Finite Elements for Fluids

Assignment – HDG (#1)

Author: Cristina Garcia Albela
MsC in Computational Mechanics

1. Problem statement

Consider the domain $\Omega = [0,1]^2$ such that $\partial\Omega = \Gamma_D \cup \Gamma_N \cup \Gamma_R$ with $\Gamma_D \cap \Gamma_R = \emptyset$, $\Gamma_D \cap \Gamma_N = \emptyset$ and $\Gamma_N \cap \Gamma_R = \emptyset$. More precisely, set

$$\Gamma_N := \{(x,y) \in \mathbb{R}^2 : x = 0\},$$

$$\Gamma_R := \{(x,y) \in \mathbb{R}^2 : y = 1\},$$

$$\Gamma_N := \partial\Omega \backslash (\Gamma_N \cup \Gamma_R)$$

The following second -order linear scalar partial differential equation is defined

$$\begin{cases} -\nabla \cdot (\kappa \nabla u) = s & in\,\Omega \\ u = u_D & on\,\Gamma_D \\ \boldsymbol{n} \cdot (\kappa \nabla u) = t & on\,\Gamma_N \\ \boldsymbol{n} \cdot (\kappa \nabla u) + \gamma u = g & on\,\Gamma_R \end{cases}$$

Where κ and γ are de diffusion and convection coefficients respectively, $\boldsymbol{n}$ is the outward unit normal vector to the boundary, s is the volumetric source term and u_D, t and g are the Dirichlet, Neumann and Robin data imposed on the corresponding portions of the boundary $\partial\Omega$.

An equivalent strong form of the 2nd order elliptic equations problem can be written in the broken computational domain as

$$\begin{cases} -\nabla \cdot (\kappa \nabla u) = s & in\,\Omega_i, for\; i = 1,2 \ldots., n\_el \\ u = u_D & on\,\Gamma_D \\ \boldsymbol{n} \cdot (\kappa \nabla u) = t & on\,\Gamma_N \\ \boldsymbol{n} \cdot (\kappa \nabla u) + \gamma u = g & on\,\Gamma_R \\ [\![u\boldsymbol{n}]\!] = \boldsymbol{0} & on\,\Gamma \\ [\![\boldsymbol{n} \cdot \kappa \nabla u]\!] = 0 & on\,\Gamma \end{cases}$$

where the two last equations represent the imposition of continuity of u and the normal fluxes along the internal interfaces between the elements.

Finally, the strong form is written in mixed form, as a system of first order equations over the broken domain

$$\begin{cases} \nabla \cdot \boldsymbol{q} = s & in\,\Omega_i \\ \boldsymbol{q} + \nabla u = \boldsymbol{0} & in\,\Omega_i \\ u = u_D & on\,\Gamma_D \\ \boldsymbol{n} \cdot q = -t & on\,\Gamma_N \\ \boldsymbol{n} \cdot \boldsymbol{q} - \gamma u = -g & on\,\Gamma_R \\ [\![u\boldsymbol{n}]\!] = \boldsymbol{0} & on\,\Gamma \\ [\![\boldsymbol{n} \cdot \mathbf{q}]\!] = 0 & on\,\Gamma \end{cases}$$

2. HDG strong and weak forms

2.1 Strong form

Working with HDG the problem should be rewrite as two equivalent problems, a local element-by-element problem in which $\hat{u}$ hat is introduced and a second global problem in which $\hat{u}$ is determined.

Starting with the local problem, the strong form of the problem can be written as

$$\begin{cases} \nabla \cdot \boldsymbol{q_i} = s & in\ \Omega_i \\ \boldsymbol{q} + \nabla u_i = \boldsymbol{0} & in\ \Omega_i \\ u_i = u_D & on\ \partial\Omega_i \cap \Gamma_D \quad for\ i = 1, \ldots .n_{el} \\ u_i = \hat{u} & on\ \partial\Omega_i \backslash \partial\Omega \end{cases}$$

From it, an element-by-element solution for q_i and u_i is obtained as a function of the unknown $\hat{u}$.

Then, the global problem is defined to determine $\hat{u}$, that corresponds to the imposition of the transmission conditions

$$\begin{cases} [\![u\boldsymbol{n}]\!] = \boldsymbol{0} & on\ \Gamma \\ [\![\boldsymbol{n} \cdot \mathbf{q}]\!] = 0 & on\ \Gamma \\ \boldsymbol{n_i} \cdot \boldsymbol{q_i} = -t & on\ \partial\Omega_i \cap \Gamma_N \\ \boldsymbol{n_i} \cdot \boldsymbol{q_i} - \gamma u_i = -g & on\ \partial\Omega_i \cap \Gamma_R \end{cases}$$

As $u = \hat{u}$ on Γ is imposed by the local problem, the continuity of the primal variable $[\![\hat{u}\boldsymbol{n}]\!] = \boldsymbol{0}$ is automatically imposed because $\hat{u}$ is unique for adjacent elements, so that the transmission conditions can be rewritten just as

$$\begin{cases} [\![\boldsymbol{n} \cdot \mathbf{q}]\!] = 0 & on\ \Gamma \\ \boldsymbol{n_i} \cdot \boldsymbol{q_i} = -t & on\ \partial\Omega_i \cap \Gamma_N \\ \boldsymbol{n_i} \cdot \boldsymbol{q_i} - \gamma \hat{u} = -g & on\ \partial\Omega_i \cap \Gamma_R \end{cases}$$

2.2 Weak forms

Starting with the local problem, the weak formulation for each of the elements can are calculated, introducing the weighted functions v and $\boldsymbol{w}$.

