Màster en Mètodes Numèrics **Aitor Bazán Escoda**

Universitat Politècnica de Catalunya 06/06/19

Finite Elements in Fluids

Homework 7: Hybridizable Discontinuous Galerkin

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

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

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

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

where κ and γ are the diffusion and convection coefficients, respectively, **n** is the outward unit normal vector to the boundary, s is a volumetric source term and u_D, t and g, 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 strong and weak forms of the local and global problems.

   We define two equivalent problems:

First, Element-by-element problem (Local- Dirichlet).

$$1. \begin{cases} \nabla\cdot\boldsymbol{q}_i = s & in\ \Omega_i \\ \boldsymbol{q}_i + \kappa\nabla u_i = 0 & in\ \Omega_i \\ u_i = u_D & in\ \partial\Omega_i \cap \Gamma_D \\ u_i = \hat{u} & on\ \partial\Omega_i \setminus \Gamma_D \end{cases}$$

Second, A global problem to determine $\hat{u}$ (Neumann-Robin Transmission conditions) (To ensure inter-element continuity when the broken comp. domain).

$$2. \begin{cases} [\![\boldsymbol{n}\cdot\boldsymbol{q}]\!] = 0 & on\ \Gamma \\ \boldsymbol{n}\cdot\boldsymbol{q} = -t & on\ \Gamma_N \\ -\boldsymbol{q}\cdot\boldsymbol{n} + \gamma u = g & on\ \Gamma_R \end{cases}$$

From now on, local problem will be called as 1 and global problem as 2.

For 1, it yields the following:

$$-(\nabla v, q_i)_{\Omega_i} + \langle v, n_i\cdot\hat{q}_i\rangle_{\partial\Omega_i} = (v,f)_{\Omega_i}$$
$$-(w, q_i)_{\Omega_i} + (\nabla\cdot w, u_i)_{\Omega_i} = \langle u_i\cdot w, u_D\rangle_{\partial\Omega_i\cap\Gamma_D} + \langle n_i\cdot w, \hat{u}\rangle_{\partial\Omega_i\setminus\Gamma_D}$$
$$n_i\cdot\hat{q}_i := \begin{cases} n_i\cdot q_i + \tau_i(u_i - u_D) & \text{on } \partial\Omega_i\cap\Gamma_D \\ n_i\cdot q_i + \tau_i(u_i - \hat{u}) & \text{elsewhere} \end{cases}$$

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

For 2, it yields,

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

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

it leads to,

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

Now, by integrating by parts the first term of the LHS of the equation for the global problem and leaving on the boundary of the element values of the flux ***q,*** in the interior,

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

Therefore, by Galerkin F.E. approximation,

$$-\left(v, \nabla \cdot q_i^h\right)_{\Omega_i} + \langle v, \tau_i u_i^h \rangle_{\partial\Omega_i} = (v, f)_{\Omega_i} + \langle v, \tau_i u_D \rangle_{\partial\Omega_i \cap \Gamma_D} + \langle v, \tau_i \hat{u}^h \rangle_{\partial\Omega_i \setminus \Gamma_D}$$
$$-\left(w, q_i^h\right)_{\Omega_i} + \left(\nabla \cdot w, u_i^h\right)_{\Omega_i} = \langle n_i \cdot w, u_D \rangle_{\partial\Omega_i \cap \Gamma_D} + \langle n_i \cdot w, \hat{u}^h \rangle_{\partial\Omega_i \setminus \Gamma_D}$$

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

After the discretization of the global and local problems, it is obtained the following system of equations,

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

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

After replacing the first system of equations into the second equation it yileds,

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

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

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

Note that term $\mathbf{A}_{\hat{u}\hat{u}}$ involves Robin conditions whereas the term $\mathbf{f}_{\hat{u}}$ computes both Neumann and Robin boundary conditions.

Refer to references stated in the assignment to check terms involving the above calculus.

