Project – Prey-Predator dynamics

Applied Finite Element Methods

Author: Amanda Seger
amanda.seger.9729@student.uu.se
2021-01-08

Contents

1 Introduction

The aim of this project is to study the Prey-Predator dynamics, modeled by partial differential equations. The system of interest is defined as following, (1)- (4), and are called the reaction-diffusion partial differential equations, where $u(\mathbf{x}, t)$ denotes the population density of prey (for example rabbits) and $v(\mathbf{x}, t)$ as the population density of predator (such as foxes).

$$\partial_t u - u(1-u) + \frac{uv}{u+\alpha} - \delta_1 \Delta u = f, (\mathbf{x}, t) \in \Omega x(0, t] \tag{1}$$

$$\partial_t v + \gamma v - \beta \frac{uv}{u+\alpha} - \delta_2 \Delta v = g, (\mathbf{x}, t) \in \Omega x(0, t] \tag{2}$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}),\ \mathbf{x} \in \Omega \tag{3}$$

$$v(\mathbf{x}, 0) = v_0(\mathbf{x}),\ \mathbf{x} \in \Omega \tag{4}$$

where $\Omega \in \mathbb{R}^d, d = 1, 2, 3$ is a bounded domain, $T > 0$ is the final time, $f(\mathbf{x}, t)$ and $g(\mathbf{x}, t)$ are given source terms, and u_0 and v_0 are given initial data. The terms in the prey equation (1) denote: change in time, prey birth and death rates, the prey consumption rate per predator as the fraction of the maximal consumption rate 1 (reaction term), and population diffusion. In the same way, the terms in the predator equation (2) are denoted by: change in time, predator death rate, the prey consumption rate per predator (reaction term), and population diffusion. The parameters α, β, γ and δ are strictly positive, describing the interaction of the two species.

The model describes the dynamics and the relationship between preys and predators. If the amount of preys increases, the amount of predators increases since they obtain more food. Further, when the amount of predators increases, the prey population decreases which leads to decreasing in the predator population, due to lack of food. And so it continues.
The population change can be estimated by computing the total rate in the computation domain, as following:

$$M_{prey}(t) = \int_\Omega u(\mathbf{x}, t) d\mathbf{x},\ M_{predator}(t) = \int_\Omega v(\mathbf{x}, t) d\mathbf{x} \tag{5}$$

The project is divided into three parts. Part A examines a simpler one-dimensional stationary diffusion problem of (1)-(4) and part B a two-dimensional time-dependent reaction-diffusion equation. In part C, finally the system (1)-(4) is being solved, using the open-source software FEniCS Project.

2 Part A

$$-\delta u''(x) = f(x), x \in (-1,1), \tag{6}$$

$$u(-1) = u(1) = 0 \tag{7}$$

Obtain the weak formulation by multiplying equation (6) with a test function, v and then integrate by parts.

$$-\delta \int_a^b u''v dx = -\delta \left[u'v\right]_a^b + \delta + \delta \int_a^b u'v' dx = \int_a^b fv dx$$

$$= -\delta(u'(b)v(b) - u'(a)v(a)) \int_a^b u'v' dx = \int_a^b fv dx$$

$V_0 = \{v : \int_a^b v^2 dx < \infty, \int_a^b (v')^2 dx < \infty, v(a) = v(b) = 0\}$
Find $u \in V_0$ such that $= \int_a^b u'v' dx = \int_a^b fv dx, \forall v \in V_{h,0}$.

Define the subinterval $I_i = (x_{i-1}, x_i)$. Construct the space V_h of all continuous piecewise linear functions as follow
$V_h = \{v : v \in C^0(I), v \in P^1(I_i, I_i = (x_{i-1}, xi), i = 1, ..., n, v(a) = v(b) = 0\}$

Find $u_h \in V_h$ such that $\int_a^b u_h' v' dx = \int_a^b fv dx, \forall v \in V_h$

$$\int_a^b u_h' v' dx = \int_a^b fv dx, \forall v \in V_h \tag{8}$$

Let the hat functions, $\{\varphi_i\}_{j=1}^{n-1}$ be the basis functions of V_h.

$$\varphi_i(x_j) = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases}$$

Inserting ansatz $u_h = \sum_{j=1}^{n-1} \xi_j \varphi_j$, and $v = \varphi_i$ into (8) yields

$$\delta \sum_{j=1}^{n-1} \xi_j \int_a^b \varphi_j' \varphi_i' dx = \int_a^b f \phi_i dx$$

With $A_{i,j} = \int_a^b \varphi_j' \varphi_i' dx, B = \int_a^b f \varphi_i dx$, we the following linear system

$$\delta A \xi = \mathbf{B} \tag{9}$$

$$\xi = \delta A \backslash \mathbf{B}$$

Denote the Laplacian as Δ_h and the L_2 projection on V_h P_h. Then the discrete Laplacian is obtained by projecting the Laplacian onto V_h, as $\Delta_h u_h = P_h \Delta u_h$.
For given $u_h \in V_h$ find $\Delta_h u_h \in V_h$ such that

$$\int_a^b \Delta_h u_h v dx = -\int_a^b u_h' v' dx \tag{10}$$

Inserting the ansatz $\Delta_h u_h = \sum_{j=1}^{N-1} \xi_j \varphi_j$ into (10), with M as the mass matrix $M_{i,j} = \int_a^b \varphi_(j) \varphi_i dx$ gives us the following linear system

$$M\xi_{Lap} = -A\xi \tag{11}$$

$$\xi_{Lap} = -M\backslash A\xi \tag{12}$$

2.1 Problem 1

Let u_h be a finite element approximation of the solution of the problem (6)-(7), with mesh $-1 = x_0 < x_1 < ... < x_N = 1$, mesh size $h_i = x_i - x_{i-1}$ and $I_i = (x_{i-1}, x_i)$ as the i-th element.
"Derive the following a posteriori error estimate in the energy norm:"

$$\|(u - u_h)'\|^2_{L^2(I)} \leq C \sum_{i=1}^{n} \eta_i^2 \tag{13}$$

where C is a constant and η is given by following expression

$$\eta_i = h_i \|f + \delta u_h''\|_{L^2(I_i)} = h_i \|R\| \tag{14}$$

where R is denoted as the residual.
Denote $e = u - u_h$, and the left hand side in (13) becomes:

$$\|e'\|^2_{L^2(I)} =$$

$$= \int_I e'e'dx$$

$$= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} e'(e - \pi_h e)'dx$$

With Integration by parts we obtain following equation. The last term becomes zero, since the interpolation error is zero in the nodes, i.e. $e(x_i) = \pi e(x_i)$ and $e(x_{i-1}) = \pi e(x_{i-1})$.

$$= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} -e''(e - \pi_h e)dx + [e'(e - \pi_h e)]_{x=x_{i-1}}^{x=x_i}$$

Setting e back to $e = u - u_h$ and inserting it into the equation above gives the following

$$= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} (-u'' + u_h'')(e - \pi_h e)dx$$

From equation (6), u'' can be expressed as $-u'' = \frac{f(x)}{\delta}$. Using the Cauchy-Schwarz inequality gives

$$= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} \frac{1}{\delta}(f + \delta u_h'')(e - \pi_h e)dx \leq \sum_{i=1}^{n} \frac{1}{\delta} \|f + \delta u_h''\|_{L^2(x_{i-1}, x_i)} \|e - \pi_h e\|_{L^2(x_{i-1}, x_i)} \tag{15}$$

Using standard interpolation estimate we obtain the following

$$\|e - \pi_h e\|_{L^2(x_{i-1}, x_i)} \leq C h_i \|e'\|^2_{L^2(x_{i-1}, x_i)}$$

Inserting the standard interpolation estimate into expression (15) and then insert δ into C, since δ and C is constants, gives the following

$$\|(u - u_h)'\|^2_{L^2(I_i)} \leq \sum_{i=1}^{n} \frac{1}{\delta} C h_i \|f + \delta u_h''\|_{L^2(I_i)} \|(u - u_h)'\|_{L^2(I_i)}$$

$$\leq C\sqrt{\sum_{i=1}^{n} h_i^2 \|f + \delta u_h''\|_{L^2(I_i)}^2}\sqrt{\sum_{i=1}^{n} \|(u-u_h)'\|_{L^2(I_i)}^2}$$

Inserting (14) gives the final expression

$$\|(u-u_h)'\|_{L^2(I_i)}^2 \leq C\sum_{i=1}^{n} h_i^2 \|f + \delta u_h''\|_{L^2(I_i)}^2 = C\sum_{i=1}^{n} \eta_i^2$$

2.2 Problem 2

In this part, (6)-(7) was solved by implementing an adaptive finite element approximation, consisting of the following steps:

1. Given a coarse mesh with n elements and a small number TOL > 0
2. while $\sum_{i=1}^{n} \eta_i > TOL$
3. compute u_h
4. compute $\eta_i^2(u_h)$ in each element $I_i, i = 1, 2, ..., n$
5. refine the elements with biggest contribution to the error.
6. end while

The starting uniform grid was 12 and the tolerance was set to $TOL = 10^{-3}$. The elements that was refined, where those who satisfied $\eta_i > \lambda \max \eta_i, i = 1, 2, ..., n$, where $0 \leq \lambda \leq 1$, was chosen to $\lambda = 0.9$

Figure 1 illustrates the results for part A. Firstly, the solution for the differential equation (6) - (7), ξ is shown and was obtained by the linear system given in equation (9).

The second image in 1 represents the error indicator, η and was obtained by using the definition of the L^2 norm in equation (14), and the integral was then implemented in Matlab using the Trapezoidal rule.

The third image illustrates the Mesh-size distribution and was obtained by plotting [1./diff(x)], where x is the coordinates of the final mesh.

The Residuals is shown in the fourth image and was obtained by (14).

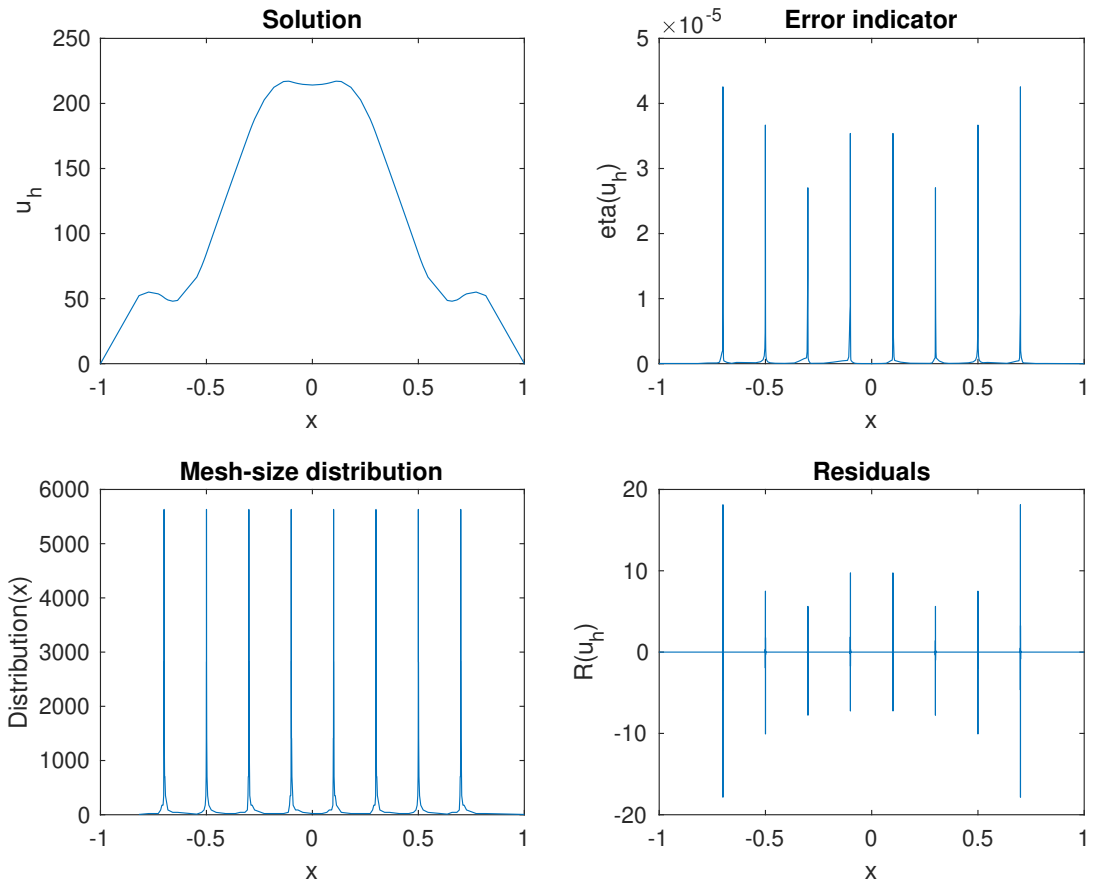


Figure 1: The solution, error indicator, mesh-size distribution and the residual, as functions of nodes x.

3 Part B

In this part, a scalar version of the model (1)-(2) is considered, in two space space dimensions. Part B consists of two problems, the first examines the prey-equation (1) without the non-linear terms and the time-derivative. The second analyzes the prey-equation (1) with a constant predator population.
The computational domain, the disc Ω, for part B1 and part B2 is implemented by defining the geomtery, using the build-in Matlab function *circleg*, and triangulate it with respect to mesh-size h_{max}, which will be defined for each problem. In the first problem, a convergence study is performed, by first computing the energy norm.

