# Evolution of constitutive bacteriocin production and release

**Amrita Bhattacharya* and Curtis M Lively**

Department of Biology, Indiana University, 1001 E 3rd Street, Bloomington IN 47405

*Corresponding author

Email:

Amrita Bhattacharya – amritab.phd@gmail.com

Curtis M Lively – clively@indiana.edu

**Keywords:** Spite, Bacteriocins, Altruistic Suicide

**Abstract**

The production of the bacterial anti-competitor toxins, called bacteriocins, is widely described as spite, because the production of the toxins is costly and often requires cell death for release. Given these costs, it reasonable to think that bacteriocin production should be induced by the presence of unrelated competitor strains. Nonetheless, recent evidence in the insect-pathogenic bacterium *Xenorhabdus koppenhoeffri* showed that bacteriocin production occurs regardless of the presence of unrelated competitors. Could the constitutive production of bacteriocins be favored by natural selection? Here we use a mathematical model to examine this question, both within and between subpopulations. The model assumes that bacteriocin production is constitutive and costly, and that toxin release only occurs during natural cell death. We found that constitutive bacteriocin producers can outcompete non-producer sensitive strains within populations, and that it can spread in a metapopulation provided there is local competition for patches. Hence, the evolution of bacteriocin production does not require the detection of competitors or "suicidal" release of toxins.

## Introduction

The evolution and maintenance of spiteful behaviors that reduce the fitness of both the actor and the recipient pose a long-standing evolutionary puzzle (Hamilton, 1964, 1970). How can a trait that reduces the fitness of the actor be favored by natural selection? Nonetheless, bacteriocins are produced by almost all known lineages of bacteria (Klaenhammer, 1988; Riley & Chavan, 2006). Bacteriocins are proteinaceous toxins with diverse mechanisms of action, that can kill closely related local competitors without killing clone-mates (Reeves, 1965; Riley & Chavan, 2006). Bacteriocin production is considered spiteful because these toxins kill their targets and impose fitness costs of production, thus reducing the fitness of both recipients and actors (Gardner et al., 2004; West et al., 2007). In addition to the costs of DNA replication, maintenance, and protein synthesis, the release of these large, proteinaceous toxins often requires cell lysis, particularly in gram-negative cells that lack a cell wall (Riley & Wertz, 2002; Riley & Chavan, 2006; Wloch-Salamon et al., 2008).

Theory predicts that the competitive environment is key for the maintenance of costly bacteriocin production (Chao & Levin, 1981; Gardner et al., 2004). Bacteriocin-production is predicted to be most favored when competitors occur at comparable frequencies to the producing lineage (Gardner et al., 2004; Inglis et al., 2009). Further, a recent meta-analysis revealed that bacteriocin production is more likely to be associated with competition-related stressors such as starvation or cellular damage, than abiotic stressors such as heat or osmotic stress (Cornforth & Foster, 2013). This led to the proposition of the ‘competition sensing hypothesis,’ which posits that bacterial populations have evolved the ability to sense and respond to competition in their environment by upregulating bacteriocin production (Cornforth & Foster, 2013). While this may be generally true, the gram negative bacteria, *Xenorhabdus koppenhoeffri*, was found in vitro to

produce and release the same levels of bacteriocins in both the presence and absence of non-self competitors (Bhattacharya et al., 2018). Why do *X koppenhoeffri* produce and release bacteriocins in the absence of competitors?

Here we address this question by building a mathematical model to ask whether the constitutive production of bacteriocins can be selected in natural bacterial populations. The model examines competition between bacteriocin-producer and bacteriocin-sensitive cells when bacteriocin production is constitutive and costly. We assume that toxins are only released upon natural cell death. Our results demonstrate that constitutive bacteriocin-producers can outcompete faster growing sensitive cells within patches, and that they increase in frequency in structured populations, if there is some degree of local competition.

## Model

We develop a model that tracks competition between two pathogenic bacterial strains: a bacteriocin producing strain and a sensitive strain that is killed by the producer's bacteriocins. The strains compete for a finite amount of a non-renewable resource within a shared host. We assume that the production of bacteriocins is constitutive and costly. In other words, we assume that bacteriocin production is not a plastic response to the presence of competitors and that it imposes a fitness cost by reducing the birth rate of producers. Finally, we assumed that the release of bacteriocins only occurs during natural cell death, independent of whether competing cells are present. Importantly, the producer death rate is fixed and independent of competitor densities enabling the test of inducibility being necessary for the maintenance of costly bacteriocin production. First, we ask whether constitutive bacteriocin production could be

favored by selection within hosts. We then examine whether constitutive bacteriocin production could increase when rare in a metapopulation in which there is between-host competition.

## Within-Host Competition

Let $B_t$ be the "birth" number per sensitive cell at time $t$. We assumed $B_t$ was both density dependent and resource dependent, as follows:

$$B_t = b\left[1 - \frac{N_t}{N_t + R_t}\right] \quad (1)$$

where $b$ is the intrinsic number of new cells produced per bacterial cell per time step, $N_t$ is the total number cells (producer + sensitive) in the population at time $t$, and $R_t$ is the number of resource units remaining in the population at time $t$. The variable $b$ was set to 1 in our simulations. We also assumed that each cell division requires one unit of the resources available at time $t$. Similarly, the death rate, $D_t$, was also density-dependent and calculated as:

$$D_t = d\left[\frac{\mathrm{N_t}}{N_t + R_t}\right] \quad (2)$$

where $d$ is the intrinsic death rate. The number of producer cells at time step, $t$+1, is then given by

$$P_{t+1} = P_t[1 + B_t(1 - c) - D_t] \quad (3)$$

where $P_t$ is the number of producer cells at time step $t$, and $c$ is the birth-rate cost of bacteriocin production.

