HDG Methods in Feel++
Basic components and toolboxes

L. Sala, R. Hild

Université de Strasbourg

13/09/2017
HDG Methods in Feel++: basic components
Overview

- Introduction to an elliptic second order model
- Presentation of the HDG method with IBC
- Formulation and discretization of HDG with IBC
- The static condensation process: local and global solver
Second order elliptic problem: presentation

\begin{align*}
  \mathbf{j} + \mathcal{K} \nabla p &= 0 \quad \text{in } \Omega \tag{1} \\
  \nabla \cdot \mathbf{j} &= f \quad \text{in } \Omega \tag{2} \\
  p &= 0 \quad \text{on } \Gamma_D \tag{3} \\
  \mathbf{j} \cdot \mathbf{n} &= g_N \quad \text{on } \Gamma_N \tag{4}
\end{align*}

where $\mathcal{K} \in (L^\infty(\Omega))^{n \times n}$ and $f \in L^2(\Omega)$

The Lax-Milgram Lemma provides existence and uniqueness of $p \in H^1(\Omega)$ and $\mathbf{j} \in H(\text{div}; \Omega)$.

The dependent variables $p$ and $\mathbf{j}$ could represent the electric potential and electric current in a circuit, or the temperature and heat flux in a solid. Another relevant application is the flow in a porous medium in stationary conditions.
The integral boundary condition

The simulation of complex systems in nanoscale electronics, thermo-electronics and ocular biomechanics, gives rise to physical properties of some portions of the boundary that do not fall into standards categories thus making necessary to introduce novel kinds of boundary conditions.

**Integral Boundary Condition (IBC):** \( \int_{\Gamma_{ibc}} \mathbf{j} \cdot \mathbf{n} = I_{target} \).

**Properties:**

1. the restriction \( p|_{\Gamma_{ibc}} \) is equal to a constant value (to be determined)

2. the integral over \( \Gamma_{ibc} \) of the normal flux \( \mathbf{j} \cdot \mathbf{n} \) is equal to a given value
The integral boundary condition applied to the HDG method is a new feature. A paper will be available soon ...
The discretization

We denote by \( T_h \) (called triangulation) the collection of elements \( K \) such that \( \Omega = \bigcup_{K \in T_h} K \). We consider a family of conforming, regular triangulations of \( \Omega \), \( \{ T_h \}_{h>0} \). We introduce:

- \( h := \max_{K \in T_h} h_K \)
- \( \partial K \) the boundary of \( K \) with its measure \( |F| \)
- \( n_{\partial K} \) is the associated unit outward normal vector.
- The skeleton of \( T_h \) is the collection of all the faces of \( T_h \) into the set \( F_h \).
- \( F_h = F_h^\Gamma \cup F_h^0 \), \( F_h^\Gamma = F_h^D \cup F_h^N \cup F_h^{ibc} \)

\[
[q]_F := q^{K_1} \cdot n_{\partial K_1}|_F + q^{K_2} \cdot n_{\partial K_2}|_F
\]

It can be proved that \( [q]_F := 0 \ \forall F \in F_h^0 \iff q \in H(div; \Omega) \).
The spaces and the numerical flux

Functions belonging to $V_h$ and $W_h$ are, in general, fully discontinuous over $\mathcal{T}_h$, whereas functions in $M_h$ are fully discontinuous on $\mathcal{F}_h$ and single-valued on each face $F \in \mathcal{F}_h$ of the skeleton of $\mathcal{T}_h$.

