INTERNATIONAL CENTRE FOR
NUMERICAL METHODS IN ENGINEERING

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER OF SCIENCE IN COMPUTATIONAL MECHANICS

Finite Element in Fluids

Hybridizable Discontinuous Galerkin Homework

Assignment # 16

Eugenio José Muttio Zavala

June 5, 2019

Submitted To:
Prof. Antonio Huerta
Prof. Matteo Giacomini
Prof. Pablo Saez

1 HYBRIDIZABLE DISCONTINUOUS GALERKIN

PROBLEM STATEMENT

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

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

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

$$\begin{cases} -\nabla\cdot(\kappa\,\nabla u) = s & \text{in } \Omega,\\ u = u_D & \text{on } \Gamma_D,\\ n\cdot(\kappa\,\nabla u) = t & \text{on } \Gamma_N,\\ n\cdot(\kappa\,\nabla u) + \gamma u = g & \text{on } \Gamma_R,\end{cases} \tag{1.1}$$

where κ and γ are the diffusion and convection coefficients, respectively, $\mathbf{n}$ 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 (1.1). 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 on both on Γ_N and Γ_R]*

STRONG FORM

The strong form for the second-order elliptic problem is shown in the equation 1.1 which represents the resolution of the Poisson equation in two dimensions in which a κ parameter is imposed and also a *Robin* boundary condition is applied together with the corresponding *Neuman* and *Dirichlet*. Then, as the HDG methodology suggests, an equivalent strong form of the second-order elliptic problem can be written in a *broken* domain as:

$$\begin{cases} -\nabla\cdot(\kappa\,\nabla u) = s & \text{in } \Omega_i, \text{ and for } i = 1,...,n_{el},\\ u = u_D & \text{on } \Gamma_D,\\ \mathbf{n}\cdot(\kappa\,\nabla u) = t & \text{on } \Gamma_N,\\ \mathbf{n}\cdot(\kappa\,\nabla u) + \gamma u = g & \text{on } \Gamma_R,\\ [\![u\,\mathbf{n}]\!] = \mathbf{0} & \text{on } \Gamma,\\ [\![\mathbf{n}\cdot\nabla u]\!] = 0 & \text{on } \Gamma,\end{cases} \tag{1.2}$$

where the two last equations correspond to the imposition of the continuity of the primal variable u and the normal fluxes respectively along the internal interface Γ. Finally, the strong form is written in *mixed form*, in which the problem is solved as two equivalent or in other words as a system of first-order equations over the broken computational domain as:

$$\begin{cases} \nabla \cdot \mathbf{q}_i = \dfrac{s}{\kappa} & \text{in } \Omega_i, \text{ and for } i = 1, ..., n_{el}, \\ \mathbf{q} + \nabla u_i = \mathbf{0} & \text{in } \Omega_i, \text{ and for } i = 1, ..., n_{el}, \\ u_i = u_D & \text{on } \partial\Omega_i \cap \Gamma_D, \\ u_i = \hat{u} & \text{on } \partial\Omega_i \backslash \Gamma_D, \end{cases} \tag{1.3}$$

This approach assumes $\hat{u}$ given. In each element Ω_e this problem produces an element-by-element solution $\mathbf{q}_e$ and $\mathbf{u}_e$ as a function of the unknown $\hat{u}$. Now, the *transmission condition* for the global problem must be imposed as:

$$\begin{cases} [\![\mathbf{n} \cdot \mathbf{q}]\!] = 0 & \text{on } \Gamma \\ \mathbf{n} \cdot \mathbf{q} = -t & \text{on } \Gamma_N \\ \mathbf{n} \cdot \mathbf{q} = \gamma \hat{u} - g & \text{on } \Gamma_R \end{cases} \tag{1.4}$$

Note that the continuity of u along Γ is automatically satisfied because $u = \hat{u}$ on Γ, as imposed by the local problems in equation 1.3, and $u = \hat{u}$ is unique for adjacent elements.

Weak Form

The weak form for each element equivalent to 1.3 can be written as follows: $for\ i = 1, ..., n_{el}$, given u_D on Γ_D and $\hat{u}$ on $\Gamma \cup \Gamma_N \cup \Gamma_R$, find $(\mathbf{q}_i, u_i) \in \mathcal{W}(\Omega_i) \times \mathcal{V}(\Omega_i)$ that satisfies:

$$\begin{aligned} -(\nabla v,\ \mathbf{q}_i)_{\Omega_i} + \langle v,\ n_i \cdot \hat{\mathbf{q}}_i \rangle_{\partial\Omega_i} &= (v, s/\kappa)_{\Omega_i} \\ -(\mathbf{w},\ \mathbf{q}_i)_{\Omega_i} + (\nabla \cdot \mathbf{w}, u_i)_{\Omega_i} &= \langle \mathbf{n}_i \cdot \mathbf{w}, u_D \rangle_{\partial\Omega_i \cap \Gamma_D} + \langle \mathbf{n}_i \cdot \mathbf{w}, \hat{u} \rangle_{\partial\Omega_i \backslash \Gamma_D} \end{aligned} \tag{1.5}$$

where the numerical traces of the fluxes $\hat{\mathbf{q}}_i$ must be defined. This problem imposes the Dirichlet boundary conditions weakly, and the numerical traces, which formally should be $\mathbf{n}_i \cdot \hat{\mathbf{q}}_i = \mathbf{n}_i \cdot \mathbf{q}_i$, but for stability purposes they are defined element-by-element as:

