Finite Elements lab assignments

Martijn Papendrecht: 4369971

February 21, 2018

Contents

1 1D Assignment

1.1 Theory

Starting with the differential equation and boundary conditions

$$\begin{aligned} -D\frac{d^2u}{dx^2} + \lambda u = f(x), \\ -D\frac{du}{dx}(0) = 0, -D\frac{du}{dx}(1) = 0 \end{aligned} \tag{1}$$

we can derive the weak formulation.

$$\begin{aligned} \int_0^1 \left[-D\frac{d^2u}{dx^2} + \lambda u\right]\phi dx &= \int_0^1 f(x)\phi dx \\ \int_0^1 -D\left(\frac{d}{dx}\left(\frac{du}{dx}\phi\right) - \frac{d\phi}{dx}\frac{du}{dx}\right) + \lambda u\phi &= \int_0^1 f(x)\phi dx \\ -D\int_0^1 \frac{d}{dx}\left(\frac{du}{dx}\phi\right)dx + \int_0^1 D\frac{d\phi}{dx}\frac{du}{dx} + \lambda u\phi dx &= \int_0^1 f(x)\phi dx \\ \int_0^1 D\frac{d\phi}{dx}\frac{du}{dx} + \lambda u\phi dx &= \int_0^1 f(x)\phi dx \end{aligned} \tag{2}$$

Here ϕ is the test function satisfying the smoothness requirements. The boundary conditions are already contained in this weak formulation.

We can derive the Galerkin formulation by substitution $\phi = \phi_i$ and $u \approx u^N = \sum_{j=1}^N c_j\phi_j$ in eq. (2).

$$\begin{aligned} \int_0^1 D\frac{d\phi_i}{dx}\frac{d}{dx}\left(\sum_{j=1}^N c_j\phi_j\right) + \lambda\left(\sum_{j=1}^N c_j\phi_j\right)\phi_i dx &= \int_0^1 f(x)\phi_i dx \\ \sum_{j=1}^N c_j\int_0^1 D\frac{d\phi_i}{dx}\frac{d\phi_j}{dx} + \lambda\phi_i\phi_j dx &= \int_0^1 f(x)\phi_i dx \end{aligned} \tag{3}$$

And finally we can write in the form $S\vec{u} = \vec{f}$ where

$$S_{ij} = \int_0^1 D\frac{d\phi_i}{dx}\frac{d\phi_j}{dx} + \lambda\phi_i\phi_j dx \tag{4}$$

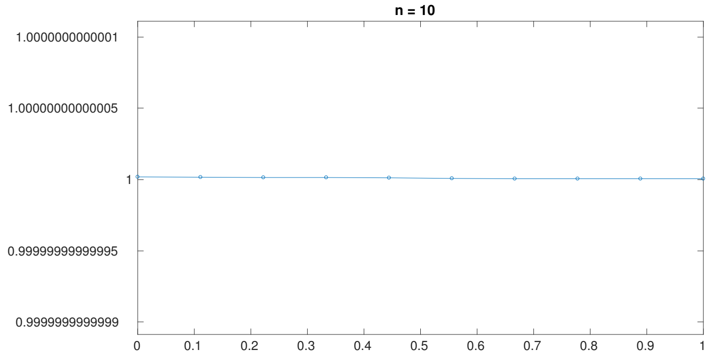
$$f_i = \int_0^1 f(x)\phi_i dx \tag{5}$$

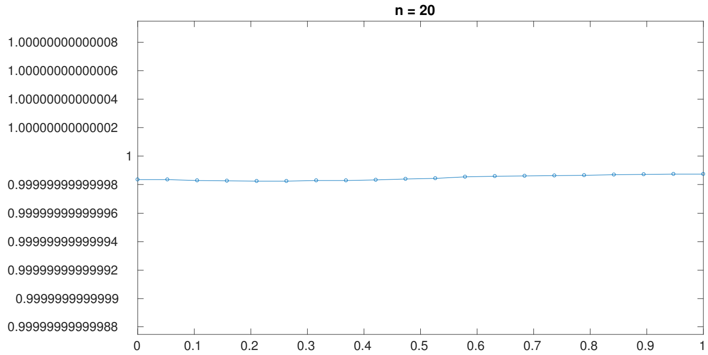
1.2 Modelling

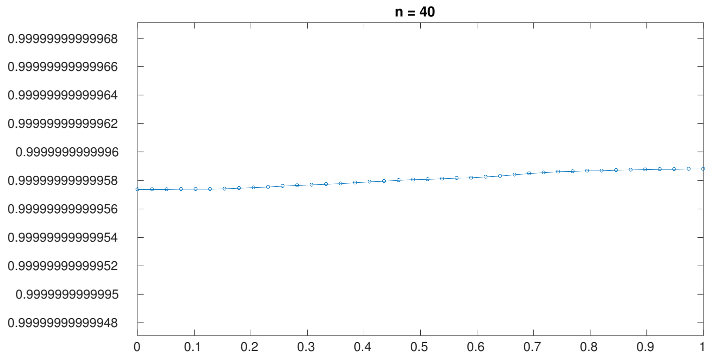
All figures are made using Matlab. The code is available at Github [1]. In these figures the elements aren't equally spaced as prescribed in the assignment. Every element has an additional random offset of $\pm\frac{1}{2n}$. This has been done to test the code without the assumption of equally spaced vertices.

