UPPSALA UNIVERSITET

Applied Finite Element Method
Project

Christos Pilichos: christos.Pilichos.9957@student.uu.se
Panagiotis Papias: panagiotis.Papias.2534@student.uu.se

Date of Submission: December 20 , 2022

Contents

1 Introduction

Our aim in this project is to get familiar with one-and two-dimensional prey-predator-mutualist dynamics. The term mutualism refers to the interaction between two or more species, in which each of them receives a benefit. In mutualism, the one species does not negatively affect the existence of the rest ones. The mathematical model describing the behaviour of mutualism is called Lotka-Volterra. The Lotka-Voltera is a system of Ode's proposed in 1910 by Alfred J. Lotka (1880-1949). Later, the same equations where analyzed by Vito Voltera (1860-1940). So, we are interested in the following dynamics of prey-predator-mutualism, which is modeled by pdes. To this end, we want to solve the following system of so called *reaction-diffusion* pdes:

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

$$\partial_t v - \beta v(1 - v) + \frac{vw}{\alpha + v + mu} - \delta_2 \Delta v = g \ , \ (\mathbf{x}, t) \in \Omega \times (0, T] \ , \tag{2}$$

$$\partial_t w - \gamma vw - \zeta \frac{vw}{\alpha + v + mu} - \delta_3 \Delta w = p \ , \ (\mathbf{x}, t) \in \Omega \times (0, T] \ , \tag{3}$$

where $\Omega \in \mathbb{R}^d$, $d = 1, 2, 3$ is a bounded domain , $T > 0$ is a finite time, $f(\mathbf{x}, t)$, $g(\mathbf{x}, t)$, $p(\mathbf{x}, t)$ are given initial terms, and u_o , v_o , w_o are initial data. Boundary conditions are to be determined. Furthermore, $u(\mathbf{x}, t)$, $v(\mathbf{x}, t)$ & $w(\mathbf{x}, t)$ represents the population density of mutualists, preys & predators respectively. The terms of (1) denotes the change with respect to time, mutualist birth and death rates and population diffusion. The terms of (2) denotes the change with respect to time, prey birth and death rates, prey consumption rate per predator as a fraction of maximum consumption rate 1 and population diffusion. In equation (3) the terms refer to change with respect to time, mortality rate of the predators, prey consumption rate per predator and population diffusion. The parameters: α, β, γ, δ_1, δ_2, δ_3 and L_o , l , m are positive. The parameters l and m denote the constraints of mutualism. It can be seen that for $l = m = 0$ we get the usual prey-predator model.

The system of equations: (1)-(3) is a dynamic model that describes the way mutualists , preys and predators are related to each other. A decrease on the number of predators is followed by a respective increase of the number of preys according to the model, and it results in an increase of the population of mutualists. However, when the number of preys increases, it helps on the reproduction of the predators and it results to a respective growth of their population. Consequently, the consumption of preys is increased and their population is decreased respectively, therefore the mutualist population follows a declining tense as well. At the same time, due to lack of preys, the predator begins to die due to starvation. The population variation for preys, predators & mutualists is given by calculating respectively the overall population rates in the Ω domain:

$$M_{mutualist}(t) = \int_\Omega u(\mathbf{x}, t) d\mathbf{x} \ , \ M_{prey}(t) = \int_\Omega v(\mathbf{x}, t) d\mathbf{x} \ , \ M_{predator}(t) = \int_\Omega w(\mathbf{x}, t) d\mathbf{x} \tag{4}$$

The problem is separated in three parts where:

- in Part A we deal with **one-dimensional stationary diffusion** problem,
- in Part B with a **two-dimensional scalar** and **system of time-dependent reaction-diffusion** equations &
- in Part C with the solution of the system using the **FeniCS** project.

2 Part A

We start by considering a 1D simplified model,

$$-\delta u''(x) = f(x), \ x \in (-1, 1) \tag{5}$$

$$u(-1) = u(1) = 0. \tag{6}$$

where *f(x)* is some forcing function. The first part of the project is on implementing **FEM-solver** for this 1D problem in Matlab, **investigate numerical errors** and perform adaptive mesh refinement based on an **a posteriori** error estimation.

2.1 Problem A.1

Assume u_h be a finite element approximation of the solution of the problem (5)-(6) on a mesh

$$-1 = x_0 < x_1 < ... < x_N = 1.$$

Let $h_i = x_i - x_{i-1}$ be a mesh-size, $I_i = (x_{i-1}, x_i)$ be the *i-th* element and $I = \cup_{i=1}^{n} I_i$ be the mesh. We want to acquire a posteriori error estimate in the energy norm:

$$\|(u - u_h)'\|_{L^2(I)}^2 \leq C \sum_{i=1}^{n} \eta_i^2, \tag{7}$$

where C stands for a constant and $\eta_i = h_i \|f + \delta u_h''\|$ is the element residual.

Let us set $e = u - u_h$ as the error. By taking the L_2 norm for it we get:

$$\|e'\|_{L^2(I)}^2 = \int_{-1}^{1} e'^2 dx = \int_{-1}^{1} e'(e - \pi_1 e)' dx = \sum_{i=1}^{n} \int_{x_i-1}^{x_i} e'(e - \pi_1 e)' dx$$

where we implemented the Galerkin Orthogonality, by subtracting the interpolant $\pi_1 e \in V_h, 0$ from e , also it is worth mentioning that the derivative of $\pi_1 e$ at the nodal points does not exist for every I_i element (although it is differentiable in between). So we get:

$$\|e'\|_{L^2(I)}^2 = \sum_{i=1}^{n} \int_{x_i-1}^{x_i} e'(e - \pi_1 e)' dx = \sum_{i=1}^{n} e'(e - \pi_1 e)\Big|_{x_i-1}^{x_i} - \sum_{i=1}^{n} \int_{x_i-1}^{x_i} e''(e - \pi_1 e) dx$$

where we integrated by parts $\forall\ I_i$ moreover the boundary terms $\forall\ [x_{i-1}, x_i]$ are zero, since the e and $\pi_1 e$ coincide in the respective nodes. Now we have:

$$\|e'\|_{L^2(I)}^2 = \sum_{i=1}^{n} \int_{x_i-1}^{x_i} (-e'')(e - \pi e) dx \tag{8}$$

Now by examining more thoroughly $-e''$ for every I_i we get:

$$-e'' = -(u - u_h)'' = -u'' + u_h'' = \frac{f}{\delta} + u_h''$$

we substituted $-u''$ by (5) where $\delta\ \neq 0$, so after this we get:

$$\|e'\|_{L^2(I)}^2 = \sum_{i=1}^{n} \int_{x_i-1}^{x_i} (\frac{f}{\delta} + u_h'')(e - \pi_1 e) dx = \sum_{i=1}^{n} \int_{x_i-1}^{x_i} \frac{1}{\delta}(f + \delta u_h'')(e - \pi_1 e) dx$$

next, we implement the Cauchy-Schwarz inequality:

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

and we use a standard interpolation error estimate where the constant $C' = \frac{C}{\delta}$ substitutes both constants:

$$\begin{aligned}
\|e'\|_{L^2(I)}^2 &\leq \sum_{i=1}^{n} \|f + \delta u_h''\|_{L^2(I_i)} C' h_i \|e'\|_{L^2(I_i)} = C' \sum_{i=1}^{n} h_i \|f + \delta u_h''\|_{L^2(I_i)} \|e'\|_{L^2(I_i)} \\
&\leq C' \left(\sum_{i=1}^{n} h_i^2 \|f + \delta u_h''\|_{L^2(I_i)}^2 \right)^{1/2} \left(\sum_{i=1}^{n} \|e'\|_{L^2(I_i)}^2 \right)^{1/2} \\
&= C' \left(\sum_{i=1}^{n} h_i^2 \|f + \delta u_h''\|_{L^2(I_i)}^2 \right)^{1/2} \|e'\|_{L^2(I)}
\end{aligned}$$

now, dividing both sides by $\|e'\|_{L^2(I)}$ we end up on the final formula:

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

or

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

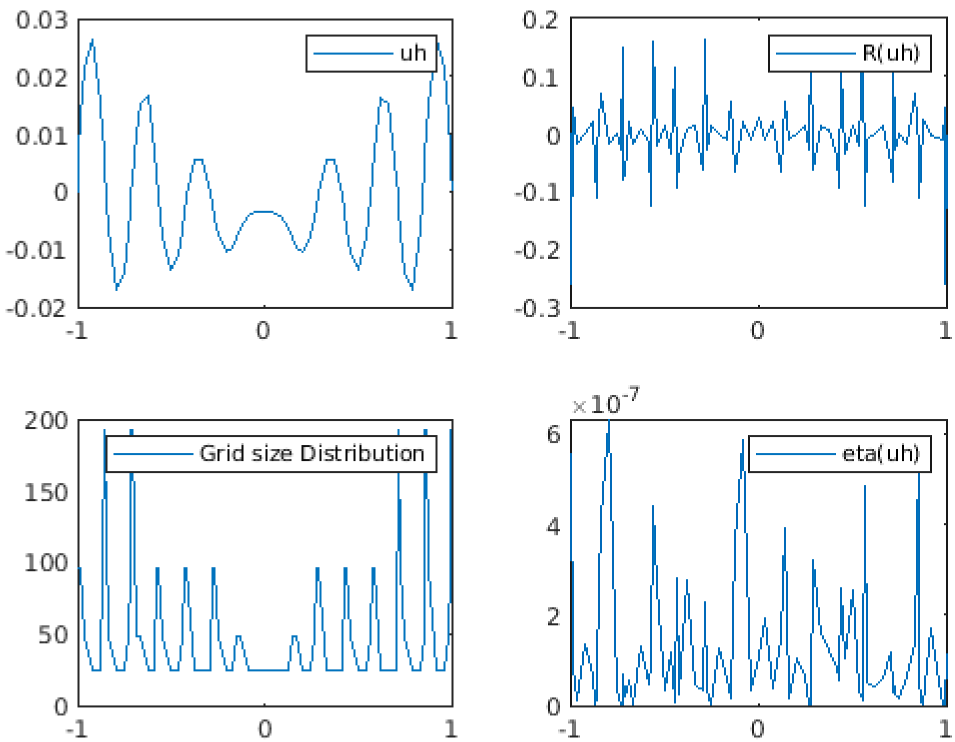
where $\eta_i = h_i \|f + \delta u_h''\|_{L^2(I_i)}$.