$$\mathbf{n}_i \cdot \hat{\mathbf{q}}_i := \begin{cases} \mathbf{n}_i \cdot \mathbf{q}_i + \tau_i (u_i - u_D) & \text{on } \partial\Omega_i \cap \Gamma_D \\ \mathbf{n}_i \cdot \mathbf{q}_i + \tau_i (u_i - \hat{u}) & \text{elsewhere} \end{cases} \tag{1.6}$$

where τ_i corresponds to a stabilization parameter defined element-by-element, whose selection has an important effect on the stability, accuracy, and convergence properties of the resulting HDG method. Once the weak form for the local problem is presented, the global problem 1.4 is of interest. The weak form equivalent to 1.4 is to find $\hat{u} \in \mathcal{M}(\Gamma \cup \Gamma_N \cup \Gamma_R)$ for all $\mu \in \mathcal{M}(\Gamma \cup \Gamma_N \cup \Gamma_R)$ such that:

$$\sum_{n=1}^{n_{el}} \langle \mu,\ \mathbf{n}_i \cdot \hat{\mathbf{q}}_i \rangle_{\partial\Omega_i \backslash \partial\Omega} + \sum_{n=1}^{n_{el}} \langle \mu,\ \mathbf{n}_i \cdot \hat{\mathbf{q}}_i + t \rangle_{\partial\Omega_i \cap \Gamma_N} + \sum_{n=1}^{n_{el}} \langle \mu,\ \mathbf{n}_i \cdot \hat{\mathbf{q}}_i + g - \gamma \hat{u} \rangle_{\partial\Omega_i \cap \Gamma_R} = 0 \tag{1.7}$$

Replacing the definition of the fluxes (1.6) into the previous equation, the global problem becomes:

$$\begin{aligned} \sum_{n=1}^{n_{el}} \Big\{ \langle \mu,\ \mathbf{n}_i \cdot \mathbf{q}_i \rangle_{\partial\Omega_i \backslash \Gamma_D} + \langle \mu, \tau_i u_i \rangle_{\partial\Omega_i \backslash \Gamma_D} - \langle \mu, \tau_i \hat{u} \rangle_{\partial\Omega_i \backslash \Gamma_D} - \langle \mu, \gamma \hat{u} \rangle_{\partial\Omega_i \cap \Gamma_R} \Big\} \\ = - \sum_{n=1}^{n_{el}} \Big\{ \langle \mu, g \rangle_{\partial\Omega_i \cap \Gamma_R} + \langle \mu, t \rangle_{\partial\Omega_i \cap \Gamma_N} \Big\} \end{aligned} \tag{1.8}$$

SPATIAL DISCRETIZATION

To obtain the weak forms, the scalar and vector spaces were considered:

$$\begin{aligned}\mathcal{W}(D) &= \{\mathbf{w} \in [\mathcal{H}^1(D)]^2, D \subset \Omega\}\\ \mathcal{V}(D) &= \{v \in \mathcal{H}^1(D), D \subset \Omega\}\\ \mathcal{M}(S) &= \{\mu \in \mathcal{L}_2(S), S \subset \Gamma \cup \partial\Omega\}\end{aligned}$$

Now, to consider a numerical approximation of the problem a discretization of the local (1.5-1.6) and global (1.8) problems a discrete finite element spaces are introduced:

$$\begin{aligned}\mathcal{W}^h(\Omega) = &\quad \{\, w \in [\mathcal{L}_2(\Omega)]^2;\ w|_{\Omega_i} \in [\mathcal{P}^p(\Omega_i)]^2\ \forall \Omega_i\} && \subset \mathcal{W}(\Omega)\\ \mathcal{V}^h(\Omega) = &\quad \{v \in [\mathcal{L}_2(\Omega)]^2;\ v|_{\Omega_i} \in \mathcal{P}^p(\Omega_i)\ \forall \Omega_i\} && \subset \mathcal{V}(\Omega)\\ \mathcal{M}^h(S) = &\quad \{\mu \in [\mathcal{L}_2(S)]^2;\ \mu|_{\Gamma_i} \in \mathcal{P}^p(\Gamma_i)\ \forall \Omega_i \subset S \subset \Gamma \cup \partial\Omega\} && \subset \mathcal{M}(S)\end{aligned} \tag{1.9}$$

These spaces give rise to an element-by-element nodal interpolation of the corresponding variables:

$$\mathbf{q} \approx \mathbf{q}^h = \sum_{n=1}^{n_{el}} N_j\, \mathbf{q}_j \quad \in \mathcal{W}^h \tag{1.10}$$

$$u \approx u^h = \sum_{n=1}^{n_{el}} N_j\, u_j \quad \in \mathcal{V}^h \tag{1.11}$$

$$\hat{u} \approx \hat{u}^h = \sum_{n=1}^{n_{el}} \hat{N}_j\, \hat{u}_j \quad \in \mathcal{M}^h(\Gamma \cup \Gamma_N \cup \Gamma_R) \text{ or } \mathcal{M}^h(\Gamma) \tag{1.12}$$