3.1 Problem 1

In the first problem of part B, a simplified equation of the prey-equation (1), is being considered by omitting the non-linear terms and the time-derivative:

$$-\Delta u(\mathbf{x}) = f(\mathbf{x}),\ \mathbf{x} \in \Omega \tag{16}$$

$$u(\mathbf{x}) = u_{exact}(\mathbf{x}), \mathbf{x} \in \partial\Omega \tag{17}$$

where $f(\mathbf{x}) = 8\pi^2 \sin(2\pi x_1)\sin(2\pi x_2)$ and the exact solution $u_{exact}(\mathbf{x}) = \sin(2\pi x_1)\sin(2\pi x_2)$

Constructing the trial space, $V_g = \{v : \|v\|^2 + \|\nabla v\|^2 < \infty, v = u_{exact}\ on\ \partial\Omega\}$. Find $u \in V_g$ such that

$$\int_\Omega \nabla u \nabla v d\mathbf{x} = \int_\Omega f v d\mathbf{x}, \forall v \in V_0$$

where $V_0 = \{v : \|v\|^2 + \|\nabla v\|^2 < \infty, v = 0\ \text{ on }\ \partial\Omega\}$, is the test space.

Consider a triangulation of $\Omega : \tau = \{K\}$ and construct the following finite dimensional subspaces

$$V_{h,g} \subset V_g, V_{h,g} = \{v : v \in C^0(\tau_h); v|_K \in P_1(K), K \in \tau_h, v = u_{exact}\ on\ \partial\Omega\}$$
$$V_{h,0} \subset V_0, V_{h,0} = \{v : v \in C^0(\tau_h); v|_K \in P_1(K), K \in \tau_h, v = 0\ on\ \partial\Omega\}$$

Now, find $u_h \in V_{g,h}$, according to the Galerkin finite element method (GFEM) as following

$$\int_\Omega \nabla u_h \nabla v d\mathbf{x} = \int_\Omega f v d\mathbf{x}, \forall v \in V_{h,0}$$

Dividing the nodal points on the mesh into two sets; $\{N_h\}$, which is the set of all interior nodes and $\{N_b\}$, the set of all boundary nodes, as $\{N\} = \{N_h\} + \{N_b\}$

Since $u_h \in V_{h,g}$ there exists a set $\{\xi_j\}, N_j \in \{N\}$, such that

$$u_h(\mathbf{x}) = \sum_{N_j \in N} \xi_j \varphi_j(\mathbf{x}) = \sum_{N_j \in N_h} \xi_j \varphi_j(\mathbf{x}) + \sum_{N_j \in N_b} \xi_j \varphi_j(\mathbf{x})$$

From (17), $\xi_j = u_{exact}(N_j)$ for $N_j in \in_b$, which results in

$$u_h(\mathbf{x}) = \sum_{N_j \in N_h} \xi_j \varphi_j(\mathbf{x}) + \sum_{N_j \in N_b} u_{exact}(N_j)\varphi_j(\mathbf{x})$$

Inserting this into the GFEM formulation, with $v = \varphi_i$, yields

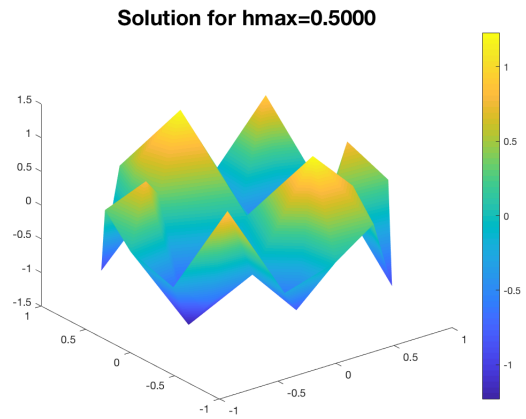
$$\underbrace{\sum_{N_j \in N_h} \xi_j}_{\xi} \underbrace{\int_\Omega \nabla\varphi_j(\mathbf{x})\nabla\varphi_i(\mathbf{x})d\mathbf{x}}_{\mathrm{A}} = \underbrace{\int_\Omega f(\mathbf{x})\varphi_i(\mathbf{x})d\mathbf{x} - \sum_{N_j \in N_b} u_{exact}(N_j) \int_\Omega \nabla\varphi_j(\mathbf{x})\nabla\varphi_i(\mathbf{x})d\mathbf{x}}_{\mathbf{b}}$$

which gives the following linear system:

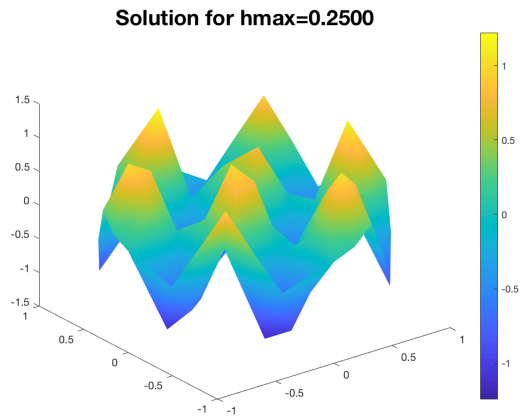
$$A\xi = \mathbf{b}$$

The system above is then implemented in Matlab, with mesh-size $h_{max} = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\}$. The homogeneous Dirichlet boundary conditions is applied strongly.

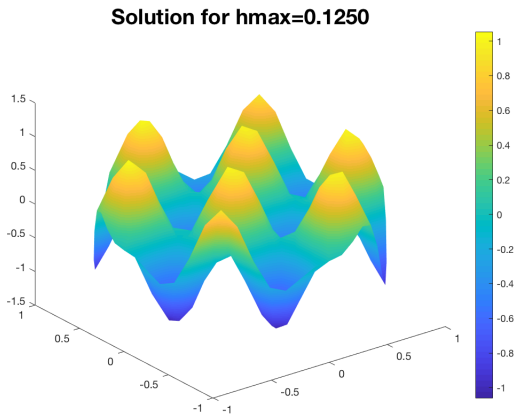
The convergence rate was found by determining the slope of the logarithm of the energy norm, Ene, as a function of the the logarithm of the mesh, h. Polyfit was used to determine this slope. And as can be seen, in 3, the convergence rate became P = 1.4.
In figure 2 the solution for problem 1 in part B is illustrated, for different mesh resolutions.



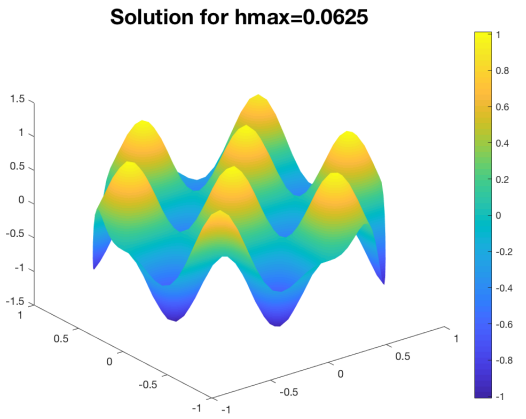
(a) $h_{max} = \frac{1}{2}$



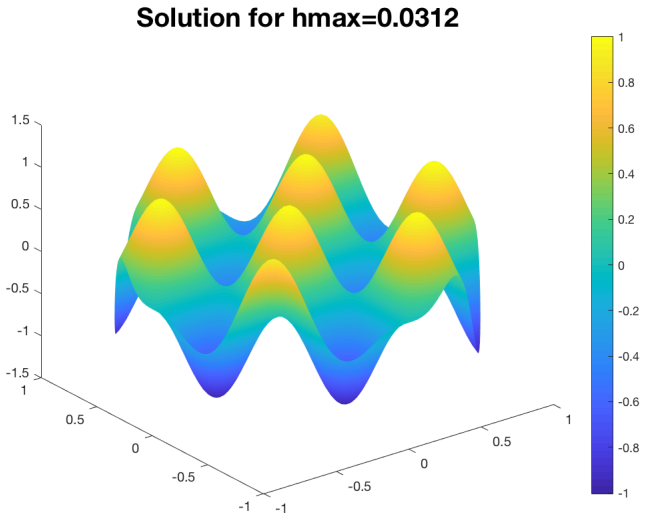
(b) $h_{max} = \frac{1}{4}$



(c) $h_{max} = \frac{1}{8}$



(d) $h_{max} = \frac{1}{16}$



(e) $h_{max} = \frac{1}{32}$

Figure 2: Solution for each mesh resolution

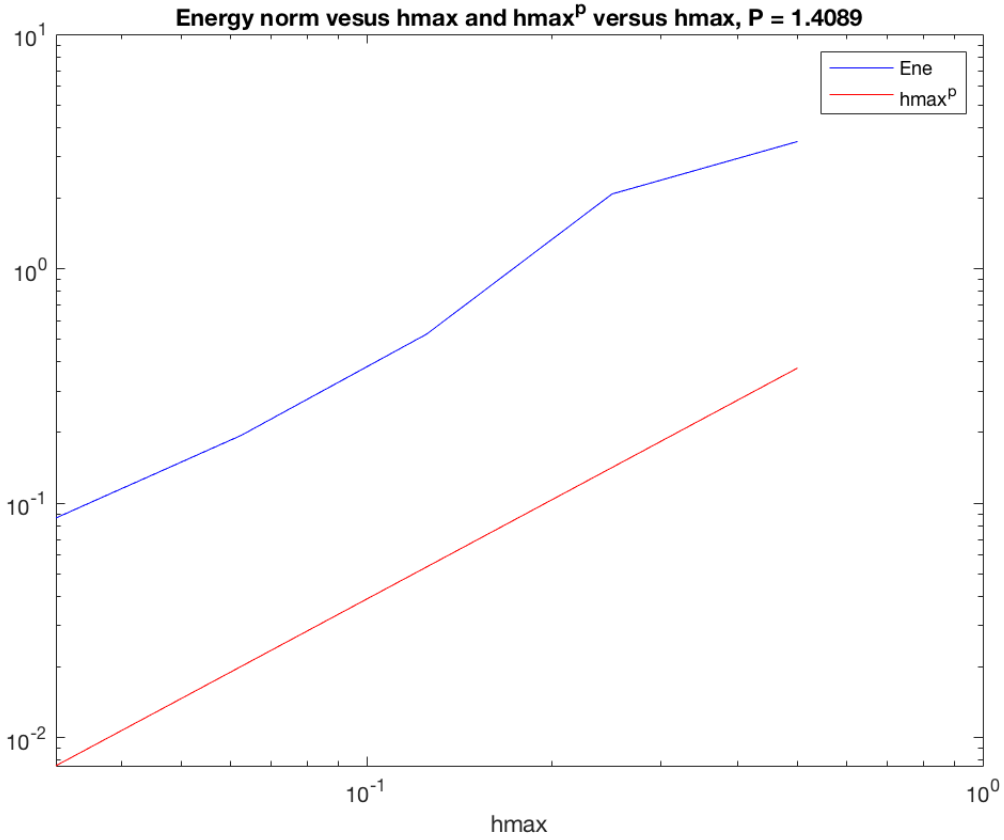


Figure 3: Energy norm versus h_{max}^P as functions of mesh-size h_{max}

3.2 Problem 2

In this part, the system (1)-(2) is considered with $v(\mathbf{x}, t)$ equals 1, meaning only the prey equation (1) in the system is considered, as following below. A randomized initial data and homogeneous Neumann boundary conditions are applied.

$$\partial_t u - u(1-u) + \frac{u}{u+\alpha} - \delta_1 \Delta u = f, (\mathbf{x}, t) \in \Omega x(0, T] \tag{18}$$

$$\partial_n u(\mathbf{x}, t) = 0 \, \forall \mathbf{x} \in \partial\Omega, \; t \in (0, T] \tag{19}$$

$$u(\mathbf{x}, t) = 1 + 20\omega(\mathbf{x}), \; \mathbf{x} \in \Omega \tag{20}$$

$$f(\mathbf{x}, t) = 0$$

Construct the test space $V_0 = \{v(\mathbf{x}, t) : \|v(\cdot, t)\|^2 + \|\nabla v(\cdot, t)\|^2 < \infty\}$, find $u \in V_0$ such that

$$\int_\Omega \partial_t uv d\mathbf{x} - \delta \int_\Omega \Delta uv d\mathbf{x} + \int_\Omega (\frac{u}{u+\alpha} - u(1-u))v d\mathbf{x} = 0, \; \forall v \in V_0, \; t \in (0, T]$$

Green's formula gives

$$\int_\Omega \partial_t uv d\mathbf{x} + \delta \int_\Omega \nabla u \nabla v d\mathbf{x} - \int_{\partial\Omega} n \nabla uv ds + \int_\Omega (\frac{u}{u+\alpha} - u(1-u))v d\mathbf{x} = 0$$

$$\int_\Omega \partial_t u v d\mathbf{x} + \delta \int_\Omega \nabla u \nabla v d\mathbf{x} + \int_\Omega (\frac{u}{u+\alpha} - u(1-u)) v d\mathbf{x} = 0$$

Form a triangulation of $\Omega : \tau_h = \{K\}$ and construct the following finite dimensional subspace

$$V_{h,0} \subset V_0, V_{h,0} = \{v(\mathbf{x},t) : v(\mathbf{x},t) \in C^0(\Omega) \forall t \in (0,t]; v|_K \in P_1(K), \forall K \in \tau_h\}$$

(GFEM) Find $u_h \in V_{h,0}$ such that

$$= \int_\Omega \partial_t u_h v d\mathbf{x} + \delta \int_\Omega \nabla u_h \nabla v d\mathbf{x} + \int_\Omega (\frac{u_h}{u_h+\alpha} - u_h(1-u_h)) v d\mathbf{x} = 0$$