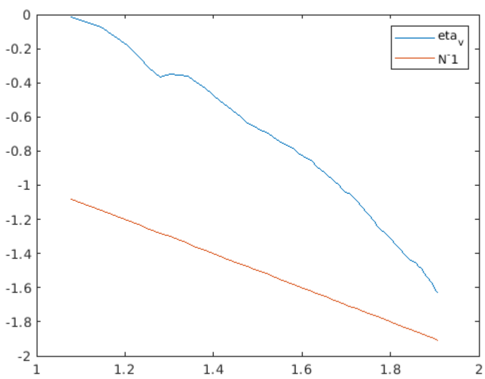
2.2 Problem A.2

Using now the a posterriori error estimate we perform an adaptive element refinement, in order to increase the accuracy of our FEM approximation and decrease the computational cost.

In the end we plot the solution u_h, the Residual $R(u_h) = f + \delta\Delta_h u_h$, the error indicator $\eta(u_h)$ and the grid size distribution.

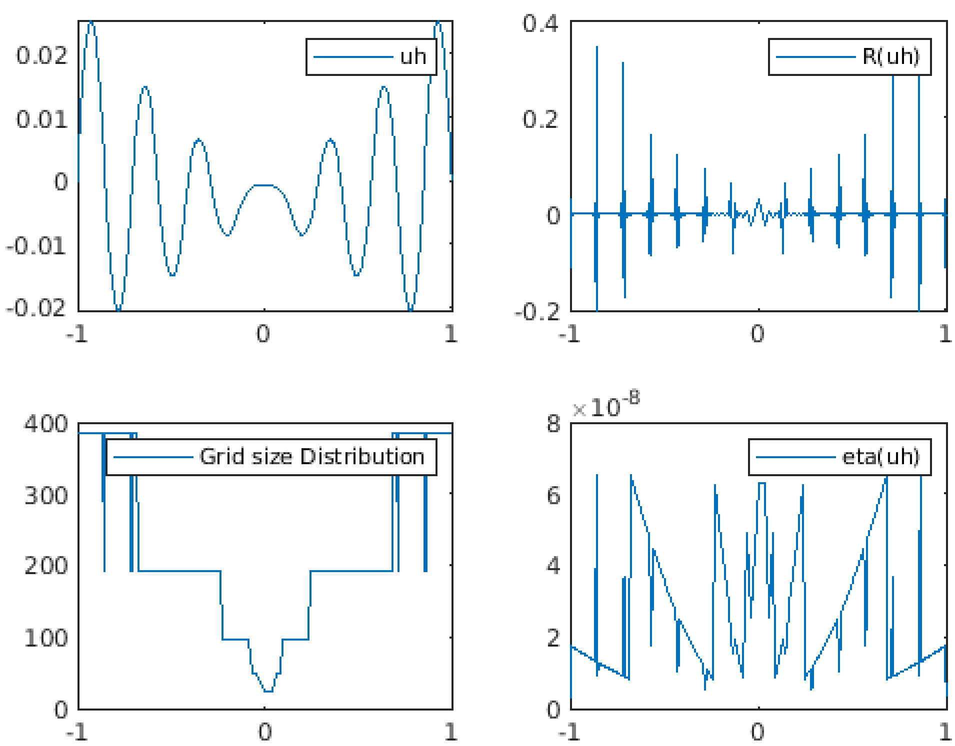
Moreover, a graph of the total number of nodes a each iteration (N) vs the sum of error indicators (eta_v) is presented, as well as the function $N = f(N^{-1})$ in the same graph.

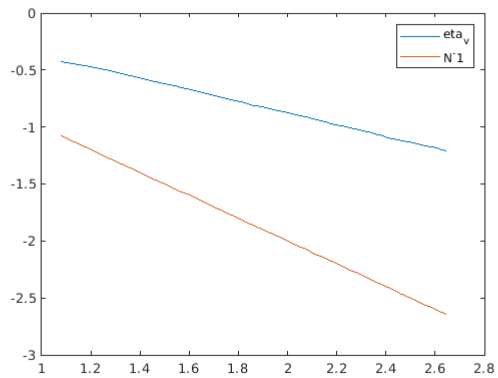




2.3 Problem A.3

Our next goal is to implement now an adaptive finite element solver omitting this time the discrete Laplacian from the a posterriori error estimate. Subsequently, the following results are presented:





Comparing the graphs of our two solvers we can see that the approximated solution does not differ. The only significant difference can be noticed in the grid size distribution and the error estimate.

We can feel the difference of the two methods in the convergence rate graph, where the convergence rate order of the method including discrete Laplacian is greater.

It's worth mentioning that we examined the convergence rate of both methods, by computing the slope of $log(N) = log(f(eta_v))$,since the graph was in logarithmic scale.

3 Part B

3.1 Problem B.1

We consider the simplified mutualist population equation, which is obtained by omitting the time-derivative and non-linear terms:

$$\begin{aligned} -\Delta u(\boldsymbol{x}) &= f(\boldsymbol{x}),\ \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= u_{exact}(\boldsymbol{x}),\ \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{9}$$

where

$$f(\boldsymbol{x}) = 8\pi^2 sin(2\pi \boldsymbol{x}_1) sin(2\pi \boldsymbol{x}_2) \ \&\ u_{exact}(x) = sin(2\pi \boldsymbol{x}_1) sin(2\pi \boldsymbol{x}_2).$$

Using the Galerkin finite element method, with continuous piece-wise linear functions, we try to derive a linear system. First of all, we produce the weak form (**WF**) of (9) by multiplying with a test function v, which is assumed to vanish on the boundary. Then we integrate using **Green's formula**:

$$\int_\Omega (-\Delta u v) dx = \int_\Omega v f dx \Rightarrow$$

$$\int_\Omega (\nabla u)(\nabla v) dx - \int_{\partial\Omega} n \cdot \nabla u\ v ds = \int_\Omega v f dx \Rightarrow$$

$$\int_\Omega (\nabla u)(\nabla v) dx = \int_\Omega v f dx \tag{10}$$

For the last step we claimed that $v = 0$ on $\partial\Omega$, since we know the behaviour of u on the boundary.

So, now it is time to introduce the appropriate spaces:

$$V := \{v : \|v\| + \|\nabla v\| < \infty\} \ \&\ V_0 := \{v : v \in V\ ;\ v|_{\partial\Omega} = 0\} \tag{11}$$

Hence, we find $u \in V_0$, such that:

$$\int_\Omega (\nabla u) \cdot (\nabla v) dx = \int_\Omega f v dx\ ,\ \forall\ v \in V_0 \tag{12}$$

Let $V_h \subset V$ be the space of all continuous piece-wise linear functions on a partition $\boldsymbol{K} = \{K\}$ of Ω into triangles with diameter (longest edge) h_K. To satisfy the boundary condition, let also $V_{h,0} \subset V_h$ being the subspace:

$$V_{h,0} := \{v \in V_h\ :\ v|_{\partial\Omega} = 0\} \tag{13}$$

We replace the space V_0 with the space $V_{h,0}$ into the variational formulation and (12) is re-written:

$$\int_\Omega (\nabla u_h) \cdot (\nabla v) dx = \int_\Omega f v dx\ ,\ \forall\ v \in V_{h,0} \tag{14}$$

Again, we can re-write (14) by using a basis of hat functions for $V_{h,0}$ (i.e. $\{\phi_i\}_{i=1}^N$, where N is the number of nodes within triangulation $\boldsymbol{K}$). Thus, since $u_h \in V_{h,0}$, then

$$u_h = \sum_{j=1}^N \xi_j \phi_j \tag{15}$$

where ξ_j are unknown coefficients to be determined and $j = \overline{1, N}$.

Inserting (15) in (14), we end up with a $N \times N$ system of linear equations:

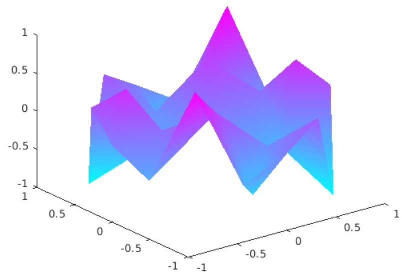
$$\sum_{j=1}^N \xi_j \left(\int_\Omega \nabla\phi_j \nabla\phi_i dx \right) = \int_\Omega f \phi_i dx\ ,\ i = \overline{1, N} \tag{16}$$

or

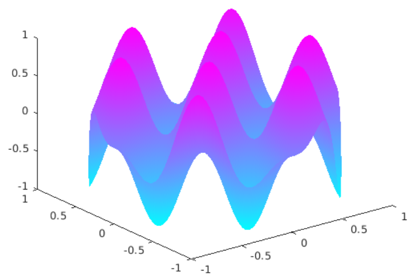
$$A\xi = b$$

where $A_{ij} = \int_\Omega \nabla\phi_j \nabla\phi_i dx$ and $b_i = \int_\Omega f\phi_i dx$ with $i, j = \overline{1, N}$.

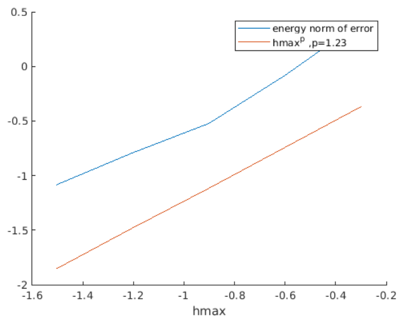
The next step is to implement a FEM solver. For five different mesh sizes m=[1/2,1/4,1/8,1/16,1/32] we compute the convergence rate of the Galerkin approximation with respect to the energy norm(p).Finally,we plot the solutions using the coarsest and finest meshes, as well as we present the graphs h_{max} vs energy norm of the error and h_{max} vs h_{max}^p.



Coarsest solution



Finest solution



Convergence

The order of convergence (p) is determined by the slope of log(h_{max})=log(energy norm of the error), since the graph is again in logarithmic scale. Its computed value is 1.23.

3.2 Problem B.2

This time we consider the prey-predator model with constant predator population density $w(x,t) = 1$ and no mutualist $u(x,t) = 0$. Hence, now the following scalar equation is derived:

$$\partial_t v - v(1-v) + \frac{v}{v+\alpha} - \delta_1 \Delta v = g \ , \ (x,t) \in \Omega \times (0,T] \tag{17}$$

while we assume that the source term is zero, i.e. $g(x,t) = 0$. At the same time we fix $\delta_1 = 0.01$ and $\alpha = 4$, as well as the homogeneous Neumann boundary condition is defined:

$$\partial_n v(x,t) = 0 \ , \ x \in \partial\Omega \ \& \ t \in (0,T]$$

Finally, the initial condition is denoted as:

$$v(x,0) = 1 + 20w(x) \ , \ w(x) \in [0,1]$$

where $w(x)$ a randomly generated number for every point $x \in \Omega$.

Again, we want to derive a linear system to be solved. In the beginning, we make use of continuous piece wise linear functions in space and we write the **Galerkin** finite element method. For the time discretization we take advantage of **Crack-Nicholson** method.

