Approaches to Modelling a Predator-Prey System in 2D Space

Jasmine Otto

June 12, 2015

Abstract

We compare two approaches to simulating predator-prey dynamics with spatial effects: as an agent-based system, and as a variant of reaction-diffusion. As a system of agents, we observe that rare predator success and slow predator respone to an increase in prey numbers both reduce the magnitude of oscillations, reducing the chances of an extinction. Apart from the consequences of discretization error (such as extinctions) and of a stochastic component in predator and prey growth rates, the agent-based model agrees with the Lotka-Volterra model of population dynamics. In our PDE-based reaction-diffusion model, oscillations initially occur over space instead of time. These spatial oscillations disappear once the space is saturated; the subsequent temporal oscillations are dampened over time, especially by fixed boundary conditions and/or self-limitation of prey, and eventually disappear.

1 Background

We will investigate predator-prey relationships in terms of population dynamics, additionally introducing spatial effects. In so doing, we seek to better understand the trophic web (generalizing the notion of 'food chain'), and thereby its parent ecology. Moreover, we will explore two distinct approaches to simulating group dynamics - by treating species in terms of individual agents, or as quantities in flux.

We begin with the ODE (ordinary differential equation) model of population dynamics alone. We let the growth rate of the prey increase with the prey population (initially, at least) and decrease with the # of prey-predator interactions. We let the growth rate of the predator population increase with said # of interactions and decrease with the predator population (i.e. competition leading to self-limitation).

Algorithm 1 Lotka-Volterra predator-prey equations, with self-limitation of prey.

$$\begin{aligned} \frac{du}{dt} &= \alpha u\left(1-u\right) - \beta uv \\ \frac{dv}{dt} &= \beta uv - \gamma v \end{aligned}$$

u, v are functions of time, t. α, β, γ are positive constants.
We require that $u(t), v(t) \in [0,1] \forall t$. Observe that $u\left(1-u\right)$ is positive if and only if $u < 1$, and so the carrying capacity of species u is 1.

When the initial population of both species is small, prey numbers tend to rise, followed by predator numbers - leading to a crash in prey numbers, followed by a crash in predator numbers, and then the cycle repeats. In the absence of diffusion, this leads to oscillatory behavior in time. Observe also that whilst the prey can persist without the predator, the predator cannot persist without the prey.

We will discuss both some results from and the technical tradeoffs of an agent-based and PDE-based approach to implementing spatial effects. PDEs (partial differential equations) generalize ODEs to multiple dimensions (e.g. 2d space in addition to time), whereas agents are defined in terms of individuals' behavior rather than the more abstract notion of a population's behavior.

In an agent-based system implemented in Repast (an agent-based modelling framework), with discrete population and stochastic predator-prey interactions, the model is prone to extinction events when 'down-swings' are severe (i.e. the amplitude of the oscillations is large).

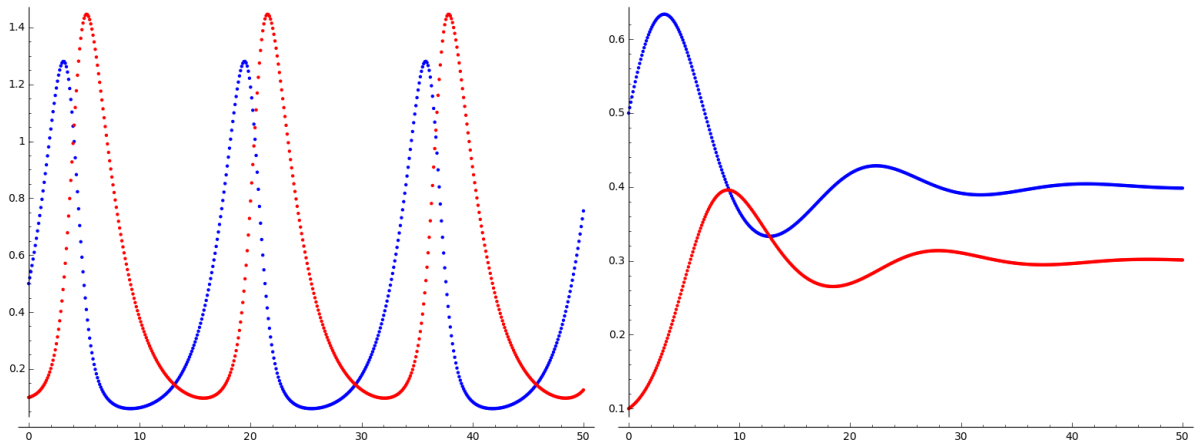

Figure 1: (a) Oscillatory steady state, in the absence of self-limitation of prey. $\alpha = .5$, $\beta = 1$, $\gamma = .4$. (b) Dampened oscillatory steady state. $\alpha = .5$, $\beta = 1$, $\gamma = .4$.