Toxin molecules were released only upon the natural death of producer cells. The total number of toxin molecules in the system at time $t$+1 was

$$X_{t+1} = (1 - z)X_t + xD_tP_t + S_t[1 - \exp\left(-\frac{\beta X_t}{S_t}\right)] \quad (4)$$

Where, $X_t$ is the number of toxin molecules in the system at time *t*; *z* is the loss probability of toxin molecules; *x* is the number of toxin molecules released by each dying producer cell; and β is the contact probability between toxin molecules and sensitive cells, where $S_t$ is the number of sensitive cells at time *t*. The first term on the right-hand side (RHS) of eq. (4) thus gives number of toxin molecules carried over from the previous time step. In the present study we set *z* equal to 0.05 to give a 5% decay rate. The second term on the RHS gives the number of new toxin molecules released into the system resulting from producer deaths. The third term on the RHS of (4) gives the number of sensitive cells that were killed by contact with toxin, where $X_t/S_t$ gives the mean number of toxin molecules per sensitive cell. Note that the exponential term $[\exp(-\frac{\beta X_t}{S_t})]$ gives the probability of the zero class in a Poisson distribution; hence [1 - $\exp\left(-\frac{\beta X_t}{S_t}\right)$] gives the probability that a sensitive cell would contact at least one toxin molecule, thus killing the sensitive cell. Thus, the third term on the RHS gives the number of toxin molecules removed from the system by killing sensitive cells, assuming each toxin-induced death removes a single toxin molecule from the system.

The number of sensitive cells at time step *t*+1 was calculated as:

$$S_{t+1} = S_t[1 + B_t - D_t - (1 - \exp(-\beta X_t/S_t)] \quad (5)$$

where $S_t$ is the number of sensitive cells at time *t*. After every time step, the number of resource units remaining in the system were calculated as:

$$R_{t+1} = R_t - P_t B_t(1 - c) - S_t B_t \quad (6)$$

where $R_t$ is the number of resource units at time t. The model thus assumes that the birth of every cell requires one resource unit.

The recursion equations above were iterated using R (R Core Team. 2019). The code is given in Supplementary Files 1 and 2. The simulation was initiated with 10 producer cells and

90 sensitive cells in a single host containing $10^8$ resource units. We ran the simulation for 100 time steps, which was more than enough for the resources to be depleted to zero. The intrinsic birth rate was set to 1 ($b$ = 1), and the intrinsic death rate was set to 0.01 ($d$ = 0.01). The cost of toxin production was set to 0.05 ($c$ = 0.05). We ran the model for a range of parameters for β and $x$ to explore the effects of the contact probability between toxin and sensitive cells, and the number of bacteriocin particles released following the death of producer cells.

## Between-Host Competition

For bacteriocin production to be favored by natural selection, producers not only have to "win" within hosts, but they also require an advantage under some level of between-host competition. Such an advantage likely depends on the degree of soft selection and migration. Here we used Frank's "scale-of-competition" method to examine bacteriocin evolution in a metapopulation (Frank, 1998). The method uses the variable '$a$' to represent the probability of competition with individuals derived from the same patch. The variable can also be seen as a combination of soft selection (s) and migration (m)(van Dyken, 2010), where

$$a = s + (1 - s)(1 - m)^2 \tag{7}$$

The model assumes a metapopulation comprising of a large number of hosts where each host is colonized by a fixed initial number of bacteria ($N$ = 100). One host in the metapopulation is colonized by a mixture of 10 producer cells and 90 sensitive cells. This is called the "local" host. All remaining hosts are colonized by 100 sensitive cells.

The relative fitness of bacteriocin producers ($W_{\text{prod}}$) was calculated as (Frank, 1998):

$$W_{\text{prod}} = \frac{W_{\text{prod}}}{a\bar{W}_{\text{local}} + (1-a)\bar{W}_{\text{global}}} \tag{7}$$

where, $W_{\text{prod}}$ is the number of producer cells at time $t$ ($P_t$) divided by the number of colonizing producer cells ($W_{\text{prod}} = P_t/P_1$) where $P_1$ =10; $\bar{W}_{\text{local}}$ is the number of cells per colonizing cell in the local host (i.e., ($P_t$ + $S_t$)/100) containing the producer cells; and $\bar{W}_{\text{global}}$ is the number of cells per colonizing cell in the non-focal hosts at dispersal, i.e., $S_{\text{tot}}$/100, where $S_{\text{tot}}$ is the total number of sensitive bacteria in non-local hosts at dispersal.

