AMATH 422 - Computational Models in Biology
Jacob Kovacs
3/18/2015

Varying spatial dependence in the evolution of bacterial biofilms under nontransitive competition

Introduction

Nontransitive competition is an ecological relationship where, as in the game of paper-scissors -rock, species A dominates B, species B dominates C, and species C dominates A in circular fashion. Nontransitive competition may occur in a multistrain community of *Escherichia coli* bacteria when members of a toxin-producing strain (P), a toxin-resistant strain (R), and a toxin-sensitive strain (S) inhabit the same region and have strain-specific average growth rates (g_i) satisfying the inequality:

$$g_P < g_R < g_S$$

An equivalent condition may be phrased in terms of costs (c_i) . Toxin production has a cost since resources are diverted from growth and longevity and invested in toxin production. Resistance to a toxin likewise carries a cost. When toxin-producers and toxin-resisters face the following relative costs, the growth rates inequality above will also be satisfied:

$$c_R < c_P$$

Since the costs of toxin-production and toxin-resistance reduce longevity, the constraint may also be expressed in terms of death rates, Δ_i , and a toxicity constant, T:

$$\Delta_{S0} < \Delta_R < \Delta_P < \frac{\Delta_S + \tau}{1 + \tau}$$

In addition to experiencing nontransitive competition, *E. coli* may form biofilms and thus constitute a spatially-structured community. Spatial structure means that individual bacteria are fixed in space and interact with their close neighbors, instead of floating freely and interacting with many more members of the population. Many researchers have found spatial structure is an important determinant of ecological dynamics (e.g. Durrett & Levin, 1997; Durrett & Levin, 1998; Kerr et al., 2002), finding in general that the effect of spatial structuring is to permit more complex dynamics to emerge.

Prado and Kerr (2008) simulate a nontransitive and spatially-structured *E. coli* community wherein members of the resistant strain, R, experience mutation, making them more or less resistant to the attacks of their toxin-producing neighbors and correspondingly more or less long-lived (since they pay a price for resistance). Prado and Kerr find that optimal evolutionary strategy in these circumstances is restraint. That is, members of the R strain maximize their share of the population by *not* minimizing their death rate (equivalently, by *not* maximizing their resistance to toxin). This paper attempts to replicate one of Prado and Kerr's findings, and then runs a new simulation using different specifications for spatial dependence.

Description of Prado and Kerr's computational model

An agent-based model is well suited for simulating a spatially-structured population of bacteria that evolves at the level of its individual members (agents). The structure is supplied by the way agents are embedded together in a matrix, and governed by rules that may depend on the value of neighboring

cells. The size of an agent's neighborhood is one important variable in the model: for instance, if the neighborhood is defined to encompass the entire population, then the community is no longer structured.

The simulation begins by randomly populating an LxL matrix with numeric codes for the values R, P, S, and E, representing three strains of bacteria and empty cells. A second LxL matrix is created to allocate and store resistance parameters corresponding to each member of strain R in the first matrix (Fig. 1). The initial resistance values are drawn randomly from the interval [0.275, 0.329], a constraint defined to preserve nontransitivity.

The simulation is run for 10 epochs, with an epoch defined as LxL iterations. At each iteration a single agent is randomly chosen from the population. This agent, the focal point, is then altered in accordance with the transition matrix summarized in Table 1. Possible transitions represent births and deaths of bacteria: an empty cell may become populated with a member of S, R, or P, while an occupied cell may either remain occupied or revert to an empty cell, E.

Figure 1. Visualization of two matrices used in the simulation. On the left, a 30x30 population of bacteria (strains R, S, and P, plus empty cells, E). On the right, a matrix storing resistance parameters for each member of strain R.

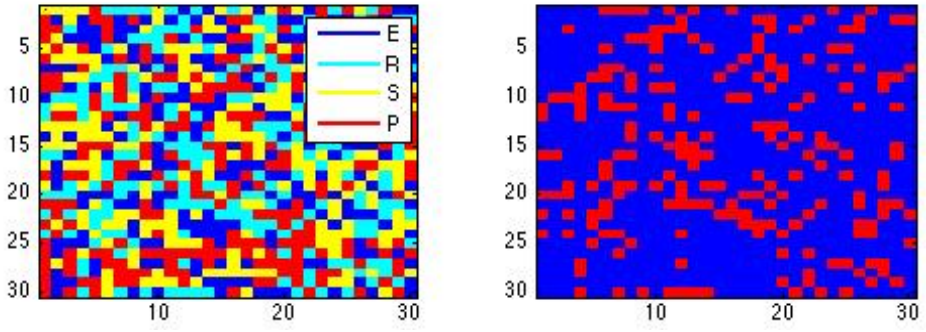


Table 1. Transition matrix. Cell a_{ij} of the transition matrix specifies the probability that a strain of type *j* at time *n* will transition to type *i* at time *n*+1. Probabilities refer to birth events (f_i) and death events (Δ_i).

