# MATH610-600

## *Programming Assignment #3*

April 09, 2019

LU SUN

# 1 Problem Specifications

## 1.1 Exercise 1: Triangulation

Implement a sequence of triangulations $\tau_i, i = 1, ..., 4$ of the L-shaped domain

$$\Omega = [-1, 1]^2 \backslash [0, 1] \times [-1, 0] \tag{1}$$

with mesh sizes $h_i$ defined in Table 1.

| level | $h$ | $a(h) = 1/2h^2$ |
|---|---|---|
| 1 | 0.2 | 0.02 |
| 2 | 0.1 | 0.005 |
| 3 | 0.05 | 0.00125 |
| 4 | 0.025 | 0.003125 |

Table 1: Mesh sizes and representative maximal area sizes.

- Write a file 'LShapedDomain.poly' that descirbed the above domain(1).
- Use command to generate the LShapedDomain.1, LShapedDomain.2, LShapedDomain.3, LShapedDomain.4 series (.poly, .node, .ele, .edge) files according to all four meshes above in the Table 1.
- Import these meshes into matlab by 'TriangleReader.m' file and plot the elements by 'exampleTriangleReader.m' file.
- Title each plot with the mesh size $h = \_\_$.

## 1.2 Exercise 2: Lagrange Interpolation with Linear Elements

Be given following 3 functions:

$$\begin{array}{lll} \text{(i)} & u(X) = & |x - y| \\ \text{(ii)} & u(X) = & \sin \pi(x + y) \\ \text{(iii)} & u(X) = & \begin{cases} 1, & x^2 + y^2 \geq \frac{1}{4} \\ 0, & otherwise \end{cases} \end{array}$$

Let $I_h$ be the Lagrange interpolation.

- Compute error $||u - I_h u||_{L^2(\Omega)}$ on each of the four meshes for each of the above three functions( element by element).
- Plot element by element.
- Describe the three linear basis elements and evaluate the function value at three nodal on each cell.
- Compile a table of errors and convergence rates for each function.
- Discuss findings: How does smoothness of function affect convergence rates.
- Choose quadrature rules based on convergence rates and accuracy.

### 1.3 Exercise 3: $L^2$ Projection with Linear Elements

Let $\Pi_h : H^1 \to \mathbb{P}^1(\tau_h)$ be the $L^2$ projector onto the linear Lagrange finite element space.

- For each of the functions $u(X)$ in Exercise 2 and the triangulations of Exercise 1, compute the discrete finite element solution( $L^2$ projection) $\Pi_h u(X)$.
- Compute the $L^2(\Omega), H^1(\Omega), L^\infty(\Omega)$ errors: $||u - \Pi_h u||$.
- Construct tables of errors and convergence rates for each function.

### 1.4 Exercise 4: Poisson Equation with Dirichlet Boundary Conditions

Solve the Poisson equation with Dirichlet boundarys on the domain above.

$$\begin{cases} -\Delta u = f, & X \in \Omega \\ u = u_d, & X \in \partial\Omega \end{cases} \tag{2}$$

Be given $u(X) = \cos(2\pi x)\cos(2\pi y)$.

- Compute $f(X)$ by hand.
- Compute $u_d$ by hand.
- Compute $u_h(X)$ on the four triangulations.
- Plot the solution on the fourth mesh $\tau_4$.
- Compute the $L^2, H^1, L^\infty$ errors($||u - u_h||$).
- Display the errors in a table with rates.
- Plot the error values in a log-log plot against the h value.

### 1.5 Exercise 5: Dirichlet problem with known solution on shaped domain

Let $(r, \theta)$ be standard polar coordinates about the origin and $u(r, \theta) = r^{\frac{2}{3}} \sin(2\theta/3)$, then $-\Delta u = 0$ in $\Omega$ and $g = u$ on $\partial\Omega$

#### 1.5.1 By hand

- Show $u \in H^1(\Omega)$
- Show $u \notin H^2(\Omega)$

#### 1.5.2 On Matlab

- Compute $u_h(X)$ on the four triangulations.
- Use the Lagrange interpolant to determine Dirichlet boundary data.
- Compute the $L^2, H^1, L^\infty$ errors($||u - u_h||$).
- Display the errors in a table with rates.
- Plot the error values in a log-log plot against the h value.
- Comment briefly on the relationship between $H^1$ convergence rate and regularity (smoothness) of u.

# 2 Preliminaries

## 2.1 Exercise 1

### 2.1.1 *LShapedDomain.poly*

This file is used for describe the above Domain(1). The first part is the 6 nodes counter-clockwise start from point (0,0). The second part is the definition for 6 sizes. The third part means no holes inside. The code is as follow:

```
6 2 0 1

1 0 0 1
2 1 0 1
3 1 1 1
4 -1 1 1
5 -1 -1 1
6 0 -1 1

6 1

1 1 2 1
2 2 3 1
3 3 4 1
4 4 5 1
5 5 6 1
6 6 1 1

0
```

### 2.1.2 *exampleTriangleReader.m*

In the *exampleTriangleReader.m*, $T$ is imported to record triangle in 'LshapedDomian.1' to 'LshapedDomian.4'and plot each of 4 meshes. The code is as follow:

```
function T=exampleTriangleReader(choose)
% import into T, plot the situation choose
% choose    hi
%   1      0.2
%   2      0.1
%   3      0.05
%   4      0.025


% run triangle with the following flags on the A.poly file
%
% ./triangle -pq28ea0.02 LShapedDomain.poly
%
% which generates LShapedDomain.1.{node, ele, edge} files

T = TriangleReader(['LShapedDomain.',num2str(choose),'.node'],...
['LShapedDomain.',num2str(choose),'.ele'],...
['LShapedDomain.',num2str(choose),'.edge']);
%T = ...
    TriangleReader('LShapedDomain.1.node','LShapedDomain.1.ele','LShapedDomain.1.edge');
```

```
figure
% another approach to plotting elements with fill color specified at each ...
    node
for i = 1:T.n_elements
    V = T.nodes( T.elements(i,:), :);
    f = [1 2 3];
    Col = V(:,1); % color by x coordinate
    patch('Faces', f, 'Vertices',V,'FaceVertexCData', Col,...
        'EdgeColor','black','FaceColor','interp','LineWidth',1);
end

switch(choose)
    case 1
        title('h=0.2');
    case 2
        title('h=0.1');
    case 3
        title('h=0.05');
    case 4
        title('h=0.025');
end
end
```

To show out the plots, we run the following code:

```
for i =1:4
    exampleTriangleReader(i);
end
```

## 2.2 Exercise 2

### 2.2.1 *feEval.m*

In this file, three linear basis elements on regular triangle ((0,0),(1,0),(0,1)) is shown. The code is as follow:

```
function [FE_at_quad] = feEval( Quad, p )
x = Quad.xhat(:,1);
y = Quad.xhat(:,2);
Identity = ones(Quad.nq,1);
if (p == 1)
    %                       phi_1, phi_2, phi_3
    % Linear elements :   1-x-y,   x,     y
    % x derivative    :     -1 ,   1,     0
    % y derivative    :     -1 ,   0,     1
    FE_at_quad.hat_phi  = [1-x-y, x, y]; %nq * 3

    FE_at_quad.hat_phix = [-1, 1, 0].* Identity; %nqx3
    FE_at_quad.hat_phiy = [-1, 0, 1].* Identity; %nqx3
end
```

### 2.2.2 *getQuadOnRefElement.m*

In this file, the quadrature rules of weights and nodes (x,y) are recorded.