**Results**

<u>Within-Host Competition</u>

The model tracks the densities of bacteriocin-producer cells, bacteriocin-sensitive cells, and resource units within the host under varying contact probability between toxin molecules and sensitive cells, $\beta$, and varying levels of the number of bacteriocin particles released per dying producer cell, $x$ (Figure 1). Each simulation begins with 10 producer cells and 90 sensitive cells within a shared host. When $x$ is set to 0 (Fig 1 a-c) the densities of bacteriocin producers never exceed the densities of bacteriocin sensitive competitors, demonstrating that producers are unable to outcompete sensitive cells. However, bacteriocin-producing cells outcompete bacteriocin-sensitive competitors when $x$ is non-zero (Fig 1 e-i). Producers outcompete sensitive cells faster at higher levels of contact probability, β, and with higher numbers of bacteriocin particles released per dying producer cell, $x$ (Fig 1).

<u>Between-Host Competition</u>

Relative fitness of bacteriocin producers is examined across a metapopulation of hosts under initial conditions of $\beta$ = 0.1, varying levels of $x$, and three difference scales of competition (Fig 2). Relative fitness measures greater than 1 indicate invasion of the metapopulation by producers despite being rare initially. Our results demonstrate that the relative fitness of

bacteriocin producers can exceed 1 under many conditions (Fig 2 d-i) despite conservative assumptions about bacteriocin potency ($\beta = 0.1$; Fig 2). Invasion occurs faster and is favored more strongly at more local scales of competition i.e., higher values of '*a*', and with higher numbers of bacteriocin particles released per dying producer cell, *x*.

## Discussion

Bacteriocin production is an example of spiteful behavior, meaning that both the actor and the recipient pay fitness costs (Riley & Chavan, 2006). Bacteriocin production is frequently considered a 'suicide mission' where producer cells sacrifice themselves to deploy these anti-competitor toxins (West et al., 2007; Nedelcu et al., 2010). Inducible mechanisms that allow bacteriocin production to be increased by the competitive environment are predicted to be favored by selection (Gardner et al., 2004; Cornforth & Foster, 2013). Yet, contrary to expectations, bacteriocin production in *X koppenhoeffri* was found to occur even in the absence of non-self competitor cells (Bhattacharya et al., 2018). Our goal here was to understand why. We built a mathematical model to examine if and how, despite considerable costs of production and release, bacteriocin-producing lineages like *X. koppenhoeffri* persist.

Our results demonstrate that a bacteriocin-producing lineage can outcompete a faster growing sensitive lineage, if bacteriocins are released only upon natural cell death (Fig 1 and 2). The time taken for the producing lineage to outcompete the sensitive lineage decreases with increasing transmission efficiency, $\beta$, and higher numbers of bacteriocin particles released per dying producer cell, *x* (Fig 1). Further, the results show that bacteriocin producers can invade a metapopulation of sensitive cells, provided some degree of local competition (Figure 2). Consistent with previous work on the scale of competition in general (Durrett & Levin, 1997;

Gardner & West, 2004; Gardner et al., 2004), our results show that increasing local competition favors the invasion of bacteriocin production (Figure 2b). These results demonstrate that, despite cell deaths being necessary for bacteriocin release, inducible or higher-than-natural rates of cell death for 'suicidal' bacteriocin release are not required for bacteriocin-producing lineages to persist and increase in frequency when rare. Plasticity, however, could increase the advantage of bacteriocin production under some conditions (Bhattacharya, 2019; Granato & Foster, 2020; Niehus et al., 2021).

The assumptions about costs of bacteriocin production and bacteriocin potency (number of particles released per dying cell) in our models are probably conservative. The cost of bacteriocin production is set at 5% reduction in fitness ($c = 0.05$), which imposes a considerable growth disadvantage on the producer strain relative to the sensitive competitor. It has been previously argued that the fitness costs of bacteriocin production may be much less steep than commonly assumed (Dykes & Hastings, 1997). Empirical estimates in *E coli* that produce the most well-studied group of bacteriocins called colicins, suggest that every lysing producer cell is capable of releasing bacteriocin particles on the order of thousands of particles per cell (Gordon & Riley, 1999). Our model examines a wide range of this parameter ranging from 0-1000 particles of bacteriocin per dying producer cell, remaining highly conservative. Thus, our results demonstrate that the competitive benefits of bacteriocin production may be realized without invoking inducible mechanisms or 'suicide' despite high costs and low potencies of toxins produced.

It is important to note that these results do not preclude the possibility of plastic bacteriocin production or release. Indeed, recent work in *E coli* (Mavridou et al., 2018; Granato & Foster, 2020) provide evidence for increased toxin production in response to attack from a

neighboring competitor. However, bacteriocin production in *E coli* has also been reported to occur in the absence of competitors, typically at late growth stages (Eraso et al., 1996). Additionally, recent work using models to compare various mechanisms of bacteriocin production and release found that toxin sensitivity to be the preferred mode of bacteriocin production and release over unregulated mechanisms (Niehus et al., 2021). Our goal here is not to claim that bacteriocin production cannot be inducible, rather to demonstrate that non-inducible mechanisms of bacteriocin release can maintain bacteriocin production. We aim to caution against the generalization of bacteriocin production as a 'suicide' mission and to encourage additional direct investigation of the inducibility of this spiteful, yet ubiquitous bacterial trait.

