UPPSALA
UNIVERSITET

Applied Finite Element Methods

Prey-Predator Dynamics

Authors:

Matilda Tiberg

matilda.tiberg.5927@student.uu.se

Uppsala

January 8, 2021

Contents

1 Introduction

1.1 Prey-Predator Model

In this project, the two-dimensional (2D) Prey-Predator dynamic model will be studied by implementing the *Galerkin Finite Element Method* (GFEM). The most common Prey-Predator model is the Lotka-Volterra model, which consists of a system of *ordinary differential equations* (ODE:s), and plays an important role in the understanding the dynamics between species situated in different levels of the food chain. In this project however, a system of *partial differential equations* (PDE:s), where reaction-diffusion PDE:s are used, seen in equation (1).

$$\begin{cases} \partial_t u_1 - u_1(1-u_1) + \frac{u_1 u_2}{u_1+\alpha} - \delta_1 \nabla^2 u_1 = f, & (\boldsymbol{x}, t) \in \Omega \times (0, T] \\ \partial_t u_2 + \gamma u_2 - \beta \frac{u_1 u_2}{u_1+\alpha} - \delta_2 \nabla^2 u_2 = g, & (\boldsymbol{x}, t) \in \Omega \times (0, T] \\ u_1(\boldsymbol{x}, 0) = u_{1,0}(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \\ u_2(\boldsymbol{x}, 0) = u_{2,0}(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \end{cases} \tag{1}$$

Here, u_1 denotes the prey and u_2 the predators. T denotes the time, and the final time is $T > 0$. The domain Ω is bounded and $\Omega \in \mathbb{R}^d$, $d = 1, 2, 3$. The source terms $f = f(\boldsymbol{x}, t)$ and $g = g(\boldsymbol{x}, t)$, as well as the initial conditions $u_{1,0}(\boldsymbol{x})$ and $u_{2,0}(\boldsymbol{x})$, are given. The constants $\alpha, \beta, \gamma, \delta_1$ and δ_2 are given as well.

The PDE:s in equation (1) describes the behaviors of both populations, where the first PDE describes the prey, and the second PDE describes the predators. Both the populations depends on each other, for example: The predators eats the prey. The prey population will decrease, which means that there will be less food available for the predators. This will lead to a decrease of the predator population, which will lead to an increase of the prey etc. Both PDE:s contains 4 terms that are similar: term 1,3,4 and the source terms. Term 1 describes the changes in time, term 3 describes the consumption of prey per predator, and term 4 describes the diffusion of respective population. The second term, $u_1(1-u_1)$ in the first PDE denotes the prey birth and death rates, and the term γu_2 in the second PDE denotes the predator death rate. The total population of the predators and prey can be calculated by computing the integral over the domain Ω, see equation (2).

$$M_{prey}(t) = \int_\Omega u_1(\boldsymbol{x}, t) d\boldsymbol{x}, \qquad M_{predator}(t) = \int_\Omega u_2(\boldsymbol{x}, t) d\boldsymbol{x} \tag{2}$$

1.2 Project Description

The general flow scheme for GFEM used in this project is given by $(PDE) \rightarrow (WF) \rightarrow (GFEM) \rightarrow (LS)$, where (WF) is the weak formulation, and (LS) is the linear system that is to be solved using MATLAB's backslash function. In the cases when the problems are time-dependent, the Finite Difference Method (FDM) Crank-Nicholson method is implemented to fully discretize the GFEM formulation and to solve the problem.

The project consists of three parts, where equation (1) is broken down into different configurations and analyzed. In **Part A**, the one-dimensional (1D) stationary diffusion problem is considered. **Part B** discusses the 2D time-dependent scalar reaction-diffusion problem, where only the PDE considering the prey is dynamic, while the predator population is considered to be constant. Both Part A and Part B are solved by using MATLAB. **Part C** solves the full system using Python and the FEniCS Project (which is a free open-source software used for solving PDE:s using FEM).

2 Part A - 1D Simplification

In this problem, the ODE stated in equation (3) will be considered, where $x \in [-1, 1]$. The forcing function $f(x)$ is given in (4). The goal was to solve the problem with FEM, and then to estimate that occurred due to the numerical properties. The *a posteriori* error estimation was used, and the mesh was refined locally until desired error tolerance was reached.

$$\begin{cases} -\delta u''(x) = f(x), & x \in (-1, 1) \\ u(-1) = u(1) = 0 \end{cases} \tag{3}$$

$$\begin{cases} -5, & \text{if } |x| \leq 0.1 \\ 25, & \text{if } |0.2 - |x|| \leq 0.1 \\ -30, & \text{if } |0.6 - |x|| \leq 0.1 \\ 20, & \text{if } |0.9 - |x|| \leq 0.2 \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

The MATLAB code used in this part can be found in Appendix A.

2.1 Problem A.1

In this problem, the a posteriori error estimate was derived. u_h is the discrete approximation of the solution to equation (3), defined on a mesh $\tau_h : -1 = x_0 < x_1 < ... < x_N = 1$. The mesh size h is defined by $h_i = x_{i+1} - x_i$.

The Galerkin Orthogonality (G.O) is given as $\int_0^1 (u - u_h)'v'dx = 0$, where $(u - u_h)' = e'$ and e is the error. The energy norm and the L^2-norm is related by $||e||_E^2 = ||e'|| = \int_0^1 e'e'dx$. The exact solution $u \in V_0$, and the numerical solution $u_h \in V_{h,0}$ (which is the FE space). This means that the error vector does not belong to the space $V_{h,0}$, and thus an interpolant $\pi_h e \in V_{h,0}$ needs to be introduced.

$$||e||_E^2 = ||e'||^2 = \int_0^1 e'e'dx = \int_0^1 e'e'dx - \int_0^1 e'(\pi_h e)'dx \tag{5}$$

By applying the Galerkin orthogonality on equation (5), the two integrals can be written under the same integral sign, and thus the expression below is received. The next step is to solve the integral by using partial integration (P.I), however, the function $\pi_h e$ is piecewise linear, and is not differentiable at every point x. Therefore, it needs to be integrated over the subintervals given by h_i. The expression is then simplified by using $e(x_i) - \pi_h e(x_i) = 0$. This comes from that the FEM calculates the function value exactly at every node x_i.

$$\begin{aligned} \int_0^2 e'(e - (\pi_h e)'dx &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} e'(e - \pi_h e)'dx = [\text{Using P.I}] \\ &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} -e''(e - \pi_h e)dx + e'(x_i)(e(x_i) - \pi_h e(x_i)) - e'(x_{i-1})(e(x_{i-1}) - \pi_h e(x_{i-1})) \\ &= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} -(u'' - u_h'')(e - \pi_h e)dx \end{aligned}$$

Since we have the equation (3), u'' can be rewritten as $u'' = -\frac{1}{\delta} f$. This can be used to define the residual term $-R(u_h) = -\frac{1}{\delta} f - u_h''$. The Cauchy-Schwartz inequality (CS) and the residual term is used to rewrite

the equation above. After that the CS is used, the interpolation estimate $||u - \pi_h u|| \leq Ch||u'||$ is introduced, and then CS is used again.

$$\begin{aligned}\sum_{i=1}^{n}\int_{xi-1}^{x_i} R(u_h)(e-\pi_h e)dx &\leq \sum_{i=1}^{n}||R(u_h)||_{L^2(I_i)}||e-\pi_h e||_{L^2(I_i)} \\ &\leq \sum_{i=1}^{n}||R(u_h)||_{L^2(I_i)}Ch||e'||_{L^2(I_i)} \\ &\leq \sqrt{\sum_{i=1}^{n}\left(Ch_i||R(u_h)||_{L^2(I_i)}\right)^2}\sqrt{\sum_{i=1}^{n}||e'||^2_{L^2(I_i)}}\end{aligned}$$

The last sum is just the L^2-norm over the whole interval I. After rewriting $\eta_i^2 = \delta^2(h_i||R(u_h)||_{L^2(I_i)})^2$, we can then combine above equation with equation (5), which results in the a posteriori error estimation, seen in equation (6). Here, δ^2 is included in the constant C.

$$||e||_E = ||u - u_h||_E \leq C\sqrt{\sum_{i=1}^{n}\eta_i^2} \tag{6}$$

2.2 Problem A.2

In this part, u_h was estimated by the GFEM, and then the mesh was refined by using the a posteriori error estimate (derived in Problem A.1), until a threshold of $tol = 10^{-3}$ was reached, or that the amount of nodes exceeded $N > 10^4$. The GFEM of equation (3) is expressed as (7), where A is the stiffness matrix, defined by $A = \int_a^b \varphi_i'\varphi_j' dx$, and b is the load vector defined by $b = \int_a^b f\varphi_j dx$. The vector $\boldsymbol{\xi}$ represents the discrete solution u_h.

$$A\boldsymbol{\xi} = b \tag{7}$$

The residual is calculated by using u_h'', hence the Laplacian of u_h is needed. It is given by $u_h'' = -M^{-1}A\boldsymbol{\xi}$, where M is the mass matrix given by $M = \int_a^b \varphi_i\varphi_j dx$. From this, the expression for the residual becomes equation (8). The error estimate η_i^2, derived in previous problem, is calculated by using the the trapezoidal rule, see equation (9)

$$R_h(x) = f + u_h'' = f - M^{-1}A\xi \tag{8}$$

$$\eta_i^2 = h^3\frac{(R_h(i)^2 + R_h(i+1))^2}{2} \tag{9}$$

The results for the stated problem are shown in Figure 1. When the error estimate η^2 reaches tol, the residual R_h is very close to zero for most x_i, but there are some prominent peaks. Both the Error Estimate and Mesh Distribution has their peaks situated as approximately the same x_i as the peaks of the residual. This is due to that and increased residual results in an increased error. The mesh will be more refined where η^2 is big, thus containing more nodes. The peaks corresponds approximately to where the solution u_h reaches its local minimums and local maximums. The simulation started with $N = 12$ nodes, and ended with $N = 79$ nodes.