### 2.2.3 *comput_error.m*

In this file, local $L^2$ norm is calculated. It is the same as previous $L^2$ norm calculated in 1D. Nothing have been changed.

### 2.2.4 *LagrangeInterpolation.m*

In this file, the total $L^2$ error and plot of different functions(3 kinds) with different quadrature(5 kinds) and mesh sizes(4 kinds) can be shown. The code is as follow:

```
function L2error=LagrangeInterpolation(funchoose,quad_n_points,ifplot)
%funchoose     functions
%   1           u=|x-y|
%   2           u=sin(pi(x+y))
%   3           u=1 or 0
%
%quad_n_points=1;2;3;4;5;
%
%ifplot          meaning
% false          not plot
% true             plot

L2error=zeros(4,1);
p=1;

switch funchoose
    case 1
        u=@(x)abs(x(1)-x(2));
        fprintf('u=|x-y|:\n');
    case 2
        u=@(x)sin(pi*(x(1)+x(2)));
        fprintf('u=sin(pi(x+y))\n');
    case 3
        u=@(x)double((x(1)^2+x(2)^2≥1/4)&1);
        fprintf('u=1 or 0\n');
end

for i=1:4
    T = TriangleReader(['LShapedDomain.',num2str(i),'.node'],...
        ['LShapedDomain.',num2str(i),'.ele'],...
        ['LShapedDomain.',num2str(i),'.edge']);

    L2sqrd = 0;
    if ifplot
        figure;
    end
    for ele=1:T.n_elements
        %vertical V = [v1;v2;v3]=[X,Y]
        V = T.nodes(T.elements(ele,:),:);

        %The coefficients for basis function
        uh=[u(V(1,:));u(V(2,:));u(V(3,:))];
```

```

        if ifplot
            %plot
            %X=V(:,1),Y=V(:,2),Z=uh;
            patch('XData',V(:,1),'YData',V(:,2),'ZData', ...
                uh,'FaceVertexCData', uh,...
                'EdgeColor','black','FaceColor','interp','LineWidth',1);% ...
                    color by Z coordinate
            view(3);
        end

        %L2 error
        Quad_Error = getQuadOnRefElement(quad_n_points);
        FE_at_Quad_Error = feEval(Quad_Error, p);
        [localL2sqrd] = compute_error(V, uh, u, Quad_Error, ...
            FE_at_Quad_Error, p);
        L2sqrd = L2sqrd + localL2sqrd;
    end
    L2error(i) = sqrt(L2sqrd);
    switch(i)
    case 1
        title('h=0.2');
        fprintf('h=0.2,error=%f\n',L2error(i));
    case 2
        title('h=0.1');
        fprintf('h=0.1,error=%f\n',L2error(i));
    case 3
        title('h=0.05');
        fprintf('h=0.05,error=%f\n',L2error(i));
    case 4
        title('h=0.025');
        fprintf('h=0.025,error=%f\n',L2error(i));
    end

end
end
```

### 2.2.5 *BestQuadratureFinder.m*

In this file, we try to plot meshes and errors with 5 kinds quadrature in one picture for each function. The code is as follow:

```
ERROR1 = zeros(4,5);
for quadn = 1:5
    fprintf('quadrature k=%d\n',quadn);
    ERROR1(:,quadn) = LagrangeInterpolation(1,quadn,false);
end

ERROR2 = zeros(4,5);
for quadn = 1:5
    fprintf('quadrature k=%d\n',quadn);
    ERROR2(:,quadn) = LagrangeInterpolation(2,quadn,false);
end

ERROR3 = zeros(4,5);
for quadn = 1:5
```

```
    fprintf('quadrature k=%d\n',quadn);
    ERROR3(:,quadn) = LagrangeInterpolation(3,quadn,false);
end

h=[0.2,0.1,0.05,0.025];
figure;
plot(h,ERROR1(:,1),h,ERROR1(:,2),':',h,ERROR1(:,3),'--',...
    h,ERROR1(:,4),'+-',h,ERROR1(:,5),'o-');
legend('k=1','k=2','k=3','k=4','k=5');
xlabel('mesh size');
ylabel('error');
title('u=|x-y|');

figure;
plot(h,ERROR2(:,1),h,ERROR2(:,2),':',h,ERROR2(:,3),'--',...
    h,ERROR2(:,4),'+-',h,ERROR2(:,5),'o-');
legend('k=1','k=2','k=3','k=4','k=5');
xlabel('mesh size');
ylabel('error');
title('u=sin(pi(x+y))');

figure;
plot(h,ERROR3(:,1),h,ERROR3(:,2),':',h,ERROR3(:,3),'--',...
    h,ERROR3(:,4),'+-',h,ERROR3(:,5),'o-');
legend('k=1','k=2','k=3','k=4','k=5');
xlabel('mesh size');
ylabel('error');
title('u=1 or 0');
```

### 2.2.6 *Exercise2.m*

In this file, it show out the entire results. The code is as follow:

```
%Find best quadrature k for problem
BestQuadratureFinder;

%choose quadrature k=3
k = 3;
%compute L2 errors for function j=1,2,3
Errors = zeros(4,6);
for j=1:3
    Errors(:,2*j-1) = LagrangeInterpolation(j,k,true);
end

%compute convergence rate
for j=1:3
    for i=2:4
        Errors(i,2*j) = Errors(i-1,2*j-1)/Errors(i,2*j-1);
    end
end

Errors
```

## 2.3 Exercise 3

### 2.3.1 The same files as above

In Exercise 3, we will use the above files: $TriangleReader.m$, $feEval.m$, $getQuadOnRefElement.m$.

### 2.3.2 $local\_assemblyL2.m$

In this file, local matrix is assembled. It's the same as that in 1D problem, and we only need to change the dimension of $mat\_B$ form 1( 2 nodals on each interval in 1D)to 2( 3 nodals on each triangle in 2D) by the following code:

```
mat_B = [(localnodes(2,:) - localnodes(1,:))',(localnodes(3,:) - ...
    localnodes(1,:))'];
```

The way to get $rhs\_vals$ is a little different:

```
rhs_vals=zeros(Quad.nq,1);
for t=1:Quad.nq
    rhs_vals(t) = f(q_points(t,:));
end
```

### 2.3.3 $constructDoFHandler.m$

In this file, only the following code need to be changed form 1D to 2D, for each triangle have 3 nodals:

```
DoFHandler.dofs(:,1:3) = T.elements;
```

### 2.3.4 $computeLocalInfErrors.m$

It's used for compute local infinity errors. And only the following code is different form that in 1D:

```
mat_B = [(localnodes(2,:) - localnodes(1,:));(localnodes(3,:) - ...
    localnodes(1,:))]; %[(dim)x(dim)]

u_exact_at_q_points = zeros(Quad.nq,1);
for t = 1:Quad.nq
    u_exact_at_q_points(t) = u_exact(q_points(t,:));
end
```

### 2.3.5 $computeLocalErrors.m$

It's used for compute local both $L^2$ and $H^1$ errors. And only the following code is different form that in 1D:

```
mat_B = [(localnodes(2,:) - localnodes(1,:));(localnodes(3,:) - ...
    localnodes(1,:))]; %[(dim)x(dim)]
```

```

inv_B = 1/det_B*[mat_B(2,2),-mat_B(1,2);-mat_B(2,1),mat_B(1,1)];

% exact solutions and gradients at quadrature points
u_exact_at_q_points = zeros(Quad.nq,1);
grad_u_exact_at_q_points = zeros(Quad.nq,2);
for t = 1:Quad.nq
    u_exact_at_q_points(t) = u_exact(q_points(t,:));
    grad_u_exact_at_q_points(t,:) = grad_u_exact(q_points(t,:))';
end
```