- First equation ($\nabla \cdot \boldsymbol{q_i} = s$)

$$(v, (\nabla \cdot \boldsymbol{q_i}))_{\Omega_i} = (v, s)_{\Omega_i}$$

$$applying\ divergence\ theorem\ \left(\nabla \cdot (v\boldsymbol{q_i})\right)_{\Omega_i} = (v(\boldsymbol{n_i} \cdot \boldsymbol{\hat{q}_i}))_{\partial\Omega_i}$$
$$= (\boldsymbol{\nabla} v \cdot \boldsymbol{q_i})_{\Omega_i} + (v \cdot \boldsymbol{\nabla q_i})_{\Omega_i}$$

$$-(\boldsymbol{\nabla} v \cdot \boldsymbol{q_i})_{\Omega_i} + \langle v, (\boldsymbol{n_i} \cdot \boldsymbol{\hat{q}_i}) \rangle_{\partial\Omega_i} = (v, s)_{\Omega_i}$$

- Second equation ($\boldsymbol{q} + \nabla u_i = \boldsymbol{0}$)

$$(\boldsymbol{w}, \boldsymbol{q_i})_{\Omega_i} + (\boldsymbol{w}, \nabla u_i)_{\Omega_i} = \boldsymbol{0}$$

$$applying\ divergence\ theorem\ \left(\nabla \cdot (\boldsymbol{w} u_i)\right)_{\Omega_i} = ((\boldsymbol{n_i} \cdot \boldsymbol{w})u_i)_{\partial\Omega_i}$$
$$= \left((\boldsymbol{\nabla} \cdot \boldsymbol{w})u_i\right)_{\Omega_i} + (\boldsymbol{w}, \nabla u_i)_{\Omega_i}$$

$$(\boldsymbol{w}, \boldsymbol{q_i})_{\Omega_i} - \left((\boldsymbol{\nabla} \cdot \boldsymbol{w}), u_i\right)_{\Omega_i} = -\langle (\boldsymbol{n_i} \cdot \boldsymbol{w}) u_D \rangle_{\partial\Omega_i \cap \Gamma_D} - \langle (\boldsymbol{n_i} \cdot \boldsymbol{w}) \hat{u} \rangle_{\partial\Omega_i \backslash \Gamma_D}$$

Introducing the numerical traces of the fluxes, that are defined element-by-element as

$$\boldsymbol{n_i} \cdot \boldsymbol{\hat{q}_i} := \begin{cases} \boldsymbol{n_i} \cdot \boldsymbol{q_i} + \tau_i(u_i - u_D) & on\ \partial\Omega_i \cap \Gamma_D \\ \boldsymbol{n_i} \cdot \boldsymbol{q_i} + \tau_i(u_i - \hat{u}) & on\ \partial\Omega_i \cap \Gamma \end{cases}$$

The weak formulation for each element can be written defined finally as, given u_D on Γ_D and $\hat{u}$ on $\Gamma \cup \Gamma_N$ and on $\Gamma \cup \Gamma_R$, find $(\boldsymbol{q_i}, u_i)$ that satisfies

$$-(\nabla v \cdot \boldsymbol{q_i})_{\Omega_i} + \langle v, (\boldsymbol{n_i} \cdot \boldsymbol{q_i})\rangle_{\partial\Omega_i} + \langle v, \tau_i u_i \rangle_{\partial\Omega_i} = (v, s)_{\Omega_i} + \langle v, \tau_i u_i \rangle_{\partial\Omega_i \cap \Gamma_D} + \langle v, \tau_i \hat{u} \rangle_{\partial\Omega_i \setminus \Gamma_D}$$

$$-(\boldsymbol{w}, \boldsymbol{q_i})_{\Omega_i} + \big((\nabla \cdot \boldsymbol{w}), u_i\big)_{\Omega_i} = \langle (\boldsymbol{n_i} \cdot \boldsymbol{w}) u_D \rangle_{\partial\Omega_i \cap \Gamma_D} + \langle (\boldsymbol{n_i} \cdot \boldsymbol{w}) \hat{u} \rangle_{\partial\Omega_i \setminus \Gamma_D}$$

For the global problem, being the weighted function μ, the weak form is defined as find $\hat{u}$ for all μ such that

$$\sum_{i=1}^{n_{el}} \langle \mu, \boldsymbol{n_i} \cdot \boldsymbol{\hat{q}_i} \rangle_{\partial\Omega_i \setminus \partial\Omega} + \sum_{i=1}^{n_{el}} \langle \mu, (\boldsymbol{n_i} \cdot \boldsymbol{\hat{q}_i} + t) \rangle_{\partial\Omega_i \cap \Gamma_N} + \sum_{i=1}^{n_{el}} \langle \mu, (\boldsymbol{n_i} \cdot \boldsymbol{\hat{q}_i} - \gamma \hat{u} + g) \rangle_{\partial\Omega_i \cap \Gamma_N} = 0$$

Applying the numerical traces of the fluxes

$$\sum_{i=1}^{n_{el}} \big(\langle \mu, \tau_i u_i \rangle_{\partial\Omega_i \setminus \Gamma_D} + \langle \mu, \boldsymbol{n_i} \cdot \boldsymbol{q_i} \rangle_{\partial\Omega_i \setminus \Gamma_D} - \langle \mu, \tau_i \hat{u} \rangle_{\partial\Omega_i \setminus \Gamma_D} - \langle \mu, \gamma \hat{u} \rangle_{\partial\Omega_i \setminus \Gamma_R} \big)$$
$$= -\sum_{i=1}^{n_{el}} \big(\langle \mu, t \rangle_{\partial\Omega_i \cap \Gamma_N} + \langle \mu, g \rangle_{\partial\Omega_i \cap \Gamma_R} \big)$$

3. Analytical expressions

Previously to the implementation the analytical expressions for the terms u_D, t and g should be derived as they are going to be added to be part of the new calculation that the code news passing from just Dirichlet conditions to the actual problem description with Neumann and Robin.
Having the definition of $u(x, y)$ as following, Matlab tools for derivatives will be use in order to achieve the final expressions.