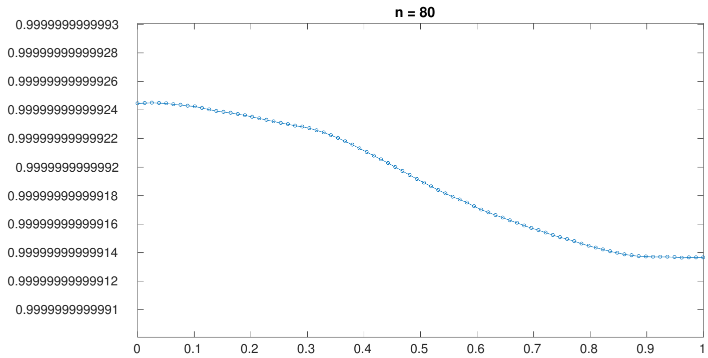
[1] `https://github.com/MPapendrecht/FiniteElements.git`

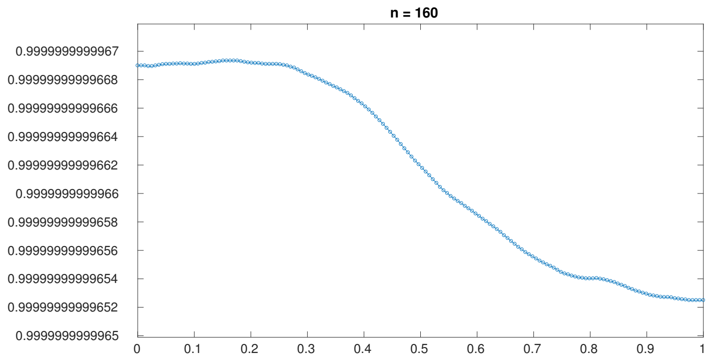
The figures below show the results for different number of vertices n. $\lambda = 1$ and $D = 1$ and the function $f(x) = 1$