### 2.3.6 *L2projection.m*

It's a file to give the errors and rates with plots. The code is as follow:

```
function [uh,E] = L2projection(funchoose,quad_n_points,p,ifplot)
%
%p: polynomial degree of lagrange finite element basis, here p=1
%
% Choose quadrature on reference element and evaluate finite element shape
%functions on reference element at quadrature points,here quad_n_points = 3

L2=zeros(4,1);
H1=zeros(4,1);
Linf=zeros(4,1);
E=zeros(4,6);

switch funchoose
    case 1
        u_exact = @(x)abs(x(1)-x(2));
        grad_u_exact = @(x)[1;-1]*double(x(1)≥x(2))+...
                [-1;1]*double(x(1)<x(2));
        fprintf('u=|x-y|\n');
    case 2
        u_exact = @(x)sin(pi*(x(1)+x(2)));
        grad_u_exact = @(x) [pi*cos(pi*(x(1)+x(2)));...
                pi*cos(pi*(x(1)+x(2)))];
        fprintf('u=sin(pi(x+y))\n');
    case 3
        u_exact = @(x)double(x(1)^2+x(2)^2≥1/4);
        grad_u_exact = @(x) [0 ;0];
        fprintf('u=0 or 1\n');
end

% right hand side function
f = @(x)u_exact(x);

% begin FEM code
for j=1:4
        T = TriangleReader(['LShapedDomain.',num2str(j),'.node'],...
            ['LShapedDomain.',num2str(j),'.ele'],...
            ['LShapedDomain.',num2str(j),'.edge']);
    %
    % setup_system
    %
    DoFHandler = constructDoFHandler(T,p);

```

```
    RHS = zeros(DoFHandler.n_dofs,1);
    A = spalloc(DoFHandler.n_dofs, DoFHandler.n_dofs, 15*T.n_nodes);

    Quad = getQuadOnRefElement(quad_n_points);

    [ FE_at_Quad] = feEval( Quad, p );

    % assemble_system
    for cell = 1:T.n_elements
        dofIndices = DoFHandler.dofs(cell,:);
        vertices = T.nodes(T.elements(cell,:),:);
        [cell_matrix,cell_rhs,¬] = local_assemblyL2(vertices,f, ...
            FE_at_Quad, Quad, p);

        % contribute local terms to global stiffness and RHS structures
        A(dofIndices,dofIndices) = A(dofIndices,dofIndices) + cell_matrix;
        RHS(dofIndices) = RHS(dofIndices) + cell_rhs;
    end

    % solve_system
    %
    % Only solve system for unconstrained dofs (the non Dirichlet ones)
    uh = A \ RHS;

    %plot uh
    if ifplot
        figure;
        for cell=1:T.n_elements
            dofIndices = DoFHandler.dofs(cell,:);
            vertices = T.nodes(T.elements(cell,:),:);
            X = vertices(:,1);
            Y = vertices(:,2);
            Z=zeros(3,1);
            for i=1:3
                Z(i)=uh(dofIndices(i));
            end
            patch('XData',X,'YData',Y,'ZData', Z,'FaceVertexCData', Z,...
            'EdgeColor','black','FaceColor','interp','LineWidth',1);% ...
                color by Z coordinate
          view(3);
        end
        switch(j)
            case 1
                title('h=0.2');
            case 2
                title('h=0.1');
            case 3
                title('h=0.05');
            case 4
                title('h=0.025');
       end
    end

    %compute error
    Quad_Error = getQuadOnRefElement(quad_n_points);
    FE_at_Quad_Error = feEval(Quad_Error, p);

    % In the L inf error, we don't need a weight, only a bunch of xhat
    %locations so we combine the Quad_Error xhats with a uniformly
```

```
    %distributed set of nodes through the reference element to make a nice
       %L inf quadrature rule

    n_inf_nodes = 10;
    QuadInf_Error.nq = Quad_Error.nq+n_inf_nodes;
    addpoints=[0,0;1/3,0;2/3,0;1,0;0,1/3;1/3,1/3;2/3,1/3;0,2/3;1/3,2/3;0,1];
    QuadInf_Error.xhat = [Quad_Error.xhat; addpoints];
    FE_at_QuadInf_Error = feEval(QuadInf_Error, p);

    % Loop through the cells and compute the local errors and aggregate
    %them according to the norms

    L2sqrd = 0;
    H1sqrd = 0;
    Linferror = 0;
    for cell = 1:T.n_elements
            dofIndices = DoFHandler.dofs(cell,:); % [1x(p+1)]  extract ...
                indices pertaining to cell nodes
            vertices = T.nodes(T.elements(cell,:),:); % [(dim+1)xdim] ...
                coordinates of vertices

            %localL2sqrd = compute_error(vertices, uh(dofIndices), ...
                u_exact, Quad_Error, FE_at_Quad_Error, p);
            [localL2sqrd, localH1sqrd] = computeLocalErrors(vertices, ...
                uh(dofIndices), u_exact, grad_u_exact, Quad_Error, ...
                FE_at_Quad_Error, p);
            L2sqrd = L2sqrd + localL2sqrd;
            H1sqrd = H1sqrd + localH1sqrd;

            localLinf = computeLocalInfErrors(vertices, uh(dofIndices), ...
                u_exact, QuadInf_Error, FE_at_QuadInf_Error,p);
            Linferror = max( Linferror, localLinf);
    end

    L2(j) = sqrt(L2sqrd);
    H1(j) = sqrt(H1sqrd);
    Linf(j)=Linferror;
end

h=[0.2,0.1,0.05,0.025];
for i=1:4
    E(i,:)=[Linf(i),0,L2(i),0,H1(i),0];

    if (i>1)
        E(i,2) = E(i-1,1)/E(i,1); %Linf error rate
        E(i,4) = E(i-1,3)/E(i,3); %L2 error rate
        E(i,6) = E(i-1,5)/E(i,5);  %H1 error rate
    end

    fprintf('%f &%f& %f &%f &%f &%f &%f \\\\ \n',h(i), ...
        E(i,1),E(i,2),E(i,3),E(i,4),E(i,5),E(i,6));
end
end
```

### 2.3.7 *Exercise3.m*

It's a file to shown all the results needed in Exercise 3.

```
k=3;
p=1;
for i = 1:3
    L2projection(i,k,p,true);
end
```

### 2.4 Exercise 4

In this exercise, we only need to rewrite two different files.