$$u(x, y) = \exp(\kappa \sin(ax + by) + \gamma \cos(cx + dy))$$

Dirichlet boundary

For Dirichlet boundary value u_D no mathematical modifications of the previous equation are needed. The point will be in apply $u(x, y)$ definition in just those points that are part of Γ_D (see in the following section how the different boundaries are recognised)

```
function u = analyticalPoisson(X)

% Parameters
k = 0.1; gamma = 0.3;
a = 5.1; b= -6.2; c = 4.3; d = 3.4;
% Points
x = X(:,1);
y = X(:,2);
% Solution
u = exp(k*sin(a*x+b*y) + *cos(c*x+d*y));
```

Neumann & Robin boundaries

In order to obtain Neumann and Robin boundary expressions, as we need to derivate u function, Matlab tools are going to be used. It is important to take into accounta that Neumann is just defined

in the vertical line $x = 0$, with normal vector $n = [-1, 0]$, whereas Robin is defined along $y = 1$ so that $n = [0, 1]$.
With help of Matlab tools (for derivations), the corresponding functions are easily obtained together with the source term, to be then implemented in the HDG code.

```
% Parameters
k= 0.1; gamma = 0.3; a = 5.1; b= -6.2; c = 4.3; d = 3.4;
% Points
x = X(:,1);
y = X(:,2);

t = (-
exp(conj(k).*sin(conj(b).*conj(y)+conj(a).*conj(x))+cos(conj(c).*conj(x)+conj(d).*conj(y)).*conj(ga
mma)).*conj(k).*(cos(conj(b).*conj(y)+conj(a).*conj(x)).*conj(a).*conj(k)-
conj(c).*conj(gamma).*sin(conj(c).*conj(x)+conj(d).*conj(y))))*(1/k);

g =
(gamma.*exp(gamma.*cos(c.*x+d.*y)+k.*sin(a.*x+b.*y))+exp(conj(k).*sin(conj(b).*conj(y)+conj(a).*con
j(x))+cos(conj(c).*conj(x)+conj(d).*conj(y)).*conj(gamma)).*conj(k).*(cos(conj(b).*conj(y)+conj(a).
*conj(x)).*conj(b).*conj(k)-conj(d).*conj(gamma).*sin(conj(c).*conj(x)+conj(d).*conj(y))))*  (1/k);

s =
mu*(1/k)*(k.*(exp(gamma.*cos(c.*x+d.*y)+k.*sin(a.*x+b.*y)).*(c.^2.*gamma.*cos(c.*x+d.*y)+a.^2.*k.*s
in(a.*x+b.*y))+exp(gamma.*cos(c.*x+d.*y)+k.*sin(a.*x+b.*y)).*(d.^2.*gamma.*cos(c.*x+d.*y)+b.^2.*k.*
sin(a.*x+b.*y))-exp(gamma.*cos(c.*x+d.*y)+k.*sin(a.*x+b.*y)).*(a.*k.*cos(a.*x+b.*y)-
c.*gamma.*sin(c.*x+d.*y)).^2-exp(gamma.*cos(c.*x+d.*y)+k.*sin(a.*x+b.*y)).*(b.*k.*cos(a.*x+b.*y)-
d.*gamma.*sin(c.*x+d.*y)).^2))
```

So that, the expression can be written as

$$t = \kappa\big((\kappa a\cos(ax+by) - \gamma c\sin(cx+dy))\exp(\kappa\sin(ax+by) + \gamma\cos(cx+dy))\big)$$

$$\begin{aligned} g = \kappa\big((\kappa b\cos(ax+by) - \gamma d\sin(cx+dy))\exp(\kappa\sin(ax+by) + \gamma\cos(cx+dy))\big) \\ + \gamma\exp(\kappa\sin(ax+by) + \gamma\cos(cx+dy)) \end{aligned}$$

4. Matlab code implementation

Once the discrete versions of the previous weak forms are determined, a matrix system of equations can be defined for each, local and global problems.

Starting with the local problem, the following system of equation will be solved at each element

$$\begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{uq} \\ \mathbf{A}_{uq}^T & \mathbf{A}_{qq} \end{bmatrix}_i \begin{Bmatrix} \mathbf{u}_i \\ \mathbf{q}_i \end{Bmatrix} = \begin{Bmatrix} \mathbf{f}_u \\ \mathbf{f}_q \end{Bmatrix}_i + \begin{bmatrix} \mathbf{A}_{u\hat{u}} \\ \mathbf{A}_{q\hat{u}} \end{bmatrix}_i \hat{\mathbf{u}}_i.$$

remark the fact that no modifications will be needed in this part with respect to the previous implementation.
Similarity, for the global problem

$$\sum_{i=1}^{\mathtt{n_{el}}} \Big\{ \begin{bmatrix} \mathbf{A}_{u\hat{u}}^T & \mathbf{A}_{q\hat{u}}^T \end{bmatrix}_i \begin{Bmatrix} \mathbf{u}_i \\ \mathbf{q}_i \end{Bmatrix} + [\mathbf{A}_{\hat{u}\hat{u}}]_i\, \hat{\mathbf{u}}_i \Big\} = \sum_{i=1}^{\mathtt{n_{el}}} [\mathbf{f}_{\hat{u}}]_i$$

Here, a modification must be done in the $\mathbf{A}_{\hat{\mathbf{u}}\hat{\mathbf{u}}}$ component, due to the contribution of Robin boundary condition, as well as in $\mathbf{f}_{\hat{\mathbf{u}}}$ as consequences of t and g fluxes in the Neumann and Robin respectably.

