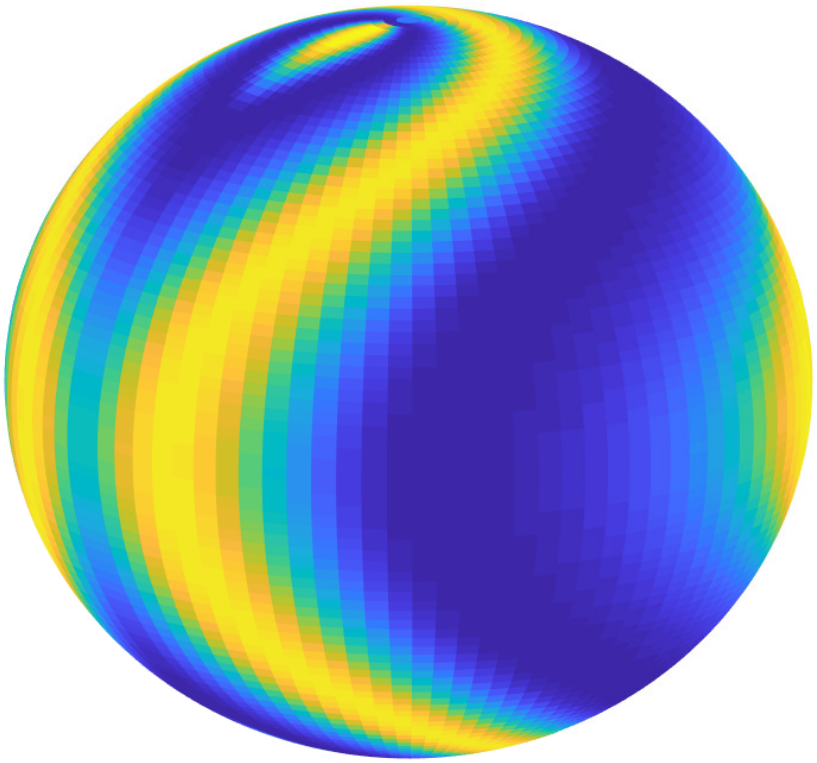
Evolution of biological Reaction Diffusion Equations on various surfaces



by
Hannah Weinmann
Jonas Kleineisel

supervised by
Prof. Dr. Alfio Borzi

University of Würzburg
August 2019

Abstract

We use ordinary differential equations to model populations of two interacting biological species, mainly focussing on predator-prey systems. Confining our discussion to predator-prey systems with limit cycle behaviour, we subsequently add a spatial dimension to consider reaction-diffusion equations. This models spatial migration of the two populations. We observe the emergence of spatial waves as a result of the population dynamics in combination with diffusion. Different boundary conditions for rectangular domains as well as the surface of a torus are considered. After shortly discussing an additonal convection term in the equation we turn our attention to the sphere, which requires some more care to be directed towards the numerical simulation. Many beautiful patterns arise from simulating different initial conditions on the sphere, which concludes our discussion.

Contents

1 Modelling of populations of two species with ODE's

A classic example of modelling with ordinary differential equations are population sizes of biological species, just think of the classical bacterial growth $u' = a\,u$ with solution $u(t) = u_0 \exp(at)$. Differential equations are well suited to model these types of systems since, at least after making some (more or less realistic) assumptions, the growth of a population is determined only by the current population size and some environmental factors like for example the amount of food available. From this one can often write down a differential equation describing a simplified version of the mechanism of reproduction. A lot of information on this topic is given in [Mur04a]. However even when considering just one species, a lot of drastically simplifying assumptions have to be made, to obtain a simple enough equation. Therefore, as always in modelling, one has to constantly weigh mathematical simplicity against realisticness.

1.1 Lotka-Volterra equations

The easiest type of model for biological populations is described by the well-known Lotka-Volterra equations

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

where u represents the population of a species of prey and v that of of predators that eat the prey. One assumes that in absence of predation, the prey just reproduce exponentially. Additionally, the population of prey is controlled by predation which is proportional to the prey population size as well as the predator population size. The predators grow proportional to the predation term in the prey equation and in absence of prey just die exponentially for lack of food.

To avoid confusion the system of equations is given in the so-called *nondimensionalized* form. This means that through a series of subsitutions all variables have been normed and parameters expressed in relation to each other to obtain a minimal number of parameters. In this case the growth rate of the preys in absence of predators is normed to be 1 and the only remaining parameter is $\alpha > 0$ describing the relative growth rate of the predators to that of the prey.

Characteristic for Lotka-Volterra equations is the periodicity of the solutions. For some given initial condition, the solution in the phase plane plot is always a closed curve on which the initial condition lies. At the same time, this is a major disadvantage of this model, since any small perturbation will move the trajectory onto

some other orbit, which may differ much from the original orbit at some later time. This is unrealistic since for real-world systems one would expect for the population size to return to the same limit cycle after some small perturbation. We will see later how to improve the model to incorporate this behavior.

We simulate the evolution of these and all subsequently described ordinary differential equations using Matlab's built-in ODE solver "ode23". We plot both of the populations over time as well as the prey-predator vector for all times in a two-dimensional plot called a phase plane plot. This kind of plot is especially useful when observing periodic behavior. We add some additional markers to indicate the time.

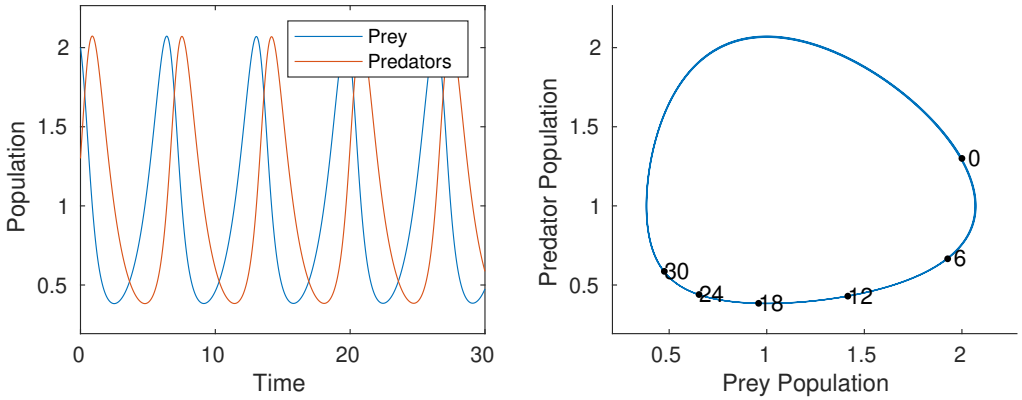


Figure 1.1: Population over time and phase plane plot for Lotka-Volterra equations with $\alpha = 1, u(0) = 2, v(0) = 1.3$

One sees that the solution is periodic since the phase plane plot is a closed curve. In figure 1.2 we see that the initial condition always lies on the orbit curve that the system runs through forever.

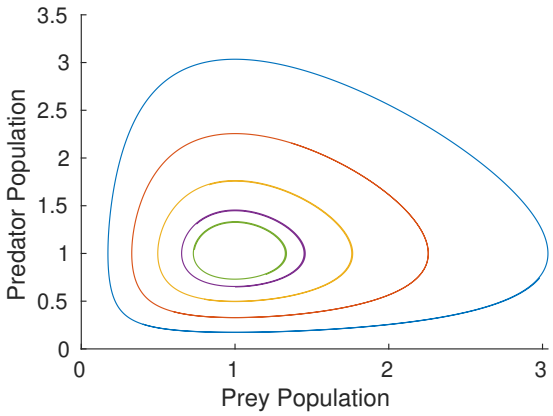


Figure 1.2: Orbits in the phase plane plot for different initial conditions in the Lotka-Volterra model

1.2 Logistic growth

One step towards a more realistic model is to guarantee that no species can grow unboundedly since this is obviously impossible in nature. For any species, the environment has a natural *carrying capacity* which indicates how many individuals organisms the environment can support sustainably. If the population size exeeds this limit, we want our equation to reflect this by decreasing the population size until it reaches again the carrying capacity where the population will stabilize. This can be achieved by adding a quadratic term to the equation which will only become significant once the population size is large. This kind of growth is called *logistic growth* and has first been proposed by Pierre-François Verhulst in 1838, see also [Mur04a, p. 3]. For one species, an equation modelling logistic growth with carrying capacity K is given by

$$\frac{du}{dt} = u\left(1 - \frac{u}{K}\right)$$

In the plots below one can see that this yields the desired behavior as the population size always returns to the carrying capacity.

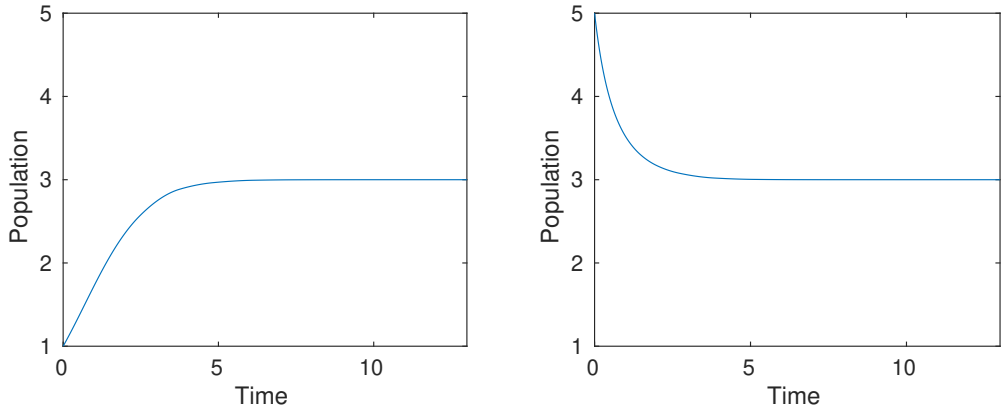


Figure 1.3: Logistic growth for carrying capacity $K = 3$ and initial contitions $u(0) = 1$ and $u(0) = 5$

In all subsequent models we will use logistic growth. As part of the nondimensionalization for the sake of simplcity, the carrying capacities will always be normed to be 1. This is equivalent to understanding all population sizes to be *percentages* of the respective carrying capacities.

1.3 Symbiosis, competition and parasitic behavior

A simple extension of Lotka-Volterra equations are models of various relationships between two species. We consider two species u and v which have now a general relationship in the sense that the prevalence of one species will have a supportive or destructive effect on the other species. The difference to the Lotka-Volterra

equations is that both species can survive alone, while in the Lotka-Volterra model the predators can not subsist without prey. We assume that each species to exhibit a logistic growth behavior in absence of the other species. Taking the symbiotic or competitive effect to be proportional to the population sizes of both species this leads to the following equations:

$$\begin{aligned} \frac{du}{dt} &= u\,(1 - u + av) \\ \frac{dv}{dt} &= \rho v\,(1 - v + bu) \end{aligned}$$

As discussed before, the system is nondimensionalized with both carrying capacities normed to 1. The remaining parameters are:

ρ	relative growth rate of species v in relation to growth rate of species u
a	coefficient determining the effect of species v on u
b	coefficient determining the effect of species u on v

For $a, b > 0$ we have symbiosis while for $a, b < 0$ we have competition. The cases with mixed signs $a < 0, b > 0$ and $a > 0, b < 0$ could be interpreted as describing two species with one beeing a parasite to the other where one species is beneficial to the other species while having a negative effect on the other species.

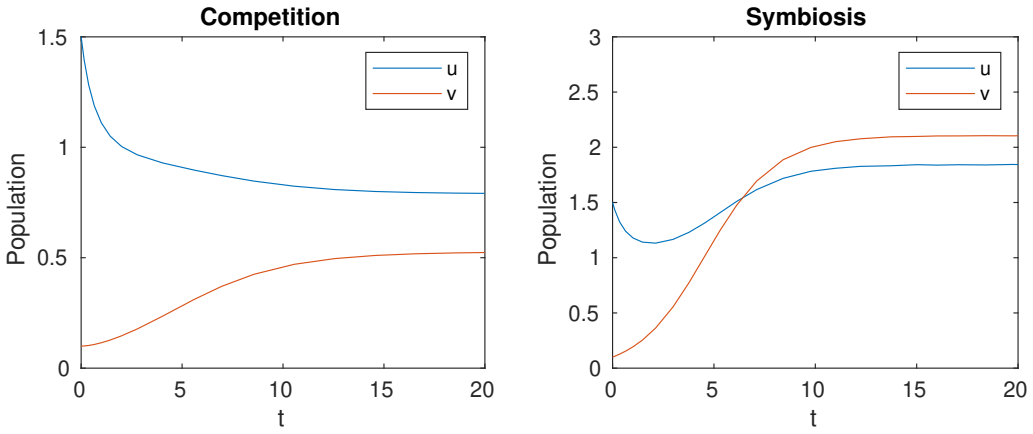


Figure 1.4: Competition with (ρ, a, b) = (1, -0.4, -0.6) and symbiosis with (ρ, a, b) = (0.4, 0.4, 0.6) and in both cases $u(0) = 1.5, v(0) = 0.1$

In these models, there is typically no perodic behavior as in the Lotka-Volterra equations, instead the populations tend to stabilize after some time. One can clearly see the effect of the interaction of the species on the steady states is to raise the limit population in case of a positive effect and decrease it in case of a negative effect. In the parasitic case we see that if the negative effect that the parasite has on the other species is too great ($b > 1$), this species will die out and just leave the parasite which will then alone just behave in a logistic way.