#### 2.4.1 *DirichletBoundaryCondition.m*

The code is as follow:

```
ifplot=True;
p = 1;  % polynomial degree of lagrange finite element basis
L2=zeros(4,1);
H1=zeros(4,1);
Linf=zeros(4,1);
E=zeros(4,6);

u_exact = @(x)cos(2*pi*x(1)).*cos(2*pi*x(2));
grad_u_exact = @(x) [-2*pi*sin(2*pi*x(1)).*cos(2*pi*x(2)),...
                     -2*pi*cos(2*pi*x(1)).*sin(2*pi*x(2))];

% right hand side function
f = @(x)(8*pi^2)*(cos(2*pi*x(1)).*cos(2*pi*x(2)));
g_D = @(x) u_exact(x);

for j=1:4
    T = TriangleReader(['LShapedDomain.',num2str(j),'.node'],...
            ['LShapedDomain.',num2str(j),'.ele'],...
            ['LShapedDomain.',num2str(j),'.edge']);

    DoFHandler = constructDoFHandler(T,p);
    uh = zeros(DoFHandler.n_dofs,1);
    RHS = zeros(DoFHandler.n_dofs,1);
    A = spalloc(DoFHandler.n_dofs, DoFHandler.n_dofs, 15*T.n_nodes);
    % upper bound on NNZ in matrix for 1D is 2*p+1 interactions per node ...
        (p on either side plus itself)

    quad_n_points = 5;
    Quad = getQuadOnRefElement(quad_n_points);
    [ FE_at_Quad] = feEval( Quad, p );

    for cell = 1:T.n_elements
        dofIndices = DoFHandler.dofs(cell,:); % [1x(p+1)]  extract ...
            indices pertaining to cell nodes
        vertices = T.nodes(T.elements(cell,:),:); % [(dim+1)xdim] ...
            coordinates of vertices

        [cell_matrix,cell_rhs,¬] = local_assembly(vertices,f, FE_at_Quad, ...
            Quad, p);

        % contribute local terms to global stiffness and RHS structures
```

```
        A(dofIndices,dofIndices) = A(dofIndices,dofIndices) + cell_matrix;
        RHS(dofIndices) = RHS(dofIndices) + cell_rhs;
    end

    % solve_system
    %
    % Only solve system for unconstrained dofs (the non Dirichlet ones)
    for t=1:size(DoFHandler.dirichletdofs)
        uh(DoFHandler.dirichletdofs(t)) = ...
            g_D(DoFHandler.dirichletdofs_coordinates(t,:));
    end

    RHS = RHS - A*uh;
    %
    % solve_system
    %
    % Only solve system for unconstrained dofs (the non Dirichlet ones)
    uh(DoFHandler.freedofs) = A(DoFHandler.freedofs, DoFHandler.freedofs) ...
        \ RHS (DoFHandler.freedofs);

    %compute error
    Quad_Error = getQuadOnRefElement(quad_n_points);
    FE_at_Quad_Error = feEval(Quad_Error, p);

    L2sqrd = 0;
    H1sqrd = 0;
    Linferror = 0;
    for cell = 1:T.n_elements
        dofIndices = DoFHandler.dofs(cell,:); % [1x(p+1)]  extract ...
            indices pertaining to cell nodes
        vertices = T.nodes(T.elements(cell,:),:); % [(dim+1)xdim] ...
            coordinates of vertices
        [localL2sqrd, localH1sqrd] = computeLocalErrors(vertices, ...
            uh(dofIndices), u_exact, grad_u_exact, Quad_Error, ...
            FE_at_Quad_Error, p);
        L2sqrd = L2sqrd + localL2sqrd;
        H1sqrd = H1sqrd + localH1sqrd;
        localLinf = computeLocalInfErrors(vertices, uh(dofIndices), ...
            u_exact, Quad_Error, FE_at_Quad_Error,p);
        Linferror = max( Linferror, localLinf);
    end
    L2(j) = sqrt(L2sqrd);
    H1(j) = sqrt(H1sqrd);
    Linf(j)=Linferror;
end

h=[0.2,0.1,0.05,0.025];
for i=1:4
    E(i,:)=[Linf(i),0,L2(i),0,H1(i),0];

   if (i>1)
       E(i,2) = E(i-1,1)/E(i,1);
       %E(i,2) = log(E(i,1)/E(i-1,1))/log(h(i)/h(i-1)); %Linf error
       E(i,4) = E(i-1,3)/E(i,3);
       %E(i,4) = log(E(i,3)/E(i-1,3))/log(h(i)/h(i-1)); %L2 error
       E(i,6) = E(i-1,5)/E(i,5);
       %E(i,6) = log(E(i,5)/E(i-1,5))/log(h(i)/h(i-1)); %H1 error
   end

```

```
    fprintf('%f&%f& %f& %f& %f& %f& %f\\\\\n', ...
        h(i),E(i,1),E(i,2),E(i,3),E(i,4),E(i,5),E(i,6));
end

if ifplot
    figure;
    for cell=1:T.n_elements
        dofIndices = DoFHandler.dofs(cell,:);
        vertices = T.nodes(T.elements(cell,:),:);
        X = vertices(:,1);
        Y = vertices(:,2);
        Z=zeros(3,1);
        for i=1:3
            Z(i)=uh(dofIndices(i));
        end
        patch('XData',X,'YData',Y,'ZData', Z,'FaceVertexCData', Z,...
        'EdgeColor','black','FaceColor','interp','LineWidth',1);% color ...
            by Z coordinate
        view(3);
    end
    title('h=0.025');
end

figure;
loglog(h,E(:,1),'+-',h,E(:,3),'o-',h,E(:,5),'*-');%,h,2*h+5,'--',h,4*h+1,':');
legend('Inf','L2','H1');%,'rate=2','rate=4');
xlabel('h');
ylabel('error');
```

### 2.4.2 *local_assembly.m*

In this file, a new way to assemble local matrix is shown in the following code:

```
function [cell_matrix, local_rhs,area] = ...
    local_assembly(localnodes,f,FE_at_Quad, Quad, p)


mat_B = [(localnodes(2,:) - localnodes(1,:))',(localnodes(3,:) - ...
    localnodes(1,:))']; %[(dim)x(dim)]
det_B = det(mat_B);
inv_B = mat_B\eye(2);

q_points = (mat_B*Quad.xhat' + repmat(localnodes(1,:)',1,Quad.nq))'; % ...
    [nqxdim] list of real quadrature points in cell not ref element
%rhs_vals=f(q_points(:,1),q_points(:,2));
rhs_vals = zeros(Quad.nq,1);
for t=1:Quad.nq
    rhs_vals(t) = f(q_points(t,:));
end

% preallocate space for cell_matrix
cell_matrix = zeros(3,3);
local_rhs = zeros(3,1);
area = abs(det_B);

for q_index = 1:Quad.nq % run through quadrature points on element
```

```
%     grad_phi_at_q_point = [FE_at_Quad.hat_phix1(q_index,:)*inv_B;...
%                             FE_at_Quad.hat_phix2(q_index,:)*inv_B;...
%                             FE_at_Quad.hat_phix3(q_index,:)*inv_B];%3x2
%
%     grad_phi_ij_matrix = grad_phi_at_q_point*grad_phi_at_q_point';  % 3x3

    grad_phi_at_q_point = [FE_at_Quad.hat_phix(q_index,:)',...
                            FE_at_Quad.hat_phiy(q_index,:)']*inv_B;

    grad_phi_ij_matrix = grad_phi_at_q_point*grad_phi_at_q_point';

    cell_matrix = cell_matrix + grad_phi_ij_matrix...
                              * Quad.what(q_index) ...
                              * abs(det_B);

%   cell_matrix = cell_matrix + phi_ij_matrix...
%                             * Quad.what(q_index) ...
%                             * det_B;

    local_rhs = local_rhs + rhs_vals(q_index)...               % f(q_point)
                          * FE_at_Quad.hat_phi(q_index,:)' ... % bases on ...
                            reference element at q_point [(p+1)x1]
                          * Quad.what(q_index)...              % ...
                            quadrature weight
                          * abs(det_B);                              % ...
                            reference area-element transform

end
end
```

