Uppsala university, Division of Scientific computing

Project – Mutualism dynamics

Applied Finite Element Methods 1TD056 (5.0 hp)

Murtazo Nazarov

November 3, 2022

Introduction

This project aims to realize and explore one- and two-dimensional prey-predator-mutualist dynamics. Mutualism is the interaction of two or more species, in which each of them receives a benefit. In mutualism, the existence of one species does not adversely affect the others. An example of this process is oxpeckers on large mammals such as zebras, see Figure 1. Oxpeckers cling to large mammals and eat parasites, ticks, and blood-sucking flies.

Figure 1: Oxpeckers and zebras *Image source:* Natural History Museum.

Mathematical models of mutualism have attracted active interest in recent decades and play an important role in understanding the interaction between species. A good initial mathematical model for this process is the well-known Lotka-Volterra model. The Lotka-Volterra model is a system of ordinary differential equations and was originally proposed by Alfred J. Lotka (1880–1949) in 1910. Later, the same set of equations was proposed and analyzed by Vito Volterra (1860–1940).

In this project, we are interested in the following dynamics of prey-predator-mutualism, which is modeled by partial differential equations. To this end, we want to solve the following system of so-called *reaction-diffusion* partial differential equations

$$\begin{aligned}
\partial_t u - \alpha u\Big(1 - \frac{u}{L_0 + lv}\Big) - \delta_1 \Delta u &= f, \quad (\boldsymbol{x}, t) \in \Omega \times (0, T], && (1)\\
\partial_t v - \beta v(1 - v) + \frac{vw}{\alpha + v + mu} - \delta_2 \Delta v &= g, \quad (\boldsymbol{x}, t) \in \Omega \times (0, T], && (2)\\
\partial_t w + \gamma w - \zeta \frac{vw}{\alpha + v + mu} - \delta_3 \Delta w &= q, \quad (\boldsymbol{x}, t) \in \Omega \times (0, T], && (3)\\
u(\boldsymbol{x}, 0) &= u_0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, && (4)\\
v(\boldsymbol{x}, 0) &= v_0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, && (5)\\
w(\boldsymbol{x}, 0) &= w_0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, && (6)
\end{aligned}$$

where $\Omega \in \mathbb{R}^d$, $d = 1, 2, 3$, is a bounded domain, $T > 0$ is a finite time, $f(\boldsymbol{x}, t)$, $g(\boldsymbol{x}, t)$ and $p(\boldsymbol{x}, t)$ are given initial terms, and $u_0(\boldsymbol{x})$, $v_0(\boldsymbol{x})$ and $w_0(\boldsymbol{x})$ are initial data. Boundary conditions are to be determined. Also, $u(\boldsymbol{x}, t)$ denotes the population density of mutualists (such as oxpeckers), and $v(\boldsymbol{x}, t)$ denotes the population density of prays (such as zebras), and $w(\boldsymbol{x}, t)$ denotes the population density of predators (for example, lions). The terms of the mutualistic equation (1) denote change over time, mutualist birth and death rates, and population diffusion. Similarly, the terms of the prey equation (2) denote change over time, prey birth and death rates, prey consumption rate per predator as a fraction of maximum consumption rate 1, and population diffusion. The terms in the predator equations (3) refer to change over time, predator mortality rate, prey consumption rate per predator, and population diffusion. In the above system, all parameters $\alpha, \beta, \gamma, \zeta, \delta_1, \delta_2, \delta_3$ and L_0, l, m are positive. The parameters l and m denote the constants of mutualism. It can be seen that for $l = m = 0$ we get the usual prey-predator model.

The system (1)–(3) is a model that describes the dynamics and relationship of mutualist and prey to predator. If the number of predators decreases, then the model shows that the prey population increases, and it results in an increase of mutualists. On the other hand, when the prey population increases, it helps the predators to reproduce more and increase their population. As the predator population increases, they consume more prey, so the prey population decreases, which also results in a decrease in the mutualist population. As the prey population decreases, the predator begins to die due to a lack of food and consumption.

