CIMNE. International Centre for Numerical Methods in Engineering

Master of Science in Computational Mechanics

Universitat Politècnica de Catalunya

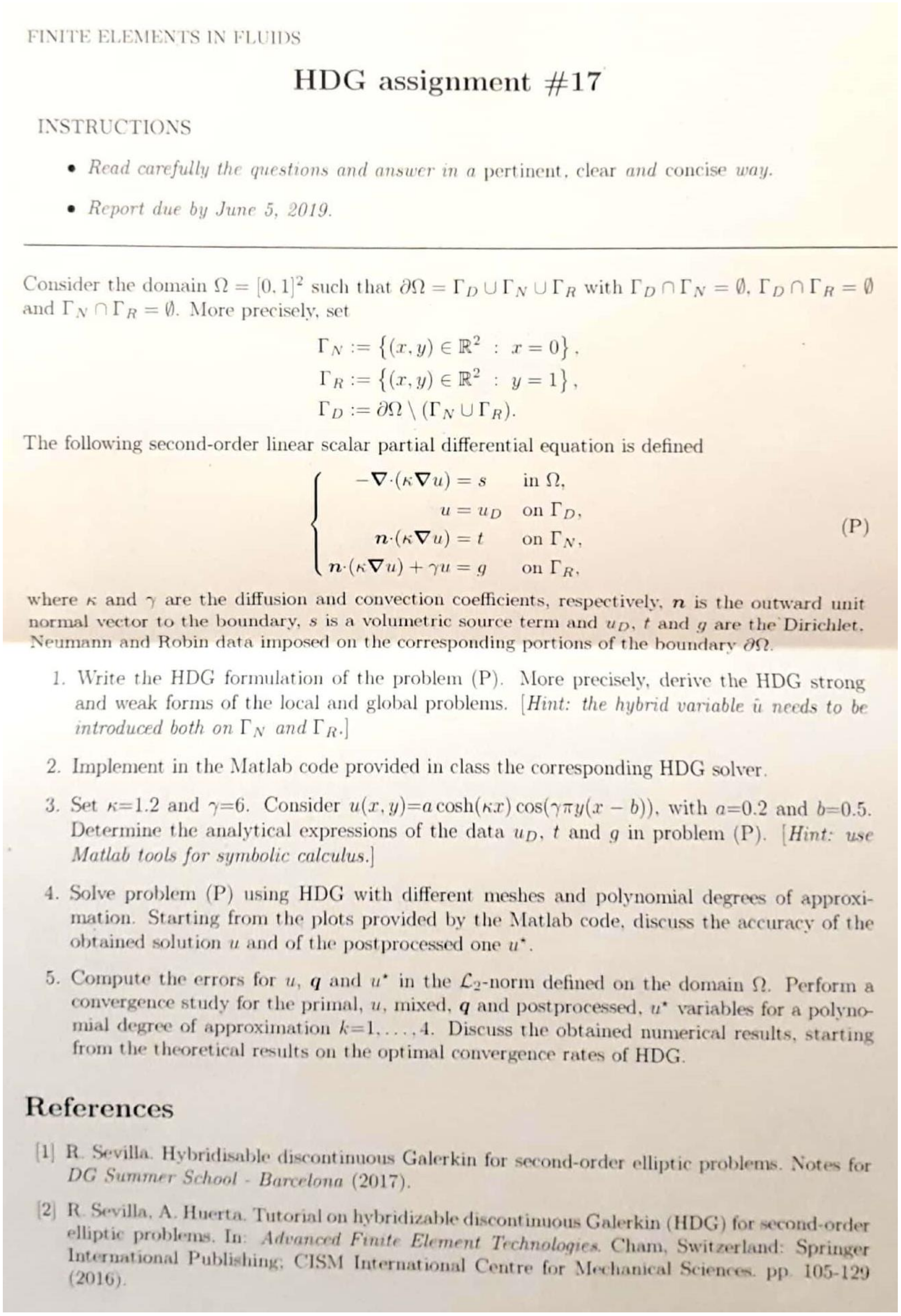
Finite elements in fluids
Homework #7

Submitted By

Aly Youssef

Table of Contents

Problem description



FINITE ELEMENTS IN FLUIDS

HDG assignment #17

INSTRUCTIONS

- *Read carefully the questions and answer in a* pertinent, clear *and* concise *way.*
- *Report due by June 5, 2019.*