- $V_h = \prod_{K \in \mathcal{T}_h} V(K)$, $V(K) = [P_k(K)]^n$
- $W_h = \prod_{K \in \mathcal{T}_h} W(K)$, $W(K) = [P_k(K)]$
- $M_h = \{ \mu \in L^2(\mathcal{F}_h) : \mu|_F \in P_k(F) \ \forall \ F \in \mathcal{F}_h \}$

Local Discontinuous Galerkin Hybridizable (LDG-H)

$$\hat{\mathbf{j}}_{\partial K} \cdot \mathbf{n}_{\partial K} = j^K_h|_{\partial K} \cdot \mathbf{n}_{\partial K} + \tau_{\partial K} \left( p^K_h|_{\partial K} - \hat{p}_h|_{\partial K} \right)$$

The quantity $\tau_{\partial K}$ is a nonnegative stabilization parameter.
Discrete formulation

Find $\mathbf{j}_h \in V_h$, $p_h \in W_h$ and $\hat{p}_h \in M_h$ such that:

$$
\sum_{K \in T_h} \left[ \left( \mathbf{K}^{-1} \mathbf{j}_h^K, \mathbf{v}_h^K \right)_K - \left( p_h^K, \nabla \cdot \mathbf{v}_h^K \right)_K + \langle \hat{p}_h, \mathbf{v}_h^K \cdot n \partial K \rangle_{\partial K} \right] = 0 \quad \forall \mathbf{v}_h \in V_h
$$

$$
\sum_{K \in T_h} \left[ - \left( \mathbf{j}_h^K, \nabla w_h^K \right)_K + \langle \hat{j}_h^K \cdot n, w_h^K \rangle_{\partial K} \right] = \sum_{K \in T_h} \left( f, w_h^K \right)_K \quad \forall w_h \in W_h
$$

$$
\sum_{K \in T_h} \langle \hat{j}_h^K \cdot n, \mu_h \rangle_{\partial K} = \langle \mathbf{g}_N, \mu_h \rangle_{\Gamma_N} + I_{\text{target}} |\Gamma_{ibc}|^{-1} \langle \mu_h, 1 \rangle_{\Gamma_{ibc}} \quad \forall \mu_h \in M_h
$$

**Static Condensation**

These discrete equations hold in the interior of each $K \in T_h$ and can be solved for each $K$ to eliminate $\mathbf{j}_h^K$ and $p_h^K$ in favor of the variable $\hat{p}_h^{\partial K}$.

*The static condensation will be discussed more in detail later ...*
Reformulation after the Static Condensation

By exploiting the property of the test function $\mu_h$ of being single-valued on each face of $\mathcal{F}_h$, we lead to three sets of distinct equations:

1. **Enforcement in a weak sense of the interelement continuity of the normal component of $\mathbf{j}_h$ across each internal face:**
   \[
   \langle [\mathbf{j}_h], \mu_h \rangle_F = 0 \quad \forall F \in \mathcal{F}_h^0 \quad \mu \in M_h
   \]

2. **Enforcement in a weak sense of the Neumann boundary condition:**
   \[
   \langle \mathbf{j}_h \cdot \mathbf{n}, \mu_h \rangle_F = \langle g_N, \mu_h \rangle_F \quad \forall F \in \mathcal{F}_h^N \quad \mu_h \in M_h
   \]

3. **Enforcement in a weak sense of the integral boundary condition (scalar equation):**
   \[
   \langle \mathbf{j}_h \cdot \mathbf{n}, \mu_h \rangle_{\Gamma_{ibc}} = \left| \Gamma_{ibc} \right|^{-1} \langle \mu_h, 1 \rangle_{\Gamma_{ibc}} \quad \forall \mu_h \in M_h
   \]
Static Condensation: why?

The renewed interest in HDG comes from the use of static condensation, indeed the resulting global system is smaller than that of other DG methods of comparable accuracy. Moreover after solving the global system, the unknowns can be recovered locally, in parallel, which is perfect to implement in Feel++. The idea of the static condensation is to recast the system onto a global linear system where only the trace of the solution on the boundaries of the mesh elements shows up. The inconvenience is that we need to add an extra variable, which will be more complicated to handle.
Comparison between continuous Galerkin and HDG method

- **CG method**: all the elements sharing node $I$ contribute to the connections between node $I$ and the remaining nodes in $\mathcal{T}_h$. The number of connections $(I, J)$ is equal to 37.