### 2.5 Exercise 5

Here, we use the same m-file as that in Exercise 4 and only need to replace the definition of exact functions as follow:

```
u_exact = @(x)(x(1)^2+x(2)^2)^(1/3)*sin(2/3*atan2(x(2),x(1)));
grad_u_exact = @(x) ...
    [2/3*(x(1)^2+x(2)^2)^(-2/3)*(x(1)*sin(2/3*atan2(x(2),x(1)))...
                            -x(2)*cos(2/3*atan2(x(2),x(1))));
                  2/3*(x(1)^2+x(2)^2)^(-2/3)*(x(2)*sin(2/3*atan2(x(2),x(1)))...
                            +x(1)*cos(2/3*atan2(x(2),x(1))))];
% right hand side function
f = @(x)0;
g_D = @(x) u_exact(x);
```

# 3 Results and Analysis

## 3.1 Exercise 1:

In the problem, it asks for plots of 4 meshes above. The plots are as follow:

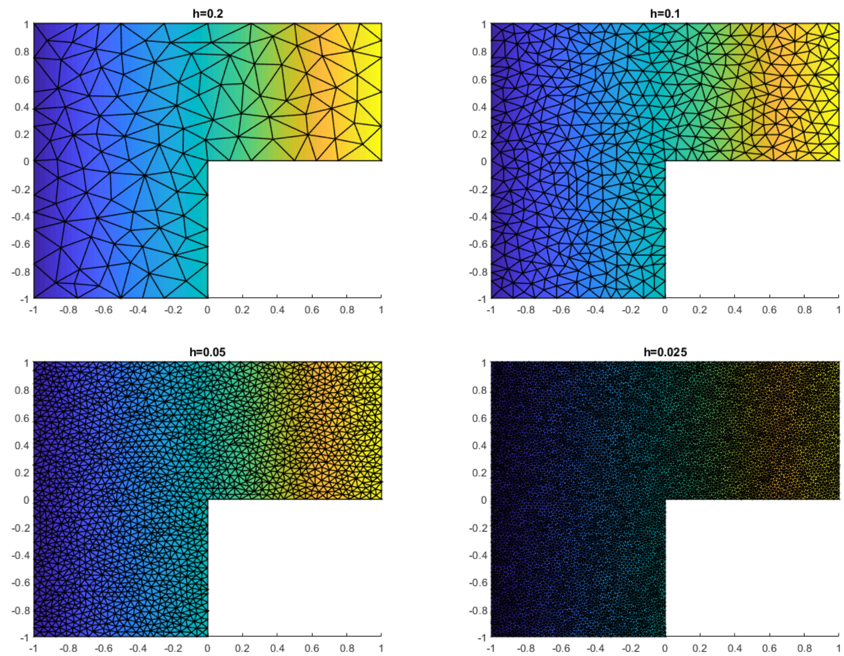


Figure 1: mesh elements

## 3.2 Exercise 2:

In this exercise, we first show the plot of three different functions with quadrature k=3( I will show the reason for k=3 in the following):