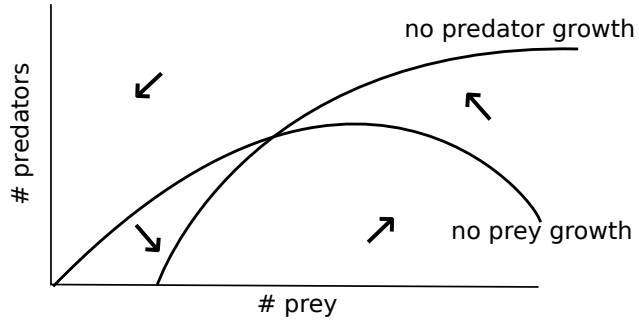

Figure 2: Limit cycle determined by nullclines (e.g. $\frac{du}{dt} = 0$) in the phase space.

In PDE-based systems with diffusion over space, we will show by numerical simulation in Jython that the oscillation occurs over space instead, until the (finite) space is saturated. The oscillations thereafter occur in time, but only if the prey-population is not self-limiting and the space lacks a fixed boundary with fixed population, as either suffices to dampen (eventually eliminating) the oscillations.

2 An Agent-Based Model

2.1 Description

Our initial approach was to model directly the behavior of individuals and analyze the resulting dynamics - an AI-like approach. An agent-based model was used, wherin each 'agent' decides where to go at each timestep (based upon its current surroundings, but also its current state), and collision with another agent counts as an interaction.

2.2 Results

Emergent behavior leads to population oscillations resembling the Lotka-Volterra equations (Figure 1), with the exception that a population may become extinct at zero population. Since predators cannot survive without prey, either both species persist, only the prey do, or neither does.

Depending on the prey population's 'average' growth rate during a given upswing (which becomes negative at some critical predator density), oscillations in their population tend to vary in magnitude. Larger oscillations seem more likely to end with an extinction, as the excess predators responding to the windfall of

Algorithm 2 An agent-based model of predator-prey dynamics.

We begin with no grass, 100 prey, and 10 predators.
Patches potentially containing grass form a 32-by-32 grid on a torus.
Prey and predators start at random locations on the torus.

Grass

- Fertility is an integer between 0 and 8, randomly generated at initialization.
- The fertility is the maximum amount of grass on the patch.
- Grass regenerates by 1 level with a 1% chance each tick.

Prey

- Dies if overstressed (stress $>$ 300) or caught by a predator. Stress ticks up slowly (+1 / tick). Travels at a fixed speed.
- Grazes when patch is not bare and not being hunted, depleting the patch and reducing own stress.
- Unless sated, moves towards as empty (of agents) a neighboring patch as it can find.
- Reproduces (asexually, producing one offspring) when stress is low, slightly raising stress (+25).

Predator

- Dies if overstressed (stress $>$ 300). Stress ticks up quickly (+10 / tick). Travels at a fixed speed.
- Tags a single prey (randomly picking from those within a 3-tile radius) when not sated, not hunting, and there is prey within said radius. Immediately begins hunting tagged prey.
- Each tick when hunting, catches the tagged prey if it is on a neighboring tile, reducing own stress. Otherwise, the hunt has a 50% chance of arbitrarily failing, recurring each tick.
  - If the hunt continues, the predator travels towards the prey's current location.
- When not hunting, moves forwards, with a 1% chance each tick of randomly changing direction.
- Reproduces (asexually, producing one offspring) when stress is low (and a short cooldown since it last reproduced is cleared), massively raising stress (+275).

prey may fail to die off (or 'depart forever') quickly enough afterwards. The magnitude of oscillations can be constrained by introducing self-limitation of prey, such as competition for grass.

The exact prey growth rate involves a stochastic component, which gives rise to minor oscillations (where the prey population dips but recovers without a collapse of the predator population) and to variable period and magnitude in the major oscillations.