It is not very hard to see how one can generalize this system of two species to a system of n interacting species. On these types of systems with $n > 3$, not much

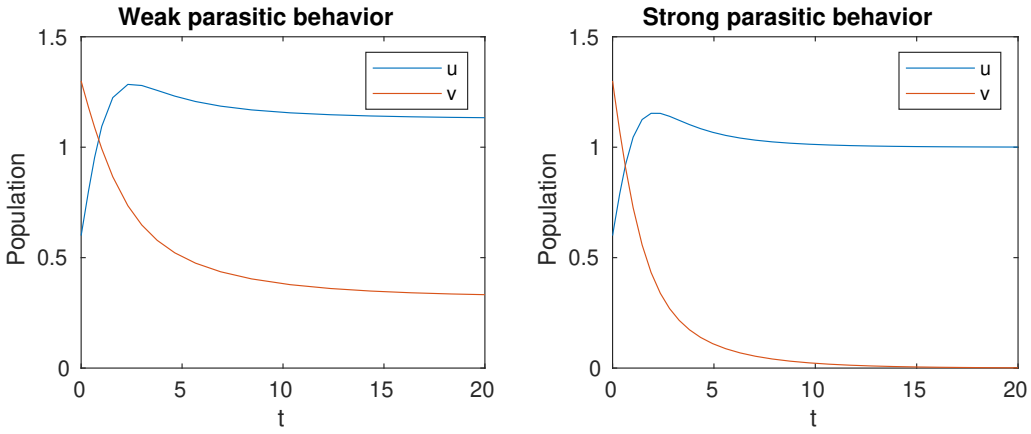


Figure 1.5: Weak parasitic behavior with (ρ, a, b) = (0.4, 0.4, -0.6) and strong parasitic behavior with (ρ, a, b) = (0.4, 0.4, -1.7) and in both cases $u(0) = 0.6, v(0) = 1.3$

research has been done. For some, yet limited, more information see [Mur04a, p. 101].

1.4 Realistic predator-prey equations

We now want to develop a system of more realistic equations for predator-prey systems. Of course there are generally many options and no one model can be definitively praised as better than all others, instead we need to decide where to make trade-offs between realisticness and simplicity. A major disadvantage of the Lotka-Volterra we saw before is the fact that small perturbations alter the orbit in ways that can have large effects later in time and will cause the model to jump to another orbit and never return to its state before the perturbation. We now want to get rid of this problem and have a model with a periodic solution which is robust to small perturbations, i.e. always returns to the same limit cycle, no matter what initial condition we choose.

For some constant environment, i.e. some constant food source we want both populations to exhibit logistic growth since this kind of growth is, as we have seen above, simple to model in a mathematical way and at the same time shows reasonably realistic behavior. As in the Lotka-Volterra equations we assume the food source of the prey to be constant such that in absence of predators the prey just grow logistically. The two factors limiting the growth of the prey is overpopulation (modelled by logistic growth) and predation. In the Lotka-Volterra equations as well as our model for competition we assumed this negative effect to be proportional to the population size of the prey as well as the predators. However this is unrealistic since for a small predator population with a large number of prey, one can expect the effect of predation on the prey to saturate and not grow any further, even if the prey increase, until the predators reproduce and therefore eat more prey. There are different options for how to model this saturation, see [Mur04a, p. 88] for some

options. We use the term $a\frac{uv}{u+d}$ with some $a, d > 0$ to describe the predation. The parameter a expresses the negative effect the predation has on the prey population in relation to the positive effect the predation has on the predator population. This is, for large a the predators need to eat many prey to reproduce some amount while for small a the predators only need to eat few prey to experience the same growth. The parameter d prevents the predation from becoming infinitely large for small prey population and additionally makes the equation more numerically stable. For the predators we want them to exhibit logistic growth for a constant environment, i.e. for a constant prey population. This is actually the same as saying the predators grow logistically with a carrying capacity depending monotonically on the prey population. We take the carrying capacity to be proportional and even equal to the prey population.
This leads to the following system of equations:

$$\begin{aligned} \frac{du}{dt} &= u\left((1-u) - a\frac{v}{u+d}\right) \\ \frac{dv}{dt} &= bv\left(1 - \frac{v}{u}\right) \end{aligned}$$

The parameter $b > 0$ expresses the linear growth rate of the predators in relation to that of the prey, at least while being away from limiting factors to the population like overpopulation and large predation. With $b > 1$ the predators reproduce faster that the prey while for $b < 1$ it is the other way around.

For the right parameter values, this system posesses a stable limit cycle. One can actually carry out a detailed analysis to determine precisely for which parameters the system is unstable or admits a stable limit cycle. In the case of the system having a stable limit cycle there is always an unstable equilibrium point encircled (in the phase plane plot) by the limit cycle. This kind of analysis however is not our main focus since we are mostly concerned with modelling and numerical simulation. For a detailed treatment of exactly our model equations see [Mur04a, p.88]. For us, it suffices to summarize that the system is always stable for $a < \frac{1}{2}$ and for any larger a we need b and d to be large enough, for example $b > \frac{1}{a}$ or $d > (a^2+4a)^{\frac{1}{2}} - (1+a)$ always suffices (see [Mur04a, p.91]).

We see numerically in the plots 1.7 and 1.6 that our system behaves as we would like it to.

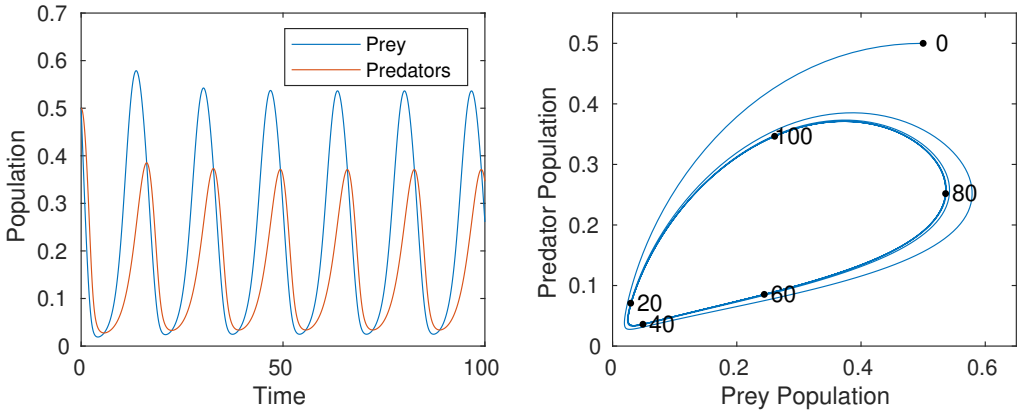


Figure 1.6: Realistic predator-prey equation with $a = 1, b = 0.5, d = 0.02$ and $u(0) = 0.5, v(0) = 0.5$

In plot 1.7, we see that the system always tends to the same limit cycle, regardless of the initial condition. For different initial conditions it may however take longer to stabilize.

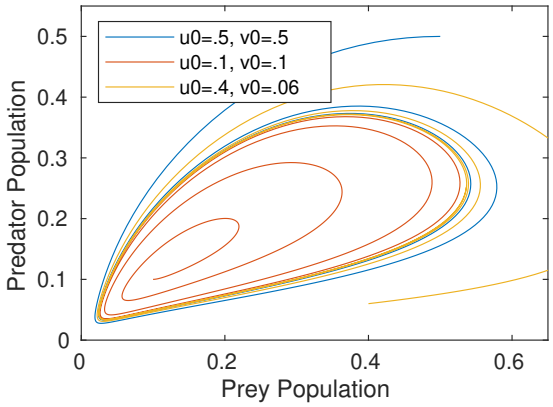


Figure 1.7: Realistic predator-prey equation with $a = 1, b = 0.5, d = 0.02$ for different initial conditions

One problem of our model is the singularity in the predator equation for $u = 0$, such that we cannot actually compute the behavior of the predators without prey. For small u, the term $\frac{u}{v}$ relates to the fact that we want the predators to die out quicker, the less prey and therefore the less food is available. This problem does not really occur when just looking at one-dimensional models since the model is uninteresting without prey anyway. However when studying spatial models it may very well be the case that through diffusion or just directly from certain initial conditions one gets predators in areas where there are no prey. One could try to regularize the equation with some small ε in the form $bv\left(1 - \frac{v}{u+\varepsilon}\right)$ but this has the disadvantage that then without prey, the predators do not die out completely but rather tend to some small limit population. Also we found in our simulations

this ε having to be rather large in order to have any regularizing effect if we did not want to make the time step dt unresonably small. A more practical solution is to program the simulation to set the predator population instantly to zero once the prey population falls below some small threshold. This does introduce some discontinuity in the model but gets around the problem of residual populations and makes it possible to work with initial conditions where there are no prey in some areas.

2 Spatial Diffusion

After analyzing the local interaction of population dynamics, we want to extend our model and add a spatial component. Since we assume that each animal of a population moves around in a random way we can think of the movement as a diffusion process. Equations that describe processes in which local interaction and additional diffusion are linked, are called reaction diffusion systems. Mathematically seen such systems take the form of a parabolic partial differential equation of second order. We consider a two-dimensional spatial domain. The system of reaction diffusion equations is given by

$$\begin{aligned} \frac{\partial u}{\partial t}(x,y,t) &= F(u(x,y,t),v(x,y,t)) &+ \sigma_{prey}\,(\Delta u)(x,y,t) \\ \frac{\partial v}{\partial t}(x,y,t) &= G(u(x,y,t),v(x,y,t)) &+ \sigma_{predator}\,(\Delta v)(x,y,t) \end{aligned}$$

The functions u and v depend on the two space variables x, y and time t. They describe the population dynamic of a predator-prey system. As before, u represents the prey population and v the predator population.

The functions F and G describe the reaction and are given by the previously discussed realistic predator-prey equations from chapter 1.4

$$\begin{aligned} F(u,v) &= u\left((1-u) - a\frac{v}{u+d}\right) \\ G(u,v) &= bv\left(1 - \frac{v}{u}\right) \end{aligned}$$

whose parameters have already been explained there.

The terms $\sigma_{prey}\,\Delta u$ and $\sigma_{predator}\,\Delta v$ describe the diffusion with diffusion coefficients $\sigma_{species}$ specific to the species. The diffusion coefficients indicate how fast the dispersion from a high to a low density takes place.

2.1 Discretization and numerical method

To solve our PDE numerically the function must be discretized. Consider a two-dimensional quadratic space of length L. Then the step size h is given by $h = \frac{L}{N-1}$ with N being the number of grid points. At each grid point (x_i, y_j) a value is assigned that represents the population of the species. This value is denoted by $u(x_i, y_j)$ respectively $v(x_i, y_j)$. As time step size we use $dt = \frac{T}{M}$ with T the total simulation time and M the number of time steps. From now on we discuss everything only for the preys u since everything goes analogous for the predators v anyway.

As an approximation for the derivative we use the explicit Euler method

$$\frac{\partial}{\partial t}u(x,y,t)\bigg|_{t_k} \approx \frac{u(x,y,t_k+dt)-u(x,y,t_k)}{dt}$$

The second derivative can be approximated by

$$\frac{\partial^2}{\partial x^2}u(x,y,t) \approx \frac{u(x+h,y,t)-2u(x,y,t)+u(x-h,y,t)}{h^2}$$

This leads to

$$\triangle u|_{(x_i,y_j)} \approx \frac{u(x_{i+1},y_j)+u(x_{i-1},y_j)-4u(x_i,y_j)+u(x_i,y_{j+1})+u(x_i,y_{j-1})}{h^2}$$

Combined with the notation of $u(x_i,y_j,t_k)$ by $u^k_{i,j}$ the discrete reaction diffusion equation is then given by

$$\frac{u^{k+1}_{i,j}-u^k_{i,j}}{dt} = F(u^k_{i,j},v^k_{i,j}) + \sigma_{prey} \ \triangle u|_{(x_i,y_j)}$$

with $k \in \{1,...,M\}$ and $i,j \in \{1,...,N\}$. Conversion leads to the iteration formula

$$u^{k+1}_{i,j} = u^k_{i,j} + dt\, F(u^k_{i,j},v^k_{i,j}) + dt\ \sigma_{prey} \ \triangle u|_{(x_i,y_j)}$$

As we have seen in the previous chapter, for the right parameter values the system of realistic predator-prey equations possesses a stable limit cycle. By maintaining these parameters for the reaction diffusion equation the question arises, what kind of spatial pattern formation can possibly occur. Of course this depends on the chosen initial conditions, for which one can for instance pick a Gaussian distribution. This choice is particularly reasonable since the special case of no preys, whose problems were discussed above, does not arise. But of course other choices for the initial conditions are possible.

In order to obtain a stable algorithm, it is important to analyze which stability conditions must be satisfied to get a well-conditioned problem. Since our main focus is not on the derivation of the stability condition, we only want to state that such a condition is relevant and in our case it must hold $\sigma\frac{dt}{h^2} < 1$. However this condition is merely necessary and by no means sufficient, since even for very small values of $\sigma\frac{dt}{h^2}$ the numerical computation can fail due to numerical errors.

2.2 Boundary conditions

In the interior of the grid, the values $u_{i,j}^{k+1}$ are well-defined by the above method with given initial conditions, but not at the boundaries. These grid points must be manually adjusted and are called boundary conditions for which different values can be chosen. In the following three different possibilities for the prey population are presented, which can be applied analogously to the predators.

2.2.1 Dirichlet boundary condition

In case of Dirichlet boundary conditions, the boundary values are set equal to zero, this means

$$\begin{aligned} u(x_1, y_j) = 0,\ u(x_N, y_j) = 0 \quad &\text{for all } j = 1, ..., N \\ u(x_i, y_1) = 0,\ u(x_i, y_N) = 0 \quad &\text{for all } i = 1, ..., N \end{aligned}$$

The figure 2.1 shows a simulation with Dirichlet boundary conditions. As initial values for both populations a Gaussian distribution with the position of the center $c_{prey} = (0.3, 0.3)$ and $c_{predator} = (0.2, 0.2)$, height $h_{prey} = 0.5$ and $h_{predator} = 0.25$, and standard deviation $d_{prey} = 0.1$ and $h_{predator} = 0.1$ is chosen. As diffusion coefficient we choose $\sigma = 0.0001$ for both populations.
An advantage of Dirichlet boundary conditions is that they are easy to compute. One could say, that this model illustrates island populations, that can not leave the island, or other szenarios where the species propagation is limited by untraversable boundaries.