Then, a system of equation can be expressed in matrix form by using the interpolation chosen in 1.10, 1.11 and 1.12:

$$\begin{bmatrix} \mathbf{A_{uu}} & \mathbf{A_{uq}} \\ \mathbf{A_{uq}}^T & \mathbf{A_{qq}} \end{bmatrix}_i \begin{bmatrix} \mathbf{u_i} \\ \mathbf{q_i} \end{bmatrix} = \begin{bmatrix} \mathbf{f_u} \\ \mathbf{f_q} \end{bmatrix}_i + \begin{bmatrix} \mathbf{A_{u\hat{u}}} \\ \mathbf{A_{q\hat{u}}} \end{bmatrix}_i \hat{\mathbf{u}}_i \tag{1.13}$$

Similarly, applying the interpolation to (1.8) produces the following system of equations:

$$\sum_{n=1}^{n_{el}} \left\{ \begin{bmatrix} \mathbf{A}_{\mathbf{u\hat{u}}}^T & \mathbf{A}_{\mathbf{q\hat{u}}}^T \end{bmatrix}_i \begin{bmatrix} \mathbf{u_i} \\ \mathbf{q_i} \end{bmatrix} + [\mathbf{A_{\hat{u}\hat{u}}}]_i\, \hat{\mathbf{u}}_i + \left[\mathbf{A}_{\hat{\mathbf{u}}\hat{\mathbf{u}}}^R\right]_i \hat{\mathbf{u}}_i \right\} = \sum_{n=1}^{n_{el}} \left\{ [\mathbf{f_{\hat{u}}}]_i + \left[\mathbf{f}_{\hat{\mathbf{u}}}^R\right]_i \right\} \tag{1.14}$$

where matrices $\mathbf{A}_{\hat{\mathbf{u}}\hat{\mathbf{u}}}^R$ and $\mathbf{f}_{\hat{\mathbf{u}}}^R$ are associated to the Robin boundary condition of the problem and can be defined as:

$$\mathbf{A}_{\hat{\mathbf{u}}\hat{\mathbf{u}}}^R = -\frac{1}{\kappa} \sum_{\partial\Omega_i \cap \Gamma_R} \gamma \sum_{g=1}^{n_{ip}^f} \hat{\mathbf{N}}_{\mathbf{n}}(\xi_g^f) \hat{\mathbf{N}}^T(\xi_g^f)\, w_g^f \tag{1.15}$$

$$\mathbf{f}_{\hat{\mathbf{u}}}^R = -\frac{1}{\kappa} \sum_{\partial\Omega_i \cap \Gamma_R} \sum_{g=1}^{n_{ip}^f} \mathbf{N}(\xi_g^f)\, g(\mathbf{x}(\xi_g^f))\, w_g^f \tag{1.16}$$

After replacing the local solution (1.13) in (1.14), the global system becomes:

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

with

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

and

$$\hat{\mathbf{f}} = \mathop{\mathbin{\mathbb{A}}}_{i=1}^{n_{el}} [\mathbf{f}_{\hat{\mathbf{u}}}]_i + [\mathbf{f}_{\hat{\mathbf{u}}}^R]_i - \begin{bmatrix} \mathbf{A}_{\mathbf{u}\hat{\mathbf{u}}}^T & \mathbf{A}_{\mathbf{q}\hat{\mathbf{u}}}^T \end{bmatrix}_i \begin{bmatrix} \mathbf{A}_{\mathbf{uu}} & \mathbf{A}_{\mathbf{uq}} \\ \mathbf{A}_{\mathbf{uq}}^T & \mathbf{A}_{\mathbf{qq}} \end{bmatrix}_i^{-1} \begin{bmatrix} \mathbf{f}_{\mathbf{u}} \\ \mathbf{f}_{\mathbf{q}} \end{bmatrix}_i$$

2. *Implement in the Matlab code provided in class the corresponding HDG solver.*

In order to implement the HDG solver including the Robin boundary condition into the provided code, it was needed to understand how the mesh is created and also how the faces of each element are stored. Then, locating the correct numbering of elements and faces (in local and global form) a new matrix space similar with the Neuman BC is considered for the Robin BC. The code provide the internal and external faces of the elements, which these are important because the external plane $x = 1$ must be imposed as Neuman BC, and the plane $y = 0$ must be Robin BC, as the problem states. A subroutine was coded in which the entire external faces are the parameter, then a loop search for the requested planes and store them depending the boundary conditions (Code 1).

Code Implementation 1

```
for i = 1:length(infoFaces.extFaces)
    El_num = T(infoFaces.extFaces(i,1),1:3); %calls the in-edge nodes
    Nod_1 = X(El_num(1),:);
    Nod_2 = X(El_num(2),:);
    Nod_3 = X(El_num(3),:);
    if Nod_1(2) == 0 && Nod_2(2) == 0 || Nod_1(2) == 0 && Nod_3(2)==0 || Nod_3(2) == 0 &&
    Nod_2(2)==0  % y=0 (Neumann faces)
        infoFaces.extFaces_N(k,:) = infoFaces.extFaces(i,:);
        k = k+1;
    elseif Nod_1(1) == 1 && Nod_2(1) == 1 || Nod_1(1) == 1 && Nod_3(1)==1 || Nod_3(1) == 1 &&
    Nod_2(1)==1 % x=1 (Robin faces)
         infoFaces.extFaces_R(m,:) = infoFaces.extFaces(i,:);
         m=m+1;
    else
        infoFaces.extFaces_D(q,:) = infoFaces.extFaces(i,:);
        q = q+1;
    end
end
```

Listing 1: Boundary Conditions in External Faces.