Once the solution of the local problem is obtained, it should be introduced in the global form, that finally becomes

$$\widehat{\mathbf{K}}\,\hat{\mathbf{u}} = \hat{\mathbf{f}}$$

Where $\widehat{\mathbf{K}}$ and $\hat{\mathbf{f}}$ are defined as

$$\widehat{\mathbf{K}} = \mathop{\mathsf{A}}_{i=1}^{\mathtt{n_{el}}} \begin{bmatrix} \mathbf{A}_{u\hat{u}}^T & \mathbf{A}_{q\hat{u}}^T \end{bmatrix}_i \begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{uq} \\ \mathbf{A}_{uq}^T & \mathbf{A}_{qq} \end{bmatrix}_i^{-1} \begin{bmatrix} \mathbf{A}_{u\hat{u}} \\ \mathbf{A}_{q\hat{u}} \end{bmatrix}_i + [\mathbf{A}_{\hat{u}\hat{u}}]_i$$

$$\hat{\mathbf{f}} = \mathop{\mathsf{A}}_{i=1}^{\mathtt{n_{el}}} [\mathbf{f}_{\hat{u}}]_i - \begin{bmatrix} \mathbf{A}_{u\hat{u}}^T & \mathbf{A}_{q\hat{u}}^T \end{bmatrix}_i \begin{bmatrix} \mathbf{A}_{uu} & \mathbf{A}_{uq} \\ \mathbf{A}_{uq}^T & \mathbf{A}_{qq} \end{bmatrix}_i^{-1} \begin{Bmatrix} \mathbf{f}_u \\ \mathbf{f}_q \end{Bmatrix}_i$$

4.1 Matlab code. Boundary conditions

Starting from a code that is designed for HDG problems where just Dirichlet boundary conditions are defined, some modifications must be done in order to introduce the new boundary conditions state, where Neumann and Robin BC will take part.
Looking at the boundary domain definition, it is clear in which zones are each of them defined, as well as that no points are shared between any of them. Thus, the first step will be to introduce a modification in the code that allows it to check between the external elements, to which boundary condition they must be placed.
Working on “GetFaces.m” function, the changes are implemented once the faces have been classified as internal or external. For each external face, with help of the connectivity matrix it is check their boundary face belong to Dirichlect, Neumann or Robin boundary.

```
function [intFaces,extFace_D,extFace_N,extFace_R] = GetFaces(X,T)

…

intFaces = intFaces(intFaces(:,1)~=0,:);
extFaces = extFaces(extFaces(:,1)~=0,:);

%%BOUNDARIES

ii = size(extFaces)
in = 1; im = 1; io = 1;

for i = 1:ii

    ele_Faces = extFaces(i,1);
    node1 = T(ele_Faces,1);
    node2 = T(ele_Faces,2);
    node3 = T(ele_Faces,3);

    n1_x = X(node1,1);
    n2_x = X(node2,1);
    n3_x = X(node3,1);

    n1_y = X(node1,2);
    n2_y = X(node2,2);
    n3_y = X(node3,2);

    if n1_x == 0 && n2_x == 0
        extFace_N(in,1) = extFaces(i,1);
        extFace_N(in,2) = extFaces(i,2);
        in = in +1;
    elseif n1_x == 0 && n3_x == 0
        extFace_N(in,1) = extFaces(i,1);
        extFace_N(in,2) = extFaces(i,2);
        in = in +1;
    elseif n3_x == 0 && n2_x == 0
        extFace_N(in,1) = extFaces(i,1);
        extFace_N(in,2) = extFaces(i,2);
        in = in +1;
    elseif n1_y == 1 && n2_y == 1
        extFace_R(im,1) = extFaces(i,1);
        extFace_R(im,2) = extFaces(i,2);
        im = im +1;
    elseif n1_y == 1 && n3_y == 1
        extFace_R(im,1) = extFaces(i,1);
        extFace_R(im,2) = extFaces(i,2);
        im = im +1;
    elseif n3_y == 1 && n2_y == 1
        extFace_R(im,1) = extFaces(i,1);
        extFace_R(im,2) = extFaces(i,2);
        im = im +1;
    else
        extFace_D(io,1) = extFaces(i,1);
        extFace_D(io,2) = extFaces(i,2);
        io = io +1;
    end
end
end
```

Once the boundaries are defined, the elements are distribution inside the F matrix must be reorganised, according the code structure in which the Dirichlet Boundaries should occupied the las positions. To do that, some modifications are introduced in "hgd_preprocess.m" function as follows

```
function [F infoFaces] = hdg_preprocess(X,T)

% create infoFaces
[intFaces,extFace_D,extFace_N,extFace_R] = GetFaces(X,T(:,1:3));

nOfElements = size(T,1);
nOfInteriorFaces = size(intFaces,1);
nOfExteriorFaces_N = size(extFace_N,1);
nOfExteriorFaces_R = size(extFace_R,1);
nOfExteriorFaces_D = size(extFace_D,1);

F = zeros(nOfElements,3);
for iFace = 1:nOfInteriorFaces
    infoFace = intFaces(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace;
    F(infoFace(3),infoFace(4)) = iFace;
end
for iFace = 1:nOfExteriorFaces_N
    infoFace = extFace_N(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace + nOfInteriorFaces;
end
for iFace = 1:nOfExteriorFaces_R
    infoFace = extFace_R(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace + nOfInteriorFaces +
nOfExteriorFaces_N;
end
for iFace = 1:nOfExteriorFaces_D
    infoFace = extFace_D(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace + nOfInteriorFaces +
nOfExteriorFaces_N + nOfExteriorFaces_N;
end

infoFaces.intFaces = intFaces;
infoFaces.extFace_D = extFace_D;
infoFaces.extFace_N = extFace_N;
infoFaces.extFace_R = extFace_R;
```

Finally, the last change with respect to this topic will be done in "mainPoissonHDG.m" function, to redefine the dof for Dirichlet and the dof of the unknows according to the new configuration

```
%Dirichlet BC
%Dirichlet face nodal coordinates
nOfFaceNodes = degree+1;
nOfInteriorFaces = size(infoFaces.intFaces,1);
nOfExteriorFaces = size(infoFaces.extFaces,1);
nOfExteriorFaces_B = size(infoFaces.extFaces,1) -
size(infoFaces.extFace_N,1) - size(infoFaces.extFace_R,1);
nOfExteriorFaces_D = size(infoFaces.extFace_D,1);


uDirichlet =
computeProjectionFaces(@analyticalPoisson,infoFaces.extFace_D,X,T,referenc
eElement);
dofDirichlet= (nOfInteriorFaces + nOfExteriorFaces_B) *nOfFaceNodes +
(1:nOfExteriorFaces_D*nOfFaceNodes);
%dofUnknown = 1:nOfInteriorFaces*nOfFaceNodes;
dofUnknown = 1:(nOfInteriorFaces + nOfExteriorFaces_B)*nOfFaceNodes ;
```

4.2 Matlab code. Elemental Matrices

Once the external faces have been classified, is time to introduce the appropriate changes in the matrix system components, having in mind the already defined matrix system of equations. It has been decided to introduce three new functions to compute the elemental matrices, one for Neumann boundary element, one for Robin boundary element and the last one for that case in which an element has faces in both boundaries, maintaining the original function as well for the rest.