- **HDG method**: the two neighboring elements are interconnected by the boundary degrees of freedom highlighted in the light blue rectangle. The number of connections $(I, J)$ is equal to 24.

**Figure**: Comparison between the connectivity of the continuous Galerkin (CG) and HDG method in the case $d = 2$ without integral boundary conditions.
Local reformulation

Let us write the space for the numerical trace $\hat{\rho}_h$ as $M_h = \tilde{M}_h \oplus M^*_h$ with

$$\tilde{M}_h = \{ \mu \in L^2(\mathcal{F}_h \setminus \mathcal{F}_{ibc}^h), \mu|_F \in \mathcal{P}_k(F) \quad \forall F \in \mathcal{F}_h \setminus \mathcal{F}_{ibc}^h, \quad \mu|_F = 0 \quad \forall F \in \mathcal{F}_D^h \}$$

$$M^*_h = \{ \mu \in L^2(\mathcal{F}_{ibc}^h), \mu|_F \in \mathbb{R} \quad \forall F \in \mathcal{F}_{ibc}^h \}.$$

With $(v_h, w_h, \mu_{1,h}, \mu_{2,h}) \in V_h \times W_h \times \tilde{M}_h \times M^*_h$ we consider the matrices related to the local bilinear forms:

$$A^K_{11} \leftrightarrow \left( \mathcal{K}^{-1} j^K_h, v^K_h \right)_K$$
$$A^K_{12} \leftrightarrow \left( p^K_h, \nabla \cdot v^K_h \right)_K$$
$$A^K_{13} \leftrightarrow \langle \hat{\rho}_h | \tilde{M}_h, v^K_h \cdot n \partial K \rangle \partial K$$
$$A^K_{14} \leftrightarrow \langle \hat{\rho}_h | M^*_h, v^K_h \cdot n \partial K \rangle \partial K$$
$$A^K_{22} \leftrightarrow \langle \tau p^K_h, w^K_h \rangle \partial K$$
$$A^K_{23} \leftrightarrow \langle \tau \hat{\rho}_h | \tilde{M}_h, w^K_h \rangle \partial K$$
$$A^K_{24} \leftrightarrow \langle \tau \hat{\rho}_h | M^*_h, w^K_h \rangle \partial K$$
$$A^K_{33} \leftrightarrow \langle \tau \hat{\rho}_h | \tilde{M}_h, \mu_{1,h} \rangle \partial K$$
$$A^K_{44} \leftrightarrow \langle \tau \hat{\rho}_h | M^*_h, \mu_{2,h} \rangle \partial K$$
Let $n_V$, $n_W$ and $n_M$ denote the dimensions of $\mathbb{P}_k(K)$, $\mathcal{P}_k(K)$ and $\mathcal{P}_k(F)$ respectively, with $K \in \mathcal{T}_h$, $F \in \partial K$. Moreover, let $N_F$ be the number of faces in $\partial K \cap (F^0_h \cup F^N_h)$.

Local reformulation

Let $n_V$, $n_W$ and $n_M$ denote the dimensions of $\mathbb{P}_k(K)$, $\mathcal{P}_k(K)$ and $\mathcal{P}_k(F)$ respectively, with $K \in \mathcal{T}_h$, $F \in \partial K$. Moreover, let $N_F$ be the number of faces in $\partial K \cap (F^0_h \cup F^N_h)$.

\[
A^K = \begin{bmatrix}
A^K_{11} & -A^K_{12} \\
(A^K_{12})^T & A^K_{22}
\end{bmatrix} \quad B^K = \begin{bmatrix}
A^K_{13} & A^K_{14} \\
-A^K_{23} & -A^K_{24}
\end{bmatrix} \quad F^K = \begin{bmatrix}
0 & A^K_f
\end{bmatrix}
\]

with $A^K \in \mathbb{R}^{(n_V+n_W) \times (n_V+n_W)}$, $B^K \in \mathbb{R}^{(n_V+n_W) \times (n_M+1)}$ and $F^K \in \mathbb{R}^{(n_V+n_W)}$