We can monitor the population change by calculating the overall population rate in the computational domain. This can be done by calculating the following integrals:

$$M_{\text{mutualist}}(t) = \int_\Omega u(\boldsymbol{x}, t)\,\mathrm{d}\boldsymbol{x}, \quad M_{\text{prey}}(t) = \int_\Omega v(\boldsymbol{x}, t)\,\mathrm{d}\boldsymbol{x}, \quad M_{\text{predator}}(t) = \int_\Omega w(\boldsymbol{x}, t)\,\mathrm{d}\boldsymbol{x}, \quad (7)$$

Below we break the problem down into three parts: Part A discusses a one-dimensional stationary diffusion problem; Part B discusses a two-dimensional scalar and system of time-dependent reaction-diffusion equations; Part C aims to solve the above system using the FEniCS project.

Part A

1D simplification

As a first step towards solving the full problem presented in this project, we consider a simpler 1D model,

$$-\delta u''(x) = f(x), \quad x \in (-1, 1), \tag{8}$$
$$u(-1) = u(1) = 0. \tag{9}$$

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

Problem A.1. Let u_h be a finite element approximation of the solution of the problem (8)–(9) on a mesh $-1 = x_0 < x_1 < \ldots < x_N = 1$. 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 $I = \bigcup_{i=1}^{n} I_i$ be the mesh. Derive the following a posteriori error estimate in the energy norm:

$$\|(u - u_h)'\|^2_{L^2(I)} \leq C \sum_{i=1}^{n} \eta_i^2, \tag{10}$$

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

Problem A.2. A typical algorithm for adaptive finite element approximation can be written as follows:

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

Let $\delta = 0.1$ and a function $g(x)$ be

$$g(x) := 10x \sin(7\pi x).$$

Then, we define the right hand side of (8) as

$$f(x) := \begin{cases} |x|, & \text{if } g(x) > |x|, \\ -|x|, & \text{if } g(x) < -|x|, \\ g(x), & \text{otherwise,} \end{cases}$$

First, approximate u_h'' using the discrete Laplacian $\Delta_h u_h$ defined in Lab 1. Then numerically integrate, say, the Trapezoid rule to calculate the error indicator η_i^2. Finally, implement an adaptive 1D finite element solver that solves the problem (8) - (9) using the adaptive algorithm described above. Start with a uniform grid with 12 nodes and stop refinement when the number of nodes exceeds 10^4 or TOL $< 10^{-3}$. 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 on the same figure in different subplots. Grid size distribution can be done by plotting `plot(x(2:end),[1./diff(x)])` where `x` are the coordinates of the ending grid.

Let's collect the number of nodes at each iteration in the vector $\boldsymbol{N}$ and the sum of the error indicators in the vector $\boldsymbol{\eta}$. Plot $\boldsymbol{N}$ vs. $\boldsymbol{\eta}$ and $\boldsymbol{N}$ vs. $\boldsymbol{N}^{-1}$ on the same figure using a loglog plot. What is the convergence rate of the adaptive iteration you are observing?

Hint: In Matlab you can implement the error indicator as follows: let's say you define a vector `eta2` to store η_i^2:

```
eta2 = zeros(N,1);
for i = 1:N
  h = x(i+1)-x(i);
  eta2(i) = ''put your numerical integration here'';
end
```

Here `x` is a vector of node coordinates and `N` is the number of elements.

There are different possibilities for selecting the elements to be refined given the element error indicator η_i. For instance, a popular method is to refine element i if

$$\eta_i > \lambda \max_{i=1,2,\ldots,N} \eta_i, \tag{11}$$

where $0 \leq \lambda \leq 1$ is a parameter which must be chosen by the user. Note that $\lambda = 0$ gives a uniform refinement, while $\lambda = 1$ gives no refinement at all.