3. Implement in the Matlab code provided in class the corresponding HDG solver.
    1. According to the global Face ID of the Dirichlet boundaries it is extracted the degrees of freedom of the unknowns.
    2. So as to gather the types of faces, since the problem differentiates among Dirichlet, Robin and Neumann boundary conditions, we have added an extra cell to extFaces.
    3. In hdgMatrixPoisson, the matrix of the global system needs to be now computed adding the extra terms so as to include Neumann and Robin. (*see appendix).*
    4. Also, in the postprocess we should add the viscosity term.

4. Set $\kappa = 1.1$ and $\gamma = 0.9$. Consider $\mathrm{u(x,y) = cosh(\kappa x - \gamma y) + sin(\pi(ax + by))}$, with $a = 4$ and $b = 2$. Determine the analytical expressions of the data u_D, t and g in the problem (P).

The analytical expressions for u_D is the same as the analytical solution. Whereas for Neumann, Robin and the source term, we have:

```
syms a b kappa gamma x y

u = cosh(kappa*x - gamma*y) + sin(pi*(a*x + b*y));

dudx = diff(u,x);
dudy = diff(u,y);

q = -kappa*[dudx;dudy];
t = -kappa*dudx;
g = kappa*dudx + gamma*u;
s = -kappa*(diff(dudx,x)+diff(dudy,y));
```

$$\mathrm{s = -\kappa \cdot \left(\gamma^2 \cosh(\gamma y - \kappa x) + \kappa^2 \cosh(\gamma y - \kappa x) - a^2\pi^2 \sin\big(\pi(ax + by)\big) - b^2\pi^2 sin(pi(ax + by))\right)}$$

$$\mathrm{t = \kappa \cdot (\kappa \cdot \sinh(\gamma y - \kappa x) - \pi\, acos\big(\pi(ax + by)\big)\,)}$$

$$\mathrm{g = \ \gamma \cdot \big(\cosh(\gamma y - \kappa x) + \sin\big(\pi(ax + by)\big)\big) - \kappa \cdot (\kappa \sinh(\gamma y - \kappa x) - \pi\, acos\big(\pi(ax + by)\big)\big)}$$

5. 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^*.

Mesh 5 – Polynomial Degree 1

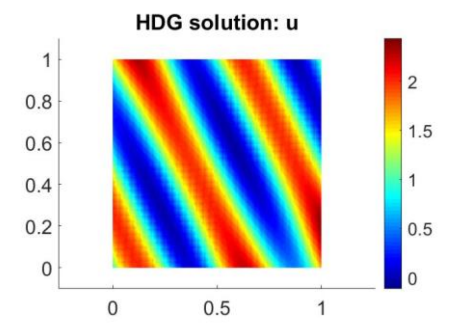

Figure 1. HDG u solution. Mesh 5, Degree 1.

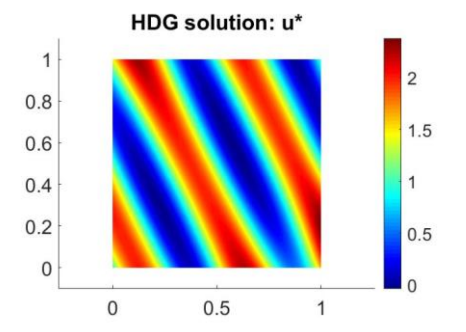

Figure 2. HDG u* solution. Mesh 5, Degree 1.

Mesh 5 – Polynomial Degree 2

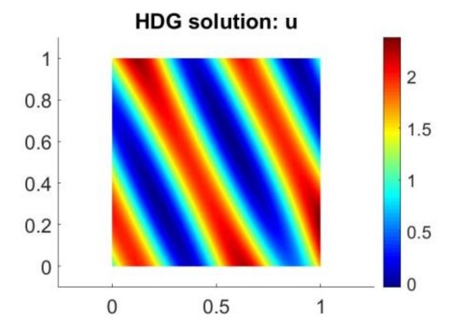

Figure 3. HDG u solution. Mesh 5, Degree 2.

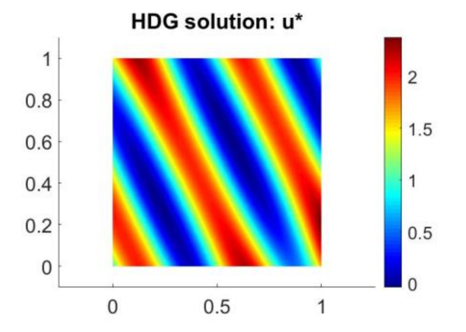

Figure 4. HDG u* solution. Mesh 5, Degree 2.