Local solutions

\[
\begin{bmatrix}
j^K \\
p^K
\end{bmatrix} = - (A^K)^{-1} B^K \begin{bmatrix}
\hat{p}_h | \check{m}_h \\
\hat{\rho}_h | \check{M}_h^*
\end{bmatrix} + (A^K)^{-1} F^K \tag{5}
\]
Flux operators

If $\partial K \cap F_h^{ibc} \neq \emptyset$, we define the matrices

$$C^K = \begin{bmatrix} (A^K_{13})^T & (A^K_{23})^T \\ (A^K_{14})^T & (A^K_{24})^T \end{bmatrix}, \quad D^K = \begin{bmatrix} A^K_{33} & 0 \\ 0 & A^K_{33} \end{bmatrix}$$

The matrix representation of the numerical flux is:

$$C^K \begin{bmatrix} j^K \\ p^K \end{bmatrix} - D^K \begin{bmatrix} \hat{p}_h | \tilde{M}_h \\ \hat{\phi}_h | M_h^* \end{bmatrix} = E^K_f - E^K \begin{bmatrix} \hat{p}_h | \tilde{M}_h \\ \hat{\phi}_h | M_h^* \end{bmatrix}$$

where

$$E^K_f = C^K (A^K)^{-1} F^K \quad E^K = C^K (A^K)^{-1} B^K + D^K$$
Global solver and boundary conditions

To get the solution \( \hat{p} \) we need to assemble all the matrices \( E^K \) produced element by element into a global matrix \( \mathbb{H} \). This matrix collects the fluxes from all the elements with the result that opposing sign fluxes in internal faces are added. The vectors \( E^K_f \) also have to be assembled into a global vector \( F \). The global system then reads

\[
\mathbb{H} \hat{p} = F + G_N + G_{ibc} \tag{6}
\]

where
- the vector \( G_N \) is coming from the Neumann boundary condition
- the vector \( G_{ibc} \) is coming from the IBC (only one entry is non-zero)
- the Dirichlet boundary condition are enforced during the definition of spaces \( \tilde{M}_h \).
HDG Toolbox in Feel++: mathematical formulations, results and implementation
Overview

- Implementation of HDG method
- Implementation of Static Condensation
- Class MixedPoisson
- Class MixedElasticity
First, we need to create $\mathcal{F}_h^{ibc}$ and $\mathcal{F}_h \setminus \mathcal{F}_h^{ibc}$.

Complement and submesh

```cpp
auto complement_integral_bdy = complement(faces(mesh),
                                           [&mesh]( auto const & e ) {
    for( int i = 0; i < ioption("nb_ibc"); ++i ) {
        auto ind = i == 0 ? "" : std::to_string(i+1);
        std::string marker = boost::str(boost::format("Ibc%1%") % ind);
        if ( e.hasMarker() && e.marker().value() ==
             mesh->markerName( marker ) )
            return true;
    }
    return false;
});

auto face_mesh = createSubmesh( mesh, complement_integral_bdy,
                                EXTRACTION_KEEP_MESH_RELATION , 0 );

auto ibc_mesh = createSubmesh( mesh, markedfaces(mesh,ibc_markers),
                                EXTRACTION_KEEP_MESH_RELATION , 0 );
```
Then, we need to create the spaces $V_h, W_h, \tilde{M}_h$ and $M_h^*$. We also want to be able to have several number of integral boundary conditions, the number of which will only be known at execution.

