UPPSALA
UNIVERSITET

Applied Finite Element Methods 1TD056

Project - Mutualism Dynamics

Author:

Joakim Svensson,

joakim.svensson7535@gmail.com

Uppsala

15th November 2022

Contents

1 Introduction

The aim of this project was to get acquainted in one- and two-dimensional PDE's describing prey-predator-mutualist dynamics using finite element methods. The following set of partial difference equations for reaction-diffusion served as inspiration for the project: [1]

$$\begin{cases} \partial_t u + \alpha u(1 - \frac{u}{L_0+lv}) - \delta_1 \Delta u = f, & (\mathbf{x}, t) \in \Omega \times (0, T], \\ \partial_t v - \beta v(1-v) + \frac{vw}{\alpha+v+mu} - \delta_2 \Delta v = g, & (\mathbf{x}, t) \in \Omega \times (0, T], \\ \partial_t w + \gamma w - \zeta \frac{vw}{\alpha+v+mu} - \delta_3 \Delta w = q, & (\mathbf{x}, t) \in \Omega \times (0, T], \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}), & \mathbf{x} \in \Omega, \\ v(\mathbf{x}, 0) = v_0(\mathbf{x}), & \mathbf{x} \in \Omega, \\ w(\mathbf{x}, 0) = w_0(\mathbf{x}), & \mathbf{x} \in \Omega. \end{cases} \tag{1}$$

where $\Omega \in \mathbb{R}^d$, $d = 1, 2, 3$ is a bounded domain, $f(\mathbf{x}, t)$, $g(\mathbf{x}, t)$ and $q(\mathbf{x}, t)$ are initial terms, $u_0(\mathbf{x})$, $v_0(\mathbf{x})$ and $w_0(\mathbf{x})$ are initial data, $T > 0$ is a finite time domain and $\alpha, \beta, \gamma, \zeta, \delta_1, \delta_2, \delta_3, L_0, l, m$ are positive parameters. The system 1 describes the prey-predator-mutualist dynamics for a certain population of animals in a eco-system.[1]

The project was divided in three parts; part A which was focused simplification for one dimensional problems, part B which was concentrated on convergence analysis of two dimensional problems and part C, which focused on FEniCS implementation. In this project, parts A and B were solved using Matlab, and part C was done using the open-source finite element program FEniCS.

2 Part A

By considering the one dimensional model:

$$\begin{cases} -\delta u''(x) = f(x), & x \in (-1, 1), \\ u(-1) = u(1) = 0. \end{cases} \tag{2}$$

for some forcing function $f(x)$, different simplifications can be made to apply a finite element solver with adaptive mesh refinement using a posteriori error estimation.

2.1 Problem A1

Assume that u_h is a finite element approximation of the model 2, on a mesh $-1 = x_0 < x_1 < ... < x_N = 1$ with step size h_i and the i-th element I_i. Hence, a posteriori error estimation can be derived in the energy norm according to:

$$||(u - u_h)'||^2_{L^2(I)} \leq C \sum_{i=1}^{n} \eta_i^2, \tag{3}$$

where $\eta_i^2 = h_i ||(f + \delta u_h'')||_{L^2(I_i)}$ and C is some constant. This could be proven by first letting the error be: $e = u - \delta u_h$. Therefore:

$$||e'||^2_{L^2(I)} = \int_I e'^2 dx = \tag{4}$$

$$= \int_I e'(e - \pi e)' dx = \tag{5}$$

$$= \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} e'(e - \pi e)' dx. \tag{6}$$

