Evolutionary Games
Dynamics of Cooperation in Social Dilemmas

C. Cannistra, M. Zecevic, J. George, M. Antonelli

December 12, 2016

Abstract

In 2006, Hauert and his colleagues [4] had conducted a study in which they considered an ecological feedback to evolutionary game dynamics. This feedback had enabled stable coexistence of cooperators and defectors in public goods games. We had reproduced their results, and made further extensions to their model that brought new results and knowledge. We studied defector punishment, benefit discounting, concrete spatial dependence, and creation of limit cycles.

1 Introduction

There are numerous examples in nature where cooperative individuals provide benefits to other individuals at some cost to themselves. Nevertheless, cooperative behavior is susceptible to abuse by defectors that gladly accept support but avoid the cost of helping others. For instance, sticklebacks approach larger predator fish by first using two scout sticklebacks to inspect the predator's identity and motivational state [1]. The two fish would move rapidly in front of the predator, in a jerky fashion, in order to provoke it to move towards them. They do this to benefit the flock even though there is a risk of being preyed upon. Similarly, vampire bats sometimes share acquired blood with non-kin bats in order to help them survive [2].

The phenomena of cooperation had been investigated, generally, by means of public good games for groups of interacting individuals [3]. Traditional approaches to studying this phenomena assume constant population sizes and thus ignore the ecology of the interacting individuals. However, Hauert et al. [4], have incorporated ecology dynamics into evolutionary games by assuming that the evolutionary game is played in populations of varying densities.

2 Starting Model

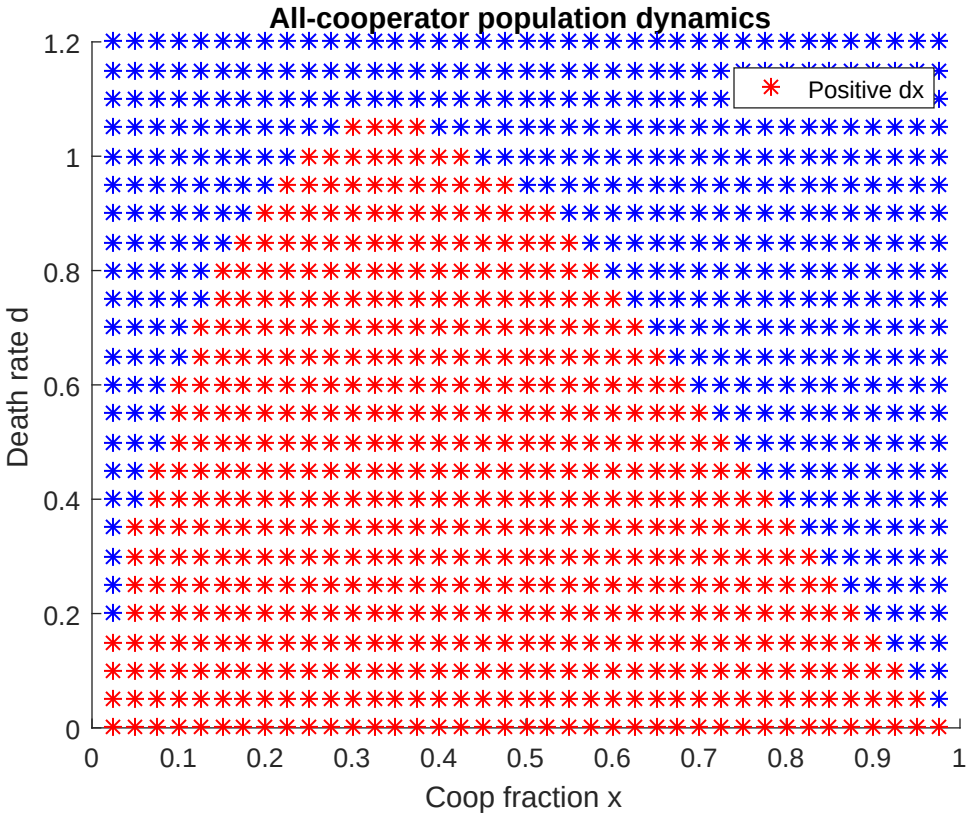
Hauert's [4] evolutionary game was modeled with a 3-D system of non-linear ODEs:

$$\dot{x} = x(zf_C - d) \tag{1}$$
$$\dot{y} = y(zf_D - d) \tag{2}$$
$$\dot{z} = -\dot{x} - \dot{y} \tag{3}$$

The model contains three state variables x, y, and z, and three parameters, d, r, and N. Variable x represents a fraction of cooperators in the population, y represents a fraction of defectors, and z represents empty space, such that $x + y + z = 1$. Parameter d represents death rate. r is the benefit-to-cost ratio to the cooperators, and N is the size of the population. The growth of cooperators or defectors is directly proportional to the available empty space z and their fitness (f_C and f_D). The model also assumes that both cooperators and defectors die at a constant rate d. Additionally, the variable z represents the connection, between game dynamics and population dynamics. The main idea here is the following: if the defection is favored, payoffs to the whole population are decaying and the overall population decreases. However, as the population density decreases, cooperation is favored, and the population rebounds. Nevertheless, the population can grow only until it runs out of space; at that point, z becomes zero, the growth terms vanish, and the population starts decaying again. This back-and-forth interaction of game dynamics and population dynamics enables both groups to coexist and reach stable equilibria in an evolutionary game.

The fitness of both groups is modeled by the following two equations:

Figure 1: The dynamics of all-cooperator populations depend on the continuous death rate. When $d = 0$, there are two stable fixed points: $x = 0$ (extinction) and $x = 1$ (full occupation). At intermediate values of d, there is a nontrivial stable fixed point and a lower unstable fixed point, along with the trivial case $x = 0$. When d is large, the only fixed point is $x = 0$. For all cases, $N = 5$ and $r = 3$.



$$f_D = r\frac{x}{1-z}(1 - \frac{1-z^N}{N(1-z)}) \tag{4}$$

$$f_C = f_D - 1 - (r-1)z^{N-1} + \frac{r}{N}\frac{1-z^N}{1-z} \tag{5}$$

These equations were constructed by computing the probability that an individual, in a randomly formed group (that is of size less or equal to N), finds itself with a certain number of cooperators, and taking the weighted average over all possible group sizes.

3 Reproducing Original Paper's Result

We have successfully reproduced Figures 1 and 2 from the original paper[4].