	S	**P**	**R**	**E**
S	$1 - \Delta_S$	0	0	f_S
P	0	$1 - \Delta_P$	0	f_P
R	0	0	$1 - \Delta_R$	f_R
E	Δ_S	Δ_P	Δ_R	$1 - f_S - f_P - f_R$

Table 2. Definitions of parameters and variables.

E	Empty cell in the matrix	R	Strain that is toxin-resistant
S	Strain that is sensitive to toxin	Δ_R	Death rate of strain R; also R's resistance to toxin
Δ_{S0}	Baseline death rate of strain S	p_R	Probability that resistance level will mutate
P	Strain that produces toxin	i_R	Size of mutation
Δ_P	Death rate of strain P	f_i	Proportion of strain *i* in the neighborhood of the focal point
T	Toxicity of strain P		

Some birth or death probabilities in the transition matrix are spatially dependent in a way that is quite intuitive. The proportion of strain *i* in the neighborhood of a given focal point is denoted f_i , and treated as the probability that a member of strain *i* will be born into that focal point. Thus the more potential parents in the area, the higher the likelihood that they will procreate. Similarly, the probability that S will die increases with the presence of toxin-producers. Prado and Kerr specify a linear relationship where the proportion of toxin-producing bacteria in the neighborhood is multiplied by per capita toxicity (a constant) and added to the intrinsic death rate of the sensitive strain: $\Delta_S = s0 + T f_P$. The remaining two parameters in the transition matrix are the death rates of P and R. The intrinsic death rate of P, Δ_P, is defined as constant, while the death rate of R is allowed to vary through mutation.

Prado and Kerr are vague on some important details regarding their implementation of evolution. They say "[t]o incorporate evolution in our simulations, we allow every cell to carry its own genotype *g* and when an offspring is 'born', a mutation can occur to change the genotype" (Prado & Kerr, 2008, p. 540). This reads straightforwardly, but applying it to the possible transitions spelled out by the transition matrix reveals the ambiguity. The relevant transitions are as follows: if the focal point holds a member of strain R, and if this R does not die, it may mutate. If the focal point is an empty cell, it may become populated with a newborn R. The first case is easy enough to implement; the second is the one for which Prado and Kerr do not provide guidance.

In the first case, when a member of R is selected as the focal point in a given timestep, the probability that R will die can be retrieved from the storage matrix, where its address in terms of matrix rows and columns corresponds with that of the focal point. If R survives, then it may mutate with probability 0.01. A random mutation increment, i_R, is drawn from a uniform distribution between -0.001 and 0.001 and added to R's current resistance value. This sum is checked against the nontransitivity constraints specified above and the final value is recorded in the storage matrix.

In the second case, when an empty cell is selected as the focal point in a given timestep, the probability that it will be populated by a member of strain R depends on the concentration of strain R within the neighborhood (f_i). This new R must be assigned a resistance value, which is then recorded in the corresponding cell of the storage matrix. Prado and Kerr do not identify their method for assigning a resistance value to a new R. Reasonable options are a random draw from the interval [0.275, 0.329]; the midpoint of this interval, 0.302; or an average of the resistance values of parent R's within the neighborhood of the focal point.

In either case, at the end of each passing epoch once a number of mutations have occurred, the average of all resistance values in the storage matrix is calculated and plotted. This reveals any trends in the evolution of strain R's resistance levels.

Results of simulations

Replicating Prado and Kerr: presence or absence of spatial structure and/or nontransitive competition

With a few adjustments to the model described above, Prado and Kerr explore how spatial structuring, nontransitive competition, or a combination of both affect the evolution of strain R's average resistance level. To simulate an unstructured community, they expand the size of the neighborhood to include the entire LxL matrix (rather than a focal point's immediate neighbors, their specification for the structured

case). To simulate nontransitive dynamics, they remove strains S and P from the matrix and allow strain R to evolve by itself, still subject to the same constraints on mutation. Their results from this exercise are presented in figure 2. I was unable to reproduce Prado and Kerr's results (see Appendix for MATLAB code); my results are presented in figure 3.

Altering spatial dependence in S's death rate

I adjusted the specification for spatial dependence by using a nonlinear functional forms for the death rate of S: $\Delta_S = s0 + T f_P + T f_P^2$ and $\Delta_S = s0\ T f_P^2$. In theory, this would intensify the effect that neighboring toxin-producers have on S, and thereby reduce competitive pressures on the toxin-resistant strain. Results of six simulations are presented together in figure 4. Use of a nonlinear form does not appear to have any significant effect on the evolution of R's death rate. All cases exhibit average death rates that decline at a comparable pace, with no clear tendency for the linear case to behave differently from the two nonlinear cases.

Figure 2. Evolution of average death rate of the resistant strain for four different scenarios. To determine transition probabilities, structured scenarios (St.) reference a small neighborhood surrounding the focal point while unstructured scenarios (Un.) reference the entire population. For some scenarios, the population consists of three strain types (All): toxin-sensitive, toxin-resistant, and toxin-producing. For other scenarios, the population is entirely toxin-resistant (R). The structured case where all three strains are present (St. All) exhibits restraint by failing to minimize its death rate; the other three scenarios evolve rapidly towards their minimum. *Source*: Prado & Kerr, 2008.