Both predator satiation and hunting failure reduce the rate of prey depletion, by reducing the number of active predators and decreasing the rate of prey capture per active predator, respectively. This reduces the chances of discretization error resulting in prey extinction (which is unrecoverable for lack of migrants).

The rate at which predators respond to high prey pop, low prey pop, is also determined by predator reproduction rate; if it is too high, the predators respond too quickly to high prey availability - the subsequent boom in their numbers tends to cause prey extinction (leading immediately to predator extinction). Alternatively, reducing the benefit gained from each prey item to the point where the predator population does not grow produces a stable, non-oscillatory stable state.

2.3 Technical

Our initial simulation in non-dedicated software (Scala) encountered efficiency issues. We switched to Repast Simphony (atop Groovy), which is optimized for simulating agents, but not especially well documented, and difficult to debug due to being interpreted instead of compiled. Little extension beyond the original feature set was accomplished; using some lighter-weight library for the simulation of agents would be preferable.

Currently, reproducibility of runs requires saving the random number generator seed, since some decisions made by agents are random. Given a sufficiently complex simulation, it should be useful to be able to reproduce runs (if some initial conditions are randomly generated, they should be saved), and to record

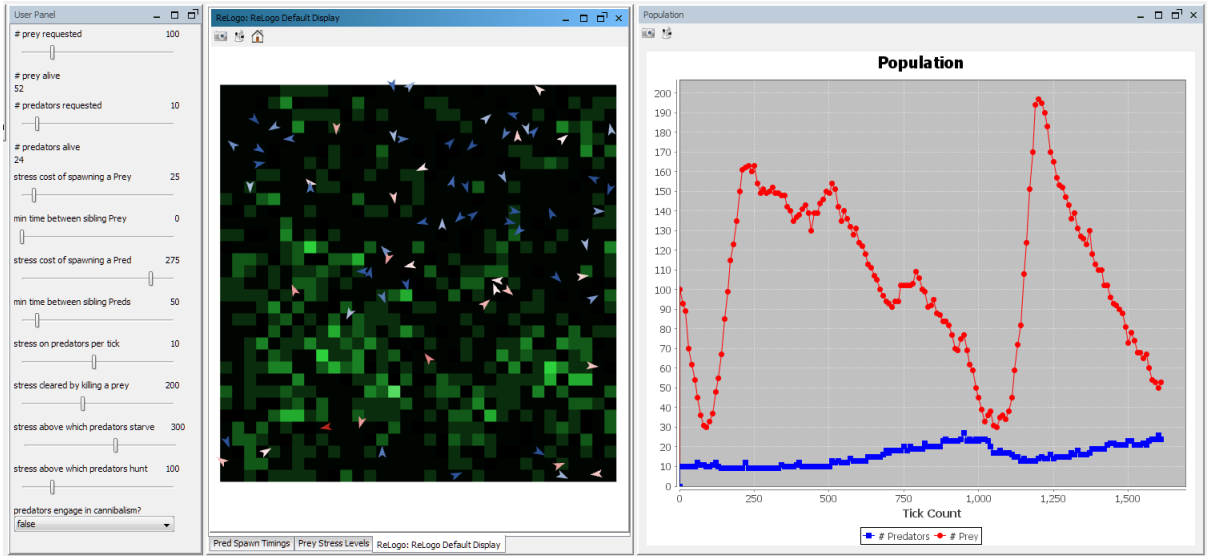

Figure 3: Visualization produced by a run of the agent-based simulation, with generated population graph to right.

output variables such as the per capita growth rate (over time) of each species.

2.4 Further Goals

- Track the spatial distribution of agents. Does it correspond to the fertility of the underlying grass? If impassable tiles and simple pathfinding were implemented, would it reflect the presence of bottlenecks? If heredity were implemented, would subspecies arise?
- Make stressed prey slower than unstressed prey, so that hereditary preference for stressed prey when hunting is adaptive, should that be implemented. Unstressed prey can then be faster than predators, in general, so that hunt failure is no longer arbitrary.
  - Besides allowing healthy prey a 'speed refuge' from predators, we may allow any prey to use one of a small number of 'hidden refuges' so long as it is unoccupied - although any decision(s) made by agents introduce a surfeit of complexity-increasing parameters.
- Add a rule that gives prey safety in numbers. Attempt to develop (either by explicit declaration or by long process of simulated evolution) prey behaviors which exploit this rule, and predator behaviors that circumvent it.
- Force grass to reseed depleted tiles, so that areas can become depleted. Determine conditions under which predators successfully exert top-down control on prey populations, preventing grass extinction.