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

$$\begin{aligned}\Gamma_N &:= \left\{(x,y) \in \mathbb{R}^2 \ : \ x = 0\right\},\\ \Gamma_R &:= \left\{(x,y) \in \mathbb{R}^2 \ : \ y = 1\right\},\\ \Gamma_D &:= \partial\Omega \setminus (\Gamma_N \cup \Gamma_R).\end{aligned}$$

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

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

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

1. Write the HDG formulation of the problem (P). More precisely, derive the HDG strong and weak forms of the local and global problems. [*Hint: the hybrid variable* $\hat{u}$ *needs to be introduced both on* Γ_N *and* Γ_R.]
2. Implement in the Matlab code provided in class the corresponding HDG solver.
3. Set κ=1.2 and γ=6. Consider $u(x,y)$=$a\cosh(\kappa x)\cos(\gamma\pi y(x-b))$, with a=0.2 and b=0.5. Determine the analytical expressions of the data u_D, t and g in problem (P). [*Hint: use Matlab tools for symbolic calculus.*]
4. Solve problem (P) using HDG with different meshes and polynomial degrees of approximation. Starting from the plots provided by the Matlab code, discuss the accuracy of the obtained solution u and of the postprocessed one $u^\star$.
5. Compute the errors for u, $\boldsymbol{q}$ and $u^\star$ in the $\mathcal{L}_2$-norm defined on the domain Ω. Perform a convergence study for the primal, u, mixed, $\boldsymbol{q}$ and postprocessed, $u^\star$ variables for a polynomial degree of approximation k=1, . . . , 4. Discuss the obtained numerical results, starting from the theoretical results on the optimal convergence rates of HDG.

References

[1] R. Sevilla. Hybridisable discontinuous Galerkin for second-order elliptic problems. Notes for *DG Summer School - Barcelona* (2017).

[2] R. Sevilla, A. Huerta. Tutorial on hybridizable discontinuous Galerkin (HDG) for second-order elliptic problems. In: *Advanced Finite Element Technologies*. Cham, Switzerland: Springer International Publishing; CISM International Centre for Mechanical Sciences. pp. 105-129 (2016).

1. HDG formulation

(1)

The following second order linear scalar pde is defined:

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

The equivalent strong form could be written in the broken computational domain as follows:

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

$\Rightarrow$ The strong form

$\rightarrow$ The local problem with dirichlet BC is defined as:

$$\begin{cases} \nabla\cdot q_i = s & \text{in } \Omega_i \\ q_i + k\nabla u_i = 0 & \text{in } \Omega_i \\ u_i = u_D & \text{on } \partial\Omega_i \cup \Gamma_D \\ u_i = \hat{u} & \text{on } \partial\Omega_i \setminus \Gamma_D \end{cases}$$

$\rightarrow$ The global problem is defined as:

$$\begin{cases} [\![n\cdot q]\!] = 0 & \text{on } \Gamma \\ n\cdot q = -t & \text{on } \Gamma_N \\ n\cdot q = \gamma u - g & \text{on } \Gamma_R \end{cases}$$

When imposing $[\![u n]\!] = 0$ is not required as $\hat{u}$ is unique for adjacent elements