Mesh 3 – Polynomial Degree 1

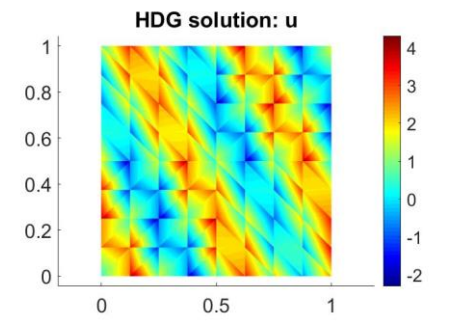

Figure 5. HDG u solution. Mesh 3, Degree 1.

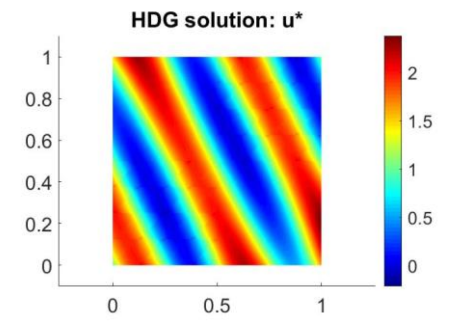

Figure 6. HDG u* solution. Mesh 3, Degree 1.

Mesh 3 – Polynomial Degree 2

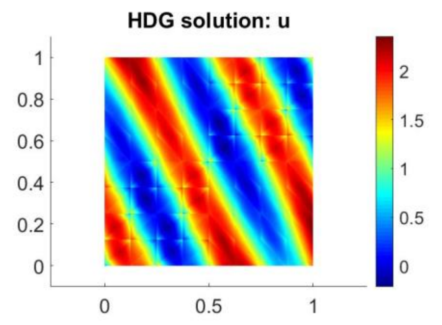

Figure 7. HDG u solution. Mesh 3, Degree 2.

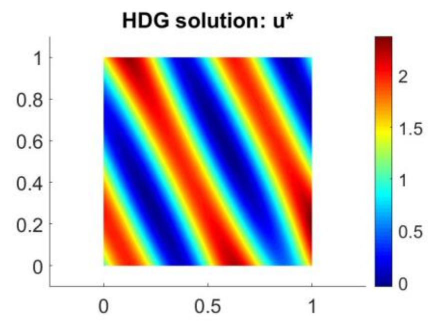

Figure 8. HDG u* solution. Mesh 3, Degree 2.

Mesh 2 – Polynomial Degree 3

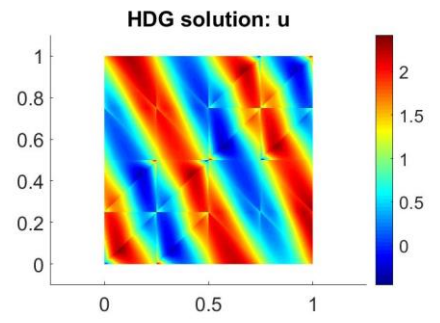

Figure 9. HDG u solution. Mesh 2, Degree 3.

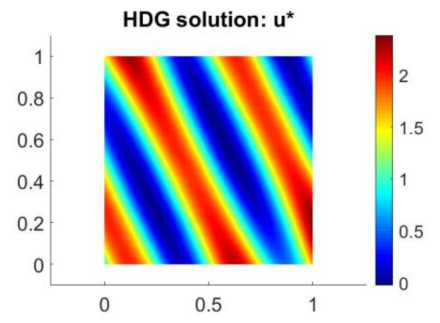

Figure 10. HDG u* solution. Mesh 2, Degree 3.