Furthermore, the code provided in class uses a *Faces matrix F* in which it stores all the constrained and unconstrained faces of the domain. First, it stores the interior faces which are not constrained. Then a loop over the external faces is done, first stores the ones with “fluxes” boundary conditions and the last are the Dirichlet faces due to the solver does not consider this ones in the computation of the solution. Then, the Robin faces should be considered just before the Dirichlet faces as the code 2 shows.

Code Implementation 2

```
F = zeros(nOfElements,3);
for iFace = 1:nOfInteriorFaces %numbering of internal faces
    infoFace = intFaces(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace;
    F(infoFace(3),infoFace(4)) = iFace;
end

for iFace = 1:nOfExteriorFaces_N %numbering of Neuman faces
    infoFace = infoFaces.extFaces_N(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace + nOfInteriorFaces;
end

for iFace = 1:nOfExteriorFaces_R %numbering of Robin faces
    infoFace = infoFaces.extFaces_R(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace + nOfInteriorFaces + nOfExteriorFaces_N;
end

for iFace = 1:nOfExteriorFaces_D %Numbering of Dirichlet faces
    infoFace = infoFaces.extFaces_D(iFace,:);
    F(infoFace(1),infoFace(2)) = iFace + nOfInteriorFaces + nOfExteriorFaces_N +
    nOfExteriorFaces_R;
end
```

Listing 2: Including Robin BC in Faces Matrix F.

As a new boundary condition is imposed (Robin), it is needed to redefine the number of degrees of freedom that the solver will compute. In the previous version of the code, only the Neuman face provide degrees of freedom in the external boundary, due to the Dirichlet nodes are known values and are not considered in the solver as dofs. In this implementation, the Robin BC acts similarly to a Neuman face, then more degrees of freedom must be considered. The code 3 show the lines that ensure the redefinition of the dofs.

Code Implementation 3

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

dofDirichlet= (nOfInteriorFaces+nOfExteriorFaces_N+nOfExteriorFaces_R)*nOfFaceNodes + (1:
    nOfExteriorFaces_D*nOfFaceNodes);

dofUnknown = 1:(nOfInteriorFaces*nOfFaceNodes+nOfExteriorFaces_N*nOfFaceNodes+
    nOfExteriorFaces_R*nOfFaceNodes);
```

Listing 3: Degrees of freedom redefined by considering Robin BC.

The matrix computation and the corresponding force due the Robin boundary condition are implemented in the subroutine `hdg_preprocess.m`. As can be seen in the code 4, the lines corresponds to the equations 1.15 and 1.16 which denotes the matrix product of the shape functions, the Robin's function $g(\mathbf{x})$ and the paramerer κ included.

Code Implementation 4

```
%Robin matrix (All_R)
if Fext_R == 1
    nodes = faceNodes(face_R_id,:); Xf = Xe(nodes,:);
    dxdxi = Nx1d*Xf(:,1); dydxi = Nx1d*Xf(:,2);
    dxdxiNorm = sqrt(dxdxi.^2+dydxi.^2); dline = dxdxiNorm.*IPw_f';
    ind_face = (face_R_id-1)*nOfFaceNodes + (1:nOfFaceNodes);
    Auu_f = N1d'*(spdiags(dline,0,ngf,ngf)*N1d)*gamma;
    Arr(ind_face,ind_face) = -(1/kappa)*Auu_f;
end

%Neumann force vector
fqN = zeros(nOfFaces*nOfFaceNodes,1);
if Fext_N == 1
    nodes_N = faceNodes(face_N_id,:);
    Xf_N = Xe(nodes_N,:);
    dxdxi = Nx1d*Xf_N(:,1); dydxi = Nx1d*Xf_N(:,2);
    dxdxiNorm = sqrt(dxdxi.^2+dydxi.^2); dline = dxdxiNorm.*IPw_f';
    aux_f = -N1d'*(spdiags(dline,0,ngf,ngf)*neumanPoisson(N1d*Xf_N));
    if face_N_id == 1
        fqN(1:nodes_of_face) = aux_f;
    elseif face_N_id == 2
        fqN(nodes_of_face+1:2*nodes_of_face) = aux_f;
    else
        fqN(2*nodes_of_face+1:3*nodes_of_face) = aux_f;
    end
end