The new term introduced to Neumann and Robin to $\mathbf{f}_{\hat{u}}$ is defined as "fn" and computed just in the element face that is part of one of the boundaries. In the case of $\mathbf{A}_{\hat{u}\hat{u}}$ parameter modifications, they are introduced just one the element face is part of the Robin boundary

```
%%NEUMAN BOUNDARY
function [Q,U,Qf,Uf,Alq,Alu,All,f_n] =
KKeElementalMatricesIsoParametric_N(mu,Xe,Te,referenceElement,tau,infoFaces,iElem,pos
_N)
...

%Identify the nodes of the face in the Bounday
FACE = infoFaces.extFace_N(pos_N,2);
FACE_NODE = faceNodes(FACE,:);

...
...

%% Faces computations
Alq = zeros(3*nOfFaceNodes,2*nOfElementNodes);
Auu = zeros(nOfElementNodes,nOfElementNodes);
Alu = zeros(3*nOfFaceNodes,nOfElementNodes);
All = zeros(3*nOfFaceNodes,3*nOfFaceNodes);
fn = zeros(3*nOfFaceNodes,1);
%Is it possible to remove this loop?
for iface = 1:nOfFaces
    tau_f = tau(iface);
    nodes = faceNodes(iface,:); Xf = Xe(nodes,:); % Nodes in the face
    dxdxi = Nx1d*Xf(:,1); dydxi = Nx1d*Xf(:,2);
    dxdxiNorm = sqrt(dxdxi.^2+dydxi.^2);
    dline = dxdxiNorm.*IPw_f';
    nx = dydxi./dxdxiNorm; ny=-dxdxi./dxdxiNorm;
%Face matrices
    ind_face = (iface-1)*nOfFaceNodes + (1:nOfFaceNodes);

    if iface == FACE
    X_fnodes = Xe([FACE_NODE],:)
    X_fg = N1d*X_fnodes;
    t_Term = analyticalPoisson_N(X_fg);
    t_vect = N1d'*(spdiags(dline,0,ngf,ngf))*t_Term;
    fn(ind_face,1) = fn(ind_face,1) + t_vect;
    end

    Alq(ind_face,2*nodes-1) = N1d'*(spdiags(dline.*nx,0,ngf,ngf)*N1d);
    Alq(ind_face,2*nodes) = N1d'*(spdiags(dline.*ny,0,ngf,ngf)*N1d);
    Auu_f = N1d'*(spdiags(dline,0,ngf,ngf)*N1d)*tau_f;
    Auu(nodes,nodes) = Auu(nodes,nodes) + Auu_f;
    Alu(ind_face,nodes) = Alu(ind_face,nodes) + Auu_f;
    All(ind_face,ind_face) = -Auu_f;
end

% Elemental mapping
Aqu = -mu*Auq'; Aul = -Alu'; Aql = mu*Alq';
A = [Auu Auq; Aqu Aqq];
UQ = -A\[Aul;Aql];
fUQ= A\[fe;zeros(2*nOfElementNodes,1)];
U = UQ(1:nOfElementNodes,:);
Uf=fUQ(1:nOfElementNodes); % maps lamba into U
```

```
%%ROBIN BOUNDARY
function [Q,U,Qf,Uf,Alq,Alu,All,f_n] =
KKeElementalMatricesIsoParametric_R(mu,Xe,Te,referenceElement,tau,infoFaces,iElem,pos
_R)
...

%Identify the nodes of the face in the Bounday
FACE = infoFaces.extFace_R(pos_N,2);
FACE_NODE = faceNodes(FACE,:);

...
...

%% Faces computations
Alq = zeros(3*nOfFaceNodes,2*nOfElementNodes);
Auu = zeros(nOfElementNodes,nOfElementNodes);
Alu = zeros(3*nOfFaceNodes,nOfElementNodes);
All = zeros(3*nOfFaceNodes,3*nOfFaceNodes);
fn = zeros(3*nOfFaceNodes,1);
kappa = 0.1;  gamma = 0.3;
%Is it possible to remove this loop?
for iface = 1:nOfFaces
    tau_f = tau(iface);
    nodes = faceNodes(iface,:); Xf = Xe(nodes,:); % Nodes in the face
    dxdxi = Nx1d*Xf(:,1); dydxi = Nx1d*Xf(:,2);
    dxdxiNorm = sqrt(dxdxi.^2+dydxi.^2);
    dline = dxdxiNorm.*IPw_f';
    nx = dydxi./dxdxiNorm; ny=-dxdxi./dxdxiNorm;
    %Face matrices
    ind_face = (iface-1)*nOfFaceNodes + (1:nOfFaceNodes);
    Alq(ind_face,2*nodes-1) = N1d'*(spdiags(dline.*nx,0,ngf,ngf)*N1d);
    Alq(ind_face,2*nodes) = N1d'*(spdiags(dline.*ny,0,ngf,ngf)*N1d);
    Auu_f = N1d'*(spdiags(dline,0,ngf,ngf)*N1d)*tau_f;
    Auu(nodes,nodes) = Auu(nodes,nodes) + Auu_f;
    Alu(ind_face,nodes) = Alu(ind_face,nodes) + Auu_f;
    All(ind_face,ind_face) = -Auu_f;

    %%%Robin modifications
    if iface == FACE
    X_fnodes = Xe([FACE_NODE],:)
    X_fg = N1d*X_fnodes;
    g_Term = analyticalPoisson_R(X_fg);
    g_vect = N1d'*(spdiags(dline,0,ngf,ngf))*g_Term;
    fn(ind_face,1) = fn(ind_face,1) + g_vect;

    All(ind_face,ind_face) =  All(ind_face,ind_face) -
N1d'*(spdiags(dline,0,ngf,ngf)*N1d)*gamma*(1/kappa);
    end

end

% Elemental mapping
Aqu = -mu*Auq'; Aul = -Alu'; Aql = mu*Alq';
A = [Auu Auq; Aqu Aqq];
UQ = -A\[Aul;Aql];
fUQ= A\[fe;zeros(2*nOfElementNodes,1)];
U = UQ(1:nOfElementNodes,:);
Uf=fUQ(1:nOfElementNodes); % maps lamba into U

Q = UQ(nOfElementNodes+1:end,:);
Qf=fUQ(nOfElementNodes+1:end); % maps lamba into Q
```