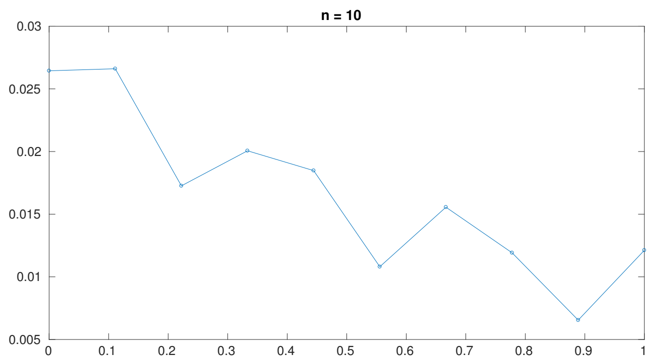


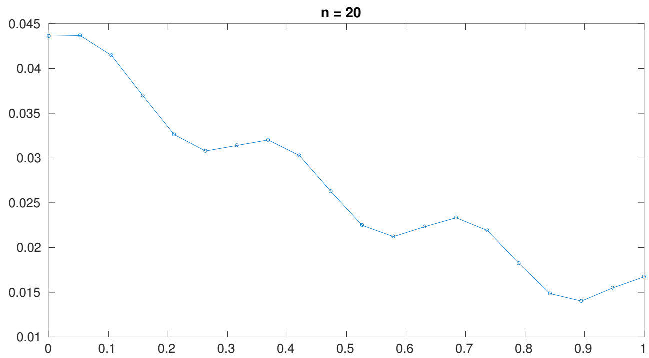


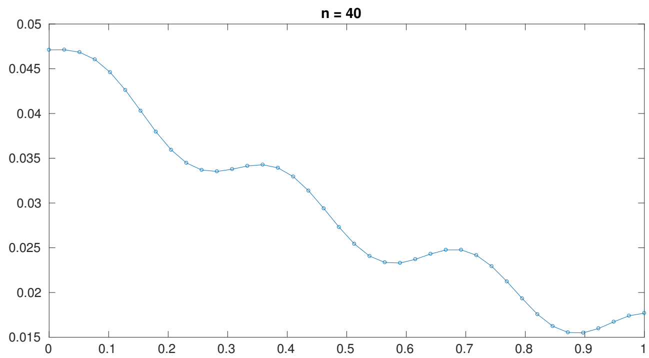


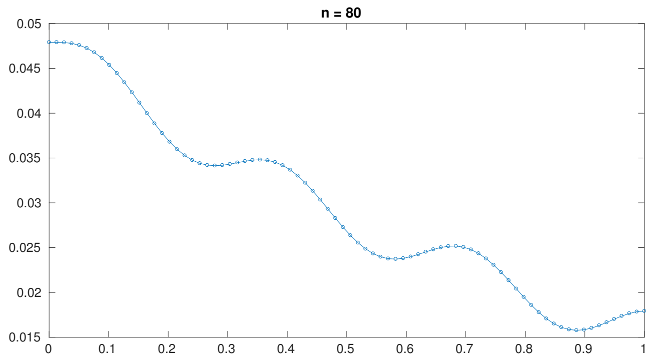
The solution is as expected very close to the analytical solution, and clearly satisfies the boundary conditions.

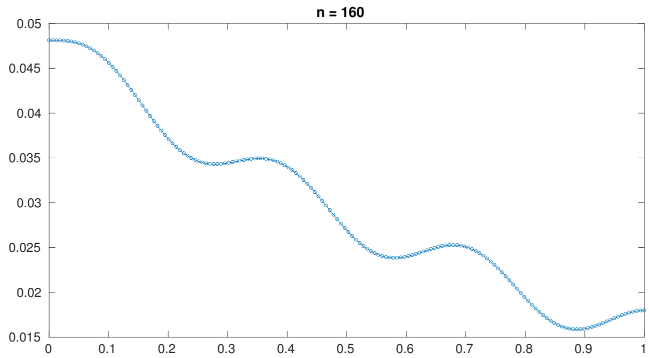
The figures below show the results for different number of vertices n as well. The same parameters are used, however the function $f(x) = sin(20x)$.



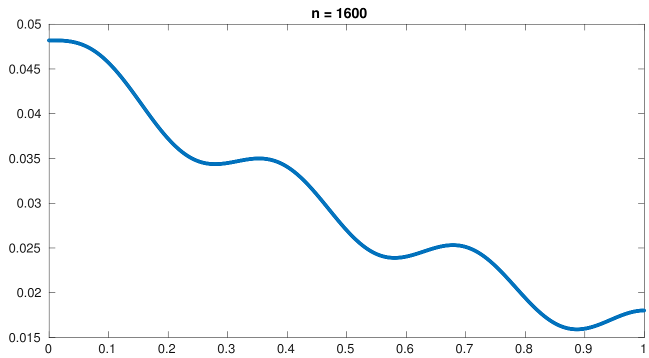

n = 20
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


n = 40
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


n = 80
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



The solution with $n = 10$ shows the boundary conditions aren't satisfied properly because the amount of vertices is too small. The other solutions satisfy the boundary conditions much better. Increasing n further doesn't seem to make the function improve much more, which can be seen in the figure below.



2 2D Assignment: Assignment 1

2.1 Theory

We start with the differential equation and boundary conditions

$$
\begin{aligned}
-\vec{\nabla}\cdot\left(D\vec{\nabla}u\right)+\lambda u = F(x,y), \quad (x,y)\in\Omega \\
F(x,y)=\exp\left(\frac{(x-0.3)^2+y^2}{0.1}\right) \\
\frac{\partial u}{\partial n}=0, \quad (x,y)\in\partial\Omega
\end{aligned}
\tag{6}
$$

where Ω is the L-shaped domain which is constructed by removing the left-upper quarter from the square of $(-1,1)\times(-1,1)$ and where $\partial\Omega$ is the boundary of Ω.

First we will derive the Weak Formulation, again using ϕ as the test function which satisfies the smoothness requirements.

$$
\begin{aligned}
\int_\Omega\left[-\vec{\nabla}\cdot\left(D\vec{\nabla}u\right)+\lambda u\right]\phi d\Omega &= \int_\Omega F(x,y)\phi d\Omega \\
-\int_\Omega \vec{\nabla}\cdot\left(D\vec{\nabla}u\right)\phi d\Omega+\int_\Omega\lambda u\phi d\Omega &= \int_\Omega F(x,y)\phi d\Omega \\
-\int_\Omega \vec{\nabla}\cdot\left(\left(D\vec{\nabla}u\right)\phi\right)-D\vec{\nabla}u\cdot\vec{\nabla}\phi d\Omega+\int_\Omega\lambda u\phi d\Omega &= \int_\Omega F(x,y)d\Omega \\
-\int_{\partial\Omega} D\left(\vec{\nabla}u\cdot\vec{n}\right)\phi d\Gamma+\int_\Omega D\vec{\nabla}u\cdot\vec{\nabla}\phi+\lambda u\phi d\Omega &= \int_\Omega F(x,y)\phi d\Omega \\
\int_\Omega D\vec{\nabla}u\cdot\vec{\nabla}\phi+\lambda u\phi d\Omega &= \int_\Omega F(x,y)\phi d\Omega
\end{aligned}
\tag{7}
$$

In eq. (7) we have used the boundary conditions and therefore they are contained in eq. (7).