2.2.2 Neumann boundary condition

A similar idea like Dirichlet boundary conditions is to consider smoother boundary values. So instead of setting them to zero, in case of Neumann boundary conditions, the boundary values are set as the next inner point value, this means not the function values are predefined, but the derivative on the boundary of the grid is set to zero, i.e.

$$\begin{aligned} u(x_1, y_j) = u(x_2, y_j),\ u(x_N, y_j) = u(x_{N-1}, y_j) \quad &\text{for all } j = 1, ..., N \\ u(x_i, y_1) = u(x_i, y_2),\ u(x_i, y_N) = u(x_i, y_{N-1}) \quad &\text{for all } i = 1, ..., N-1 \end{aligned}$$

The value of the corner points $u(x_1, y_1),\ u(x_1, y_N),\ u(x_N, y_1),\ u(x_N, y_N)$ doesn't matter, because for the computation of $u_{i,j}^{k+1}$ by the approximated reaction diffusion method the corner points are not involved.
Figure 2.2 shows a simulation with Neumann boundary conditions. As initial conditions and diffusion coefficient the same values as in the Dirichlet case are chosen. If you compare the Neumann simulation to the Dirichlet simulation one can see, that they are not very different from each other in the interior.

2.2.3 Torus boundary condition

Until now we considered a bounded rectangular with untraversable boundaries. To ensure a propagation in every direction the idea is to build a torus by folding the origin plane along the x and y axis. This means that plotted in two dimensions the left and right boundary are the same and equivalently the bottom an top boundary. In conclusion in case of torus boundary conditions for the initial values holds

$$\begin{aligned} u(x_i, y_1) &= u(x_j, y_N) \qquad \forall\ i = 1, ..., N \\ u(x_1, y_j) &= u(x_N, y_j) \qquad \forall\ j = 1, ..., N \end{aligned}$$

To ensure that the initial condition satisfies the voundary condition the initial values are set as the mean value between the two nearest points, i.e.

$$\begin{aligned} u(x_i, y_1) &= \frac{1}{2}(u(x_i, y_2) + u(x_i, y_{N-1})) \qquad \forall\ i = 1, ..., N \\ u(x_1, y_j) &= \frac{1}{2}(u(x_2, y_j) + u(x_{N-1}, y_j)) \qquad \forall\ j = 1, ..., N \end{aligned}$$

This means that the initial corner points must be equal. It follows

$$\begin{aligned} u(x_1, y_1) = \frac{1}{8}(&u(x_2, y_1) + u(x_1, y_2) + u(x_N - 1, y_1) + u(x_N, y_2) \\ &+ u(x_1, y_{N-1}) + u(x_2, y_N) + u(x_{N-1}, y_N) + u(x_N, y_{N-1})) \\ u(x_1, y_1) = u(x_1, y_N) &= u(x_N, y_1) = u(x_N, y_N) \end{aligned}$$

Figure 2.3 shows a simulation with Torus boundary conditions. As initial conditions and diffusion coefficient the same values as in the Dirichlet and Neumann case are chosen. One can see, that the waves are spreading over the edge, creating a completely different pattern as in the first two cases.
Since in the case of Torus boundary conditions the populations are considered on a torus, this of course leads to the idea of plotting the populations on a torus. To this end we implemented by another plot function. The disadvantage of such a plot style is that a large part is not visible and the propagation can not be comprehended so well. The next figure 2.4 shows a simulation on a torus and figure 2.5 shows the same simulation in a two dimensional plane plot as before.

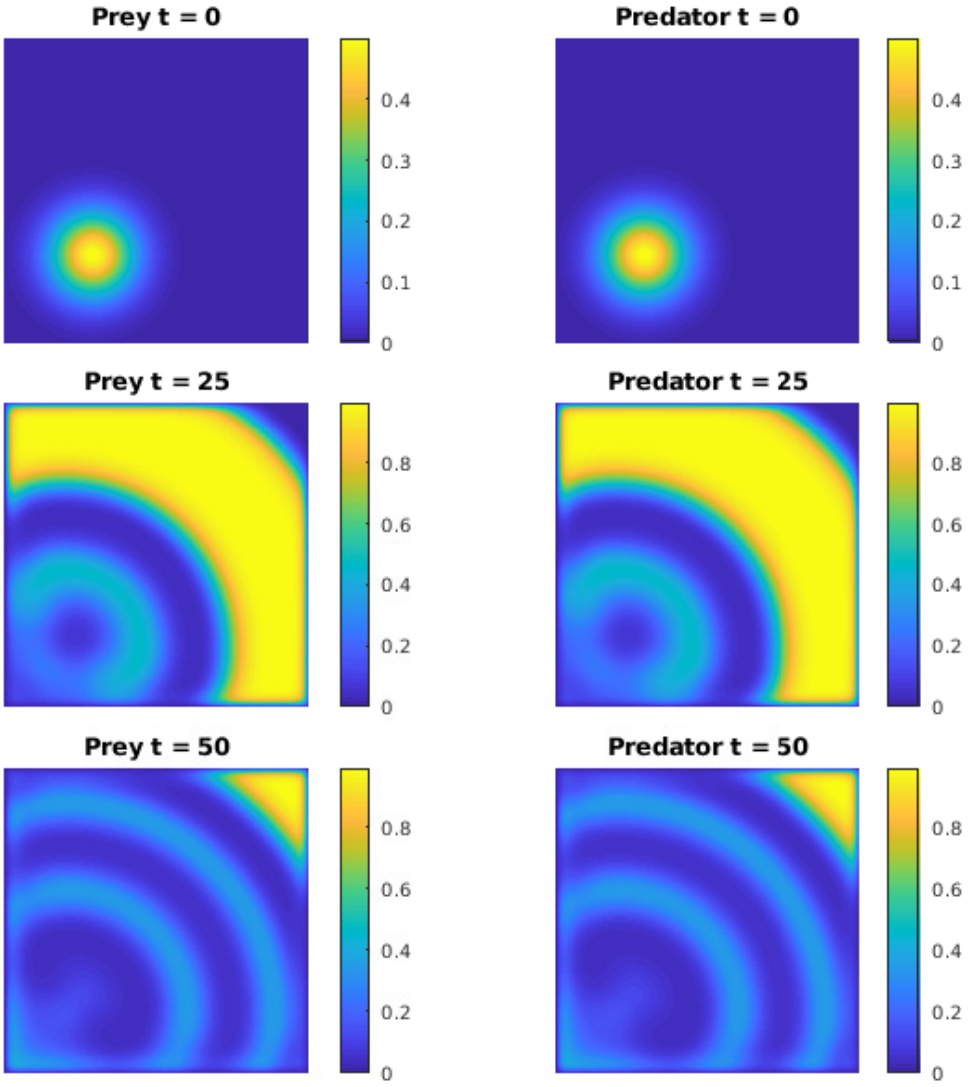


Figure 2.1: Simulation over time of reaction diffusion system and Dirichlet boundary conditions

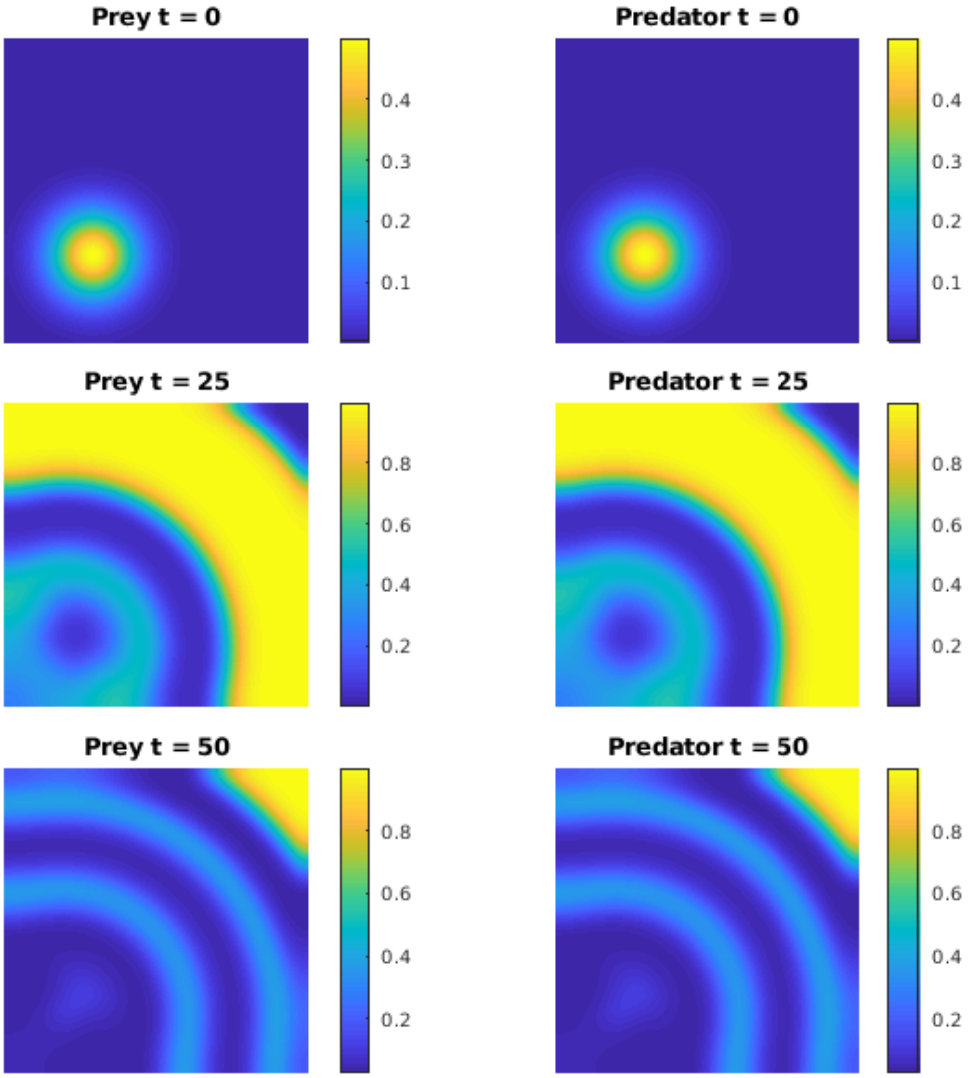


Figure 2.2: Simulation over time of reaction diffusion system and Neumann boundary conditions

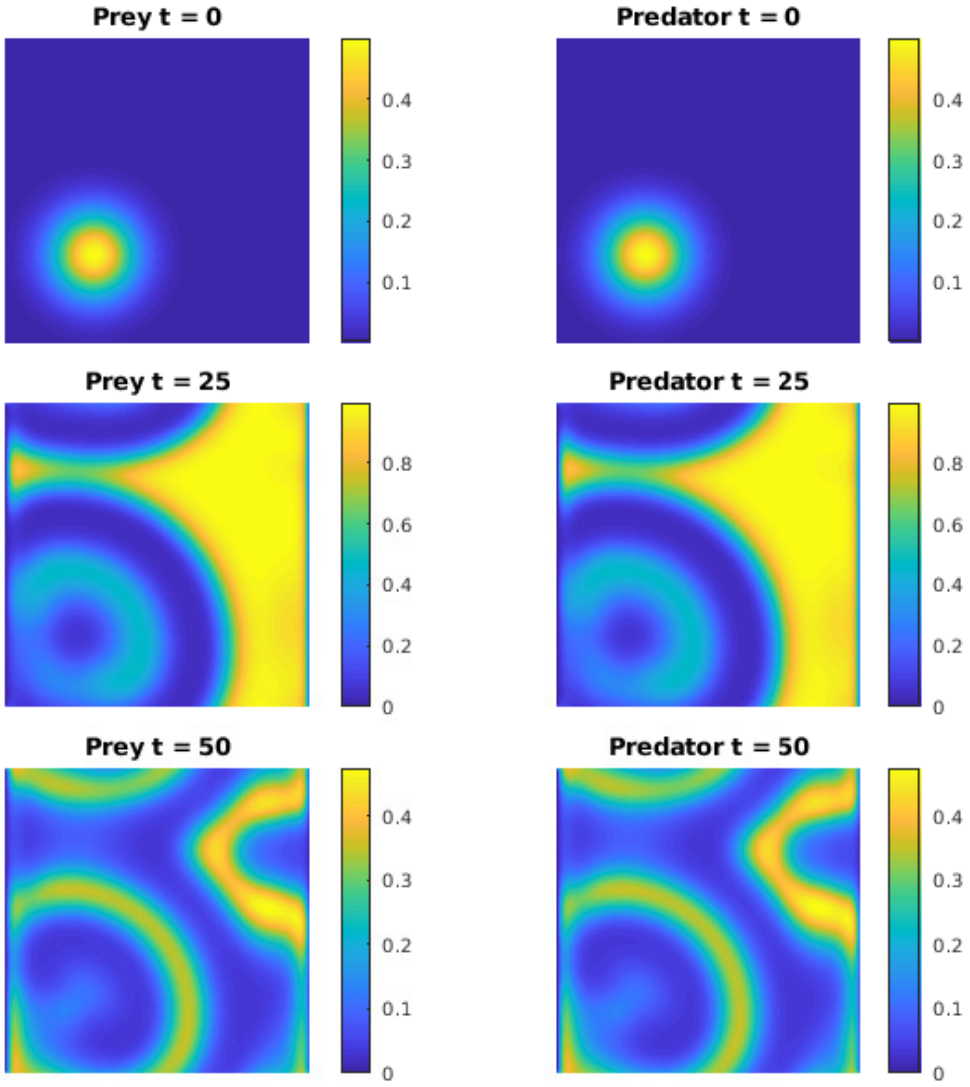


Figure 2.3: Simulation over time of reaction diffusion system and Torus boundary conditions