In the first place, a **variational formulation** of (17) is derived by multiplying with a test function $k(x,t)$ integrating using **Green's formula**.

$$\int_\Omega \partial_t v k dx - \int_\Omega v(1-v) k dx + \int_\Omega \frac{v}{v+\alpha} k dx - \int_\Omega \delta_1 \Delta v k dx = 0 \Rightarrow$$

$$\delta_1 \int_\Omega (\nabla v) \cdot (\nabla k) \ dx - \delta_1 \int_{\partial\Omega} (n \cdot \nabla v) k \ ds + \int_\Omega \partial_t v k dx - \int_\Omega k \left(v(1-v) - \frac{v}{v+\alpha} \right) dx = 0 \Rightarrow$$

$$\delta_1 \int_\Omega (\nabla v) \cdot (\nabla k) \ dx + \int_\Omega \dot{v} k dx - \int_\Omega v k dx + \int_\Omega \left(v^2 + \frac{v}{v+\alpha} \right) k \ dx = 0 \tag{18}$$

where in the last step we used that $\partial_n v(x,t) = 0$, on the boundary.

Next, we introduce the space:

$$V_o := \{ k : \|k\| + \|\nabla k\| \ < \infty \ ; \ k \mid_{\partial\Omega} = 0 \}$$

Hence, we find v such that, for $t \in (0,T]$ and $u \in V_o$, v verifies the **variational formulation** (33).

Now, it is the time to introduce the subspace V_{h_o} , where it is the space of all piece wise linear functions on a partition $k = \{k\}$ of Ω into triangles, which moreover satisfies the boundary conditions.

So , we find v_h such that:

$$\delta_1 \int_\Omega (\nabla v_h) \cdot (\nabla k) \ dx + \int_\Omega \dot{v_h} k dx - \int_\Omega v_h k dx + \int_\Omega \left(v_h^2 + \frac{v_h}{v_h+\alpha} \right) k \ dx = 0 \tag{19}$$

We set the non linear term $S(v_h) = v_h^2 + \frac{v_h}{v_h+\alpha}$.

and if we use the hat functions for V_{h_o} , i.e. ϕ_i , $i = \overline{1, n_i}$ & the space for $v_h = \sum_{j=1}^N \xi_j(t)\phi_j$ (where $\xi_j(t)$ time-dependant unknown coefficients), we can rewrite (33) as:

$$\delta_1 \sum_{j=1}^N \xi_j(t) \int_\Omega (\nabla\phi_j \cdot \nabla\phi_i) \ dx + \sum_{j=1}^N \dot{\xi}_j(t) \int_\Omega \phi_j \phi_i dx - \sum_{j=1}^N \xi_j(t) \int_\Omega \phi_j \phi_i dx = - \sum_{j=1}^N \int_\Omega (S_j(t)\phi_j) \, \phi_i \ dx \tag{20}$$

where in the last step we handled our non-linear term as its interpolant.

Finally, this can be written as:

$$M\dot{\xi} + A\xi - M\xi = -MS \tag{21}$$

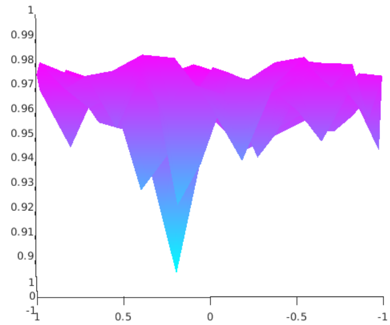
where M & A is the **mass** and **stiffness** matrix respectively, S the non-linear vector.

For the time discretization, we use **Crank-Nicholson** method as follows:

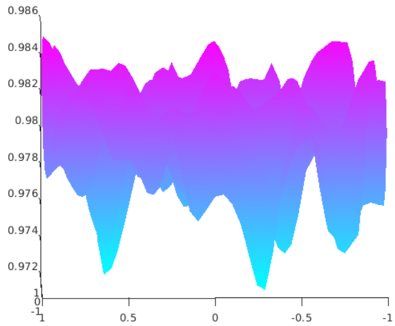
$$M\frac{\xi_{l+1}-\xi_l}{k_l}+(A-M)\frac{\xi_{l+1}+\xi_l}{2}=MS_l$$

where k_{t_l} shows the time steps $k_l = t_l - t_{l-1}$, $l=\overline{1,m}$

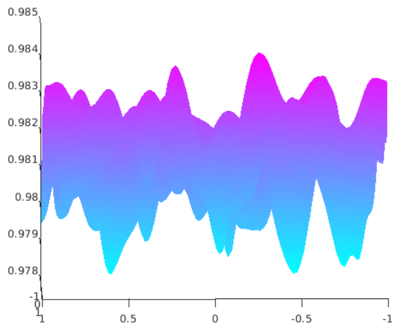
We implement now the variational problem obtained above and plot the solution at T=2 for 3 different meshes h_{max}=[1/5, 1/20, 1/40]. In addition, the population rates for every mesh as a function of time are presented.



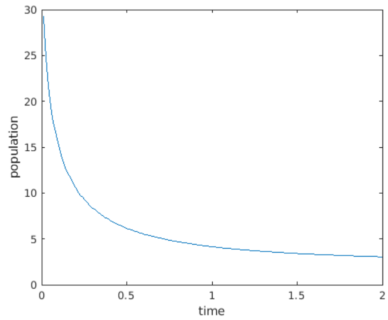
Solution for m=1/5



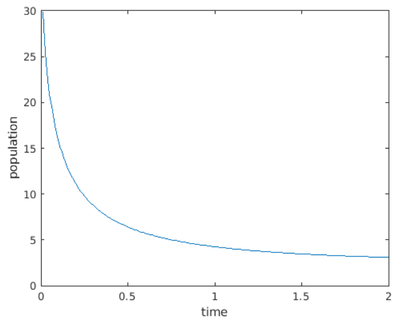
Solution for m=1/20



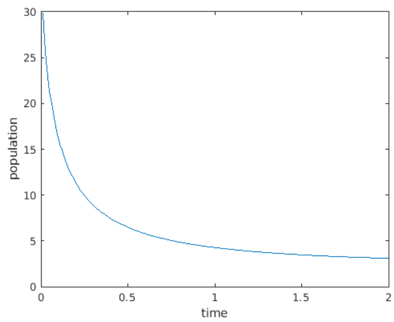
Solution for m=1/40



Population rate for m=1/5



Population rate for m=1/20



Population rate for m=1/40

4 Part C

In this part we consider the prey-predator model described above in equation system: (1) - (3) where we take the source terms $f(\mathbf{x},t) = g(\mathbf{x},t) = q(\mathbf{x},t) = 0$. The parameters are taken as: $\alpha = 0.4$, $\beta = 1$, $\gamma = 0.8$, $\zeta = 2$, $L_o = 0.4$, $l = 0.4$ and $m = 0.12$. The boundary condition is *homogeneous Neumann* in all boundary points (i.e. $\theta_n u(\mathbf{x},t) = \theta_n v(\mathbf{x},t) = \theta_n w(\mathbf{x},t) = 0 \ \forall \ \mathbf{x} \in \theta\Omega \ and \ t \in (0,T]$). We attempt to solve the problem by the use of "FEniCS" , an open-source finite element software for solving partial differential equations in any space dimensions and polynomial degrees. Therefore we write down the finite element discretization of the system: (1) - (3) by making use of the Crank -Nicholson scheme in time.

So, we obtain the variational formulation by multiplying each equation of the system: (1) - (3) by a test function (k_1, k_2, k_3) respectively, integrating the terms, summing up the equations and defining the bilinear and linear forms.

More specifically:

For mutualists' population:

Variational Form

$$\int_\Omega \dot{u} k_1 dx + \delta_1 \int_\Omega (\nabla u) \cdot (\nabla k_1) \ dx - \int_\Omega \alpha u k_1 dx + \alpha \int_\Omega \frac{u}{L_0 + lv} k_1 \ dx = 0 \tag{22}$$

Galerkin Finite Element Method (GFEM)

Find u_h such that:

$$\int_\Omega \dot{u_h} k_1 dx + \delta_1 \int_\Omega (\nabla u_h) \cdot (\nabla k_1) \ dx - \int_\Omega \alpha u_h k_1 dx + \alpha \int_\Omega \frac{u_h}{L_0 + lv} k_1 \ dx = 0 \tag{23}$$

GFEM after the use of hat functions and introduction of bilinear form

$$s_1(u,v) = \frac{u_h}{L_0 + lv} = \sum_{j=1}^{N} s_{1j} \phi_j \tag{24}$$

where we approximated the non-linear term by its linear interpolant and we rewrite GFEM:

$$\sum_{j=1}^{N} \dot{\xi}_j(t) \int_\Omega \phi_j \phi_i dx + \delta_1 \sum_{j=1}^{N} \xi_j(t) \int_\Omega (\nabla \phi_j \cdot \nabla \phi_i) \ dx - \alpha \sum_{j=1}^{N} \xi_j(t) \int_\Omega \phi_j \phi_i dx = -\alpha \sum_{j=1}^{N} \int_\Omega (s_{1j}(t) \phi_j) \, \phi_i \ dx \tag{25}$$

Derived Linear System after Crank-Nicholson Discretization

$$M \frac{\xi_{l+1} - \xi_l}{k_l} + (\delta_1 A - \alpha M) \frac{\xi_{l+1} + \xi_l}{2} = -\alpha M S_{1l} \tag{26}$$

For preys' population:

Variational Form

$$\int_\Omega \dot{v} k_2 dx + \delta_2 \int_\Omega (\nabla v) \cdot (\nabla k_2) \ dx - \int_\Omega \beta v k_2 + \beta v^2 k_2 + \frac{v\omega}{a + v + mu} k_2 \ dx = 0 \tag{27}$$

Galerkin Finite Element Method(GFEM)
Find u_h such that:

$$\int_\Omega \dot{v_h} k_2 dx + \delta_2 \int_\Omega (\nabla v_h) \cdot (\nabla k_2) \ dx - \int_\Omega \beta v_h k_2 + \beta v_h^2 k_2 + \frac{v\omega}{a + v_h + mu} k_2 \ dx = 0 \tag{28}$$

GFEM after the use of hat functions and introduction of bilinear form

$$s_2(u,v,\omega) = v^2 k_2 + \frac{v\omega}{a+v+mu} \tag{29}$$

where we approximated the non-linear term by its linear interpolant and we rewrite the GFEM:

$$\sum_{j=1}^{N} \dot{y}_j(t) \int_\Omega \phi_j \phi_i dx + \delta_2 \sum_{j=1}^{N} y_j(t) \int_\Omega (\nabla\phi_j \cdot \nabla\phi_i)\ dx - \beta \sum_{j=1}^{N} y_j(t) \int_\Omega \phi_j\phi_i dx = -\sum_{j=1}^{N} \int_\Omega (s_{2j}(t)\phi_j)\, \phi_i\ dx \tag{30}$$