4 Novel Results

4.1 Benefit Discounting – w Manipulation

Hauert et al. assumes that the value of available benefit increases linearly with the number of cooperators[4]. In reality, this is not always true. It is more common for the overall value of a nutrient to plateau at large concentrations, due to saturation of cellular transport proteins and metabolic pathways. To account for nonlinear benefit scaling, we introduce the variable w in the equation for payoff, where each respective cooperator's contribution is valued at w times the last contribution.

$$P = \frac{rk}{N}(1 + w + w^2 + \cdots + w^{k-1})$$

where k is the number of cooperators and N is the total number of individuals. When $w = 1$, benefit scales linearly as in the original model. When $w < 1$, benefit value is discounted as it accumulates,

Figure 2: Heterogenous population dynamics in four parameter sets. Top row: In populations with a right benefit-to-cost ratio r, there exists a stable fixed point not along the $x + y = 0$ boundary. This point can either represent cohabitation of cooperators and defectors or complete defector extinction. Bottom row: In populations with low r, intermediate fixed points become unstable, making extinction the only attractive equilibrium point in a heterogenous population. Left column: High death rate gives rise to intermediate fixed points with relatively low population density and cooperator fraction. Right column: Low death rate makes defector extinction the more attractive equilibrium point. (a) $r = 3.0$, $d = 0.5$ (b) $r = 5.0$, $d = 1.6$ (c) $r = 2.7$, $d = 0.5$ (d) $r = 2.1$, $d = 0.5$. $N = 8$ for all cases.

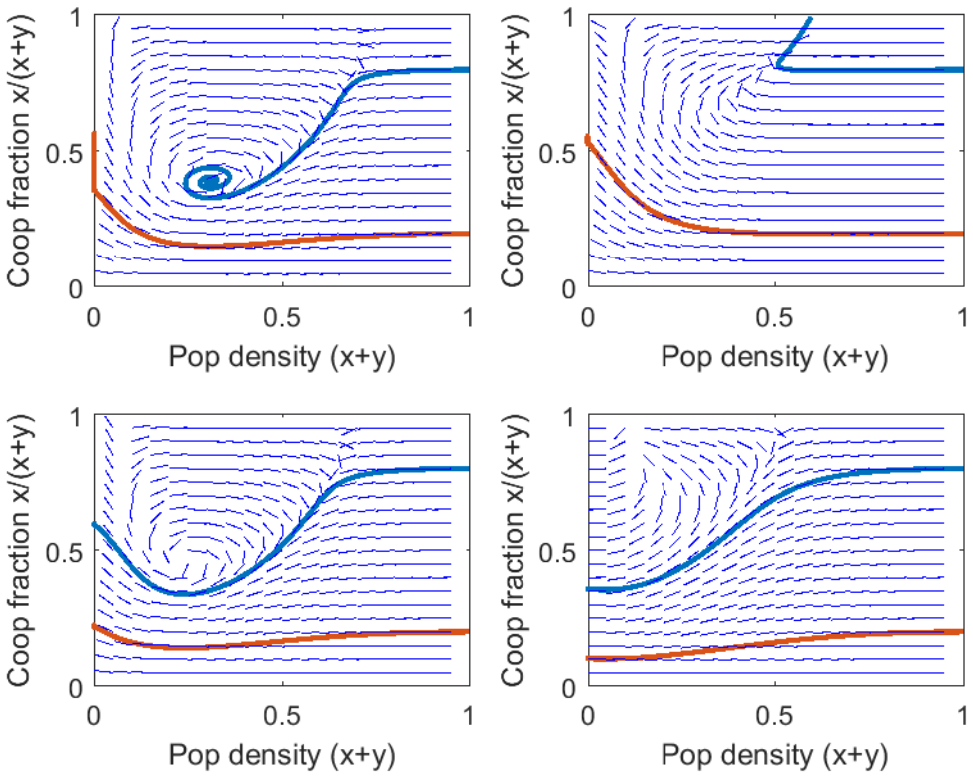
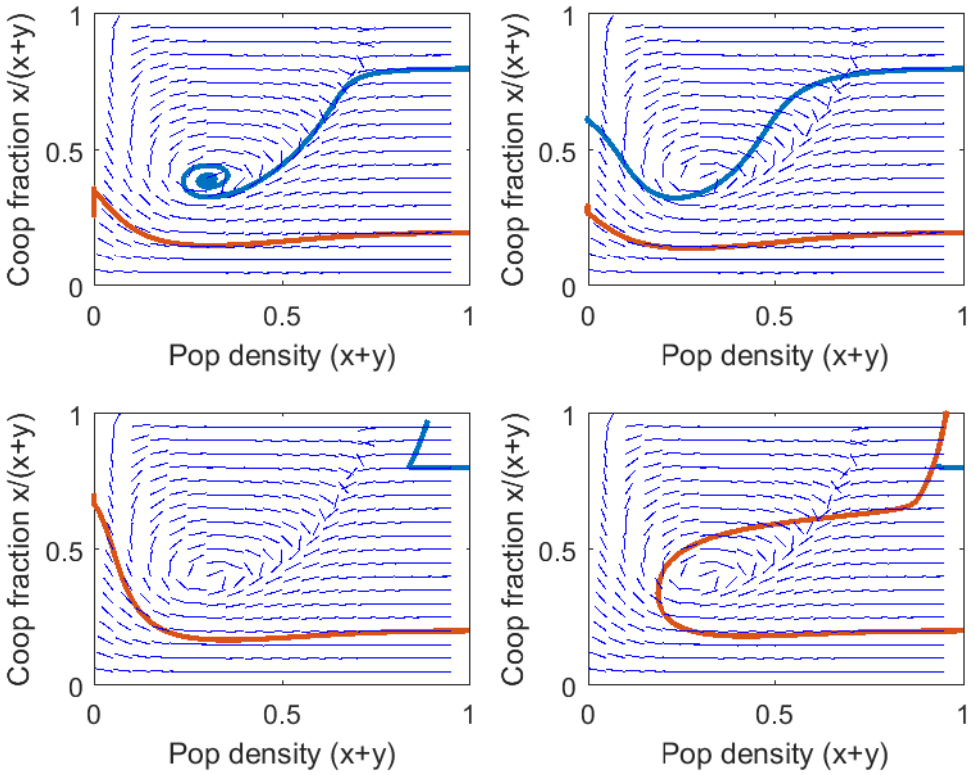


Figure 3: Dynamics of a heterogenous population with varying values of w. Low w makes extinction more likely, and inevitable in the case of this system. High w makes survival more likely, with two stable fixed points instead of one. Very high w makes the intermediate fixed point vanish, making defector extinction inevitable.



which makes the social dilemma more strict. In these cases, the intermediate fixed point vanishes, and population extinction becomes inevitable. When $w > 1$, conditions are more favorable for cooperators, and a wider range of initial conditions are able to survive while a stable fixed point for defector extinction emerges.

4.2 Limit Cycle

The original authors believed that it would be interesting to extend the model so that it can generate limit cycles. Below paragraphs will explain how we addressed this extension of the model.

4.2.1 The extended model

In order to generate limit cycles we have added the nonlinear migration term (mxy) to the original model:

$$\dot{x} = x(zf_C - d - mxy) \tag{6}$$
$$\dot{y} = y(zf_D - d + mxy) \tag{7}$$

where parameter m represents the migration rate. The main idea here is the following: For negative values of m, the migration term will negatively impact the growth of defectors, and positively impact the growth of cooperators. Moreover, while the population densities of both groups are small, the migration terms are negligible. However, as the population densities of either of the groups increase, the migration term becomes relevant and increases migration into the cooperators group. However, as the cooperators grow, the death term grows, available space z shrinks and populations start to decay - which makes the migration term insignificant again. Thus, this model favors cooperation: when large density of defector group is reached, defectors start migrating to cooperators to preserve the species; also, large number of cooperators tends to stimulate even more cooperation.

