Pilot Testing: The HDG applied to a rotational Equation

Exploring and develop of the pilot example

A short overview by *@alujan*

```
clear
format short
syms x y z real
```

```
% Let's add to the path the src and drivers: Original Article
addpath('/Users/alujan/UN/HDG/949_Sayas/Matlab/Drivers/')
addpath('/Users/alujan/UN/HDG/949_Sayas/Matlab/Src/')

% Let's add the path of the gmsh structure: Load Mesh structure
addpath('/Users/alujan/UN/HDG/gmsh')

% Let's add the path of the structures created: Curl definitions
addpath('/Users/alujan/UN/HDG/949_Sayas/Matlab')
```

Load of structures for analisys

Let's load the reference structres we're going to use to set the pilot example

```
% Importing of a singular structure for first case scenario: Unitary
cube
T = load_mesh('cubeD1.msh')
% Listing of elements for growth of method testing
listT = {T};
```

Definitions of reference solutions

First let's define the values we're going to compare with, in this case the functions u and z, defined for the auxiliar equation:

$$z - \nabla \times u = 0 \text{ in } \Omega$$

$$u + \nabla \times z = f \text{ in } \Omega$$

$$u \times n = g \times n \; on \; \partial\Omega$$

Take in mind this a vectorial PDE, for this scenario every evaluation of the values must be in components of the vector

```
% Let's define a simple value for U: harmonic vector
Ux = x^4 + 2*x*y - z^4; Uy = (x - z)^4 * (x + y)^4; Uz = y^2 - 2*x*y^3
+ z^4;

U(x, y, z) = [Ux; Uy; Uz]
% Partial derivarives of the component
Uxy = diff(Ux, y); Uxz = diff(Ux, z);
Uyx = diff(Uy, x); Uyz = diff(Uy, z);
```

```
Uzx = diff(Uz, x); Uzy = diff(Uz, y);

% Definition of auxiliar variable: Z
Zx = Uzy - Uyz; Zy = Uxz - Uzx; Zz = Uyx - Uxy;

Z(x, y, z) = [Zx; Zy; Zz]
% Partial derivarives of the component
Zxy = simplify(diff(Zx, y)); Zxz = simplify(diff(Zx, z));
Zyx = simplify(diff(Zy, x)); Zyz = simplify(diff(Zy, z));
Zzx = simplify(diff(Zz, x)); Zzy = simplify(diff(Zz, y));

% Definition of curl vector of Z: curlZ
curlZx = Zzy - Zyz; curlZy = Zxz - Zzx; curlZz = Zyx - Zxy;

curlZ = [curlZx; curlZy; curlZz]
% Definition of forcement: F
Fx = Ux + curlZx; Fy = Uy + curlZy; Fz = Uz + curlZz;

F(x, y, z) = [Fx; Fy; Fz]
% Definition of g element: Dirichlet condition G
Gx = Ux; Gy = Uy; Gz = Uz;

G(x, y, z) = [Gx; Gy; Gz]
% Postprocessing of the resultant values
ux = matlabFunction(Ux, Vars={x, y, z});
uy = matlabFunction(Uy, Vars={x, y, z});
uz = matlabFunction(Uz, Vars={x, y, z});

fx = matlabFunction(Fx, Vars={x, y, z});
fy = matlabFunction(Fy, Vars={x, y, z});
fz = matlabFunction(Fz, Vars={x, y, z});

gx = matlabFunction(Gx, Vars={x, y, z});
gy = matlabFunction(Gy, Vars={x, y, z});
gz = matlabFunction(Gz, Vars={x, y, z});

zx = matlabFunction(Zx, Vars={x, y, z});
zy = matlabFunction(Zy, Vars={x, y, z});
zz = matlabFunction(Zz, Vars={x, y, z});
```