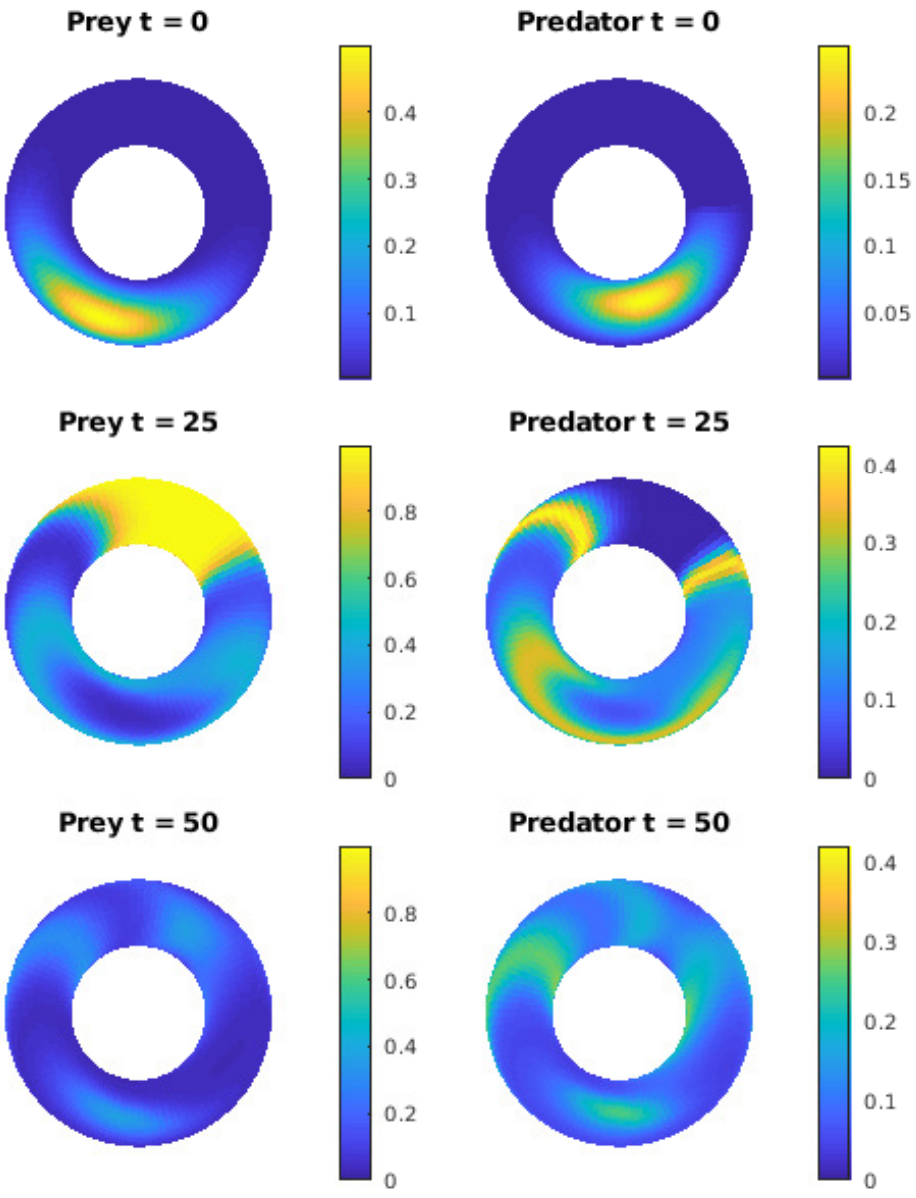


Figure 2.4: Simulation over time on a torus of reaction diffusion system and Torus boundary conditions plotted on a torus

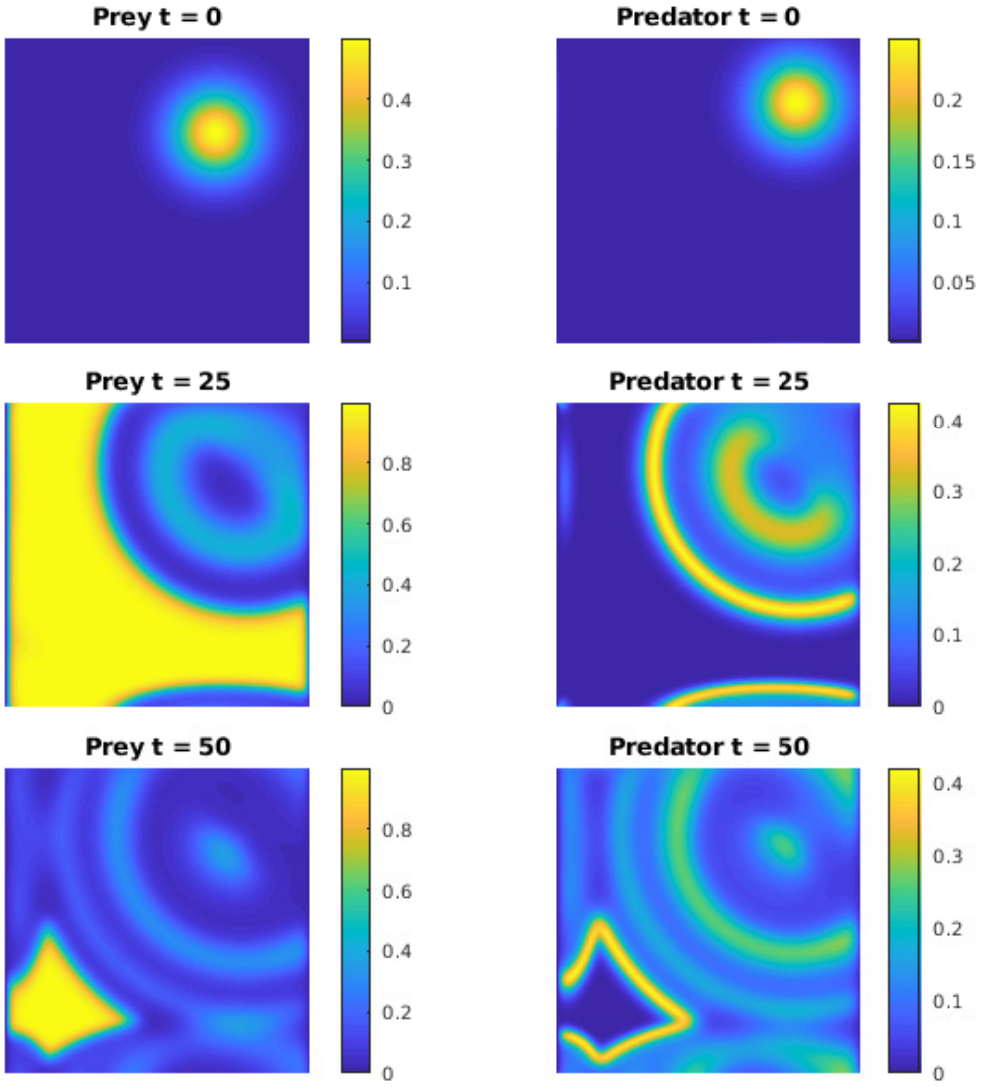


Figure 2.5: Simulation over time on a plane of reaction diffusion system and Torus boundary conditions as a planar plot

3 Convection

So far our only spatial effect was diffusion, which, as we saw, can be regarded as a constant drive in the system to equalize the population density in space. Another physical possibility is a force acting (constant in time) on the species. This can be isotropic, meaning the force vector field is constant in space, or anisotropic. Objects subjected to force of course resist this force in a way proportional to their mass. Here we interpret the population density as mass. If we imagine an isotropic convection to be similar to wind blowing things in some direction, then this basically just means that low concentrations are blown away easily while larger amounts are harder to displace.

If the convection force is represented by a vector field $\varphi : \Omega \to \mathbb{R}^2$, then the effect of convection can be modelled by adding the convection term $\varphi \cdot \nabla u$ to the partial differential equation. We thus obtain a system of *reaction-diffusion-convection* equations:

$$\begin{array}{rcllll} \frac{\partial u}{\partial t} & = & F(u,v) & - c_{prey}\,\varphi \cdot \nabla u & + \sigma_{prey}\Delta u \\ \frac{\partial v}{\partial t} & = & G(u,v) & - c_{predator}\,\varphi \cdot \nabla v & + \sigma_{predator}\Delta v \end{array}$$

We always take the same convection force to act on both predators and prey, but we allow for different convection speeds for the species, i.e. we rescale φ by a factor $c_{species}$. One could also consider completely different convection fields for the different species.

For the numerical simulation we take the forward Euler method to approximate the derivative. This leaves us with the choice to take the forward difference $\frac{\partial u}{\partial x}\big|_{(i,j)} \approx \frac{1}{h}(u_{(i+1,j)} - u_{(i,j)})$ or the backward difference $\frac{\partial u}{\partial x}\big|_{(i,j)} \approx \frac{1}{h}(u_{(i,j)} - u_{(i-1,j)})$. To increase stability, we always take the one, in which the convection direction for the x-component φ_x goes, i.e. for $\varphi_x > 0$ we take the forward difference and for $\varphi_x < 0$ we take the backward difference. We use the expression

$$\left.\frac{\partial u}{\partial x}\right|_{(i,j)} \approx \max(0,\varphi_x)\left(\frac{1}{h}(u_{(i,j)} - u_{(i-1,j)})\right) + \min(0,\varphi_x)\left(\frac{1}{h}(u_{(i+1,j)} - u_{(i,j)})\right)$$

to encapsulate this behavior. Everything goes analogous for $\frac{\partial u}{\partial y}$, $\frac{\partial v}{\partial x}$ and $\frac{\partial v}{\partial y}$ of course.

Concerning the numerical simulation, it is very noticable, how the simulation becomes significantly less stable. Even for moderate convection speeds one has to reduce the time step `dt` by a large factor. Of course this has the consequence of increasing the computation time considerably and only allowing small simulation times `T`. But for too large time steps the numerics is so inaccurate that in areas where there are only small populations the population becomes negative, which makes no sense. If decreasing `dt` is not an option one could always counteract this problem by cutting the solution at 0 in every step, which means to always assign at

least 0 as population everywhere. While this significantly increases stability it also distorts the solution in ways not intended by the model and generates numerical solutions which may deviate from actual solutions to the PDE arbitrarily much. Therefore we refrain from cutting and instead use small time steps and initial conditions that are easier to simulate.

We consider two examples, both with Neumann boundary condition. As an example for isotropic convection we take the constant convection direction $(-1, 0)^T$, convection speeds $c_{prey} = 1$, $c_{predator} = 0.01$ and diffusion speeds $\sigma_{prey} = 0.00001, \sigma_{predator} = 0.02$. This results in the dynamics shown in figure 3.1. Notice how the movement of the prey is almost exclusively driven by convection due to their high convection and low diffusion speed.

For anisotropic convection we consider a circular swirl around the center $(\frac{L}{2}, \frac{L}{2})$ of the domain. This means that the center remains fixed while any point (i, j) experiences convection with the vector field

$$\varphi(i, j) = \begin{pmatrix} -j + \frac{L}{2} \\ i - \frac{L}{2} \end{pmatrix}.$$

The parameters we use here are convection speeds $c_{prey} = 1$, $c_{predator} = 0.1$ and diffusion speeds $\sigma_{prey} = 0.00001, \sigma_{predator} = 0.02$. We see nicely in figure 3.2 how the prey, due to their high convection speed, get pulled in a swirl around the center. The predators have a significantly lower convection speed but we can still make ot the deformation the convection has on the predator population.

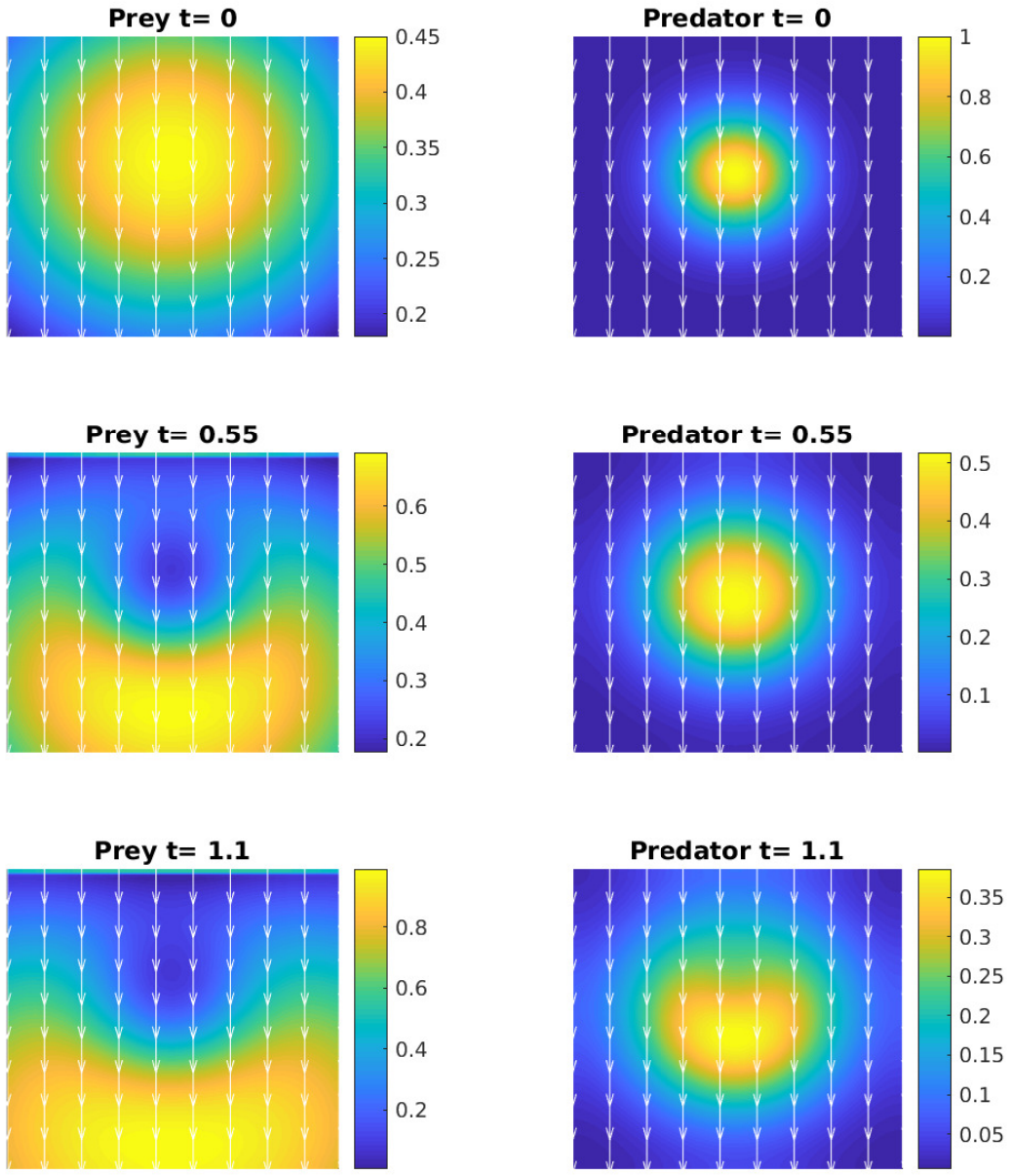


Figure 3.1: Isotropic convection with convection direction $\varphi = (-1, 0)^T$ and convection speeds $c_{prey} = 1$, $c_{predator} = 0.01$