4.2.2 Parametrization of the model

To search the solution space for limit cycles we have run large number of simulations using random sets of parameters. For this purpose we utilized Latin Hypercube Sampling (LHS) method (see code in section A.2.1). This method first partitions the multidimensional parameter space into equal number of partitions, and than randomly samples from each partition. This is performed in such a way so that each parameter's range is fairly represented; in other words, sampling is stratified. The parameter ranges we used are presented below:

$$\begin{aligned} x_0 &= 0.1 \\ y_0 &= 0.1 \\ N &= 8 \\ r &\in [0, 6] \\ d &\in [0, 1] \\ m &\in [-1, 1] \end{aligned}$$

After the random sets of parameters have been created, each set has been simulated using RK4 ODE solver (see code in section A.2.2), and the solution was examined for the existence of limit cycles. Detection of limit cycles was performed in automated fashion using the following algorithm:

```
Tol1 = 1e-3
Tol2 = 0.01
for each parameter set do:
    solve the ODE system using RK4
    sort the solution vector for cooperators C in descending order and store it as X
    for j = 2 to 20:
        if |X(1) - X(j)| < Tol1 and varience(C(2000:end)) > Tol2
            then LimitCycle = True
        else
            LimitCycle = False
            break
        end
    if LimitCycle
        then print the current parameter set
end
```

4.2.3 Results

After the parameters that generate limit cycle have been discovered, the aforesaid were used to simulate the model (see code in section A.2.3) and produce the phase plot depicted in Figure 4.

Note that all of the trajectories with the initial conditions on the inside, or on the right side of the cycle, eventually converge to the limit cycle. However, the trajectories starting on the left of the limit cycle terminate at the origin. This is because the limit cycle is pushed against the separatrix that divides the phase space into two different basins of attraction.

4.3 Defector Punishment

Cooperation is a behavior that favors the society. In our scenario, we notice that, the population becomes extinct if the cooperators die. As such, society tries to develop strategies for cooperators to survive in the form of rules, regulations and punishment. A major inspiration for this section was "The nature of human altruism" [5] by Fehr and Fishbacher where they simulated multi-person prisoners' dilemma. They showed that cooperation survives when not only when defectors are punished but cooperators that don't punish defectors are punished as well.

Naturally, since our model is different, we won't be able to go to those length without significant modification. But we can start out with incorporating the punishment as an increased death rate for the defectors.

$$\dot{x} = x(zf_C - d) \tag{8}$$

$$\dot{y} = y(zf_D - d - p) \tag{9}$$

Here, the defectors have a probability p of getting caught and if caught, they are punished by execution.

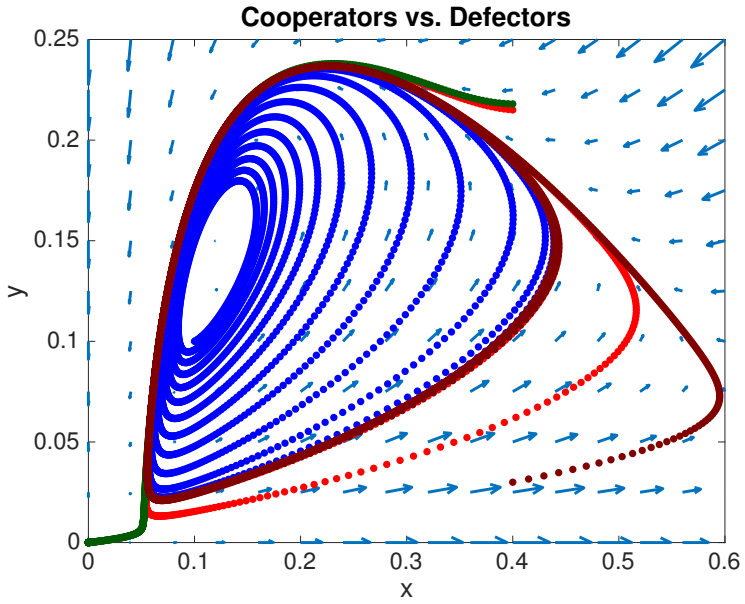


Figure 4: Phase plot with stable limit cycle: $N = 8, r = 2.7, d = 0.5, m = -0.35$

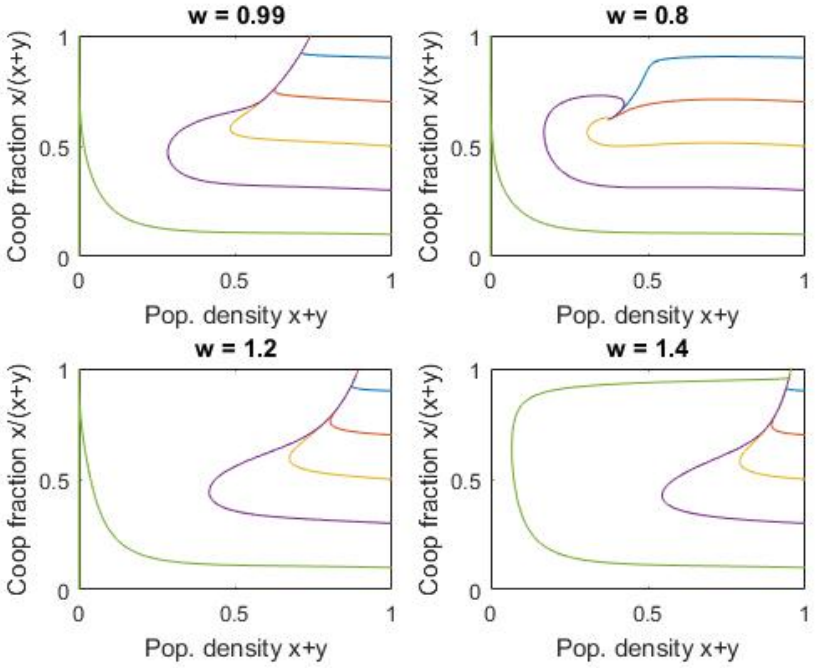


Figure 5: Probability of getting caught is 0.2

We see that

- For w =0.8, the population doesn't go extinct and has an equilibrium of coexistence.
- For w=0.99, the stable focus disappears and for a sufficiently large size of initial cooperators, the end result is an equilibrium with no defectors.
- It is the same for higher values of w.
- For higher values of w, even initial conditions which previously went extinct now survive.

Do things change with a higher value of punishment?