Derived Linear System after Crank-Nicholson Discretization

$$M\frac{y_{l+1}-y_l}{k_l} + (\delta_2 A - \beta M)\frac{y_{l+1}+y_l}{2} = -MS_{2l} \tag{31}$$

For predators' population:

Variational Form

$$\int_\Omega \dot{\omega} k_3 dx + \delta_3 \int_\Omega (\nabla\omega)\cdot(\nabla k_3)\ dx + \int_\Omega \gamma\omega k_3 dx + \zeta \int_\Omega \frac{\omega v}{\alpha+v+mu} k_3\ dx = 0 \tag{32}$$

Galerkin Finite Element Method(GFEM)

Find u_h such that:

$$\int_\Omega \dot{\omega_h} k_3 dx + \delta_3 \int_\Omega (\nabla\omega_h)\cdot(\nabla k_3)\ dx + \int_\Omega \gamma\omega_h k_3 dx + \zeta \int_\Omega \frac{\omega_h v}{\alpha+v+mu} k_3\ dx = 0 \tag{33}$$

GFEM after the use of hat functions and introduction of bilinear form

$$s_3(u,v,\omega) = -\frac{\zeta\omega_h v}{a+v+mu} = \sum_{j=1}^{N} s_{3j}\phi_j \tag{34}$$

where we approximated the non-linear term by its linear interpolant and we rewrite GFEM:

$$\sum_{j=1}^{N} \dot{z}_j(t) \int_\Omega \phi_j \phi_i dx + \delta_3 \sum_{j=1}^{N} z_j(t) \int_\Omega (\nabla\phi_j \cdot \nabla\phi_i)\ dx + \gamma \sum_{j=1}^{N} z_j(t) \int_\Omega \phi_j\phi_i dx = -\sum_{j=1}^{N} \int_\Omega (s_{3j}(t)\phi_j)\, \phi_i\ dx \tag{35}$$

Derived Linear System after Crank-Nicholson Discretization

$$M\frac{z_{l+1}-z_l}{k_l} + (\delta_3 A + \gamma M)\frac{z_{l+1}+z_l}{2} = -MS_{3l} \tag{36}$$

When using the "FEniCS", it is better to consider the system of pdes as a vector of equations. The test functions are collected in a vector too, and the variational formulation is the inner product of the vector PDE and the vector test function.

$$\int_\Omega \left(\nabla\left(\frac{u_{n+1}+u_n}{2}\right)\cdot\nabla k_1\right)\,dx + \int_\Omega \frac{1}{\delta_1}\left(\frac{u_{n+1}-u_n}{\Delta t}\right)k_1 dx - \int_\Omega \frac{\alpha}{\delta_1}\left(\frac{u_{n+1}+u_n}{2}\right)k_1 dx + \int_\Omega \frac{\alpha}{\delta_1}\left(\frac{u_n^2}{L_o+lu_n}\right)k_1\ dx = 0 \tag{37}$$

$$\int_\Omega \left(\nabla\left(\frac{v_{n+1}+v_n}{2}\right)\cdot\nabla k_2\right)\,dx + \int_\Omega \frac{1}{\delta_2}\left(\frac{v_{n+1}-v_n}{\Delta t}\right)k_2 dx - \int_\Omega \frac{\beta}{\delta_2}\left(\frac{v_{n+1}+v_n}{2}\right)k_2 dx + \int_\Omega \frac{\beta v_n^2}{\delta_2}k_2\ dx + \int_\Omega \frac{1}{\delta_2}\left(\frac{v_n w_n}{\alpha+v_n+mu_n}\right)k_2\ dx = 0 \tag{38}$$

$$\int_\Omega \left(\nabla\left(\frac{w_{n+1}+w_n}{2}\right)\cdot\nabla k_3\right)\,dx + \int_\Omega \frac{1}{\delta_3}\left(\frac{w_{n+1}-w_n}{\Delta t}\right)k_3 dx + \int_\Omega \frac{\gamma}{\delta_3}\left(\frac{w_{n+1}+w_n}{2}\right)k_3 dx - \int_\Omega \frac{\zeta}{\delta_3}\left(\frac{v_n w_n}{\alpha+v_n+mu_n}\right)k_3\ dx = 0 \tag{39}$$

and in the end, we sum up Eq.(37) with (38) and (39). Then, we distinguish the summation into the bilinear and linear form with the use of *rhs()* & *lhs()* functions respectively.

4.1 Problem C.1

We try to implement a "FEniCS" solver who solves the system by taking firstly a zero mutualist population ($u = 0$) and secondly the following initial condition:

$$u_o = 0.01v_o\ ,$$

$$v_o = \frac{4}{15} - 2\cdot 10^{-7}(x_1 - 0.1x_2 - 350)(x_1 - 0.1x_2 - 67)\ ,$$

$$w_o = \frac{22}{45} - 3\cdot 10^{-5}(x_1 - 450) - 1.2\cdot 10^{-4}(x_2 - 15)\ .$$

We will run our code with $\delta_1 = 1$, $\delta_2 = 1$, $\delta_3 = 1$ and $\Delta t = 0.5$ until $T = 500$ by the use of **circle.xml**. Then we plot the solutions u and v at times $T = 0, 100, 200, 300, 400$. Next we run the code until $T = 1000$ and plot the solutions at the final time. Then we plot the population rates for prey and predator at the same figure and discuss why they behave so. Then, we set and run the code while $T = 1000$. We plot all the population rates in one figure. The following graphs are subsequently presented:

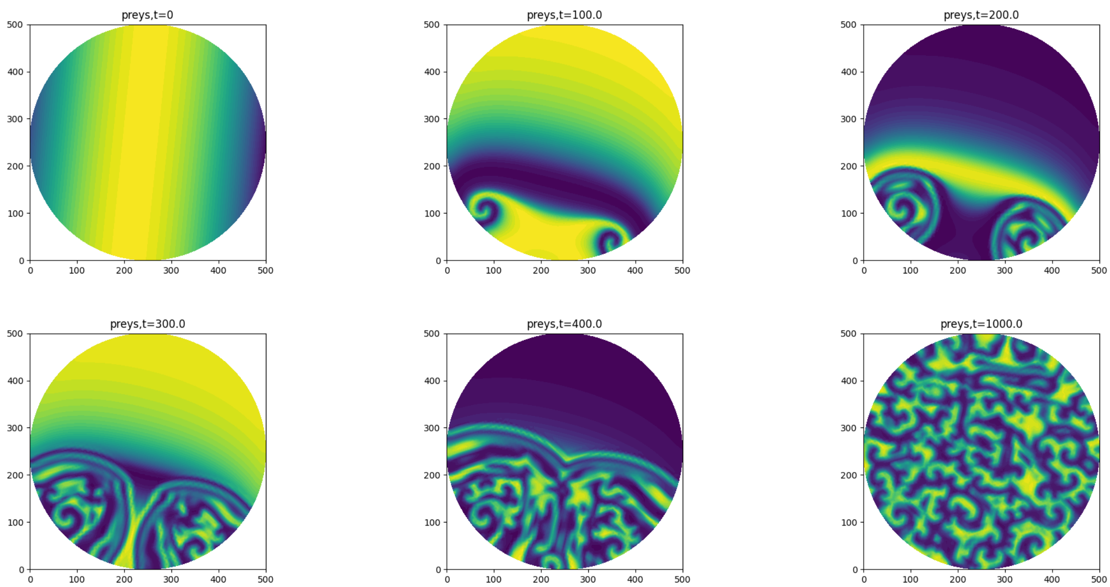


Figure 1: Solution of v , ($u_0 = 0$)

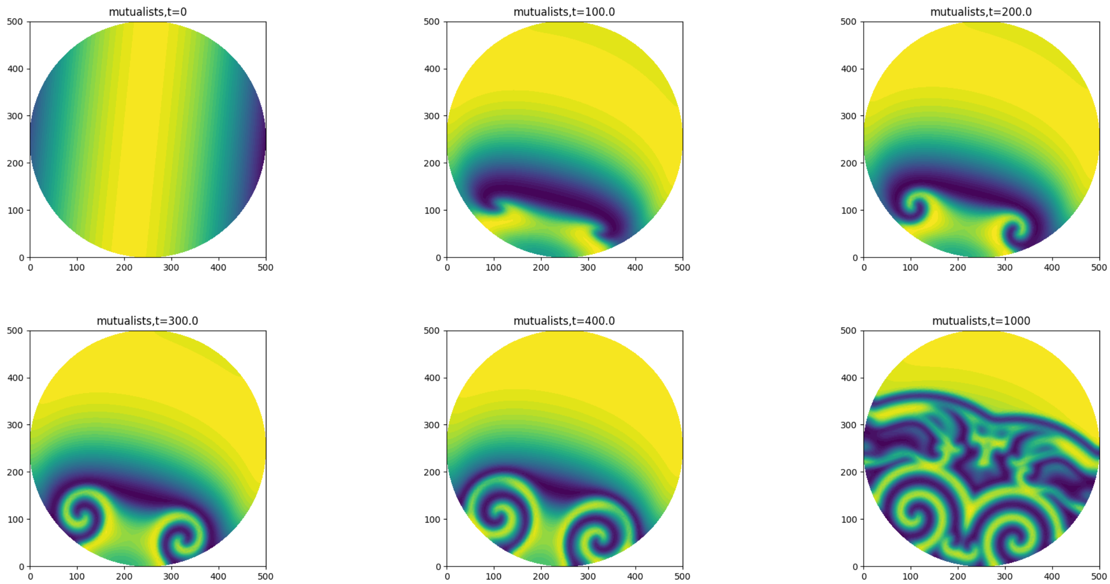


Figure 2: Solution of u , ($u_0 = 0.01v_0$)