%Robin force vector
fqR = zeros(nOfFaces*nOfFaceNodes,1);
if Fext_R == 1
    nodes_R = faceNodes(face_R_id,:);
    Xf_R = Xe(nodes_R,:);
    dxdxi = Nx1d*Xf_R(:,1); dydxi = Nx1d*Xf_R(:,2);
    dxdxiNorm = sqrt(dxdxi.^2+dydxi.^2); dline = dxdxiNorm.*IPw_f';
    aux_f = -N1d'*(spdiags(dline,0,ngf,ngf)*robinPoisson(N1d*Xf_R));
    if face_R_id == 1
        fqR(1:nodes_of_face) = aux_f;
    elseif face_N_id == 2
        fqR(nodes_of_face+1:2*nodes_of_face) = aux_f;
    else
        fqR(2*nodes_of_face+1:3*nodes_of_face) = aux_f;
    end
end
```

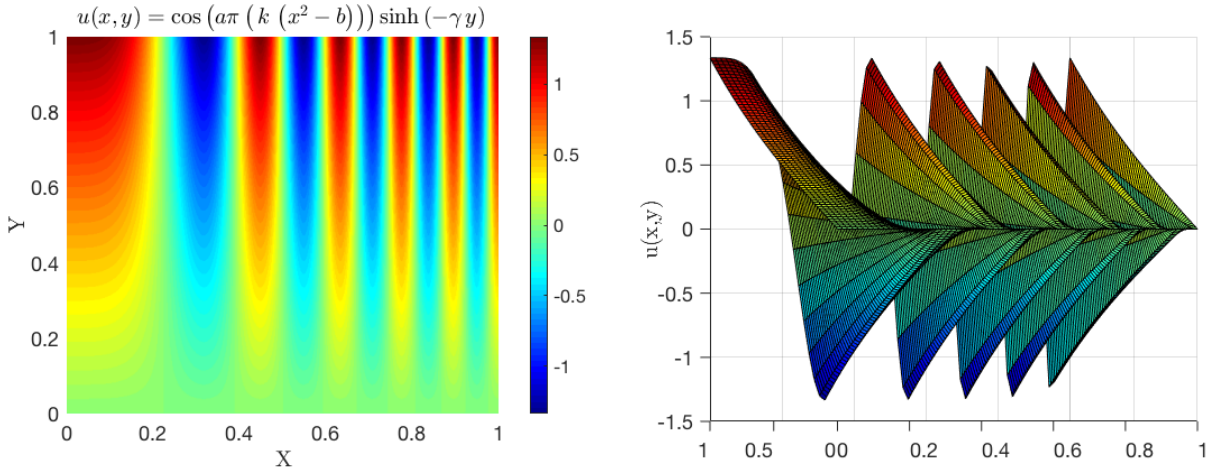
Listing 4: Robin matrix and force vector computation.

3. *Set* $\kappa = 2.5$ *and* $\lambda = 1.1$. *Consider* $u(x,y) = \cos\left(a\pi\left(k\left(x^2-b\right)\right)\right)\sinh\left(-\gamma\, y\right)$, *with* $a = 4$ *and* $b = 0.3$. *Determine the analytical expressions of the data* u_D, t *and* g *in problem (1.1). [Hint: Use Matlab tools for symbolic calculus]*

In order to introduce the given function u for this task, which corresponds to the #16 Assignment:

$$u(x,y) = \cos\left(a\pi\left(k\left(x^2-b\right)\right)\right)\sinh\left(-\gamma\, y\right) \tag{1.17}$$

Plotting this function, it is worthy to say that this specific function is a great challenge to approximate due to the high gradients that present, and the complexity of the function. (fig. 1.1).



(a) (b)

Figure 1.1: Function imposed to the boundary conditions of the problem #16.

Then, taking into account the *hint* given, using the Matlab symbolic calculus it was introduced the analytical expression 1.17, and computed the expression for Neuman t and for Robin g as:

Dirichlet:

$$u_D = \begin{cases} \sinh\left(1.1\, y\right) & \text{for } x = 0 \\ -1.33565 \cdot \cos\left(31.4159 x^2 - 9.42478\right) & \text{for } y = 1 \end{cases} \tag{1.18}$$

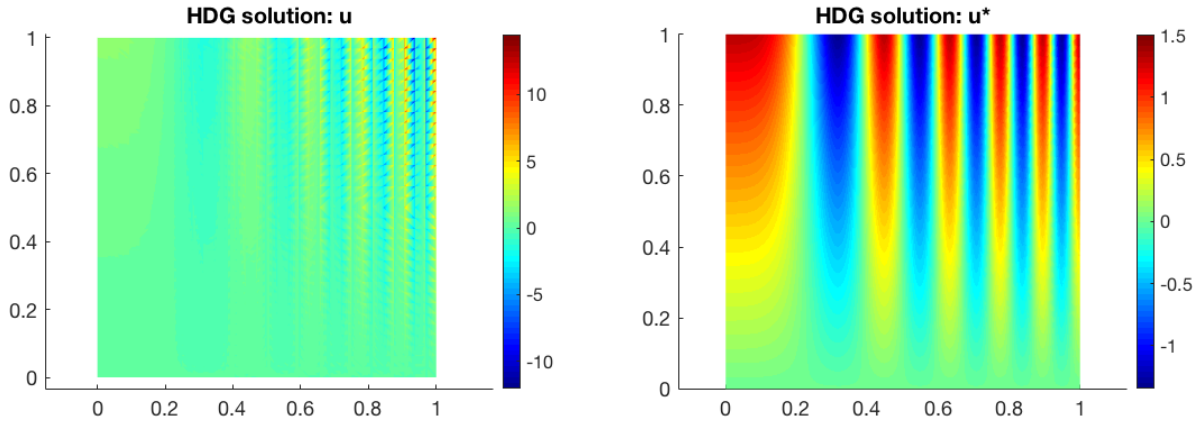
Neuman ($x = 1$):

$$t = 50\,\pi \sin\left(4\,\pi\left(\frac{5\,x^2}{2} - \frac{3}{4}\right)\right)\sinh\left(1.1\, y\right) x \tag{1.19}$$