Since $u_h(x,t) \in V_{h,0}, \exists \{\xi_j(t)\}_{N_j \in N_h}$ such that

$$u_h = \sum_{N_j \in N_h} \xi_(j) \varphi(\mathbf{x}) \tag{21}$$

Denote the non-linear terms as $S = \frac{u_h}{u_h+\alpha} - u_h(1-u_h)$ and take linear interpolant of S as following
$\Pi_h S \approx S$ $\Pi_h S \in V_{h,0} \Rightarrow \Pi_h S(x,t) = \sum_{N_j \in N_h} S_j(t) \varphi_j(x)$
$S = \frac{u_h}{u_h+\alpha} - u_h(1-u_h) \Rightarrow (u_h)_j^t = \xi_j(t)$

$$S_j(t) = \frac{\xi_j(t)}{\xi_j(t)+\alpha} - \xi_j(t)(1-\xi_j(t)) = \frac{\xi_j(t)}{\xi_j(t)+\alpha} - \xi_j(t) + (\xi_j(t))^2$$

$$\Pi_h S(\mathbf{x},t) = \sum_{N_j \in N_h} (\frac{\xi_j(t)}{\xi_j(t)+\alpha} - \xi_j(t) + (\xi_j(t))^2) \varphi_j \tag{22}$$

Inserting (21) and (22) into GFEM we obtain the following

$$\underbrace{\sum_{N_j \in N_h} \frac{\partial \xi_j(t)}{\partial t}}_{\frac{\partial \xi}{\partial t}} \underbrace{\int_\Omega \varphi_j(\mathbf{x}) \varphi_i(\mathbf{x}) d\mathbf{x}}_{\mathrm{M}} + \delta_1 \underbrace{\sum_{N_j \in N_h} \xi_j(t)}_{\xi} \underbrace{\int_\Omega \nabla \varphi_j(\mathbf{x}) \nabla \varphi_i(x) d\mathbf{x}}_{\mathrm{A}} + \underbrace{\sum_{N_j \in N_h} S_j(t)}_{\mathbf{S}} \underbrace{\int_\Omega \varphi_j \varphi_i d\mathbf{x}}_{\mathrm{M}}$$

Which gives the following linear system

$$\begin{cases} M \frac{\partial \xi_(t)}{\partial t} + M\mathbf{S} + \delta_1 A \xi = 0 = \\ \xi(0) = 1 + 20\omega(\mathbf{x}), \mathbf{x} \in \Omega \end{cases}$$

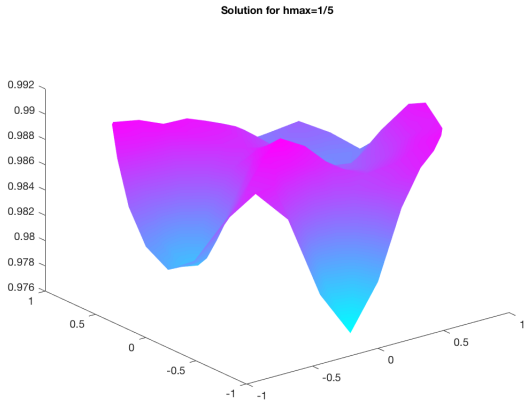
Using Crank-Nicholson for discretizing time yields

$$M \frac{\xi^{n+1} - \xi^n}{k_n} + M(\frac{\xi^n}{\xi^n+\alpha} - \xi^n + (\xi^n)^2) + \delta_1 A(\frac{\xi^{n+1}+\xi^n}{2}) = 0$$

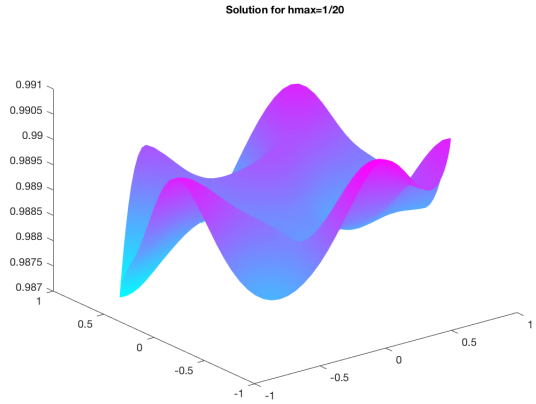
Multiplying both sides by k_n and sort the ξ terms by n and n+1, yields:

$$\xi^{n+1} = (M + \frac{\delta_1}{2} k_n A) \backslash ((M - \frac{\delta_1}{2} k_n A)\xi^n - k_n M(\frac{\xi^n}{\xi^n+\alpha} - \xi^n + (\xi^n)^2))$$

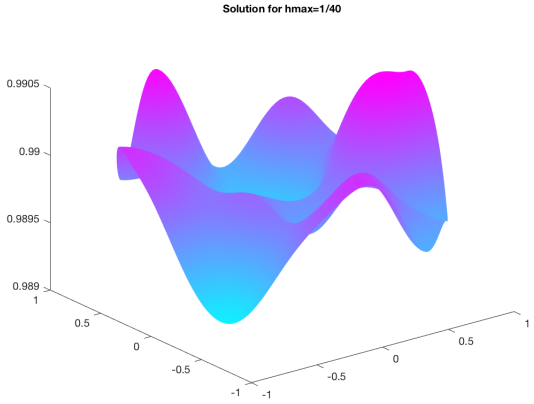
Implementing the above system into Matlab, with mesh-size $h_{max} = \{\frac{1}{5}, \frac{1}{20}, \frac{1}{40}\}$, diffusion parameter = 0.01, reaction parameter $\alpha = 4$ and stop-time $T = 2$, yields following solutions, illustrated in figure 4 and 5, for different mesh resolutions. Since the initial data is random, the solution changes shape for every mesh-size. Implementing the equation for the population change of preys (5), using the Trapezoidal rule a $\int_K F(\mathbf{x}) d\mathbf{x} \approx \frac{|K|}{3} \sum_{i=1}^3 F(N_i)$, where $N_i, i = 1,2,3$, is the nodal points of the cell K.
yields the graph, seen in 6:



(a) $h_{max} = \frac{1}{5}$

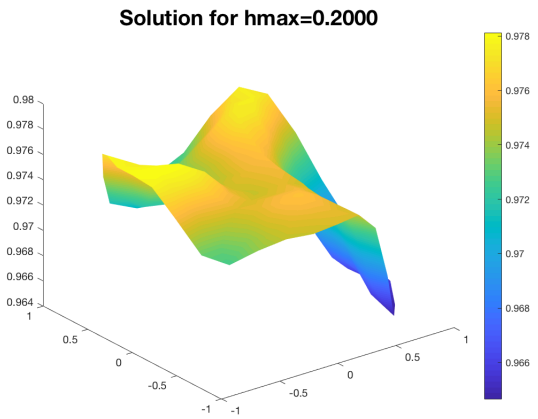


(b) $h_{max} = \frac{1}{20}$

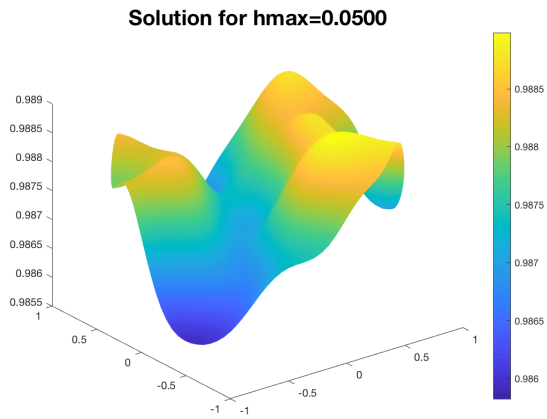


(c) $h_{max} = \frac{1}{40}$

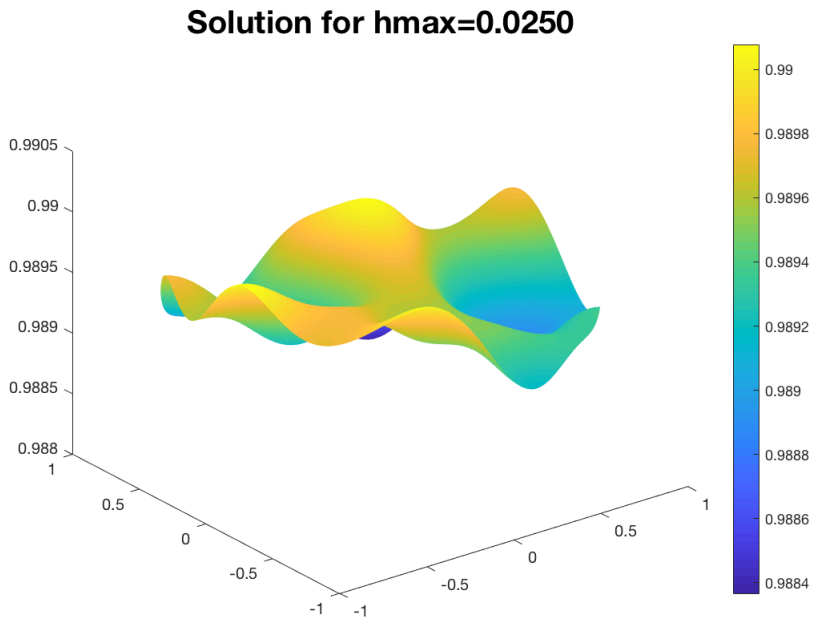
Figure 4: Result for B2



(a) $h_{max} = \frac{1}{5}$



(b) $h_{max} = \frac{1}{20}$



(c) $h_{max} = \frac{1}{40}$

Figure 5: Solution for each mesh resolution

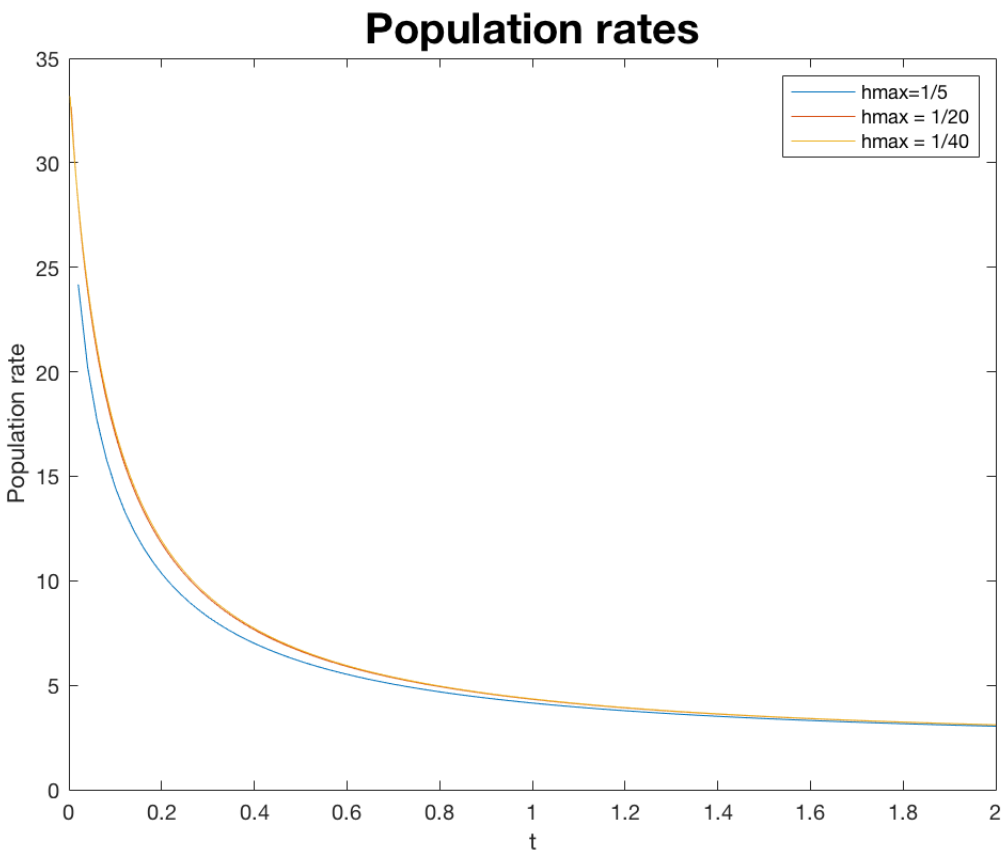


Figure 6: Population rates of preys as function of time

4 Part C

In part C, the prey-predator model (1)-(2) is considered. The problem is defined with zero source terms, i.e. $f(\mathbf{x},t) = g(\mathbf{x},t) = 0$, and with parameters $\delta_1 = \delta_2 = 1, \alpha = 0.4, \beta = 2, \gamma = 0.8$. For the sake of simplicity; let the solution $\mathbf{u}(\mathbf{x},t) = (u_1(\mathbf{x},t), u_2(\mathbf{x},t))$ and let $\mathbf{v}(\mathbf{x},t) = v_1(\mathbf{x},t), v_2(\mathbf{x},t)$ denote the test functions. The initial data will be defined for each problem. However, homogeneous Neumann boundary conditions in all boundary points applies to both of the problems, as:

$$\partial_n u_1(\mathbf{x},t) = \partial_n u_2(\mathbf{x},t) = 0$$

Multiplying (1) by v_1 and (2) by v_2 and integrate, using Green's formula for the Δ-term, leads to the weak formulation: Find $\mathbf{u}(\mathbf{x},t) = (u_1, u_2) \in W = \{\mathbf{v}(\mathbf{x},t), \mathbf{v}(\mathbf{x},t) \in H^1(\Omega) \forall t \in [0,T]\}$ such that

$$\int_\Omega \partial_t u_1 v_1 d\mathbf{x} + \delta_1 \int_\Omega \nabla u_1 \nabla v_1 d\mathbf{x} + \int_\Omega (\frac{u_1 u_2}{u_1 + \alpha} - u_1(1 - u_1)) v_1 d\mathbf{x} = 0, \ \forall \mathbf{v} \in W$$

$$\int_\Omega \partial_t u_2 v_2 d\mathbf{x} + \delta_2 \int_\Omega \nabla u_2 \nabla v_2 d\mathbf{x} + \int_\Omega (\gamma u_2 - \beta \frac{u_1 u_2}{u_1 + \alpha}) v_2 d\mathbf{x} = 0, \ \forall \mathbf{v} \in W$$