$\Rightarrow$ Weak formulation

$\rightarrow$ For the local problem, the weak form is defined as follows when q_i, u_i are unknowns $\in W(\Omega_i) \times V(\Omega_i)$

$$\left.\begin{aligned} &-(\nabla v, q_i)_{\Omega_i} + \langle v, n_i \cdot \hat{q}_i \rangle_{\partial\Omega_i} = (v, s)_{\Omega_i} \\ &-(w, q_i)_{\Omega_i} + k(\nabla \cdot w, u_i)_{\Omega_i} = k\langle n_i \cdot w, u_D \rangle_{\partial\Omega_i \cap \Gamma_D} + k\langle n_i \cdot w, \hat{u} \rangle_{\partial\Omega_i \setminus \Gamma_D} \end{aligned}\right\} \circledast$$

Where the numerical traces $\hat{q}$ are defined as:

$$n \cdot \hat{q}_i := \left.\begin{cases} n_i \cdot q_i + \tau_i(u_i - u_D) & ; \text{ on } \partial\Omega_i \cap \Gamma_D \\ n_i \cdot q_i + \tau_i(u_i - \hat{u}) & ; \text{ elsewhere} \end{cases}\right\} \text{(**)}$$

Integrating by parts the second term of the first equation in $\circledast$ and substituting with (**)

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

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

$\rightarrow$ For the global problem, the weak form is defined as follows when $\hat{u} \in \mathcal{M}(\Gamma \cup \Gamma_N \cup \Gamma_R)$

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

Substituting in (**)

$$\sum_{i=1}^{n_{el}} \langle \mu, n_i q \rangle_{\partial\Omega_i \setminus \Gamma_D} + \langle \mu, \tau_i u_i \rangle_{\partial\Omega_i \setminus \Gamma_D} - \langle \mu, \tau_i \hat{u} \rangle_{\partial\Omega_i \setminus \Gamma_D} - \langle \mu, \gamma \hat{u} \rangle_{\partial\Omega_i \setminus \Gamma_R}$$

$$= -\sum_{i=1}^{n_{el}} \langle \mu, t \rangle_{\partial\Omega_i \cap \Gamma_N} + \langle \mu, g \rangle_{\partial\Omega_i \cap \Gamma_R}$$

The element by element nodal interpolation is as follows:

$$q \simeq q^h = \sum_{n=1}^{nd} N_j q_j \in \omega^h$$

$$u \simeq u^h = \sum_{n=1}^{nd} N_j u_j \in V^h$$

$$\hat{u} \simeq \hat{u}^h = \sum_{n=1}^{nd} \hat{N}_j \hat{u}_j \in \eta^h$$

This gives rise to the following elemental system of equations

$$\begin{bmatrix} A_{uu} & A_{uq} \\ k A_{uq} & A_{qq} \end{bmatrix} \begin{bmatrix} u_i \\ q_i \end{bmatrix} = \begin{bmatrix} f_u \\ k f_q \end{bmatrix} + \begin{bmatrix} A_{u\hat{u}} \\ k A_{q\hat{u}} \end{bmatrix} \hat{u}_i \qquad \boxed{*}$$

$$\sum_{n=1}^{nd} \begin{bmatrix} A^T_{u\hat{u}} & A^T_{q\hat{u}} \end{bmatrix} \begin{bmatrix} u_i \\ q_i \end{bmatrix} + [A_{\hat{u}\hat{u}}]_i \hat{u}_i + [A_{\hat{u}\hat{u}}]_i \hat{u}_i = \sum_{n=1}^{nd} [f_{\hat{u}}]_i + [f^R_{\hat{u}}]_i \qquad \boxed{**}$$

Where:

$$A^R_{\hat{u}\hat{u}} = -\sum_{\partial\Omega_i \cup \Gamma_R} \sum_{g=1} \gamma \hat{N}_n(\xi^f_g) \hat{N}^T(\xi^f_g) w^f_g$$

$$f^R_{\hat{u}} = -\sum_{\partial\Omega \cup \Gamma_R} \sum_{g=1} N(\xi^f_g)\, g(x(\xi^f_g)) w^f_g$$