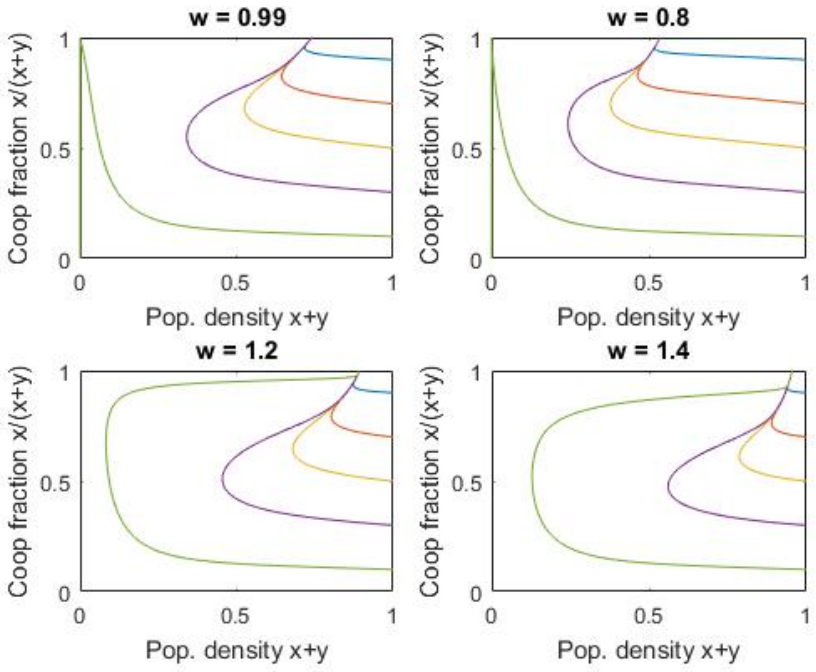


Figure 6: Probability of getting caught is 0.4

We see that

- For w =0.8, the equilibrium of coexistence disappears and the defectors die off.
- For w=1.2, the initial condition which previously became extinct now survives.

So far, during the evolution of the population, the defectors get caught with a fixed probability. We now make it random and see the differences. Since this is a more accurate representation of society, we take the probability of capture to have a mean of 0.01 and a standard deviation of 0.08.

```
%Gaussian punishment
rng(randn);
p  = 0.01 + 0.08*randn
```

The results are shown in 7. The lines look jagged like a random walk. The focus no longer looks like a perfect spiral but more like a group of intertwined threads because of the stochasticity.

However, the most surprising result is that there is a new equilibrium for a single initial condition for w=0.99. This doesn't exist with stochasticity and is reproducible with different seeds. This new equilibrium has more cooperators than defectors than the one it would have ended at. This tempts us to make a guess that stochasticity might favor the cooperators.

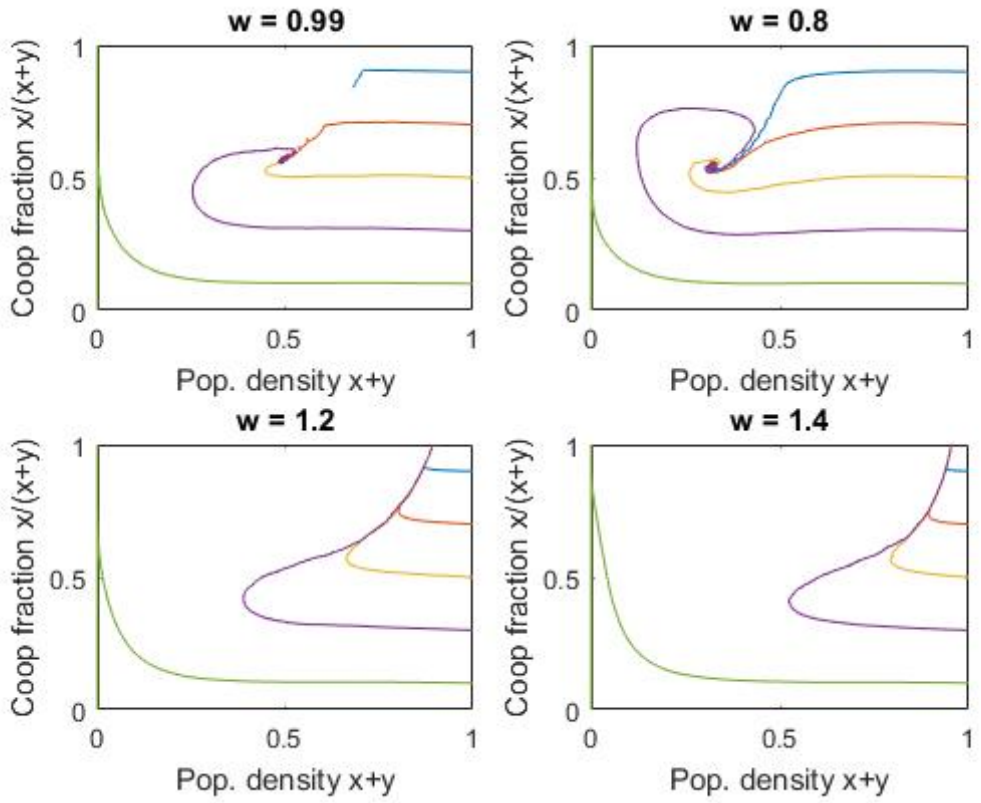


Figure 7: Random punishment

4.4 Concrete Spatial Dependence

The differential equations modeled above allow for quick analysis of many factors, but do not capture one possible source of relief for cooperators – persistence of local neighborhoods. The above model assumes a well-mixed population, but that might not be accurate if cooperators are able to create a region with few or no defectors. In particular, groups of cooperators close to each other might be more resilient to the parasitical defectors, since they could support each other at need.

4.4.1 Concrete Spatial Model

To test this hypothesis, we created a new discrete running along similar lines to the ODE's, but with concrete positions for each individual. An individual's neighborhood is persistent from one time to another. We chose to experiment on a two-dimensional grid, where each grid point is either a cooperator, a defector or empty. In addition, we chose a stencil which defines a neighborhood for each point. An individual can only benefit from a shared resource if it is produced within the individual's neighborhood. Figure 8 shows several possible stencils.

At each time step, every individual's fitness, f, is calculated by summing the local resources and subtracting any costs (cooperators must pay c to create the resource). Any individual whose fitness dropped below a certain level, m (mortality), died immediately. Individuals with fitness above another level, b (breeding), were given a chance to breed to an empty spot within their neighborhood. Finally, individuals with fitness between these two constants $m < f < b$ simply survived to the next time step with no chance to reproduce.

4.4.2 Concrete Spatial Results

A few results can be spotted before you even run the model. As with the ODE model, if r is too low, there's not enough food and everyone dies. Similarly, if r is very high, then cooperators can withstand any amount of local defection, and everyone lives. More interesting results happen in between.