$$V_h \times W_h \times \tilde{M}_h \times (M_h^*)^n$$

### Spaces initialization and product space

```cpp
Vh_ptr_t Vh = Pdhv<OrderP>( mesh, true );
Wh_ptr_t Wh = Pdh<OrderP>( mesh, true );
Mh_ptr_t Mh = Pdh<OrderP>( face_mesh, true );
Ch_ptr_t Ch = Pch<0>( ibc_mesh, true );
auto ibcSpaces = boost::make_shared<ProductSpace<Ch_ptr_t,true> > ( ioption("nb_ibc"), Ch);
auto ps = product2( ibcSpaces, Vh, Wh, Mh );
```
We can choose at the execution if we solve the problem using static condensation or the monolithic strategy.

```cpp
solve::strategy strategy = boption("sc.condense")?solve::strategy ::static_condensation:solve::strategy ::monolithic;
auto a = blockform2(ps, strategy, backend());
auto rhs = blockform1(ps, strategy, backend());

// Assembling the right hand side
rhs(1_c) += integrate(_range=elements(mesh),_expr=-f*id(w));

// Assembling the main matrix
a(0_c,0_c) += integrate(_range=elements(mesh),
    _expr=(trans(lambda*idt(u))*id(v)));

a(2_c,2_c) += integrate(_range=markedfaces(mesh,"Neumann"),
    _expr=-tau_constant*idt(phat)*id(l)*(pow(h(),M_tau_order)));
```
We can access a dynamic block of the matrix by adding the relative index. We then create an element of the product space as usual, and access its component in the same way.

```cpp
a( 3_c, 0_c, i, 0 ) += integrate( _range=markedfaces(mesh,"Ibc"),
   _expr=(trans(idt(u))*N()) * id(nu) );
auto U = ps.element();
a.solve( _solution=U, _rhs=rhs,
   _condense=boption("sc.condense"), _name="hdg" );
auto up = U(0_c);
auto pp = U(1_c);
auto ip = U(3_c,0);
```
Static Condensation

How the static condensation is implemented?

- blockforms use MaxtrixCondensed that can use the StaticCondensation class when using static condensation strategy.
- To be able to differentiate the case where we have integral boundary condition or not, we use the hana library from boost:

\[
\text{std::enable_if_t<decltype(hana::size(e.functionSpace()))::value == 3>}
\]

- We first need to collect the local matrices \(A_{ij}^K\) using the syncLocalMatrix function into a map ordered by block, and then element.

\[
\text{std::unordered_map<block_index_t, std::unordered_map<block_element_t, local_matrix_t, boost::hash<block_element_t>, boost::hash<block_index_t>>, boost::hash<block_index_t>>> M_local_matrices;}
\]
How the static condensation is implemented?

- Then we can call the `condense` method to form the $\mathbf{H}$ matrix and the $\mathbf{F} + \mathbf{G}_N + \mathbf{G}_{ibc}$ vector.

  ```cpp
  auto const & A00K = M_local_matrices[std::make_pair(0,0)];
  extractBlock( A00K.at(key), A01K.at(key), A10K.at(key), AK, e1, e2);
  auto Aldlt = AK.ldlt();
  auto AinvF = Aldlt.solve( FK );
  EKF=-CK*AinvF+F2;
  V.addVector( dofs.data(), dofs.size(), EKF.data(),
               invalid_size_type_value, invalid_size_type_value );
  ```

- This is used in to solve the global problem to find $\hat{\mathbf{p}}_h$ and $\hat{\mathbf{p}}_h$.

- We can then solve the local problems to find $\mathbf{j}^K$ and $\mathbf{p}^K$. 
We want to have a class allowing in an easy and reliable way to solve the following problem with the BC:

\[
\begin{align*}
j + K \nabla p &= g \quad \text{in } \Omega \\
\nabla \cdot j &= f \quad \text{in } \Omega \\
p &= g_D \quad \text{on } \Gamma_D \\
j \cdot n &= g_N \quad \text{on } \Gamma_N \\
j \cdot n + g^1_R p &= g^2_R \quad \text{on } \Gamma_N \\
\int_{\Gamma_{ibc}} j \cdot n &= g_I \quad \text{on } \Gamma_{ibc}
\end{align*}
\]