By using integration of parts equation 6 can be simplified to:

$$\sum_{i=1}^{n}\int_{x_{i-1}}^{x_i}(e'')(e-\pi e)dx. \tag{7}$$

Since the Galerkin orthogonality can reduce πe to e it is obvious that e and πe corresponds in the nodes and also e'' on I_i can now be simplified as:

$$-e'' = -(u-\delta u_h)'' = -u'' + \delta u_h'' = f + \delta u_h''. \tag{8}$$

Inserting equation 8 into 7 and using the Cauchy-Schwarz identity and a standard interpolation error estimation, it can be proven, since:

$$||e'||^2_{L^2(I)} = \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i}(f+\delta u_h'')(e-\pi e)dx \tag{9}$$

$$\leq \sum_{i=1}^{n}||f+\delta u_h''||_{L^2(I_i)}||(e-\pi e)||_{L^2(I_i)} \tag{10}$$

$$\leq \sum_{i=1}^{n}||f+\delta u_h''||_{L^2(I_i)}Ch_i||e'||_{L^2(I_i)} \tag{11}$$

$$= C\sum_{i=1}^{n}h_i||f+\delta u_h''||_{L^2(I_i)}||e'||_{L^2(I_i)} \tag{12}$$

$$\leq C\left(\sum_{i=1}^{n}h_i||f+\delta u_h''||^2_{L^2(I_i)}\right)^{\frac{1}{2}}\left(\sum_{i=1}^{n}h_i||e'||^2_{L^2(I_i)}\right)^{\frac{1}{2}} \tag{13}$$

$$= C\left(\sum_{i=1}^{n}h_i^2||f+\delta u_h''||^2_{L^2(I_i)}\right)^{\frac{1}{2}}||e'||_{L^2(I)} \tag{14}$$

Dividing both sides by $||e'||_{L^2(I)}$ leads to equation 3 where $\eta_i = h_i||(f+\delta u_h'')||_{L^2(I_i)}$, hence it can be proven and therefore it is derived.

2.2 Problem A2

In problem A2 the task was to solve equation 2 using the finite element method with adaptive mesh refinement. $\delta = 0.1$ ws given and the weight function $f(x)$ could be defined as:

$$f(x) := \begin{cases} |x|, & g(x) > |x|, \\ -|x|, & g(x) < |x|, \\ g(x), & otherwise, \end{cases} \tag{15}$$

where $g(x) = 10x\sin(7\pi x)$. The stopping criteria for the adaptive finite element approximation was either $\sum_{i=1}^{n}\eta_i^2 < 0.001$ or $N > 10000$. With all of this in mind the problem could be solved. The solution was made on Matlab and the following plots for 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 were given:

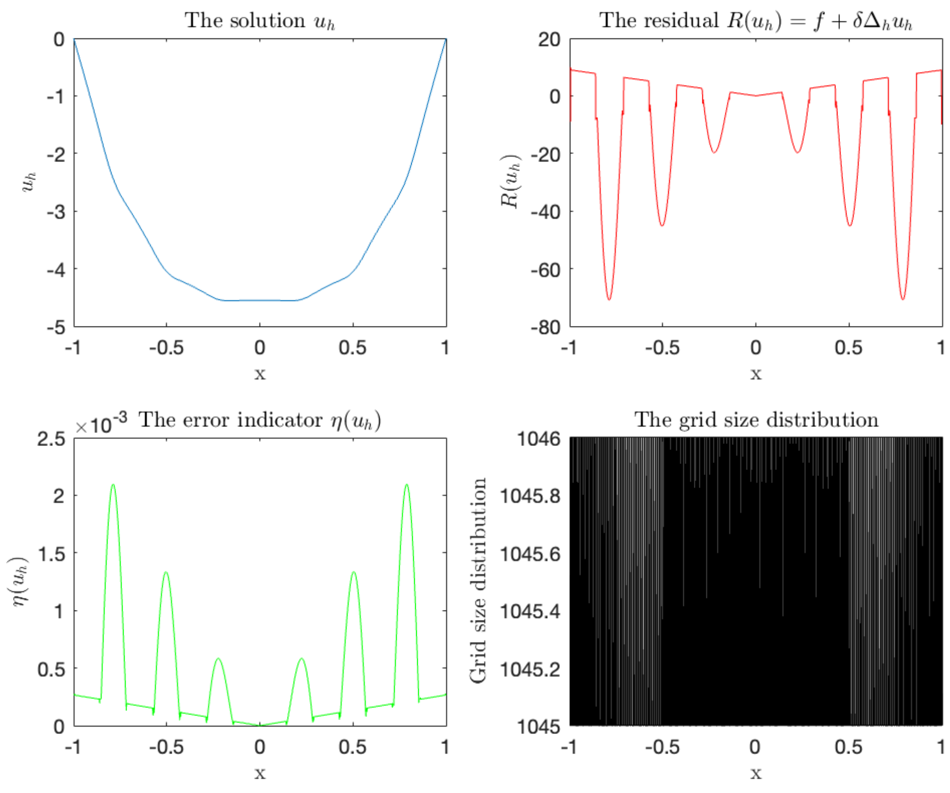

Figure 1: Four different subplots from the adaptive finite element approximation.

For the convergence of the problem the following plot could be made:

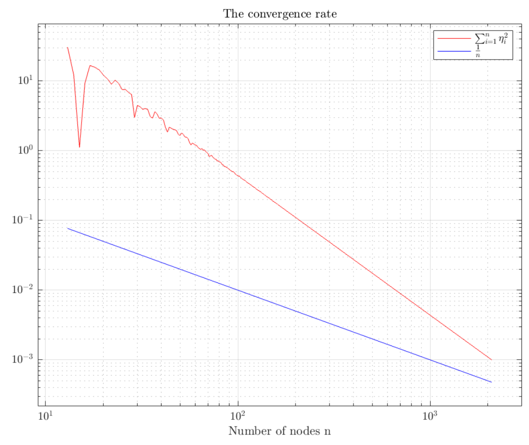

Figure 2: The convergency of the problem A2.

The criterion for the tolerance $TOL = 0.001$ was achieved when $N = 2091$. In figure 2 a comparison between the convergence rate for the used method and $\frac{1}{n}$ can be made. The convergence rate for the used method is almost as fast as $\frac{1}{n}$ for large n, where n is the number of nodes for the final grid. Hence, the convergence rate for the adaptive finite element method is $q = 1$.

2.3 Problem A3

In problem A3, the task was to investigate exactly the same equation as in A2, with the only difference that instead of defining $\eta_i = h_i||(f + \delta u_h''))||_{L^2(I_i)}$, make the assumption that he second derivative of u_h disappears on every interval I_i because u_h is a continuous piecewise linear function. The error indicator changes as a result when the discrete Laplacian is removed from the error estimate, hence η_i is instead defined as $\eta_i = h_i||(f)||_{L^2(I_i)}$.

The stopping criteria for the adaptive finite element approximation was either $\sum_{i=1}^{n} \eta_i^2 < 0.001$ or $N > 10000$ The solution was made on Matlab and the following plots for 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 were given:

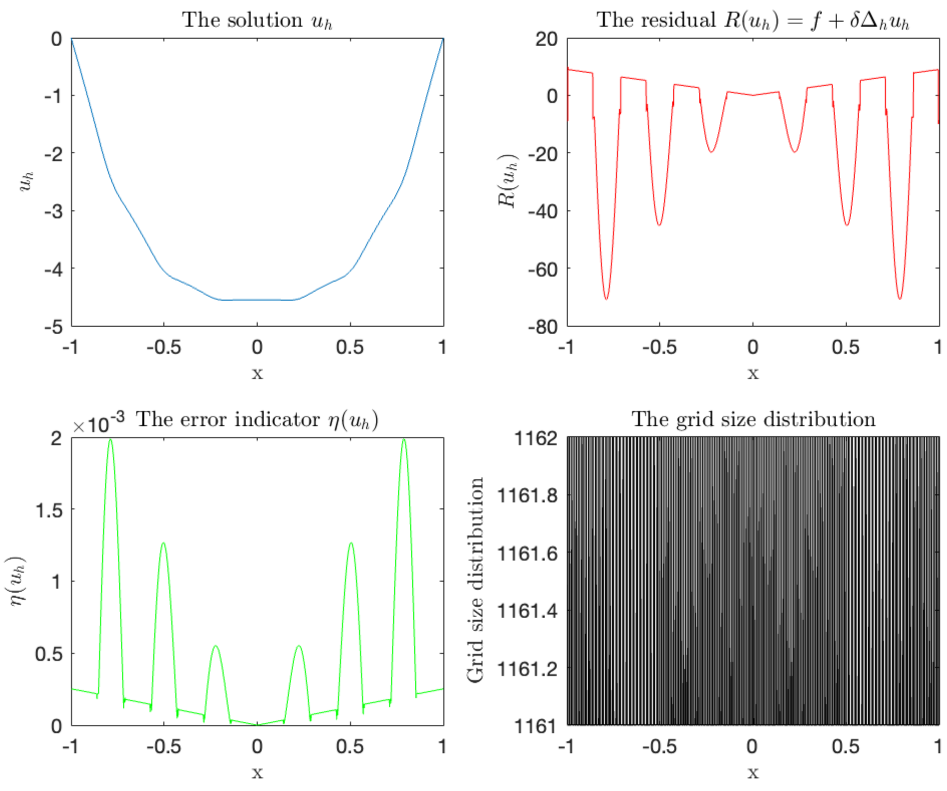

Figure 3: Four different subplots from the adaptive finite element approximation in A3.

For the convergency of the problem the following plot could be made:

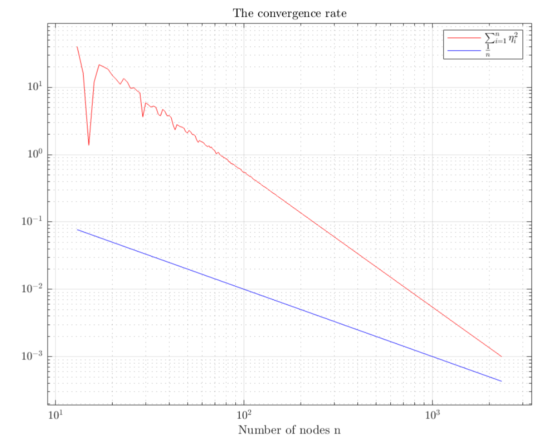

Figure 4: The convergency of the problem A3.

The criterion for the tolerance $TOL = 0.001$ was achieved when $N = 2323$. The differences between A2 and A3 are minor. It takes approximately 200 iterations more for the tolerance to be achieved, hence the method in A3 has a slower convergence rate, compared with the method in A2. This becomes obvious when watching the grid size distributions in figure 1 and 3. The method in A2 has a better distribution over the grid, since the error indicator in A2 are taking the discrete laplacian into account. With that said, the method where the discrete laplacian is being omitted can still be used, even tough it offers more advantages to include the discrete laplacian in the a posteriori estimation.

References

[1] Murtazo Nazarov. *Project – Mutualism dynamics*. Uppsala University, Division of Scientific computing, Applied Finite Element Methods 1TD056. Accessed 2022-11-08. 2022.