For moderate r, we see several emergent behaviors. Although defectors can still overpower and kill lone cooperators, groups of cooperators are more robust. Depending on how you select the neighborhood as

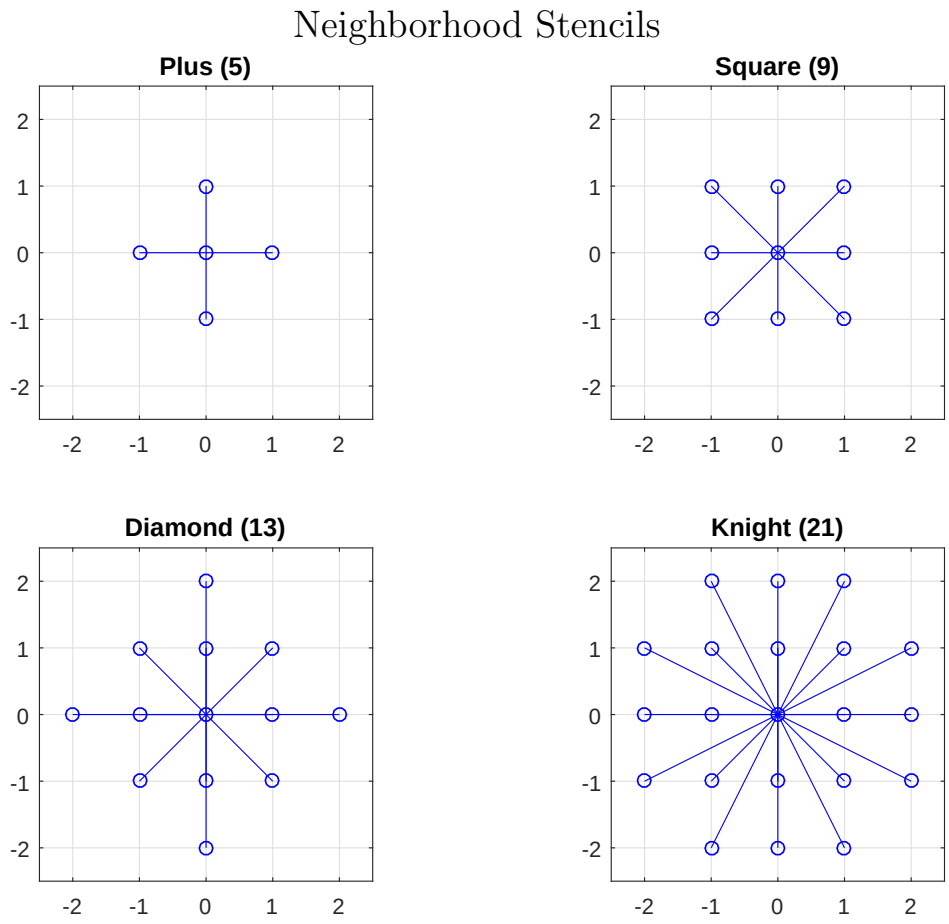


Figure 8: A selection of possible stencils for the concrete spatial modeling. Larger stencils favor defectors, while smaller stencils favor cooperators. The square stencil proved easiest to work with experimentally.

well as the r, m and b terms, different sized clusters of cooperators are required to become stable. These locally stable groups act as factories, allowing the population to spread into nearby empty regions.

The concrete grid exhibits the same types of dynamics seen in the ODE version, though generally behaving more favorably for cooperators. Figure 9 shows a phase diagram exhibiting the same sort of clines, fixed points and rotation as the ODEs.

5 Conclusion

Our experiments confirm the original paper's results. In addition, we find several other mechanisms which encourage cooperation, despite the increased cost to the individual. Punishment, benefit synergy and neighborhood-clearing all work to improve the fitness of cooperators beyond the naïve prisoner's dilemma expectation.

References

[1] Milinski, M. (1987). Tit for tat in sticklebacks and the evolution of cooperation. nature, 325(6103), 433-435.

[2] Wilkinson, G. S. (1984). Reciprocal food sharing in the vampire bat. Nature, 308(5955), 181-184.

[3] Roth, A. E. (1995). pBargaining Experimentsqin JH Kagel and AE Roth (Eds.), Handbook of Experimental Economics.

[4] Hauert, C., Holmes, M., & Doebeli, M. (2006). Evolutionary games and population dynamics: maintenance of cooperation in public goods games. Proceedings of the Royal Society of London B: Biological Sciences, 273(1600), 2565-2571.

[5] Fehr, Ernst, and Urs Fischbacher. "The nature of human altruism." Nature 425.6960 (2003): 785-791.

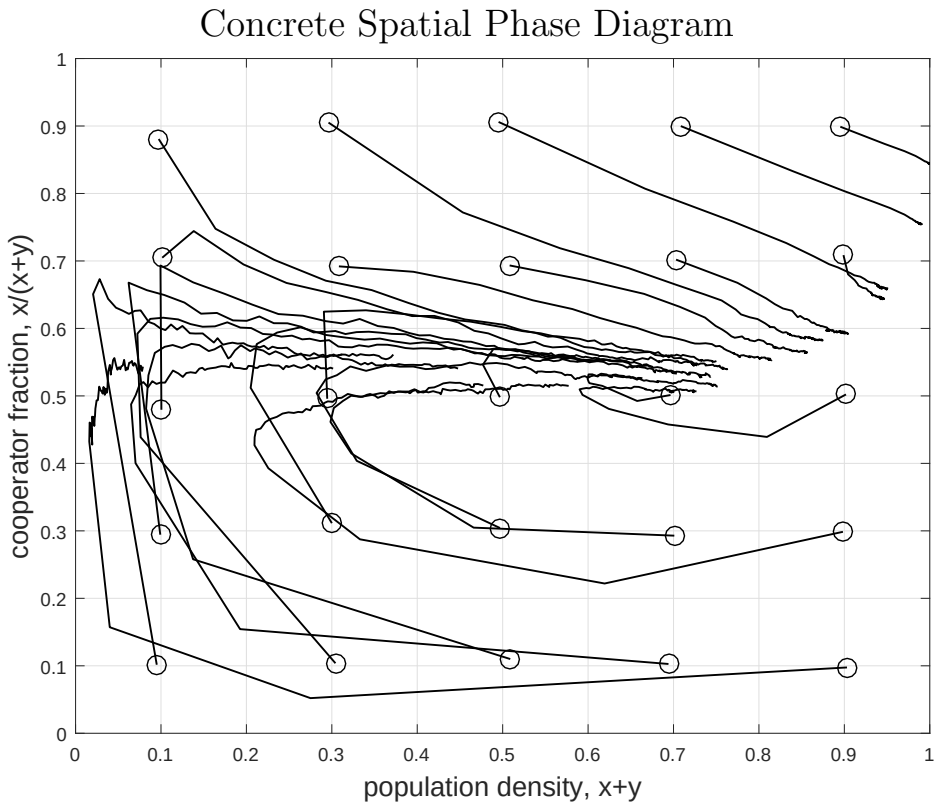


Figure 9: Behavior of the concrete spatial model with $r = 2.7c$, $m = 0.3c$, $b = c$ and a square stencil.