Finally, for each element **K** and **f** matrices will be

```
%Elemental matrices to be assembled
KKe = Alq*Qe + Alu*Ue + All;
ffe = - fn - (Alq*Qfe + Alu*Ufe);
```

5. Problem solution

Once the code is implemented, defined meshes “mesh4”, with 512 triangular elements, “mesh2” with 32 triangular elements and “mesh1” with 8 triangular elements will be tried to solve the problem, to reach the first conclusions about the problem behaviour and results accuracy.

CASE 1: mesh1_P2 & P4

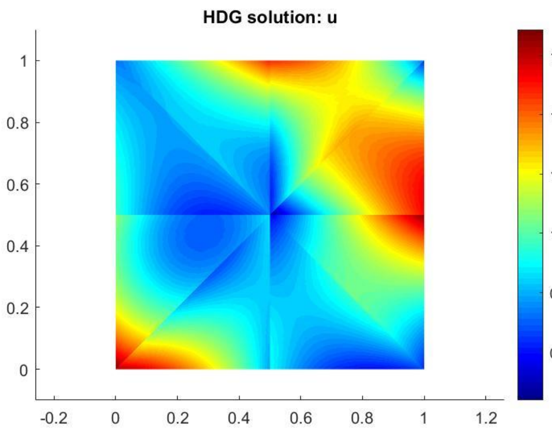

Figure 1. HDG solution u P2 mesh 1

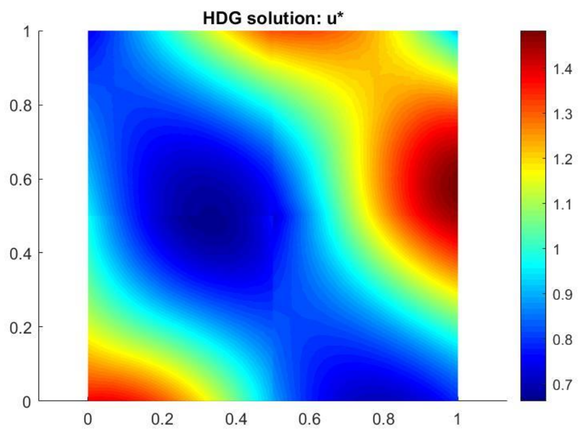

Figure 2. HDG solution u^* P2 mesh 1

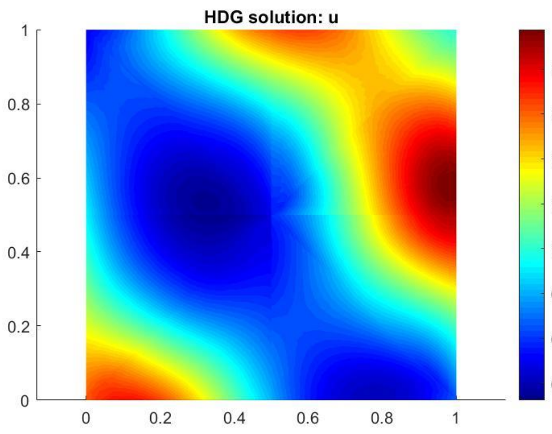

Figure 3. HDG solution u P4 mesh 1

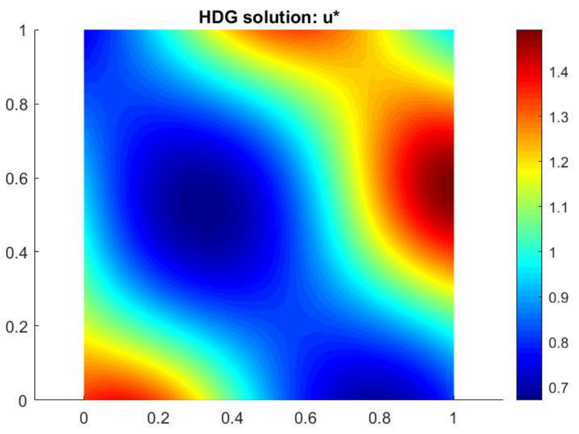

Figure 4. HDG solution u^* P4 mesh 1

Working with such a rude mesh in which there are just 8 elements, first results show the importance of the polynomial degree approximation, most especially in u results. See how for the polynomial interpolation of degree 4 quite good results are already obtained. However, more conclusion will be reached in the following convergence study.