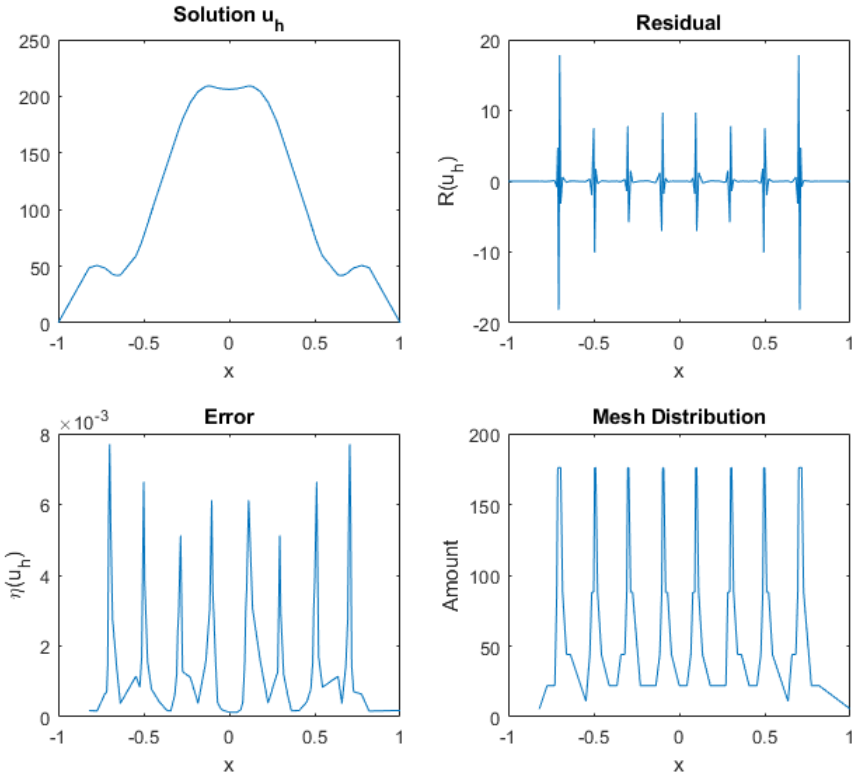

Figure 1: *The result for the FEM with adaptive mesh refinement by a posteriori error estimation. Here, the condition $\eta^2 < 10^{-3}$ was met. The peaks of the error and the mesh distribution are centered to around the peaks of the residual.*

If the condition $\sqrt{\eta^2} < 10^{-3}$ was used instead, the peaks became more gathered. Here, the error tolerance was met at $N = 159$ nodes. The results can be seen in Figure 2.

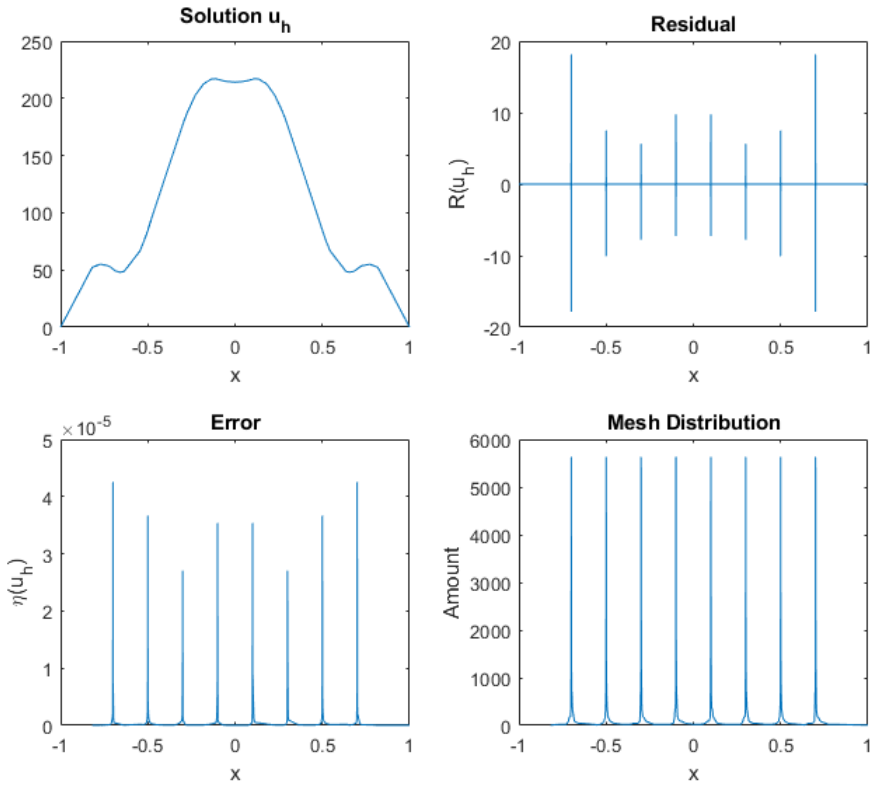

Figure 2: *In this case,* $\sqrt{(\eta^2)} < 10^{-3}$ *was used as stopping condition in the while-loop instead. The peaks were thinner in both the error estimate, and the mesh distribution, and no visible oscillations in the residual.*

3 Part B - 2D Convergence Analysis

In this part, the 2D scalar version of the predator-pray model is considered. The problem is implemented with the GFEM in MATLAB. The domain Ω that is used throughout this part consists of a disc of radius 1, and the coordinates on the disc is given by $\boldsymbol{x} = (x_1, x_2)$. The codes used can be found in Appendix B.

3.1 Problem B.1

3.1.1 Theory

In this problem, a simplified version of the prey-predator model is given, and can be seen in equation (10). It is the Poisson equation, which is the stationary diffusion equation with a source term. The boundary condition is given by the exact solution $u_{exact}(\boldsymbol{x}) = sin(2\pi x_1)sin(2\pi x_2)$. The right hand side function f is given by $f(\boldsymbol{x}) = 8\pi^2 sin(2\pi x_1)sin(2\pi x_2)$. The task is to use GFEM to calculate a solution, the energy norm of the errors and the convergence of the GFEM for this problem.

$$\begin{cases} -\nabla^2 u(\boldsymbol{x}) = f(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) = u_{exact}(\boldsymbol{x}), & \boldsymbol{x} \in \partial\Omega \end{cases} \tag{10}$$

The energy norm of the error is given by equation (11), where u is the exact solution, and u_h is the discrete solution. The energy norm is implemented in MATLAB by `EnE = sqrt((u-u_h)'*A*(u-u_h);`, where A is the stiffness matrix. The convergence is determined in two ways: by fitting a first order plynomial to the EnE-data points by using MATLAB's function `polyfit`. The second way to calculate the convergence is by calculating the slope of EnE between each h_{max} using the logarithmic scale, see equation (12), where the maximal mesh-sizes are given by $h_{max} = [\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}]$.

$$||u - u_h||_E = \left(\int_{\mathcal{B}} (\nabla u - \nabla u_h) \cdot (\nabla u - \nabla u_h) d\boldsymbol{x} \right)^{1/2} \tag{11}$$

$$q_i = \frac{\log(EnE_i - EnE_{i-1})}{\log(h_{max,i} - h_{max,i-1})}, \qquad i = 2, 3, 4, 5 \tag{12}$$

3.1.2 Results

The calculated errors and convergence q can be seen in Table 1. It can be seen that the error is relatively big (reaches an order 10^{-1}. The convergence q states that the convergence reaches a rate of around 1.2, which is reasonable, since GFEM has a convergence proportional to $\mathcal{O}(h_{max})$ (which means a convergence rate equal to 1). As can be seen, q attains bigger values in the beginning, but since the meshes are very coarse at those values, those results can be neglected.

Table 1: *he calculated error EnE for each H_{max} and the computed convergence q. EnE is calculated as the energy norm, and q as the slope.*

h_{max}	**EnE**	**q**
1/2	3.4862	-
1/4	2.0831	0.7429
1/8	0.5260	1.9856
1/16	0.1946	1.4346
1/32	0.0864	1.1710

In Figure 3, the error is plotted against h_{max}, both attributes in a logarithmic scale. As stated in previous table, the error decreases together with a smaller grid size h_{max}. When fitting a first degree polynomial to

the data points, it gives a convergence of $P = 1.4$, which also is plotted in the same figure as h_{max}^{P}. However, this method accounts for all the error data points, and will thus result in a somewhat faulty convergence rate. Hence, it is better to use the convergence q from Table 1 to determine the convergence of the method.

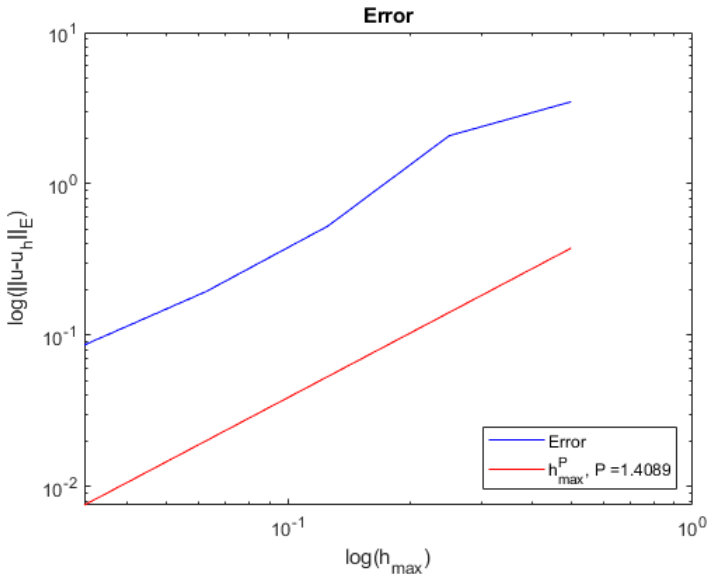

Figure 3: *The calculated energy norm of the error for Problem B.1. Here compared to a line of slope* $h_{max}^{1.4}$.

The results for the calculated u_h for the coarsest mesh ($h_{max} = 1/2$) and the finest mesh ($h_{max} = 1/32$ can be seen in Figure 4 and 5. As the value of h_{max} increased, the resulting mesh became smoother and smoother. The similarities between the two meshes was that they attained a similar form, however, the finer mesh had more peaks that are evenly distributed over the surface.

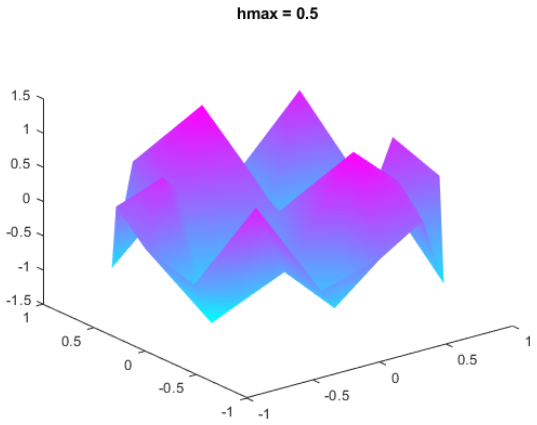

Figure 4: *The result of the calculated* u_h *for the coarsest mesh, given by* $h_{max} = 1/2$.