Robin ($y = 0$):

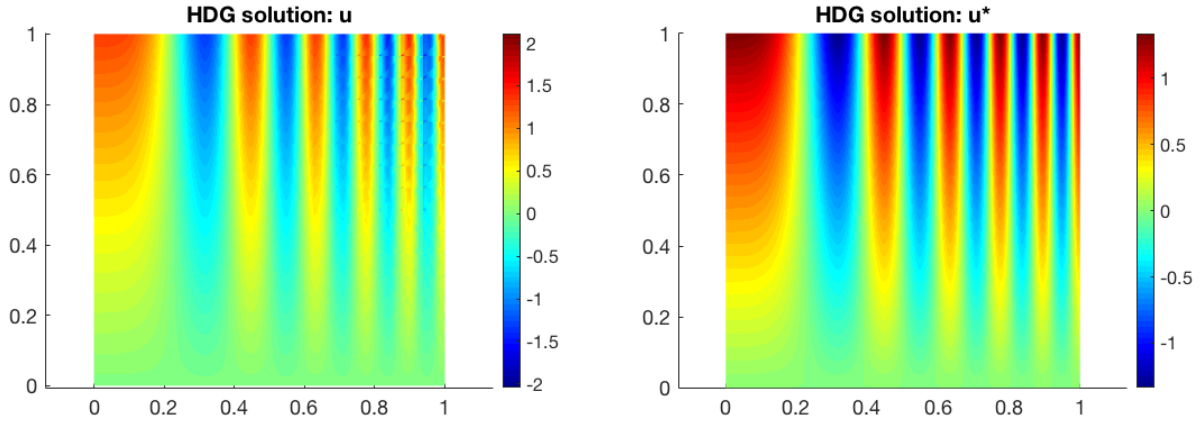
$$g = 1.1\cosh\left(\frac{11\, y}{4}\right)\cos\left(10\pi\left(x^2 - 0.3\right)\right) - 1.1\cos\left(10\,\pi\,\left(x^2 - 0.3\right)\right)\sinh\left(1.1\, y\right) \tag{1.20}$$

4. *Solve problem (1.1) 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 post-processed one u^*.*



(a) (b)

Figure 1.2: Comparison of the a) solution u and b) the postprocessed solution u^* using a linear approximation in a 2048 elements mesh.



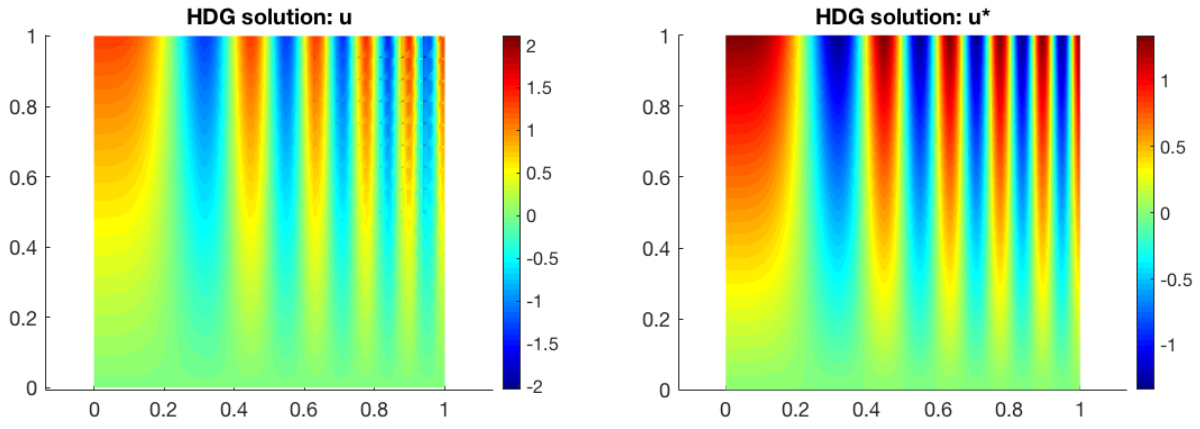
(a) (b)

Figure 1.3: Comparison of the a) solution u and b) the postprocessed solution u^* using an approximation of fourth order polynomial in a 512 elements mesh.

In order to compare the results provided by the HDG method, some experiments with different meshes and polynomial degrees of approximation were performed, some of the most relevant are shown next. First, a linear approximation using a fine mesh with 2048 elements is tested, as can be seen in figure 1.2a, the graph does not look similar to the analytical solution of figure 1.1, even though the error computed using the $\mathcal{L}_2$ norm is $7.606669 \cdot (10^{-1})$. Nevertheless, the post-processed solution u^* is incredible accurate because it has an error of $8.298004 \cdot (10^{-3})$ by using linear elements, and as can be seen in figure 1.2b it is closer to the analytical solution. Now, using a discretization with less elements (512), but a higher polynomial degree of approximation ($p = 4$), a comparison of the results can be discussed. The figure 1.3a shows the solution

u, in which is evident a better approximation by using less elements but polynomial of degree 4, the graph is quite similar with the analytical solution and the error is $3.426272 \cdot (10^{-2})$ which is a very good improvement compared with the previous result using a dense mesh. Moreover, the result of the postprocess result is even better than before, with an error of $2.771389 \cdot (10^{-4})$.