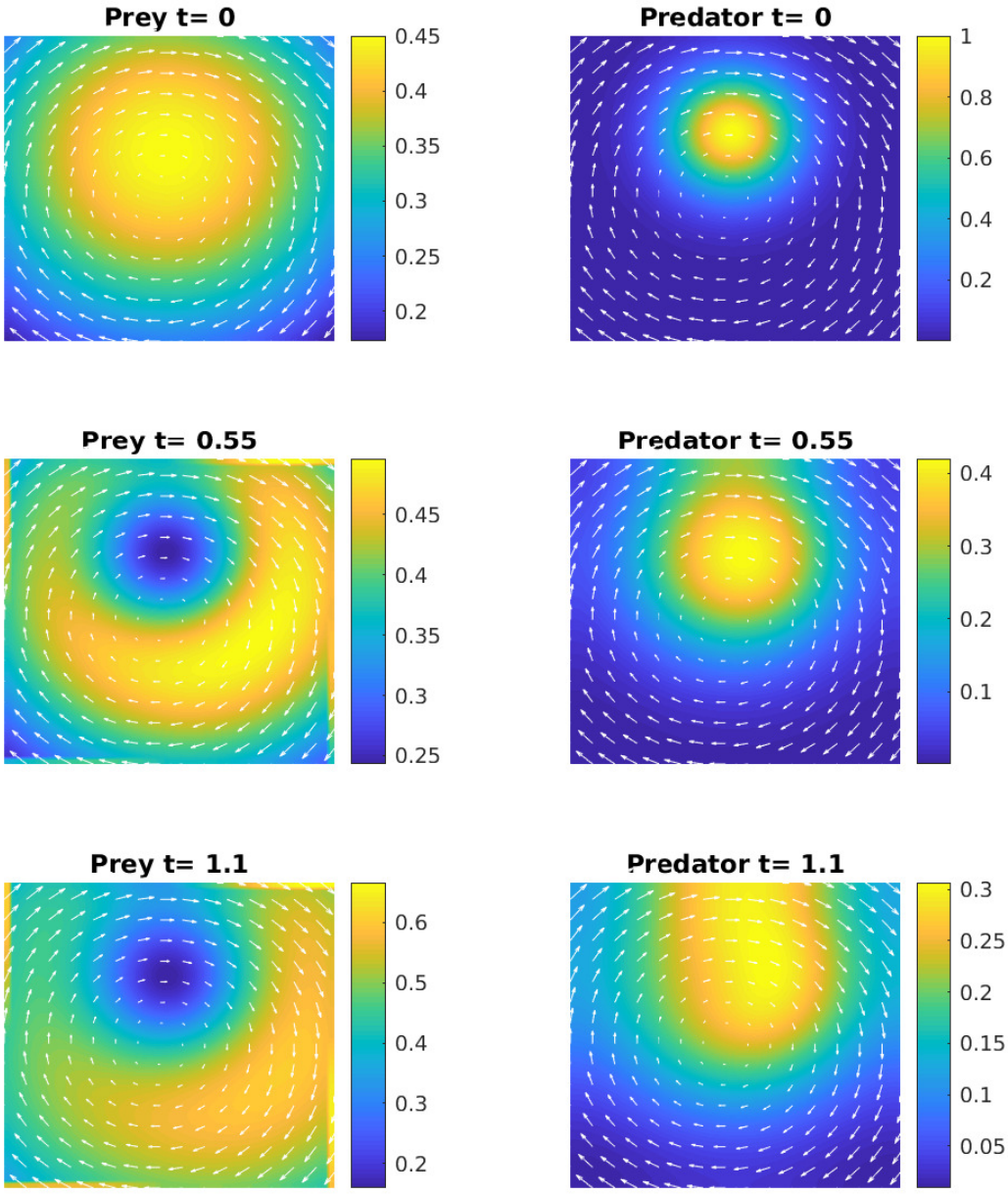


Figure 3.2: Circular convection with convection speeds $c_{prey} = 1$, $c_{predator} = 0.1$

4 Simulation on the Sphere

We will now disregard convection and return to our reaction-diffusion equation

$$\begin{array}{rcll} \frac{\partial u}{\partial t} & = & F(u,v) & +\sigma_{prey}\Delta u \\ \frac{\partial v}{\partial t} & = & G(u,v) & +\sigma_{predator}\Delta v \end{array}$$

with

$$\begin{aligned} F(u,v) &= u\left((1-u) - a\frac{v}{u+d}\right) \\ G(u,v) &= bv\left(1-\frac{v}{u}\right). \end{aligned}$$

for $u, v : \Omega \times [0,T] \to \mathbb{R}$. So far, we discussed a quadratic spatial domain $\Omega = [0,L]^2$ with different boundary conditions. Now, we want to take the sphere $\Omega = r\,S^2 \subseteq \mathbb{R}^3$ for some fixed radius $r > 0$ as spatial domain.

4.1 Discretization and numerical method

The natural choice for coordinates are spherical coordinates with latitude $\vartheta \in [0,\pi)$, longitude $\varphi \in [0,2\pi)$ and radius r. To write the equation in spherical coordinates we need to transform the Laplace operator from the usual cartesian coordinates to spherical coordinates. Using the transformation

$$\begin{aligned} x &= r\sin\vartheta\cos\varphi \\ y &= r\sin\vartheta\sin\varphi \\ z &= r\cos\vartheta \end{aligned}$$

we obtain the Laplacian in spherical coordinates

$$\begin{aligned} \Delta u &= \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \\ &= \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial u}{\partial r}\right) + \frac{1}{r^2\sin\vartheta}\frac{\partial}{\partial\vartheta}\left(\sin\vartheta\frac{\partial u}{\partial\vartheta}\right) + \frac{1}{r^2(\sin\vartheta)^2}\frac{\partial^2 u}{\partial\varphi} \end{aligned}$$

To discretize the sperical coordinates we define a $N_1 \times N_2$ grid by discretizing the longitude $\varphi \in [0,2\pi)$ into $N_1 - 1$ and the latitude $\vartheta \in [0,\pi)$ into $N_2 - 1$ steps, i.e. we set

$$\begin{aligned} \varphi_i &:= \frac{(i-1)2\pi}{(N_1-1)} && \forall i = 1,...,N_1 \\ \vartheta_j &:= \frac{(j-1)\pi}{(N_2-1)} && \forall j = 1,...,N_2 \end{aligned}$$

resulting in step sizes

$$d\varphi \;=\; \frac{2\pi}{(N_1-1)} \qquad\qquad d\vartheta \;=\; \frac{\pi}{(N_2-1)}.$$

Note that this way of discretizing the sphere has the disadvantage of distorting areas and lengths, since in very high or very low latitudes, one step in the longitude or the latitude direction constitutes a significantly shorter distance than at the equator. For the same reason one square in the grid near the poles has a smaller area than one square around the equator. We treat it though, as if every step in the discretization has the same physical length by treating every gridpoint equally in the diffusion. This has the effect of slowing diffusion around the poles and speeding it up near the equator. There are better ways of discretizing a sphere. It is however really simple to work with this discretization, which is why we prefer it for now.

We discretize time by specifying a simulation time T and a number of time steps M, resulting in a time step

$$dt \;:=\; \frac{T}{M}.$$

We treat only the prey function $u : rS^2 \times [0,T] \to \mathbb{R}$ but handle the predators v in exactly the same way. We set

$$u_{i,j,k} := u(r, \varphi_i, \vartheta_j, k\,dt) \text{ for } k = 1, ..., M$$

for abbreviation since u depends on both space dimensions as well as time but the radius is constant anyway.

Using the forward Euler method with forward differences in space we proceed to discretize the Laplacian in spherical coordinates

$$\begin{aligned}
\Delta u|_{(i,j,k)} &= \frac{1}{r^2 \sin\vartheta}\frac{\partial}{\partial\vartheta}\left(\sin\vartheta\frac{\partial u}{\partial\vartheta}\right)\Big|_{(i,j,k)} + \frac{1}{r^2(\sin\vartheta)^2}\frac{\partial^2}{\partial\varphi^2}\Big|_{(i,j,k)} = \\
&= \frac{1}{r^2 \sin\vartheta_{i,j,k}\, d\vartheta}\left(\sin\vartheta\,\frac{\partial u}{\partial\vartheta}\Big|_{(i,j+\frac{1}{2},k)} - \sin\vartheta\,\frac{\partial u}{\partial\vartheta}\Big|_{(i,j-\frac{1}{2},k)}\right) + \\
&+ \frac{1}{r^2(\sin\vartheta_j)^2(d\varphi)^2}\left(u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}\right) = \\
&= \frac{1}{r^2 \sin\vartheta_j\,(d\vartheta)^2}\Big(u_{i,j+1,k}\sin\vartheta_{i,j+\frac{1}{2},k} - u_{i,j,k}(\sin\vartheta_{i,j+\frac{1}{2},k} + \sin\vartheta_{i,j-\frac{1}{2},k}) + \\
&\quad + u_{i,j-1,k}\sin\vartheta_{i,j-\frac{1}{2},k}\Big) + \frac{1}{r^2(\sin\vartheta_j)^2(d\varphi)^2}\left(u_{i+1,j,k} - 2u_{i,j,k} + u_{i-1,j,k}\right)
\end{aligned}$$

by using

$$\begin{aligned}
\sin\vartheta\,\frac{\partial u}{\partial\vartheta}\Big|_{(i,j+\frac{1}{2},k)} &\approx \sin\vartheta_{j+\frac{1}{2}}\frac{u_{j+1}-u_j}{d\vartheta} \\
\sin\vartheta\,\frac{\partial u}{\partial\vartheta}\Big|_{(i,j-\frac{1}{2},k)} &\approx \sin\vartheta_{j-\frac{1}{2}}\frac{u_j-u_{j-1}}{d\vartheta}
\end{aligned}$$

Since we work on a sphere where the radius is constant, we have $\frac{\partial u}{\partial r} = 0$ and thus can omit the first term in the spherical Laplacian.

For time differentiation we use the explicit Euler method

$$\frac{u_{i,j,k+1} - u_{i,j,k}}{dt} = F(u_{i,j,k}) + \sigma \, \Delta u|_{i,j,k}$$

In summary, the update formula for the simulation is

$$u_{i,j,k+1} = u_{i,j,k} + dt \, F(u_{i,j,k}) + dt \, \sigma \; \Delta u|_{i,j,k} \quad \forall i = 1, ..., N_1 - 1, \; j = 2, ..., N_2 - 1$$

with $\Delta u|_{i,j,k}$ given above. Since the longitudes $\varphi = 0$ and $\varphi = 2\pi$ both correspond to zero-meridian, we have to set

$$u_{N_1,j,k} \; := \; u_{1,j,k} \;\; \forall \, j = 1, ..., N_2$$

at every time step. Note also that for $i = 1$ we have to replace $u_{i-1,j,k}$ in the discrete laplacian by $u_{N_1-1,j,k}$ in the discrete laplacian since $u_{i-1,j,k}$ is not well-defined for $i = 1$.

This leaves only the update behavior for the poles ($j = 1$ and $j = N_2$) to be discussed. At the north pole for example, i.e. $j = N_2$, we can not meaningfully say what an "eastern" neighbour $u_{i+1,j,k}$ should be, since at the north pole we can go nowhere but south. Since all points with $j = N_2 - 1$ are neighbours, it is reasonable to instead of the four adjacent neighbour we consider all points with $j = N_2 - 1$ for the diffusion and take the mean of all those values as replacement for the laplacian. This means, we update the north pole via

$$u_{i,N_2,k+1} = F(u_{1,N_2,k}) + dt \, \sigma \sum_{i=1}^{N_1} \frac{u_{i,N_2,k}}{N_2}$$

and analogously the south pole $j = 1$.

This concludes the description of our numerical method for the simulation of reaction-diffusion equations on the sphere. Additional information concerning the numerical solution of differential equations on the sphere with more advanced methods can also be found in [Bar89].

4.2 Results

With the method described above, we can generate many beautiful pictures. As initial conditions we limit ourselves to taking two Gauss curves with different centers and standard deviations. As already in the planar case, we see the emergence of spatial waves as a result of the population dynamics in combination with diffusion. But now, the waves traverse aroung the globe and interact with themselves. This can be nicely seen in figure 4.2.

Another example is given in figure 4.3. In this simulation, we chose the diffusion speeds by a factor of 10^{-2} lower than in figure 4.2. Therefore, the waves have much more defined edges and travel slower. If one looks closely, one can see how the wave front of the preys is always a little bit ahead of the wave of predators, which follows them. It is rather astonishing to see this behavior of pursuit emerging from only a few partial differential equations.