Next we can derive the Galerkin formulation by substituting $\phi=\phi_i$ and $u\approx u^N=\sum_{j=1}^N c_j\phi_j$ into eq. (7).

$$
\begin{aligned}
\int_\Omega D\vec{\nabla}\left(\sum_{j=1}^N c_j\phi_j\right)\cdot\vec{\nabla}\phi_i+\lambda\left(\sum_{j=1}^N c_j\phi_j\right)\phi_i d\Omega &= \int_\Omega F(x,y)\phi_i d\Omega \\
\int_\Omega D\sum_{j=1}^N c_j\vec{\nabla}\phi_j\cdot\vec{\nabla}\phi_i=\lambda\sum_{j=1}^N c_j\phi_j\phi_i d\Omega &= \int_\Omega F(x,y)\phi_i d\Omega \\
\sum_{j=1}^N c_j\int_\Omega D\vec{\nabla}\phi_j\cdot\vec{\nabla}\phi_i+\lambda\phi_j\phi_i d\Omega &= \int_\Omega F(x,y)\phi_i d\Omega
\end{aligned}
\tag{8}
$$

We can write this as a linear system of equations $S\vec{u}=\vec{f}$ where

$$
\begin{aligned}
S_{ij} &= \int_\Omega D\vec{\nabla}\phi_j\cdot\vec{\nabla}\phi_i+\lambda\phi_j\phi_i d\Omega \\
f_i &= \int_\Omega F(x,y)\phi_i d\Omega
\end{aligned}
\tag{9}
$$

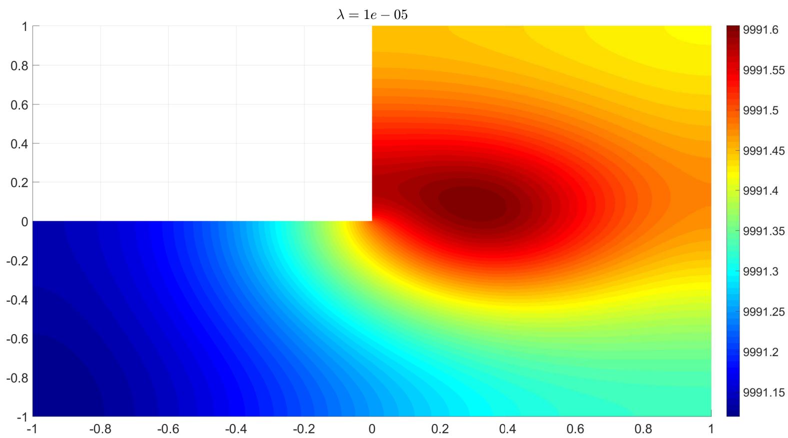
Using triangular elements e_k we find for the internal elements

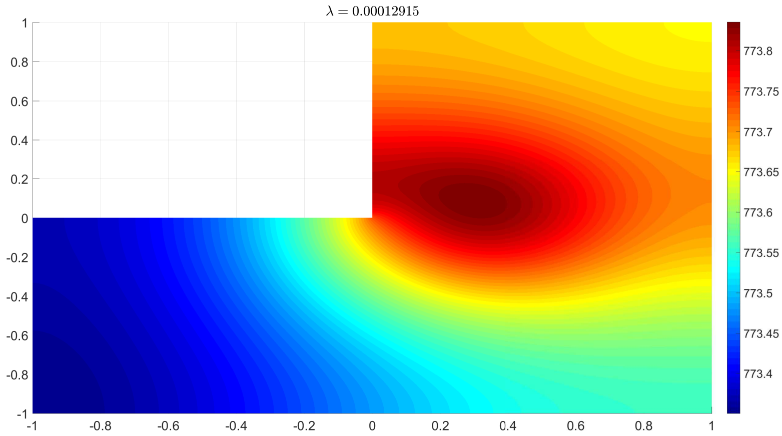
$$\begin{aligned}
S_{ij}^{e_k} &= \int_{e_k} D \begin{bmatrix} \beta_j & \gamma_j \end{bmatrix} \cdot \begin{bmatrix} \beta_i & \gamma_i \end{bmatrix} + \lambda \phi_j \phi_i d\Omega \\
&= \frac{|\Delta e_k|}{2} D \left(\beta_i \beta_j + \gamma_i \gamma_j\right) + \frac{|\Delta e_k|}{24} \lambda \left(1 + \delta_{ij}\right) \\
f_i &= \int_{e_k} F(x,y) \phi_i d\Omega \\
&= \frac{|\Delta e_k|}{6} \sum_{p=1}^{3} F(x_p, y_p) \phi_i(x_p, y_p) \\
&= \frac{|\Delta e_k|}{6} F(x_i, y_i).
\end{aligned} \tag{10}$$