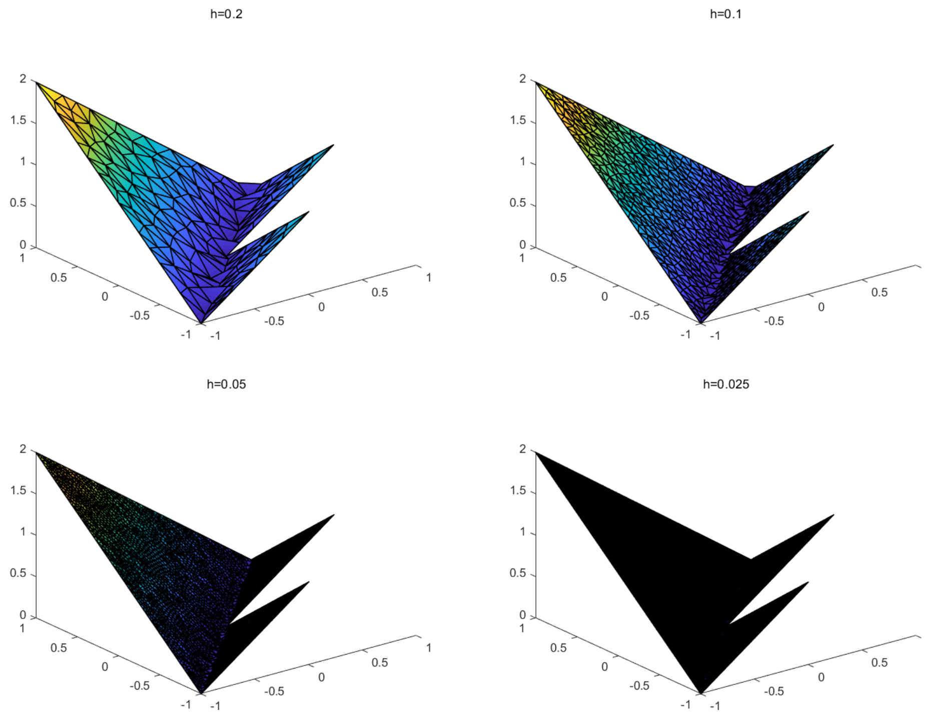


Figure 2: function(i) $u := |x - y|$

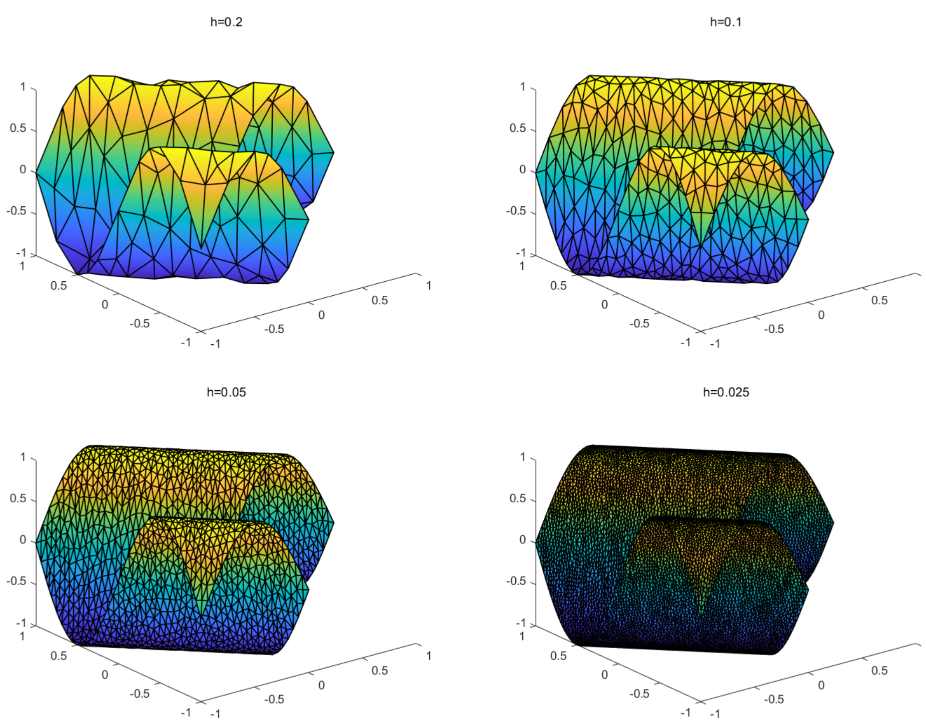


Figure 3: function(ii) $u := \sin(\pi(x + y))$

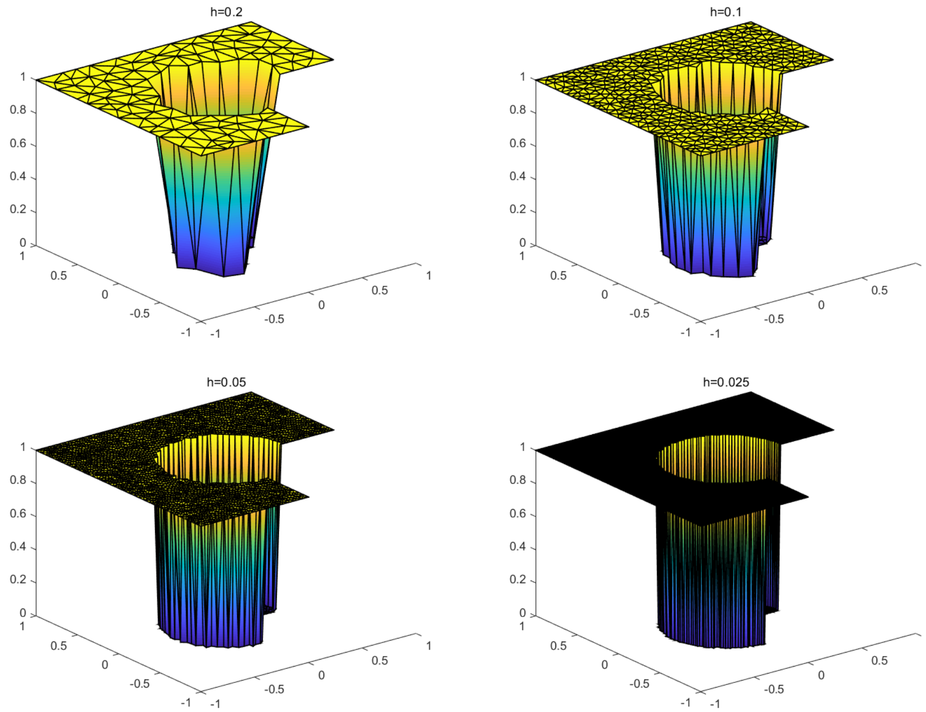


Figure 4: function(iii) $u := 1 \ or \ 0$

In the above code, we have shown the linear basis elements and their derivative in $feEval.m$:

| | $\phi_1$ | $\phi_2$ | $\phi_3$ |
|---|---|---|---|
| Linear elements | 1-x-y | x | y |
| x derivative | -1 | 1 | 0 |
| y derivative | -1 | 0 | 1 |

The table of errors and convergence rates is shown as follow:

| h | $L^2$ error | rate |
|---|---|---|
| 0.2 | 0.0317 | 0 |
| 0.1 | 0.0098 | 3.2252 |
| 0.05 | 0.0036 | 2.6951 |
| 0.025 | 0.0012 | 2.9798 |

Table 2: function(i) $u := |x - y|$

Here, we define the rate to be $errorrate := \frac{||u - I_{h_{i-1}}u||_{L^2}}{||u - I_{h_i}u||_{L^2}}$.

| h | $L^2$ error | rate |
|---|---|---|
| 0.2000 | 0.0576 | 0 |
| 0.1000 | 0.0151 | 3.8088 |
| 0.0500 | 0.0037 | 4.1002 |
| 0.0250 | 0.0009 | 4.2167 |

Table 3: function(ii) $u := \sin(\pi(x+y))$

| h | $L^2$ error | rate |
|---|---|---|
| 0.2000 | 0.2363 | 0 |
| 0.1000 | 0.1728 | 1.3675 |
| 0.0500 | 0.1156 | 1.4945 |
| 0.0250 | 0.0815 | 1.4190 |

Table 4: function(iii) $u := 0 \; or \; 1$

From the above table, we find that errors comes down with the decreasing of mesh sizes h and all the 3 functions with converge rate larger than one, which means that Lagrange Interpolation converges to the function above.

We can show all the rates of three functions in the following table:

| h | func(i) | func(ii) | func(iii) |
|---|---|---|---|
| 0.2000 | 0 | 0 | 0 |
| 0.1000 | 3.2252 | 3.8088 | 1.3675 |
| 0.0500 | 2.6951 | 4.1002 | 1.4945 |
| 0.0250 | 2.9798 | 4.2167 | 1.4190 |

Table 5: Smoothness and Convergence rate

We have known the smoothness relationship: function(ii)>function(i)>function(iii). Here'>'mean 'more smooth than'. And from the above table, we can learn the relationship of convergence rate: function(ii)>function(i)>function(iii) which is accord with the smoothness relationship. Finally, we can show the finding: the more smooth a function is, the larger convergence rates the $L^2$ norm will get.

In the end part, the reason for choosing $k = 3$ is shown as follow:

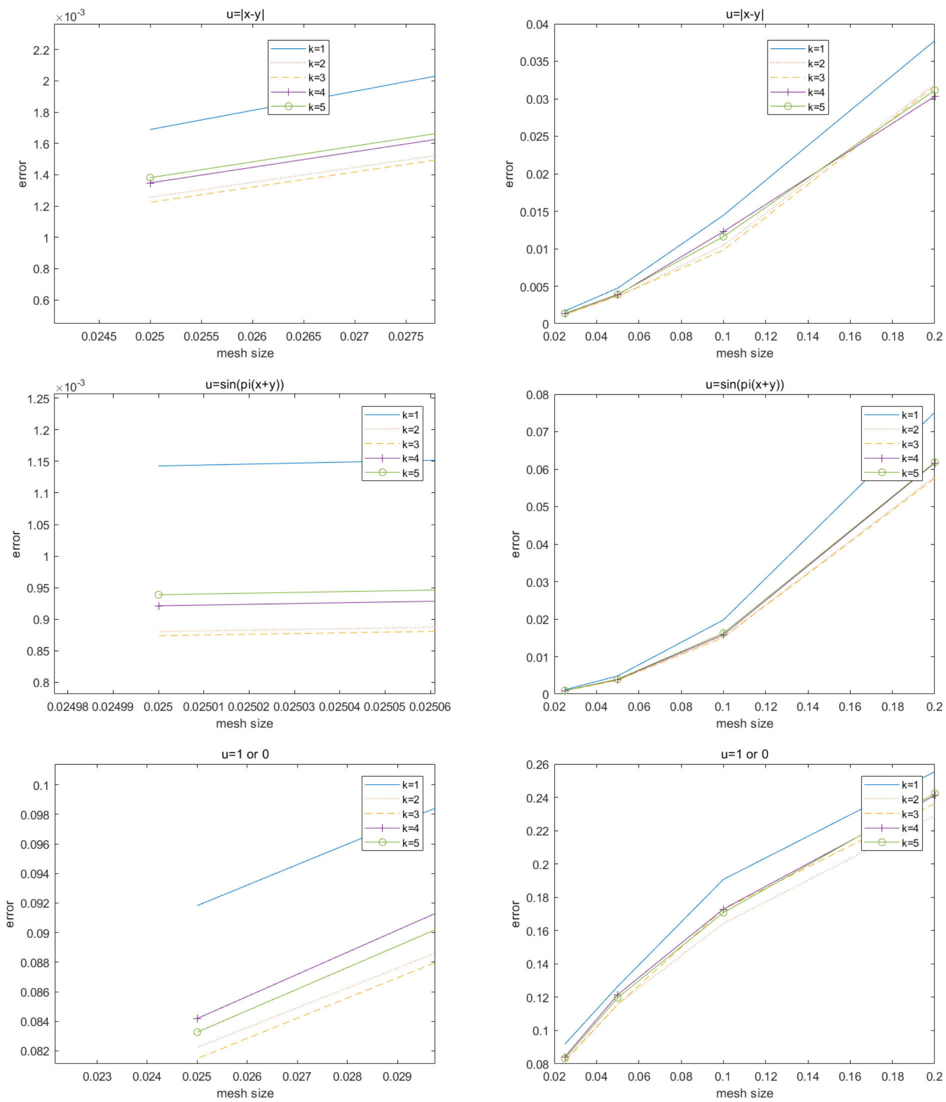


Figure 5: Quadrature to choose

The left row is the enlarge plot of the right row at $h = 0.025$. From the above plots, we can learn that when $k = 3$, all the 3 functions get the smallest errors, namely the largest accuracy, and the highest slop, namely the largest rates. According to accuracy and convergence rates, we choose $k = 3$.