To summarize, the model described in this study demonstrates, using conservative assumptions, that inducible mechanisms of bacteriocin expression aren't necessary to explain the maintenance of bacteriocin production in nature. Further, these results challenge the common assumption that bacteriocin production and release is a 'suicide mission.' Our models provide an alternative mechanism to explain the maintenance of spiteful bacteriocin production through canalized, constitutive expression instead of inducible production. We do not claim that canalized spite is the only, or even the most common, mechanism of bacteriocin production. However, we do offer a previously underexamined mechanism to explain the early evolution of bacteriocin production, and how spiteful bacteriocin production may be maintained in some species.

Acknowledgements

We would like to thank the Center for Integrative Studies in Animal Behavior at Indiana University for fellowship funding to AB when this work was conducted. We also thank Mike Wade and Farrah Bashey for helpful comments on a previous draft

Conflict of Interest: The authors declare no competing interests.

## References Cited


Bhattacharya, A. 2019. The Evolution of Antagonistic Social Interactions in Bacteria: An Investigation of Spite, Competition, and Virulence in *Xenorhabdus* spp. Dissertation, Indiana University

Bhattacharya, A., Pak, H.T. & Bashey, F. 2018. Plastic responses to competition: Does bacteriocin production increase in the presence of nonself competitors? Ecology and Evolution 8: 6880–6888.

Chao, L. & Levin, B.R. 1981. Structured habitats and the evolution of anticompetitor toxins in bacteria. Proceedings of the National Academy of Sciences 78: 6324–8.

Cornforth, D.M. & Foster, K.R. 2013. Competition sensing: the social side of bacterial stress responses. Nature Reviews Microbiology 11: 285–93.

Durrett, R. & Levin, S. 1997. Allelopathy in spatially distributed populations. Journal of Theoretical Biology 185: 165–171.

Dykes, G.A. & Hastings, J.W. 1997. Selection and fitness in bacteriocin-producing bacteria. Proceedings of the Royal Society B: Biological Sciences 264: 683–7.

Eraso, J.M., Chidambaram, M. & Weinstock, G.M. 1996. Increased production of colicin E1 in stationary phase. Journal of Bacteriology 178: 1928–35.

Frank, S.A. 1998. Foundations of Social Evolution. Princeton University Press. Princeton, NJ.

Gardner, A. & West, S.A. 2004. Spite and the scale of competition. Journal of Evolutionary Biology 17: 1195–203.

Gardner, A., West, S.A. & Buckling, A. 2004. Bacteriocins, spite and virulence. Proceedings of the Royal Society B: Biological Sciences 271: 1529–35.

Gordon, D.M. & Riley, M.A. 1999. A theoretical and empirical investigation of the invasion dynamics of colicinogeny. Microbiology (N Y) 145: 655–661.

Granato, E.T. & Foster, K.R. 2020. The evolution of mass cell suicide in bacterial warfare. Current Biology 30: 2836-2843.e3.

Hamilton, W.D. 1970. Selfish and spiteful behaviour in an evolutionary model. Nature 228: 1218–1220.

Hamilton, W.D. 1964. The genetical evolution of social behaviour. Journal of Theoretical Biology 7: 1–16.

Inglis, R.F., Gardner, A., Cornelis, P. & Buckling, A. 2009. Spite and virulence in the bacterium *Pseudomonas aeruginosa*. Proceedings of the National Academy of Sciences 106: 5703–5707.

Klaenhammer, T.R. 1988. Bacteriocins of lactic acid bacteria. Biochimie 70: 337–49.

Mavridou, D.A.I., Gonzalez, D., Kim, W., West, S.A. & Foster, K.R. 2018. Bacteria use collective behavior to generate diverse combat strategies. Current Biology 28: 1–11.

Nedelcu, A.M., Driscoll, W.W., Durand, P.M., Herron, M.D. & Rashidi, A. 2010. On the paradigm of altruistic suicide in the unicellular world. Evolution 65: 3–20.

Niehus, R., Oliveira, N.M., Li, A., Fletcher, A.G. & Foster, K.R. 2021. The evolution of strategy in bacterial warfare via the regulation of bacteriocins and antibiotics. eLife 10:e69756.

Reeves, P. 1965. The Bacteriocins. Bacteriological Review 29: 24–45.

Riley, M.A. & Chavan, M.A. 2006. Bacteriocins: Ecology and Evolution. Springer-.

Riley, M.A. & Wertz, J.E. 2002. Bacteriocin diversity: ecological and evolutionary perspectives. Biochimie 84: 357–364.

van Dyken, J.D. 2010. The components of kin competition. Evolution 64: 2840–2854.

West, S.A., Griffin, A.S. & Gardner, A. 2007. Social semantics: altruism, cooperation, mutualism, strong reciprocity and group selection. Journal of Evolutionary Biology 20: 415–432.

Wloch-Salamon, D.M., Gerla, D., Hoekstra, R.F. & de Visser, J.A.G.M. 2008. Effect of dispersal and nutrient availability on the competitive ability of toxin-producing yeast. Proceedings of the Royal Society B: Biological Sciences 275: 535–41.