Matlab Code

Part A

Listing 1: Problem A2

```matlab
clear all; close all; clc;
%% Problem A2

%% Initial data etc.
N = 12;
a = -1; % left end point of interval
b = 1; % right
TOL = 1e-3;
delta = 0.1;

%% The main loop
while N < 1e+4
    h = abs(b-a)/N;
    x = a:h:b; % node coords
    A=Stiffness_assembler(x);
    B=Load_vector(x);
    M=Mass_assembler(x);
    xi = A\B; % solve system of equation
    zeta = -inv(M)*A*xi; % the discrete laplacian
    eta = zeros(N ,1); % allocate element residuals
    eta2 = zeros(N,1); % allocate element residuals
    for i = 1:N % loop over elements
        h = x(i+1) - x(i); % element length
        if 10*x(i)*sin(7*pi*x(i)) > abs(x(i))
            f = @(x) abs(x)/delta;
        elseif 10*x*sin(7*pi*x(i)) < abs(x(i))
            f = @(x) -abs(x)/delta;
        else
            f = @(x) (10*x*sin(7*pi*x))/delta;
        end
        R(i) = f(x(i)) + delta*zeta(i);
        a2 = f(x(i)) + delta*zeta(i); % temporary variables
        b2 = f(x(i+1)) + delta*zeta(i+1);
        t = (a2^2+b2^2)*h/2; % integrate f^2. Trapezoidal rule
        eta(i) = h*sqrt(t);
        eta2(i) = h^2*t; % element residual
    end
    sum_eta2(N) = sum(eta2, 'all');
    n_nodes(N) = length(x);
    if sum_eta2(N) < TOL
        break
    end
    alpha = 0.9; % refinement parameter
    for i = length(eta2)
        if eta2(i) > alpha*max(eta2) % if large residual
            x = [x (x(i+1)+x(i))/2]; % insert new node point
        end
    end
    x = sort(x); % sort node points accendingly
    N = N+1;
end

%% PLOTS
% First figure
figure(1)
set(gcf,'color','w');
set(gca,'TickLabelInterpreter','latex')
grid on
axis padded

subplot(2,2,1)
plot(x,xi)
title('The solution $u_{h}$', 'Interpreter', 'latex')
xlabel('x','Interpreter','latex');ylabel('$u_{h}$','Interpreter','latex');

subplot(2,2,2)
plot(x(2:end), R, 'r')
title('The residual $R(u_{h}) = f + \delta \Delta_{h} u_{h}$', 'Interpreter', 'latex')
xlabel('x','Interpreter','latex');ylabel('$R(u_{h})$','Interpreter','latex');

subplot(2,2,3)
plot(x(2:end),eta, 'g')
title('The error indicator $ \eta (u_{h}) $', 'Interpreter','latex')
xlabel('x','Interpreter','latex');ylabel('$\eta (u_{h})$','Interpreter','latex');

subplot(2,2,4)
plot(x(2:end),round([1./diff(x)]), 'black')
title('The grid size distribution', 'Interpreter','latex')
xlabel('x','Interpreter','latex');ylabel('Grid size distribution','Interpreter','latex');

% Next figure
figure(2)
loglog(n_nodes, sum_eta2 , 'r')
hold on
loglog(n_nodes, n_nodes.^(-1), 'b')
title('The convergence rate', 'Interpreter','latex')
set(gcf,'color','w');
legend('$\sum_{i=1}^{n} \eta_{i}^{2}$', '$\frac{1}{n}$','Interpreter','latex','Location','
    best');
set(gca,'TickLabelInterpreter','latex')
xlabel('Number of nodes n','Interpreter','latex');
grid on
axis padded
```