Definition of Cuadrature and Polynomial degrees

Now for the first definitions for the method, we must import the cuadrature that's going to be used, and the polynomial degree that's going to be used for the solution

```
% It's possible to change the values below to alter the polynomial and
% cuadrature precision for more general viewing
TablesQuadForm3d        % Importing of cuadrature for thetrahedron
TablesQuadForm          % Importing of cuadrature for triangles

k = 1;  % Polynomial Degree

% Switch case for the selection of formulas: cuadratures
```

```
switch k
    case 0
        formulas={tetra3,tetra1,matrix0,matrix4};
    case 1
        formulas={tetra5,tetra3,matrix4,matrix9};
    case 2
        formulas={tetra7,tetra5,matrix9,matrix11};
    case 3
        formulas={tetra9,tetra7,matrix11,matrix14};
    case 4
        formulas={tetra9,tetra9,matrix14,matrix16};
end

% Reduction of weights: Sum of weights it's defined to be equal to 2,
for
% normalization it's reduced to 1.
formulas{3}(:,4)=formulas{3}(:,4)/2;
formulas{4}(:,4)=formulas{4}(:,4)/2;
```

Norm of solutions for error definitions

The `errorElem` function it's used for this task, the idea used it's comparison with the origin that leaves the norm of elements $u$and $q$with known solution

$$|p - p^*|_{L^2} = |p|_{L^2} : p^* = 0$$

```
% Error elements definition for display handle
ErrorU=[];
ErrorQ=[];
ErrorUhat=[];
ErrorPu=[];
ErrorPuhat=[];
ErrorUstar=[];
h=[];

% Norms of unknowns for relative error
Tmax=listT{end};                    % Listing of T elements
Nelts=size(Tmax.elements,1);        % Number of elements enlisted
d3=nchoosek(k+3,3);                 % d_i definition of set
(cardinality)

% Computation of norm for U & G
normU = errorElem(Tmax,ux,zeros(d3,Nelts),k,formulas{1})...
      + errorElem(Tmax,uy,zeros(d3,Nelts),k,formulas{1})...
      + errorElem(Tmax,uz,zeros(d3,Nelts),k,formulas{1});

normG = errorElem(Tmax,gx,zeros(d3,Nelts),k,formulas{1})...
      + errorElem(Tmax,gy,zeros(d3,Nelts),k,formulas{1})...
      + errorElem(Tmax,gz,zeros(d3,Nelts),k,formulas{1});
```

HDG: The main iteration

Firstly the method it's created to run in a single run *(presence of one element only)*, with this in mind, when going to expand the idea must be iterated over the different structures

```
% Definition of the action of tau: Identity
tau = ones(4,Nelts);

% -------------------------------- HDG 3D main function
--------------------------------------- %

% Definitions asociated with the tetrahedrization
d2 = nchoosek(k+2,2);
d3 = nchoosek(k+3,3);

block3 = @(x) (1+(x-1)*d3):(x*d3);

Nelts  = size(T.elements,1);
Nfaces = size(T.faces,1);
Ndir   = size(T.dirichlet,1);
Nneu   = size(T.neumann,1);

% 4th intervention

face = T.facebyele';                    % 4 x Nelts
face = (face(:)-1)*2*d2;                % First degree of freedom of
each face by element
face = bsxfun(@plus,face,1:2*d2);       % 4*Nelts x d2 (d.o.f. for
each face)
face = reshape(face',4*2*d2,Nelts);     % d.o.f. for the 4 faces of
each element

[J,I] = meshgrid(1:4*2*d2);

R = face(I(:),:); R=reshape(R,4*2*d2,4*2*d2,Nelts);   % Mesh grid
definition inside T_i element "Row-Axis"
C = face(J(:),:); C=reshape(C,4*2*d2,4*2*d2,Nelts);   % Mesh grid
definition inside T_i element "Column-Axis"

% R_ij^K d.o.f. for local (i,j) d.o.f. in element K ; R_ij^K=C_ji^K
RowsRHS = reshape(face,4*2*d2*Nelts,1);            % Definition of
solution shape

% 4th intervention

% Dirichlet faces: defintion
dirfaces = (T.dirfaces(:)-1)*2*d2;          % First degree of freedom
of each face by element
dirfaces = bsxfun(@plus,dirfaces,1:2*d2);   % Bitwise sum of dirfaces
and 1:d2 > dirfaces + 1:d2
dirfaces = reshape(dirfaces',2*d2*Ndir,1);  % Reshape of dirfaces

% Reduced version of free
free = (1: 2*d2*Nfaces);
```

```
% Cleaning of dirfaces: Positions not needed
free(dirfaces) = [];      % Empty of dirfaces positions on free

% Neumann faces: defintion
neufaces = (T.neufaces(:)-1)*2*d2;          % First degree of freedom
of each face by element
neufaces = bsxfun(@plus,neufaces,1:2*d2);   % Bitwise sum of neufaces
and 1:d2 > neufaces + 1:d2
neufaces = reshape(neufaces',2*d2*Nneu,1);

% Preprocessing step: Normalization of normals:
% From the structure we define the normals as a proportion of the area,
% We're removing this step and normalizing this structure
normals   = T.normals
normals   = reshape(normals', [3, 4*Nelts])
normals   = normals ./ sqrt(sum(normals.^2))
normals   = reshape(normals, [3*4, Nelts])
T.normals = normals';
```