GFEM: Find $u_h \in W_h = \{\mathbf{v}(\mathbf{x},t), \mathbf{v}(\mathbf{x},t) \in C^0(\Omega) \forall t \in [0,T], \mathbf{v}|_K \in P_1(K), \forall K \in \tau_h\}$ such that

$$\int_\Omega \partial_t u_{1,h} v_1 d\mathbf{x} + \delta_1 \int_\Omega \nabla u_1 \nabla v_1 d\mathbf{x} + \int_\Omega (\frac{u_{1,h} u_{2,h}}{u_{1,h} + \alpha} - u_{1,h} + u_{1,h}^2) v_1 d\mathbf{x} = 0, \ \forall \mathbf{v} \in W_h$$

$$\int_\Omega \partial_t u_{2,h} v_2 d\mathbf{x} + \delta_2 \int_\Omega \nabla u_{2,h} \nabla v_2 d\mathbf{x} + \int_\Omega (\gamma u_{2,h} - \beta \frac{u_{1,h} u_{2,h}}{u_{1,h} + \alpha}) v_2 d\mathbf{x} = 0, \ \forall \mathbf{v} \in W_h$$

The finite element space $W_h \subset W, dim(W_h) < \infty$ is a vector valued space and can be written as: $W_h = V_h \times V_h$, where $V_h = span(\{\varphi\}_{j N_j \in N_h})$. The basis set for W_h is therefore given by:

$$\{\begin{bmatrix}\varphi_0 \\ 0\end{bmatrix}, \begin{bmatrix}\varphi_1 \\ 0\end{bmatrix}, ..., \begin{bmatrix}\varphi_N \\ 0\end{bmatrix}, \begin{bmatrix}0 \\ \varphi_0\end{bmatrix}, \begin{bmatrix}0 \\ \varphi_1\end{bmatrix}, ..., \begin{bmatrix}0 \\ \varphi_N\end{bmatrix}\},$$

Since $u_h \in W_h \exists \{\xi_j\}_{N_j \in N_h}$ such that

$$u_h = \sum_{N_j \in N_h} \xi_j \varphi_j(\mathbf{x}), \ \xi_j = (\xi_{1,j}, \xi_{1,j})^T \tag{23}$$

Denote the non-linear terms as $S_1 = \frac{u_{1,h} u_{2,h}}{u_{1,h} + \alpha} + u_{1,h}^2$ and $S_2 = \frac{u_{1,h} u_{2,h}}{u_{1,h} + \alpha}$, take linear interpolant of S as following $\Pi_h S \approx S \ \Pi_h S \in W_h \Rightarrow \Pi_h S(\mathbf{x},t) = \sum_{N_j \in N_h} S_j(t) \varphi_j(\mathbf{x})$

$$S_{1,j}(t) = \frac{\xi_{1,j}(t)}{\xi_{1,j}(t) + \alpha} + (\xi_{1,j}(t))^2$$

$$\Pi_h S_1(\mathbf{x},t) = \sum_{N_j \in N_h} (\frac{\xi_{1,j} \xi_{2,j}(t)}{\xi_{1,j}(t) + \alpha} + (\xi_{1,j}(t))^2) \varphi_j(\mathbf{x}) \tag{24}$$

$$S_{2,j}(t) = \frac{\xi_{1,j} \xi_{2,j}(t)}{\xi_{1,j}(t) + \alpha}$$

$$\Pi_h S_2(\mathbf{x},t) = \sum_{N_j \in N_h} (\frac{\xi_{1,j} \xi_{2,j}(t)}{\xi_{1,j}(t) + \alpha}) \varphi_j(\mathbf{x}) \tag{25}$$

Inserting (23), (24) and (25) into GFEM yields the following systems:

$$\underbrace{\sum_{N_j\in N_h}\frac{\partial\xi_{1,j}(t)}{\partial t}}_{\frac{\partial\xi_1}{\partial t}}\underbrace{\int_\Omega\varphi_j\varphi_i d\mathbf{x}}_{\mathrm{M}}+\delta_1\underbrace{\sum_{N_j\in N_h}\xi_{1,j}(t)}_{\xi_1}\underbrace{\int_\Omega\nabla\varphi_j\nabla\varphi_i d\mathbf{x}}_{\mathrm{A}}+\underbrace{\sum_{N_j\in N_h}S_{1,j}(t)}_{\mathbf{S}_1}\underbrace{\int_\Omega\varphi_j\varphi_i d\mathbf{x}}_{\mathrm{M}}-\underbrace{\sum_{N_j\in N_h}\xi_{1,j}(t)}_{\xi_1}\underbrace{\int_\Omega\varphi_j\varphi_i d\mathbf{x}}_{\mathrm{M}}$$

$$\underbrace{\sum_{N_j\in N_h}\frac{\partial\xi_{2,j}(t)}{\partial t}}_{\frac{\partial\xi_2}{\partial t}}\underbrace{\int_\Omega\varphi_j\varphi_i d\mathbf{x}}_{\mathrm{M}}+\delta_2\underbrace{\sum_{N_j\in N_h}\xi_{2,j}(t)}_{\xi_2}\underbrace{\int_\Omega\nabla\varphi_j\nabla\varphi_i d\mathbf{x}}_{\mathrm{A}}-\beta\underbrace{\sum_{N_j\in N_h}S_{2,j}(t)}_{\mathbf{S}_2}\underbrace{\int_\Omega\varphi_j\varphi_i d\mathbf{x}}_{\mathrm{M}}+\gamma\underbrace{\sum_{N_j\in N_h}\xi_{2,j}(t)}_{\xi_2}\underbrace{\int_\Omega\varphi_j\varphi_i d\mathbf{x}}_{\mathrm{M}}$$

Which leads to the following system:

$$\begin{cases} M\frac{\partial\xi_1(t)}{\partial t}+M\mathbf{S}_1+\delta_1A\xi_1-M\xi_1(t)=0 \\ M\frac{\partial\xi_2(t)}{\partial t}-\beta M\mathbf{S}_2+\delta_2A\xi_{\mathbf{2}}+\gamma M\xi_2(t)=0 \\ \xi_1(0)=u_{1,0}(\mathbf{x}) \\ \xi_2(0)=u_{2,0}(\mathbf{x}) \end{cases}$$

Using Crank-Nicholson for discretizing time yields

$$M\frac{\xi_1^{n+1}-\xi_1^n}{k_n}+M(\frac{\xi_1^n\xi_2^n}{\xi_1^n+\alpha}+(\xi_1^n)^2)-M(\frac{\xi_1^{n+1}+\xi_1^n}{2})+\delta_1A(\frac{\xi_1^{n+1}+\xi_1^n}{2})=0$$

$$M\frac{\xi_2^{n+1}-\xi_2^n}{k_n}-\beta M(\frac{\xi_1^n\xi_2^n}{\xi_1^n+\alpha})+\gamma M(\frac{\xi_2^{n+1}+\xi_2^n}{2})+\delta_2A(\frac{\xi_2^{n+1}+\xi_2^n}{2})=0$$

Multiplying both sides by k_n and sort the ξ terms by n and n+1, yields

$$\xi_1^{n+1}=(M+\frac{\delta_1}{2}k_nA-\frac{1}{2}k_nM)\backslash((M-\frac{\delta_1}{2}k_nA+\frac{1}{2}k_nM)\xi_1^n-k_nM(\frac{\xi_1^n\xi_2^n}{\xi_1^n+\alpha}+(\xi_1^n)^2))$$

$$\xi_2^{n+1}=(M+\frac{\delta_2}{2}k_nA+\gamma\frac{1}{2}k_nM)\backslash((M-\frac{\delta_2}{2}k_nA-\gamma\frac{1}{2}k_nM)\xi_1^n-k_nM\beta\frac{\xi_1^n\xi_2^n}{\xi_1^n+\alpha})$$

A FEnICS solver is implemented to solve the system (1)-(2), using the Crank-Nicholson scheme above and a given file *circle_finest.xml* for the mesh.

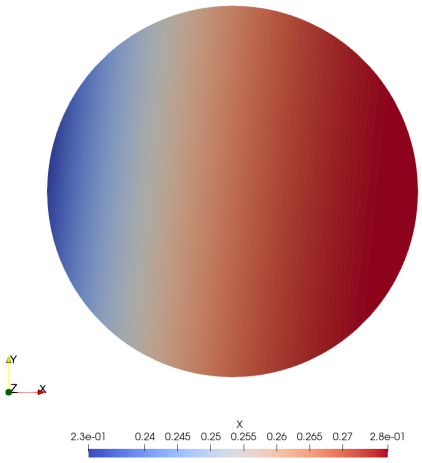
4.1 Problem 1

The given initial data for this problem is defined below

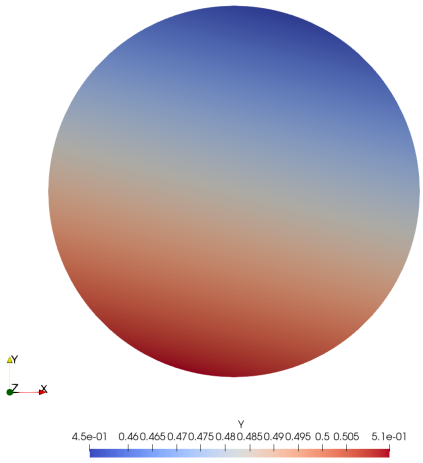
$$u_1(\mathbf{x},0)=\frac{4}{15}-2\cdot10^{-7}(x_1-0.1x_2-225)(x_1-0.1x_2-675)$$

$$u_2(\mathbf{x},0)=\frac{22}{45}-3\cdot10^{-5}(x_1-450)-1.2\cdot10^{-4}(x_2-150)$$

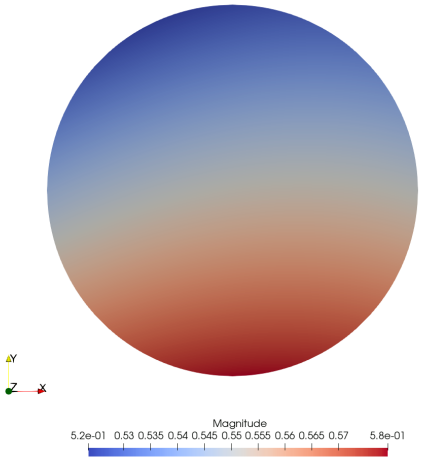
The code is performed with time-step $\Delta t=0.5$ until time $T=1000$, where the solutions are saved at time *T = 0, 50, 100, 150, 1000*, illustrated in 7-11. Scheme (a) corresponds to the population density for preys - solution of equation (1), scheme (b) to the to the population density for predators - solution of (2), and scheme (c) the magnitude of the population density of preys and predators - the system (1)-(2). Scheme (d) illustrates a line chart for each solution, which was obtained by plotting on the diagonal line.