Figure 1

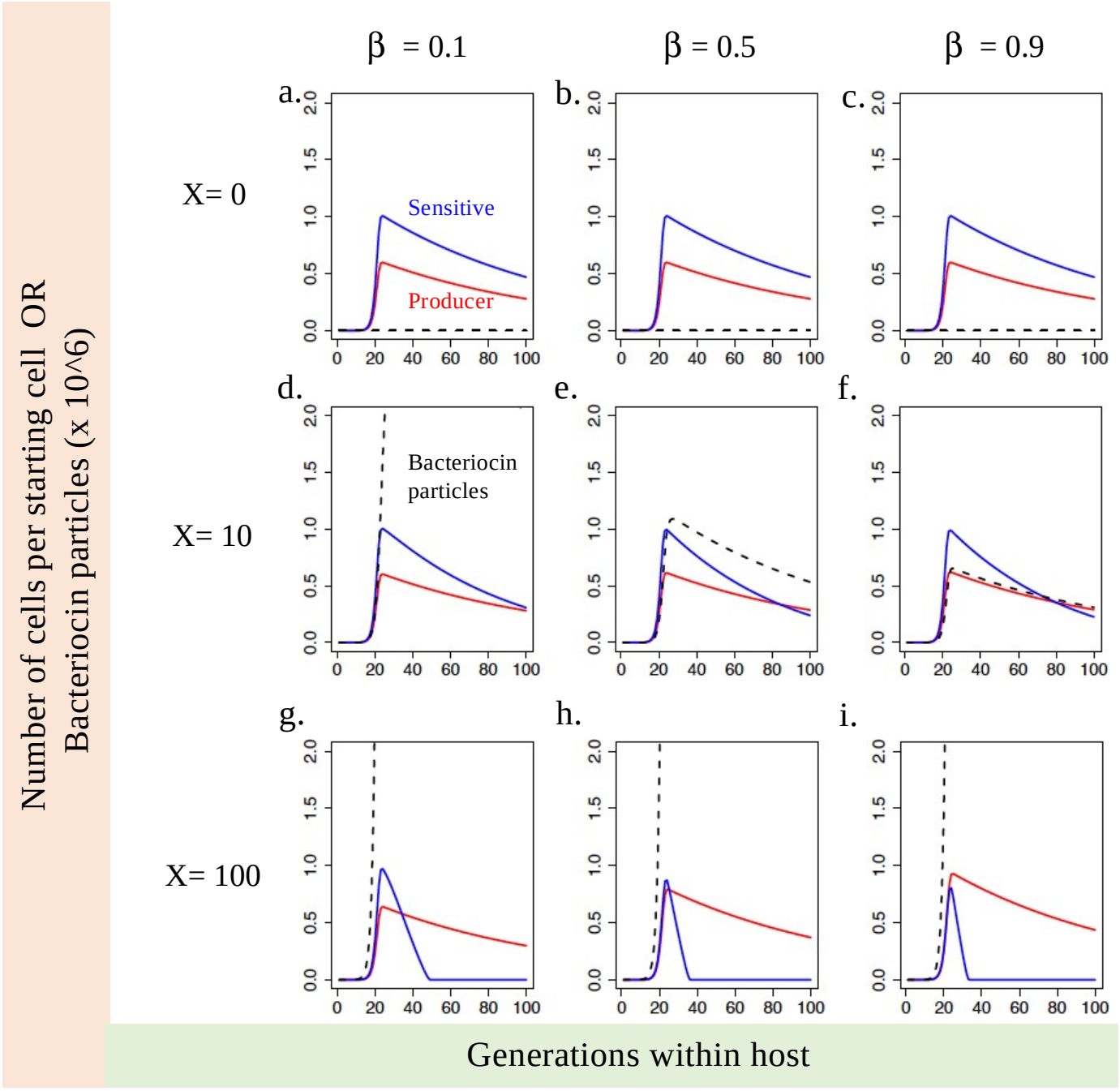


Figure 1: Within-host competitive dynamics of bacteriocin producing cells (red) and bacteriocin sensitive competitors (blue). Red and blue lines represent the number of producer or sensitive bacterial cells, respectively, relative to the initial number of cells of each type. Variable *x* represents the number of bacteriocin particles released by every dying producer cell. Variable β

represents the contact probability between bacteriocin particles and sensitive cells. Black dashed lines represent the total number of unused bacteriocin particles in the environment. Panels a-c represent competitive dynamics when no bacteriocin is not effective at killing sensitive cells. Here sensitive cells outcompete producer cells every time and the slower producer growth represents the growth cost of bacteriocin production incurred by producer cells. Producer cells can outcompete sensitive cells when $x > 0$ and do so faster with increasing contact probability $\beta$ between bacteriocins and sensitive cells (panels e – i) .