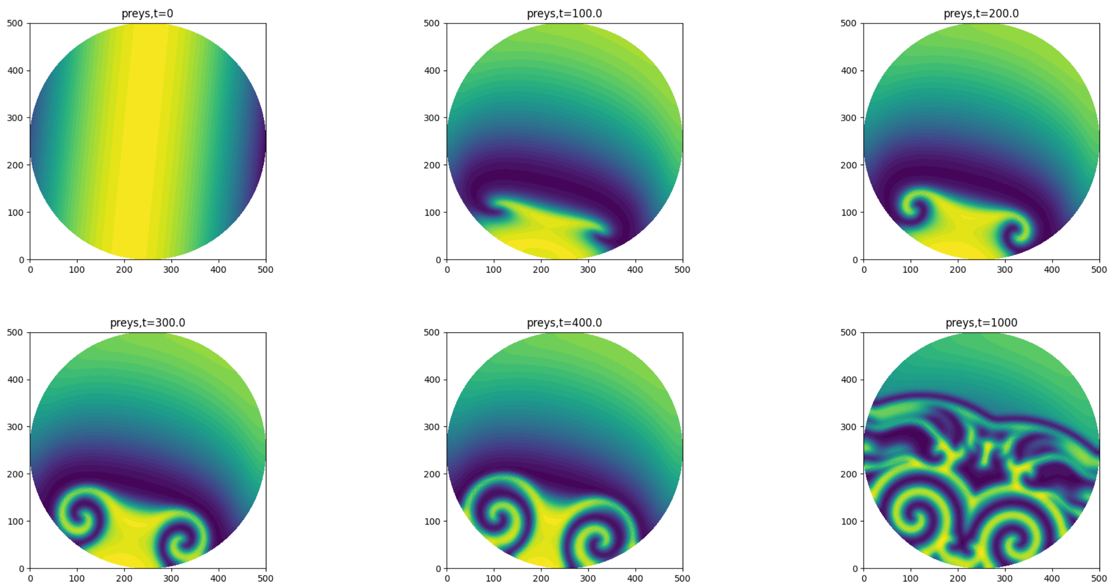
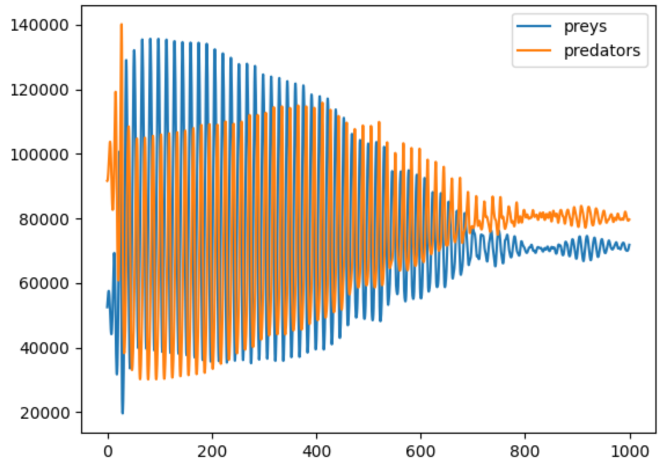
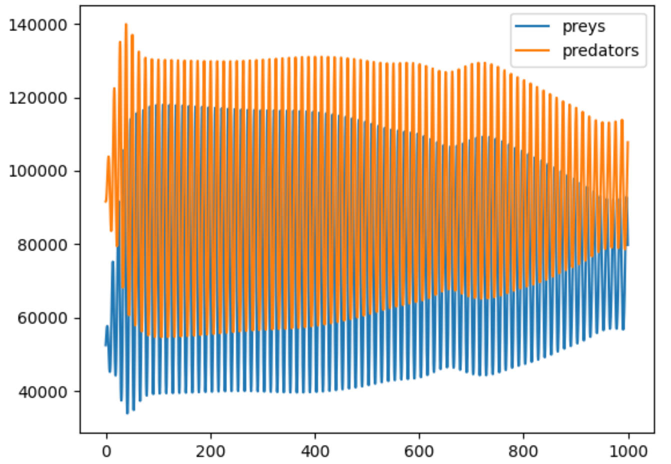


Figure 3: Solution of v ,($u_0 = 0.01v_0$)

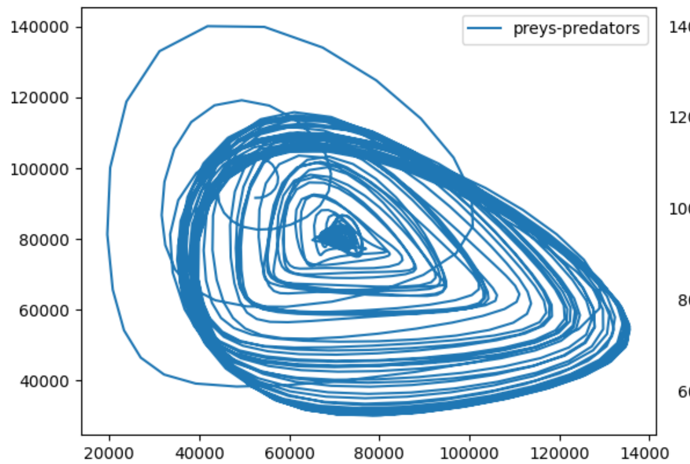


(a) Population rate ($u_0 = 0$)

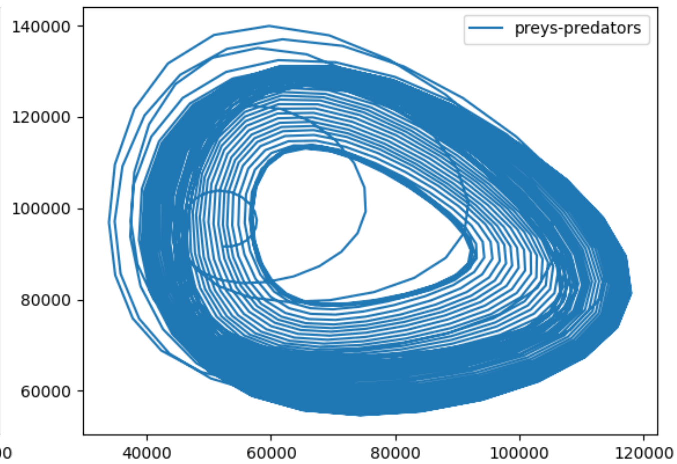


(b) Population rate($u_0 = 0.01v_0$)

Figure 4: Here, we see the population rates for both cases

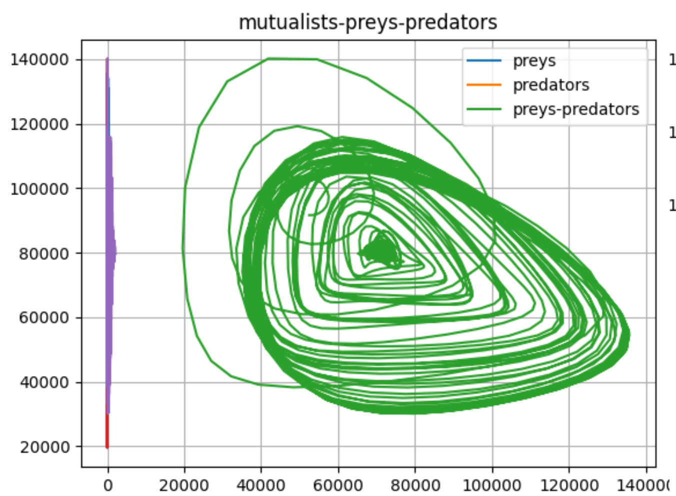


(a) Phase Portrait(v,w) for $u_0 = 0$

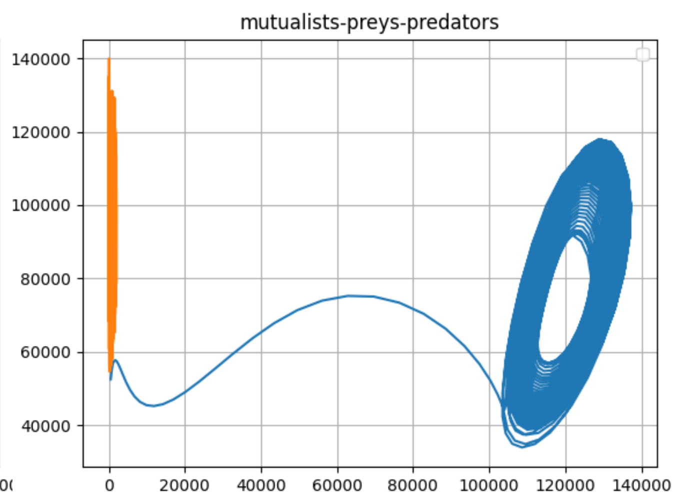


(b) Phase Portrait (v,w) for $u_0 = 0.01v_0$

Figure 5: Here, we see the phase portraits for both cases



(a) Phase Portrait(u,v,w) for $u_0 = 0$



(b) Phase Portrait (u,v,w) for $u_0 = 0.01v_0$

Figure 6: In these figures we present the whole phases for both cases

4.2 Problem C.2

In our next problem we are going to model the dynamics of prey-predator-mutualist in Sweden. We suppose that the initial population is randomly defined bellow Sundsvall as

$$u_o(\mathbf{x}_i) = \frac{5}{1000} random(\mathbf{x}_i),$$

$$v_o(\mathbf{x}_i) = \frac{1}{2}(1 - random(\mathbf{x}_i)),$$

$$w_o(\mathbf{x}_i) = \frac{1}{4} + \frac{1}{2} random(\mathbf{x}_i),$$

while for the region over Sundsvall the initial conditions are defined as:

$$u_o(\mathbf{x}_i) = \frac{1}{100}$$

$$v_o(\mathbf{x}_i) = \frac{1}{100}$$

$$w_o(\mathbf{x}_i) = \frac{1}{100}$$

for every nodal point $\mathbf{x}_i \in \Omega$. Here the function $random(\mathbf{x}_i) : \mathbb{R}^2 \longmapsto [0, 1]$ is a random number between zero and one. We will run our code with $\Delta t = 0.5$ until $T = 1200$ by using the **sweden.xml.gz**. Then we plot the solutions u and v at times $T = 0, 100, 300, 600, 1200$ and the population rates for u, v and w. Last but not least plot the phase portrait of the population rates.