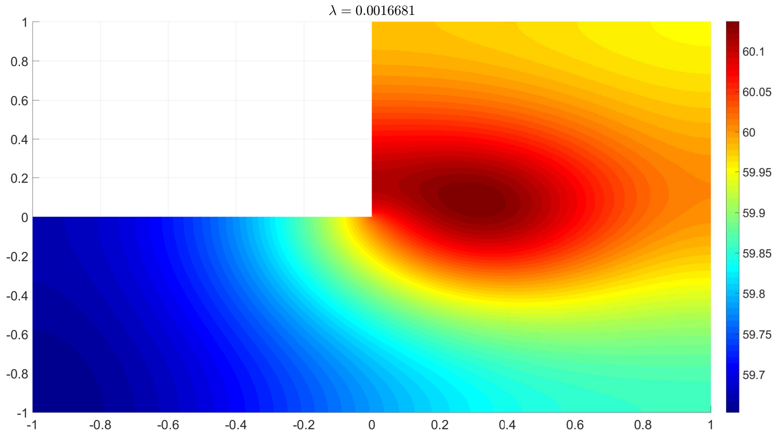
Because eq. (8) doesn't have boundary contributions the boundary elements are all 0.

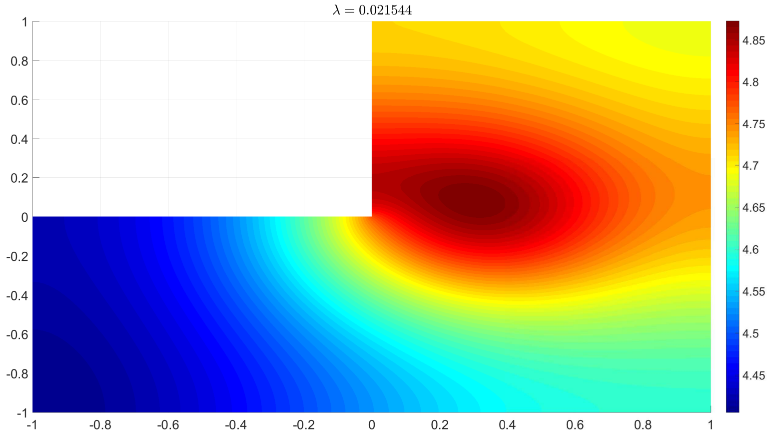
2.2 Modelling

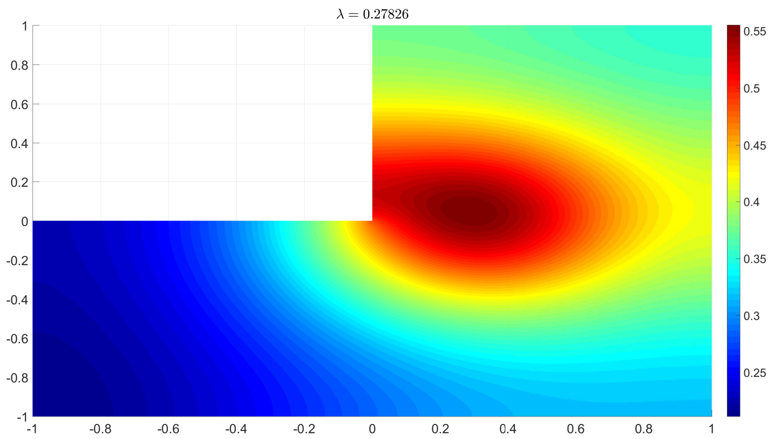
In the following figures $D = 0.23$ and λ is changed. All code is again available on Github (see footnote 1).