Now it's the point to use the local solvers, here are defined the change of basis and the change on the matrices for calculations needed.

```
% ----------------------------- Local solvers definition function
------------------------------ %

% Test element \eta on f: element-wise of the vector
fx_test = testElem(fx,T,k,formulas{1});
fy_test = testElem(fy,T,k,formulas{1});
fz_test = testElem(fz,T,k,formulas{1});

Af=zeros(6*d3,Nelts);           % Storage reservation (6 x d3 x Nelts)

% Storage of vector with impact on the U equation:
Af((0 + 3)*d3+1:(0 + 1 + 3)*d3,:) = fx_test;    % Impact on u -> x
coordinate
Af((1 + 3)*d3+1:(1 + 1 + 3)*d3,:) = fy_test;    % Impact on u -> y
coordinate
Af((2 + 3)*d3+1:(2 + 1 + 3)*d3,:) = fz_test;    % Impact on u -> z
coordinate

% Computation of the Curl Matrix: Return in the already desired form
% - Computation of Volume Integrals
[Mi, CMy_xz, CMz_xy, CMz_yx, CMx_yz, CMx_zy, CMy_zx] =
curlMatrix(T,k,formulas{1});

% - Computation of Faces Integrals
[tauPPxx,  tauPPxy,  tauPPxz, ...
 tauPPyx,  tauPPyy,  tauPPyz, ...
 tauPPzx,  tauPPzy,  tauPPzz, ...
 tauDPx,   tauDPy,   tauDPz,  ...
 tauDPx_t, tauDPy_t, tauDPz_t,...
 nxDP,     nyDP,     nzDP,    ...
 nxPP,     nyPP,     nzPP,  ...
 tauDD] = convFace(T, tau, k, formulas{4});

% Matrices defintions: A solvers local
A1=zeros(6*d3,6*d3,Nelts);  % Solver storage reservation
A2=zeros(6*d3,4*2*d2,Nelts);  % Solver storage reservation

% Matrices defintions: C solvers Flux
CM=zeros(2*4*d2,2*4*d2,Nelts);   % Solver storage reservation
Cf=zeros(2*4*d2,Nelts);        % Solver storage reservation

% Matrices related to solvers

% Definition of the Mass matrix asociated with a_1: 6*d3 x 6*d3
O=zeros(d3,d3,Nelts);   % Big O cero matrix: Shapes juntions

% Block definition of matrix
Mk = [Mi ,O  ,O   ; ...
      O  ,Mi ,O   ; ...
      O  ,O  ,Mi ];

pcurlPP = [ O                       , permute(CMz_yx, [2 1 3]) ,
-permute(CMy_zx, [2 1 3]) ; ...
           -permute(CMz_xy, [2 1 3]) ,  O                        ,
permute(CMx_zy, [2 1 3]) ; ...
           permute(CMy_xz, [2 1 3]) , -permute(CMx_yz, [2 1 3]) ,
O                       ];

nPP = [    0, -nzPP,  nyPP ; ...
       nzPP,     0,  -nxPP ; ...
       -nyPP,  nxPP,    0 ];

tauPP = [tauPPxx  ,-tauPPxy ,-tauPPxz ; ...
         -tauPPyx ,tauPPyy  ,-tauPPyz ; ...
         -tauPPzx ,-tauPPzy ,tauPPzz  ];

A1 = [Mk             ,-pcurlPP   ;
      pcurlPP + nPP ,Mk + tauPP ];

% Definition of  the Mass matrix asociated with a_2: 6*d3 x 2*(4*d2)
nDP = [nxDP ,nyDP ,nzDP]; tauDP = [tauDPx ,tauDPy ,tauDPz];

A2 = [  permute(nDP, [2 1 3])    ; ...
       -permute(tauDP, [2 1 3]) ];

% Definition of the Mass matrix asociated with a_3: 2*(4*d2) x 6*d3
nDP = [nxDP ,nyDP ,nzDP]; tauDP_t = [tauDPx_t ,tauDPy_t ,tauDPz_t];

A3 = [nDP, tauDP_t];

% Parallel creation of flux operators
```

```
parfor i=1:Nelts
    CM(:,:,i) = A3(:,:,i)/A1(:,:,i) * A2(:,:,i) + tauDD(:, :, i);
    Cf(:,i)   = A3(:,:,i)/A1(:,:,i) * Af(:,i);
end
```