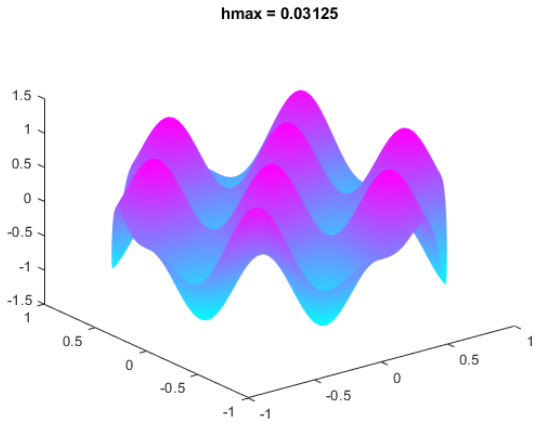

Figure 5: *The result of the calculated* u_h *for the finest mesh, given by* $h_{max} = 1/32$.

3.2 Problem B.2

In this problem, the first PDE in equation (1) is considered, while the predator population density is assumed to be constant, (in this case $u_2(\boldsymbol{x}, t) = 1$. Thus, the Predator-Prey model will be given by the time-dependent scalar equation (13). In this problem, the source term $f(\boldsymbol{x}, t) = 0$. Additionally, $\delta = 0.01$, $\alpha = 4$ and $\omega(\boldsymbol{x}) \in [0, 1]$ for all $\boldsymbol{x} \in \Omega$, where $\omega(\boldsymbol{x})$ is randomly generated. The term $u_1 = u$ in the following equations for simplicity.

$$\begin{cases} \partial_t u - u(1-u) + \frac{u}{u+\alpha} - \delta \nabla^2 u = f & (\boldsymbol{x}, t) \in \Omega \times (0, T] \\ \partial_n u(\boldsymbol{x}, t) = 0, & \boldsymbol{x} \in \partial\Omega, t \in (0, T] \\ u(\boldsymbol{x}, 0) = 1 + 20\omega(\boldsymbol{x}), & \boldsymbol{x} \in \Omega \end{cases} \tag{13}$$

The task in the problem was to implement GFEM to discretize the PDE, described in (13), together with the finite difference method *Crank-Nicolson method* to discretize time, which has a convergence rate of $\mathcal{O}(k^2)$, where k is the time step.

3.2.1 Derivation of GFEM

In order to derive the GFEM formulation, the PDE in equation (13) first needs to be rewritten on weak form before it can be discretized. Firstly, the PDE is multiplied by a test function v. The definition of the test space is given by $V_0 := \{v(\boldsymbol{x}, t) : ||v(\cdot, t)|| \leq \infty,\ ||\nabla v(\cdot, t)|| \leq \infty\}$. It is then integrated over the domain Ω, as seen in equation (14).

$$\int_\Omega \frac{\partial u}{\partial t} v \, d\boldsymbol{x} - \int_\Omega u(1-u)v \, d\boldsymbol{x} + \int_\Omega \frac{u}{u+\alpha} v \, d\boldsymbol{x} - \delta \int_\Omega \nabla^2 u v \, d\boldsymbol{x} = 0 \tag{14}$$

Using the definition of Green's theorem, the term containing the Laplacian operator in above equation can be rewritten. Since the problem set up consists of homogeneous Neumann boundary conditions, one term disappears, see equation (15)

$$\delta \int_\Omega \nabla^2 u v \, d\boldsymbol{x} = -\delta \int_\Omega \nabla u \cdot \nabla v \, d\boldsymbol{x} + \delta \int_\Omega n \cdot \nabla u v \, d\boldsymbol{x} = [\text{Using B.C.}] = -\delta \int_\Omega \nabla u \cdot \nabla v \, d\boldsymbol{x} \tag{15}$$

By inserting (15) in (14), the weak formulation is achieved, where we want to find $u \in V_0$ such that equation (16) holds true.

$$\int_\Omega \frac{\partial u}{\partial t} v \, d\boldsymbol{x} - \int_\Omega u(1-u)v \, d\boldsymbol{x} + \int_\Omega \frac{u}{u+\alpha} v \, d\boldsymbol{x} + \delta \int_\Omega \nabla u \cdot \nabla v \, d\boldsymbol{x} = 0, \qquad \forall v \in V_0 \tag{16}$$

To find the discrete solution u_h, we assume that $\tau_h = \{\kappa\}$ is the triangulated mesh defined on the domain Ω, and $\{\mathcal{N}_h\}$ is all the interior nodes on said mesh. The discrete trial space $V_{h,0}$ is constructed, where $V_{h,0} \subset V_0$

$$V_{h,0} = \{v : v \in C^0(\Omega), v|_\kappa \in P_1(\kappa), \forall \kappa \subset \tau_h\}$$

$$\int_\Omega \frac{\partial u_h}{\partial t} v \, d\boldsymbol{x} + \int_\Omega \left(\frac{u_h}{u_h + \alpha} - u_h(1-u_h) \right) v \, d\boldsymbol{x} + \delta \int_\Omega \nabla u_h \cdot \nabla v \, d\boldsymbol{x} = 0, \qquad \forall v \in V_{h,0} \tag{17}$$

Since $u_h(1-u_h)$ will result in a non-linear term, the terms in the second integral is rewritten as the expression $S = \frac{u_h}{u_h - \alpha} - u_h(1-u_h)$. This is then approximated with linear interpolation $S \approx \pi_h S$, where $\pi_h S \in V_{h,0}$. By using the definition $u_h = \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j(t)\varphi_j(\boldsymbol{x})$, equation (17) is rewritten as equation (18), and the interpolation of S is rewritten as $\pi_h S(\boldsymbol{x}, t) = \sum_{\mathcal{N}_| \in \mathcal{N}_h} S_j(t)\varphi_j(\boldsymbol{x})$. By using the definition that $(u_h)_j(t) = \xi_j(t)$, the term $S_j(t)$ can be expressed as $S_j(t) = \frac{\xi_j(t)}{\xi_j(t)+\alpha} - \xi_j(t)(1-\xi_j(t))$.

$$
\begin{aligned}
0 &= \int_\Omega \frac{\partial u_h}{\partial t} v \, d\boldsymbol{x} + \int_\Omega \pi_h S(\boldsymbol{x}, t) v \, d\boldsymbol{x} + \delta \int_\Omega \nabla u_h \cdot \nabla v \, d\boldsymbol{x} \\
&= \sum_{\mathcal{N}_j \in \mathcal{N}_h} \frac{d\xi_j(t)}{dt} \int_\Omega \varphi_j \varphi_i \, d\boldsymbol{x} + \sum_{\mathcal{N}_j \in \mathcal{N}_h} S_j(t) \int_\Omega \varphi_j \varphi_i \, d\boldsymbol{x} + \delta \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_j(t) \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\boldsymbol{x} \\
&= M \frac{d\boldsymbol{\xi}}{dt} + MS + \delta A \boldsymbol{\xi}
\end{aligned} \tag{18}
$$

By using the previously stated definition of S, the semi-discrete GFEM formulation of the problem can be seen, together with its initial condition, in equation (19), where $\boldsymbol{\xi} = \boldsymbol{\xi}(t)$. M is the mass matrix, $M = \int \varphi_j \varphi_i d\boldsymbol{x}$, and A is the stiffness matrix, $A = \int \nabla \varphi_j \cdot \nabla \varphi_i d\boldsymbol{x}$.

$$
\begin{cases}
M \frac{d\boldsymbol{\xi}}{dt} + M\left(\frac{\boldsymbol{\xi}}{\boldsymbol{\xi} + \alpha} + \boldsymbol{\xi}(1 - \boldsymbol{\xi}) \right) + \delta A \boldsymbol{\xi} = 0, & t \in (0, T] \\
\boldsymbol{\xi}(0) = 1 + 20\omega(\boldsymbol{x}), & \boldsymbol{x} \in \Omega
\end{cases} \tag{19}
$$

The problem now consists of a system of ODEs of first degree, and a numerical ODE solver needs to be implemented. In this case, the Crank-Nicholson method is used, which is second-order accurate FDM, and is unconditionally stable for diffusion problems.

$$
\frac{a_i^{n+1} - a_i^n}{\Delta t} = \frac{1}{2}[F_i^{n+1} + F_i^n] \tag{20}
$$

Applying this on equation (19), we get the following derivation seen in equation (21), where k is our time step and $S(\boldsymbol{\xi}) = \frac{\boldsymbol{\xi}}{\boldsymbol{\xi} + \alpha} + \boldsymbol{\xi}(1 - \boldsymbol{\xi})$.

$$
\begin{aligned}
0 &= M \frac{\boldsymbol{\xi}^{n+1} - \boldsymbol{\xi}^n}{k} + MS(\boldsymbol{\xi}) + \delta A \frac{\boldsymbol{\xi}^{n+1} + \boldsymbol{\xi}^n}{2} \\
&= M(\boldsymbol{\xi}_i^{n+1} - \boldsymbol{\xi}_i^n) + kMS(\boldsymbol{\xi}) + \frac{\delta k}{2} A(\boldsymbol{\xi}^{n+1} + \boldsymbol{\xi}^n) \\
&= (M + \frac{\delta k}{2} A)\boldsymbol{\xi}^{n+1} - M\boldsymbol{\xi}^n + kMS(\boldsymbol{\xi}) + \frac{\delta k}{2} A \boldsymbol{\xi}^n
\end{aligned} \tag{21}
$$

The fully discrete problem can then be rewritten as equation (22), which is then used to iterate through time.

$$
\boldsymbol{\xi}^{n+1} = [M + \frac{\delta k}{2} A]^{-1} [M\boldsymbol{\xi}^n - kMS(\boldsymbol{\xi}) - \frac{\delta k}{2} A \boldsymbol{\xi}^n] \tag{22}
$$

3.2.2 Implementation and Results

In the simulation, a time step $k = 0.01 h_{max}$ was used. However, since the Crank-Nicholson method is unconditionally stable for this kind of problem, a bigger time step could have been used. The results of u_h for the different h_{max} are presented in Figure 6-8, where both the solution u_h at time $T = 0$ and the final u_h at time $T = 2$ are shown.