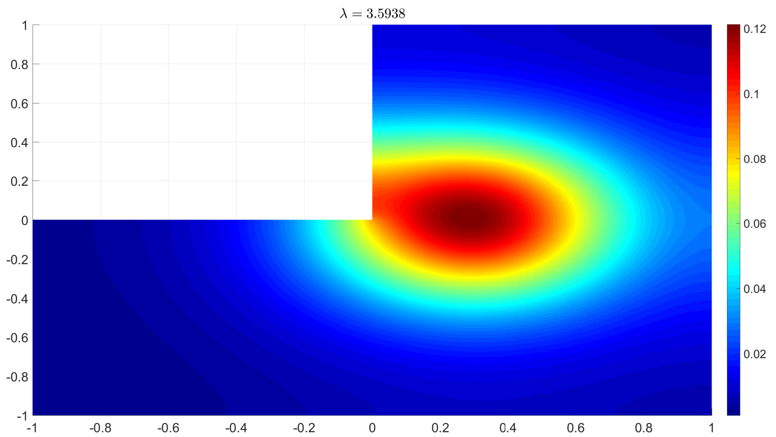

$\lambda = 0.00012915$


$\lambda = 0.0016681$

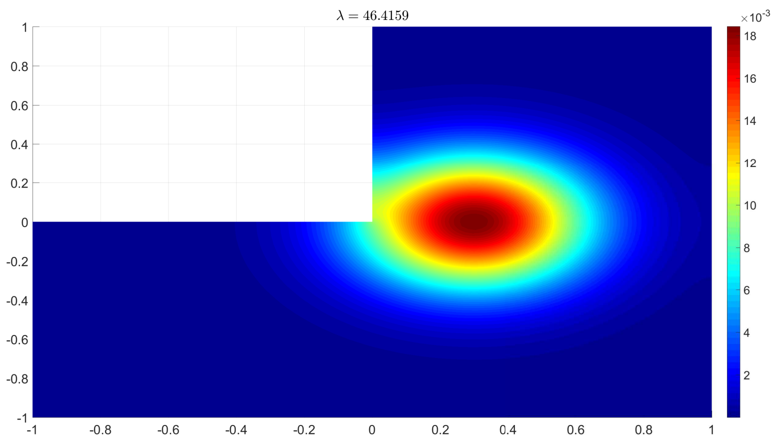

$\lambda = 0.021544$



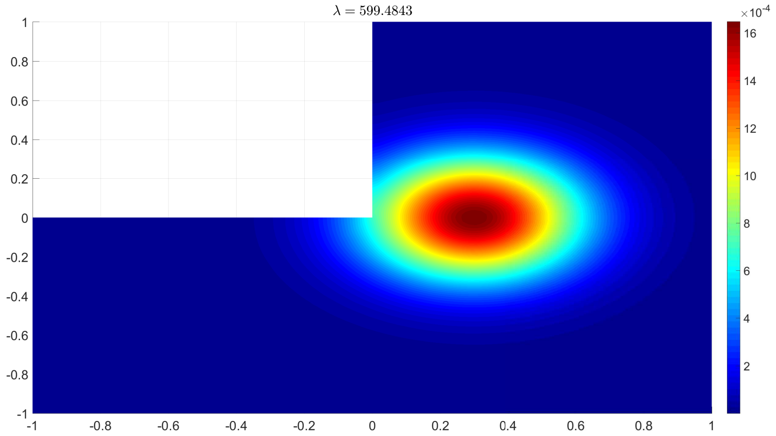
$\lambda = 0.27826$

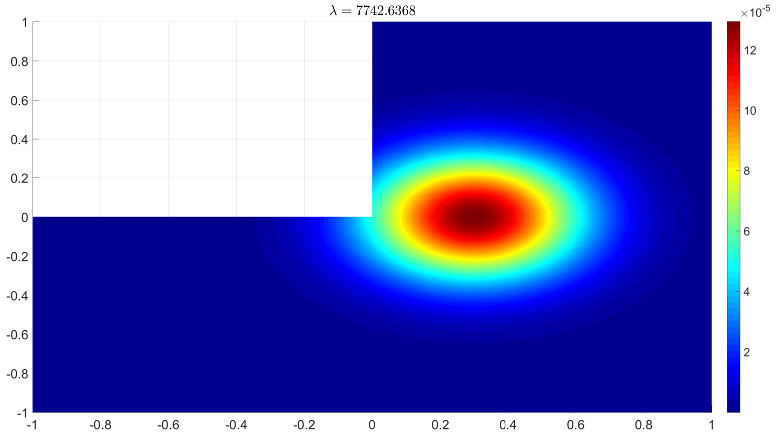


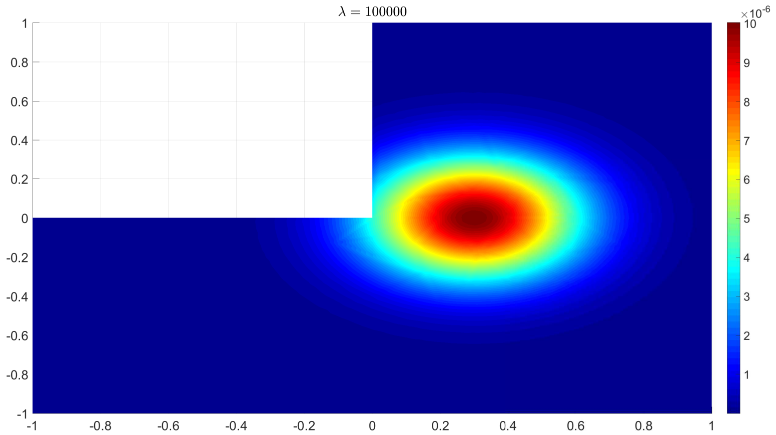
$\lambda = 3.5938$



$\lambda = 46.4159$
$\times 10^{-3}$


$\lambda = 599.4843$


$\lambda = 7742.6368$


$\lambda = 100000$

In these figures we see a stationary solution to eq. (6). The diffusion of the source is balanced with the decomposition. For high λ we see only the source remains and before much diffusion can happen the lactates are already decomposed. When λ is small, the lactates have to be in a higher concentration for the decomposition rate to catch up. This is because the decomposition rate is linear with u. Furthermore, because of the boundary conditions the lactates cannot leave the upper right corner, so here they reach a higher concentration. In the lower left corner more space is available to decompose the lactates. Here the concentration will always be lower than the upper right corner. This is also the reason for the fan shape in the lower right corner. Because the solution has to be smooth a fan transition between the high and low concentration areas is necessary.