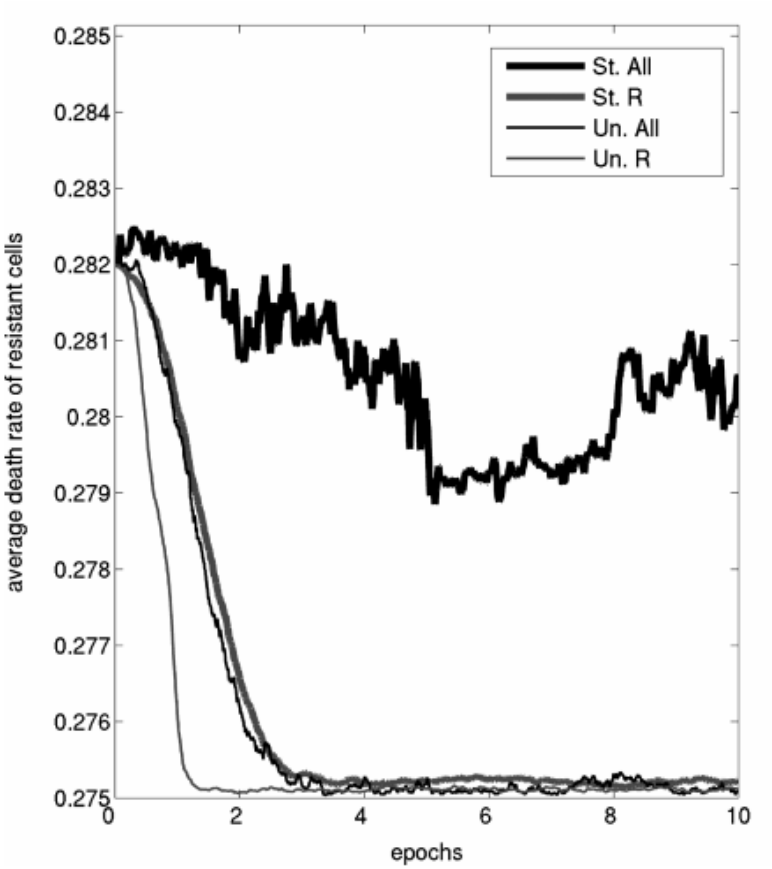


Figure 3. Evolution of average death rate of the resistant strain for four different scenarios. Attempted replication of Figure 2. All cases exhibit declining average death rates, none as dramatic as Prado and Kerr's; three cases may be leveling out at a stable value while the structured three-strain case continues declining, exactly the opposite of Prado and Kerr's findings.