Figure 2

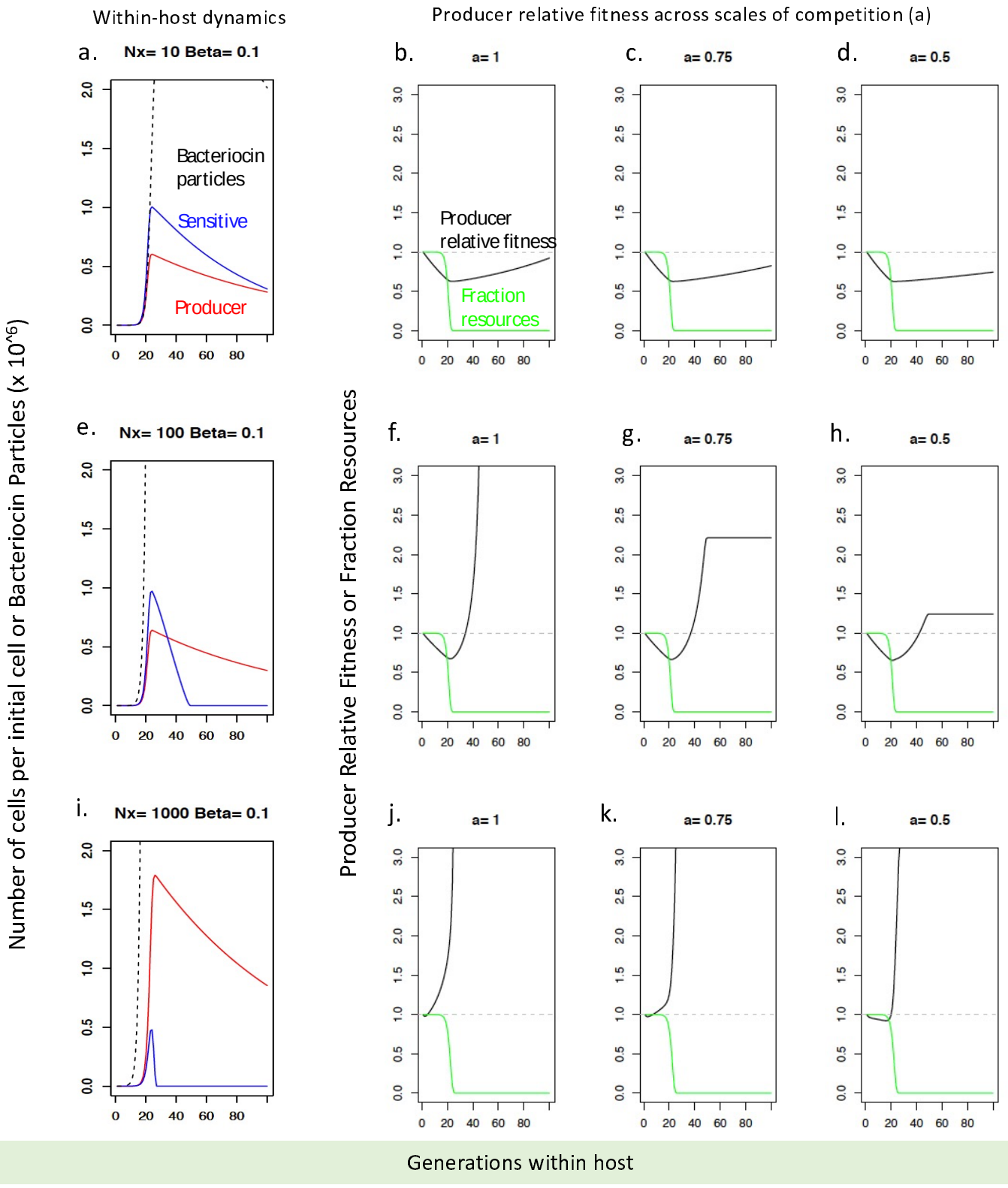
Within-host dynamics
Producer relative fitness across scales of competition (a)
a. Nx= 10 Beta= 0.1
b. a= 1
c. a= 0.75
d. a= 0.5
Bacteriocin particles
Sensitive
Producer
Producer relative fitness
Fraction resources
e. Nx= 100 Beta= 0.1
f. a= 1
g. a= 0.75
h. a= 0.5
i. Nx= 1000 Beta= 0.1
j. a= 1
k. a= 0.75
l. a= 0.5
Number of cells per initial cell or Bacteriocin Particles (x 10^6)
Producer Relative Fitness or Fraction Resources
Generations within host

Figure 2: Within-host dynamics and relative fitness of bacteriocin producer cells across the metapopulation of hosts. Fig 2 a, e and i represent the within-host growth of bacteriocin producers and sensitive competitors for varying number of bacteriocin particles released per dying producer cell (X). Panels 2 b-d, f-h, j-l show the relative fitness of producer cells across the metapopulation of hosts for varying values of scale of competition, ‘a’ (Frank, 1994). Scale of competition, a can range from completely local (a = 1) to completely global (a =0).