Mesh	**Degree**	$\left\|\boldsymbol{u}-\boldsymbol{u^h}\right\|_{\mathcal{L}_2(\Omega)}$	$\left\|\boldsymbol{u}^*-\boldsymbol{u}^{*\boldsymbol{h}}\right\|_{\mathcal{L}_2(\Omega)}$
5	1	5.647363e-02	4.579051e-02
5	2	4.579945e-02	4.578149e-02
3	1	4.915640e-01	5.058000e-02
3	2	8.921951e-02	4.586373e-02
6	4	0.045781497311794	0.045781497300583
2	3	1.373176e-01	4.644098e-02

Table 1. L2norm computation for HDG u solution and postprocessed.

As it is observed, the error of the variables decrease as long as the degree of the approximation is increased or as we refine the mesh.

Also as a minor remark, the accuracy that the postprocessed solution u* prompts out is higher than the one obtained for the solution for the same mesh and degree of approximation.

6. Compute the errors for u, $\boldsymbol{q}$ and u^* 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^* variables for a polynomial degree of approximation $k = 1, \dots, 4$. Discuss the obtained numerical results, starting from the theoretical results on the optimal convergence rates of HDG.

The figures shown below, depict the convergence of the $\mathcal{L}_2$-norm error of u, $\boldsymbol{q}$ and u^* for a range of polynomial degree of approximation going from 1 to 4. The convergence rate optimal (k+1) for u and q is obtained. The superconvergence rate (k+2) is obtained for the postprocessed solution u*.

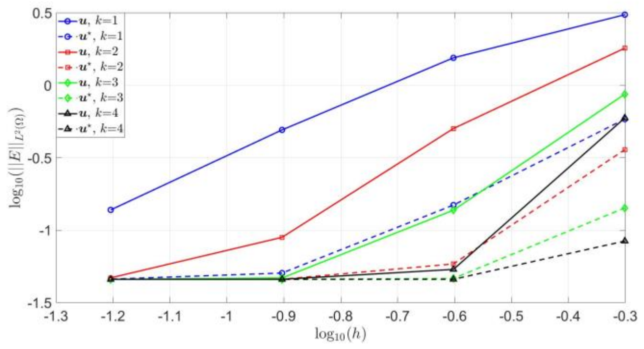

Figure 11. Convergence analysis for u and u* with p=1,…,4. Mesh 4.

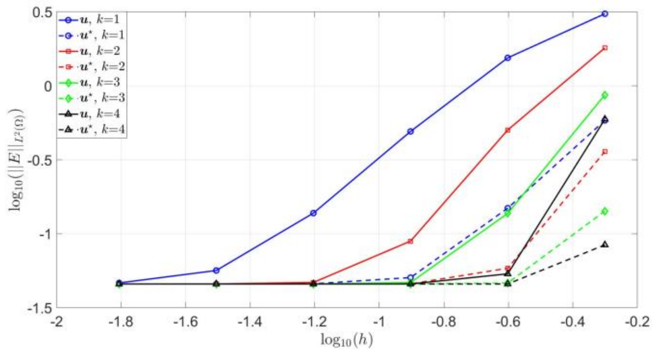

Figure 12. Convergence analysis for u and u* with p=1,…,4. Mesh 6.

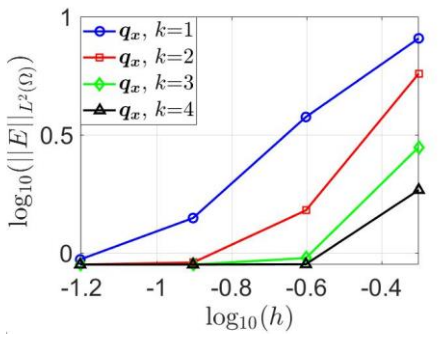

Figure 13. Convergence analysis for component x of q with p=1,…,4. Mesh 4.

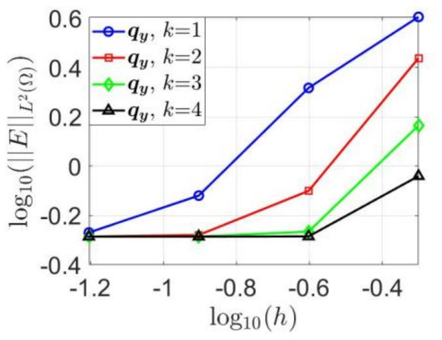

Figure 14. Convergence analysis for component y of q with p=1,…,4. Mesh 4.