CASE 2: mesh2_P1&P2 & P4

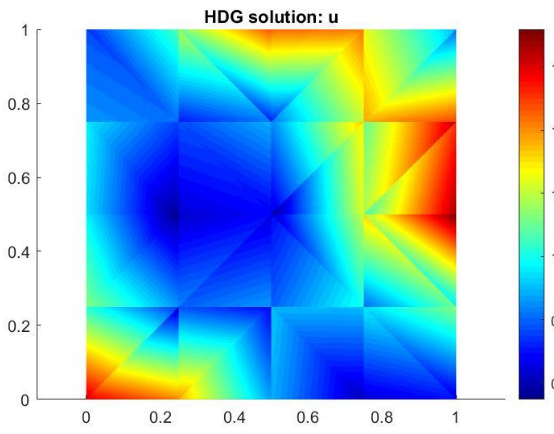

Figure 5. HDG solution u P1 mesh 2

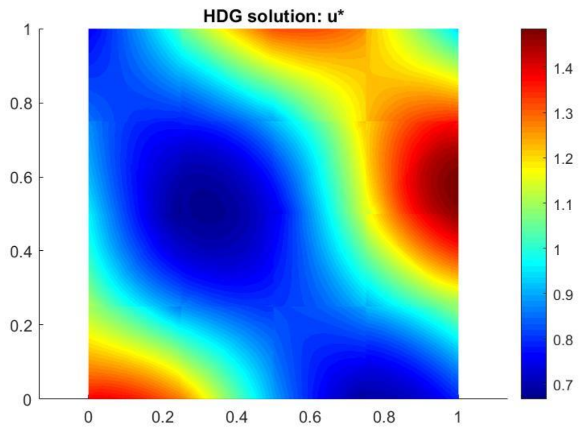

Figure 6. HDG solution u^* P1 mesh 2

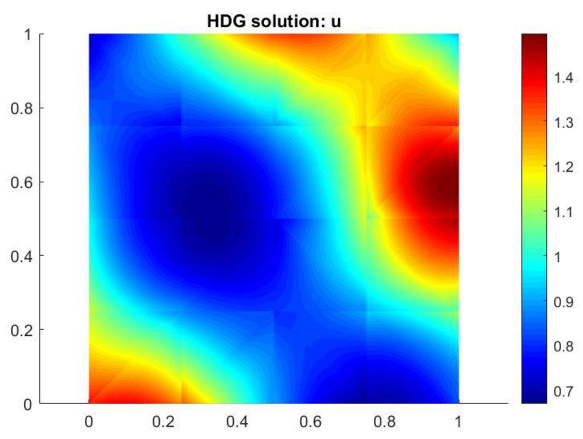

Figure 7. HDG solution u P2 mesh 2

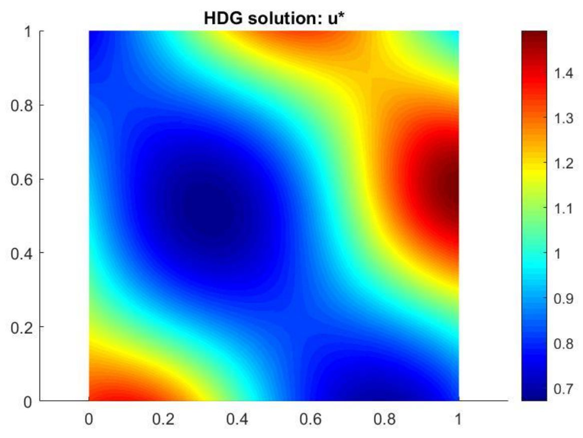

Figure 8. HDG solution u^* P2 mesh 2

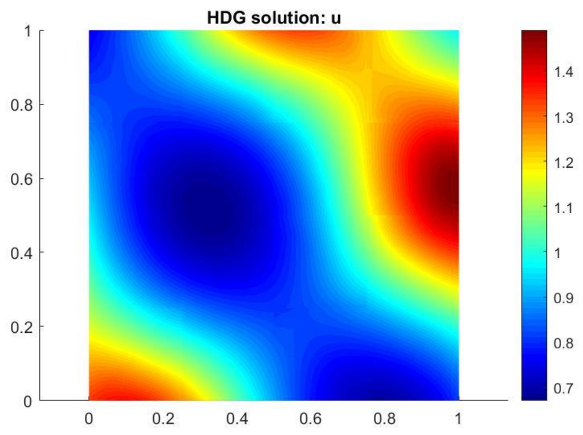

Figure 9. HDG solution u P3 mesh 2

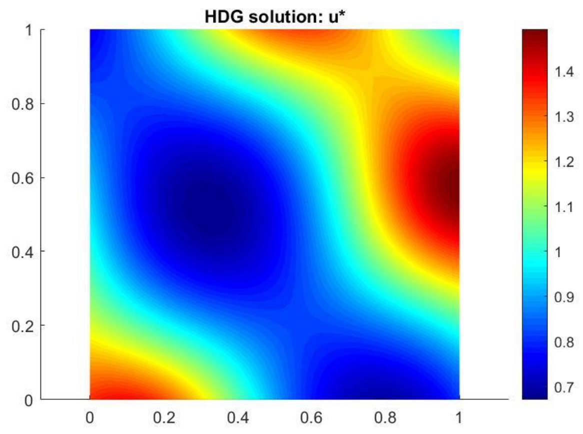

Figure 10. HDG solution u^* P3 mesh 2

For mesh 2, three different polynomial degrees have been introduced to check more in detail how it affects to the solution accuracy. Once again, the principal effects appear un u solution. See how rude the solution is with 3 nodes por element (Figure 5) and how it is improved until reach such a good approximation, in which we can see the "mesh lines" with 6 nodes per element (Figure 7) and 9 nodes per element (Figure 9).