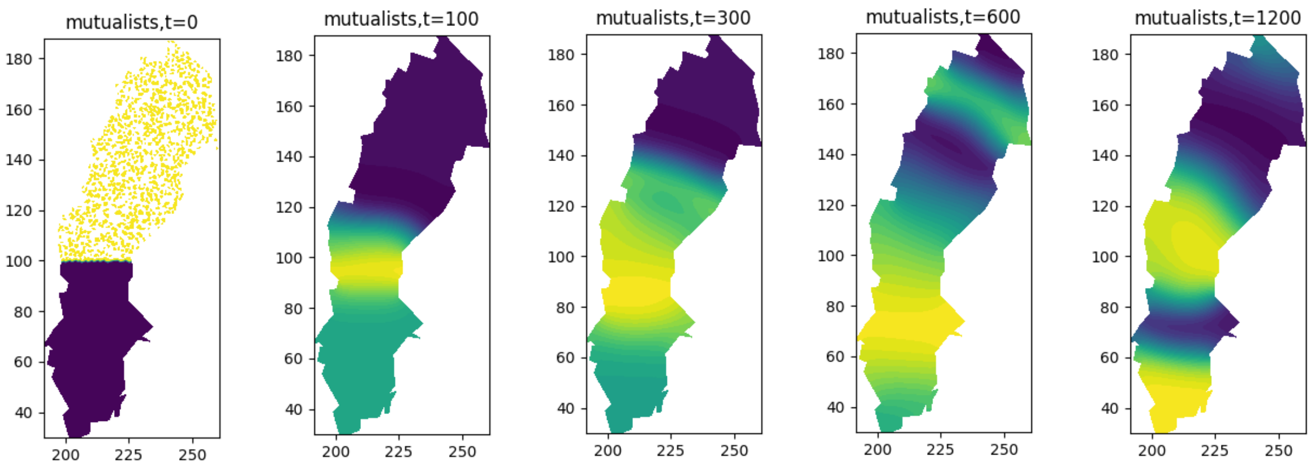


Figure 7: Solution of u

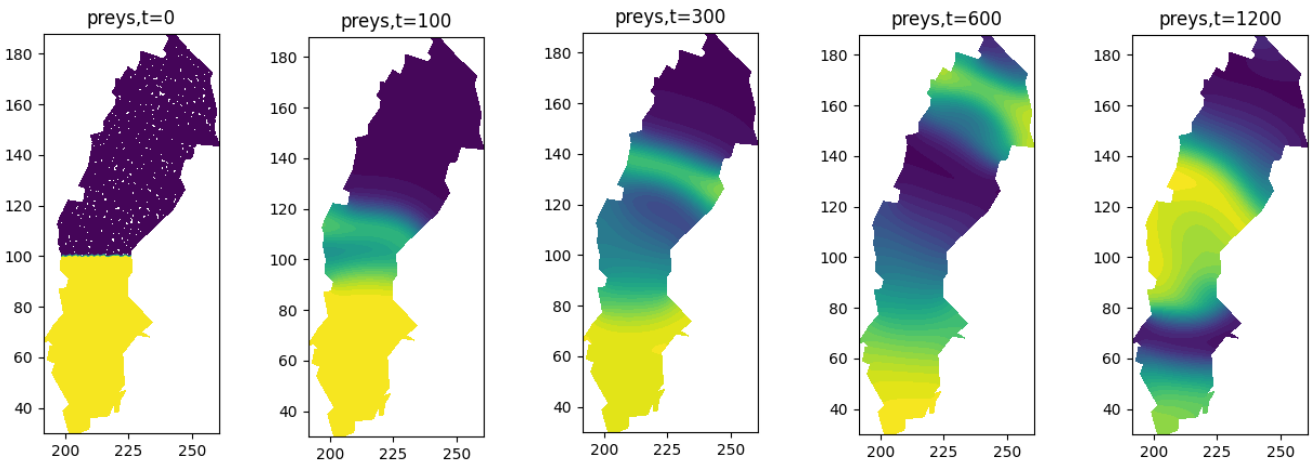
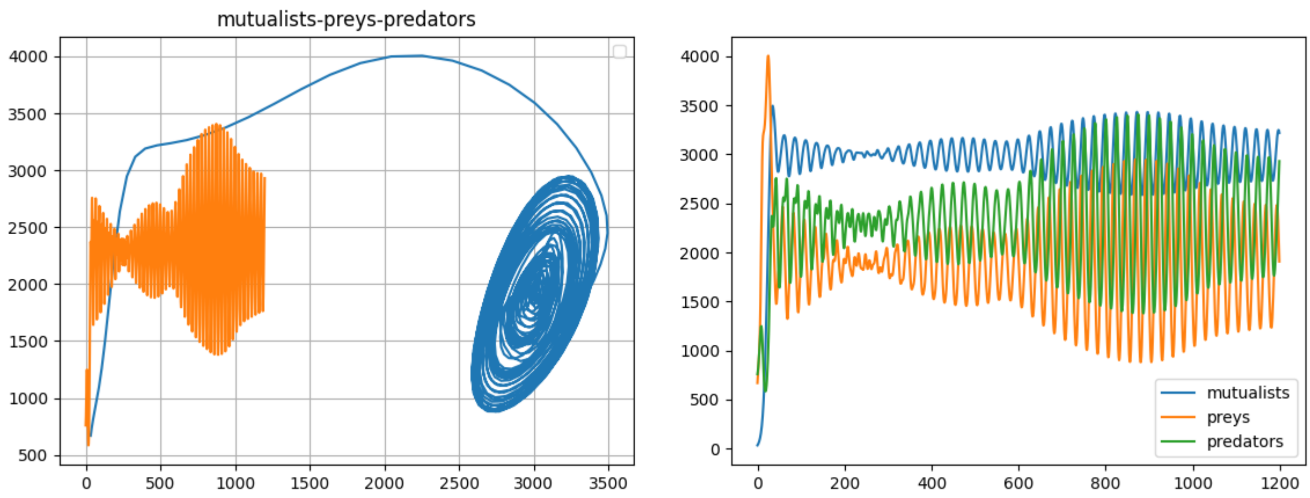


Figure 8: Solution of v



(a) Phase Portrait (u,v,w)

(b) Population rate (u,v,w)

Figure 9: The phase portrait & population rate for mutualists, preys and predators

5 Conclusions

In this project, we examined a problem of two-dimensional mutualism dynamics and specifically the Lotka-Volterra system described by a set of partial-differential equations.

For the solution of this system, step by step we developed a Finite Element Solver, whom the approximated solution is evaluated through the appropriate graphs.

The population rates of Preys and Predators for different mutualists population is presented in Figure 4. We see that the equilibrium between preys-predators seems unstable (Fig.4a) and the cycles between their population density have an irregular form (Fig.5a). On the other hand the presence of mutualists is associated with a greater equilibria (Fig.4b) and a smoother form of cycles (Fig.4b).

In addition, seeing Figure 1 and Figure 2, one can come to the conclusion that with the absence of mutualists, preys and predators compete constantly. Therefore, for different time steps we observe the dominance of either one species or the other. On the contrary, mutualists ensure an equilibrium which leads to the survival of all populations.

Further exploration can be achieved looking C2 Figures.

A Matlab-Code

A.1 for Problem A.2

```matlab
clear all; clc ; close all


%Boundary conditions
a=-1;
b=1;

%Initial values
delta=0.1;
TOL=10^(-5);
N=12;
lamda=0.9; %for refinement



% spatial discretization:
xvec=linspace(a,b,N+1);
NN=[];%nodes at each iteration
NN(1)=N;

%Initialize solution vectors
xi=delta*A(xvec)\B(xvec);
zeta=M(xvec)\(-A(xvec)*xi);

%Initialize eta2
eta2=eta2_as(xvec,delta,zeta);

%Initialize sum of errors
summa_eta=[];
summa_eta(1)=sum(sqrt(eta2));



 while sum(eta2)>TOL
    %Refinement of the elements with biggest contribution error
    for i = 1:length(eta2)
        if eta2(i) > lamda*max(eta2) % if large residual
            xvec = [xvec (xvec(i+1)+xvec(i))/2]; % insert new node point
        end
    end
     xvec = sort(xvec); % sort node points accendingly

     NN=[NN,length(xvec)];


  %Compute uh ,laplace
   xi=delta*A(xvec)\B(xvec);
   zeta=M(xvec)\(-A(xvec)*xi);

  %Compute eta2(uh)
   eta2=eta2_as(xvec,delta,zeta);
   summa_eta=[summa_eta,sum(sqrt(eta2))];

 end


   xi=delta.*A(xvec)\B(xvec);
   zeta=M(xvec)\(-A(xvec)*xi);
   %f(xvec)
   F= arrayfun( @(x) f(x), xvec);



% %Plot the solution uh
     figure(1);
     subplot(2,2,1)
     xlabel('x domain')
     plot(xvec,xi,'DisplayName','uh')
     legend show
%
% % % %PLot R(uh)
%      %Compute Residual
     R_uh=F+delta*zeta';
     subplot(2,2,2)
     xlabel('x domain')
     plot(xvec,R_uh,'DisplayName','R(uh)')
     legend show
%
% %Plot Grid Size distribution
   subplot(2,2,3)
   plot(xvec(2:end),1./diff(xvec),'DisplayName','Grid size Distribution')
   legend show
%
% %Plot error indicator
    subplot(2,2,4)
   plot(xvec(2:end),(eta2)','DisplayName','eta(uh)')
    legend show
```

```
 %Plot nodes-sum of errors at each iteration
     figure(6);

     plot(log10(NN),log10(summa_eta),'DisplayName','eta_v')
     hold on
     plot(log10(NN),log10(1./(NN)),'DisplayName','N^-1 ')
     hold on

     legend show





%%%%%%%%%%%%%%%%%%%%%%%%%%% All functions used %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%function f(x)
function f = f(x)
 g=10.*x.*sin(7*pi.*x);
 if g>abs(x)
     f=abs(x);

 elseif g<-1*abs(x)
     f=-1*abs(x);

 else
     f=g;
 end
end

 function B=B(x)
% Input is a vector x of node coords.
N = length(x) - 1;
B = zeros(N+1, 1);
    for i = 1:N
        h = x(i+1) - x(i);
        n = [i i+1];
        B(n) = B(n) + [f(x(i)); f(x(i+1))]*h/2;
    end
 end

function M=M(x)
    %
    % Returns the assembled mass matrix A.
    % Input is a vector x of node coords.
    %
    N = length(x) - 1;
    M = zeros(N+1, N+1); % Initialize matrix to zero
    for i = 1:N % loop over elements
        h = x(i+1) - x(i);
        n = [i i+1];
        M(n,n) = M(n,n) + [1/3 1/6; 1/6 1/3].*h; % Our diagonal consists of 2h/3, our others consist of h/6
    end

    M(1,1)=1.e+6;
    M(N+1,N+1)=1.e+6;
end
% Returns the assembled stiffness matrix A.
function A=A(x)
    % 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;

    A(N+1,N+1)=1.e+6;
end


%Calculation of element residuals
function eta2=eta2_as(x,delta,zeta)

N=length(x)-1;
eta2= zeros(N,1); % allocate element residuals

for i = 1:N  % loop over elements
    h = x(i+1)-x(i);
    a2 = f(x(i))+delta*zeta(i);% temporary variables
    b2 = f(x(i+1))+delta*zeta(i);
    t = (a2^2+b2^2)*h/2; % integrate f^2. Trapezoidal rule
    eta2(i) = (h*sqrt(t))^2; % element residual
end

end
```

