd, dCd, dId, dUd, dVd, dSs, dEs, dCs, dIs, dUs, dVs)))
  })
}

# Simulation over range of deltad, qd, p
patch.sim <- function(x){
  Pars <- c(alpha = alpha, beta = beta, deltad = x[1], deltas = deltas, gamma = gamma,
mu = mu, p = x[2], qd = x[3], qs = qs, a = a, md = md, ms = ms)
  State <- c(Sd = Sd, Ed = Ed, Cd = Cd, Id = Id, Ud = Ud, Vd = Vd, Ss = Ss, Es = Es, Cs
= Cs, Is = Is, Us = Us, Vs = Vs)
  Time <- seq(0, 1000, by = 1)
  model.out <- as.data.frame(ode(func = ca.patch,
                                y = State,
                                parms = Pars,
                                times = Time))
  model.dat <- c(model.out[nrow(model.out),]$Id,
                 model.out[nrow(model.out),]$Vd,
                 model.out[nrow(model.out),]$Is,
                 model.out[nrow(model.out),]$Vs)
  return(c(x, model.dat))
}
patch.dat <- apply(patch.grid, 1, patch.sim)
Patch.dat <- data.frame(t(patch.dat))
names(Patch.dat) <- c("deltad", "p", "qd", "Id*", "Vd*", "Is*", "Vs*")

## Restructure dataset for plotting
plotdat <- reshape(Patch.dat, direction = "long", timevar = "variable",
                   times = names(Patch.dat[,4:ncol(Patch.dat)]),
                   varying = 4:ncol(Patch.dat), v.names = "response")
plotdat$pf <- factor(ifelse(plotdat$p == 0.1, "p = 0.1",
                    ifelse(plotdat$p == 1, "p = 1", "p = 10")))
plotdat$variable <- factor(plotdat$variable)


## Contourplot
# Iterative function for interpolation
lplotfun <- function(x){
  subdat <- subset(plotdat, variable == x[1] & pf == x[2])
  patch.interp <- interp(x = subdat$qd, y = subdat$deltad, z = subdat$response,
                    xo = seq(min(subdat$qd), max(subdat$qd), length = 100),
                    yo = seq(min(subdat$deltad), max(subdat$deltad), length = 100))
  patch.eg <- expand.grid(patch.interp$y, patch.interp$x)
  zzrc <- recast(as.data.frame(patch.interp), formula = ... ~ ., id.var = c("x", "y"))
  as.data.frame(cbind(patch.eg, zzrc[,4]))
}
eg <- expand.grid(levels(plotdat$variable), levels(plotdat$pf))
lplotdat <- apply(eg, 1, lplotfun)
lpd.patch <- do.call("rbind", lplotdat)
names(lpd.patch) <- c("deltad", "qd", "response")
nr <- nrow(as.data.frame(lplotdat))
lpd.patch$variable <- factor(rep(c(rep("Id*", nr), rep("Is*", nr), rep("Vd*", nr),
rep("Vs*", nr)), 3))
lpd.patch$pf <- factor(c(rep("p = 0.1", nr*4), rep("p = 1", nr*4), rep("p = 10", nr*4)))
lpd.patch$response.perc <- ifelse(lpd.patch$variable == "Vs*" | lpd.patch$variable ==
"Vd*",
                            (lpd.patch$response/T)*100, lpd.patch$response) # Population
equilibria as percent
summary(lpd.patch)
lpd.patch.sub <- subset(lpd.patch, !(pf == "p = 1") & (variable == "Is*" | variable ==
"Vs*"))
summary(lpd.patch.sub)