After substituting $\boxed{*}$ in $\boxed{**}$ the global problem becomes:

$$\hat{k}\hat{u} = \hat{f}$$

where:

$$\hat{k} = \mathop{\mathrm{A}}_{i=1}^{nel} \begin{bmatrix} A^T_{u\hat{u}} & A^T_{q\hat{u}} \end{bmatrix}_i \begin{bmatrix} A_{uu} & A_{uq} \\ kA^T_{uq} & A_{qq} \end{bmatrix}^{-1}_i \begin{bmatrix} A_{u\hat{u}} \\ kA_{q\hat{u}} \end{bmatrix} + [A_{u\hat{u}}]_i + [A^R_{\hat{u}\hat{u}}]_i$$

$$\hat{f} = \mathop{\mathrm{A}}_{i=1}^{nel} [f_{\hat{u}}]_i + [f^R_{\hat{u}}]_i - \begin{bmatrix} A^T_{u\hat{u}} & A^T_{q\hat{u}} \end{bmatrix}_i \begin{bmatrix} A_{uu} & A_{uq} \\ kA^T_{uq} & A_{qq} \end{bmatrix}^{-1}_i \begin{bmatrix} f_u \\ k f_q \end{bmatrix}_i$$

4. MATLAB implementation

The following modifications where applied to the provided code in order to match the case of the implementation in hand (the code is provided in the Appendix):

- The function GetFaces is modified in order to store the information regarding the boundary face types (Appendix 1).
- The degrees of freedom are to be computed from the global face number of the Dirichlet boundaries (Appendix 2).
- The face type is specified through the function hdg_preprocess, where the variable face_type takes the value 0 for Dirichlet, 1 for Neumann, 2 for Robin and 3 for internal faces.
- The function hdgMatrixPoisson is modified in order to accommodate the new definition of the global matrices taking into account the Robin boundary condition (Appendix 4).
- The functions defining the analytical solution, source, traction and Robin boundaries condition are added (Appendix 5).

3. Analytical solution

The analytical solution is defined as follows where $\kappa = 1.2;\ \gamma = 6;\ \mathrm{a} = 0.2;\ \mathrm{b} = 0.5$:

$$u_D = a\cos\left(\gamma y\pi(b-x)\right); \text{for } x = 0$$

$$u_D = \cosh(\kappa x); \text{for } y = 0$$

$$t = -\kappa\left(a\kappa\cos\left(\gamma y\pi(b-x)\right)\sinh(\kappa x) + a\gamma y\pi\sin\left(\gamma y\pi(b-x)\right)\cosh(\kappa x)\right)$$

$$g = \kappa\left(a\kappa\cos\left(\gamma y\pi(b-x)\right)\sinh(\kappa x) + a\gamma y\pi\sin\left(gammay\pi(b-x)\right)\cosh(\kappa x)\right) + a\gamma cos\left(\gamma y\pi(b-x)\right)\cosh(\kappa x)$$

The analytical solution of the function is plotted in the following Figure 1:

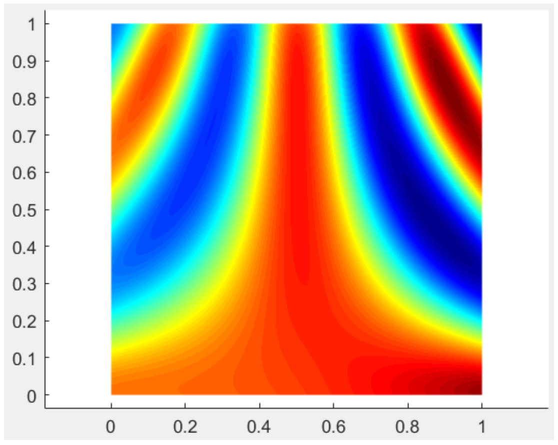
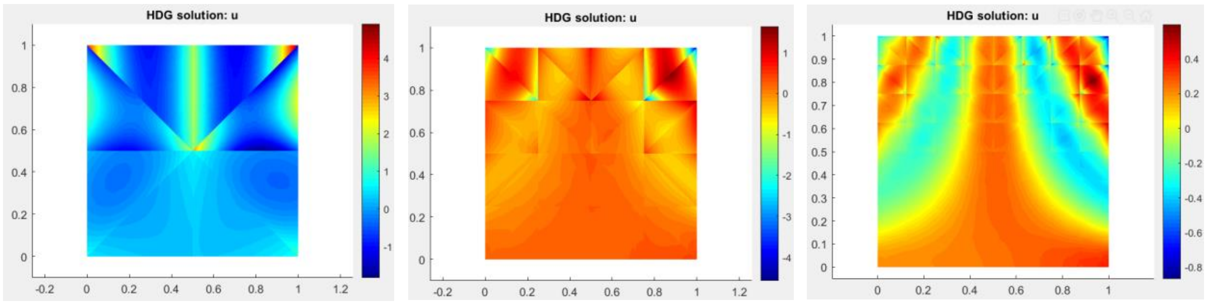