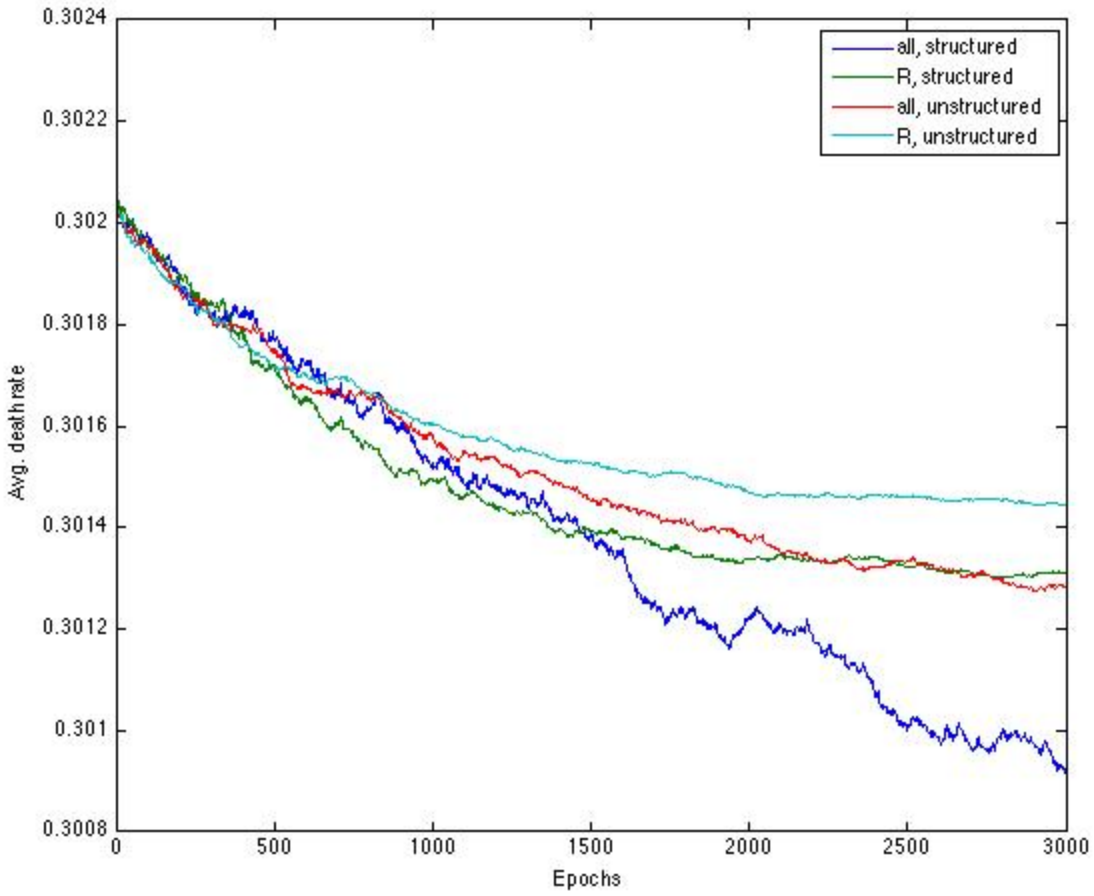
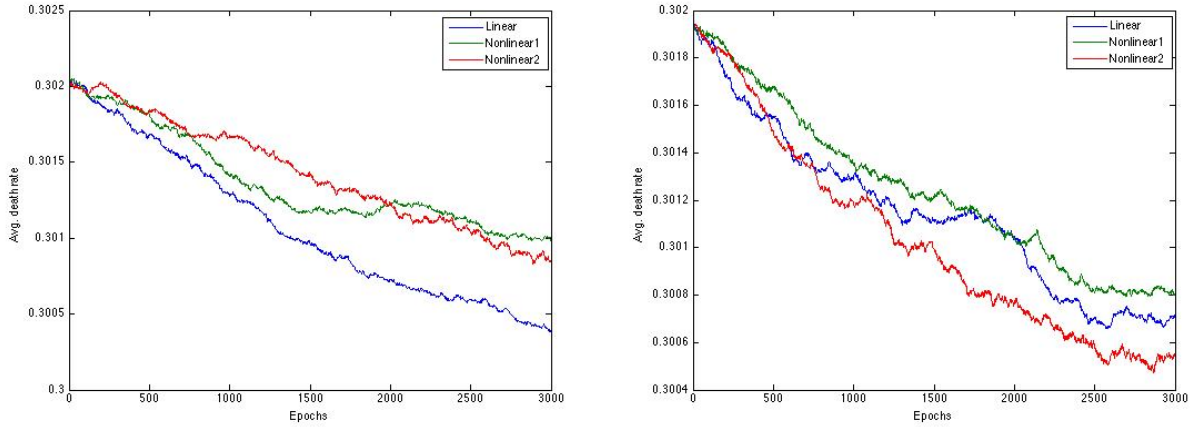
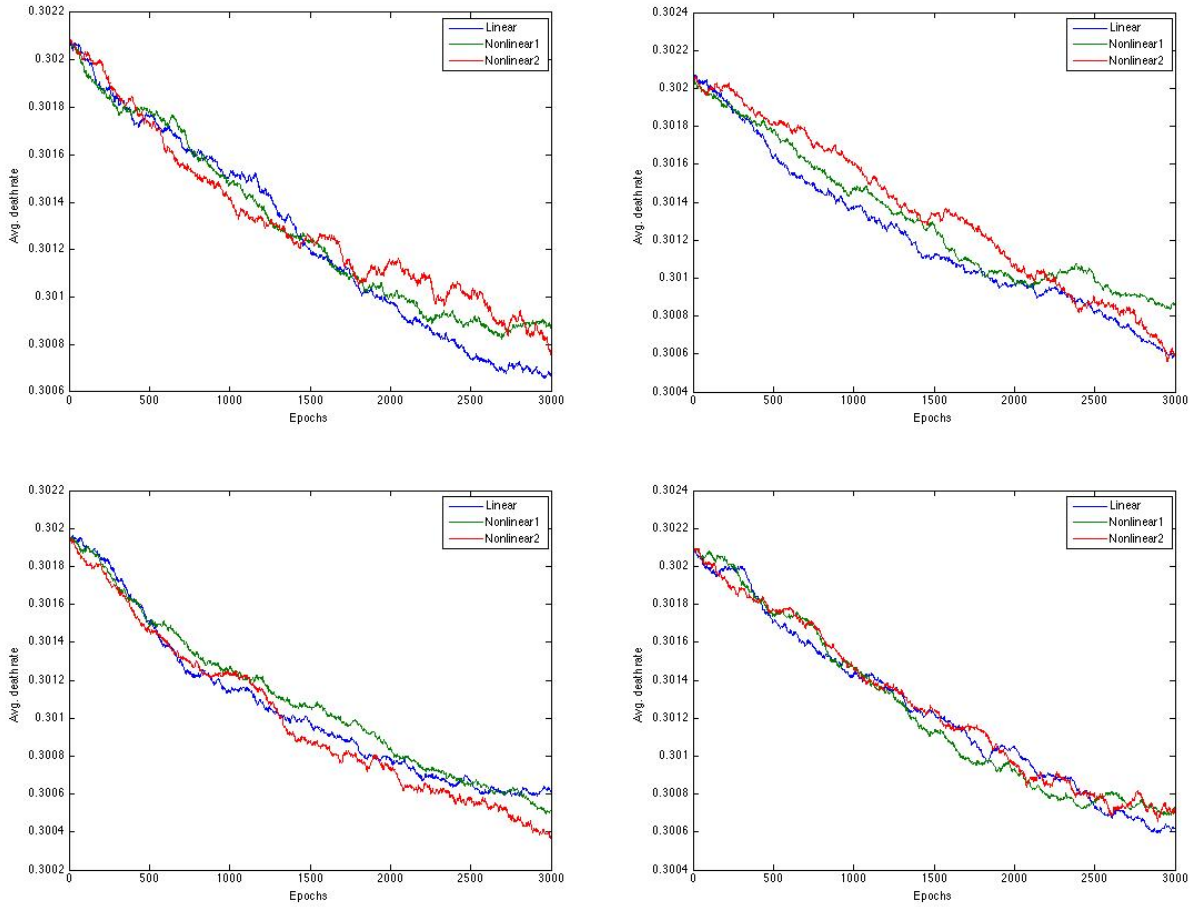