# levelplot function
levelplot(response.perc ~ deltad * qd|variable*pf, data = lpd.patch.sub,
          # col.regions = topo.colors(24),
          col.regions = grey(seq(0,1,by=0.01)),
          colorkey = list(labels = list(cex = 1.3)),
          scales = list(alternating = 1, tck = c(1,0), relation = "free", axs = "i", cex
= 1.3,
                        x = list(limits = c(0,2), at = c(0, 1, 2),
                                 labels = list(rep("",3),
                                               rep("",3),
                                               c(0, 1, 2),
                                               c(0, 1, 2))),
                        y = list(limits = c(0,1), at = c(0, 0.5, 1),
                                 labels = list(c(0, 0.5, 1),
                                               rep("",3),
                                               c(0, 0.5, 1),
                                               rep("",3)))),
          layout = c(2,2), aspect = 1, as.table = TRUE, strip = FALSE,
          ylab = list(expression("Proportion Carriers in defended patch (q  "[d]*")"),
cex = 1.3),
          xlab = list(as.expression(bquote("Incubation rate in defended
patch"~(delta[d]))), cex = 1.3))

################################################################################
#### 2 patch SEIC model with preference-based patch dispersal
################################################################################
md = 0.5; ms = 0.5
Sd = 100; Ss = 100; Ed = 0; Es = 0
Cd = 0; Cs = 0; Id = 0; Is = 0
Ud = 200; Us = 199; Vd = 0; Vs = 1
Nd = Sd + Ed + Cd + Id; Ns = Ss + Es + Cs + Is
b = 0; c = 0.5; d = 0.1

# Sequences of delta, p, and q values
ddv <- seq(0, 2, by = 0.1) # Values for deltad
qdv <- seq(0, 1, by = 0.05) # Values for qd
pv <- c(0.1, 1, 10) # Values for p
mip.grid <- expand.grid(ddv, qdv, pv)
# Adding values of b and d to mip.grid
mip.grid$b <- ifelse(mip.grid$Var3 == 0.1, 0.1, 0)
mip.grid$d <- ifelse(mip.grid$Var3 == 1, 0, 0.1)
names(mip.grid) <- NULL # Removing column names from expand.grid() object is critical to
use apply() with ode() functions

# Numerical Simulation, over ranges of delta, q, and p
# Model equations
mip <- function(Time, State, Pars){
  with(as.list(c(State, Pars)), {
    # Defended patch
    dSd <- gamma*(Ed + Cd + Id) - (beta*Sd*Vd)/(p*Id + Sd + Ed + Cd)
    dEd <- (beta*Sd*Vd)/(p*Id + Sd + Ed + Cd) - (deltad + gamma)*Ed
    dCd <- qd*deltad*Ed - gamma*Cd
    dId <- (1-qd)*deltad*Ed - gamma*Id
    dUd <- mu*Vd - (alpha*p*Id*Ud)/(p*Id + Sd + Ed + Cd) - (a*alpha*Cd*Ud)/(p*Id + Sd +
Ed + Cd) - ((b*Id + c)/(d*Id + 1))*Ud + ((b*Is + c)/(d*Is + 1))*Us
    dVd <- (alpha*p*Id*Ud)/(p*Id + Sd + Ed + Cd) + (a*alpha*Cd*Ud)/(p*Id + Sd + Ed + Cd)
- mu*Vd - ((b*Id + c)/(d*Id + 1))*Vd + ((b*Is + c)/(d*Is + 1))*Vs
    # Susceptible patch
    dSs <- gamma*(Es + Cs + Is) - (beta*Ss*Vs)/(p*Is + Ss + Es + Cs)
    dEs <- (beta*Ss*Vs)/(p*Is + Ss + Es + Cs) - (deltas + gamma)*Es
    dCs <- qs*deltas*Es - gamma*Cs
    dIs <- (1-qs)*deltas*Es - gamma*Is
    dUs <- mu*Vs - (alpha*p*Is*Us)/(p*Is + Ss + Es + Cs) - (a*alpha*Cs*Us)/(p*Is + Ss +
Es + Cs) - ((b*Is + c)/(d*Is + 1))*Us + ((b*Id + c)/(d*Id + 1))*Ud
    dVs <- (alpha*p*Is*Us)/(p*Is + Ss + Es + Cs) + (a*alpha*Cs*Us)/(p*Is + Ss + Es + Cs)
- mu*Vs - ((b*Is + c)/(d*Is + 1))*Vs + ((b*Id + c)/(d*Id + 1))*Vd
    return(list(c(dSd, dEd, dCd, dId, dUd, dVd, dSs, dEs, dCs, dIs, dUs, dVs)))
  })
}