A Code

A.1 Benefit Discounting

```
function dy = coopmodel(t,y,params)
    N = params(1);
    r = params(2);
    d = params(3);
    w = params(4);

    z = 1 - y(1) - y(2);
    f = y(1)/(y(1) + y(2));

    fc = 0;
    fd = 0;
    for S=2:N
        Pc_sum = 0;
        Pd_sum = 0;
        for m=0:(S-1)
           Pc = r/S * (1 - w^(m+1))/(1 - w) - 1;
           Pd = r/S * (1 - w^m)/(1 - w);
           Pc_sum = Pc_sum + Pc * nchoosek(S-1,m) * f^m * (1-f)^(S-1-m);
           Pd_sum = Pd_sum + Pd * nchoosek(S-1,m) * f^m * (1-f)^(S-1-m);
        end
        fc = fc + Pc_sum * nchoosek(N-1,S-1) * z^(N-S) * (1-z)^(S-1);
        fd = fd + Pd_sum * nchoosek(N-1,S-1) * z^(N-S) * (1-z)^(S-1);
    end

    dy = zeros(2,1);
    dy(1) = y(1)*(fc*z - d);
    dy(2) = y(2)*(fd*z - d);
end

function coopsimulate(N,r,d,w)

    params = [N r d w];
    tspan = linspace(0,100,1001);

    [t,y] = ode45(@(t,y) coopmodel(t,y,params), tspan, [0.9; 0.1]);
    [t,y1] = ode45(@(t,y) coopmodel(t,y,params), tspan, [0.7; 0.3]);
    [t,y2] = ode45(@(t,y) coopmodel(t,y,params), tspan, [0.5; 0.5]);
    [t,y3] = ode45(@(t,y) coopmodel(t,y,params), tspan, [0.3; 0.7]);
    [t,y4] = ode45(@(t,y) coopmodel(t,y,params), tspan, [0.1; 0.9]);

    plot(y(:,1) + y(:,2), y(:,1)./(y(:,1) + y(:,2)));
    hold on;
    plot(y1(:,1) + y1(:,2), y1(:,1)./(y1(:,1) + y1(:,2)));
    plot(y2(:,1) + y2(:,2), y2(:,1)./(y2(:,1) + y2(:,2)));
    plot(y3(:,1) + y3(:,2), y3(:,1)./(y3(:,1) + y3(:,2)));
    plot(y4(:,1) + y4(:,2), y4(:,1)./(y4(:,1) + y4(:,2)));
end

subplot(2,2,1)
coopsimulate(8,3.0,0.5,0.99);
xlim([0 1]);
ylim([0 1]);
xlabel('Pop. density x+y');
ylabel('Coop fraction x/(x+y)');
title('w = 0.99');
subplot(2,2,2)
coopsimulate(8,3.0,0.5,0.8);
xlim([0 1]);
ylim([0 1]);
xlabel('Pop. density x+y');
```

```
ylabel('Coop fraction x/(x+y)');
title('w = 0.8');
subplot(2,2,3)
coopsimulate(8,3.0,0.5,1.2);
xlim([0 1]);
ylim([0 1]);
xlabel('Pop. density x+y');
ylabel('Coop fraction x/(x+y)');
title('w = 1.2');
subplot(2,2,4)
coopsimulate(8,3.0,0.5,1.4);
xlim([0 1]);
ylim([0 1]);
xlabel('Pop. density x+y');
ylabel('Coop fraction x/(x+y)');
title('w = 1.4');
```

A.2 Limit Cycle

A.2.1 LHS.m

```
function [ samples ] = LHS( n,lb,ub )
%The function LHS applies Latin Hypercube Sampling to a range of parameters
%in order to create stratified normaly distributed parameter space
%
% INPUT:
% n = number of samples
% lb = vector of lover bound values for each parameter
% ub = vector of upper bound values for each parameter
%
% OUTPUT:
% x = n x p matrix of parameter samples (where p is a number of
% parameters)
%
%   Mladen Zecevic, 11-24-2016

p = numel(lb);                  % number of parameters
xn = lhsdesign(n,p,'criterion','correlation');
samples = bsxfun(@plus,lb,bsxfun(@times,xn,(ub-lb))); % apply to the ranges

end
```

A.2.2 Parametrize.m

```
%Parametrize calls function LHS to crete a stratified space of parameter values with n
% numer of sets, and then simulates the model with each set of parameters.
%
% Mladen Zecevic, 11-24-2016
clear all; close all; clc; format compact;
Y0 = [0.1 0.1]; % fixed initial conditions
N = 8; % model parameter (fixed here)

n = 1000; % number of sets of parameter samples

% vector of lower bounds for 3 parameters (r,d,a)
lb = [0 0 -1];
% vector of upper bounds for 3 parameters (r,d,a)
ub = [6 1 1];

options = odeset('RelTol', 1e-5, 'AbsTol',1e-5);
tspan = 0:0.25:1000;

P = []; % dummy matrix
for k = 1:20 % 20 searches
    H = LHS(n,lb,ub); % create parameter space
```

```
    for j = 1:n
        p = [H(j,1), N, H(j,2), H(j,3)]; % parametrs vector
        [t,Y] = ode45(@(t,y) coopODE(t,y,p), tspan, Y0, options);

        % sorted
        xSorted = sort(Y(:,1),'descend');
        ySorted = sort(Y(:,2),'descend');

        % check if periodic
        TOL = 1e-3;
        for m = 2:20
            if abs(xSorted(1)-xSorted(m))<TOL && (var(Y(2000:end,1)) > 0.01)
                limCycle = 1; % there is limit cycle
            else
                limCycle = 0; % there is NO limit cycle
                break;
            end
        end

        % save if physical and limit cycle
        if (min(min(Y))>=0) && (max(max(Y))<=1) && limCycle
            P = [P; p] % save I.Cs and the parameters (r, N, d, a)
        end
    end % for loop
    if size(P,1) > 0 % if limit cycles found in this LHS run
        break;
    end
end
```

A.2.3 coopRK4.m

```
%coopRK4 plots the direction field and trajectories for 4 different I.Cs
%
%   Mladen Zecevic, 11-24-2016
clear all, close all, clc;
t = 0;
tspan = 0:0.1:1000;

N = 8; % theoretical population size

r = 2.7; % return on investment
d = 0.5; % death rate
m = -0.35; % migration rate
Y0 = [0.1 0.1]; % initial conditions
p = [r N d m]; % parameters vector

options = odeset('RelTol', 1e-8, 'AbsTol',1e-8);

%% first
[t,Y] = ode45(@(t,y) coopODE(t,y,p), tspan, Y0, options);

%% second
Y0 = [0.4 0.215];
[t,Y1] = ode45(@(t,y) coopODE(t,y,p), tspan, Y0, options);

%% third
Y0 = [0.40 0.218];
[t,Y3] = ode45(@(t,y) coopODE(t,y,p), tspan, Y0, options);

%% fourth
Y0 = [0.40 0.03];
[t,Y4] = ode45(@(t,y) coopODE(t,y,p), tspan, Y0, options);

%% Phase plots
```

```
% here we plot the phase plane with 4 different I.Cs

% the grid
xMin = 0;
xMax = 0.6;
yMin = 0;
yMax = 0.25;
xSplicing = 0.04;
ySplicing = 0.025;
xList = xMin:xSplicing:xMax;
yList = yMin:ySplicing:yMax;
[xMatrix,yMatrix] = meshgrid(xList,yList);
dxdt_matrix = zeros(size(xMatrix));
dydt_matrix = zeros(size(yMatrix));

% populate the direction field matrices
for j=1:length(yList)
    for k=1:length(xList)
        % return a vector F of velocities in x1 and x2 directions
        F = coopODE(t,[xMatrix(j,k); yMatrix(j,k)],p);

        dxdt_matrix(j,k) = F(1);
        dydt_matrix(j,k) = F(2);
    end
end

% Plot the arrows
figure
set(gca,'FontSize',16)
quiver(xMatrix,yMatrix,dxdt_matrix,dydt_matrix,0.8,'LineWidth',2)
xlabel('x','FontSize',20),ylabel('y','FontSize',20)
xlim([xMin xMax]),ylim([yMin yMax])
hold on

% plot the trajectories
plot(Y(:,1),Y(:,2),'b.','MarkerSize',15)
plot(Y1(:,1),Y1(:,2),'r.','MarkerSize',15)
plot(Y3(:,1),Y3(:,2),'.','color',[0 0.351 0],'MarkerSize',15)
plot(Y4(:,1),Y4(:,2),'.','color',[0.502 0 0],'MarkerSize',15)
title('Cooperators vs. Defectors'), set(gca,'FontSize',16);
```