A last comparison is done by using a finer mesh and a polynomial approximation of degree 4 (fig. 1.4), in which it can be seen the best approximation that can be obtained in this work. The error of the solution with the $\mathcal{L}_2$ norm is $3.460436 \cdot (10^{-5})$ and the postprocess error is $6.663120 \cdot (10^{-8})$, which is almost the analytical result.



(a) (b)

Figure 1.4: Comparison of the a) solution u and b) the postprocessed solution u^* using a linear approximation in a 8192 elements mesh.

5. *Compute the errors for u, q, and u^* in the $\mathcal{L}_2$-norm defined in the domain Ω. Perform a convergence study for the primal, u, mixed, q and post-processed, u^* 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.*

To prove the behavior of the numerical results comparing the polynomial approximation degree some experiments were carried out by using the same mesh of 512 elements, the figure 1.5a shows the solution u with linear approximation, it shows an error of 2.413887 with the $\mathcal{L}_2$ norm, and it can be seen that the approximation is not as good as the $4th$ degree polynomial solution with an error of $3.426272 \cdot (10^{-2})$ of the figure 1.5b. And also, the flux $\mathbf{q}$ computed by HDG methodology is shown in the figures 1.5c and 1.5d for the X-component using the linear and fourth degree approximation respectively. It is verified that the approximation using linear elements lacks of accuracy as well, in contrast the fourth degree polynomial approximation is more adequate. The same conclusion can be done in the Y-component of the flux, in which is more evident that higher polynomials have an impressive impact in the HDG results.

Then, a comparison between post-processed solutions is shown in the figure 1.6 using the same mesh of 512 elements but different approximation polynomial, where even though it is obvious that using a higher degree of polynomial the results are getting better, the linear approximation (fig. 1.6a) is quite accurate and incredible superior than the solution itself (fig. 1.5a), which indicates the importance of the implementation of the postprocess.