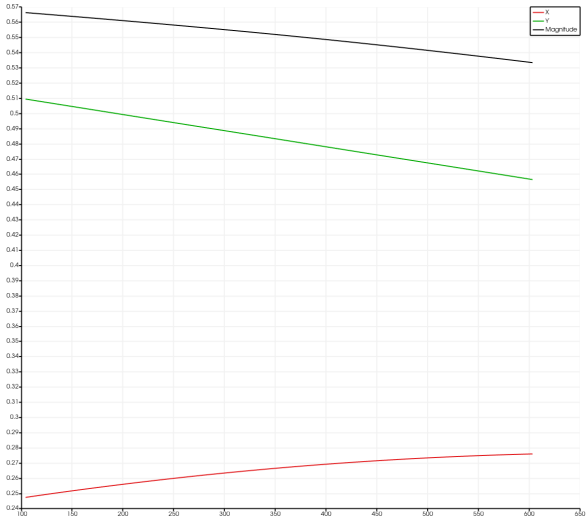
(a) Population density of prey



(b) Population density of predator

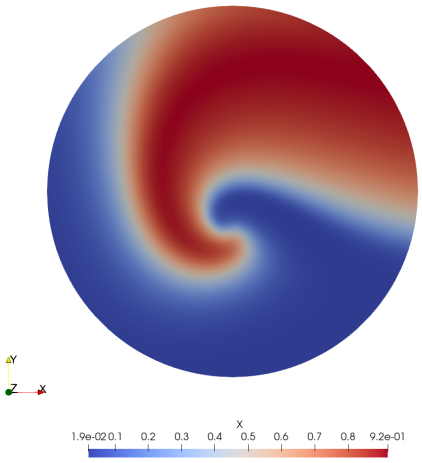


(c) The magnitude of total density of preys and predators

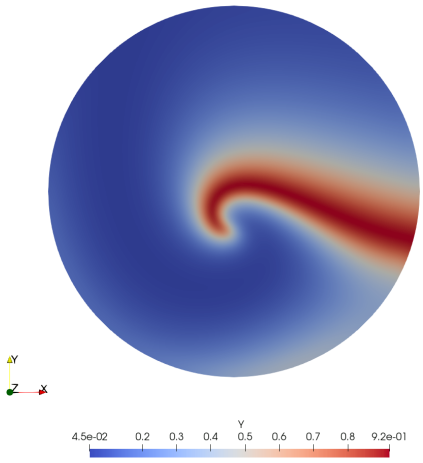


(d) Line chart of preys, predators and the total population

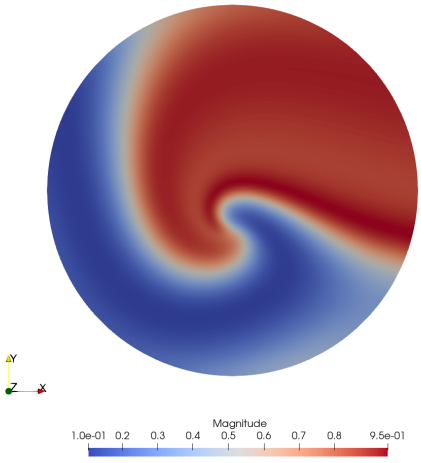
Figure 7: Solution at time t = 0



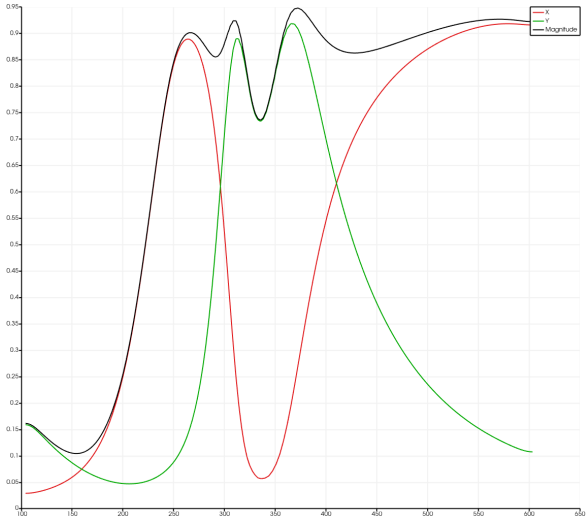
(a) Population density of prey



(b) Population density of predator

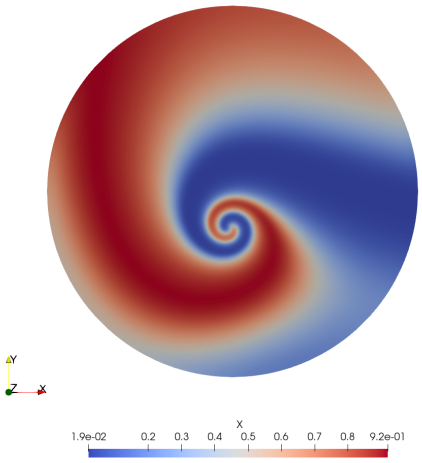


(c) The magnitude of total density of preys and predators

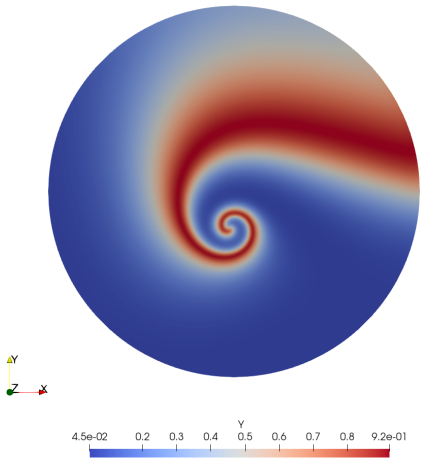


(d) Line chart of preys, predators and the total population

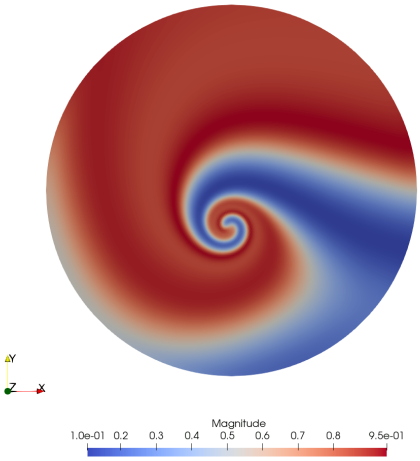
Figure 8: Solution at time t = 50



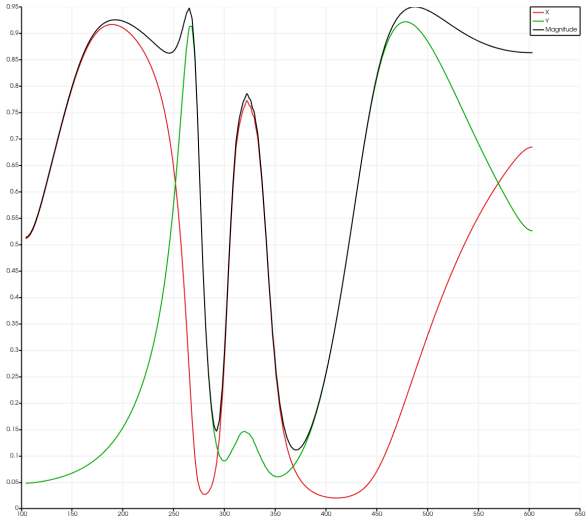
(a) Population density of prey



(b) Population density of predator

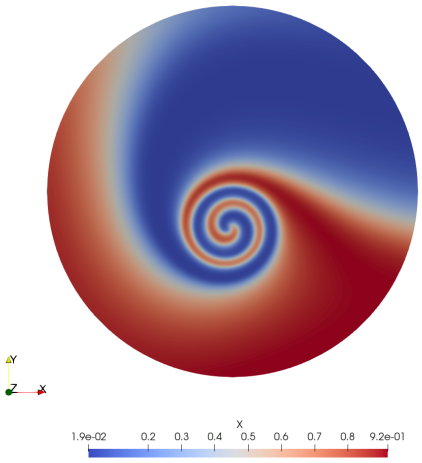


(c) The magnitude of total density of preys and predators

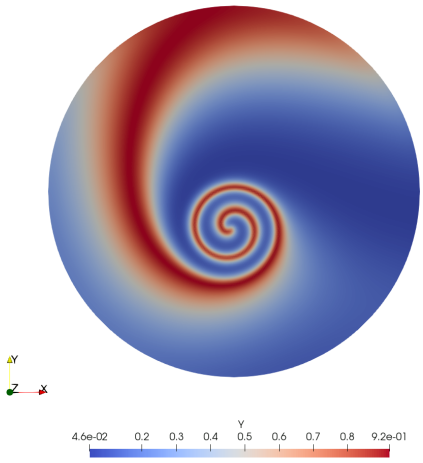


(d) Line chart of preys, predators and the total population

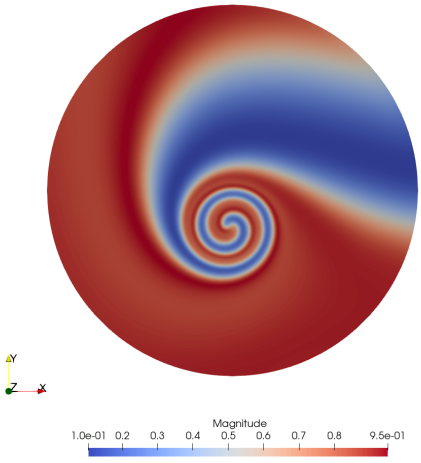
Figure 9: Solution at time t = 100



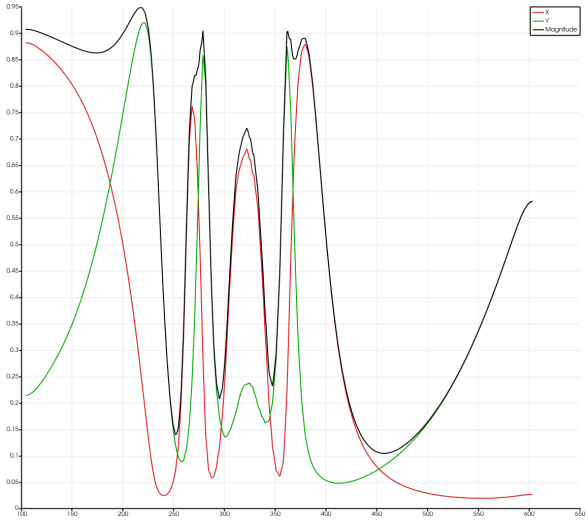
(a) Population density of prey



(b) Population density of predator

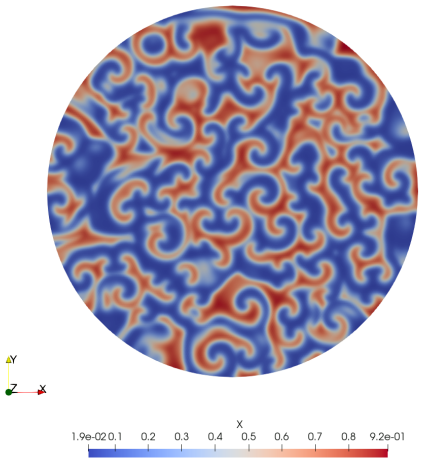


(c) The magnitude of total density of preys and predators

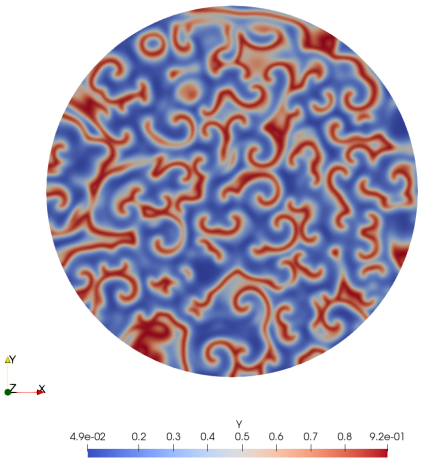


(d) Line chart of preys, predators and the total population

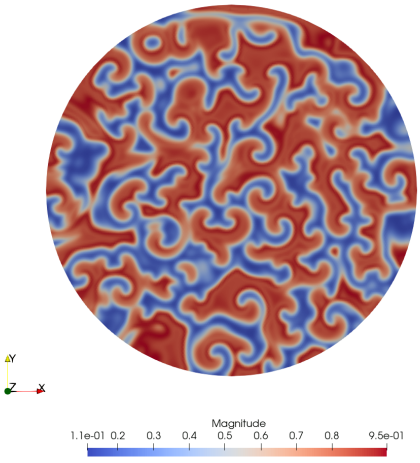
Figure 10: Solution at time t = 150



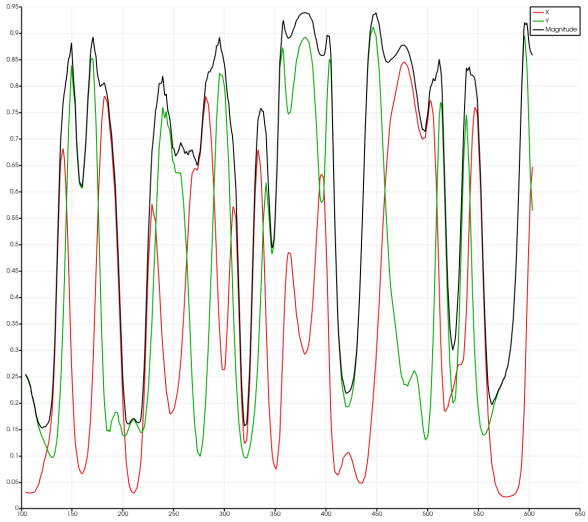
(a) Population density of prey



(b) Population density of predator



(c) The magnitude of total density of preys and predators



(d) Line chart of preys, predators and the total population

Figure 11: Solution at time t = 1000

Implementing the equations for the population change of preys and predators, from equation (5), yields the graph presented in 6.

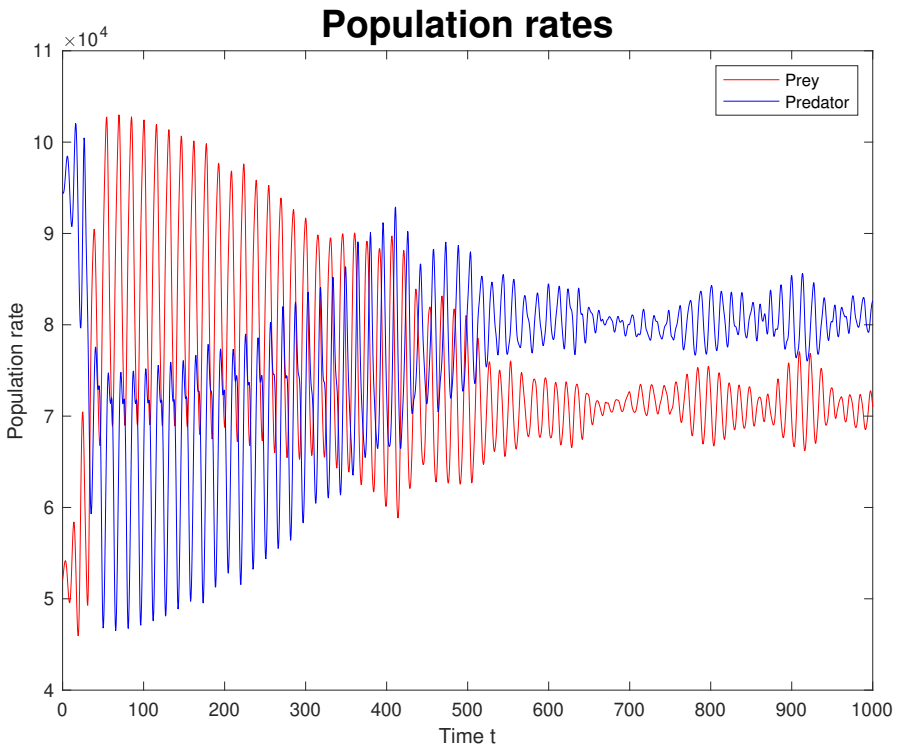


Figure 12: Population rates of preys and predators as functions of time

4.2 Problem 2

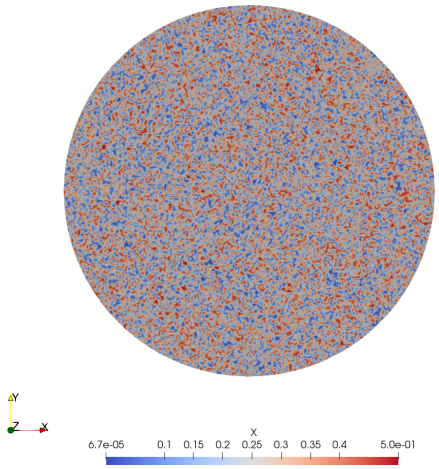
In this problem, a random initial data is used, defined as:

$$u_1(\mathbf{x}, 0) = \frac{1}{2}(1 - random(\mathbf{x}))$$

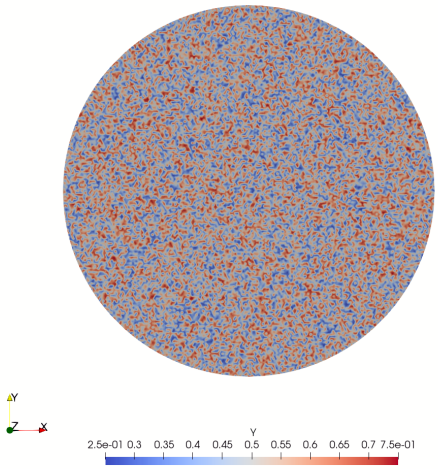
$$u_2(\mathbf{x}, 0) = \frac{1}{4} + \frac{1}{2}(1 - random(\mathbf{x}))$$

where $random(\mathbf{x}) : \mathbb{R}^2 \mapsto [0, 1]$, is a random number between 0 and 1.