It is a little counterintuitive to plot data that represents a spherical surface on a planar map. We mainly prefer to plot the projection, since this allows us to see the entire surface in one picture. However, some effects can be visualized better on the surface of a sphere. In figure 4.4 we plot the same data as in figure 4.3, but on a sphere. In figure 4.1 we focus on the last two plots of figure 4.4 for $t = 80$ and rotate the spheres appropriately. This allows us to observe how the wave front interacts with itself after travelling around the globe and shrinks to a point near the north pole. The next wave is already closing in to repeat the process.

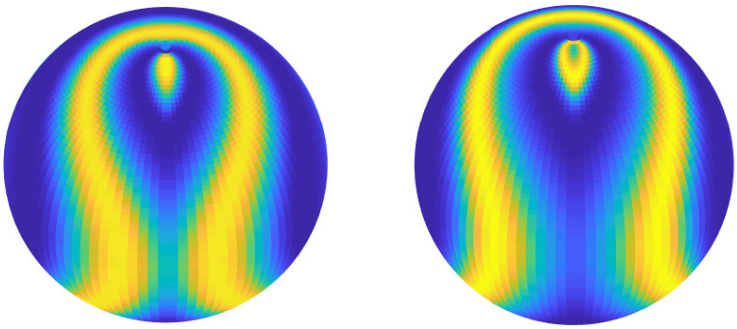


Figure 4.1: The wave front after travelling around the globe

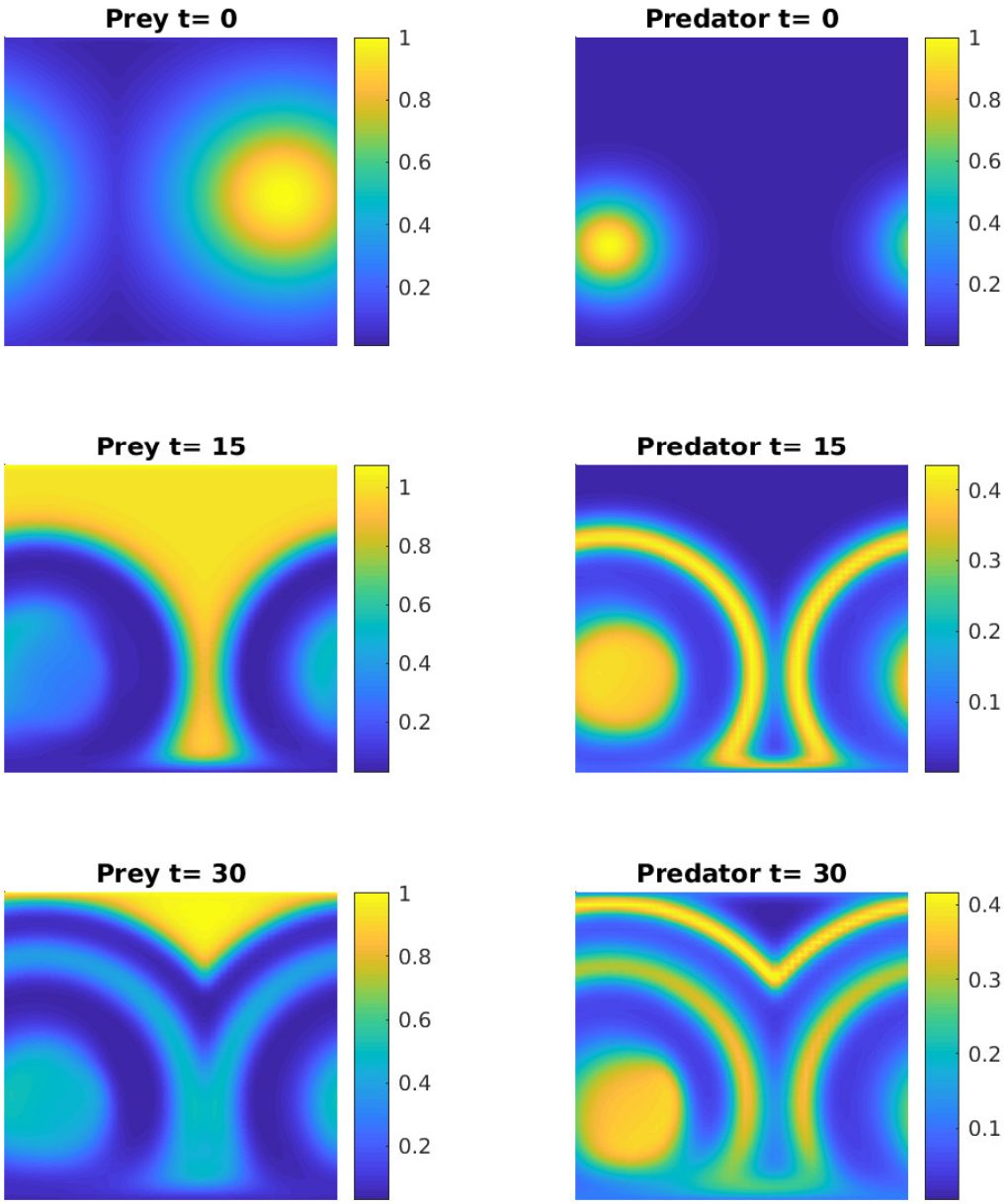


Figure 4.2: Simulation of two Gauss curves on the sphere with $\sigma_{prey} = 0.001$, $\sigma_{predator} = 0.0001$

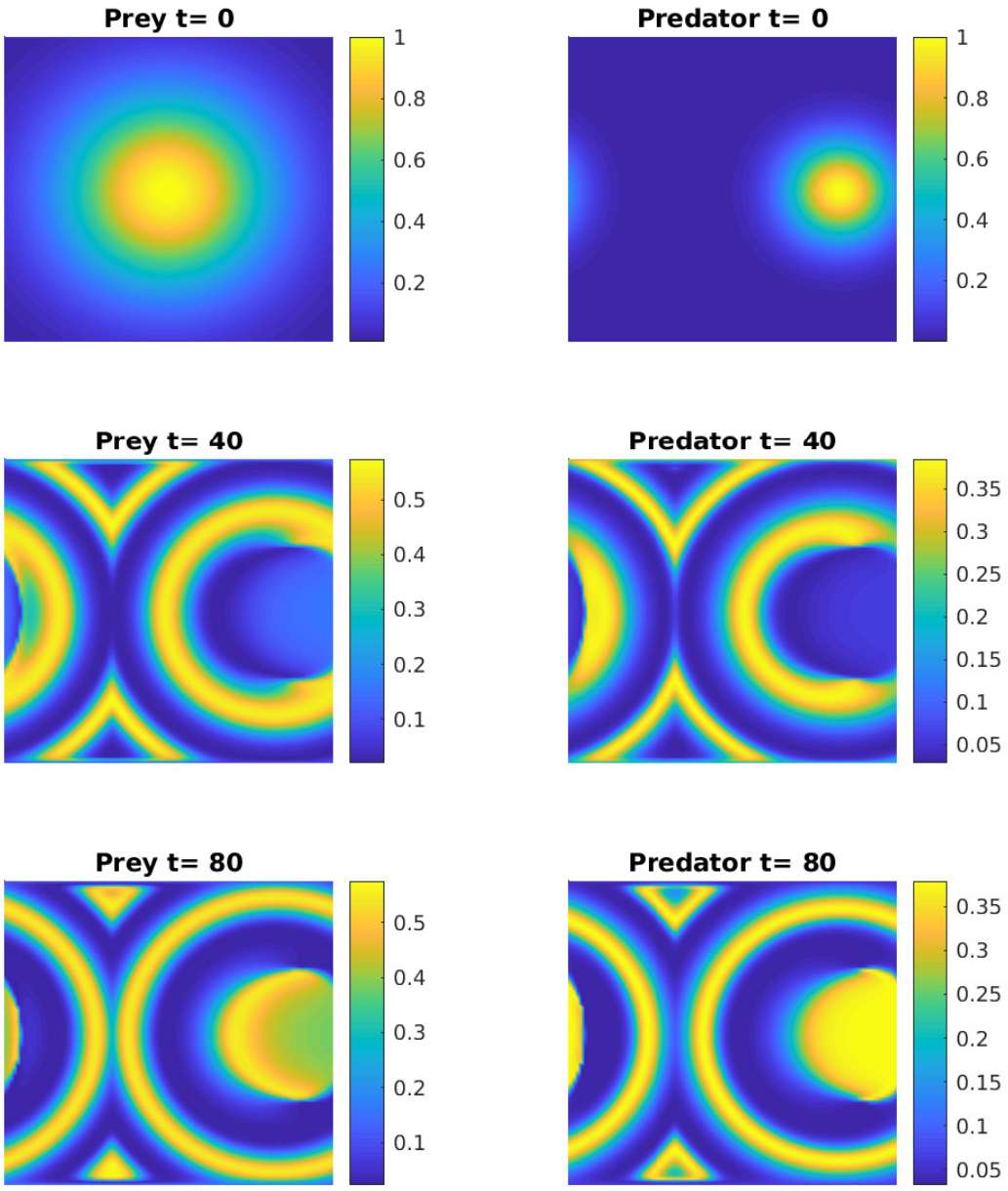


Figure 4.3: Simulation of two Gauss curves on the sphere with $\sigma_{prey} = 0.00001$, $\sigma_{predator} = 0.000001$

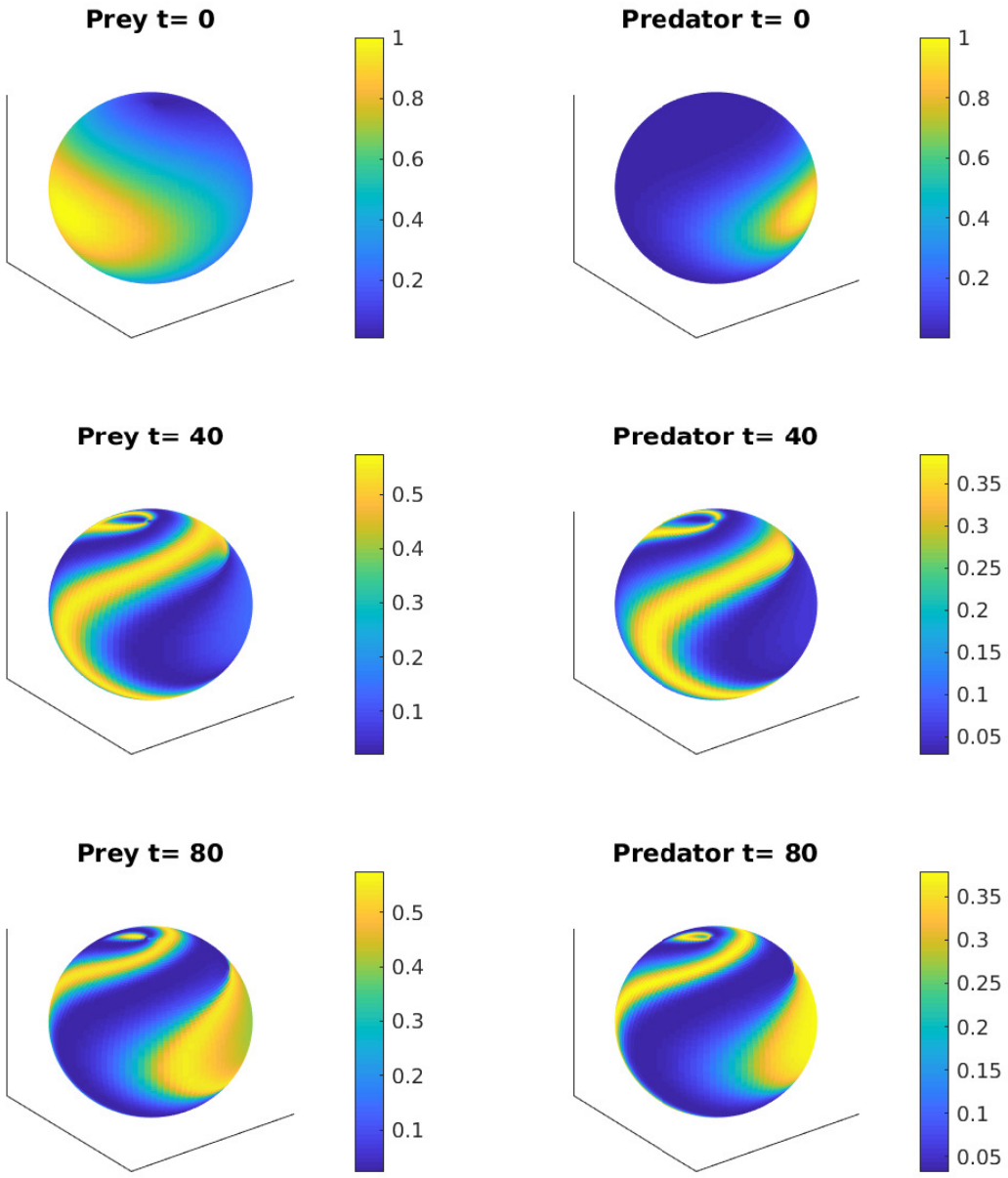


Figure 4.4: Simulation of two Gauss curves on the sphere with $\sigma_{prey} = 0.00001$, $\sigma_{predator} = 0.000001$ as a spherical plot

4.3 Conclusion and outlook

One could argue that choosing a sphere as spatial domain is a natural choice, considering that we and every form of life we know about lives on the spherical surface of the earth. Yet, the realisticness is still doubtful since no organism let alone a system of predators and prey could possibly diffuse around the entire globe due to for example oceans or landmasses and other environmental factors like different climate zones. An attempt could be made to incorporate effects like these by, for example taking the diffusivities σ dependant on space to model areas like oceans, where no migration is possible or areas like mountains where migration is aggravated. Also the parameters in the reaction term could be taken to depend on space to model different climate zones where animals reproduce differently or can not survive at all like for example deserts. With all these effects incorporated, the numerical effort to simulate such a model on an at least somewhat realistic map of the earth would be enormous. And even then is the real world much, much to complicated to be squeezed into a handful of equations. There is after all still an unresolvable conflict between realisticness and computability. Nevertheless the pictures generated above are very beautiful and the techniques could be used for some other equation on the sphere with more practical value.