Now it's time to solve the system, and recover all the elements used. This is done solving the post-processing system

```
% Recovery of all elements
M=sparse(R(:),C(:),CM(:));

% This step takes 2 things in count:
%  - accumarray(ind, data): sum of common elements defined by the ind
%    and the data array it's the information to operate.
%  - the matrix Cf: Remember it's one of the flux elements defined.
phif=accumarray(RowsRHS,Cf(:));

% Boundary conditions evaluation
% The resulting terms are:
%  - uhartD : Reference term evaluation on the Dirichlet faces
%  - qhatN  : Reference term evaluation on the Neumann faces
[uhatD,qhatN]=BC3D(gx, gy, gz, @(x, y, z)0.*x, @(x, y, z)0.*x, @(x, y,
z)0.*x, T, k, formulas{4});

% Dirichlet BC: Storage
uhatD=reshape(uhatD,2*d2*Ndir,1); % uhatD stored as a vector: d2 *
Nneu x 1 : todo > Adjust space
Uhatv=zeros(2*d2*Nfaces, 1);      % Storage reservation
Uhatv(dirfaces)=uhatD;            % Uhat stored as a vector: d2*Nfaces

% RHS Re-assemly
rhs=zeros(2*d2*Nfaces,1);               % Storage reservation: RHS
rhs(free)=phif(free);                   % phi f from accumarray
evaluation on free faces
qhatN=reshape(qhatN,3*d2*Nneu,1);       % qhatN stored as a vector: d2
* Nneu x 1

% Construction of RHS from the equation (5.3) : Denominator terms.
rhs(neufaces)=rhs(neufaces)+qhatN;      % Perturbation on Neumann
faces from the BC3D
rhs=rhs-M(:,dirfaces)*Uhatv(dirfaces);  % Perturbation on Dirichlet
faces from the BC3D

% Export of the system
system={M,rhs,free,dirfaces};
solvers={A1,A2,Af};
Uh=[];
Qxh=[];
Qyh=[];
Qzh=[];
Uhat=[];
```

```
% RHS currently has the information of the frontier conditions
% and the information of inner skeleton: solution of uhat
Uhatv(free)=M(free,free)\rhs(free);     % Division by matrix CM (5.3)
```

```
Uhat=reshape(Uhatv,2*d2, Nfaces);          % Reshape on d2 x Nfaces
(Proyections)
% ---------------------------------- Uh Qxh Qyh Qzh
------------------------------ %
% Reconstruction and posprocessing of the skeleton solution to recover
% information on the elements and faces

% Recover of values per elements
faces=T.facebyele'; faces=faces(:);             % Recovery of faces
information
uhhataux=reshape(Uhat(:,faces),[2*4*d2,Nelts]); % Reshape on the faces
of Uhat
sol=zeros(6*d3,Nelts);                          % Reservation of
storage

% Parallel solution of system
parfor K=1:Nelts
    sol(:,K)=A1(:,:,K)\(Af(:,K)-A2(:,:,K)*uhhataux(:,K));
end

% Solution separation on coordinates
Qxh = sol(block3(1),:); Qyh = sol(block3(2),:); Qzh = sol(block3(3),:);
Uxh = sol(block3(4),:); Uyh = sol(block3(5),:); Uzh = sol(block3(6),:);
```