Figure 1: Analytical solution of the function

4. Results and discussion

The results for the HDG method are shown in Figure 2. The results are obtained using the provided meshes which increase element density (h-refinement) and quadratic approximation. It could be seen that the solution becomes similar to the analytical solution starting from the fourth mesh. However, the effect of the discontinuity of the mesh are still observable. This is reduced as the mesh become more fine where the 6th mesh achieves a solution comparable to the analytical one.



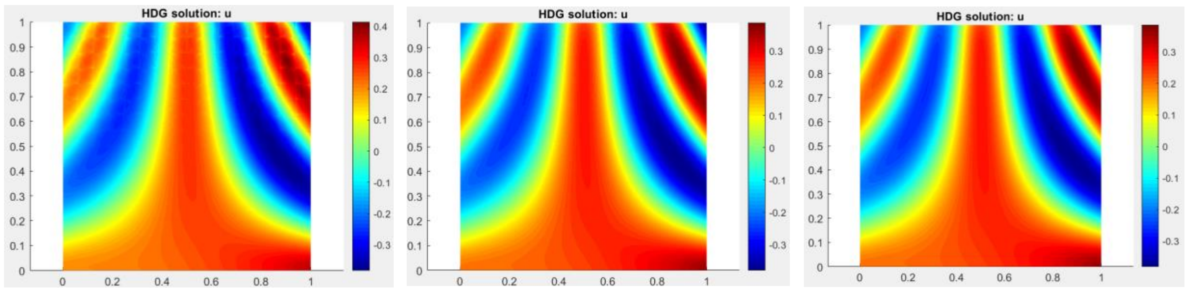


Figure 2 HDG solution while increasing the mesh size

Another analysis could be performed regarding the effect of the approximation's degree. For this analysis, the HDG solution is obtained with the 2nd mesh and degrees from 1 to 4. It could be observed in Figure 3 that the 4th order mesh achieve an adequate solution comparable to the analytical one. It could be noted that this is achieved using a relatively coarse mesh.