The refinement procedure consists of the insertion of new nodes at the center of the elements chosen for refinement. In other words, if we are refining $[x_i, x_{i+1}]$, then we replace it by $[x_i, (x_i + x_{i+1})/2] \cup [(x_i + x_{i+1})/2, x_{i+1}]$. This is easily implemented by adding the coordinate of the midpoint of all the elements to be refined to the vector of nodes `x`, and then to sort it. We list the code:

```
lambda = 0.9;
for i = 1:length(eta2)
  if eta2(i) > lambda*max(eta2)
    x = [x (x(i+1)+x(i))/2];
  end
end
x = sort(x);
```

The stopping criterion determines when the adaptive algorithm should stop. For example, it may take the form of a maximum bound on the number of nodes, memory usage, computation time, total residual size, or a combination of these.

Problem A.3. Since u_h is a continuous piecewise linear function, its second derivative vanishes on every interval I_i, $i = 1, 2, \ldots, N$. Therefore, some people omit the discrete Laplacian from the (10) error estimate, i.e. the error indicator becomes $\eta_i = h_i \|f\|$. Run your code from the previous task using this simplified error indicator and create the appropriate plots. What would you say to these people: can they omit the discrete Laplacian or not?

Part B

2D convergence analysis

Let's look at the scalar version of the prey-predator model (1)–(3) in two spatial dimensions. The purpose of this part is to study Matlab's *implementation* of piecewise linear finite element approximation in 2D space and to study *convergence* of the finite element method.

The computational domain, *i.e.* disk Ω is a circle of radius 1, and it can be triangulated in Matlab as follows: define the geometry using `circleg`, a Matlab built-in function, and triangulate this with respect to the maximum mesh-size $h_{\max}$:

```
geometry = @circleg;
hmax = 1/5;
[p,e,t] = initmesh(geometry,'hmax',hmax);
```

where `[p,e,t]` are point, edge and element data given by the triangulation in Matlab.

Provided that you have written functions to assemble the stiffness matrix and load vector, the homogeneous Dirichlet boundary condition is applied *strongly* (*i.e.* after assembling the linear system $\boldsymbol{A\xi} = \boldsymbol{b}$) as follows:

```
I = eye(length(p));          % construct the identity matrix
A = AssembleA(...);          % stiffness matrix
b = Assembleb(...);          % load vector
A(e(1,:),:) = I(e(1,:),:); % replace the rows corresponding
                             % to the boundary nodes by
                             % corresponding rows of I
b(e(1,:)) = 0;               % put the boundary value into the RHS
```

Furthermore, the energy norm of the error

$$\|u - u_h\|_E = \Big(\int_{\mathcal{B}} (\nabla u - \nabla u_h) \cdot (\nabla u - \nabla u_h) \, \mathrm{d}\boldsymbol{x} \Big)^{\frac{1}{2}}$$

can be computed in Matlab as `EnE=sqrt(err'*A*err);`, where `A` is the stiffness matrix and `err=u-u_h` is the error.

Problem B.1. Let us consider a simplified the mutualist population equation, which is obtained by omitting the time-derivative and nonlinear terms:

$$-\Delta u(\boldsymbol{x}) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in \Omega, \tag{12}$$

$$u(\boldsymbol{x}) = u_{\text{exact}}(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega. \tag{13}$$

The right hand side is set as $f(\boldsymbol{x}) = 8\pi^2 \sin(2\pi \boldsymbol{x}_1) \sin(2\pi \boldsymbol{x}_2)$, which gives the exact solution $u_{\text{exact}}(\boldsymbol{x}) = \sin(2\pi \boldsymbol{x}_1) \sin(2\pi \boldsymbol{x}_2)$.

Write a Galerkin finite element method for this problem using continuous piecewise linear basis functions in space, and then implement it in Matlab. Calculate the energy rate in the sequence of meshes obtained by fitting: $h_{\max} = \left\{ \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32} \right\}$. Then find the *rate of convergence* of the Galerkin approximation with respect to the energy norm numerically and denote it by p. Plot $h_{\max}$ vs. the energy norm of error and $h_{\max}$ vs. $h_{\max}^p$ on the same figure using `loglog`- plot in Matlab. Explain how you determined the value of p and state its value.

Plot the solutions that are obtained using the coarsest and the finest meshes.