**Supplementary Material 1 : R code used to simulate within-host dynamics and produce figure 1.**

```
# Authors: Curt Lively and Amrita Bhattacharya
rm(list=ls()) # remove ALL objects from memory

####### Establish vectors to be filled###############

outProd = vector()  # outProd is for the producer
outSens = vector()  # outSens is for the sensitive strain.
outSens0 = vector()  # outSens is for the sensitive strain in sensitives only.
outR = vector()  # output for resources
outR0 = vector()  # output for resources in senstives only run
outCount = vector()  # output for counter
outTotRinSys = vector()  # output for totalresources in system
outTotNofDeaths = vector()  # output for total deaths
outTotBirths = vector()
outNtot = vector()
outX = vector()
Wfocal =vector()
Wlocal = vector()
Wglobal = vector()
Wrel = vector()

################## KEY VARIABLES ###############
T = 100 # T is the number of time steps in addition to time step 0
time = c(1:T)  #sets up do loop for i = 1 to T
b = 1 # b is the intrinsic birth rate
d = 0.01  # d is the intrinsic death rate
c = 0.05  # c (for cost) is the selection against producers (the selection coefficient)
Ncells = 100 #number of total cells entering host.  Leave fixed at 100
N = 10 #N is the number of producer cellss at time 0
S = (Ncells - N) # S is the number of sensitives at time 0.  This cannot be zero, as we have div by S
R = 10^8 # Resource units at time 0
S0 = N + S #number of sensitives in sensitive only.
R0 = R # Resource units at time 0 in sensitives only
X = 0 #number of toxin molecules at time 1
toggle = 0 ### Use 0 for DI deaths   use 1 for DD deaths.
NewRes = 0  #new resources into system every time step.
Ntot = N + S  # total number of cells
D = d*(Ntot/(Ntot + R))  # intrinsic death rate is density dependent.
# D does not count the deaths due to toxin production (see z)
D0 = d*(S0/(S0 + R0))  # intrinsic death rate is density dependent.
# D does not count the deaths due to toxin production (see z)
B = b*(1 - Ntot/(Ntot + R)) # birth rate is density dependent.
B0 = b*(1 - S0/(S0 + R0)) # birth rate is density dependent sensitives only.
```

```
TotSenDeaths = 0
TotProdDeaths = 0
TotDeaths = TotSenDeaths+TotProdDeaths
TotResInSystem = Ntot + R

# do loop for T time steps of recursions.  "{" begins the loop.  "}" ends the loop
par(mfrow = c(3,3))
for(Nx in c(0,10,100)){

  for (Beta in c(0.1,0.5,0.9))  {


for(i in 1:T)
{

  D[i] = ifelse(toggle > 0, D[i], d)
  D0[i] = ifelse(toggle > 0, D0[i], d)
 #N is the number of producers
  N[i+1] = N[i] - (N[i]*D[i]) + (N[i]*B[i]*(1-c))
  N[i+1] = ifelse(N[i+1] < 0, 0, N[i+1])  # "ifelse" is the if/then/else command.  Prevents N<0
  #N[i+1]=floor(N[i+1])
  outProd[i] = N[i]/N
  S[i+1] = S[i] - (S[i]*D[i]) + (S[i]*B[i]) - S[i]*(1-exp(-Beta*X[i]/(S[i]))) ## exp(-X[i]/S[i])
gives the prob of the 0 class (the survivors)
  S[i+1] = ifelse(S[i+1] < 0, 0, S[i+1])  #Prevents S<0
  #S[i+1]=floor(S[i+1]) ## rounds down to whole integer
  ## if S goes to zero the we get an NA error, as the exponential term is undefined.
  outSens[i] = S[i]/(Ncells - N)

  S0[i+1] = S0[i] - (S0[i]*D0[i]) + (S0[i]*B0[i])
  S0[i+1] = ifelse(S0[i+1] < 0, 0, S0[i+1])
  outSens0[i] = S0[i]

  Ntot[i+1] = N[i+1] + S[i+1]
  outNtot[i] = Ntot[i]

  R[i+1] = R[i] - (N[i]*B[i]*(1-c)) - (S[i]*B[i]) + NewRes
  R[i+1] = ifelse(R[i+1]<0, 0, R[i+1])
  outR[i] = R[i]

  R0[i+1] = R0[i] - (S0[i]*B0[i]) + NewRes
  R0[i+1] = ifelse(R0[i+1]<0, 0, R0[i+1])
  outR0[i] = R0[i]

  D[i+1] = d*(Ntot[i+1]/(Ntot[i+1]+R[i+1]))
  B[i+1] = b*(1 - Ntot[i+1]/(Ntot[i+1]+R[i+1]))
```

```
  B0[i+1] = b*(1 - S0[i+1]/(S0[i+1]+R0[i+1]))

  X[i+1] = 0.95*X[i] + (D[i]*N[i]*Nx) - S[i]*(1-exp(-Beta*X[i]/(S[i])))
  outX[i] = X[i]
}

plot(time,outProd/10^6, type="l", lwd = 2, xlab = "", ylab = "", xaxt = 'n', yaxt = 'n', las = '1', col
= "red",main = paste("Nx=", Nx, "Beta=",Beta), ylim=range(c(outProd/10^6,2))) #,
lines(time, outSens/10^6,type="l", lwd = 2, col = "blue", ylim=range(c(outProd/10^6,2)))
lines(time, outX/10^6,type="l", lty=2, lwd =2, col = "black", ylim=range(c(outProd/10^6,2)))
axis(2, cex.axis=1.5)
axis(1, cex.axis=1.5)
 }
}
```