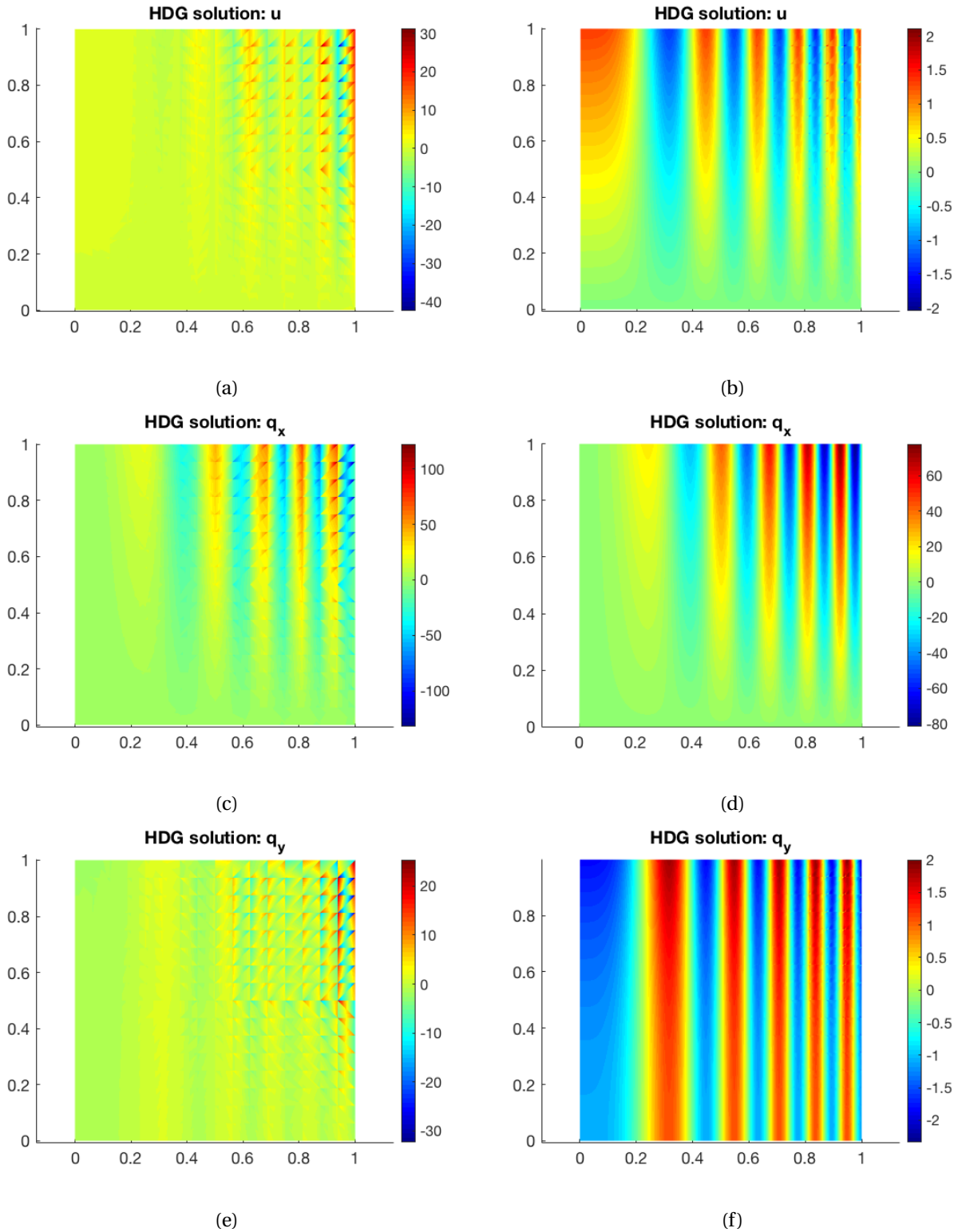
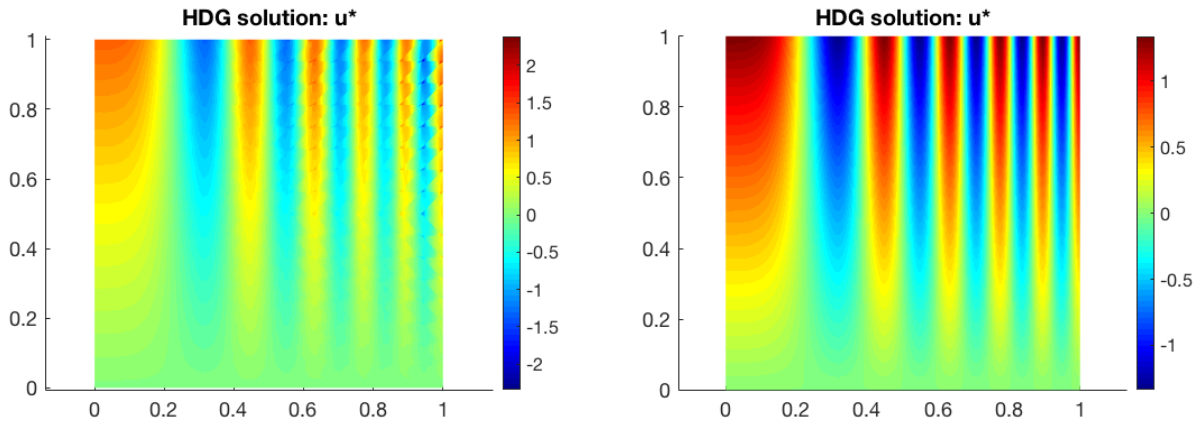


Figure 1.5: Comparison between polynomial degree 1 (left) and 4 (right) using the same discretization with 512 elements.



(a) (b)

Figure 1.6: Comparison of the postprocess solution u^* with a) $p = 1$ degree ($error = 6.102481 \cdot (10^{-2})$) and b) $p = 4$ degree ($error = 2.771389 \cdot (10^{-4})$) of approximation in a 512 elements mesh.

Furthermore, a comparison between the evolution of the refinement in the mesh and the degrees of the polynomials can be done through a convergence analysis. The figure 1.7 shows a graph in which the colors represent the polynomial degree, and as can be seen as the element is getting small, the error diminishes which is logical, but one of the extraordinary improvements of the HDG methodology resides in the post-process computation, where the solution u^* (dashed lines) have a clearly gain of accuracy. For example the finest mesh with degree of approximation 4 corresponds to the solid line (—), and the postprocessed solution of degree p=2 (– –) obtains the same result, then it can be said that the postprocess is two times more accurate than the solution computed. Moreover, the results obtained for this problem show that the optimal (i.e., p + 1 for the solution and p + 2 for the postprocessed solution) rate of convergence is obtained.

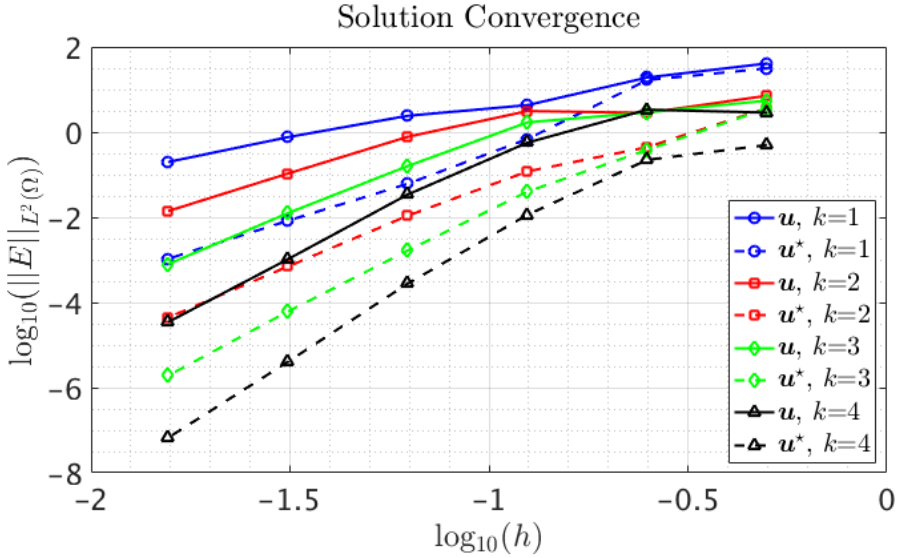


Figure 1.7: Convergence analysis with discretization meshes from 8 elements to 8192, and using polynomial degrees from 1 to 4: p=1(—), p=2 (—), p=3 (—), p=4 (—).

REFERENCES

[1] Ruben Sevilla. Hybridisable discontinuous galerkin for second-order elliptic problems. notes for dg summer school - barcelona, 2017.

[2] Ruben Sevilla and Antonio Huerta. *Tutorial on Hybridizable Discontinuous Galerkin (HDG) for Second-Order Elliptic Problems*, pages 105–129. 05 2016.