A.3 Defector Punishment

A.3.1 Constant punishment

```
function dy = constp(t,y,params)
N = params(1);
r = params(2);
d = params(3);
w = params(4);

z = 1 - y(1) - y(2);
f = y(1)/(y(1) + y(2));

fc = 0;
fd = 0;
for S=2:N
Pc_sum = 0;
Pd_sum = 0;
for m=0:(S-1)
Pc = r/S * (1 - w^(m+1))/(1 - w) - 1;
Pd = r/S * (1 - w^m)/(1 - w);
Pc_sum = Pc_sum + Pc * nchoosek(S-1,m) * f^m * (1-f)^(S-1-m);
Pd_sum = Pd_sum + Pd * nchoosek(S-1,m) * f^m * (1-f)^(S-1-m);
end
```

```
fc = fc + Pc_sum * nchoosek(N-1,S-1) * z^(N-S) * (1-z)^(S-1);
fd = fd + Pd_sum * nchoosek(N-1,S-1) * z^(N-S) * (1-z)^(S-1);
end

p=0.2;
dy = zeros(2,1);
dy(1) = y(1)*(fc*z - d);
dy(2) = y(2)*(fd*z - d-p);
end
```

A.3.2 Gaussian punishment

```
function dy = randp(t,y,params)
N = params(1);
r = params(2);
d = params(3);
w = params(4);

z = 1 - y(1) - y(2);
f = y(1)/(y(1) + y(2));

fc = 0;
fd = 0;
for S=2:N
Pc_sum = 0;
Pd_sum = 0;
for m=0:(S-1)
Pc = r/S * (1 - w^(m+1))/(1 - w) - 1;
Pd = r/S * (1 - w^m)/(1 - w);
Pc_sum = Pc_sum + Pc * nchoosek(S-1,m) * f^m * (1-f)^(S-1-m);
Pd_sum = Pd_sum + Pd * nchoosek(S-1,m) * f^m * (1-f)^(S-1-m);
end
fc = fc + Pc_sum * nchoosek(N-1,S-1) * z^(N-S) * (1-z)^(S-1);
fd = fd + Pd_sum * nchoosek(N-1,S-1) * z^(N-S) * (1-z)^(S-1);
end

rng(randn);
p  = 0.01 + 0.08*randn;

dy = zeros(2,1);
dy(1) = y(1)*(fc*z - d);
dy(2) = y(2)*(fd*z - d-p);
end
```

A.4 Concrete Spatial Dependence

A.4.1 mja_spatial_main.m

```
%% AMATH 422/522 Project Exploration
% Miranda Antonelli
clearvars; close all; clc; format compact;
MAKE_MOVIE = true;
rng(2016);

%% Problem Parameters
nSteps = 2^10;
xMax = 96;
yMax = xMax;
c = 1;              % Cost to create public good.
r = 2.7;            % Multiplicative value (ROI) on public good.
pCoop = 0.1;        % Probability cell cooperating at start (state = 1)
pDefect = 0.05;     % Probability cell defecting (cheating) at start (2)
survive = .85;      % Minimum fitness to live
breed = 1.5;        % Minimum fitness to attempt breeding
```

```
neighborhood = [0 1 0 ; 1 1 1 ; 0 1 0];
%neighborhood = ones([3 3]);
%neighborhood = [0 1 1 1 0 ; 1 1 1 1 1 ; 1 1 1 1 1; 1 1 1 1 1; 0 1 1 1 0];

% We will work with an expanded grid to make certain neighbor calculations
% easier.
xRadius = (size(neighborhood, 2)-1)/2;
yRadius = (size(neighborhood, 1)-1)/2;

rr = 1:(yMax+2*yRadius);    % The indices of all the expanded y points (rows)
cc = 1:(xMax+2*xRadius);    % The indices of all the expanded x points (cols)
rri = yRadius + (1:yMax);   % The indices of the interior y points
cci = xRadius + (1:xMax);   % The indices of the interior x points

params = [yMax xMax c r xRadius yRadius survive breed];

% The needed grids
population = zeros(length(rr), length(cc));    % 0 Empty, 1 Coop, 2 Defect
localPop = 0 * population;  % Local population (self + neighbors)
resources = 0 * population; % Locations of resources divided by local pop
fitness = 0 * population;   % Total fitness from all resources

%% Initial Conditions
randStart = rand(yMax, xMax);
pop0 = randStart < (pCoop+pDefect);
pop0 = pop0 + (randStart < pDefect);
population(rri,cci) = pop0;

hFigStart = figure('name', 'Starting Population', 'position', [600 600 400 400]);
imagesc(population); axis equal; axis tight;
%tightfig;
cMapPop = [1 1 1; 0 0 1 ; 1 0 0 ];
colormap(cMapPop);

%% Time Loop
allPops = zeros(yMax, xMax, nSteps);
allPops(:,:,1) = population(rri,cci);

for tIdx = 2:nSteps
    fprintf('Beginning time step: %3i with ', tIdx)
    nCoop = sum(sum(population == 1));
    fprintf(' %3i cooperators and ', nCoop)
    nDefect = sum(sum(population == 2));
    fprintf(' %3i defectors.\n', nDefect);

    localPop = compute_local_pop(population, neighborhood);
    resources = compute_resources(population, localPop, neighborhood, params);
    fitness = compute_fitness(population, resources, neighborhood, params);

    % Cull unfit members
    population = population .* (fitness >= survive);

    % Attempt to breed
    postBreeding = attempt_breeding(population, fitness, neighborhood, params);

    population = postBreeding;
    allPops(:,:,tIdx) = population(rri,cci);
end

hFigComputed = figure('name', 'Neighbors', 'position', [500 100 1000 1000]);
sRows = 2;
sCols = 2;
subplot(sRows, sCols, 1);
```

```
imagesc(allPops(:,:,1)); axis equal; axis tight;
title('Start Population')

subplot(sRows, sCols, 2);
imagesc(resources(rri,cci)); axis equal; axis tight;
title('Resources')

subplot(sRows, sCols,3)
imagesc(fitness(rri,cci)); axis equal; axis tight;
title('Fitness')

subplot(sRows, sCols,4)
imagesc(allPops(:,:,end)); axis equal; axis tight;
title('Pop Post-Breeding')

%% Movie
if MAKE_MOVIE
    vidObj = VideoWriter('evo_game.avi');
    set(vidObj, 'Quality', 100)
    open(vidObj);

    nCoop = zeros(1,nSteps);
    nDefect = zeros(1,nSteps);
    nTotalPop = zeros(1,nSteps);
    popRatio = zeros(1,nSteps);
    for tIdx = 1:nSteps
        nCoop(tIdx) = sum(sum(allPops(:,:,tIdx) == 1));
        nDefect(tIdx) = sum(sum(allPops(:,:,tIdx) == 2));
        nTotalPop(tIdx) = (nCoop(tIdx) + nDefect(tIdx))/(xMax*yMax);
        if nDefect(tIdx) == 0
            if nCoop(tIdx) == 0
                popRatio(tIdx) = 0;
            else
                popRatio(tIdx) = 1;
            end
        else
            popRatio(tIdx) = nCoop(tIdx) / (nCoop(tIdx) + nDefect(tIdx));
        end
    end


    hFigMovie = figure('name', 'movie', 'position', [100 100 1600 800]);
    % Plot of populations and ratio over time
    subplot(1,2,2);
    hPlotDot = plot(nTotalPop(1), popRatio(1), 'ro', 'markersize', 12); hold on;
    hPlotCurve = plot(nTotalPop(1), popRatio(1), 'k-');
    axis([0 1 0 1]);
    xlabel('population density, x+y', 'fontsize', 14);
    ylabel('cooperator fraction, x/(x+y)', 'fontsize', 14);
    grid on;

    for tIdx = 1:nSteps
        % Update population map
        subplot(1,2,1);
        imagesc(allPops(:,:,tIdx));
        title(sprintf('t = %i of %i', tIdx, nSteps));

        % Update behavior curve
        subplot(1,2,2);
        set(hPlotDot, 'XData', nTotalPop(tIdx));
        set(hPlotDot, 'YData', popRatio(tIdx));
        set(hPlotCurve, 'XData', nTotalPop(1:tIdx));
        set(hPlotCurve, 'YData', popRatio(1:tIdx));
```

```
        % Write frame
        currentFrame = getframe(hFigMovie);
        writeVideo(vidObj, currentFrame);
    end

    close(vidObj);
end
```