Appendix

To help readers reproduce our results, we collected here some representative Matlab - codes for our simulations. Any plot we presented above can be generated by some straightforward modifications of the following code.

Chapter 1

Since we use Matlab's buit-in solver for ODE's, the codes for the entire chapter are really simple and can all be derived from the following example.

"codes/ODEs/simPredPreyReal.m"

```
% Numerically solves realistic predator-prey equations and
% displays population over time and phase plane plot

% solve differential equation numericallly
[t,solxy] = ode23(@predPreyRealEq,[0:0.01:100],[0.5; 0.5]);
figure('Position', [0 0 700 250])
newplot

% display population over time plot
subplot (1, 2, 1)
plot(t,solxy)
xlabel('Time')
ylabel('Population')
legend('Prey','Predators')
ylim([0 0.71])

% display phase plane plot
subplot (1, 2, 2)
plot(solxy(:,1),solxy(:,2))
xlabel('Prey Population');
ylabel('Predator Population');
hold on
ylim([0 0.55])

% add time labels to phase plane plot
start_time = 1;
step = round(length(solxy) / 5);
text(solxy(start_time:step:end,1), solxy(start_time:step:end,2),...
    num2str(t(start_time:step:end)));
scatter(solxy(start_time:step:end,1), solxy(start_time:step:end,2), ...
    5, "marker", 'o', "markerfacecolor", 'black', "markeredgecolor",...
    'black');
```

"codes/ODEs/predPreyRealEq.m"

```
% Realistic predator-prey equations
function dxdy = predPreyRealEq(t,xy)
```

```
a = 1;
b = 0.5;
d = 0.02;
dxdy=[((1-xy(1))-a*xy(2)/(xy(1)+d))*xy(1);(b*(1-xy(2)/xy(1)))*xy(2)];
end
```

Chapter 2

The simulation is implemented as described above. To visualize the results, we use Matlab's surf function for both plot cases, the planar plot and the torus plot.

"codes/Spatial/simulationWithDirichlet.m"

```
function simulationWithDirichlet()
    simulationLotkaGeneral(@computeDirichletInitial,...
    @computeDirichletBoundary);
end
```

"codes/Spatial/simulationWithNeumann.m"

```
function simulationWithNeumann()
    simulationLotkaGeneral(@computeNeumannInitial,...
    @computeNeumannBoundary);
end
```

"codes/Spatial/simulationWithTorus.m"

```
function simulationWithTorus()
    simulationLotkaGeneral(@computeTorusInitial,@computeTorusBoundary);
end
```

"codes/Spatial/simulationLotkaGeneral.m"

```
function simulationLotkaGeneral (computeInitial, computeBoundary)
    T = 50;                 % Simulation Time
    M = 1000;               % Number of Time steps
    N = 50;                 % Mesh Size
    L = 1;                  % Space Size
    h = L/(N-1);            % Spatial Step
    dt = T/M;               % Time Step
    sigma = 0.0001;         % Diffusion coefficient (for both species)
    plotNumber = 3;         % Number Of Plots

    % Generate Initial Condition
    u = zeros(N,N,2);
    uplus = zeros(N,N,2);
    uSavePlot = zeros(N,N,plotNumber,2);

    for i = 1:N
```

```
        for j = 1:N
            u(i,j,1) = 1/(2) * exp(-1/2 * (((i*h-0.3)/0.1)^2 +...
            ((j*h-0.3)/0.1)^2));
            u(i,j,2) = 1/(4) * exp(-1/2 * (((i*h-0.2)/0.1)^2 +...
            ((j*h-0.2)/0.1)^2));
        end
    end

    %Initial boundary condition
    u = computeInitial(u);

    uSavePlot(:,:,1,:) = u(:,:,:);

    % Time loop
    for k = 1:M
        % Loop along x axis
        for i = 2:N-1
            % Loop along y axis
            for j = 2:N-1
                Diff = squeeze((u(i+1,j,:)+u(i-1,j,:)-4*u(i,j,:)+...
                u(i,j+1,:)+u(i,j-1,:))/(h.^2));
                Reac = lotka2D(squeeze(u(i,j,:)));
                uplus(i,j,:)=squeeze(u(i,j,:))+dt*Reac+dt*sigma*Diff;
                if(isnan(uplus(i,j,1)) || isnan(uplus(i,j,2)))
                disp('error nan');
                return
            end

        end

    end
    % Computation of Boundary values
    uplus = computeBoundary (u, uplus, dt, sigma, h);

    u = uplus;

    for p = 1:plotNumber -1
        if k == floor(p*M/(plotNumber -1))
            uSavePlot(:,:,p+1,:) = u(:,:,:);
        end
    end

end

% Plot result
%plotGeneral(h, L, T, uSavePlot);
plotTorus(T, uSavePlot);

end
```

”codes/Spatial/computeDirichletInitial.m”

```
function u =  computeDirichletInitial(u)
    [N,m,n] = size(u);
    u(1,:,:) = zeros(1,m,n);
```

```
    u(N,:,:) = u(1,:,:);
    u(:,N,:) = u(1,:,:);
    u(:,1,:) = u(1,:,:);
end
```

"codes/Spatial/computeDirichletBoundary.m"

```
function uplus = computeDirichletBoundary(u, uplus, dt, sigma, h)
    uplus = computeDirichletInitial(uplus);
end
```

"codes/Spatial/computeNeumannInitial.m"

```
function u = computeNeumannInitial(u)
    [N,~,~] = size(u);

    u(N,:,:) = u(N-1,:,:);
    u(:,N,:) = u(:,N-1,:);
    u(1,:,:) = u(2,:,:);
    u(:,1,:) = u(:,2,:);
end
```

"codes/Spatial/computeNeumannBoundary.m"

```
function uplus = computeNeumannBoundary(u, uplus, dt, sigma, h)
    uplus = computeNeumannInitial(uplus);
end
```

"codes/Spatial/computeTorusInitial.m"

```
function u = computeTorusInitial(u)
    [N,~,~] = size(u);
    for j = 2:N-1
        % Convolution around y axis
        midval = 0.5 * (u(2,j,1)+u(N-1,j,1));
        u(1,j,1) = midval;
        u(N,j,1) = midval;
        midval = 0.5 * (u(2,j,2)+u(N-1,j,2));
        u(1,j,2) = midval;
        u(N,j,2) = midval;

        % Convolution around x axis
        midval = 0.5 * (u(j,2,1)+u(j,N-1,1));
        u(j,1,1) = midval;
        u(j,N,1) = midval;
        midval = 0.5 * (u(j,2,2)+u(j,N-1,2));
        u(j,1,2) = midval;
        u(j,N,2) = midval;
    end
    % Corner computation
    u(1,1,:) = 1/8 * (u(2,1,:)+u(N-1,1,:)+u(1,2,:)+u(1,N-1,:)+...
```

```
    u(2,N,:)+u(N-1,N,:)+u(N,N-1,:)+u(N,2,:));
    u(1,N,:) = u(1,1,:);
    u(N,1,:) = u(1,1,:);
    u(N,N,:) = u(1,1,:);
end
```

"codes/Spatial/computeTorusBoundary.m"

```
function uplus = computeTorusBoundary(u, uplus, dt, sigma, h)
    [N,~,~] = size(u);
    % Torus boundary computation
    i = 1;
    for j = 2:N-1
        uplus = computeUPlus(uplus,i,j);
    end
    j = 1;
    for i = 2:N-1
        uplus = computeUPlus(uplus,j,i);
    end
    % Corner computation
    i = 1;
    j = 1;
    Diff = squeeze((u(i+1,j,:)+u(N-1,j,:)-4*u(i,j,:)+u(i,j+1,:)+...
    u(i,N-1,:))/(h.^2));
    Reac = lotka2D(squeeze(u(i,j,:)));
    uplus(i,j,:) = squeeze(u(i,j,:)) + dt * Reac + dt * sigma * Diff;

    uplus(N,:,:) = uplus(1,:,:);
    uplus(:,N,:) = uplus(:,1,:);

    uplus(1,N,:) = uplus(1,1,:);
    uplus(N,1,:) = uplus(1,1,:);
    uplus(N,N,:) = uplus(1,1,:);

    function [uplus] = computeUPlus(uplus,i,j)
        Diff = squeeze((u(i+1,j,:)+u(N-1,j,:)-4*u(i,j,:)+...
        u(i,j+1,:)+u(i,j-1,:))/(h.^2));
        Reac = lotka2D(squeeze(u(i,j,:)));
        uplus(i,j,:) = squeeze(u(i,j,:))+dt*Reac+dt*sigma * Diff;
    end
end
```

"codes/Spatial/plotGeneral.m"

```
function plotGeneral(h, L, T, uSavePlot)
    [X Y] = meshgrid(0:h:L,0:h:L);
    [~,~,plotNumber,~] = size(uSavePlot);
    newplot
    for species = 1:2
        for i = 1:plotNumber
            subplot(plotNumber,2,2*(i-1)+species);
            hold on;
            surf(X,Y,uSavePlot(:,:,i,species),'FaceColor','interp');
            shading interp
```

```
            hold off;
            c = colorbar;
            axis([0 L 0 L 0 1]);
            axis square;
            axis off;

            t = (i-1)*T/(plotNumber-1);
            if (species == 1)
                title(['\fontsize{6}Prey t = ', num2str(t)])
                else
                    title(['\fontsize{6}Predator t = ', num2str(t)])
                end
                set(c,'FontSize',5);

                view(2);
            end
        end
end
```

"codes/Spatial/plotTorus.m"

```
function plotTorus(T, uSavePlot)

    aminor = 1.; % Torus minor radius
    Rmajor = 3.; % Torus major radius
    [s1,s2,plotNumber,~] = size(uSavePlot);

    theta  = linspace(-pi, pi, s1)   ; % Poloidal angle
    phi    = linspace(0., 2.*pi, s2) ; % Toroidal angle

    [t, p] = meshgrid(phi, theta);

    X = (Rmajor + aminor.*cos(p)) .* cos(t);
    Y = (Rmajor + aminor.*cos(p)) .* sin(t);
    Z = aminor.*sin(p);

    newplot
    for species = 1:2
        for i = 1:plotNumber
            subplot(plotNumber,2,2*(i-1)+species);
            hold on;
            surf(X,Y,Z,uSavePlot(:,:,i,species)', 'LineStyle','none');
            hold off;
            c = colorbar;
            axis square;

            axis off;

            t = (i-1)*T/(plotNumber-1);
            if (species == 1)
                title(['\fontsize{6}Prey t = ', num2str(t)])
                else
                    title(['\fontsize{6}Predator t = ', num2str(t)])
                end
                set(c,'FontSize',5);
```

```
                view(2);

            end
        end
end
```

Chapter 3

The code for convection is similar to the codes used in chapter 2 with an added convection term. The convection force field is given as a function, allowing for easy switching between different fields. For display, we additionally plot the convection vector field using Matlab's `quiver3` plot.

"codes/Convection/main.m"

```
% Parameters
plot_number = 3; % Number of Plots

% Isotropic Convection
% M = 20000;     % Number of Time Steps
% T = 1.1;       % Simulation Time
% N = 100;       % Mesh Size
% L = 1;         % Space size
% sigma = [0.00001;0.01]; % Diffusion Speeds [Prey, Predator]
% c = [1,0.01]; % Convection speed [Prey, Predator]
% %Initial Condition parameters
% preyCenter = [round(N/2)+10, round(N/2)];
% preySdtDev = 0.8;
% preyM = 0.45;
% predCenter = [round(N/2)+5, round(N/2)];
% predSdtDev = 0.2;
% predM = 1;
% convectionField = @isotropicConvection;

%Circular Convection
M = 1000;      % Number of Time Steps
T = 1.1;       % Simulation Time
N = 30;        % Mesh Size
L = 1;         % Space size
sigma = [0.00001;0.02]; % Diffusion Speeds [Prey, Predator]
c = [1,0.1]; % Convection speed [Prey, Predator]
%Initial Condition parameters
preyCenter = [round(N/2)+10, round(N/2)];
preySdtDev = 0.8;
preyM = 0.45;
predCenter = [round(N/2)+15, round(N/2)];
predSdtDev = 0.2;
predM = 1;
convectionField = @circularConvection;

% Generate Initial Condition
h = L/(N-1);
u = zeros(N, N, 2);
```

```
for i = 2:N-1
    for j = 2:N-1
        u(i,j,1)=preyM*exp(-(power(norm(h*([i,j]-preyCenter))/...
            preySdtDev,2) ));
        u(i,j,2)=predM*exp(-(power(norm(h*([i,j]-predCenter))/...
            predSdtDev,2) ));
    end
end
% Neumann boundary condition for initial condition
u(:,1,:) = u(:,2,:);
u(:,N,:) = u(:,N-1,:);
u(1,:,:) = u(2,:,:);
u(N,:,:) = u(N-1,:,:);
% Initial Condition done

plots = simConvection(u,T,M,L,N,convectionField,c,sigma,plot_number);

planeConvectionPlot(plots, T,L, convectionField);
```