Figure 4. Evolution of average resistance levels comparing linear and nonlinear specifications for S's death rate, using $\Delta_S = s0 + T f_P$ ('Linear', in blue), $\Delta_S = s0 + T f_P + T f_P^2$ ('Nonlinear1', in green) and $\Delta_S = s0\ T f_P^{\ 2}$ ('Nonlinear2', in red).





Conclusion

This paper was unable to replicate Prado and Kerr's (2008) finding of restraint in nontransitive, spatially-structured populations, most likely because of unidentified errors in the code used. Modifying the specification for spatial dependence had no obviously interesting effects. Additional efforts to modify the spatial structure (by adjusting the size of the neighborhood, to explore the effects of scale in between the extremes of structured/unstructured used by Prado and Kerr) were not successful. With additional time, I would focus on replicating Prado & Kerr's results and develop a way to implement a mid-sized neighborhood in my code.

References

Durrett, R., & Levin, S. (1997). Allelopathy in spatially distributed populations. *Journal of Theoretical Biology, 185*(2), pp. 165-171.

Durrett, R., & Levin, S. (1998). Spatial aspects of interspecific competition. *Theoretical Population Biology, 53*(1), pp. 30-43.

Kerr, B., Riley, M. A., Feldman, M. W., & Bohannan, B. J. M. (2002). Local dispersal promotes biodiversity in a real-life game of rock–paper–scissors. *Nature, 418*, pp. 171-174.

Prado, F. & Kerr, B. (2008). The evolution of restraint in bacterial biofilms under nontransitive competition. *Evolution, 62*(3), pp. 538-548.