### 3.3 Exercise 3:

In this part, results of $u_h$ will be shown in the following plots:

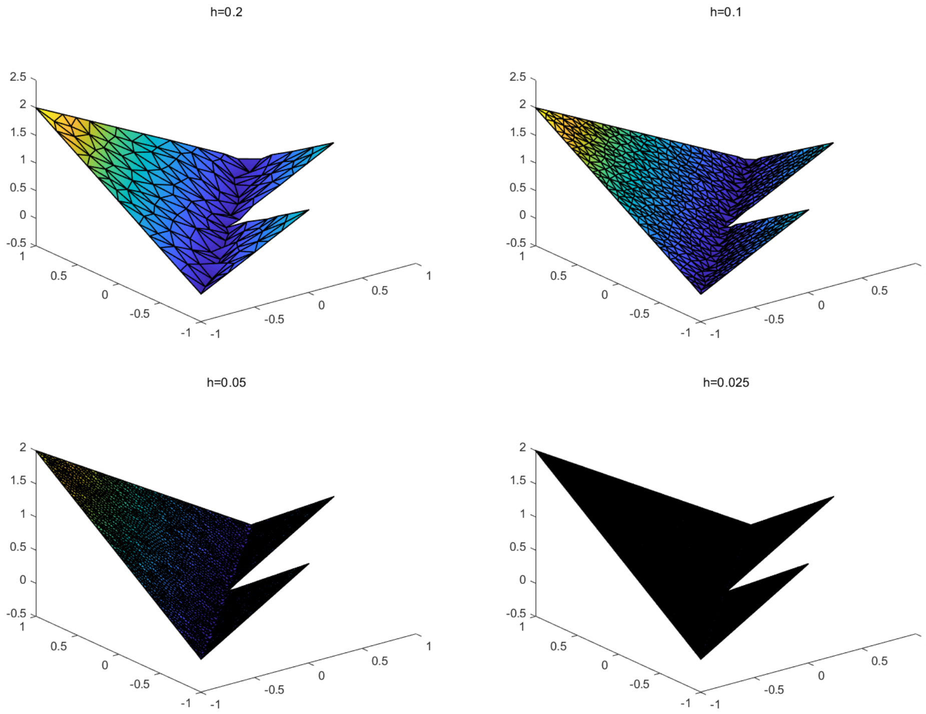


Figure 6: function(i)projection $u := |x - y|$

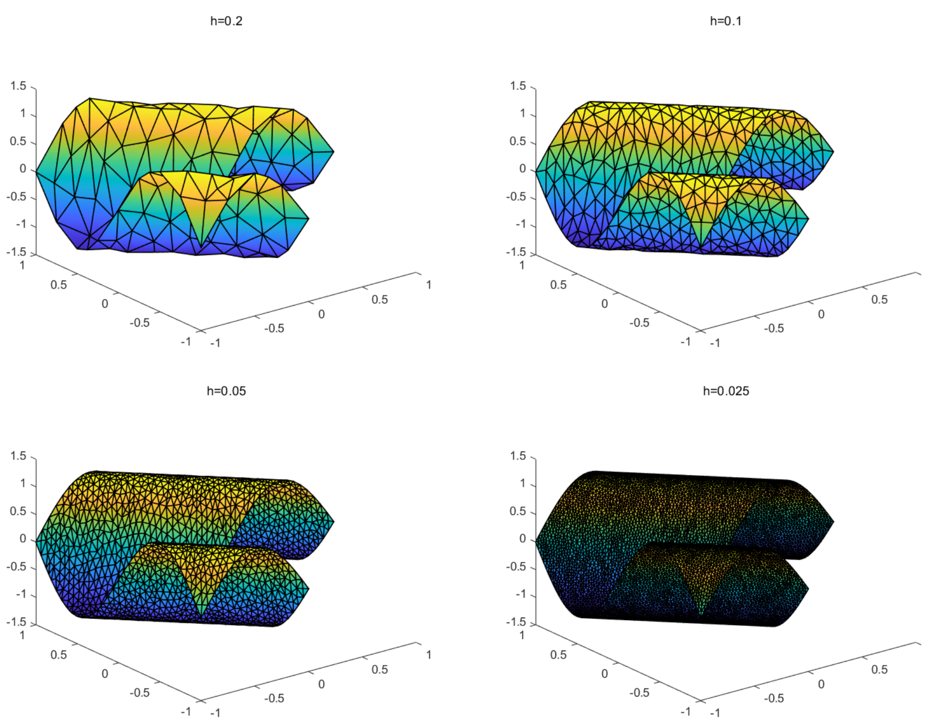


Figure 7: function(ii)projection $u := \sin(\pi(x + y))$

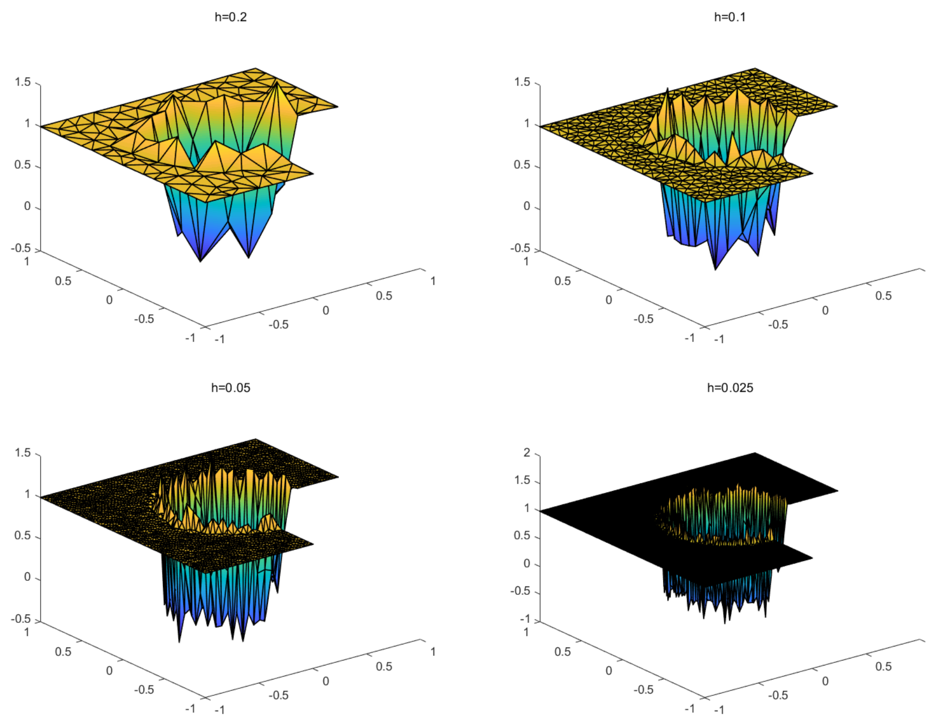


Figure 8: function(iii)projection $u := 1$ *or* $0$

The table of errors and convergence rates is shown as follow:

| h | $L^\infty$error | $L^\infty$rate | $L^2$error | $L^2$rate | $H^1$error | $H^1$rate |
|---|---|---|---|---|---|---|
| 0.200000 | 0.088067 | 0.000000 | 0.018525 | 0.000000 | 0.852758 | 0.000000 |
| 0.100000 | 0.054943 | 1.602890 | 0.005901 | 3.139484 | 0.585915 | 1.455430 |
| 0.050000 | 0.026496 | 2.073632 | 0.002279 | 2.589217 | 0.432974 | 1.353233 |
| 0.025000 | 0.013294 | 1.993067 | 0.000748 | 3.047071 | 0.294875 | 1.468333 |