Problem B.2. Now, let us consider the prey-predator model with constant predator population density $w(\boldsymbol{x}, t) \equiv 1$ and no mutualist $u(\boldsymbol{x}, t) \equiv 0$. Then the system (1)–(3) reduces to the following scalar equation:

$$\partial_t v - v(1-v) + \frac{v}{v+\alpha} - \delta_1 \Delta u = g, \quad (\boldsymbol{x}, t) \in \Omega \times (0, T]. \tag{14}$$

Let us assume that the source term is zero, *i.e.* $g(\boldsymbol{x}, t) \equiv 0$. In addition, take $\delta_1 = 0.01$, $\alpha = 4$. Assume the boundary condition is homogenous Neumann everywhere, *i.e.* $\partial_n v(\boldsymbol{x}, t) = 0$ for all $\boldsymbol{x} \in \partial\Omega$ and $t \in (0, T]$, and initial condition is defined as

$$v(\boldsymbol{x}, 0) = 1 + 20\,\omega(\boldsymbol{x}),$$

where $\omega(\boldsymbol{x}) \in [0, 1]$ is a randomly generated number for every point $\boldsymbol{x} \in \Omega$.

Task 1. Formulate the Galerkin finite element method using a continuous piecewise linear approximation of (14). Discretize the time derivative using the Crank-Nicholson method and write down the corresponding linear algebra matrices and vectors.

Task 2. Now implement the variational problem obtained in Task 1. in Matlab. Implement a numerical integration in 2D to calculate the population rate (7). For example, to calculate the integral of a general function $F(\boldsymbol{x})$, you can use the Trapezoid rule in an element K:

$$\int_K F(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \frac{|K|}{3} \sum_{i=1}^{3} F(N_i),$$

where N_i, $i = 1, 2, 3$, is the nodal points of the cell K.

Run your code until $T = 2$ with $h_{\max} = \frac{1}{5}$, $h_{\max} = \frac{1}{20}$, and $h_{\max} = \frac{1}{40}$. Plot the population rates as function of time on one figure. Also, plot the corresponding solution at $T = 2$ for each mesh resolution.

Hint: Let us denote the nonlinear terms of (14) by $S(u)$. In your implementation, you can treat this term explicitly, *i.e.* compute $S(u)$ using the solution from previous time step. Also, it is useful to take a linear interpolant of $S(u)$ in the implementation. That is $S(u) \approx \pi_h S = \sum_{j=1}^{\mathcal{N}} S_j \varphi_j$, where $\mathcal{N}$ is the total number of nodes.

Part C

FEniCS implementation

This part of the project is considered to be the final and main part, where we consider the prey-predator model (1)–(3) with zero source terms: $f(\boldsymbol{x}, t) = g(\boldsymbol{x}, t) = q(\boldsymbol{x}, t) = 0$. We take the parameters as follows: $\alpha = 0.4$, $\beta = 0.8$, $\gamma = 0.8$, $\zeta = 2$, $L_0 = 0.4$, $l = 0.6$, $m = 0.12$, The boundary condition is homogeneous Neumann condition in all boundary points, *i.e.* $\partial_n u(\boldsymbol{x}, t) = \partial_n v(\boldsymbol{x}, t) = \partial_n w(\boldsymbol{x}, t) = 0$ for all $\boldsymbol{x} \in \partial\Omega$ and $t \in (0, T]$.

We will solve the problem using the open-source finite element code FEniCS. The FEniCS project is a collection of open-source finite element software for solving partial differential equations in any space dimensions and polynomial degrees. Laboratory 3 can be a good starting point for implementing a time-dependent heat equation in FEniCS. Below are some useful tools of FEniCS that could be useful in the implementation. The structure of your solver could look like the following:

```
from dolfin import *

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

# Construct the finite element space
P1 = FiniteElement("Lagrange", mesh.ufl_cell(), 1)
TH = P1 * P1 * P1
W = FunctionSpace(mesh, TH)

# Define parameters:
T      = ...
dt     = ...
delta1 = ...
delta2 = ...
delta3 = ...
alpha  = ...
gamma  = ...
zeta   = ...
L_0    = ...
l      = ...
m      = ...

# Class representing the intial conditions
class InitialConditions(UserExpression):
    def eval(self, values, x):
        values[0] = ...
        values[1] = ...
        values[2] = ...

    def value_shape(self):
        return (3,)

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

# Create bilinear and linear forms
a = ...
L = ...

# Set an output file
...

# Set initial condition
...
```

```
# Time-stepping
while t < T:

    # assign u0
    u0.assign(u)
    ...
    ...
```

