NGsolve::Come to the edge

Facet spaces and hybrid methods

Jay Gopalakrishnan

Portland State University

Winter 2015

Download code (works only on version 6.0) for these notes from here.

"Come to the edge, he said/ We are afraid, they said" –Guilliame Apollinaire (French Poet)

Vector finite elements

- Unlike `ScalarFiniteElement`, vector finite elements have vector shape functions. These are needed, e.g., for flux computations (H(div)), electromagnetic field computations (H(curl)), etc.
- A typical example is the space built from `HDivFiniteElement`, accessible in a pde file using, e.g.,

```
fespace RT -type=hdivho -order=2
```

- Functions in the `hdivho` space have their normal components continuous across finite element interfaces.

Quiz 1: Project

Task: Given a vector function $\vec{G}$, compute its L^2-projection into the `hdivho` finite element space.

- The projection $\vec{q}$ in the `hdivho` finite element space satisfies

$$\int_\Omega \vec{q} \cdot \vec{r} = \int_\Omega \vec{G} \cdot \vec{r}$$

  for all $\vec{r}$ in the `hdivho` space.

- Make bilinear and linear forms out of the left and the right hand sides above using pde file commands. The integrators you will need are `masshdiv` and `sourcehdiv`.

- Write a complete pde file to compute the projection $\vec{q}$.

Quiz 2: Project into broken spaces

A **broken finite element space** is a finite element space whose continuity requirements across element interfaces have been removed. They are implemented in NGsolve using a "-discontinuous" flag:

```
fespace L  -type=hdivho -order=2 -discontinuous
```

Task: Given a vector function $\vec{G}$, compute its L^2-projection $\vec{q}'$ into the above broken `hdivho` finite element space.

- Write a pde file to compute $\vec{q}'$.
- Check if $\|\vec{q}' - \vec{q}\|_{L^2(\Omega)} \neq 0$.

Quick introduction to "hybridization" of FE spaces

Noting that

$$\texttt{hdivho}\text{ space } \subset \text{ broken } \texttt{hdivho} \text{ space},$$

we may compute the projection $\vec{q}$ remaining within the broken `hdivho` space, provided we add the interlement continuity constraints of the `hdivho` space as additional equations in the definition of $\vec{q}$.

Computing $\vec{q}$ using the broken `hdivho` *space:*

$$\sum_{\textit{elements } K} \int_K \vec{q} \cdot \vec{r} + \sum_{\textit{edges } E} \int_E \lambda \, \textit{jump}(q \cdot n) = \int_\Omega \vec{G} \cdot \vec{r}$$

$$\sum_{\textit{edges } E} \int_E \textit{jump}(q \cdot n) \, \mu = 0$$

where $\vec{q}$ and $\vec{r}$ are in the broken `hdivho` space and λ, μ are in a Lagrange multiplier space. (Precise definitions are in the upcoming code.)

Facet finite element spaces

The Lagrange multipliers live on facets (edges in 2D, faces in 3D).

- Use `class FacetFESpace` and `class FacetVolumeFiniteElement` for implementing these spaces.
- Functions on different facets have no continuity accross subfacets like vertices (or edges in 3D)
- `FacetFESpace` functions are accessed via the mesh elements which then can access their facets.
- In PDE file: `fespace M -type=facet -order=2`
- Need to write our own integrator for terms like

$$\sum_{edges\ E} \int_E \lambda\, jump(q \cdot n)$$

which is equivalent to

$$\sum_{elements\ K} \int_{\partial K} \lambda\, q \cdot n, \qquad \lambda|_{\partial\Omega} = 0.$$

A hybridized implementation in pde file

```
#                                               FILE: hybridproj.pde
fespace L  -type=hdivho -order=2 -discontinuous
fespace M  -type=facet  -order=2 -dirichlet=[1]
fespace LM -type=compound -spaces=[L,M]

#  ...

bilinearform a3 -fespace=LM -symmetric
masshdiv    one -comp=1  # Integrator from NGSolve.
njump       one          # Integrator not provided by NGSolve!

linearform b3 -fespace=LM
sourcehdiv  G -comp=1

gridfunction ql  -fespace=LM  -addcoef

numproc bvp nbvp3 -bilinearform=a3 -linearform=b3
                  -symmetric -eliminate_internal
                  -gridfunction=ql  -solver=direct
```

Writing integrators in C++

```
template<int D>                    // FILE: njumpintegrator.cpp
class FacetNormalJump :  public BilinearFormIntegrator  {

  shared_ptr<CoefficientFunction> coef;

public:

  FacetNormalJump(const Array<shared_ptr<CoefficientFunction>> &
    : BilinearFormIntegrator() {  coef = coeffs[0]; }

  virtual string Name () const { return "FacetNormalJump"; }
  virtual int DimElement () const { return D-1; }
  virtual int DimSpace () const { return D; }
  virtual bool BoundaryForm () const { return false; }
  void CalcElementMatrix(const FiniteElement& fel,
                         const ElementTransformation& eltrans,
                         FlatMatrix<double> elmat,
                         LocalHeap& lh) const  ;
    // Make the element matrix for the form
    //      J( (q,l),  (r,m) ) = <l, r.n> + <q.n, m>        ....
```

Integrators

- Study line by line `njumpintegrator.cpp` (heavily commented).
- Check out `my_little_ngsolve` for other examples on writing your own integrators.

Student Team Project: Learn about the "HDG method," and implement it for the Helmholtz equation.