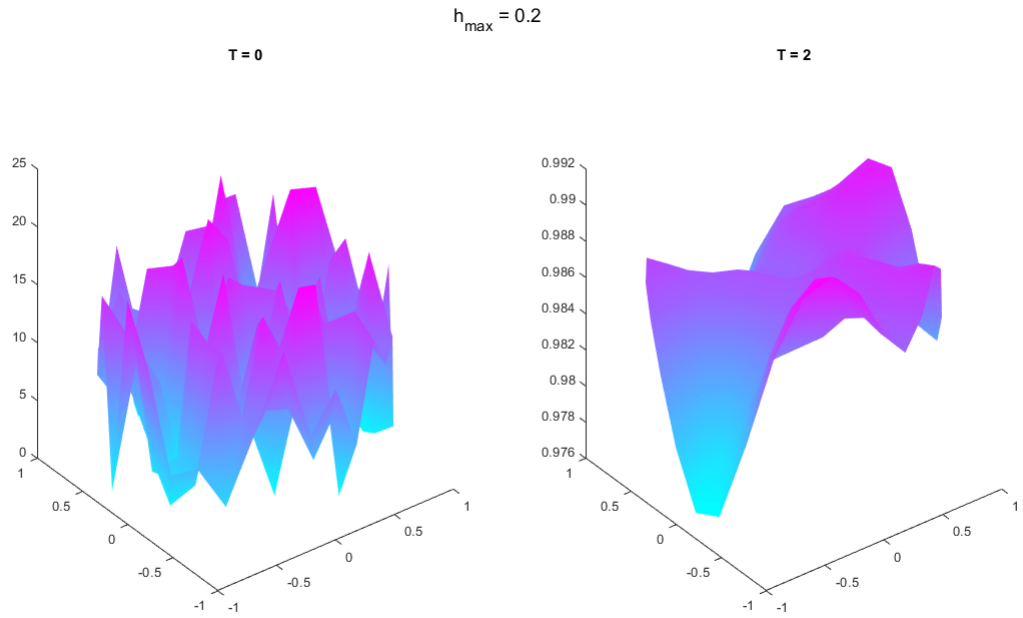

Figure 6: *The solution u_h at $T = 0$ and at the $T = 2$ for $h_{max} = 1/5$. Here, a time step of $k = 0.01h_{max}$ was used.*

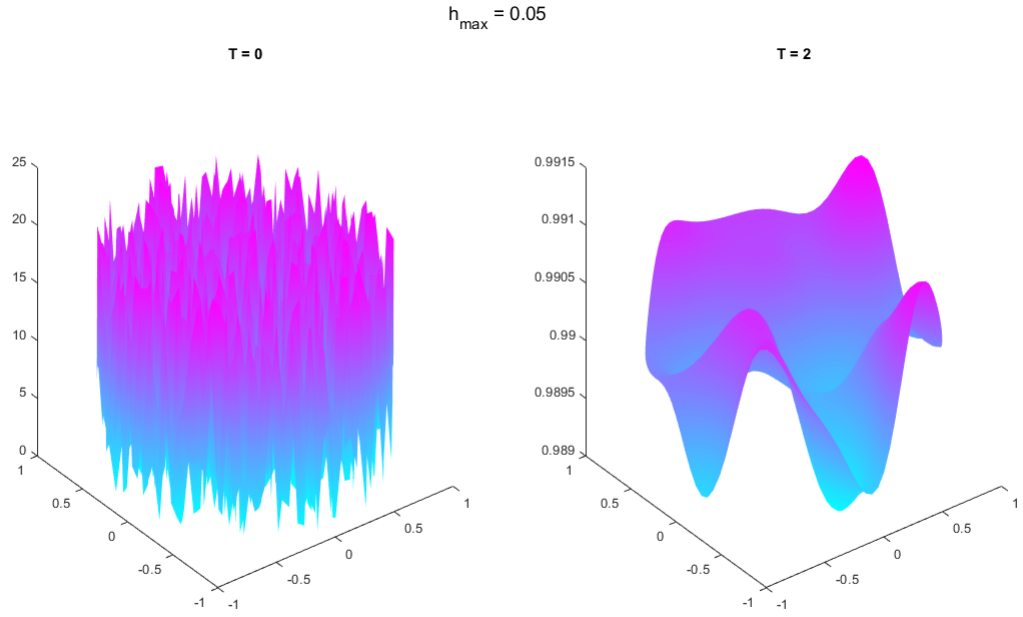

Figure 7: *The solution u_h at $T = 0$ and at the $T = 2$ for $h_{max} = 1/20$. Here, a time step of $k = 0.01h_{max}$ was used.*

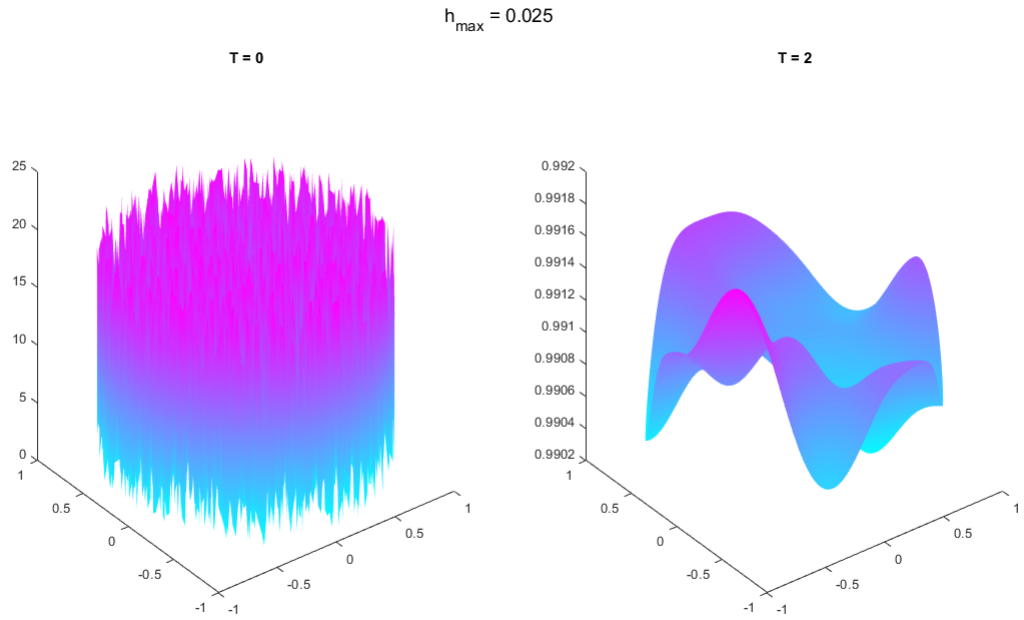

Figure 8: *The solution u_h at $T = 0$ and at the $T = 2$ for $h_{max} = 1/40$. Here, a time step of $k = 0.01h_{max}$ was used.*

Since the initial conditions were randomly generated, the solutions in Figure 6-8 attained different forms, but it can be seen that the solutions with a finest mesh is smoother than the solution with the coarsest mesh. If a bigger time step would have been used, the differences may have become more prominent. As seen, the initial condition $u_h(\boldsymbol{x}, 0)$ ranges from 0 to around 21 for every mesh size, while the solutions $u_h(\boldsymbol{x}, 2)$ is nearly flat in comparison. This is due to that the source term is set to 0, so the diffusion term in equation (13) becomes dominant. In terms of *"prey and predators"*, it can be described by that the amount of predators does not decrease when the prey decrease, thus the population of prey will continue to decrease, and not recover as it would had if the system of PDEs would have been considered (this since the constant predator density $v(\boldsymbol{x}, t) = 1$).

Calculating M_{prey} form equation (2), by using the trapezoidal rule seen in equation (23) (N_i is the nodal points in cell K), the total population of the prey can be plotted over time for the different meshes, see Figure 9. There it can be seen that the total population of the prey has an exponential decrease over time.

$$M_{prey}(t) = \int_K u_h(\boldsymbol{x}, t) d\boldsymbol{x} \approx \frac{|K|}{3} \sum_{i=1}^{3} u_h(N_i) \tag{23}$$

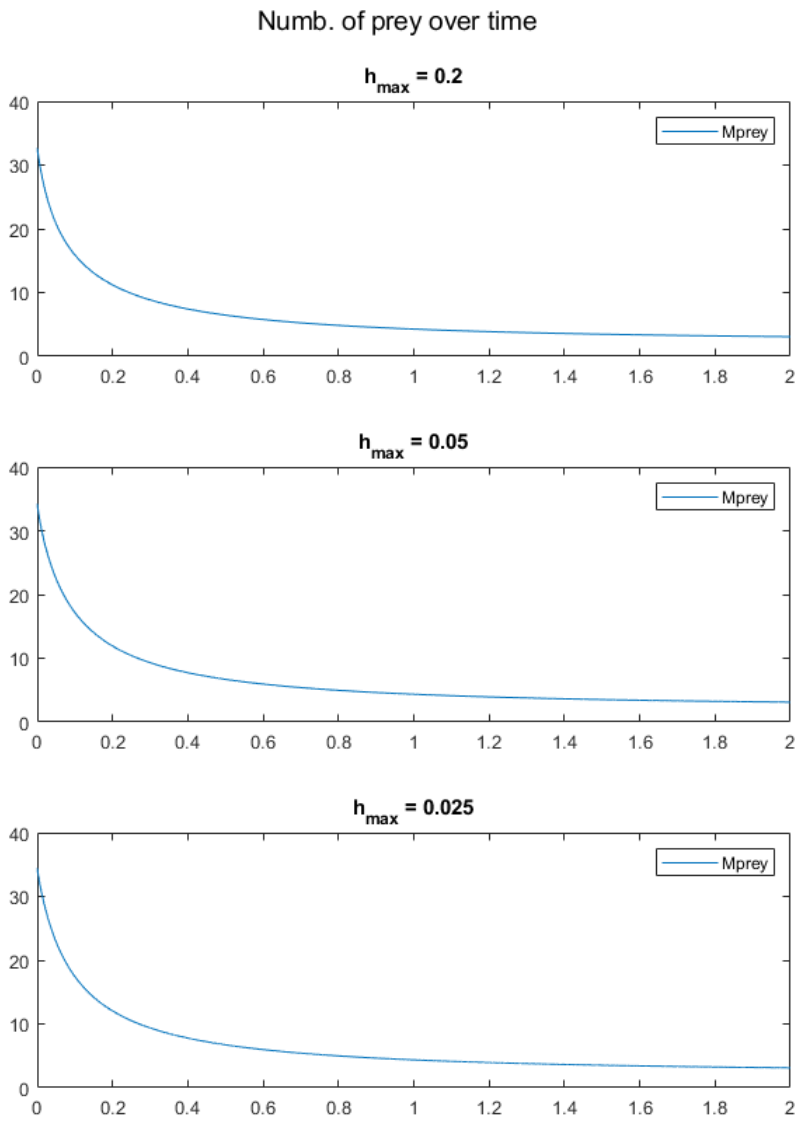

Figure 9: *How the number of prey where affected over time t. Constant number of predators was used.*