As a final remark $\lambda = 0$ will result in a non-stationary solution, because of the lack of decomposition. The source will keep increasing the amount of lactates, and the diffusion will equalize the concentration, but there is not point at which decomposition will match regeneration.

3 Matlab Code

3.1 1D Assignment

3.1.1 GenerateMesh.m

```
x = linspace(0,1,n);
%x = x + [0, (2*rand(1,n-2)-1)/(2*n) , 0];
```

3.1.2 GenerateTopology.m

```
elmat = zeros(n-1,2);
elmat(:,1) = 1:(n-1);
elmat(:,2) = 2:n;
```

3.1.3 GenerateElementMatrix.m

```
S_elem = zeros(2,2);
h = abs(x(elmat(i,2)) - x(elmat(i,1)));
for k = 1:2
    for l = 1:2

        S_elem(k,l) = 1/h * D*(-1)^(k-l) + h*lambda*(1+(k==l))/6;
    end
end
```

3.1.4 GenerateElementVector.m

```
xc = [x(elmat(i,1)), x(elmat(i,2))];
h = abs(xc(1) - xc(2));

f_elem = h/2 * func(xc);
```

3.1.5 AssembleMatrix.m

```
S = zeros(n,n);
for i = 1:(n-1)
    GenerateElementMatrix
    for j = 1:2
        for k = 1:2
            S(elmat(i,j),elmat(i,k)) = S_elem(j,k) + S(elmat(i,j),elmat(i,k));
        end
    end
end
```

3.1.6 AssembleVector.m

```
f = zeros(n,1);
for i = 1:(n-1)
    GenerateElementVector;
    for j = 1:2
```

```
            f(elmat(i,j)) = f_elem(j) + f(elmat(i,j));
        end
end
```

3.1.7 func.m

```
function [ f ] = func( x )
%F Summary of this function goes here
%   Detailed explanation goes here
    f = sin(20*x);
end
```

3.1.8 Femsolve1d.m

```
clear
close all

N = [10, 20, 40, 80, 160];
nN = numel(N);
snN = ceil(sqrt(nN));
for q=1:nN

    n = N(q);
    lambda = 1;
    D = 1;
    GenerateMesh;
    GenerateTopology;
    AssembleMatrix;
    AssembleVector;
    u = S\f;

    figure(q)
    plot(x,u,'-o');
    set(gca,'FontSize',24)
    title(['n = ', num2str(n)])
end
```

3.2 2D Assignment

3.2.1 WI4243Mesh.m

```
[p,e,t] = initmesh(Geometry);

for i = 1:loops
[p,e,t] = refinemesh(Geometry,p,e,t); % gives gridrefinement
end

pdemesh(p,e,t); % plots the geometry and mesh

x = p(1,:); y = p(2,:);

n = length(p(1,:));

elmat = t(1:3,:);
elmat = elmat';
elmatbnd = e(1:2,:);
elmatbnd = elmatbnd';
```

3.2.2 WI4243Comp.m

```
% Construction of linear problem

BuildMatricesandVectors;

% Solution of linear problem

u = S \ f;
```

3.2.3 GenerateElementMatrix.m

```

% Module for element mass matrix for reactive term
%
% Output: Selem ====== 2 x 2 matrix
%
% Selem(1,1), Selem(1,2), Selem(1,3), Selem(2,1), Selem(2.2), Selem(2,3),
%
% Selem(3,1), Selem(3,2), Selem(3,3) to be computed in this routine.
%
% elmat(i,1), elmat(i,2), elmat(i,3) give the index numbers of the vertices of element i
%
% x(elmat(i,j)), y(elmat(i,j)) give the coordinates of the vertices
%
```

```
% i = index number of element, imported from AssemblyStepStiffnessMatrix.m
%
% Selem(index1,index2) = (grad phi(elmat(i,index1)),grad phi(i,index2))
%

xc=zeros(1,topology);
yc = zeros(1,topology);
Selem = zeros(topology,topology);

for index1 = 1:topology
        xc(index1) = x(elmat(i,index1));
        yc(index1) = y(elmat(i,index1));
end

D = [1 xc(1) yc(1);1 xc(2) yc(2);1 xc(3) yc(3)];
Delta = det(D);

B_mat = D \ eye(3);

alpha = B_mat(1,1:3);
beta  = B_mat(2,1:3);
gamma = B_mat(3,1:3);

for index1 = 1:topology
    for index2 = 1:topology
                Selem(index1,index2) = abs(Delta)/2*DiffCoeff*(beta(index1)*beta(index2)+gamma(index1)*gamma(index2)) + lambda * (1+(index1==index2)) * abs(Delta)/24;
    end
end
```