"codes/Convection/simConvection.m"

```
function plots = simConvection(u, T, M,L,...
    N,convectionField,c, sigma, plot_number)

dt = T / M;
h = L/(N);

un = zeros(N, N, 2);
uo = u;

plots = zeros(N,N,2,plot_number);
plots(:,:,:,1) = u;

for k = 1:M
    for i = 2:N-1
        for j = 2:N-1
            Diff = squeeze((uo(i+1, j,:) + uo(i-1, j,:) -...
                4*uo(i,j,:) + uo(i, j+1,:) + uo(i,j-1,:))/(h*h));
            Reac = lotka2D(squeeze(uo(i,j,:)));

            phi = convectionField(i,j, L , N);

            % for phi(1) < 0 take (u(i+1,j,:) - u(i,j,:))*(1/h) and
            % for phi(1) > 0 take (u(i,j,:) - u(i-1,j,:))*(1/h)
            Conv =  max(0,phi(1)) * (u(i,j,:) - u(i-1,j,:))*(1/h)...
                + min(0,phi(1)) * (u(i+1,j,:) - u(i,j,:))*(1/h) + ...
                max(0,phi(2)) * (u(i,j,:) - u(i,j-1,:))*(1/h)...
                + min(0,phi(2)) * (u(i,j+1,:) - u(i,j,:))*(1/h);
            for g = 1:2
                un(i,j,g) = uo(i,j,g) + dt * Reac(g) +...
                    dt* sigma(g) * Diff(g)  - dt*c(g)*Conv(g);
                if(isnan(un(i,j,g)) || un(i,j,g) <0)
                    fprintf(strcat("Error at k = ", num2str(k),...
                        " / t = ", num2str(k*dt), "\n"));
                    return;
```

```
                end
            end
        end
    end

    %Neumann-Randbedingung
    un(:,1,:) = un(:,2,:);
    un(:,N,:) = un(:,N-1,:);
    un(1,:,:) = un(2,:,:);
    un(N,:,:) = un(N-1,:,:);

    uo = un;
    for p = 1:plot_number-1
        if (k == floor(p*M/(plot_number-1)))
            plots(:,:,:,p+1) = un;
        end
    end
end
end
```

"codes/Convection/lotka2D.m"

```
function dxdy = lotka2D(xy)
a = 1;
b = 0.5;
d = 0.02;
dxdy = [((1-xy(1))-a*xy(2)/(xy(1)+d))*xy(1);(b*(1-xy(2)/xy(1)))*xy(2)];
end
```

"codes/Convection/circularConvection.m"

```
function phi = circularConvection(i,j, L, N)
    h = L/N;
    phi = [ -(j*h - L/2); (i*h - L/2)];
end
```

"codes/Convection/isotropicConvection.m"

```
function phi = isotropicConvection(i,j, L, N)
    phi = [-1,0];
end
```

"codes/Convection/planeConvectionPlot.m"

```
function planeConvectionPlot(plots, T,L , convectionField)

S = size(plots);
N = S(1);
plot_number = S(4);

h = L/(N-1);
```

```
[X,Y] = meshgrid( h*(0:1:N-1),h*(0:1:N-1));
maxprey  = max(max(max(max(plots(:,:,1, :)))),1e-99)*1.1;
minprey  = min(min(min(min(plots(:,:,1, :)))),0)*0.9;
maxpred  = max(max(max(max(plots(:,:,2, :)))),1e-99)*1.1;
minpred  = min(min(min(min(plots(:,:,2, :)))),0)*0.9;

extrema = [minprey, maxprey, minpred,maxpred];

figure('Position', [0 0 750 150+225*plot_number])
hold off

K = 5; % Take every Kth grid point to draw convection speed arrow
% save convection directions as matrices
NK = floor(N/K);
xConvDirs = zeros(NK,NK);
yConvDirs = zeros(NK,NK);
for a=1:NK
    for b=1:NK
        phi = convectionField(a*K,b*K, L, N);
        xConvDirs(a,b) = phi(2);
        yConvDirs(a,b) = phi(1);
    end
end

for species = 1:2
    for j = 1:plot_number
        % species horizontal
        % subplot(2,plot_number, j+(plot_number*(species-1)))
        % species vertical
        subplot(plot_number,2, 2*(j-1)+species)
        hold on
        surf(X,Y,plots(:,:,species,j), "FaceColor", "interp");
        shading interp
        % plot convection vector field
        quiver3(X(1:K:N-1,1:K:N-1)*NK/(NK-1),Y(1:K:N-1,1:K:N-1)...
            *NK/(NK-1),ones(NK,NK)*extrema(2 + 2*(species-1))+...
            ones(NK,NK),0.9*xConvDirs,0.9*yConvDirs,zeros(NK,NK),...
            "Color", "white");
        hold off
        view(2)
        if(species == 1)
            spName = "Prey t= ";
        elseif(species ==2)
            spName = "Predator t= ";
        end
        title(strcat(spName, num2str((j-1)*T/(plot_number-1))));
        axis([0,L,0,L,extrema(1 + 2*(species-1)),...
            extrema(2 + 2*(species-1))])
        ax = gca;
        ax.Clipping = 'off';
        xticks({})
        yticks({})
        %caxis manual;
        %caxis([0 extrema(2 + 2*(species-1))]);
        colorbar
        view(2)
```

```
    end
end


end
```

Chapter 4

The basic structure for the simulation on the sphere is the same as before. However the computation of the diffusion is more complicated and we need to handle the computation for the zero-meridian and the poles separately in every step. For the visualization we implement both the planar plot as well as the plot on the sphere.

"codes/Sphere/main.m"

```
% Parameters
plot_number = 3; % Number of Plots
M = 5000;     % Number of Time Steps
T = 50;       % Simulation Time
N = 50;       % Mesh Size
N1 = N;       % X = Longitude Mesh Size
N2 = N;       % Y = Latitude Mesh Size
r = 1;        % Radius
sigma = [0.01;0.01]; % Diffusion Speeds [Prey, Predator]
%Initial condition parameters
preyCenter = [round(N1/2),round(N2/2)];
preySdtDev = 1;
preyM = 1;
predCenter = [round(5*N1/6),round(N2/2)];
predSdtDev = 0.5;
predM = 1;

% Generate Initial Condition
u = zeros(N1, N2, 2);
dth = pi/N2;
dph = 2*pi/N1;
% rotate globe to get nice initial condition if the gauss curves
% are close to the zero-meridian
for i = 1:N1
    for j = 1:N2
        prey = 0;
        pred = 0;
        for k = -1:1
            preyCandidate = preyM*exp(-( power(norm(dth* ...
                ([i+(N1-1)*k, j]-preyCenter)  ) / preySdtDev,2) ));
            if(preyCandidate > prey)
                prey = preyCandidate;
            end
            predCandidate = predM*exp(-( power(norm(dth* ...
                ([i+(N1-1)*k, j]-predCenter)  ) / predSdtDev,2) ));
            if(predCandidate > pred)
                pred = predCandidate;
            end
```

```
        end
        u(i,j,1) = prey;
        u(i,j,2) = pred;
    end
end

% Take mean on zero-meridian so that the data satisfies
% the sphere boundary condition
% Donut for Meridian
for q = 1:N2
    Zmean = 0.5*(u(2,q,:) + u(N1-1,q,:));
    u(1,q,:) = Zmean;
    u(N1,q,:) = Zmean;
end
%Set poles to mean of adjacent longitudes
for q = 1:N1
    u(q, N2,:) = squeeze(mean(u(1:N1-1,N2-1,:)));
    u(q, 1,:) = squeeze(mean(u(1:N1-1,2,:)));
end
% Initial Condition done

plots = simSpherical(u, T, M, N1, N2, r,sigma,  plot_number);

planePlot(plots, T);
spherePlot(plots, T,r);
```

"codes/Sphere/simSpherical.m"

```
function plots = simSpherical(u, T, M, N1, N2, r, sigma, plot_number)

dt = T / M;
dth = pi/N2;
dph = 2*pi/N1;

un = zeros(N1, N2, 2);
uo = u;

plots = zeros(N1,N2,2,plot_number);
plots(:,:,:,1) = u;

for k = 1:M
    for i = 1:N1-1
        for j = 2:N2-1
            if(i == 1)
                e = squeeze(uo(N1-1, j,:));
            else
                e = squeeze(uo(i-1, j,:));
            end
            Diff = CalcDiff((j-1)*dth, (i-1)*dph, squeeze(uo(i, j,:)),
                squeeze(uo(i, j+1,:)),squeeze(uo(i, j-1,:)), squeeze(uo
                (i+1, j,:)), e , dth, dph, r);
            Reac = lotka2D(squeeze(uo(i,j,:)));
            for g = 1:2
                un(i,j,g) = uo(i,j,g) + dt * Reac(g) + dt* sigma(g) *
                    Diff(g);
```

```
                % Set predators to 0 if preys very small
                %if uo(i,j,1) < 1e-10
                %    un(i,j,2) = 0;
                %end

                if(isnan(un(i,j,g)) || isinf(un(i,j,g)))
                    fprintf(strcat("Error at k = ", num2str(k), " / t =
                        ", num2str(k*dt), "\n"));
                    return;
                end
            end
        end
    end

    % mirror null-meridian in the east
    un(N1, :,:) = un(1,:,:);

    % Calculate Poles start
    % North pole:
    nDiff = squeeze(mean(uo(1:N1-1,N2-1,:)));
    nReac = lotka2D(squeeze(uo(1,N2,:)));
    for g = 1:2
        for q = 1:N1
            un(q, N2,g) = uo(1,N2,g) + dt*nReac(g) + dt*(1/(pi*(r*dth)
                ^2))*sigma(g)*nDiff(g);
        end
    end
    % South Pole:
    sDiff = squeeze(mean(un(1:N1-1,2,:)));
    sReac = lotka2D(squeeze(uo(1,1,:)));
    for g = 1:2
        for q = 1:N1
            un(q, 1,g) = uo(1,1,g) + dt*sReac(g) + dt*(1/(pi*(r*dth)^2)
                )*sigma(g)*sDiff(g);
        end
    end
    % Calc Poles done

    uo = un;
    for p = 1:plot_number-1
        if (k == floor(p*M/(plot_number-1)))
            plots(:,:,:,p+1) = un;
        end
    end
end
end

function d = CalcDiff(th, ph, prev, n,s,e,w, dth, dph,r)
d = (n*sin(th + dth/2) - prev*(sin(th+0.5*dth)+sin(th-0.5*dth)) + ...
    s * sin(th-0.5*dth))/(r^2*sin(th)*dth^2) + 1/(r^2 * sin(th)^2)*(e -
        2*prev + w)/(dph^2);
end
```

"codes/Sphere/lotka2D.m"

```
function dxdy = lotka2D(xy)
a = 1;
b = 0.5;
d = 0.02;
dxdy = [ ((1-xy(1)) - a*xy(2)/(xy(1)+d)) * xy(1); (b*(1- xy(2)/(xy(1)))
    )*xy(2) ];
end
```

"codes/Sphere/planePlot.m"

```
function planePlot(plots, T )

S = size(plots);
N1 = S(1);
N2 = S(2);
plot_number = S(4);

dth = pi/N2;
dph = 2*pi/N1;

[X,Y]=meshgrid((N1/(N1-1))*dth*(0:1:N1-1),(N2/(N2-1))*dph*(0:1:N2-1));
maxprey  = max(max(max(max(plots(:,:,1, :)))),1e-99)*1.1;
minprey  = min(min(min(min(plots(:,:,1, :)))),0)*0.9;
maxpred  = max(max(max(max(plots(:,:,2, :)))),1e-99)*1.1;
minpred  = min(min(min(min(plots(:,:,2, :)))),0)*0.9;

extrema = [minprey, maxprey, minpred,maxpred];

figure('Position', [0 0 750 150+225*plot_number])
hold off

for species = 1:2
    for j = 1:plot_number
        % species horizontal
        % subplot(2,plot_number, j+(plot_number*(species-1)))
        %s
        subplot(plot_number,2, 2*(j-1)+species) % species vertical
        surf(Y,X,plots(:,:,species,j), "FaceColor", "interp");
        shading interp
        view(2)
        if(species == 1)
            spName = "Prey t= ";
        elseif(species ==2)
            spName = "Predator t= ";
        end
        title(strcat(spName, num2str((j-1)*T/(plot_number-1))));
        axis([0,2*pi,0,pi,extrema(1 + 2*(species-1)),...
            extrema(2 + 2*(species-1))])
        ax = gca;
        ax.Clipping = 'off';
        %caxis manual;
        %caxis([0 extrema(2 + 2*(species-1))]);
        xticks({})
        yticks({})
        colorbar
```

```
        view(2)
    end
end
end
```

"codes/Sphere/spherePlot.m"

```
function spherePlot(plots, T, r)

S = size(plots);
N1 = S(1);
N2 = S(2);
plot_number = S(4);

dth = pi/N2;
dph = 2*pi/(N1-1);

phs = (0:1:N1-1)*dph;
ths = (0:1:N2)*dth;
[ph, th] = meshgrid(phs, ths);
X = r.*sin(th).*cos(ph);
Y = r.*sin(th).*sin(ph);
Z = r*cos(th);

maxprey  = max(max(max(max(plots(:,:,1, plot_number)))),1e-99)*1.1;
minprey  = min(min(min(min(plots(:,:,1, plot_number)))),0)*0.9;
maxpred  = max(max(max(max(plots(:,:,2, plot_number)))),1e-99)*1.1;
minpred  = min(min(min(min(plots(:,:,2, plot_number)))),0)*0.9;

extrema = [minprey, maxprey, minpred,maxpred];

figure('Position', [0 0 750 150+225*plot_number])
hold off

for species = 1:2
    for j = 1:plot_number
        % species horizontal
        % subplot(2,plot_number, j+(plot_number*(species-1)))
        subplot(plot_number,2, 2*(j-1)+species) % species vertical
        surf(X,Y,Z,plots(2:end,:,species,j)', "LineStyle", "none");
        if(species == 1)
            spName = "Prey t= ";
        elseif(species ==2)
            spName = "Predator t= ";
        end
        title(strcat(spName, num2str((j-1)*T/(plot_number-1))));
        ax = gca;
        ax.Clipping = 'off';
        %caxis manual;
        %caxis([0 extrema(2 + 2*(species-1))]);
        xticks({})
        yticks({})
        zticks({})
        colorbar
        grid off
```

```
    end
end
end
```

List of Figures

Bibliography

[Bar89] Barros, Saolo R. M.: Multigrid methods for Two- and Three- Dimensional Poisson-Type Equations on the Sphere. In: *Journal of Computational Physics* (1989), 10, Nr. 92, S. 313–348

[Mur04a] Murray, James D.: *Mathematical biology /1: An introduction.* 3rd Edition. New York : Springer, 2004

[Mur04b] Murray, James D.: *Mathematical biology /2: Spatial models and biomedical applications.* 3rd Edition. New York : Springer, 2004