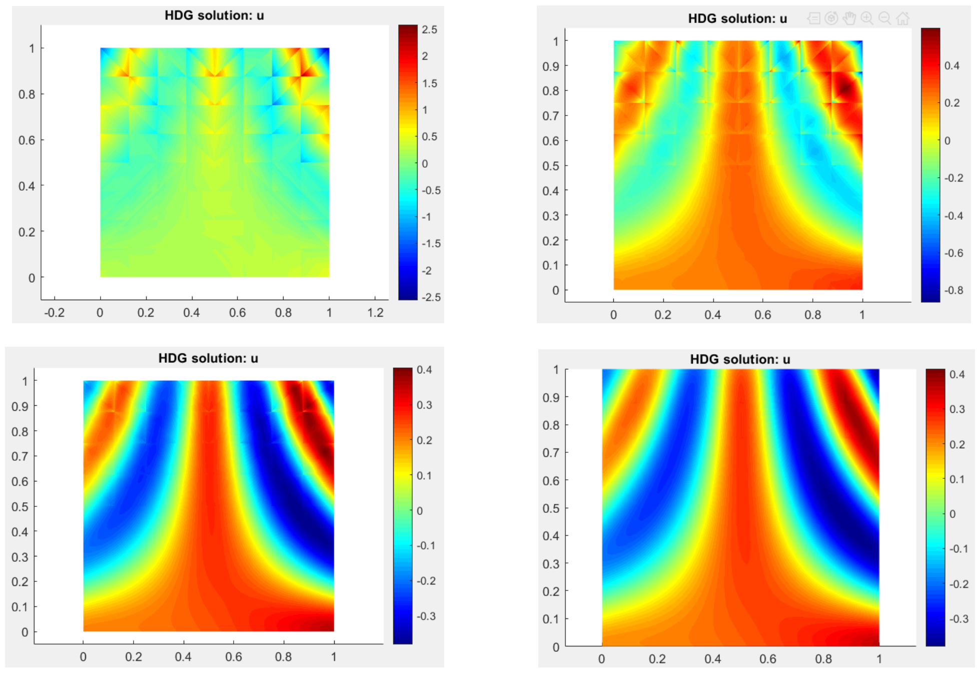


Figure 3 HDG solution while increasing the approximation order

The goal of post-processing is to allow the usage of a coarser mesh and lower degree of approximation thus reducing the computational cost. This is achieved through recalculating the unknowns inside the elements using the obtained fluxes on the edges. It could be observed from Figure 4 that the postprocessing method achieves a satisfactory solution that is comparable to the analytical one. Its advantages are seen compared to the normal method of solution where a coarse mesh with a low degree of approximation is used to provide a sufficient result that is wouldn’t be otherwise achieved .

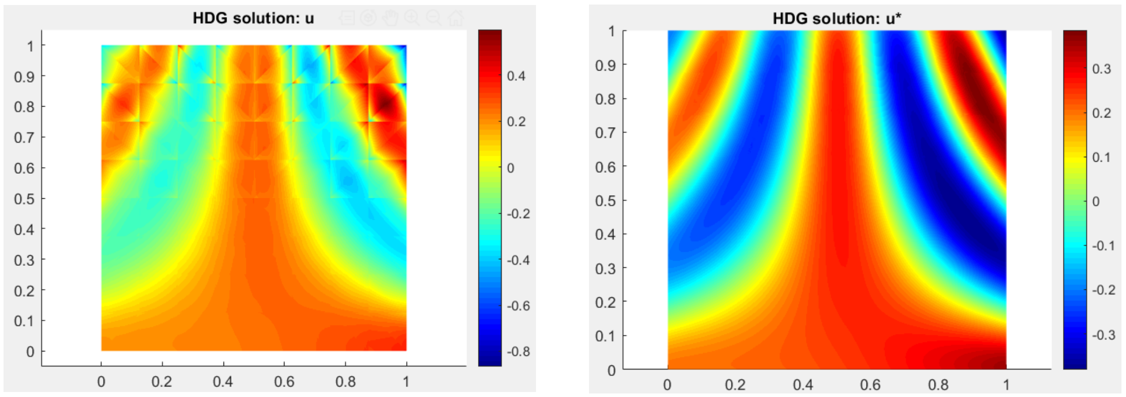


Figure 4 Postprocessing effect on the solution

The convergence rate plot is shown in Figure 5. This curve is conducted by plotting the error for each tested case compared to the analytical solution while increasing the mesh size for each of the approximation degrees. Both the not post-processed and the post-processed method of solution are considered. It could be noted that as the degree of approximation increases, the rate of convergence increase. This entails that a coarser mesh with higher degree or approximation could be used in order to achieve the same level of solution accuracy. Thus, it is more beneficial to increase the degree of approximation than to increase the fineness of the mesh. The convergence rate plot also shows the effect of post-processing. It could be observed that post processing achieves better convergence compared to the normal HDG method. Moreover, a better solution is achieved using a lower degree of approximation. Thus, it could be said that utilizing post-processing s more beneficial than increasing the degree of the polynomial.