3 A PDE Model

3.1 Description

Following presentation of the agent-based approach, our revised plan became to model the behavior of populations rather than that of discrete individuals via a reaction-diffusion equation - like those used in cellular biology to model reaction dynamics and pattern formation (as in growth & development). This results in a simpler model which is less prone to tipping into a failure state, whilst retaining substantial descriptive power.

Algorithm 3 Same as Algorithm 1, but with diffusion term.

$$\begin{aligned} \frac{du}{dt} &= \alpha u\,(1-u) - \beta uv - D\nabla^2 u \\ \frac{dv}{dt} &= \beta uv - \gamma v - D\nabla^2 v \end{aligned}$$

We discretized space via a crude finite difference method. We solved the resulting system of ODEs iteratively over discrete time. For our purposes - capturing qualitative behavior, - this level of approximation appears to suffice.

3.2 Results

We saw spatial oscillations until the space became saturated, then rapid (or immediate) convergence to a non-oscillatory steady state. When neither self-limitation of prey nor fixed boundaries were present, the oscillations over time (following saturation of the space) persisted much longer (Figure 4b).

See Figures 4, 5. For all runs, we used parameter values $\alpha = .5$, $\beta = 1$, $\gamma = .4$ and a 100x100 grid.

3.3 Technical

We implemented this model in Jython, and were thus able to use Processing for visualization. We converted the PDE to a system of ODEs via finite difference method on a rectangular grid. We computed the time-course of the PDE iteratively, over discrete time, using the first-order Runge-Kutta method (whose accuracy appeared to suffice).

3.4 Further Goals

Using the reaction-diffusion approach, we may implement 'high-level' interpretations of many of the same ideas as proposed for the agent-based model. Specifially, what if not every tile has the same carrying capacity? What if concentrated prey are resilient to predation? What if grass is modelled explicitly, as a third species subject to diffusion?

Although reaction-diffusion models cannot track individuals (except in a statistical sense), they suffice to capture emergent behavior - that is, the consequences of group dynamics.

4 Conclusion

We find a distinction between AI-like approaches (concentrating on the details of interaction between individuals with complex state) and development-like approaches (concentrating on the interactions between species with simple state) to modelling group dynamics. Either can be used to model both temporal and spatial effects, but the complexity of the AI-like approach seems more difficult to manage as the model's scope expands.

In the course of this initial foray, we chose to examine predator-prey dynamics. The population dynamics of our agent-based model behave like the Lotka-Volterra equations, up to discretization error, as expected. (Slow predator response to increasing prey population, with rapid predator response to decreasing prey population, appeared necessary to prevent extinctions.) Except when discretization error predominates, at small population sizes, we conclude that systems of ODEs are suitable for simulating ecological phenomena.

Generalizing from ODEs to PDEs is then a natural way to introduce spatial dynamics. It appears that diffusion alone (when not dominated by boundary conditions or prey self-limitation) suffices to dampen predator-prey oscillations when migration between locales is possible.

Studying predator-prey dynamics alone, considerable future work appears possible. Greater quantitative rigor may also be desirable, to be achieved using techniques from statistics (for agent-based models) and numerical simulation (for PDE models).

Figure 4: Visualization of a run (a) on a rectangle with boundaries fixed at $(u, v) = (0, 0)$ and (b) on a torus, without self-limitation of prey. Blue represents the predator population in each tile, and green represents the prey population.