mip.sim <- function(x){
  Pars <- c(alpha = alpha, beta = beta, deltad = x[1], deltas = 2, gamma = gamma, mu =
mu, qd = x[2], qs = 0, a = a, p = x[3], b = x[4], c = c, d = x[5])
```

```
  State <- c(Sd = Sd, Ed = Ed, Cd = Cd, Id = Id, Ud = Ud, Vd = Vd, Ss = Ss, Es = Es, Cs
= Cs, Is = Is, Us = Us, Vs = Vs)
  Time <- seq(0, 1000, by = 1)
  model.out <- as.data.frame(ode(func = mip,
                               y = State,
                               parms = Pars,
                               times = Time))
  model.dat <- c(model.out[nrow(model.out),]$Id,
               model.out[nrow(model.out),]$Vd,
               model.out[nrow(model.out),]$Is,
               model.out[nrow(model.out),]$Vs)
  return(c(x, model.dat))
}
mip.dat <- apply(mip.grid, 1, mip.sim)
MIP.dat <- data.frame(t(mip.dat))
names(MIP.dat) <- c("deltad", "qd", "p", "b", "d", "Id*", "Vd*", "Is*", "Vs*")

# Restructuring dataset for plotting
plotdat <- reshape(MIP.dat, direction = "long", timevar = "variable",
                   times = names(MIP.dat[,6:9]), varying = 6:9, v.names = "response")
plotdat$pf <- factor(ifelse(plotdat$p == 0.1, "p = 0.1",
                            ifelse(plotdat$p == 1, "p = 1", "p = 10")))
plotdat$variable <- factor(plotdat$variable)

## Contourplots
# Iterative function for interpolation
lplotfun <- function(x){
  subdat <- subset(plotdat, variable == x[1] & pf == x[2])
  patch.interp <- interp(x = subdat$qd, y = subdat$deltad, z = subdat$response,
                         xo = seq(min(subdat$qd), max(subdat$qd), length = 100),
                         yo = seq(min(subdat$deltad), max(subdat$deltad), length = 100))
  patch.eg <- expand.grid(patch.interp$y, patch.interp$x)
  zzrc <- recast(as.data.frame(patch.interp), formula = ... ~ ., id.var = c("x", "y"))
  as.data.frame(cbind(patch.eg, zzrc[,4]))
}
eg <- expand.grid(levels(plotdat$variable), levels(plotdat$pf))
lplotdat <- apply(eg, 1, lplotfun)
lpd.mip <- do.call("rbind", lplotdat)
names(lpd.mip) <- c("deltad", "qd", "response")
nr <- nrow(as.data.frame(lplotdat))
lpd.mip$variable <- factor(rep(c(rep("Id*", nr), rep("Is*", nr), rep("Vd*", nr),
rep("Vs*", nr)), 3))
lpd.mip$pf <- factor(c(rep("p = 0.1", nr*4), rep("p = 1", nr*4), rep("p = 10", nr*4)))
lpd.mip$response.perc <- ifelse(lpd.mip$variable == "Vd*" | lpd.mip$variable == "Vs*",
                            (lpd.mip$response/T)*100, lpd.mip$response)
summary(lpd.mip)