A.2 for Problem A.3

```
clear all; clc ; close all


%Boundary conditions
a=-1;
b=1;

%Initial values
delta=0.1;
TOL=10^(-5);
N=12;
lamda=0.9; %for refinement



% spatial discretization:
xvec=linspace(a,b,N+1);
NN=[];%nodes at each iteration
NN(1)=N;

%Initialize solution vectors
xi=delta*A(xvec)\B(xvec);
zeta=M(xvec)\(-A(xvec)*xi);

%Initialize eta2
eta2=eta2_as(xvec,delta,zeta);

%Initialize sum of errors
summa_eta=[];
summa_eta(1)=sum(sqrt(eta2));



 while sum(eta2)>TOL
    %Refinement of the elements with biggest contribution error
    for i = 1:length(eta2)
        if eta2(i) > lamda*max(eta2) % if large residual
            xvec = [xvec (xvec(i+1)+xvec(i))/2]; % insert new node point
        end
    end
     xvec = sort(xvec); % sort node points accendingly

     NN=[NN,length(xvec)];


  %Compute uh ,laplace
   xi=delta*A(xvec)\B(xvec);
   zeta=M(xvec)\(-A(xvec)*xi);

  %Compute eta2(uh)
   eta2=eta2_as(xvec,delta,zeta);
   summa_eta=[summa_eta,sum(sqrt(eta2))];

 end


   xi=delta.*A(xvec)\B(xvec);
   zeta=M(xvec)\(-A(xvec)*xi);
   %f(xvec)
   F= arrayfun( @(x) f(x), xvec);



% %Plot the solution uh
     figure(1);
     subplot(2,2,1)
     xlabel('x domain')
     plot(xvec,xi,'DisplayName','uh')
     legend show
%
% % % %PLot R(uh)
%      %Compute Residual
     R_uh=F+delta*zeta';
     subplot(2,2,2)
     xlabel('x domain')
     plot(xvec,R_uh,'DisplayName','R(uh)')
     legend show
%
% %Plot Grid Size distribution
   subplot(2,2,3)
   plot(xvec(2:end),1./diff(xvec),'DisplayName','Grid size Distribution')
   legend show
%
% %Plot error indicator
    subplot(2,2,4)
   plot(xvec(2:end),(eta2)','DisplayName','eta(uh)')
    legend show

 %Plot nodes-sum of errors at each iteration
     figure(6);
```

```matlab

    plot(log10(NN),log10((summa_eta)),'DisplayName','eta_v')
    hold on
    plot(log10(NN),log10(1./(NN)),'DisplayName','N^-1 ')
    hold on

    legend show





%%%%%%%%%%%%%%%%%%%%%%%%%%% All functions used %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%function f(x)
function f = f(x)
 g=10.*x.*sin(7*pi.*x);
 if g>abs(x)
     f=abs(x);

 elseif g<-1*abs(x)
     f=-1*abs(x);

 else
     f=g;
 end
end

 function B=B(x)
% Input is a vector x of node coords.
N = length(x) - 1;
B = zeros(N+1, 1);
    for i = 1:N
        h = x(i+1) - x(i);
        n = [i i+1];
        B(n) = B(n) + [f(x(i)); f(x(i+1))]*h/2;
    end
 end

function M=M(x)
    %
    % Returns the assembled mass matrix A.
    % Input is a vector x of node coords.
    %
    N = length(x) - 1;
    M = zeros(N+1, N+1); % Initialize matrix to zero
    for i = 1:N % loop over elements
        h = x(i+1) - x(i);
        n = [i i+1];
        M(n,n) = M(n,n) + [1/3 1/6; 1/6 1/3].*h; % Our diagonal consists of 2h/3, our others consist of h/6
    end

    M(1,1)=1.e+6;
    M(N+1,N+1)=1.e+6;
end
% Returns the assembled stiffness matrix A.
function A=A(x)
    % 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;

    A(N+1,N+1)=1.e+6;
end


%Calculation of element residuals
function eta2=eta2_as(x,delta,zeta)

N=length(x)-1;
eta2= zeros(N,1); % allocate element residuals

for i = 1:N  % loop over elements
    h = x(i+1)-x(i);
    a2 = f(x(i));% temporary variables
    b2 = f(x(i+1));
    t = (a2^2+b2^2)*h/2; % integrate f^2. Trapezoidal rule
    eta2(i) = (h*sqrt(t))^2; % element residual
end

end
```

A.3 for Problem B.1

```
clear all;   clc



%Define our geometry
geometry = @circleg ;
hmax=[1/2,1/4,1/8,1/16,1/32];%1/4,1/8,1/16,1/32%;


EnE=zeros(1,length(hmax));
pp=zeros(1,length(hmax));


for i =1:length(hmax)
    [p ,e , t ] = initmesh( geometry , 'hmax' , hmax(i) );
    m=size(t,2); %number of elements
    [A,B,uex]=assembler(p,t);



% %Boundary conditions
      I=eye(length(p));
      A(e(1,:),:) = I(e(1,:),:);
      B(e(1,:))=uex(e(1,:));


    xi=A\B;

      %Plot solutions
%        figure();
%        pdeplot(p,[],t,'XYData',xi,'ZData',xi,'ColorBar','off')



    %Error
    err=uex-xi;
    EnE(i)=sqrt(err'*A*err);








end
  figure()
  hold on
  xlabel('hmax')
  loglog(hmax,EnE,'DisplayName','energy norm of error')

  pp=polyfit(log10(hmax),log10(EnE),1);   %Convergence Rate
%   figure(1)
  loglog(hmax,hmax.^pp(1),'DisplayName','hmax^p ,p=1.23')
  legend show





%%%%%%%%%%%%%%%%%%%%%%%%% Functions used %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%








%Computation of aplha and hat-gradients

function [b,c] = gradients(x,y,KK)

    b=[y(2)-y(3); y(3)-y(1); y(1)-y(2)]/2/KK;
    c=[x(3)-x(2); x(1)-x(3); x(2)-x(1)]/2/KK;
end


function[f]=f(x,y)
    f=8*((pi)^2).*sin(2.*pi.*x).*sin(2.*pi.*y);
end



%Stiffness matrix assembler and Load vector assembler

function [A,B,uex]=assembler(p,t)

    n=size(p,2); %number of nodes
    m=size(t,2); %number of elements
```

```
    A=sparse(n,n);
    B=zeros(n,1);
    uex=zeros(n,1);


    for K=1:m
        nodes=t(1:3,K);
        x=p(1,nodes); % x coord of nodes
        y=p(2,nodes); % y coord of nodes
        KK=polyarea(x,y);
       [b,c] = gradients(x,y,KK); %computation of gradients of hat functions
       xc=mean(x);
       yc = mean(y);
       bk=f(xc,yc)*KK/3;
       AK = (b*b'+c*c')*KK; % element stiffness matrix
       B(nodes)=B(nodes)+bk;
       A(nodes,nodes) = A(nodes,nodes)+AK;
       uex(nodes)=f(x,y)/(8*((pi)^2));
    end
end
```

A.4 for Problem B.2

```
clear all; clc;



%Define our geometry
geometry = @circleg ;
hmax=[1/5,1/20,1/40];

%Initial Values
alpha=4;
delta=0.0005;

for i =1:length(hmax)
    [p ,e , t ] = initmesh( geometry , 'hmax' , hmax(i) );



     %%%%% FEM SOLVER %%%%%
     xi_old=1+20*rand(length(p(1,:)),1); % solution (t=0)
     xi_final=xi_old;
     S=assemblerb(xi_old);
     kl=0.01; %time step
     time=0;
     T=2;

     %Initial values for pop.rate
     pop_counter=1;
     time_matrix=[];


     [A,M]=assembler(p,t);
     A=delta*A;





       while time<T %Crank-Nicholson
           S=assemblerb(xi_old);
           xi_final=(M/kl-M/2+A)\(M*xi_old/kl+M*xi_old/2-M*S-A*xi_old);


           xi_old=xi_final;
           time=time+kl;




       %%%%%% Population Rate %%%%%%%%%
                 integral=0;

                pop(pop_counter)=assemblerc(xi_old,integral,t,p);
                time_matrix(pop_counter)=time;

                pop_counter=pop_counter+1;
       end







        figure()
        pdesurf(p,t,xi_old)

        figure()
        plot(time_matrix,pop)
        xlabel("time")
        ylabel("population")











end





%%%%%%%%%%%%%%%%%%%%%%%%% Functions used %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Computation of aplha and hat-gradients

function [b,c] = gradients(x,y,KK)

    b=[y(2)-y(3); y(3)-y(1); y(1)-y(2)]/2/KK;
    c=[x(3)-x(2); x(1)-x(3); x(2)-x(1)]/2/KK;
end





%Stiffness matrix assembler and Load vector assembler

function [A,M]=assembler(p,t)

    n=size(p,2); %number of nodes
    m=size(t,2); %number of elements
    A=sparse(n,n); %Initialize stiffness matrix
    M=sparse(n,n); %Initialize mass matrix



    for K=1:m
        nodes=t(1:3,K);
        x=p(1,nodes); % x coord of nodes
        y=p(2,nodes); % y coord of nodes
        KK=polyarea(x,y);
       [b,c] = gradients(x,y,KK); %computation of gradients of hat functions

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

       MK=KK/12*[2,1,1; 1,2,1;1,1,2];% element mass matrix
       M(nodes,nodes)=M(nodes,nodes)+MK;
    end



    end


function S=assemblerb(xi)
    alpha=4;
    S=xi.^2+(xi./(xi+alpha));
 end



 function pop=assemblerc(xi,integral,t,p)


    m=size(t,2);
    for K=1:m
      nodes=t(1:3,K);
      x=p(1,nodes); % x coord of nodes
      y=p(2,nodes); % y coord of nodes
      KK=polyarea(x,y);
      integral=integral+(KK/3)*sum(xi(nodes));
    end
    pop=integral;

end
```

A.5 for Problem C.1

```python
#!/usr/bin/env python
# coding: utf-8

# In[148]:


from dolfin import *
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d


# In[149]:


#Create a mesh and define function space
mesh = Mesh("/home/chris/Desktop/Computational Science/Courses/Applied Finite Element Methods/Project PartC/circle
    .xml.gz")


# In[150]:


# Construct the finite element space
P1 = FiniteElement("Lagrange",mesh.ufl_cell(),1)
TH = P1 * P1 * P1  # Taylor Hood Element
V = FunctionSpace(mesh,TH) # Creates space for u, v, w


# In[151]:


#Define parameters:
# Define parameters:
T = 1000
dt = 0.5
delta1 = Constant(1.0)
delta2 = Constant(1.0)
delta3 = Constant(1.0)
alpha = Constant(0.4)
beta = Constant(1)
gamma = Constant(0.8)
zeta = Constant(2.0)
L_0 = Constant(0.4)
l = Constant(0.6)
m = Constant(0.12)


# In[152]:


# Class representing the initial conditions
class InitialConditions(UserExpression):
    def eval(self,values,x):
        values[0]=0.01*(4/15-2*pow(10,-7)*(x[0] - 0.1*x[1] - 350)*(x[0] - 0.1*x[1] - 67))
        values[1]=4/15-2*pow(10,-7)*(x[0] - 0.1*x[1] - 350)*(x[0] - 0.1*x[1] - 67)
        values[2]=22/45-3*pow(10,-5)*(x[0] - 450)-1.2*pow(10,-4)*(x[1] - 15)

    def value_shape(self):
        return (3,)


# In[153]:


# Define Test and Trial Functions

#Trial-Initial Conditions
indata = InitialConditions(degree=2)
trial_0=TrialFunction(V)
trial_0=interpolate(indata, V)

#Test
k=TestFunction(V)

#Trial
trial=TrialFunction(V)

#For 3-PDEs System

#Trial_Initial
uz=trial_0[0]
vz=trial_0[1]
wz=trial_0[2]

#Test
k1=k[0]
k2=k[1]
k3=k[2]