**Example code**

```cpp
typedef FeelModels::MixedPoisson<FEELPP_DIM,FEELPP_ORDER> mp_type;
auto MP = mp_type::New("mixedpoisson");
MP -> init();
MP->assembleAll();
MP->solve();
MP->exportResults();
```
MixedPoisson: Configuration

All the configuration is made through a json file

```json
{
    "Name": "HDG-Mixed-Poisson",
    "ShortName": "MP",
    "Model": "hdg",
    "<section>":
    {
        ...
    }
}
```

For example, if Model, contains the word “picard”, we will use this method to solve the problem (Newton is not yet implemented).

Sections can describe the materials used, the boundary conditions or the post process.
"Materials":
{
  "<marker>":
  {
    "name": "copper",
    "alpha": "3.35e-3",
    "T0": "293",
    "k0": "0.38",
    "k": "k0*T/((1+alpha*(T-T0))*T0):k0:T:alpha:T0"
  }
},

For each volume, give the material properties.

- The conductivity $\mathcal{K}$ is the value of the key given by the option conductivity_json (k0).
- In the non linear case, you can use conductivityNL_json (k), only if it depends on $p$. 
"BoundaryConditions":
{
  "<field>":
  {
    "<type>":
    {
      "<marker>":
      {
        "expr": "0.25"
      }
    }
  }
}

The fields and the type available are the following:
- potential
  - SourceTerm
  - Dirichlet
  - Neumann
  - Robin
- flux
  - SourceTerm
  - Integral
Mixed Poisson: Convergence

(a) 2D

(b) 3D
MixedPoisson: Results

(a) Current density in a portion of a bitter

(b) Temperature in a portion of a bitter
The MixedElasticity class allows us in an easy and reliable way to solve the following problem with the BC:

\[
\begin{align*}
A\sigma - \varepsilon(u) &= 0 & \text{in } \Omega \subset \mathbb{R} \\
\frac{\partial^2 u}{\partial t^2} + \nabla \cdot \sigma &= f & \text{in } \Omega \\
u &= g_D & \text{on } \Gamma_D \\
\sigma \cdot n &= g_N & \text{on } \Gamma_N \\
\int_{\Gamma_{ibc}} \sigma \cdot n &= F_{target} & \text{on } \Gamma_{ibc}
\end{align*}
\]

\[
c_1 = \frac{1}{2\mu} \\
c_2 = \frac{-\lambda}{2\mu (3\lambda + 2\mu)} \\
\varepsilon(u) = \frac{1}{2} \left( \nabla u + (\nabla u)^T \right)
\]
**MixedElasticity class: main methods**

**Example code**

```cpp
typedef FeelModels::MixedElasticity<FEELPP_DIM,FEELPP_ORDER,FEELPP_GEO> me_type;

// Initialization
auto ME = me_type::New("mixedelasticity");
auto mesh = loadMesh( _mesh=new me_type::mesh_type);
ME -> init(mesh);

// Assembling the system
ME -> assembleCst();
ME -> assembleNonCst();

// Solve the system
ME -> solve();

// Post-processing
ME -> exportResults( mesh);
```
For each volume, give the material properties, in particular the mass density $\rho$ and the Lamé coefficients $\lambda$ and $\mu$. 
MixedElasticity: configuration

Boundary conditions section

"BoundaryConditions":
{
  "<field>":
  {
    "<type>":
    {
      "<marker>":
      {
        "expr": "<value>"
      }
    }
  }
}

The fields and the type available are the following:
- displacement
- Dirichlet
- stress
- SourceTerm
- Neumann
- Integral
Mixed Elasticity: convergence

\[ \| \cdot \|_{L^2} \]

(a) Displacement 3D
(b) Stress 3D
Next steps in the HDG context:

- Singularity images
- New features:
  - MixedPoissonElasticity
  - MixedStokes
  - MixedMaxwell
- Use of reduced-basis method
Questions?

Thanks for your attention!