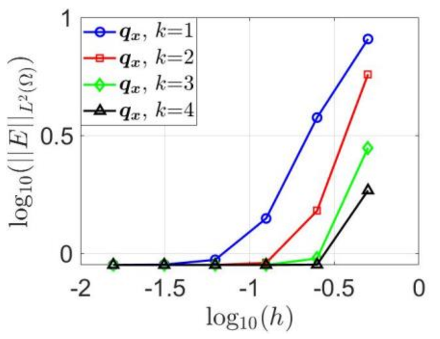

Figure 15. Convergence analysis for component x of q with p=1,…,4. Mesh 6.

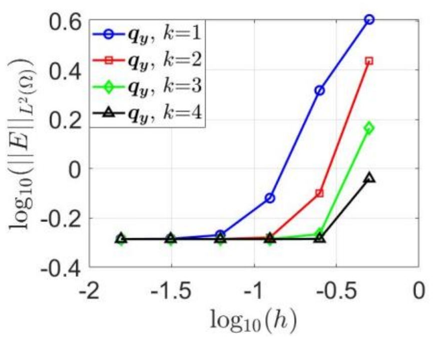

Figure 16. Convergence analysis for component y of q with p=1,…,4. Mesh 6.

APPENDIX

```
function [KK, f, QQ, UU, Qf, Uf] =
hdgMatrixPoisson(muElem,X,T,F,referenceElement,infoFaces,tau)
```

```
nOfFaces = max(max(F));
nOfElements = size(T,1);
nOfInteriorFaces = size(infoFaces.intFaces,1);
nOfFaceNodes = size(referenceElement.NodesCoord1d,1);
nDOF = nOfFaces*nOfFaceNodes;
f = zeros(nDOF,1);
QQ = cell(nOfElements,1); UU = cell(nOfElements,1); Qf = cell(nOfElements,1); Uf =
cell(nOfElements,1);

% loop in elements
indK = 1; n = nOfElements*(3*nOfFaceNodes)^2; ind_i  = zeros(1,n); ind_j  = zeros(1,n);
coef_K = zeros(1,n);
for iElem = 1:nOfElements
    Te = T(iElem,:);
    Xe = X(Te,:);
    Fe = F(iElem,:);
    isFeInterior = (Fe <= nOfInteriorFaces); %Boolean (1=interior face or Neuman)

    [Fext_N,ind_N] = ismember(iElem,infoFaces.extFaces_N(:,1));
    if Fext_N ==1
        face_N_id = infoFaces.extFaces_N(ind_N,2);
    else
        face_N_id =0;
    end

    [Fext_R,ind_R] = ismember(iElem,infoFaces.extFaces_R(:,1));
    if Fext_R ==1
        face_R_id = infoFaces.extFaces_R(ind_R,2);
    else
        face_R_id =0;
    end


    % elemental matrices
    [Qe,Ue,Qfe,Ufe,Alq,Alu,All,fqN,fqR,Arr] =
KKeElementalMatricesIsoParametric(muElem(iElem),Xe,referenceElement,tau(iElem,:),Fext_N,
face_N_id,Fext_R,face_R_id);

    % Interior faces seen from the second element are flipped to have
    % proper orientation
    flipFace = zeros(1,3); %Boolean (1=face to be flipped)
    flipFace(isFeInterior)=any(infoFaces.intFaces(Fe(isFeInterior),[1 3])<iElem,2);
    indL=1:3*nOfFaceNodes;
    aux=nOfFaceNodes:-1:1; indflip=[aux,nOfFaceNodes+aux,2*nOfFaceNodes+aux];
    aux=ones(1,nOfFaceNodes); aux = [flipFace(1)*aux, flipFace(2)*aux, flipFace(3)*aux];
    indL(aux==1)=indflip(aux==1); %permutation for local numbering

    Qe=Qe(:,indL);    Ue=Ue(:,indL);
    Alq=Alq(indL,:);  Alu=Alu(indL,:);   All=All(indL,indL);
    Arr = Arr(indL,indL);

    %The local problem solver is stored for postprocess
    QQ{iElem} = sparse(Qe);  UU{iElem} = sparse(Ue);
    Qf{iElem} = sparse(Qfe); Uf{iElem} = sparse(Ufe);

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

    aux = (1:nOfFaceNodes);
    indRC = [(Fe(1)-1)*nOfFaceNodes + aux,(Fe(2)-1)*nOfFaceNodes + aux,(Fe(3)-
1)*nOfFaceNodes + aux];
    f(indRC) = f(indRC) + ffe;
    for irow = 1:(3*nOfFaceNodes)
        for icol = 1:(3*nOfFaceNodes)
            ind_i(indK)  = indRC(irow); ind_j(indK)  = indRC(icol);