3.2.4 GenerateElementVector.m

```

% Module for element mass matrix for reactive term
%
% Output: felem ===== vector of two components
%
% felem(1), felem(2) to be computed in this routine.

xc=zeros(1,topology);
yc = zeros(1,topology);
felem = zeros(1,topology);

for index1 = 1:topology
        xc(index1) = x(elmat(i,index1));
```

```
        yc(index1) = y(elmat(i,index1));
end


for index1 = 1:topology
                felem(index1) = abs(Delta)/6 * Fxy(xc(
                    index1), yc(index1));
end
```

3.2.5 GenerateBoundaryElementMatrix.m

```
xc = zeros(1,topologybnd);
yc = zeros(1,topologybnd);
BMelem = zeros(topologybnd,topologybnd);

for index1=1:topologybnd
        xc(index1) = x(elmatbnd(i,index1));
        yc(index1) = y(elmatbnd(i,index1));
end

lek = sqrt((xc(2)-xc(1))^2 + (yc(2)-yc(1))^2);

for index1=1:topologybnd
                BMelem(index1,index1) = 0;
end
```

3.2.6 GenerateBoundaryElementVector.m

```
xc = zeros(1,topologybnd);
yc = zeros(1,topologybnd);
bfelem = zeros(topologybnd,topologybnd);

for index1 = 1:topologybnd
        xc(index1) = x(elmatbnd(i,index1));
        yc(index1) = y(elmatbnd(i,index1));
end

lek = sqrt((xc(2)-xc(1))^2+(yc(2)-yc(1))^2);

for index1 = 1:topologybnd
                bfelem(index1) = 0;
end
```

3.2.7 BuildMatricesandVectors.m

```

%
% This routine constructs the large matrices and vector.
%
% The element matrices and vectors are also dealt with.
%
```

```
%
% First the internal element contributions
%
% First Initialisation of large discretisation matrix, right-hand side vector

S                = sparse(n,n); % stiffness matrix

f                = zeros(n,1); % right-hand side vector

%
% Treatment of the internal (triangular) elements
%

for i = 1:length(elmat(:,1)) % for all internal elements
        GenerateElementMatrix; % Selem
    for ind1 = 1:topology
        for ind2 = 1:topology
            S(elmat(i,ind1),elmat(i,ind2))      = S(elmat(i,ind1),elmat(i,ind2)) + Selem(ind1,ind2);
        end
    end
        GenerateElementVector; % felem
    for ind1 = 1:topology
        f(elmat(i,ind1)) = f(elmat(i,ind1)) + felem(ind1);
    end
end

% Next the boundary contributions

for i = 1:length(elmatbnd(:,1)) % for all boundary elements extension of mass matrix M and element vector f
        GenerateBoundaryElementMatrix; % BMelem
    for ind1 = 1:topologybnd
        for ind2 = 1:topologybnd
            S(elmatbnd(i,ind1),elmatbnd(i,ind2)) = S(elmatbnd(i,ind1),elmatbnd(i,ind2)) + BMelem(ind1,ind2);
        end
    end
        GenerateBoundaryElementVector; % bfelem
    for ind1 = 1:topologybnd
        f(elmatbnd(i,ind1)) = f(elmatbnd(i,ind1)) + bfelem(ind1);
    end
end
```

3.2.8 Fxy.m

```
function [ F ] = Fxy( x, y )
%FXY Summary of this function goes here
%   Detailed explanation goes here
    F = exp( -((x-0.3).^2 + y.^2)/0.1 );
end
```

3.2.9 WI4243Post.m

```
figure();
trisurf(elmat,x,y,u)
figure(3);
subplot(plots(1), plots(2), loop)
trisurf(elmat,x,y,u);
title(['$$\lambda = ', num2str(lambda), '$$'], '
    Interpreter', 'latex')
set(gca,'FontSize',20)
view(2); shading interp; colormap jet; colorbar; set(gcf,
    'renderer','zbuffer')
```

3.2.10 Run.m

```
clear;
close all;
Geo = {'circleg', 'squareg', 'lshapeg'};

%% Generate mesh
Geometry = Geo{3};
loops = 2;

WI4243Mesh;

%% Compute S, f and u
DiffCoeff = 0.23;
plots = [2,5];
spacing = plots(1)*plots(2);

lambdaV = logspace(-5,5,spacing);
for I = 1:plots(1)
   for J = 1:plots(2)
        loop = (I-1)*plots(2)+J;
        lambda = lambdaV(loop);
        topology = 3; topologybnd = 2;
        WI4243Comp;

        %% Generate solution
        WI4243Post;
    end
end
```