CASE 3: mesh4_P2

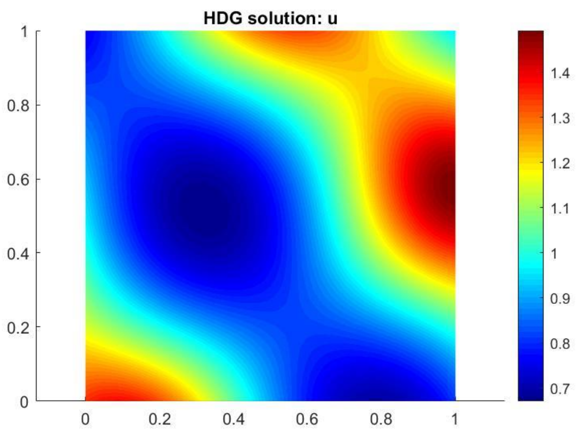

Figure 11. HDG solution u P2 mesh 4

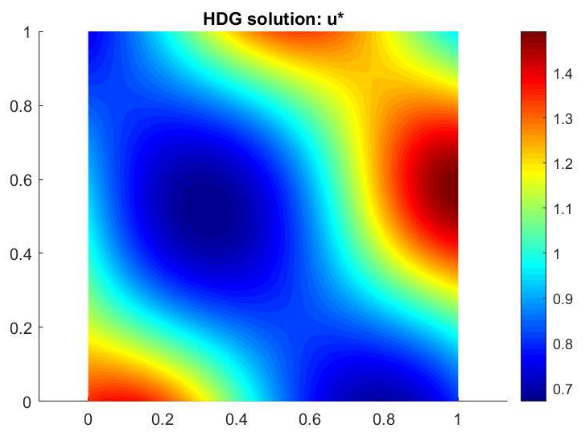

Figure 12. HDG solution u^* P2 mesh 4

Finally, an example for mesh 4, quite a good mesh with 512 is done. See how for a finest mesh, it is not needed to use polynomial degrees of order 3 or higher to obtain quite accurate solutions.

6. Convergence study

Until the moment, different problems have been solved for different meshes and polynomial approximations. From those plots, first conclusion can be reached but it will be with the convergence error study with which a deeper analysis of the results accuracy can be done. For it, the problem will be solved with 4 meshes for k = 1,2,3,4. Then, results will be plot following the scheme of the notes for a better interpretation of them.

First of all, the error variation for a same mesh with different k polynomial approximations is shown. In this way, it can check if it works as expected going lower as the degrees of approximation increases. Then, plots for mesh 2 and mesh 4 are shown.

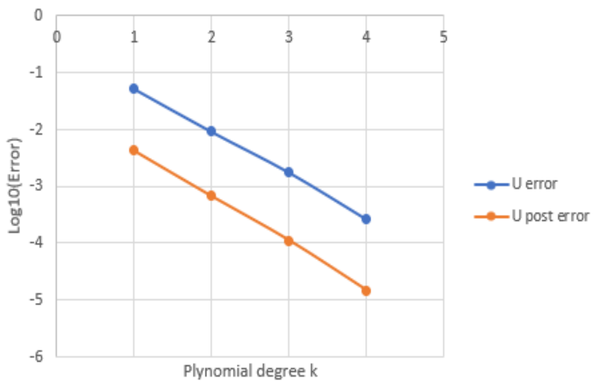

Figure 13. Error plots for mesh 2

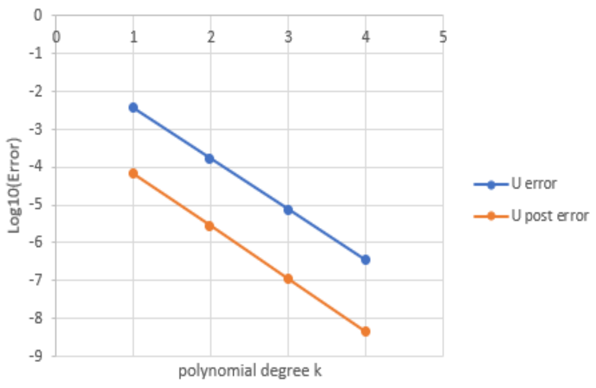

Figure 14. Error plot for mesh 4

Figure above shows error plots for both meshes, that in both cases have the expected behaviour. First of all, the error for post-processing solution is always smaller that for u solution. Both errors have the same tendency, going to a smaller name as k goes larger. Moreover, if we compare the results between them, the error value for u as for u^* is always smaller for the finest mesh (mesh 4, Figure 14).

Taking all of this into account, it seems that the obtained results will be validated having such a good accuracy, but for be sure about it, a convergence study with four different meshes is done.

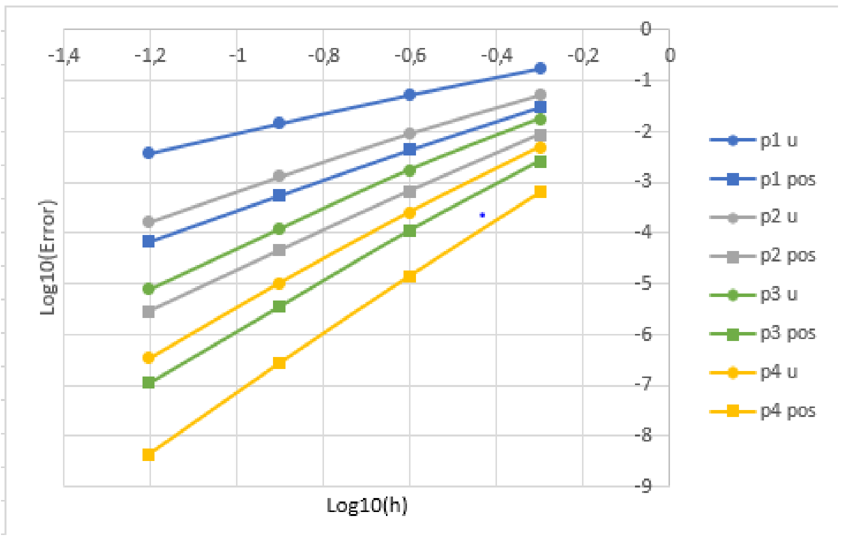

Figure 15 Convergence plot

The figure above shows error results, grouping in a same line those that has the same polynomial degree p for u and u^*. See how for a same size h, the error of $u^*(p)$ is lower than the error $u(p+1)$, being the solution u^* more accurate for lower degree than the solution u. This is quite a good indicator of the good implementation of the method, working as expected if it is compared with the convergence plots of the biography. Thus, we can conclude that the method is correctly implemented and can be sure about the accuracy of the results.