            coef_K(indK) = KKe(irow,icol); indK = indK+1;
        end
    end
end
KK = sparse(ind_i,ind_j,coef_K);
```

```
function [Q,U,Qf,Uf,Alq,Alu,All,fqN,fqR,Arr] =
KKeElementalMatricesIsoParametric(mu,Xe,referenceElement,tau,Fext_N,face_N_id,Fext_R,fac
e_R_id)
```

```
nOfElementNodes = size(referenceElement.NodesCoord,1);
nOfFaceNodes = size(referenceElement.NodesCoord1d,1);
faceNodes = referenceElement.faceNodes;
nOfFaces = 3; %triangles
nodes_of_face = size(referenceElement.faceNodes,2);

% Information of the reference element
N = referenceElement.N;
Nxi = referenceElement.Nxi; Neta = referenceElement.Neta; %elemental
N1d = referenceElement.N1d; Nx1d = referenceElement.N1dxi; %face
%Numerical quadrature
IPw_f = referenceElement.IPweights1d; ngf = length(IPw_f);
IPw = referenceElement.IPweights; ngauss = length(IPw);

%%Volume computations
% Jacobian
J11 = Nxi*Xe(:,1); J12 = Nxi*Xe(:,2);
J21 = Neta*Xe(:,1); J22 = Neta*Xe(:,2);
detJ = J11.*J22-J12.*J21;
%maybe we should use bsxfun instead of diagonal matrices...
dvolu = spdiags(referenceElement.IPweights.*detJ,0,ngauss,ngauss);
invJ11 = spdiags(J22./detJ,0,ngauss,ngauss);
invJ12 = spdiags(-J12./detJ,0,ngauss,ngauss);
invJ21 = spdiags(-J21./detJ,0,ngauss,ngauss);
invJ22 = spdiags(J11./detJ,0,ngauss,ngauss);
% xy-derivatives
Nx = invJ11*Nxi + invJ12*Neta;
Ny = invJ21*Nxi + invJ22*Neta;

%Computation of r.h.s. source term (analytical laplacian)
Xg = N*Xe;
sourceTerm = sourcePoisson(Xg,mu);
fe = N'*(dvolu*sourceTerm);

%Elemental matrices
Me = N'*(dvolu*N);
```

```matlab
Aqq = zeros(2*size(Me));
aux = 1:2:2*nOfElementNodes; aux2 = 2:2:2*nOfElementNodes;
Aqq(aux,aux)=Me; Aqq(aux2,aux2)=Me; %Aqq
Auq = zeros(nOfElementNodes,2*nOfElementNodes); %Auq
Auq(:,aux) = N'*(dvolu*Nx); %x derivatives & 1st component of q
Auq(:,aux2)= N'*(dvolu*Ny); %y derivatives & 2nd component of q
```

```matlab
Alq = zeros(3*nOfFaceNodes,2*nOfElementNodes);
Auu = zeros(nOfElementNodes,nOfElementNodes);
Alu = zeros(3*nOfFaceNodes,nOfElementNodes);
All = zeros(3*nOfFaceNodes,3*nOfFaceNodes);
Arr = 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

gamma = 0.7;
kappa = 0.75;

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

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)*Neuman(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

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)*Robin(N1d*Xf_R));

    if face_R_id == 1
        fqR(1:nodes_of_face) = aux_f;
    elseif face_R_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
% 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
```