Write down the finite element discretization of the system (1)–(3) using the Crank-Nicholson scheme in time. The nonlinear terms can be computed from the previous time-step.

Problem C.1. Implement a FEnICS solver the solve the problem (1)–(3) using the following initial condition:

$$\begin{aligned} u_0 &= 0, \\ v_0 &= \frac{4}{15} - 2 \cdot 10^{-7}(x_1 - 0.1x_2 - 350)(x_1 - 0.1x_2 - 67), \\ w_0 &= \frac{22}{45} - 3 \cdot 10^{-5}(x_1 - 450) - 1.2 \cdot 10^{-4}(x_2 - 15). \end{aligned}$$

Run your code with $\delta_1 = 1$, $\delta_2 = 1$, $\delta_3 = 1$, and $\Delta t = 0.5$ until $T = 500$ using the mesh `circle.xml`. Plot your solutions u and v at times $T = 0, 100, 200, 300, 400$. Now, run your code until $T = 1000$ and plot your solution at the final time. What do you observe now? Plot the population rates for prey and predator at the same figure and discuss why they behave so. It is also useful to plot the so-called phase portrait of the solutions: `plot(popolation_v, population_w)`. The phase portrait shows the dynamic of the populations, i.e if there is an equilibrium or one species dies out.

Now, set

$$u_0 = 0.1v_0,$$

and run your code while $T = 1000$. What are you observing? Plot all the population rates on one figure. Use `plot3(population_u, population_v, population_w)` to build a phase portrait and discuss the results.

Problem C.2. We are interested in modeling the dynamics of prey-predator-mutualist in Sweden. Suppose below Sundsvall, the initial population is randomly defined as

$$\begin{aligned} u_0(\boldsymbol{x}_i) &= \frac{5}{1000}\text{random}(\boldsymbol{x}_i), \\ v_0(\boldsymbol{x}_i) &= \frac{1}{2}(1 - \text{random}(\boldsymbol{x}_i)), \\ w_0(\boldsymbol{x}_i) &= \frac{1}{4} + \frac{1}{2}\text{random}(\boldsymbol{x}_i), \end{aligned}$$

and above Sundsvall the initial data is

$$u_0(\boldsymbol{x}_i) = v_0(\boldsymbol{x}_i) = w_0(\boldsymbol{x}_i) = \frac{1}{100}.$$

for every nodal points $\boldsymbol{x}_i \in \Omega$. Here $\text{random}(\boldsymbol{x}_i) : \mathbb{R}^2 \mapsto [0, 1]$ is a random numbers between zero and one. In Python one can do:

```
import random
...
r =  random.uniform(0, 1)
...
```

Run your code with $\Delta t = 0.5$ until $T = 1200$ using the mesh `sweden.xml.gz`. Plot your solution u and v at different time instances: $T = 0, 50, 100, 600, 1200$. Plot also the population rates for u, v and w. Plot also the phase portrait of the populations rates. Discuss your result.

Hint: The integrals (7) or in general any functionals in FEniCS can be implemented as follows:

```
# construct a vector function
u = Function(W)

# Define the integrals
M0 = u[0] * dx
M1 = u[1] * dx
M2 = u[2] * dx

# compute the functional
population_u = assemble(M0)
population_v = assemble(M1)
population_w = assemble(M2)
```

Now you just need to call this assembly inside your time loop and save it to a file. If your computer is fast, you can solve the problem with a finer meshes `circle_fine.xml.gz` and `sweden_fine.xml.gz`. Your figures will be clearer and you will see more detail.

Good luck!
Murtazo
Uppsala, November 2022