4 Part C - FEniCS implementation

In this part, equation (1) is computed by using the open source finite element solver **FEniCS** written in Python. The Python code was run on Linux as a subsystem to Windows. The computed results for u_1 and u_2 were then saved as a *pvd*-file for certain time steps and then rendered in **Paraview**. The source terms $f, g = 0$, and the other parameters used were: $\alpha = 0.4$, $\beta = 2$, $\delta_1 = \delta_2 = 1$ and $\gamma = 0.8$. Neumann conditions were used on the boundaries, given in equation (24). The initial conditions differs in both problems, and are stated in each subsection.

$$\begin{cases} \partial_n u_1(\boldsymbol{x}, t) = 0, & \forall \boldsymbol{x} \in \partial\Omega, t \in (0, T] \\ \partial_n u_2(\boldsymbol{x}, t) = 0, & \forall \boldsymbol{x} \in \partial\Omega, t \in (0, T] \end{cases} \tag{24}$$

4.1 Derivation of GFEM

Here, the derivation of the GFEM formulation will only be demonstrated on the PDE shown in equation (25)), since the derivation of the GFEM formulation for the other PDE in the system is executed in the same way.

$$\partial_t u_1 - u_1(1 - u_1) + \frac{u_1 u_2}{u_1 + \alpha} - \delta_1 \nabla^2 u_1 = 0 \tag{25}$$

Defining $\boldsymbol{u} = (u_1, u_2)^T$ and $\boldsymbol{v} = (v_1, v_2)^T$, the weak formulation should be formulated such that $\boldsymbol{u} \in W_0 := \{\boldsymbol{v}(\boldsymbol{x}, t) : \ \boldsymbol{v} \in H^1(\Omega), \ \forall t \in (0, T]\}$, where $H^1(\Omega)$ is a Hilbert space. From this, the weak formulation is derived as following:

$$\int_\Omega \partial_t u_1 v_1 \, d\boldsymbol{x} - \int_\Omega u_1(1 - u_1) v_1 \, d\boldsymbol{x} + \int_\Omega \frac{u_1 u_2}{u_1 + \alpha} v_1 \, d\boldsymbol{x} - \delta_1 \int_\Omega \nabla^2 u_1 v_1 \, d\boldsymbol{x} = 0, \qquad \forall \boldsymbol{v} \in W_0 \tag{26}$$

By applying Greens theorem and the Neumann boundary conditions given in the problem description, equation (27) is achieved.

$$\int_\Omega \partial_t u_1 v_1 \, d\boldsymbol{x} - \int_\Omega u_1(1 - u_1) v_1 \, d\boldsymbol{x} + \int_\Omega \frac{u_1 u_2}{u_1 + \alpha} v_1 \, d\boldsymbol{x} + \delta_1 \int_\Omega \nabla u_1 \cdot \nabla v_1 \, d\boldsymbol{x} = 0, \qquad \forall \boldsymbol{v} \in W_0 \tag{27}$$

To find the discrete solution $u_{1,h}$, $\tau_h = \{\kappa\}$ is assumed to be the triangulated mesh defined on the domain Ω, and $\{\mathcal{N}_h\}$ is all the interior nodes on said mesh. The discrete trial space W_h is constructed as below, where $W_h \subset W_0$ and $dim(W_h < \infty)$.

$$W_{h,0} = \{\boldsymbol{v}(\boldsymbol{x}, t) : \boldsymbol{v} \in C^0(\Omega) \, \forall t \in [0, \, T], \ \boldsymbol{v}|_\kappa \in P_1(\kappa), \forall \kappa \subset \tau_h\}$$

To construct the GFEM formulation, a $\boldsymbol{u}_h$ should be found such that $\boldsymbol{u}_h \in W_h$, which results in equation (28).

$$\int_\Omega \partial_t u_{1,h} v_1 \, d\boldsymbol{x} - \int_\Omega u_{1,h}(1 - u_{1,h}) v_1 \, d\boldsymbol{x} + \int_\Omega \frac{u_{1,h} u_{2,h}}{u_{1,h} + \alpha} v_1 \, d\boldsymbol{x} + \delta_1 \int_\Omega \nabla u_{1,h} \cdot \nabla v_1 \, d\boldsymbol{x} = 0, \qquad \forall \boldsymbol{v} \in W_h \tag{28}$$

The subset W_h is given by $W_h = V_h \times V_h$, where $V_h = \text{span}(\{\varphi_j\}_{\sum_{\mathcal{N}_j \in \mathcal{N}_h}})$, and since $\boldsymbol{u}_h \in W_h$, $\exists \{\boldsymbol{\xi}(t)\}_{\mathcal{N}_j \in \mathcal{N}_h}$ such that $u_h = \sum_{\mathcal{N}_j \in \mathcal{N}_h} \boldsymbol{\xi}_j(t) \varphi_j(\boldsymbol{x})$, where $\boldsymbol{\xi}_j(t) = (\xi_{1,j}(t), \, \xi_{2,j}(t))^T$. Linear interpolation of the nonlinear terms is implemented in the same way as in the GFEM derivation in Part B, and the following formulation is achieved:

$$0 = \sum_{\mathcal{N}_j \in \mathcal{N}_h} \frac{d\xi_{1,j}(t)}{dt} \int_\Omega \varphi_j \varphi_i \, d\boldsymbol{x} - \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_{1,j}(t) \int_\Omega \varphi_j \varphi_i \, d\boldsymbol{x} + \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_{1,j}^2(t) \int_\Omega \varphi_j \varphi_i \, d\boldsymbol{x} + \\ + \sum_{\mathcal{N}_j \in \mathcal{N}_h} \frac{\xi_{1,j}(t)\xi_{2,j}(t)}{\xi_{1,j}(t) + \alpha} \int_\Omega \varphi_j \varphi_i \, d\boldsymbol{x} + \delta_1 \sum_{\mathcal{N}_j \in \mathcal{N}_h} \xi_{1,j}(t) \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_i \, d\boldsymbol{x} \quad (29)$$

By using the definition of the stiffness matrix, $A = \int_\Omega \varphi_j' \varphi_i' d\boldsymbol{x}$, and the mass matrix $M = \int_\Omega \varphi_j \varphi_i d\boldsymbol{x}$, equation (29) can be rewritten as equation (30).

$$0 = M\frac{d\boldsymbol{\xi}_1}{dt} - M\boldsymbol{\xi}_1 + M\boldsymbol{\xi}_1^2 + M\frac{\boldsymbol{\xi}_1\boldsymbol{\xi}_2}{\boldsymbol{\xi}_1 + \alpha} + \delta_1 A\boldsymbol{\xi}_1 \quad (30)$$

The same process is used to derive the FEM formulation for u_2, and then we achieve the following semi-discrete system, where $\boldsymbol{\xi}_1(0) = u_{1,0}$ and $\boldsymbol{\xi}_2(0) = u_{2,0}$:

$$\begin{cases} M\frac{d\boldsymbol{\xi}_1}{dt} - M\boldsymbol{\xi}_1 + M\boldsymbol{\xi}_1^2 + M\frac{\boldsymbol{\xi}_1\boldsymbol{\xi}_2}{\boldsymbol{\xi}_1+\alpha} + \delta_1 A\boldsymbol{\xi}_1 = 0 \\ M\frac{d\boldsymbol{\xi}_2}{dt} + \gamma M\boldsymbol{\xi}_2^2 - \beta M\frac{\boldsymbol{\xi}_1\boldsymbol{\xi}_2}{\boldsymbol{\xi}_1+\alpha} + \delta_2 A\boldsymbol{\xi}_2 = 0 \end{cases} \quad (31)$$

By applying the Crank-Nicholson scheme, the semi-discrete system can be fully discretized in the same way as in Part B, by following the scheme in equation (20). The nonlinear terms are treated explicitly. The results are shown in equation (32) and (33). The terms on the left hand sides was of both equations were used to set the bilinear form in the FEniCS implementations, and the right hand sides were used to set the linear form.

$$(M - \frac{k_n}{2}M + \delta_1\frac{k_n}{2}A)\boldsymbol{\xi}_1^{n+1} = (M + \frac{k_n}{2}M - \delta_1\frac{k_n}{2}A)\boldsymbol{\xi}_1^n - k_n M S_1^n - M S_2^n \quad (32)$$

$$(M + \gamma\frac{k_n}{2}M + \delta_2\frac{k_n}{2}A)\boldsymbol{\xi}_2^{n+1} = (M - \gamma\frac{k_n}{2}M - \delta_2\frac{k_n}{2}A)\boldsymbol{\xi}_2^n + \beta k_n M S_2^n \quad (33)$$

4.2 Problem C.1

In this problem, the initial conditions were stated in equation (34) were used. The used mesh was the provided file: *circle_fine.xml.gz*. As for the time, a step of $k = 0.5$ was used on the domain $T \in [0,\ 1000]$. The Python-script can be found in Appendix C.

$$\begin{cases} u_{1,0} = \frac{4}{15} - 2 \cdot 10^{-7}(x_1 - 0.1x_2 - 225)(x_1 - 0.1x_2 - 675) \\ u_{2,0} = \frac{22}{45} - 3 \cdot 10^{-5}(x_1 - 450) - 1.2 \cdot 10^{-4}(x_2 - 150) \end{cases} \quad (34)$$

The rendered results for $\boldsymbol{u}$ is shown i Figure 10-19, where u_1 represents the density of the prey, and u_2 represents the density of the predators on the domain. Red color represents a higher density and blue color represents a lower density of prey/predator. The results for u_1 is shown in the left column, and u_2 is shown in the right column. The figures shows the rendered results for time $T = 0,\ 50,\ 150,\ 1000$.

As can be seen, the for $T = 50$ to 150, the results attains a spiral form for both the prey and the predators, and it is noticeable that the prey has a higher density where the predators has a lower density, and vice versa. This is agrees with the analogy that if there are a lot of foxes in one area, the amount of rabbits while lower in said area since they are being eaten by the foxes. If there are less foxes present, the amount

of rabbits will be able to increase. In other terms: u_1 and u_2 are co-dependent of each other. As time goes on, the solutions of u_1 and u_2 disperses to a more random pattern, which is illustrated at $T = 1000$. At this time, there is harder to draw any conclusions form the rendered results.