Appendix

```
clear all; close all; clc

%% DEFINE PARAMETERS
   rand('state', sum(100*clock))
   L=300; % size of lattice
   epoch=3000; its=300; % duration of simulation = 9,000,000 iterations
   R=1; S=2; P=3; E=0; % codes for types of bacteria
   dr_min=0.275; dr_max=0.329; % constraints preserving nontransitive competition
   prm=0.01; % probability that R mutates
   s0=0.25; % intrinsic death rate of strain S
   dp=0.3333; % intrinsic death rate of strain P
   T=0.65; % toxicity of strain P

%% RANDOMLY POPULATE LATTICES
   [LM1,resM1]=pop_all(L,R,S,P,E,dr_min,dr_max); % initial pop, 3 strains
   LM2=LM1; resM2=resM1;
   [LM3,resM3]=pop_R(L,R,E,dr_min,dr_max); % initial pop, 1 strain
   LM4=LM3; resM4=resM3;

  % TEST CODE, checking population (LM) and resistance (resM) matrices
%    figure
%    subplot(2,2,1)
%    N=4;
%    imagesc(LM1)
%    cmap = jet(N);
%    colormap(cmap), hold on
%    Li=line(ones(N),ones(N), 'LineWidth',2);
%    set(Li,{'color'},mat2cell(cmap,ones(1,N),3));
%    legend('E','R','S','P')
%    subplot(2,2,2)
%    imagesc(resM1)
%    subplot(2,2,3)
%    I=find(resM1~=0); a=resM1(I);
%    hist(a)
%    %max(max(resM1))
%    %b=resM1>0; min(resM1(b))
%    I=find(resM1~=0); [m,n]=size(resM1(I)); sum(resM1(I))/m
%    I=LM1==R; a=sum(sum(I)); I=LM1==S; b=sum(sum(I)); I=LM1==P; c=sum(sum(I));
%    a+b+c
%    figure
%    subplot(2,2,1)
%    N=2;
%    imagesc(LM3)
%    cmap = jet(N);
%    colormap(cmap), hold on
%    Li=line(ones(N),ones(N), 'LineWidth',2);
%    set(Li,{'color'},mat2cell(cmap,ones(1,N),3));
%    legend('E','R')
%    subplot(2,2,2)
%    imagesc(resM3)
```

```
%    subplot(2,2,3)
%    I=find(resM3~=0); a=resM3(I);
%    hist(a)

%% EVOLVE POPULATIONS
   [avgR1]=evolve_scen1(L,R,S,P,E,T,s0,dp,dr_min,dr_max,prm,LM1,resM1,epoch,its);
   %avgR6 and avgR6 come from exact copies of the evolve_scen1 function,
   % except one line changing the functional form of ds
   [avgR2]=evolve_scen2(L,R,S,P,E,T,s0,dp,dr_min,dr_max,prm,LM2,resM2,epoch,its);
   [avgR3]=evolve_scen3(L,R,E,dr_min,dr_max,prm,LM3,resM3,epoch,its);
   [avgR4]=evolve_scen4(L,R,E,dr_min,dr_max,prm,LM4,resM4,epoch,its);

   et=1:epoch;
   plot(et,avgR1, et,avgR2, et,avgR3, et,avgR4)
   legend('all, structured','R, structured','all, unstructured','R, unstructured')
   %plot(et,avgR1, et,avgR5, et,avgR6)
   %legend('Linear','Nonlinear1','Nonlinear2')
   xlabel('Epochs')
   ylabel('Avg. death rate')
```

```
function [LM,resM] = pop_all(L,R,S,P,E,dr_min,dr_max)
   rand('state', sum(100*clock))
   % RANDOMLY POPULATE LATTICE
   LM=rand(L,L);
   for i=1:L
      for j=1:L
         a=LM(i,j);
         if a < 0.25
            LM(i,j)=R;
         elseif a < 0.5
            LM(i,j)=S;
         elseif a < 0.75
            LM(i,j)=P;
         else
            LM(i,j)=E;
         end
      end
   end
   % CONSTRUCT STORAGE MATRIX FOR RESISTANCE PARAMETERS
   resM=dr_min + (dr_max - dr_min)*rand(L,L); % matrix of random dr w/in [0.275, 0.329]
   I=find(LM~=R); resM(I)=0; % zeros out dr values corresponding to non-R cells
end
```

```
function [LM,resM] = pop_R(L,R,E,dr_min,dr_max)
   rand('state', sum(100*clock))
   % RANDOMLY POPULATE LATTICE
   LM=rand(L,L);
   for i=1:L
      for j=1:L
         a=LM(i,j);
         if a < 0.5
```

```
            LM(i,j)=R;
        else
            LM(i,j)=E;
        end
    end
end
% CONSTRUCT STORAGE MATRIX FOR RESISTANCE PARAMETERS
resM=dr_min + (dr_max - dr_min)*rand(L,L);
I=find(LM~=R); resM(I)=0;
end
```

```
function [avgR1] = evolve_scen1(L,R,S,P,E,T,s0,dp,dr_min,dr_max,prm,LM1,resM1,epoch,its)

    for j=1:epoch

        % CALCULATE AVG RESISTANCE
        I=find(resM1~=0);
        avgR1(1,j)=mean(mean(resM1(I)));

        for i=1:its

            % RANDOMLY SELECT FOCAL POINT
            Lb=linspace(0,1,L); rd1=rand; rd2=rand;
            Lb1=find(Lb<=rd1); Lb2=find(Lb<=rd2);
            Lrow=Lb1(end); Lcol=Lb2(end);
            LM1(Lrow,Lcol); % focal point

            % TEST CODE: do focal point draws look appropriately random?
            %plot(i,LM1(Lrow,Lcol),'r+'), hold on

            % DEFINE NEIGHBORHOOD OF FOCAL POINT
            n=1; % 'radius' of square neighborhood
            if Lrow-n>=1 && Lcol-n>=1 % problem-free scenario, not an edge case