**Supplementary Material 2 R code used to simulate between-host competition dynamics and produce figure 2.**

Authors: Curt Lively and Amrita Bhattacharya

```
rm(list=ls()) # remove ALL objects from memory
# Establish vectors to be filled
outProd = vector()  # outProd is for the producer
outSens = vector()  # outSens is for the sensitive strain.
outSens0 = vector()  # outSens is for the sensitive strain in sensitives only.
outR = vector()  # output for resources
outR0 = vector()  # output for resources in senstives only run
outCount = vector()  # output for counter
outTotRinSys = vector()  # output for totalresources in system
outTotNofDeaths = vector()  # output for total deaths
outTotBirths = vector()
outNtot = vector()
outX = vector()
Wfocal =vector()
Wlocal = vector()
Wglobal = vector()
Wrel = vector()

################## KEY VARIABLES ###############
Beta = 0.1 #contact rate, similar to transmission coeffiecent.  Takes into account mixing within
the host
T = 100 # T is the number of time steps in addition to time step 0
time = c(1:T)  #sets up do loop for i = 1 to T
b = 1 # b is the intrinsic birth rate
d = 0.01  # d is the intrinsic death rate
c = 0.05  # c (for cost) is the selection against producers (the selection coefficient)
Ncells = 100 #number of total cells entering host.
N = 10 # N is the number of producer cells at time 0
S = (Ncells - N) # S is the number of sensitives at time 0.  This cannot be zero, as we have div by S
R = 10^8 # Resource units at time 0
S0 = N + S #number of sensitives in sensitive only.
R0 = R # Resource units at time 0 in sensitives only
X = 0 #number of toxin molecules at time 1
toggle = 0 ### Use 0 for DI deaths   use 1 for DD deaths.
NewRes = 0  #new resources into system every time step.
Ntot = N + S  # total number of cells
D = d*(Ntot/(Ntot + R))  # intrinsic death rate is density dependent.
D0 = d*(S0/(S0 + R0))  # intrinsic death rate is density dependent.
B = b*(1 - Ntot/(Ntot + R)) # birth rate is density dependent.
B0 = b*(1 - S0/(S0 + R0)) # birth rate is density dependent sensitives only.
TotSenDeaths = 0
```

```
TotProdDeaths = 0
TotDeaths = TotSenDeaths+TotProdDeaths
TotResInSystem = Ntot + R

# do loop for T time steps of recursions.  "{" begins the loop.  "}" ends the loop
par(mfrow = c(3,4))
for(Nx in c(10,100,1000)){
  for (a in c(1.0, 0.75, 0.5))  {
for(i in 1:T)
{
  D[i] = ifelse(toggle > 0, D[i], d)
  D0[i] = ifelse(toggle > 0, D0[i], d)
  Wfocal[i] = N[i]/N  ##here N is starting number of producers
  Wlocal[i] = (N[i] + S[i])/(Ncells)
  Wglobal[i] = S0[i]/(Ncells)
  Wrel[i] = Wfocal[i]/(a*Wlocal[i] + (1 - a)*Wglobal[i])
  #N is the number of producers
  N[i+1] = N[i] - (N[i]*D[i]) + (N[i]*B[i]*(1-c))
  N[i+1] = ifelse(N[i+1] < 0, 0, N[i+1])  # "ifelse" is the if/then/else command.  Prevents N<0
  outProd[i] = N[i]/N
  S[i+1] = S[i] - (S[i]*D[i]) + (S[i]*B[i]) - S[i]*(1-exp(-Beta*X[i]/(S[i])))
  S[i+1] = ifelse(S[i+1] < 0, 0, S[i+1])  #Prevents S<0
  outSens[i] = S[i]/(Ncells - N)
  S0[i+1] = S0[i] - (S0[i]*D0[i]) + (S0[i]*B0[i])
  S0[i+1] = ifelse(S0[i+1] < 0, 0, S0[i+1])  #Prevents S0<0
  outSens0[i] = S0[i]
  Ntot[i+1] = N[i+1] + S[i+1]
  outNtot[i] = Ntot[i]
  R[i+1] = R[i] - (N[i]*B[i]*(1-c)) - (S[i]*B[i]) + NewRes
  R[i+1] = ifelse(R[i+1]<0, 0, R[i+1])  #prevents negative Resource values
  outR[i] = R[i]
  R0[i+1] = R0[i] - (S0[i]*B0[i]) + NewRes
  R0[i+1] = ifelse(R0[i+1]<0, 0, R0[i+1])  #prevents negative Resource values
  outR0[i] = R0[i]
  D[i+1] = d*(Ntot[i+1]/(Ntot[i+1]+R[i+1]))
  D0[i+1] = d*(S0[i+1]/(S0[i+1]+R0[i+1]))
  B[i+1] = b*(1 - Ntot[i+1]/(Ntot[i+1]+R[i+1]))
  B0[i+1] = b*(1 - S0[i+1]/(S0[i+1]+R0[i+1]))
  X[i+1] = 0.95*X[i] + (D[i]*N[i]*Nx) - S[i]*(1-exp(-Beta*X[i]/(S[i])))
  outX[i] = X[i]
}

if (a > 0.9){
plot(time,outProd/10^6, type="l", xlab = "", ylab = "", col = "red",
ylim=range(c(outProd/10^6,2)), main = paste("Nx=", Nx, "Beta=",Beta))
par(new = TRUE)
```

```
plot(time,outSens/10^6, type="l", xlab = "", ylab = "", main = "", col = "blue",
ylim=range(c(outProd/10^6,2)))
par(new = TRUE)
plot(time,outX/10^6, type="l",lty = 2, xlab = "", ylab = "", main = "", col = "black",
ylim=range(c(outProd/10^6,2)))

###############Between hosts######################
plot(time, Wrel, type ='l', xlab = "", ylab = "", col ="black", ylim = range(c(0,3)), main =
paste("a=",a))
abline(h = 1, lty = 2, col = "grey")
lines(time, outR/R[1], type = 'l', col = "green")
}
else
 {
plot(time, Wrel, type ='l', xlab = "", ylab = "", col ="black", ylim = range(c(0,3)), main =
paste("a=",a))
abline(h = 1, lty = 2, col = "grey")
lines(time, outR/R[1], type = 'l', col = "green")}
}
}
```