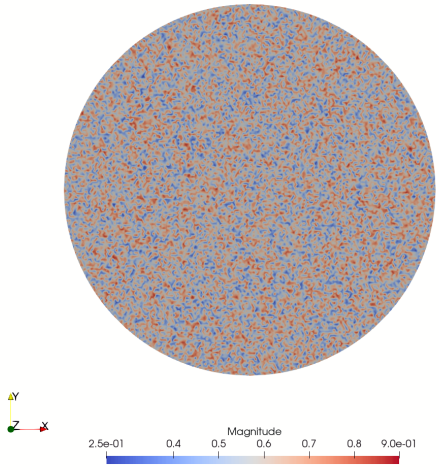
The initial data is implemented using the function *random.uniform(0,1)* in Python. The same time-step is used here, $\Delta t = 0.5$ and the code runs until time $T = 1000$, where the solutions are saved at time *T=0,50,100,500,1000*, illustrated in 13-17. Scheme (a) corresponds to the population density for preys; the solution of equation (1), scheme (b) to the to the population density for predators; the solution of (2), and scheme (c) the magnitude of the population density of preys and predators; the system (1)-(2). Scheme (d) illustrates a line chart with each solution.



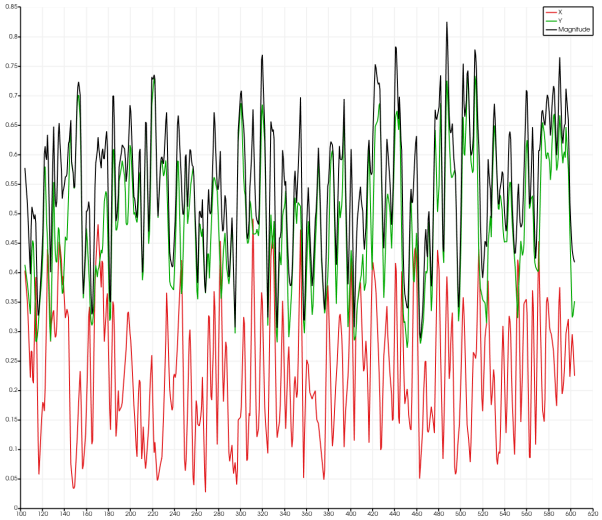
(a) Population density of prey



(b) Population density of predator

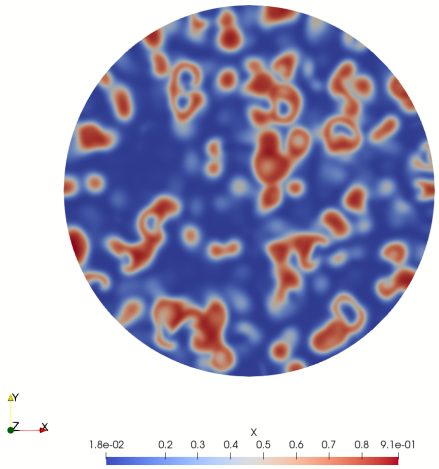


(c) The magnitude of total density of preys and predators

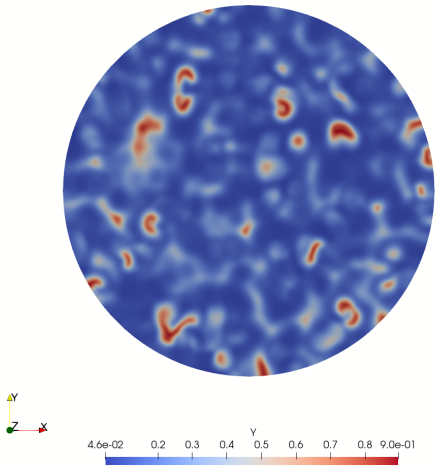


(d) Line chart of preys, predators and the total population

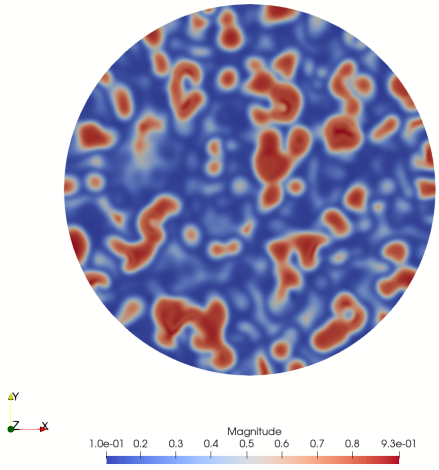
Figure 13: Solution at time t = 0



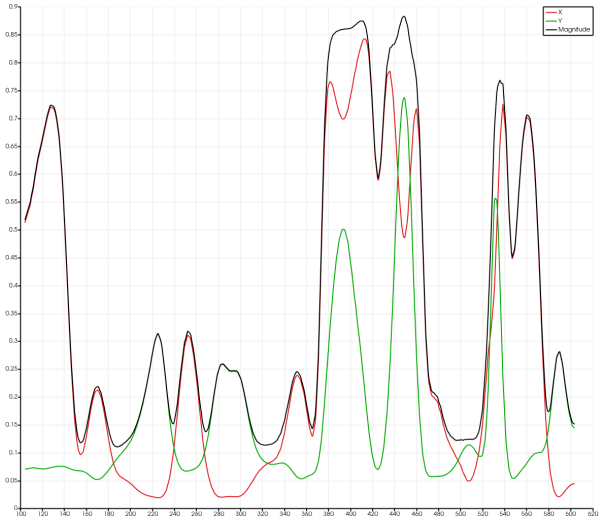
(a) Population density of prey



(b) Population density of predator

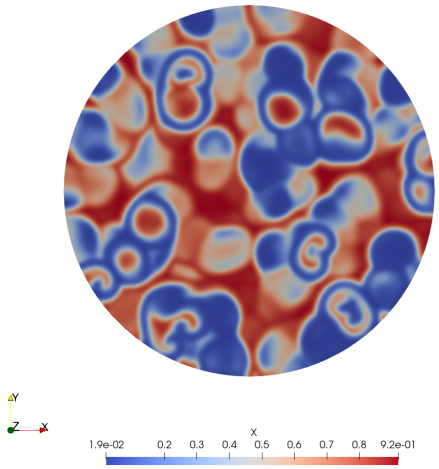


(c) The magnitude of total density of preys and predators

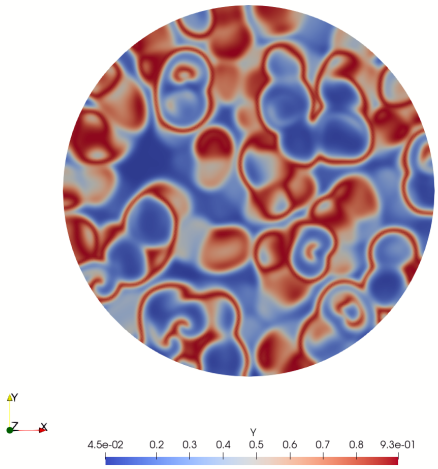


(d) Line chart of preys, predators and the total population

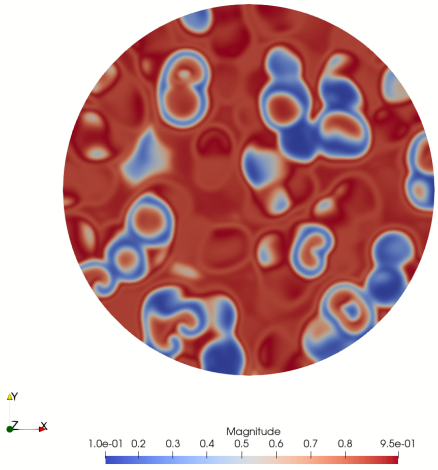
Figure 14: Solution at time t = 50



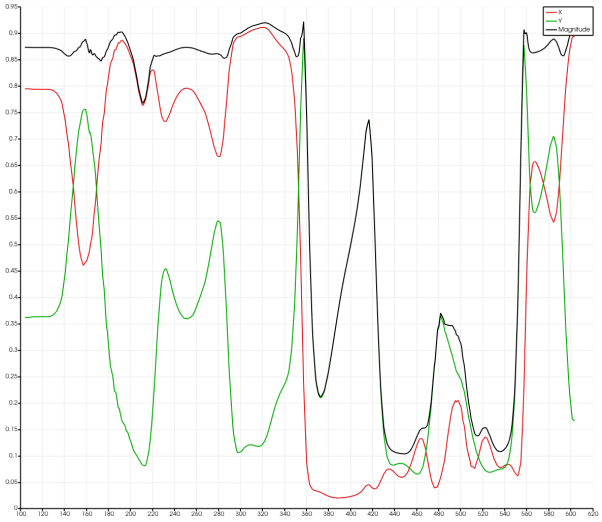
(a) Population density of prey



(b) Population density of predator

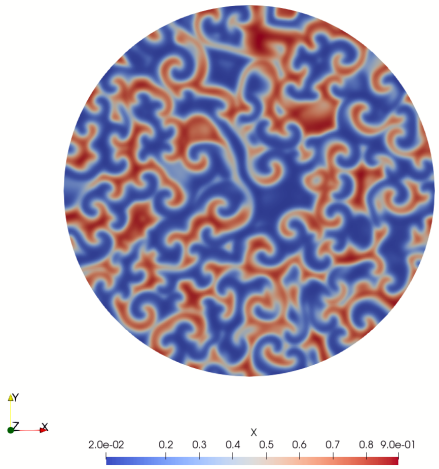


(c) The magnitude of total density of preys and predators

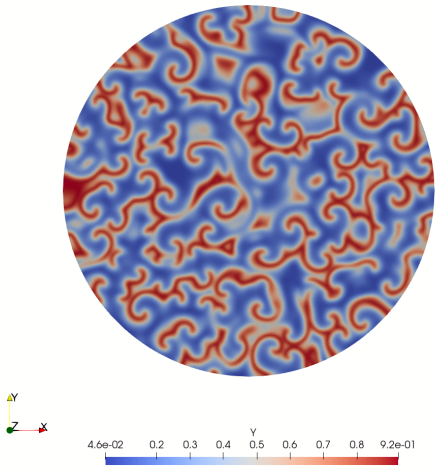


(d) Line chart of preys, predators and the total population

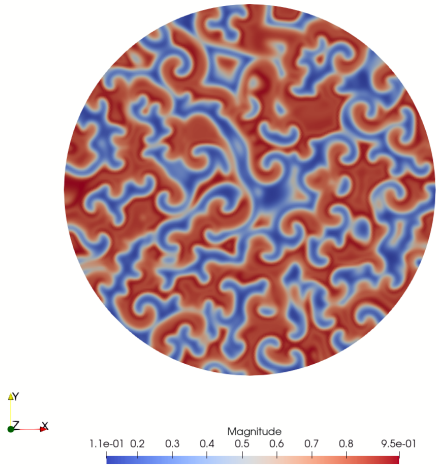
Figure 15: Solution at time t = 100



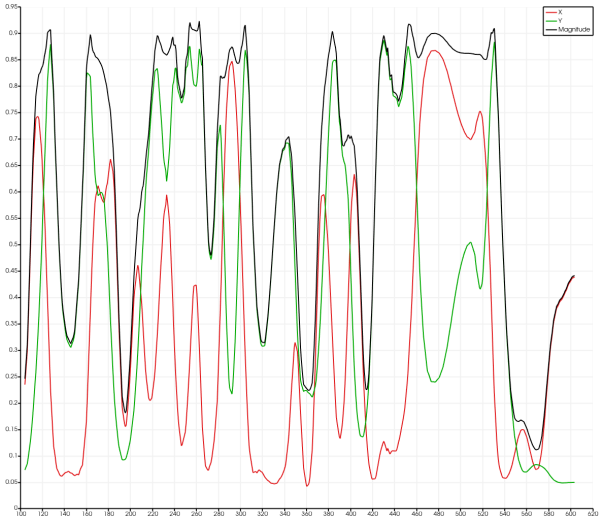
(a) Population density of prey



(b) Population density of predator

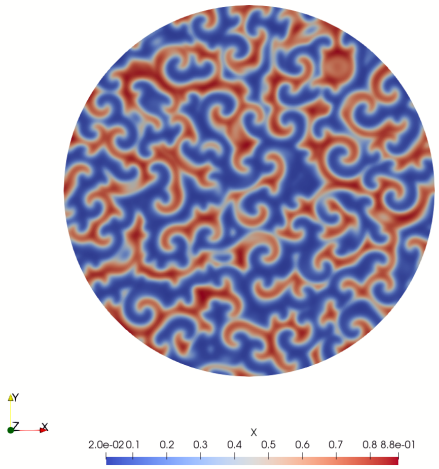


(c) The magnitude of total density of preys and predators

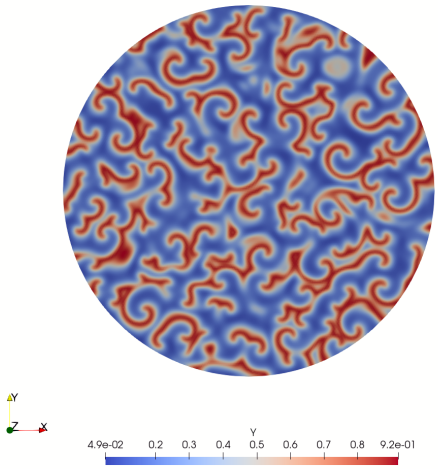


(d) Line chart of preys, predators and the total population

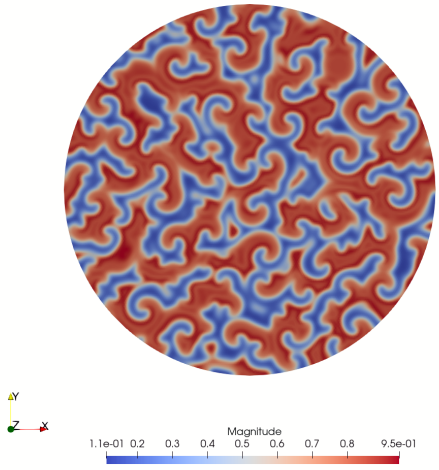
Figure 16: Solution at time t = 500



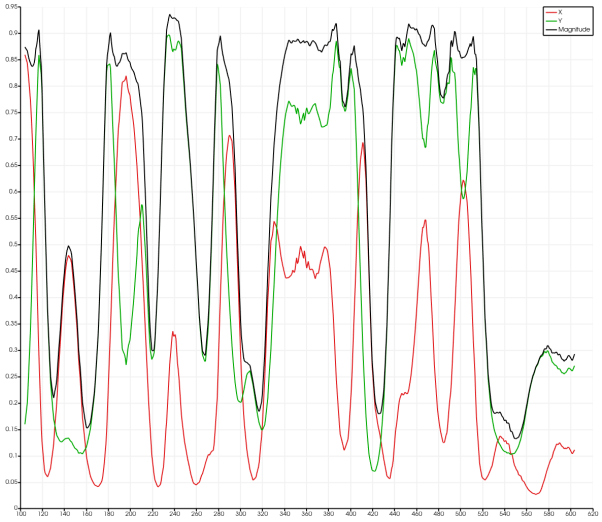
(a) Population density of prey



(b) Population density of predator



(c) The magnitude of total density of preys and predators



(d) Line chart of preys, predators and the total population

Figure 17: Solution at time t = 1000

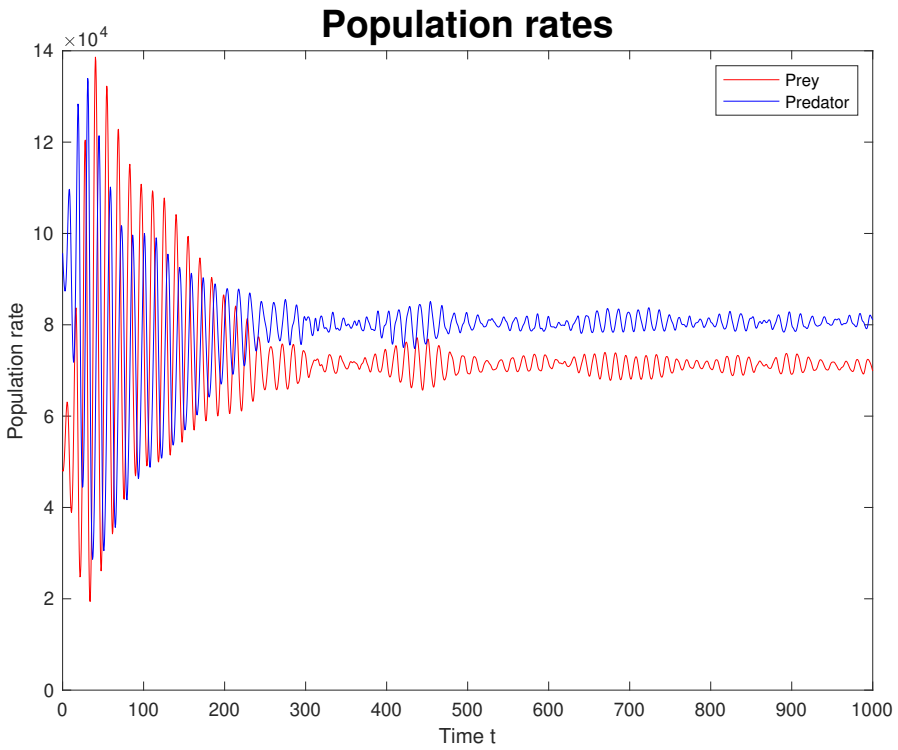


Figure 18: Population rates of preys and predators as functions of time

5 Discussion and conclusion

In Part A, a simple one-dimensional equation was considered. First of all, a posteriori error estimate in the energy norm was derived and then implemented using algorithm for adaptive finite element approximation. In the second problem of part A, an adaptive 1D finite element solver was implemented in order to solve the task given by (6)-(7). The solution u_h, the residual $R(u_h)$, error indicator $\eta(u_h)$ and the mesh-size distribution was plotted, shown in 1. What can be concluded from these plots is that the error indicator and the residual reach its maximum at the turning point of the solution.

In the first problem of Part B, a simplified equation of the prey-equation was considered. Firstly, the Galerkin finite element method was produced and then the system was implemented in Matlab and plotted for different mesh resolutions, shown in 2. It was clear, that for low mesh resolutions, the solution was defective. For mesh-sizes below $h = 1/12$, the solution was more accurate and kept the same. Since an exact solution existed, the energy norm could be computed, and from the energy norm the convergence rate was calculated to 1.4.

In the second problem of part B, the prey-equation was considered with constant predator-density and a random initial data. The Galerkin finite element method was found and then the Crank-Nicholson method was used to discretize the time-derivative. The results for solution for different mesh-sizes is shown in 5 and 4. The population rate for preys was computed, independently of mesh-size, the population clearly decreased and converged to the same value. Since the predator density remains constant, there is no actual reaction term (since the population density for predators never decreases, the prey population never increases), and therefore the diffusion term "dominates" the equation, and that is why the population rate diffuses. After a while, the population rate for preys seems to have reached some type of equilibrium in relation to the constant predator density.