Table 6: function(i) $u := |x - y|$

| h | $L^\infty$error | $L^\infty$rate | $L^2$error | $L^2$rate | $H^1$error | $H^1$rate |
|---|---|---|---|---|---|---|
| 0.200000 | 0.090707 | 0.000000 | 0.018489 | 0.000000 | 1.213229 | 0.000000 |
| 0.100000 | 0.028660 | 3.164972 | 0.004369 | 4.231449 | 0.613977 | 1.976017 |
| 0.050000 | 0.007393 | 3.876808 | 0.001091 | 4.006347 | 0.300825 | 2.040979 |
| 0.025000 | 0.002033 | 3.636147 | 0.000256 | 4.262422 | 0.145780 | 2.063559 |

Table 7: function(ii) $u := \sin(\pi(x + y))$

| h | $L^\infty$error | $L^\infty$rate | $L^2$error | $L^2$rate | $H^1$error | $H^1$rate |
|---|---|---|---|---|---|---|
| 0.200000 | 0.766479 | 0.000000 | 0.160636 | 0.000000 | 4.990635 | 0.000000 |
| 0.100000 | 0.907637 | 0.844477 | 0.124621 | 1.288998 | 6.948984 | 0.718182 |
| 0.050000 | 0.947707 | 0.957719 | 0.085404 | 1.459191 | 9.536437 | 0.728677 |
| 0.025000 | 0.975872 | 0.971138 | 0.059333 | 1.439407 | 13.907986 | 0.685681 |

Table 8: function(iii) $u := 0\ or\ 1$

### 3.4 Exercise 4

Solution on the fourth mesh is shown in the following plot:

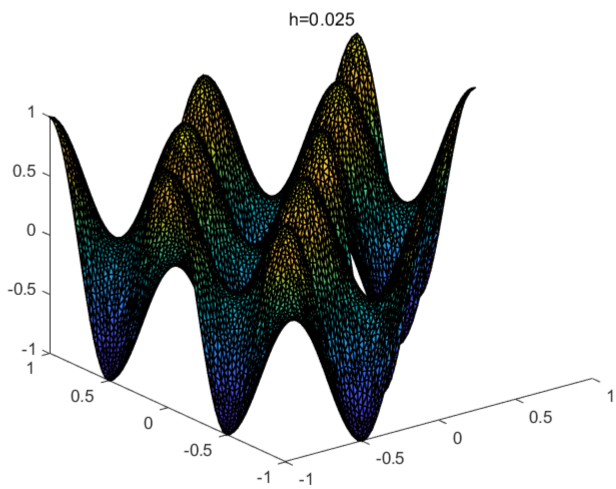


Figure 9: Solution on the fourth mesh

The errors are in the following table:

| h | $L^\infty$error | $L^\infty$rate | $L^2$error | $L^2$rate | $H^1$error | $H^1$rate |
|---|---|---|---|---|---|---|
| 0.200000 | 0.276922 | 0.000000 | 0.185416 | 0.000000 | 3.296411 | 0.000000 |
| 0.100000 | 0.086160 | 3.214031 | 0.047493 | 3.904099 | 1.631578 | 2.020382 |
| 0.050000 | 0.026214 | 3.286850 | 0.012288 | 3.865089 | 0.816495 | 1.998270 |
| 0.025000 | 0.006685 | 3.920983 | 0.003025 | 4.061902 | 0.407380 | 2.004261 |

Table 9: Errors and rates

The plot of errors is as follow:

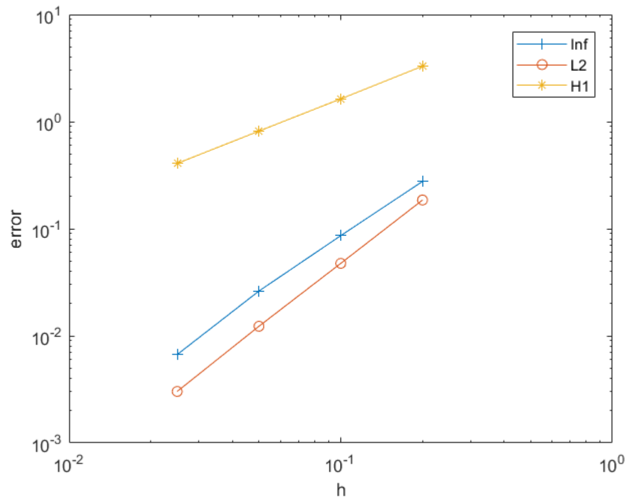


Figure 10: Errors against h

### 3.5 Exercise 5

Here, the errors in a table with rates are as follow:

| h | $L^\infty$error | $L^\infty$rate | $L^2$error | $L^2$rate | $H^1$error | $H^1$rate |
|---|---|---|---|---|---|---|
| 0.200000 | 1.580154 | 0.000000 | 0.348605 | 0.000000 | 1.669226 | 0.000000 |
| 0.100000 | 1.474614 | 1.071571 | 0.306806 | 1.136239 | 1.821625 | 0.916339 |
| 0.050000 | 1.473371 | 1.000844 | 0.293445 | 1.045532 | 2.291479 | 0.794956 |
| 0.025000 | 1.489071 | 0.989457 | 0.293296 | 1.000508 | 2.370845 | 0.966524 |

Table 10: Errors and rates

The plot of errors is as follow:

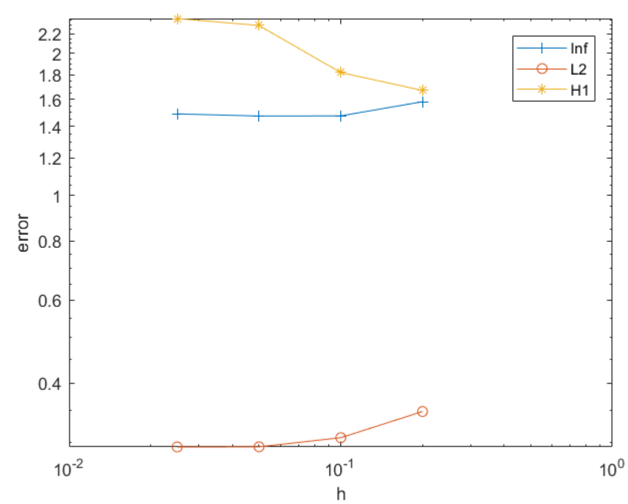


Figure 11: Errors against h

From the rate of $H^1$ which is approximate of 1, and $u \in H^{k+1}$ with $k = 0$, we can see that $H^1$ norm is not bounded by $h$ (since h is the relationship of 2 multiple). This suit with the inequality we learn on class that:

$$||u - I_h u||_{H^1(T)} \leq Ch_T^k |u|_{H^{k+1}(T)},$$

here $k = 0$.