A.4.2 compute_local_pop.m

```
function result = compute_local_pop(population, neighborhood)
% We will work with an expanded grid to make certain neighbor calculations
% easier.
xRadius = (size(neighborhood, 2)-1)/2;
yRadius = (size(neighborhood, 1)-1)/2;
yMax = size(population, 1) - 2*yRadius;
xMax = size(population, 2) - 2*xRadius;

rr = 1:size(population,1);  % The indices of all the expanded y points (rows)
cc = 1:size(population,2);  % The indices of all the expanded x points (cols)
rri = yRadius + (1:yMax);   % The indices of the interior y points
cci = xRadius + (1:xMax);   % The indices of the interior x points

result = 0*population;
occupied = population ~= 0;

for row = rri
    rrNeigh = row + (-yRadius:yRadius);
    for col = cci
        ccNeigh = col + (-xRadius:xRadius);
        result(row, col) = sum(sum(neighborhood.*occupied(rrNeigh, ccNeigh)));
    end
end
```

A.4.3 compute_resources.m

```
function resources = compute_resources(population, localPop, neighborhood, params)
% We will work with an expanded grid to make certain neighbor calculations
% easier.
yMax = params(1);
xMax = params(2);
c = params(3);
r = params(4);
yRadius = params(5);
xRadius = params(6);

rr = 1:size(population,1);  % The indices of all the expanded y points (rows)
cc = 1:size(population,2);  % The indices of all the expanded x points (cols)
rri = yRadius + (1:yMax);   % The indices of the interior y points
cci = xRadius + (1:xMax);   % The indices of the interior x points

resources = 0*localPop;

for row = rri
    for col = cci
        if population(row,col) == 1     % Cooperator
            resources(row,col) = r*c/localPop(row,col);
        end
    end
end
```

A.4.4 compute_fitness.m

```
function fitness = compute_fitness(population, resources, neighborhood, params)
% We will work with an expanded grid to make certain neighbor calculations
% easier.
yMax = params(1);
xMax = params(2);
c = params(3);
r = params(4);
yRadius = params(5);
xRadius = params(6);

rr = 1:size(population,1);  % The indices of all the expanded y points (rows)
cc = 1:size(population,2);  % The indices of all the expanded x points (cols)
rri = yRadius + (1:yMax);   % The indices of the interior y points
cci = xRadius + (1:xMax);   % The indices of the interior x points

fitness = 0*population;

for row = rri
    rrNeigh = row + (-yRadius:yRadius);
    for col = cci
        ccNeigh = col + (-xRadius:xRadius);
        if population(row,col) == 2
            fitness(row,col) = sum(sum(resources(rrNeigh,ccNeigh).*neighborhood));
        elseif population(row,col) == 1
            fitness(row,col) = sum(sum(resources(rrNeigh,ccNeigh).*neighborhood)) - c;
        end
    end
end
```

A.4.5 attempt_breeding.m

```
function result = attempt_breeding(population, fitness, neighborhood, params)
% We will work with an expanded grid to make certain neighbor calculations
% easier.
yMax = params(1);
xMax = params(2);
c = params(3);
r = params(4);
yRadius = params(5);
xRadius = params(6);
survive = params(7);
breed = params(8);

rr = 1:size(population,1);  % The indices of all the expanded y points (rows)
cc = 1:size(population,2);  % The indices of all the expanded x points (cols)
rri = yRadius + (1:yMax);   % The indices of the interior y points
cci = xRadius + (1:xMax);   % The indices of the interior x points

breedSpots = neighborhood;
breedSpots(yRadius+1, xRadius+1) = 0;

postBreeding = population;

for row = rri
    for col = cci
        if fitness(row,col) >= breed
            %fprintf('  Breeding from point: (%3i, %3i) ', row, col)
            I = find(breedSpots == 1);
            idx = randi(length(I));
            [rBreed, cBreed] = ind2sub(size(neighborhood), idx);
            rBreed = row + rBreed - yRadius - 1;
```

```
            cBreed = col + cBreed - xRadius - 1;
            if population(rBreed, cBreed) == 0
                %fprintf('to point(%3i, %3i)\n', rBreed, cBreed)
                postBreeding(rBreed, cBreed) = randi(2);
            else
                %fprintf(' but spot was occupied.\n')
            end
        end
    end
end

result = 0 * population;
result(rri,cci) = postBreeding(rri,cci);    % Cull from buffer regions
```