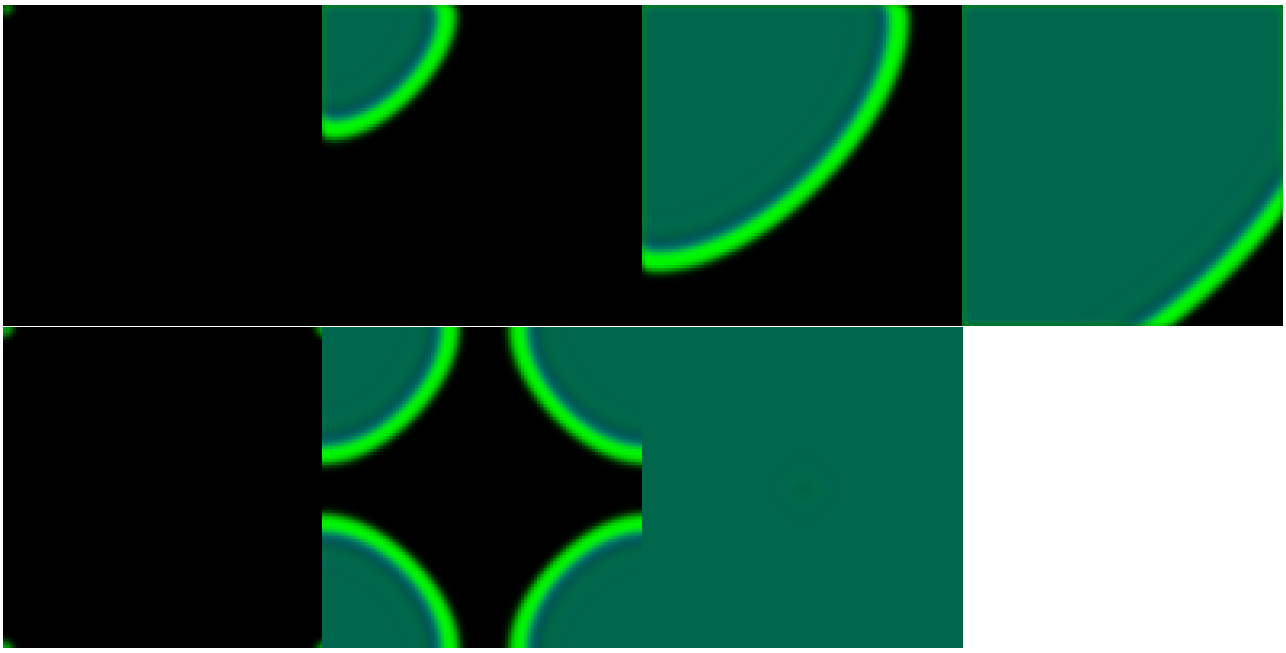

Figure 5: Visualization of a run (a) on a rectangle with boundaries fixed at $(u, v) = (0, 0)$ and (b) on a torus, *with* self-limitation of prey. Note that the stable-state *predator* population is now much lower (actually about half as big), whereas the stable-state prey population is actually slightly higher.

Code Appendix

Algorithm 4 Jython code for our PDE model.

```
W = 100; H = 100; scale = 8

class Cell(object):
    def __init__(self, u, v):
        self.var = [u, v]
class Boundary(Cell):
    def __init__(self): # assuming boundary conditions are '0 at all points'
        self.var = [0, 0]

cells = [[Boundary() if (i == H or j == W) else Cell(0, 0) \
          for j in range(W+1)] for i in range(H+1)]
cells[0][0] = Cell(1,1)

# to remove boundary & make space toroidal
# (FIXME: upper and left border are too large, as cells[i][0] == cells[i][-1].)
for j in range(W): cells[W][j] = cells[0][j]
for i in range(H): cells[i][H] = cells[i][0]
cells[H][W] = cells[0][0]

idx_u = 0; idx_v = 1
def neighbors(i, j, c = cells): # assuming a square grid:
    return [(c[i-1][j], c[i+1][j]), (c[i][j-1], c[i][j+1])]

def laplacian(idx, i,j):
    c = cells[i][j]; accel = []

    for (n0, n1) in neighbors(i, j):
        diffL = c.var[idx] - n0.var[idx]
        diffR = n1.var[idx] - c.var[idx]
        accel += [diffR - diffL]
    return accel

def diffuse(idx):
    laps = [[laplacian(idx, i,j) for j in range(W)] for i in range(H)]
    mags = [[sum(lap) for lap in row] for row in laps]

    print idx, ':', cells[1][1].var[idx], laps[1][1], mags[1][1]
    for i in range(H):
        for j in range(W):
            cells[i][j].var[idx] += .1 * mags[i][j]

def react(f):
    for i in range(H):
        for j in range(W):
            tmp = cells[i][j].var
            for k in range(len(f)):
                cells[i][j].var[k] += f[k](*tmp)

# f_u = lambda u,v: .5*u*(1-u) - u*v  # with self-limitation of prey
f_u = lambda u,v: .5*u - u*v          # without
f_v = lambda u,v: u*v - .4*v

# SNIPPED: window setup, drawing calls
def update():
    diffuse(idx_u); diffuse(idx_v)  # FIXME: compute both components
    react([f_u, f_v])               #        of du/dt before updating u
```