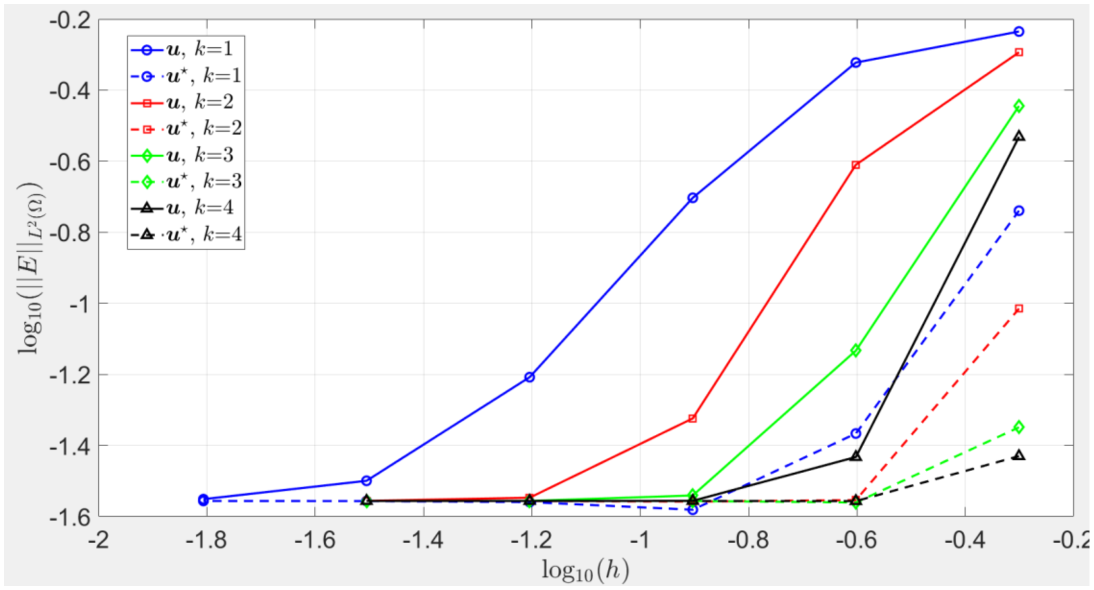


Figure 5 Convergence rate analysis

Appendix

Appendix 1

Modification to GetFace.m

```
function [intFaces,extFaces] = GetFaces(X,T)

for iElem=1:nElem
    for iFace=1:nfaceel
        if(markE(iElem,iFace)==0)
            markE(iElem,iFace)=1;
            nodesf = T(iElem,Efaces(iFace,:));

            jelem = FindElem(iElem,nodesf);

            if(jelem~=0)
                [jface,node1]=FindFace(nodesf,T(jelem,:),Efaces);
                intFaces(intF,:)=[iElem,iFace,jelem,jface,node1];
                intF = intF +1;
                markE(jelem,jface)=1;
            else
                 x = X( nodesf , 1 ) ;
                 if x  ( 1 )==0  && x  ( 2 )==0
                    extFaces  ( extF  ,  :  )  =[  iElem  ,  iFace  ,  1  ]  ;
                 elseif x  ( 1  )==1  && x  ( 2  )==1
                    extFaces  ( extF  ,  :  )  =[  iElem  ,  iFace  ,  2  ]  ;
                 else
                    extFaces  ( extF  ,  :  )  =[  iElem  ,  iFace  ,  0  ]  ;
                 end
                 extF = extF + 1;
            end
        end
    end
```