Listing 2: Problem A3

```
clear all; close all; clc;
%% Problem A3

%% Initial data etc.
N = 12;
a = -1; % left end point of interval
b = 1; % right
TOL = 1e-3;
delta = 0.1;

%% The main loop
while N < 1e+4
    h = abs(b-a)/N;
    x = a:h:b; % node coords
    A=Stiffness_assembler(x);
    B=Load_vector(x);
    M=Mass_assembler(x);
    xi = A\B; % solve system of equation
    zeta = -inv(M)*A*xi; % the discrete laplacian
    eta = zeros(N ,1); % allocate element residuals
    eta2 = zeros(N,1); % allocate element residuals
    for i = 1:N % loop over elements
        h = x(i+1) - x(i); % element length
        if 10*x(i)*sin(7*pi*x(i)) > abs(x(i))
            f = @(x) abs(x)/delta;
        elseif 10*x*sin(7*pi*x(i)) < abs(x(i))
            f = @(x) -abs(x)/delta;
        else
            f = @(x) 10*x*sin(7*pi*x)/delta;
        end
        R(i) = f(x(i))+ delta*zeta(i);
        a2 = f(x(i)); % temporary variables
        b2 = f(x(i+1));
        t = (a2^2+b2^2)*h/2; % integrate f^2. Trapezoidal rule
        eta(i) = h*sqrt(t);
        eta2(i) = h^2*t; % element residual
    end
    sum_eta2(N) = sum(eta2, 'all');
    n_nodes(N) = length(x);
    if sum_eta2(N) < TOL
        break
    end
    alpha = 0.9; % refinement parameter
    for i = length(eta2)
        if eta2(i) > alpha*max(eta2) % if large residual
            x = [x (x(i+1)+x(i))/2]; % insert new node point
        end
    end
    x = sort(x); % sort node points accendingly
    N = N+1;
```

```
end

%% PLOTS
% First figure
figure(1)
set(gcf,'color','w');
set(gca,'TickLabelInterpreter','latex')
grid on
axis padded

subplot(2,2,1)
plot(x,xi)
title('The solution $u_{h}$', 'Interpreter', 'latex')
xlabel('x','Interpreter','latex');ylabel('$u_{h}$','Interpreter','latex');

subplot(2,2,2)
plot(x(2:end), R, 'r')
title('The residual $R(u_{h}) = f + \delta \Delta_{h} u_{h}$', 'Interpreter', 'latex')
xlabel('x','Interpreter','latex');ylabel('$R(u_{h})$','Interpreter','latex');

subplot(2,2,3)
plot(x(2:end),eta, 'g')
title('The error indicator $ \eta (u_{h}) $', 'Interpreter','latex')
xlabel('x','Interpreter','latex');ylabel('$\eta (u_{h})$','Interpreter','latex');

subplot(2,2,4)
plot(x(2:end), round([1./diff(x)]), 'black')
title('The grid size distribution', 'Interpreter','latex')
xlabel('x','Interpreter','latex');ylabel('Grid size distribution','Interpreter','latex');

% Next figure
figure(2)
loglog(n_nodes, sum_eta2 , 'r')
hold on
loglog(n_nodes, n_nodes.^(-1), 'b')
title('The convergence rate', 'Interpreter','latex')
set(gcf,'color','w');
legend('$\sum_{i=1}^{n} \eta_{i}^{2}$', '$\frac{1}{n}$','Interpreter','latex','Location','
    best');
set(gca,'TickLabelInterpreter','latex')
xlabel('Number of nodes n','Interpreter','latex');
grid on
axis padded
```

Part A Methods

Listing 3: Load vector assembler

```
    function b=Load_vector(x)
%
% Returns the assembled load vector b.
% Input is a vector x of node coords.
%
delta = 0.1;
N = length(x) - 1;
b = zeros(N+1, 1);
for i = 1:N
    if 10*x(i)*sin(7*pi*x(i)) > abs(x(i))
        f = @(x) abs(x)/delta;
    elseif 10*x*sin(7*pi*x(i)) < abs(x(i))
        f = @(x) -abs(x)/delta;
    else
        f = @(x) (10.*x.*sin(7*pi.*x))/delta;
    end
    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
```

Listing 4: Mass matrix assembler

```
function M = Mass_assembler(x)
    n = length(x) - 1;
    M = sparse(zeros(n+1,n+1));
    for i = 1:n
        h = x(i+1) - x(i);
        M(i,i) = M(i,i) + h/3;
        M(i,i+1) = M(i,i+1) + h/6;
        M(i+1,i) = M(i+1,i) + h/6;
        M(i+1,i+1) = M(i+1,i+1) + h/3;
    end
end
```

Listing 5: Stiffness matrix assembler

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