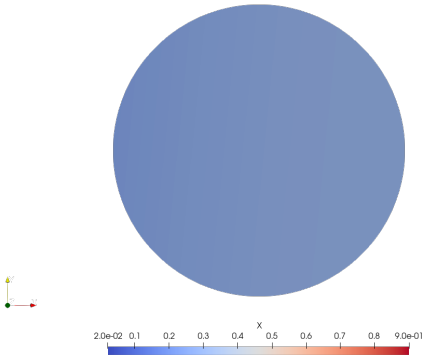

Figure 10: u_1 *for* $T = 0$

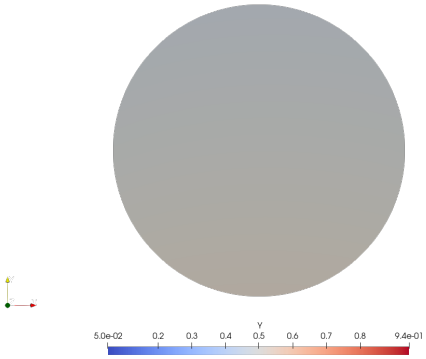

Figure 11: u_2 *for* $T = 0$

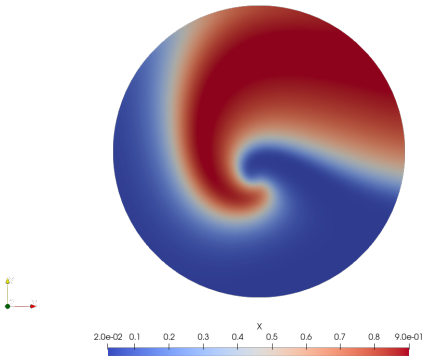

Figure 12: u_1 *for* $T = 50$

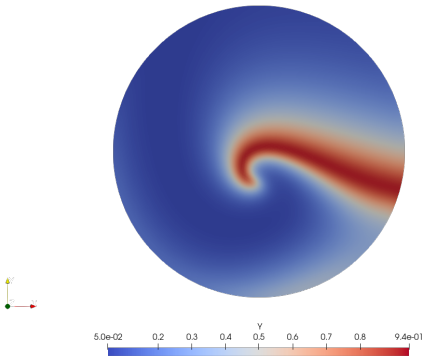

Figure 13: u_2 *for* $T = 50$

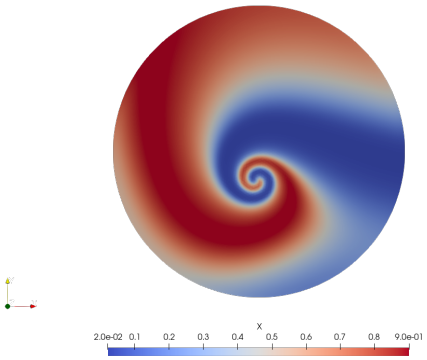

Figure 14: u_1 *for* $T = 100$

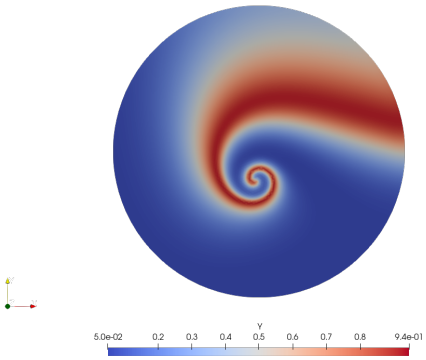

Figure 15: u_2 *for* $T = 100$

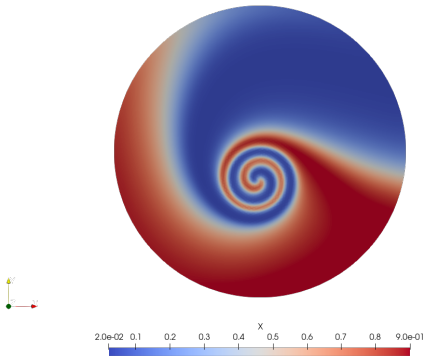

Figure 16: u_1 *for* $T = 150$

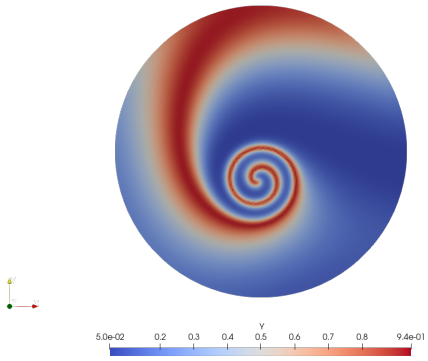

Figure 17: u_2 *for* $T = 150$

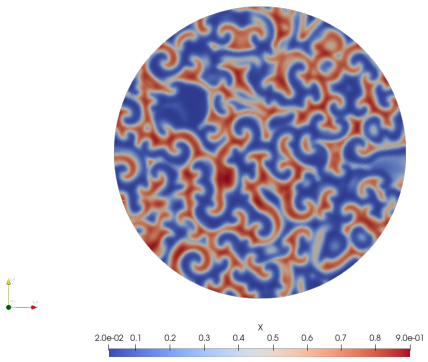

Figure 18: u_1 *for* $T = 1000$

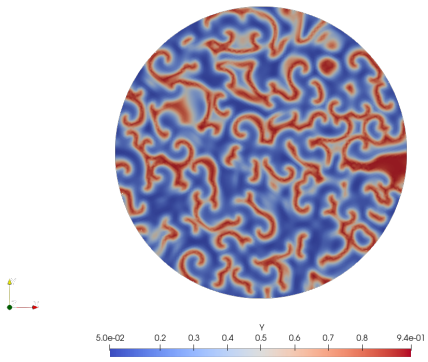

Figure 19: u_2 *for* $T = 1000$

In Figure 20, the total populations of both prey and predators were calculated according to equation (2), and then plotted as a function of time T. Here, it can be easily seen, from $T = 0$ to around $T = 50$, that the amount of predators decreases due to a low amount of prey. The amount of prey rises quickly while the amount of predators decreases at the same rate. After $T = 50$, the populations works towards a form of equilibrium, were there are oscillations in the populations, but they somewhat oscillates around a respective mean value. However, the amplitude of the oscillations are quite high.

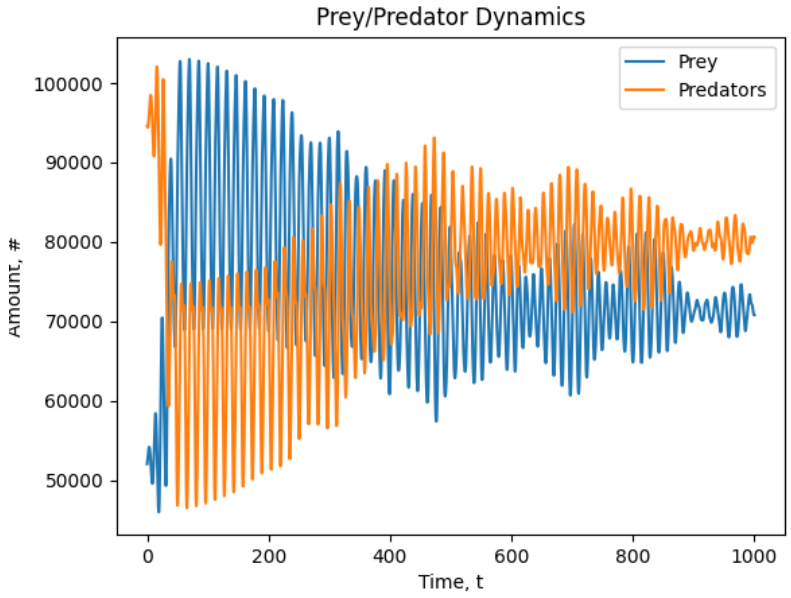

Figure 20: *Illustration of the total populations of prey and predators and the dynamics between them.*

4.3 Problem C.2

In this problem, the initial conditions were stated in equation (35) were used, where $\text{random}(\boldsymbol{x}) : \mathbb{R}^2 \mapsto [0, 1]$ is randomly generated numbers. The Python function `numpy.random.uniform` was used to generate the random numbers, were the numbers are randomly drawn from a uniform distribution. The rest of the set-up was the same as in previous problem: mesh given by the provided file: *circle_fine.xml.gz*, and the time was given by $T \in [0,\ 1000]$, were the time step was given by $k = 0.5$. The Python-script can be found in Appendix C.

$$\begin{cases} u_{1,0} = \frac{1}{2}(1 - \text{random}(\boldsymbol{x})) \\ u_{2,0} = \frac{1}{4} + \frac{1}{2}\text{random}(\boldsymbol{x}) \end{cases} \tag{35}$$

Figure 21-30 shows the rendered results for the the prey density u_1 and the predator density u_2. Since the initial conditions were randomly generated for each coordinate $\boldsymbol{x}$, the pattern in the solution not as ordered as in Problem C.1, and therefor the results are harder to interpret by just looking at the plotted densities.