Appendix 2

Modification to mainPoissonHDG.m and convergencePoisson.m

```
%Dirichlet BC
 %Dirichlet face nodal coordinates
 nOfFaceNodes = degree+1;
 nOfInteriorFaces = size (infoFaces.intFaces,1) ;
 nOfExteriorFaces = size (infoFaces.extFaces,1) ;
 res = 1 : nOfFaceNodes ;
 res1 = find (infoFaces.faceTypes==0) ;
 res2 = find (infoFaces.faceTypes~=0) ;
 nOfDirichletFaces = size (aux1,1) ;
 nOfNonDirichletFaces = nOfExteriorFaces - nOfDirichletFaces ;
 dofDirichlet = zeros ( nOfDirichletFaces * nOfFaceNodes , 1 ) ;
 for i =1: size ( aux1 , 1 )
    dofDirichlet (1+( i-1)*nOfFaceNodes : i *nOfFaceNodes ) = ( res1(i)-
1)*nOfFaceNodes + res ;
 end
 dofUnknown = zeros ( ( nOfNonDirichletFaces + nOfInteriorFaces ) * nOfFaceNodes , 1 )
;
 for i =1: size ( res2 , 1 )
```

```
    dofUnknown(1+(i-1)*nOfFaceNodes : i*nOfFaceNodes ) = (res2(i)-1)*nOfFaceNodes+
aux0 ;
 end

uDirichlet =
computeProjectionFaces(@analyticalPoisson,infoFaces.extFaces,X,T,referenceElement);
```

Appendix 3

Modification to hdg_preprocess.m

```
for iFace = 1:nOfExteriorFaces
    infoFace = extFaces(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace + nOfInteriorFaces;
    faceTypes ( iFace+ nOfInteriorFaces ) = infoFace ( 3 ) ;
end
```

Appendix 4

Modification to hdgMatrixPoisson.m

```
%% Faces computations
Alq = zeros(3*nOfFaceNodes,2*nOfElementNodes);
Auu = zeros(nOfElementNodes,nOfElementNodes);
Alu = zeros(3*nOfFaceNodes,nOfElementNodes);
All = zeros(3*nOfFaceNodes,3*nOfFaceNodes);
%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;
end
```

Appendix 5

```
function u = analyticalPoisson(X)

% Parameters
kappa = 1.2;
gamma = 6;
a = 0.2;
b = 0.5;
% Points
x = X(:,1);
y = X(:,2);

% Solution
 u = a*cosh(kappa*x).*cos(gamma*pi*y.*(x-b)) ;
```

```
function s = sourcePoisson(X, mu)

% Parameters
kappa = 1.2;
gamma = 6;
a = 0.2;
b = 0.5;

% Points
x = X(:,1);
y = X(:,2);

% Minus Laplacian
s = -kappa*(a*kappa^2*cos(gamma*pi*y.*(b - x)).*cosh(kappa*x) -
a*gamma^2*pi^2*y.^2.*cos(gamma.*y.*pi.*(b - x)).*cosh(kappa.*x) -
a.*gamma.^2.*pi.^2.*cos(gamma.*y.*pi.*(b - x)).*cosh(kappa.*x).*(b - x).^2
+...
    2.*a.*gamma.*kappa.*y.*pi.*sin(gamma.*y.*pi.*(b - x)).*sinh(kappa.*x));

function t = tractionPoisson(X, mu)

% Parameters
kappa = 1.2;
gamma = 6;
a = 0.2;
b = 0.5;

% Points
x = X(:,1);
y = X(:,2);

% Minus Laplacian
t = -kappa*(a*kappa*cos(gamma*pi*y.*(b - x)).*sinh(kappa*x) +
a*gamma*pi*y.*sin(gamma*pi*y.*(b - x)).*cosh(kappa*x));

function g = RobinPoisson(X, mu)

% Parameters
kappa = 1.2;
gamma = 6;
a = 0.2;
b = 0.5;

% Points
x = X(:,1);
y = X(:,2);

% Minus Laplacian
g = kappa*(a*kappa*cos(gamma*pi*y.*(b - x)).*sinh(kappa*x) +...
    a*gamma*pi*y.*sin(gamma*pi*y.*(b - x)).*cosh(kappa*x)) +
a*gamma*cos(gamma*pi*y.*(b - x)).*cosh(kappa*x);
```