# levelplot function
levelplot(response.perc ~ deltad * qd|pf*variable,
          data = lpd.mip[!(lpd.mip$pf == "p = 1"),],
          col.regions = grey(seq(0,1,by=0.01)),
          colorkey = list(labels = list(cex = 0.9)),
          layout = c(2,4), aspect = 1, as.table = TRUE, strip = TRUE,
          scales = list(alternating = 1, tck = c(1,0), relation = "free",
                        axs = "i", cex = 0.9,
                        x = list(limits = c(0,2), at = c(0, 1, 2),
                                 labels = list(rep("",3), rep("",3), rep("",3),
rep("",3),
                                               rep("",3), rep("",3), c(0, 1, 2), c(0, 1,
2))),
                        y = list(limits = c(0,1), at = c(0, 0.5, 1),
                                 labels = list(c(0, 0.5, 1), rep("",3), c(0, 0.5, 1),
rep("",3),
                                               c(0, 0.5, 1), rep("",3), c(0, 0.5, 1),
rep("",3))
                                 )),
          ylab = as.expression(bquote("Proportion Carriers in defended patch
(q"[d]*")")),
          xlab = as.expression(bquote("Incubation rate in defended patch"~(delta[d]))))
```

Appendix 3

Figure A1. Transient dynamics of single-patch model with carrier state.

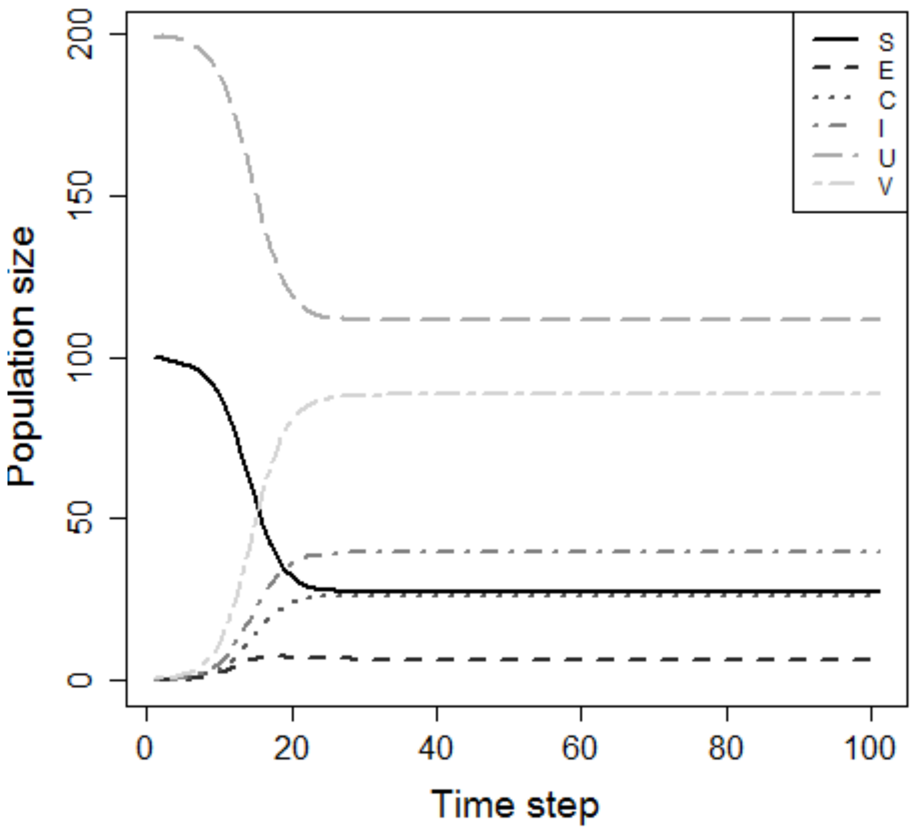

Figure A1. Transient dynamics of models generally reached equilibria quickly and monotonically, as shown in this example for the single-patch SEIC model (system 3 in main text). Parameter values: $\alpha = \beta = 0.6$, $\gamma = 0.2$, $\mu = 0.5$, $\delta = 2$, $p = 1$, $q = 0.4$, $a = 1$, N = 100, T = 200.