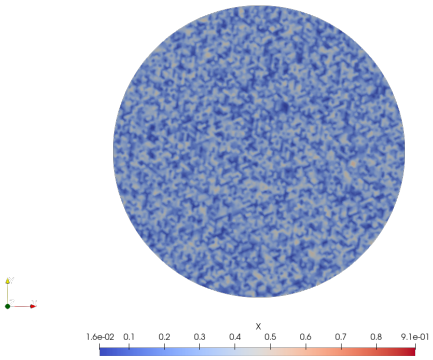

Figure 21: u_1 *for* $T = 0$

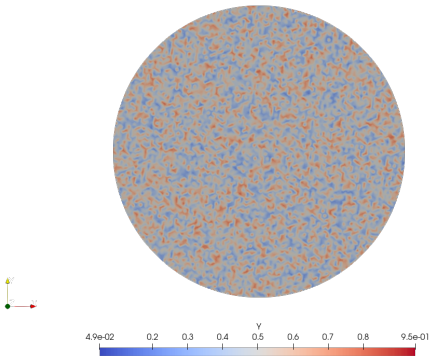

Figure 22: u_2 *for* $T = 0$

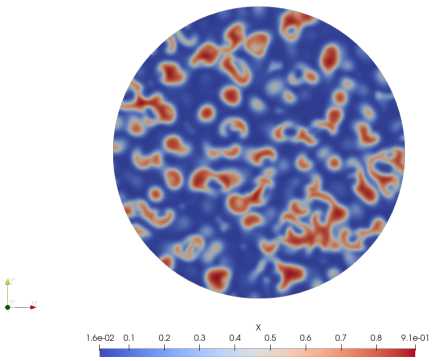

Figure 23: u_1 *for* $T = 50$

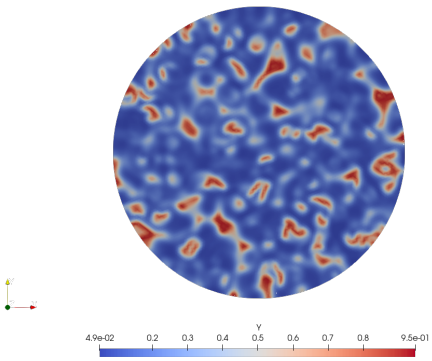

Figure 24: u_2 *for* $T = 50$

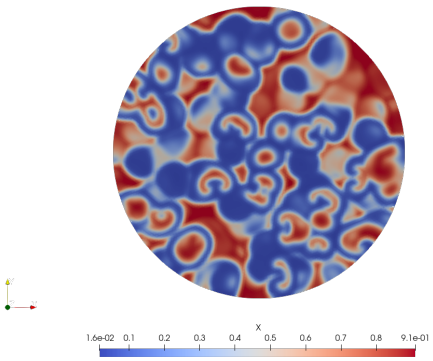

Figure 25: u_1 *for* $T = 100$

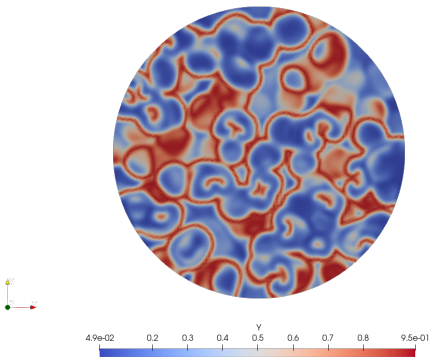

Figure 26: u_2 *for* $T = 100$

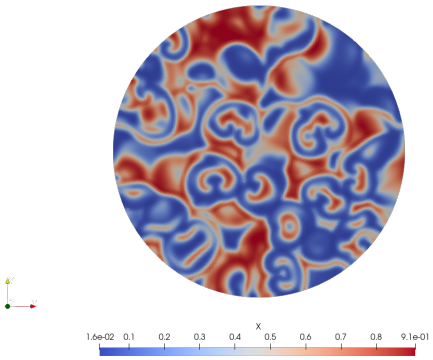

Figure 27: u_1 *for* $T = 150$

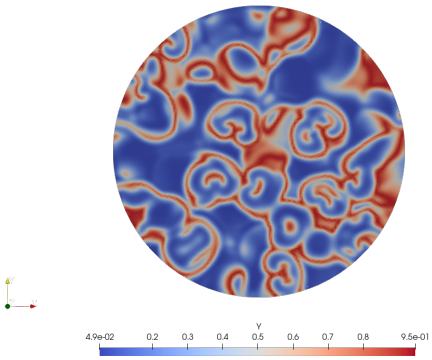

Figure 28: u_2 *for* $T = 150$

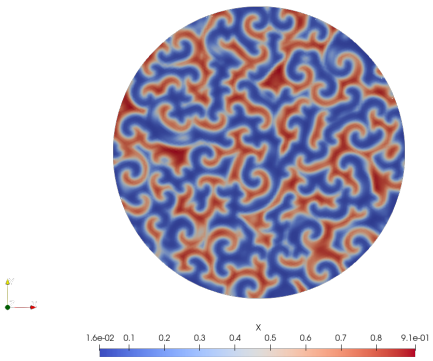

Figure 29: u_1 *for* $T = 1000$

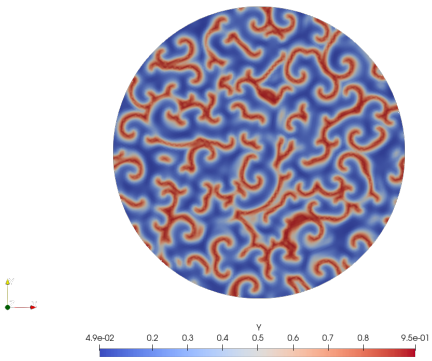

Figure 30: u_2 *for* $T = 1000$

The total populations in the domain were again calculated by equation (2), and plotted against time T in Figure 31. It can be seen that the first $T = 200$, the solution reminds of a dampened harmonic oscillator, as opposed to the more drastic changes initially in Problem C.1, before the solutions oscillates around an "equilibrium". This is probably due to the initial values being uniformly generated, hence creating a more evenly distributed densities of prey/predators, which helps the process to reach a steady state faster. Here, it can also be seen that the amount of predators is slightly above the amount of prey, and the co-dependence is clearly shown at around e.g. $T = 400$. When the amplitude of the oscillations of predators increases, the oscillations of the prey behaves similarly. The same happens when the amplitude decreases.

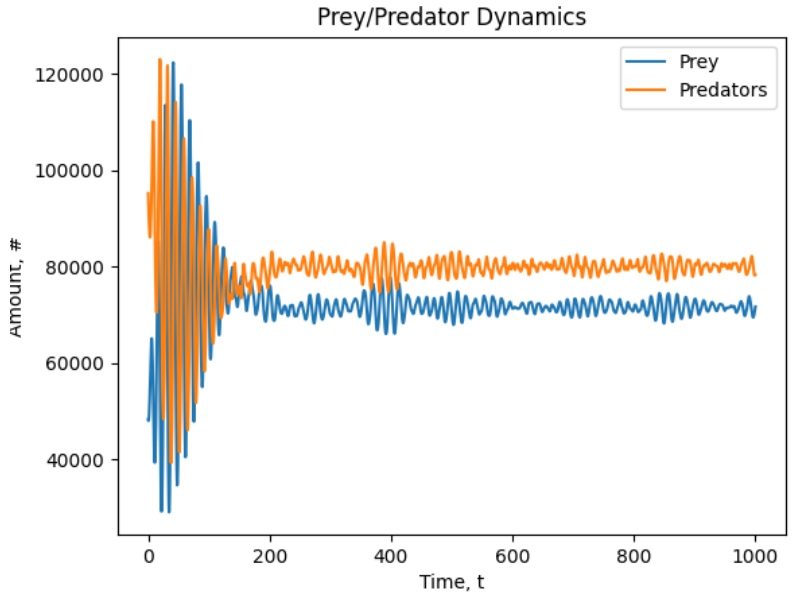

Figure 31: *Illustration of the total populations of prey and predators and the dynamics between them. Here, the initial conditions were randomly generated.*

5 Conclusion

In **Part A**, the focus was to derive the a posteriori error estimate that then was used to implement an adaptive mesh refinement for the 1D Poisson equation. The simulation started with $N = 12$ nodes, and ended with $N = 79$ nodes when desired error tolerance (calculated by the a posteriori error) was reached. A connection between the residuals, the error and the mesh distribution could be seen.

A 2D convergence analysis of the GFEM was performed in **Part B**, here implemented on another version of the Poisson equation. The results showed that the energy norm of the error had a superlinear relationship, $P = 1.4$, to the maximal mesh-size h_{max}. The time-dependent GFEM formulation, discretized with Crank-Nicholson scheme, were then derived and implemented in MATLAB for a simplified version of the Prey-Predator model, where the amount of predators were considered constant, but the amount of prey were dynamic. It could be noted that the amount of prey kept decreasing over time without any tendency to recover.

In **Part C**, the full system of PDE of the Predator-Prey model was implemented, here using the FEniCS-project in Python instead of MATLAB. It was clearly noted that the dynamics of the predators and the prey were highly dependent on each other, which could be seen in both the density of the of both species over the domain, as well as the total populations over time. As opposed to Part B, the total populations both increased and decreased heavily before oscillating around some form of equilibrium. There was a notable difference in using predetermined initial conditions and randomly generated initial conditions.

Appendix A

The MATLAB-script for Problem A.2. The main script comes first, and is then followed by the functions used in the main script.

```
1
2
3  %%% Main script for part A.
4  %%% Finds the solution for u_h by using FEM, caluclates
5  %%% a posteriori error estimation, and then runs the loop
6  %%& until desired error tolerance is reached.
7
8  close all; clear all;
9  a = -1;
10 b = 1;
11 Δ = 0.01;
12 N = 12;
13 x = linspace(a,b,N);
14 stopTOL = 10^-3;
15 stopNodes = 10^4;
16 condition = 1;
17
18 while (condition > stopTOL && N < stopNodes)
19     %FEM
20     f = RHSfunc(x)./(Δ);
21     bLoad = load_vect(x,f);
22     S = S_matrix(x,length(x));
23     M = M_matrix(x,length(x));
24     uh = (S\bLoad);
25     uhLap = -(M\(S*uh));
26
27     %Calculates residual and eta^2
28     Rh = Δ.*f+Δ.*uhLap';
29     eta2 = zeros(N-1,1);
30     for i=1:N-1
31         h = x(i+1)-x(i);
32         temp = h/2*(Rh(i)^2+Rh(i+1)^2);
33         eta2(i) = h^2*temp;
34     end
35
36     %Refines the mesh
37     lambda = 0.9;
38     xold = x;
39     for i = 1:length(eta2)
40         if eta2(i) > lambda*max(eta2)
41             x = [x (x(i+1)+x(i))/2];
42         end
43     end
44     x = sort(x);
45     N = length(x);
46     %condition = sum(eta2);
47     condition = sum(sqrt(eta2));
48 end
49
50 % Plots the results
51 subplot(2,2,1)
52 plot(xold,uh);
53 title("Solution u_h");
54 xlabel("x");
55
56 subplot(2,2,2)
57 plot(xold,Rh);
58 title("Residual");
```

```
xlabel("x");
ylabel("R(u_h)");

subplot(2,2,3)
plot(xold(2:end),sqrt(eta2));
title("Error");
xlabel("x");
ylabel("\eta(u_h)");

subplot(2,2,4)
plot(xold(2:end),[1./diff(xold)])
title("Mesh Distribution");
xlabel("x");
ylabel("Amount");
```

```
%%% Sets the RHS of the second degree ODE.
function f = RHSfunc(x)
f = zeros(1, length(x));
    for i=1:length(x)
        if abs(x(i)) ≤ 0.1
            f(i) = -5;
        elseif abs(0.2-abs(x(i))) ≤0.1
            f(i) = 25;
        elseif abs(0.6-abs(x(i))) ≤0.1
            f(i) = -30;
        elseif abs(0.9-abs(x(i))) ≤0.2
            f(i) = 20;
        else
            f(i) = 1;
        end
    end
end
```

```
%%% Creates the load vector used in FEM
%%% Code obtained from instructions to Lab1
%%% in the course
%%% "Applied Finite Element methods" @ UU, fall 2020
function B=load_vect(x,RHS)
N = length(x);
B = zeros(N, 1);
for i = 1:N-1
    h = x(i+1) - x(i);
    n = [i i+1];
    B(n) = B(n) + [RHS(i); RHS(i+1)]*h/2;
end
```

```
%%% Creates the Stiffness matrix S used in FEM
function S = S_matrix(x,N)
S = zeros(N,N);
for i=1:N-1
    h = x(i+1)-x(i);
    node = [i i+1];
    S(node, node) = S(node, node) + [1 -1; -1 1]/h;
end

% Sets BC to 0
S(1,1) = 10^10;
S(N,N) = 10^10;
end
```

```
%%% Creates the Mass matrix M used in FEM
function M = M_matrix(x,N)
M = zeros(N,N);
for i=1:N-1
    h = x(i+1)-x(i);
    node = [i i+1];
    M(node,node) = M(node, node) + [2 1; 1 2]*h/6;
end
end
```