                LN=LM1(Lrow-n:Lrow+n, Lcol-n:Lcol+n); % defining the neighborhood
                rLN=resM1(Lrow-n:Lrow+n, Lcol-n:Lcol+n);
            elseif Lrow-n<1 % spilled over north edge of lattice
                if Lcol-n<1 % also spilled over west edge
                    LN=LM1(1:Lrow+n,1:Lcol+n); % defining the neighborhood
                    rLN=resM1(1:Lrow+n,1:Lcol+n);
                elseif Lcol+n>L % " east edge
                    LN=LM1(1:Lrow+n, Lcol-n:L);
                    rLN=resM1(1:Lrow+n, Lcol-n:L);
                else % column position is OK
                    LN=LM1(1:Lrow+n, Lcol-n:Lcol+n);
                    rLN=resM1(1:Lrow+n, Lcol-n:Lcol+n);
                end
            elseif Lrow+n>L % " south edge
                if Lcol-n<1 % " west edge
                    LN=LM1(Lrow-n:L, 1:Lcol+n);
                    rLN=resM1(Lrow-n:L, 1:Lcol+n);
                elseif Lcol+n>L % " east edge
                    LN=LM1(Lrow-n:L, Lcol-n:L);
```

```
        rLN=resM1(Lrow-n:L, Lcol-n:L);
    else
        LN=LM1(Lrow-n:L, Lcol-n:Lcol+n);
        rLN=resM1(Lrow-n:L, Lcol-n:Lcol+n);
    end
elseif Lcol-n<1 % spilled over west edge, but row position is OK (per above)
    LN=LM1(Lrow-n:Lrow+n, 1:Lcol+n);
    rLN=resM1(Lrow-n:Lrow+n, 1:Lcol+n);
elseif Lcol+n>L % " east edge, "
    LN=LM1(Lrow-n:Lrow+n, Lcol-n:L);
    rLN=resM1(Lrow-n:Lrow+n, Lcol-n:L);
end

% COUNT EACH STRAIN WITHIN THE NEIGHBORHOOD
[m,n]=size(LN); nn=m*n; % size of neighborhood
C=find(LN==S); fs=length(C)/nn; % proportion of S
C=find(LN==P); fp=length(C)/nn;
C=find(LN==R); fr=length(C)/nn;

% REFER TO TRANSITION MATRIX & UPDATE FOCAL POINT
rd3=rand;
if LM1(Lrow,Lcol)==S
    ds=s0+T*fp;
    %ds=s0+T*fp+T*fp^2; % change to nonlinear functional form
    %ds=s0*T*fp;
    if rd3 < ds % does S die?
        LM1(Lrow,Lcol)=E;
    end
elseif LM1(Lrow,Lcol)==P
    if rd3 < dp
        LM1(Lrow,Lcol)=E;
    end
elseif LM1(Lrow,Lcol)==R
    if rd3 < resM1(Lrow,Lcol) % does R die?
        LM1(Lrow,Lcol)=E; % zeros out population matrix
        resM1(Lrow,Lcol)=0; % zeros out resistance matrix
    else
        rd4=rand;
        if rd4 < prm % does R mutate?
            drinc=0.002*rand -0.001; % by how much?
            a=resM1(Lrow,Lcol)+drinc;
            if a >= dr_max % check against nontransitive competition constraint
                resM1(Lrow,Lcol)=dr_max; % record mutation in resistance matrix
            elseif a <= dr_min
                resM1(Lrow,Lcol)=dr_min;
            else
                resM1(Lrow,Lcol)=a;
            end
        end
    end
else % if the cell is empty ...
    if rd3<fs % is an S born into it?
        LM1(Lrow,Lcol)=S;
```

```
            elseif rd3<(fs+fp) % or a P?
               LM1(Lrow,Lcol)=P;
            elseif rd3<(fs+fp+fr) % or an R?
               LM1(Lrow,Lcol)=R;
               I=find(rLN~=0);
               resM1(Lrow,Lcol)=mean(mean(rLN(I))); % child assigned parents' avg value
            end
         end
      end
   end
end
```

```
function [avgR2] = evolve_scen2(L,R,S,P,E,T,s0,dp,dr_min,dr_max,prm,LM2,resM2,epoch,its)

   for k=1:epoch

      % CALCULATE AVG RESISTANCE
      I=find(resM2~=0); avgR2(1,k)=mean(mean(resM2(I)));

      for i=1:its

         % RANDOMLY SELECT FOCAL POINT
         Lb=linspace(0,1,L); rd1=rand; rd2=rand;
         Lb1=find(Lb<=rd1); Lb2=find(Lb<=rd2);
         Lrow=Lb1(end); Lcol=Lb2(end);
         LM2(Lrow,Lcol);

         % GET GLOBAL COUNTS
         nn=L*L;
         C=find(LM2==S); fs=length(C)/nn;
         C=find(LM2==P); fp=length(C)/nn;
         C=find(LM2==R); fr=length(C)/nn;

         % REFER TO TRANSITION MATRIX & UPDATE FOCAL POINT
         rd3=rand;
         if LM2(Lrow,Lcol)==S
            ds=s0+T*fp;
            if rd3 < ds
               LM2(Lrow,Lcol)=E;
            end
         elseif LM2(Lrow,Lcol)==P
            if rd3 < dp
               LM2(Lrow,Lcol)=E;
            end
         elseif LM2(Lrow,Lcol)==R
            if rd3 < resM2(Lrow,Lcol)
               LM2(Lrow,Lcol)=E;
               resM2(Lrow,Lcol)=0;
            else
               rd4=rand;
               if rd4 < prm
                  drinc=0.002*rand-0.001;
```

```
                a=resM2(Lrow,Lcol)+drinc;
                if a >= dr_max