Part C1 In figure 12, we see that as initial data there is higher population rate for predator (fox) than for preys (rabbits). The reaction-term in (1) ($\frac{uv}{u+\alpha}$), in (2) ($-\beta\frac{uv}{u+}$) explains how the two species interact with each other: When v increases, u decreases, and when then u decreases v decreases too. This explains what happens in the population rate graph 12. There is a larger density of predators than preys at first, which makes the predator population decrease first and the preys increase fast. Then we can see that, as the predator increases, preys decreases. After a while, the population rates becomes less fast in decreasing and increasing, meaning more "constant", we have probably reach some stability of the prey-predator model. This can also be seen in the solutions and the line charts, 7-11. We see can clearly see in the line charts, that at the beginning the population densities are very "tupsy-turvy", this behavior is probably because the reaction term is dominating at first, and the model is therefore very sensitive to changes in the prey and predator densities, since the densities are so great at first. At the final time, T = 1000, we can see that the density of preys and predators have reached some stability (which was also concluded from the population rates). We still have decreasing and increasing of the densities, but it is clearer to see what is happening now. What is meant with stability, is stability between the diffusion and the reaction term. It looks like the predator is always some degrees behind the preys, meaning that, for example, firstly the preys increases, leading to that the predators increases, which further leads to that the preys decreases which leads to that the predators decreases and then the preys increases again, and so on.

C2: In the second problem of part C, results shown in 13-17 and 18, a initial data was used, which made the model more stable at the beginning, compared to the first problem. Otherwise, the same assumptions is made here: that there are great population densities and a lot of reaction at first, but later the model "stabilizes", as we have more diffusion.

Appendix

Part A

Listing 1: Main script for part A

```
a = -1; % left end point of interval
b = 1; % right
N = 12; % number of intervals (start)
x = linspace(a,b,N); % Vector of nodes
Δ = 0.01;
TOL = 1e-3;
lambda = 0.9;
eta2 = 10; %start value to get in the while lop
while sum(sqrt(eta2))>TOL

    %Assemble matricies
    A = my_stiffness_matrix_assembler(x);
    B = my_load_vector_assembler(x,Δ);
    M = my_mass_matrix_assembler(x);

    %Solving eq. systems
    xi = A\B;
    xi_biss = -M\(A*xi); %Discrete Laplcacian

    eta2 = my_eta2(x,xi_biss, Δ); %error indicator

    %Residuals
    R = zeros(length(x),1);
    for i = 1:length(x)
        R(i) = my_f(x(i)) + Δ*xi_biss(i);
    end

    x1 =x;   %x value to plot with

    %Refinement
    for i = 1:length(eta2)
        if eta2(i) > lambda*max(eta2)
            x = [x (x(i+1)+x(i))/2];
        end
    end
    x = sort(x);
    N=length(x);
end

%Plotting the results
subplot(2,2,1)
plot(x1,xi)
title ('Solution')
xlabel('x')
ylabel('u_h')

subplot(2,2,2)
plot(x1,sqrt(eta2))
title ('Error indicator')
xlabel('x')
ylabel('eta(u_h)')

subplot(2,2,3)
plot(x1(2:end),[1./diff(x1)])
title ('Mesh-size distribution')
xlabel('x')
ylabel('Distribution(x)')
```

```
58
59 subplot(2,2,4)
60 plot(x1,R)
61 title ('Residuals')
62 xlabel('x')
63 ylabel('R(u_h)')
```

Listing 2: Matlab script for function f(x)

```
1  function f=my_f(x)
2      if abs(x)≤0.1
3          f = -5;
4
5      elseif abs(0.2-abs(x))≤0.1
6          f = 25;
7
8      elseif abs(0.6-abs(x))≤0.1
9          f = -30;
10
11     elseif abs(0.9-abs(x))≤0.2
12         f = 20;
13     else
14         f=1;
15     end
16 end
```

Listing 3: Matlab script for function η^2

```
1  % error indicator "eta"
2  function eta2 = my_eta2(x, xi_biss, Δ)
3  N = length(x);
4  eta2 = zeros(N,1);
5  for i = 1:N-1
6  h = x(i+1)-x(i);
7
8  eta2(i) = eta2(i)+ 1/2*h^3*((my_f(x(i))+Δ*xi_biss(i)).^2 +(my_f(x(i+1))+Δ*xi_biss(i+1)).^2);
9
10 end
11 end
```

Listing 4: Matlab script for load vector B

```
1  function B=my_load_vector_assembler(x,Δ)
2  %
3  % Returns the assembled load vector b.
4  % Input is a vector x of node coords.
5
6  N = length(x) - 1;
7  B = zeros(N+1, 1);
8  for i = 1:N
9  h = x(i+1) - x(i);
10 n = [i i+1];
11 B(n) = B(n) +(1/Δ)*[my_f(x(i)); my_f(x(i+1))]*h/2;
12 end
13 end
```

Listing 5: Matlab script for Stiffness matrix A

```
function A=my_stiffness_matrix_assembler(x)
%
% Returns the assembled stiffness matrix A.
% Input is a vector x of node coords.
%
N = length(x) - 1;   % number of elements
A = zeros(N+1, N+1); % initialize stiffnes matrix to zero
for i = 1:N          % loop over elements
  h = x(i+1) - x(i); % element length
  n = [i i+1];       % nodes
  A(n,n) = A(n,n) + [1 -1; -1 1]/h; % assemble element stiffness
end
A(1,1) = 1.e+6; % adjust for BC
A(N+1,N+1) = 1.e+6;
end
```

Listing 6: Matlab script for Mass matrix M

```
function M = my_mass_matrix_assembler(x)
    N = length(x)-1; % number of subintervals
    M = zeros(N+1,N+1); % allocate mass matrix
    for i = 1:N % loop over subintervals
        h = x(i+1) - x(i); % interval length
        M(i,i) = M(i,i) + h/3; % add h/3 to M(i,i)
        M(i,i+1) = M(i,i+1) + h/6;
        M(i+1,i) = M(i+1,i) + h/6;
        M(i+1,i+1) = M(i+1,i+1) + h/3;
    end
end
```