Appendix B

The MATLAB-scripts for Part C. The first two scripts are the main scripts used for Problem B.1 and Problem B.2 respectively. The remaining scripts are functions used in the main functions. See comments for more information.

```
%%% Main program for problem B1
clear all; close all;

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

for i=1:length(hmax)
    [p,e,t] = initmesh(geometry, 'hmax', hmax(i)); %point, edge, element data

    u_exact = @(x1,x2) sin(2*pi*x1).*sin(2*pi*x2);
    I = eye(length(p));
    A = stiffness2D(p,t);
    b = loadvect2D(p,t,@RHS);

    % BC
    boundaries = zeros(length(e(1,:)),1);
    for j=1:length(e(1,:))
        x1 = p(1,e(1,j));
        x2 = p(2,e(1,j));
        boundaries(j,1) = u_exact(x1,x2);
    end
    A(e(1 ,:) ,:) = I(e(1 ,:) ,:);
    b(e(1,:),:) = boundaries(:,1);

    uh = A\b;

    figure(i)
    pdesurf(p,t,uh);
    title("hmax = " + hmax(i));

    %calc energy norm of error
    error = u_exact(p(1,:),p(2,:))'-uh;
    EnE(i) = sqrt(error'*A*error);
end

P = polyfit(log10(hmax), log10(EnE),1);
figure(i+1)
loglog(hmax,EnE,'-b', hmax, hmax.^(P(1)), '-r');
title("Error");
xlabel("log(h_{max})")
ylabel("log(||u-u_h||_E)");
legend("Error", "h_{max}^P, P =" + P(1),'Location', 'southeast');

%Convergence between each hmax
q = zeros(1,length(hmax)-1);
for i=1:length(q)
    q(i) = (log10(EnE(i+1))-log10(EnE(i)))/(log10(hmax(i+1))-log10(hmax(i)));
end
```

```
%%% The main program for problem B2.
clear all; close all;

% Geometry setup
geometry = @circleg;
```

```
hmax = [1/5 1/20 1/40];

% Constants etc
% Δ = 0.01;
% alpha = 4;
T = 2;
k = 0.01*hmax;
uh = cell(1,length(hmax));
Mprey = cell(1,length(hmax));
p = cell(1,length(hmax));
t = cell(1,length(hmax));

for i=1:length(hmax)
    [p{1,i},e,t{1,i}] = initmesh(geometry, 'hmax', hmax(i));
    A = stiffness2D(p{1,i},t{1,i});
    M = mass2D(p{1,i},t{1,i});

    %Crank-Nicolson
    [uh{1,i}, Mprey{1,i}] = crankNic(M,A,T,k(i),p{1,i},t{1,i});

    %Plots uh
    figure(i)
    subplot(1,2,1)
    pdesurf(p{1,i}, t{1,i}, uh{1,i}(:,1));
    title("T = 0")
    subplot(1,2,2)
    pdesurf(p{1,i}, t{1,i}, uh{1,i}(:,end));
    title("T = " + T);
    suptitle("h_{max} = " + hmax(i))
end

%Plots Mprey
figure(i+1)
for j=1:length(hmax)
    time = 0:k(j):T;
    subplot(3,1,j);
    plot(time, Mprey{1,j})
    title("h_{max} = " + hmax(j));
    legend("Mprey");
    suptitle("Numb. of prey over time")
end
```

```
%%% Constructs the stiffness matrix
function A = stiffness2D(p,t)

N = size(p,2);
A = sparse(N,N);

for K =1:size(t,2)
    nodes = t(1:3,K);
    x = p(1,nodes);
    y = p(2,nodes);
    area_K = polyarea(x,y);
    bi = [y(2)-y(3); y(3)-y(1); y(1)-y(2)]/(2*area_K);
    ci = [x(3)-x(2); x(1)-x(3); x(2)-x(1)]/(2*area_K);
    AK = (bi*bi'+ci*ci')*area_K;
    A(nodes,nodes) = A(nodes,nodes)+AK;
end
end
```

```
%%% Constructs the load vector
function b = loadvect2D(p,t,f)

N = size(p,2);
b = sparse(N,1);

for K = 1:size(t,2)
    nodes = t(1:3,K);
    x = p(1,nodes);
    y = p(2,nodes);
    area_K = polyarea(x,y);
    bK = [f(x(1),y(1)); f(x(2), y(2)); f(x(3),y(3))]/3*area_K;
    b(nodes) = b(nodes)+bK;
end
end
```

```
%%% Constructs the mass matrix
function M = mass2D(p,t)
N = size(p,2);
M = sparse(N,N);

for K =1:size(t,2)
    nodes = t(1:3,K);
    x = p(1,nodes);
    y = p(2,nodes);
    area_K = polyarea(x,y);

    Mk = (1/12)*[2 1 1; 1 2 1; 1 1 2]*area_K;
    M(nodes, nodes) = M(nodes,nodes) + Mk;
end
```

```
%%% Sets the value of f in B1.
function f = RHS(x1,x2)
    f = 8*pi.^2.*sin(2*pi.*x1).*sin(2*pi.*x2);
end
```

```
%%% The function for the Crank Nicolson scheme, etc.
%%% Used in the main program for problem B2
function [uh, Mprey] = crankNic(M,A,T,k,p,t)
%Constants
alpha = 4;
Δ = 0.01;
s = @ (xi) xi./(xi+alpha)-xi.*(1-xi);
q = Δ*k/2;

% Sets inital data for uh
u_initial = zeros(length(p),1);
for j=1:length(p)
    omega = rand(); % open interval though
    u_initial(j,1) = 1+20*omega;
end
xi = u_initial;
uh = xi;

%Sets the initial number of prey
for K = 1:size(t,2)
    nodes = t(1:3,K);
    x = p(1,nodes);
    y = p(2,nodes);
```

```
    area_K = polyarea(x,y);
    prey(K) = area_K/3*(xi(nodes(1))+xi(nodes(2))+xi(nodes(3)));
end
Mprey(1) = sum(prey);

%The acutal Crank Nic
time=0;
i = 1;
while time≤T
    num = M*xi-k*M*s(xi)-q*A*xi;
    denum = M+q*A;
    xi = denum\num;
    uh(:,i+1)= xi;

    %Calculates prey
    for K = 1:size(t,2)
        nodes = t(1:3,K);
        x = p(1,nodes);
        y = p(2,nodes);
        area_K = polyarea(x,y);
        prey(K) = area_K/3*(xi(nodes(1))+xi(nodes(2))+xi(nodes(3)));
    end
    Mprey(i+1) = sum(prey);
    i=i+1;
    time=time+k;
end
end
```

Appendix C

The code used in both problems in Part C. Here, the code is written in Python. For Problem C.1, the line 34-35 is uncommented, and line 38-39 is commented with #. Names of folders where results are being saved can also be changed for Problem C.1 (e.g. line 69)

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jan  5 23:44:48 2021

@author: matil
"""

import matplotlib.pyplot as plt
import numpy as np
from dolfin import *

# Create mesh and define function space
mesh = Mesh("circle_fine.xml.gz")

# Construct the finite element space
V = VectorFunctionSpace (mesh, 'P', 1)

# Define parameters
T = 1000
k = 0.5
alpha = 0.4
beta = 2
gamma = 0.8
Δ1 = 1
Δ2 = 1

# Class representing the intial conditions
# Randomly generated values for problem C2
class InitialConditions(UserExpression):
    def eval(self, values,  x):

        # Values for Problem C1
        # values[0] = 4/15-2*(10**(-7))*(x[0]-0.1*x[1]-225)*(x[0]-0.1*x[1]-675)
        # values[1] = 22/45-3*(10**(-5))*(x[0]-450)-1.2*(10**(-4))*(x[1]-150)

        # Values for Problem C2
        values[0] = 1/2*(1-np.random.uniform(0,1))
        values[1] = 1/4+1/2*np.random.uniform(0,1)

    def value_shape(self):
        return (2,)

# Define initial condition
indata = InitialConditions(degree=2)
u_ini = Function(V)
u_ini = 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*k*u[0]*v[0]*dx + 1/2*k*Δ1*inner(grad(u[0]), grad(v[0]))*dx
a1 = u[1]*v[1]*dx + gamma*k*1/2*u[1]*v[1]*dx + 1/2*k*Δ2*inner(grad(u[1]), grad(v[1]))*dx

L0 = u_ini[0]*v[0]*dx + 1/2*k*u_ini[0]*v[0]*dx -\
```

```
    (1/2*k*Δ1*inner(grad(u_ini[0]), grad(v[0]))*dx) -\
    (k*((u_ini[0]*u_ini[1])/(u_ini[0]+alpha) + u_ini[0]*u_ini[0])*v[0]*dx)

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

a = a0 + a1
L = L0 + L1

# Set an output file
file = File("C2_fine/sol_c2.pvd","compressed")

# Calculating solution
u = Function(V)
u.assign(u_ini)
pop_prey = []
pop_pred = []
time = []
M_prey = u[0]*dx
M_pred = u[1]*dx
prey = assemble(M_prey)
pred = assemble(M_pred)

# Time step
t = 0.0
file << (u,t)
while t ≤ T:
    time.append(t)
    u_ini.assign(u)
    A = assemble(a)
    b = assemble(L)

    solve (A, u.vector(), b, "lu")

    prey = assemble(M_prey)
    pred = assemble(M_pred)
    pop_prey.append(prey)
    pop_pred.append(pred)

    #Saving solution to file in steps of t = 50
    if t % 50 == 0:
        print("Saved, t = " + str(t))
        file << (u,t)

    t += k


# Plotting stuff
plt.plot(time,pop_prey, label = 'Prey')
plt.plot(time, pop_pred, label = 'Predators')
plt.title("Prey/Predator Dynamics")
plt.xlabel("Time, t")
plt.ylabel("Amount, #")
plt.legend(loc = 'upper right')

# Saving stuff
plt.savefig("C2_fine/plot.png")
np.savetxt("C2_fine/time.txt", time)
np.savetxt("C2_fine/pop_prey.txt", pop_prey)
np.savetxt("C2_fine/pop_pred.txt", pop_pred)
```