                   resM2(Lrow,Lcol)=dr_max;
                elseif a <= dr_min
                   resM2(Lrow,Lcol)=dr_min;
                else
                   resM2(Lrow,Lcol)=a;
                end
             end
          end
       else
          if rd3<fs
             LM2(Lrow,Lcol)=S;
          elseif rd3<(fs+fp)
             LM2(Lrow,Lcol)=P;
          elseif rd3<(fs+fp+fr)
             LM2(Lrow,Lcol)=R;
             I=find(resM2~=0); resM2(Lrow,Lcol)=mean(mean(resM2(I)));
          end
       end
    end
  end
end
```

```
function [avgR3] = evolve_scen3(L,R,E,dr_min,dr_max,prm,LM3,resM3,epoch,its)

  for l=1:epoch

    % CALCULATE AVG RESISTANCE
    I=find(resM3~=0); avgR3(1,l)=mean(mean(resM3(I)));

    for i=1:its

      % RANDOMLY SELECT FOCAL POINT
      Lb=linspace(0,1,L); rd1=rand; rd2=rand;
      Lb1=find(Lb<=rd1); Lb2=find(Lb<=rd2);
      Lrow=Lb1(end); Lcol=Lb2(end);
      LM3(Lrow,Lcol);

      % DEFINE NEIGHBORHOOD OF FOCAL POINT
      n=1;
      if Lrow-n>=1 && Lcol-n>=1
         LN=LM3(Lrow-n:Lrow+n, Lcol-n:Lcol+n);
         rLN=resM3(Lrow-n:Lrow+n, Lcol-n:Lcol+n);
      elseif Lrow-n<1
         if Lcol-n<1
            LN=LM3(1:Lrow+n,1:Lcol+n);
            rLN=resM3(1:Lrow+n,1:Lcol+n);
         elseif Lcol+n>L
            LN=LM3(1:Lrow+n, Lcol-n:L);
            rLN=resM3(1:Lrow+n, Lcol-n:L);
         else
```

```
            LN=LM3(1:Lrow+n, Lcol-n:Lcol+n);
            rLN=resM3(1:Lrow+n, Lcol-n:Lcol+n);
        end
    elseif Lrow+n>L
        if Lcol-n<1
            LN=LM3(Lrow-n:L, 1:Lcol+n);
            rLN=resM3(Lrow-n:L, 1:Lcol+n);
        elseif Lcol+n>L
            LN=LM3(Lrow-n:L, Lcol-n:L);
            rLN=resM3(Lrow-n:L, Lcol-n:L);
        else
            LN=LM3(Lrow-n:L, Lcol-n:Lcol+n);
            rLN=resM3(Lrow-n:L, Lcol-n:Lcol+n);
        elseif
    elseif Lcol-n<1
        LN=LM3(Lrow-n:Lrow+n, 1:Lcol+n);
        rLN=resM3(Lrow-n:Lrow+n, 1:Lcol+n);
    elseif Lcol+n>L
        LN=LM3(Lrow-n:Lrow+n, Lcol-n:L);
        rLN=resM3(Lrow-n:Lrow+n, Lcol-n:L);
    end

    % COUNT EACH STRAIN WITHIN THE NEIGHBORHOOD
    [m,n]=size(LN); nn=m*n; C=find(LN==R); fr=length(C)/nn;

    % REFER TO TRANSITION MATRIX & UPDATE FOCAL POINT
    rd3=rand;
    if LM3(Lrow,Lcol)==R
        if rd3 < resM3(Lrow,Lcol)
            LM3(Lrow,Lcol)=E;
            resM3(Lrow,Lcol)=0;
        else
            rd4=rand;
            if rd4 < prm
                drinc=0.002*rand-0.001;
                a=resM3(Lrow,Lcol)+drinc;
                if a >= dr_max
                    resM3(Lrow,Lcol)=dr_max;
                elseif a <= dr_min
                    resM3(Lrow,Lcol)=dr_min;
                else
                    resM3(Lrow,Lcol)=a;
                end
            end
        end
    else
        if rd3 < fr
            LM3(Lrow,Lcol)=R;
            I=find(rLN~=0); resM3(Lrow,Lcol)=mean(mean(rLN(I)));
        end
    end
  end
end
```

```
end
```

```
function [avgR4] = evolve_scen4(L,R,E,dr_min,dr_max,prm,LM4,resM4,epoch,its)

   for m=1:epoch

      % CALCULATE AVG RESISTANCE
      I=find(resM4~=0); avgR4(1,m)=mean(mean(resM4(I)));

      for i=1:its

         % RANDOMLY SELECT FOCAL POINT
         Lb=linspace(0,1,L); rd1=rand; rd2=rand;
         Lb1=find(Lb<=rd1); Lb2=find(Lb<=rd2);
         Lrow=Lb1(end); Lcol=Lb2(end);
         LM4(Lrow,Lcol);

         % GET GLOBAL COUNTS
         nn=L*L;  C=find(LM4==R); fr=length(C)/nn;

         % REFER TO TRANSITION MATRIX & UPDATE FOCAL POINT
         rd3=rand;
         if LM4(Lrow,Lcol)==R
            if rd3 < resM4(Lrow,Lcol)
               LM4(Lrow,Lcol)=E;
               resM4(Lrow,Lcol)=0;
            else
               rd4=rand;
               if rd4 < prm
                  drinc=0.002*rand-0.001;
                  a=resM4(Lrow,Lcol)+drinc;
                  if a >= dr_max
                     resM4(Lrow,Lcol)=dr_max;
                  elseif a <= dr_min
                     resM4(Lrow,Lcol)=dr_min;
                  else
                     resM4(Lrow,Lcol)=a;
                  end
               end
            end
         else
            if rd3 < fr
               LM4(Lrow,Lcol)=R;
               I=find(resM4~=0); resM4(Lrow,Lcol)=mean(mean(resM4(I)));
            end
         end
      end
   end
end
```