Part B

Listing 7: Main script for part B1

```
clear all;
geometry = @circleg;
h = [1/2 1/4 1/8 1/16 1/32];
EnE = zeros(length(h),1);

for i = 1:length(h)
    hmax = h(i);
    [p,e,t] = initmesh(geometry ,'hmax',hmax);
    [A,M,b] = assemble(p,t);   %stiffness matrix

    u_e = @sin;
    x1 = p(1,:);
    x2 = p(2,:);
    u_exact = u_e(2*pi*x1).*u_e(2*pi*x2);

    for k = 1:length(e(1,:))
        x11 = p(1,e(1,k));
        x22 = p(2,e(1,k));
        RHS(k,1) = u_e(2*pi*x11).*u_e(2*pi*x22);
    end

    I = eye(length(p));          %construct the identity matrix
    A(e(1,:),:) = I(e(1,:),:);  %replace the rows corresponding
                                    %to the boundary nodes by
                                        % corresponding rows of I
    b(e(1,:)) = RHS(:,1);        %put the boundary value into the RHS

    u_h = A\b;                   %solving linear eq.

    err=u_exact'-u_h;             %Error
    EnE(i,1)=sqrt(err'*A*err);  %Energy norm

    figure();
    pdeplot(p,[],t,'XYData',u_h,'XYStyle','interp',...
         'ZData',u_h,'ZStyle','continuous',...
         'ColorBar','on', 'ColorMap','default');
    %pdesurf(p,t,u_h)
    caption =sprintf('Solution for hmax=%.4f',hmax);
    title(caption, 'Fontsize', 20)
end
figure(6)
    P = polyfit(log(h)',log(EnE),1);
    P = P(1);       %convergence rate

loglog(EnE,h, 'b', h.^P,h, 'r')
title('hmax versus hpmax')
legend('hmax', 'hpmax')
```

Listing 8: Main script for part B2

```
clear all;
T = 2;
h = [1/5 1/20 1/40];
alpha = 4;
Δ = 0.01;
geometry = @circleg;

for i = 1:length(h)
```

```
9
10      hmax = h(i);
11      timestep = 0.1*hmax;
12      tvec = linspace(0,T, round(T/timestep));
13      k = tvec(2)-tvec(1);
14
15
16      [p,e,t] = initmesh(geometry ,'hmax',hmax);
17
18      [A,M,b] = assemble(p,t);   %stiffness matrix A and mass matrix M
19
20      for j = 1:length(p)
21          u_0(j,1) = 1+20*rand();
22      end
23      u_former = u_0;
24
25      for l = 1:length(tvec)-1
26          LHS = (M+(Δ/2)*k.*A);
27          RHS = (M-(Δ/2)*k.*A)*u_former(:,l)...
28              -k*M*(u_former(:,l)./(u_former(:,l)+alpha)-u_former(:,l)+(u_former(:,l)).^2);
29          u_h = LHS\(RHS);
30          u_former(:,l+1) = u_h;
31
32     % Population rates for every timestep
33          for K = 1:size(t,2)
34              nodes = t(1:3,K);
35              x = p(1,nodes);
36              y = p(2,nodes);
37              area_K = polyarea(x,y);
38              Mp(K) = (u_h(nodes(1))+ u_h(nodes(2))... %Trapezoidal rule
39                   +u_h(nodes(3)))/3*area_K;
40          end
41          Mprey(l) = sum(Mp);
42      end
43  figure()
44  %pdesurf(p,t,u_h)
45  pdeplot(p,[],t,'XYData',u_h,'XYStyle','interp',...
46           'ZData',u_h,'ZStyle','continuous',...
47           'ColorBar','on', 'ColorMap','default');
48
49  caption =sprintf('Solution for hmax=%.4f', hmax);
50  title(caption, 'Fontsize', 20)
51
52  figure(3)
53  plot(tvec(2:end),Mprey)
54  hold on;
55  caption =sprintf('Population rates');
56  legend('hmax=1/5', 'hmax = 1/20', 'hmax = 1/40')
57  title(caption, 'Fontsize', 20)
58  end
```

Listing 9: Matlab script for function f(x1,x2) for part B1

```
1  function f = func(x1,x2)
2
3      f = 8*pi^2*sin(2*pi*x1).*sin(2*pi*x2);
4  end
```

Listing 10: Matlab script for assembling the mass matrix M, the stiffness matrix A and the load vector b

```
function [A,M,Bvec] = assemble(p,t)
N = size(p,2);
A = sparse(N,N);
M = sparse(N,N); % allocate mass matrix
Bvec = zeros(N,1);

% assemble stiffness matrix A, mass matrix M and load vector b.
for K = 1:size(t,2)
    nodes = t(1:3,K);
    x1 = p(1,nodes);
    x2 = p(2,nodes);

    area_K = polyarea(x1,x2);
    b=[x2(2)-x2(3); x2(3)-x2(1); x2(1)-x2(2)]/2/area_K;
    c=[x1(3)-x1(2); x1(1)-x1(3); x1(2)-x1(1)]/2/area_K;

    AK = (b*b'+c*c')*area_K;                % element stiffness matrix
    A(nodes,nodes) = A(nodes,nodes) + AK;

    MK = [2 1 1;
        1 2 1;
        1 1 2]/12*area_K;                   % element mass matrix
    M(nodes,nodes) = M(nodes,nodes) + MK; % add element masses to M

        bK = [func(x1(1),x2(1));
        func(x1(2),x2(2));
        func(x1(3),x2(3))]/3*area_K;

    Bvec(nodes) = Bvec(nodes) + bK;
end
end
```

Part C

Listing 11: Python script for prolem 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 14 17:04:05 2020

@author: amandaseger
"""
from fenics import *
from dolfin import *
import numpy as np

##############################################################################################################
##

#class DirichletBoundary(SubDomain):
 #  def inside(self, x, on_boundary):
  #     return on_boundary
#####################################################

# Create mesh and define function space
mesh = Mesh("circle.xml")
# Construct the finite element space
V = VectorFunctionSpace(mesh, 'P', 1)

# Define parameters:
T = 150
dt = 0.5
alpha = 0.4
beta = 2
gamma = 0.8
Δ1 = 1
Δ2 = 1

# Class representing the intial conditions
class InitialConditions(UserExpression):
    def eval(self, values, x):
        values[0] = (4/15)-2*pow(10,-7)*(x[0]-0.1*x[1]-225)*(x[0]-0.1*x[1]-675)
        values[1] = (22/45)-3*pow(10,-5)*(x[0]-450)-1.2*pow(10,-4)*(x[1]-150)
    def value_shape(self):
        return (2,)



# Define initial condition
indata = InitialConditions(degree=2)
u0 = Function(V)
u0 = interpolate(indata , V)

# Test and trial functions
u = TrialFunction(V)
v = TestFunction(V)


# Create bilinear and linear forms
a0 = u[0]*v[0]*dx + 1/2*dt*Δ1*inner(grad(u[0]), grad(v[0]))*dx -1/2*dt*u[0]*v[0]*dx
a1 = u[1]*v[1]*dx + 1/2*dt*Δ2*inner(grad(u[1]), grad(v[1]))*dx + gamma*dt*1/2*u[1]*v[1]*dx

L0 = u0[0]*v[0]*dx -(1/2*dt*Δ1*inner(grad(u0[0]), grad(v[0]))*dx) -\
    (dt*((u0[0]*u0[1])/(u0[0]+alpha)+u0[0]*u0[0])*v[0]*dx) + 1/2*dt*u0[0]*v[0]*dx
```

```
60
61  L1 = u0[1]*v[1]*dx -(1/2*dt*Δ2*inner(grad(u0[1]), grad(v[1]))*dx) -\
62      (dt*(-(beta*u0[1]*u0[0])/(u0[0]+alpha))*v[1]*dx) - gamma*1/2*dt*u0[1]*v[1]*dx
63
64  a = a0+a1
65  L = L0+L1
66
67  #Set upp boundary condition
68  bc = [] #Neumann
69
70  #solve(a==L, u, bc) #Could just solve directly
71
72  # Assemble matrix
73  #A = assemble(a)
74  #b = assemble(L)
75  #bc.apply(A)
76
77  # Set an output file
78  out_file = File("results/solution.pvd", "compressed")
79
80  # Set initial condition
81  u = Function(V)
82  u.assign(u0)
83
84  t = 0
85
86  out_file << (u,t)
87
88  u_initial = Function(V)
89  u_initial.assign(u0)
90
91  t_save = 0
92  num_samples = 20
93
94  #Initial population rate
95  pop_u = []
96  pop_v = []
97  # Define the integrals
98  M0 = u0[0] * dx
99  M1 = u0[1] * dx
100 # compute the functional
101 population_u = assemble(M0)
102 population_v = assemble(M1)
103
104 time = []
105 # Time-stepping
106 while t < (T+1):
107 # assign u0
108     time.append(t)
109     u0.assign(u)
110
111     # Assemble vector and apply boundary conditions
112
113     A = assemble(a)
114     b = assemble(L)
115     #bc.apply(b)
116
117     # Solve linear system
118     #solve(A, u.vector(), b, "bicgstab", "default")
119     #solve(A, u.vector(), b, "gmres", "ilus")
120     solve(A, u.vector(), b, "lu")
121     #solve(a==L,u,bc)
122
123     t_save += dt
124
```

```
    # compute the functional
    population_u = assemble(M0)
    population_v = assemble(M1)


    #if t_save > T / num_samples  or t≥ T-dt:
     #   print("Saving solution")
        # Plot solution
        # plot(u)
    if t==50:
        print('Saving at 50')
        plot(u)
        out_file << (u,t)

    if t==100:
        print('Saving at 100')
        plot(u)
        out_file << (u,t)

    if t==150:
        print('Saving at 150')
        plot(u)
        out_file << (u,t)

        #t_save = 0

    # Move to next interval and adjust boundary condition
    t += dt
np.savetxt('Mprey.dat',pop_u)
np.savetxt('Mpredator.dat',pop_v)
np.savetxt('time.dat',time)
```

Listing 12: Python script for prolem 2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Dec 19 16:19:32 2020

@author: amandaseger
"""
from fenics import *
from dolfin import *
import numpy as np
import random

################################################################################

#class DirichletBoundary(SubDomain):
 #  def inside(self, x, on_boundary):
  #     return on_boundary
################################################

# Create mesh and define function space
mesh = Mesh("circle_finest.xml")
# Construct the finite element space
V = VectorFunctionSpace(mesh, 'P', 1)

# Define parameters:
T = 1000
dt = 0.5
alpha = 0.4
beta = 2
```

```
gamma = 0.8
Δ1 = 1
Δ2 = 1

# Class representing the intial conditions
class InitialConditions(UserExpression):
    def eval(self, values, x):
        r = random.uniform(0, 1)
        values[0] = (1/2)*(1-r)
        values[1] = (1/4)+ ((1/2)*r)
    def value_shape(self):
        return (2,)


# Define initial condition
indata = InitialConditions(degree=2)
u0 = Function(V)
u0 = interpolate(indata , V)

# Test and trial functions
u = TrialFunction(V)
v = TestFunction(V)


# Create bilinear and linear forms
a0 = u[0]*v[0]*dx + 1/2*dt*Δ1*inner(grad(u[0]), grad(v[0]))*dx -1/2*dt*u[0]*v[0]*dx
a1 = u[1]*v[1]*dx + 1/2*dt*Δ2*inner(grad(u[1]), grad(v[1]))*dx + gamma*dt*1/2*u[1]*v[1]*dx

L0 = u0[0]*v[0]*dx -(1/2*dt*Δ1*inner(grad(u0[0]), grad(v[0]))*dx) -\
    (dt*((u0[0]*u0[1])/(u0[0]+alpha)+u0[0]*u0[0])*v[0]*dx) + 1/2*dt*u0[0]*v[0]*dx

L1 = u0[1]*v[1]*dx -(1/2*dt*Δ2*inner(grad(u0[1]), grad(v[1]))*dx) -\
    (dt*(-(beta*u0[1]*u0[0])/(u0[0]+alpha))*v[1]*dx) - gamma*1/2*dt*u0[1]*v[1]*dx

a = a0+a1
L = L0+L1

#Set upp boundary condition
bc = [] #Neumann

#solve(a==L, u, bc) #Could just solve directly

# Assemble matrix
#A = assemble(a)
#b = assemble(L)
#bc.apply(A)

# Set an output file
out_file = File("resultsPartC2finest/C2solution.pvd", "compressed")

# Set initial condition
u = Function(V)
u.assign(u0)

t = 0

out_file << (u,t)

u_initial = Function(V)
u_initial.assign(u0)

t_save = 0
num_samples = 20

#Initial population rate
```

```
Mprey= []
Mpred = []
# Define the integrals
M0 = u[0] * dx
M1 = u[1] * dx

time = []
# Time-stepping
while t ≤ (T):
# assign u0
    time.append(t)
    u0.assign(u)

    # Assemble vector and apply boundary conditions

    A = assemble(a)
    b = assemble(L)
    #bc.apply(b)

    # Solve linear system
    #solve(A, u.vector(), b, "bicgstab", "default")
    #solve(A, u.vector(), b, "gmres", "ilus")
    solve(A, u.vector(), b, "lu")
    #solve(a==L,u,bc)

    t_save += dt
    #Calculate population rate
    # compute the functional
    population_u = assemble(M0)
    population_v = assemble(M1)


    Mprey.append(population_u)
    Mpred.append(population_v)

    if t==50:
        print('Saving at 50')
        plot(u)
        out_file << (u,t)

    if t==100:
        print('Saving at 100')
        plot(u)
        out_file << (u,t)

    if t==500:
        print('Saving at 500')
        plot(u)
        out_file << (u,t)
    if t == 1000:
        print('Saving at 1000')
        plot(u)
        out_file << (u, t)

    # Move to next interval and adjust boundary condition
    t += dt
np.savetxt('C2Mprey.txt',Mprey)
np.savetxt('C2Mpred.txt',Mpred)
np.savetxt('C2time.txt',time)
```