```

```
#Trial
u=trial[0]
v=trial[1]
w=trial[2]



# In[154]:


# #Plot solution at t=0

# # Mutualists (t=0)
# plot(trial_0[0])
# plt.title("mutualists,t="+str(0))
# plt.savefig("u"+str(0)+".png")
# plt.show()
# plt.clf()

# # Preys (t=0)
# plot(trial_0[1])
# plt.title("preys,t="+str(0))
# plt.savefig("v"+str(0)+".png")
# plt.show()
# plt.clf()


# In[155]:


#Define Variational Problem - System of PDEs
F=(((u-uz)/dt)*k1)*dx \
  +(((v-vz)/dt)*k2)*dx \
  +(((w-wz)/dt)*k3)*dx \
  +(delta1*inner(grad(u+uz),grad(k1)))*0.5*dx \
  +(delta2*inner(grad(v+vz),grad(k2)))*0.5*dx \
  +(delta3*inner(grad(w+wz),grad(k3)))*0.5*dx \
  -(alpha*((u+uz)*0.5)*k1)*dx \
  -(beta*((v+vz)*0.5)*k2)*dx \
  +(gamma*((w+wz)*0.5)*k3)*dx \
  +(alpha*uz**2)/(L_0+l*vz)*k1*dx\
  +beta*(vz**2)*k2*dx \
  +vz*wz/(alpha + vz + m*uz)*k2*dx \
  -(zeta*(wz*vz)/(alpha+vz+m*uz))*k3*dx



# In[156]:


a= lhs(F)
L=rhs(F)
trial=Function(V)


# In[157]:


upop=[assemble(uz*dx)]
vpop=[assemble(vz*dx)]
wpop=[assemble(wz*dx)]

time=[]





# In[158]:


t=0
while t < T:
    time.append(t)
    t=t+0.5

    solve(a==L,trial)
    trial_0.assign(trial)

    utemp=assemble(trial_0[0]*dx)
    vtemp=assemble(trial_0[1]*dx)
    wtemp=assemble(trial_0[2]*dx)

    upop.append(utemp)
    vpop.append(vtemp)
    wpop.append(wtemp)

     # Plot

    # if int(t)%100==0:

    #     #Mutualists
```

```
    #     plot(trial_0[0])
    #     plt.title("mutualists,t="+str(t))
    #     plt.savefig("u"+str(t)+".png")
    #     plt.show()
    #     plt.clf()

    #     #Preys

    #     plot(trial_0[1])
    #     plt.title("preys,t="+str(t))
    #     plt.savefig("v"+str(t)+".png")
    #     plt.show()
    #     plt.clf()


























# In[161]:


vpop=vpop[:-1]
wpop=wpop[:-1]
upop=upop[:-1]
# plt.plot(time,vpop, label="preys")
# plt.plot(time,wpop, label="predators")
# plt.legend()
# plt.show
# plt.savefig("u,v"+str(t)+".png")
# plt.clf


# #Phase portrait (preys-predators)
# plt.plot(vpop,wpop, label="preys-predators")
# plt.legend()
# plt.show
# plt.savefig("v,w"+".png")
# plt.clf

#Phase portrait (mutualists-preys-predators)
plt.plot(upop,vpop,wpop)
plt.title("mutualists-preys-predators")
plt.legend()
plt.grid()
plt.show
plt.savefig("u,v,w"+".png")
plt.clf




# In[160]:


# Mutualists (t=1000)
# plot(trial_0[0])
# plt.title("mutualists,t="+str(1000))
# plt.savefig("u"+str(t)+".png")
# plt.show()
# plt.clf()

# # Preys (t=1000)
# plot(trial_0[1])
# plt.title("preys,t="+str(1000))
# plt.savefig("v"+str(t)+".png")
# plt.show()
# plt.clf()


# In[ ]:
```

A.6 for Problem C.2

```
#!/usr/bin/env python
# coding: utf-8

# In[67]:


from dolfin import *
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import random


# In[68]:


#Create a mesh and define function space
mesh = Mesh("/home/chris/Desktop/Computational Science/Courses/Applied Finite Element Methods/Project PartC/sweden
    .xml.gz")


# In[69]:


# Construct the finite element space
P1 = FiniteElement("Lagrange",mesh.ufl_cell(),1)
TH = P1 * P1 * P1  # Taylor Hood Element
V = FunctionSpace(mesh,TH) # Creates space for u, v, w


# In[70]:


#Define parameters:
# Define parameters:
T = 1200
dt = 0.5
delta1 = Constant(1.0)
delta2 = Constant(1.0)
delta3 = Constant(1.0)
alpha = Constant(0.4)
beta = Constant(1)
gamma = Constant(0.8)
zeta = Constant(2.0)
L_0 = Constant(0.4)
l = Constant(0.6)
m = Constant(0.12)
r=random.uniform(0,1)


# In[71]:


# Class representing the initial conditions
class InitialConditions(UserExpression):
    def eval(self,values,x):
        if x[1]<100:
            values[0]=5/1000*r
            values[1]=1/2*(1 - r)
            values[2]=1/4 + 1/2*r
        else :
            values[0]=1/100
            values[1]=1/100
            values[2]=1/100





    def value_shape(self):
        return (3,)


# In[72]:


# Define Test and Trial Functions

#Trial-Initial Conditions
indata = InitialConditions(degree=2)
trial_0=TrialFunction(V)
trial_0=interpolate(indata, V)

#Test
k=TestFunction(V)

#Trial
trial=TrialFunction(V)

#For 3-PDEs System
```

```

#Trial_Initial
uz=trial_0[0]
vz=trial_0[1]
wz=trial_0[2]



#Test
k1=k[0]
k2=k[1]
k3=k[2]

#Trial
u=trial[0]
v=trial[1]
w=trial[2]





# In[73]:


# #Plot solution at t=0

# Mutualists (t=0)
# plot(trial_0[0])
# plt.title("mutualists,t="+str(0))
# plt.savefig("u"+str(0)+".png")
# plt.show()
# plt.clf()

# # Preys (t=0)
# plot(trial_0[1])
# plt.title("preys,t="+str(0))
# plt.savefig("v"+str(0)+".png")
# plt.show()
# plt.clf()

# # Predators (t=0)
# plot(trial_0[2])
# plt.title("predators,t="+str(0))
# plt.savefig("w"+str(0)+".png")
# plt.show()
# plt.clf()


# In[74]:


#Define Variational Problem - System of PDEs
F=(((u-uz)/dt)*k1)*dx \
  +(((v-vz)/dt)*k2)*dx \
  +(((w-wz)/dt)*k3)*dx \
  +(delta1*inner(grad(u+uz),grad(k1)))*0.5*dx \
  +(delta2*inner(grad(v+vz),grad(k2)))*0.5*dx \
  +(delta3*inner(grad(w+wz),grad(k3)))*0.5*dx \
  -(alpha*((u+uz)*0.5)*k1)*dx \
  -(beta*((v+vz)*0.5)*k2)*dx \
  +(gamma*((w+wz)*0.5)*k3)*dx \
  +(alpha*uz**2)/(L_0+l*vz)*k1*dx\
  +beta*(vz**2)*k2*dx \
  +vz*wz/(alpha + vz + m*uz)*k2*dx \
  -(zeta*(wz*vz)/(alpha+vz+m*uz))*k3*dx



# In[75]:


a= lhs(F)
L=rhs(F)
trial=Function(V)


# In[76]:


upop=[assemble(uz*dx)]
vpop=[assemble(vz*dx)]
wpop=[assemble(wz*dx)]

time=[]





# In[77]:

```

```
t=0
while t < T:
    time.append(t)
    t=t+1

    solve(a==L,trial)
    trial_0.assign(trial)

    utemp=assemble(trial_0[0]*dx)
    vtemp=assemble(trial_0[1]*dx)
    wtemp=assemble(trial_0[2]*dx)

    upop.append(utemp)
    vpop.append(vtemp)
    wpop.append(wtemp)

     # Plot

    # if int(t)%100==0:

        #Mutualists
        # plot(trial_0[0])
        # plt.title("mutualists,t="+str(t))
        # plt.savefig("u"+str(t)+".png")
        # plt.show()
        # plt.clf()

        # #Preys

        # plot(trial_0[1])
        # plt.title("preys,t="+str(t))
        # plt.savefig("v"+str(t)+".png")
        # plt.show()
        # plt.clf()

    #     #Predators

    #     plot(trial_0[2])
    #     plt.title("preys,t="+str(t))
    #     plt.savefig("w"+str(t)+".png")
    #     plt.show()
    #     plt.clf()

























# In[80]:


# vpop=vpop[:-1]
# wpop=wpop[:-1]
# upop=upop[:-1]

plt.plot(time,upop,label="mutualists")
plt.plot(time,vpop, label="preys")
plt.plot(time,wpop, label="predators")
plt.legend()
plt.show
plt.savefig("u,v"+str(t)+".png")
plt.clf


# #Phase portrait (preys-predators)
# plt.plot(vpop,wpop, label="preys-predators")
# plt.legend()
# plt.show
# plt.savefig("v,w"+".png")
# plt.clf

```

```
# Phase portrait (mutualists-preys-predators)
plt.plot(upop,vpop,wpop)
plt.title("mutualists-preys-predators")
plt.legend()
plt.grid()
plt.show
plt.savefig("u,v,w"+".png")
plt.clf




# In[79]:


# Mutualists (t=1000)
# plot(trial_0[0])
# plt.title("mutualists,t="+str(1000))
# plt.savefig("u"+str(t)+".png")
# plt.show()
# plt.clf()

# # Preys (t=1000)
# plot(trial_0[1])
# plt.title("preys,t="+str(1000))
# plt.savefig("v"+str(t)+".png")
# plt.show()
# plt.clf()


# In[ ]:
```

Hence, the effect of every method was examined, by measuring the time needed for the program to compute and update the variables of motion of every particle, after its interaction with the rest *N-1* particles. Specifically, we used the following function :

```c
#include <stdlib.h>
#include <math.h>
#include <unistd.h>
#include <sys/time.h>


typedef struct particle{
    double x;
    double y;
    double m;
    double ux;
```