Error comparison with solution

Now we can review our process checking if the error obtained for the process done. Let's check the associated errors

```
error_q = errorElem(T,zx,Qxh,k,formulas{1})...
        + errorElem(T,zy,Qyh,k,formulas{1})...
        + errorElem(T,zz,Qzh,k,formulas{1});

error_u = errorElem(T,ux,Uxh,k,formulas{1})...
        + errorElem(T,uy,Uyh,k,formulas{1})...
        + errorElem(T,uz,Uzh,k,formulas{1});

normQ = errorElem(T,zx,zeros(size(Qxh)),k,formulas{1})...
      + errorElem(T,zy,zeros(size(Qxh)),k,formulas{1})...
      + errorElem(T,zz,zeros(size(Qxh)),k,formulas{1});

% Errores en Volumen: Variable auxiliar Z = curl(u)
ErrorQ = [ error_q/normQ]
% Errores en Volumen: Variable U
ErrorU=[ error_u/normU]
normUhat   = errorFace(T, ux, uy, uz,zeros(size(Uhat)),k,formulas{4});
error_uhat = errorFace(T, ux, uy, uz, Uhat, k,formulas{4});
```

```
% Errores en caras: Variable U_hat
ErrorUhat = [ error_uhat/normUhat]
%error_Puhat=errorFaces(T,@(x) 0.*x,Uhat-Puhat,k,formulas{4});
%error_Pu=errorElem(T,@(x) 0.*x,Pu-Uh,k,formulas{1});

%ErrorPu=[ErrorPu error_Pu/normU];
%ErrorPuhat=[ErrorPuhat error_Puhat/normUhat];






% Postprocessing
%Uhstar=postprocessing(T,km,Qxh,Qyh,Qzh,Uh,k,formulas{1});

%error_Ustar=errorElem(T,ux,Uhstar,k+1,formulas{1});
%ErrorUstar=[ErrorUstar error_Ustar/normU];

%h=[h 1/(2^i)];
```

```
% Rate of sucesion: Convergence verification
rateQ=log2(ErrorQ(1:end-1)./ErrorQ(2:end))
rateU=log2(ErrorU(1:end-1)./ErrorU(2:end))
rateUhat=log2(ErrorUhat(1:end-1)./ErrorUhat(2:end))
```

The change in space for the flux: Verification of values for existence of conditions

We defined the space for the face polynomials to be:

$\mathbb{M} = \left\{ \eta_h \in [L^2(\mathscr{E})]^3 : \eta_h|_k \in [\mathbb{P}_k(K)]^3 \,\forall K \in \mathscr{E}_h / (\eta_h \cdot n) = 0 \right\}$, for this task, we must introduce the new base:

```

```

Redefinition of Boundary conditions

The Boundary conditions involve the integrals related to the external faces
of the defined geometry, this reason envolves the new space defined:

$\mathbb{M} = \left\{ \eta_h \in [L^2(\mathscr{E})]^3 : \eta_h|_k \in [\mathbb{P}_k(K)]^3 \,\forall K \in \mathscr{E}_h / (\eta_h \cdot n) = 0 \right\}$, for this task, we must define the new basis in the faces.

```

```