General Assembler and XFEM in LifeV

Iori Guido
guido.iori@mail.polimi.it

Politecnico di Milano

Lausanne
January 7th, 2013

Outline

A more flexible assembler

In `LifeV` we can find a lot of classes for the purpose of building the matrix for a specific problem: `ADRAssembler`, `OseenAssembler`, `DarcySolver`...

Is it possible to have a tool able to build the matrix for whatever discrete problem? YES!

- ET module, already in the master
- the new general `Assembler` (coming soon on your PC screen, I hope!)

The new assembler, implemented by G.Iori, A.Cervone and A.Fumagalli, performs:

- assembly on blocks of the global matrix
- assembly of whatever operator, passed like a policy to the assembler
- assembly of xfem operators

Interface - ADR with General Assembler

```
// creation of the assembler
Assembler assembler;

// stiffness
assembler.addOperator< AssemblyStiffness >( uFESpace, uFESpace,
                                            systemMatrix, epsilon );
// advection
vector_Type beta( betaFESpace->map(),Repeated );
betaFESpace->interpolate( betaFct, beta, 0.0 );
assembler.addOperator< AssemblyAdvection >( uFESpace, uFESpace,
                                            betaFESpace,
                                            systemMatrix, beta );
// source
assembler.addOperator< AssemblyMassRhs >( uFESpace,
                                          rhs, fRhs );
```

Interface - Stokes with General Assembler

```
// creation of the assembler
Assembler assembler;

// structured global matrix
matrixPtr_Type systemMatrix( new matrix_Type(
                             uFESpace->map() | pFESpace->map() ) );

// build the matrix
assembler.addOperator< AssemblyStiffness >( uFESpace, uFESpace,
                                            systemMatrix->block(0,0),
                                            stiffCoeff );
assembler.addOperator< AssemblyGradient  >( uFESpace, pFESpace,
                                            systemMatrix->block(0,1),
                                            gradCoeff );

assembler.addOperator< AssemblyDivergence>( pFESpace, uFESpace,
                                            systemMatrix->block(1,0),
                                            divCoeff );
```

Implementation (1)

```
template< typename AssemblyPolicy, typename FESpacePtrType,
          typename TensorPtrType, typename CoefficientType >
void
Assembler::addOperatorOnSelectedElements(FESpacePtrType fespace_1,FESpacePtrType fespace_2
                        TensorPtrType matrix, CoefficientType& coefficient,
                        const meshEntityFlag_Type& flag, const predicate_Type& predicate )
{
    Operator<AssemblyPolicy,CoefficientType,FESpacePtrType> op;

    std::vector<FESpacePtrType> fespaces (2);
    fespaces [0] = fespace_1; fespaces [1] = fespace_2;
    op.setup ( fespaces );

    extractedElements_Type elements =
            fespace_1->mesh()->elementList().extractElementsWithFlag( flag, predicate );

    typename extractedElements_Type::const_iterator it;

    for ( it = elements.begin(); it != elements.end(); it++ )
    {
        op.updateCurrentFE( **it );
        op.buildLocalTensor ( coefficient, **it );
        op.assembleLocalTensor ( *matrix );
    }
}
```

Implementation (2)

```
template < typename AssemblyPolicy, typename CoefficientType, typename FESpacePtrType >
class Operator
{

public:
// some typedefs...

Operator( );

void setup( const fespacePtrContainer_Type& fespace );
void updateCurrentFE( const element_Type& geoEle );


template < typename GlobalTensorType >
void assembleLocalTensor ( GlobalTensorType& globalTensor )
{  return assembleLocalTensor ( M_localTensor, globalTensor ); }

void buildLocalTensor ( coefficient_Type& coefficient,
                        const element_Type& geoEle )
{  M_localTensor->zero();
   buildLocalTensor( M_localTensor, coefficient, geoEle ); }

private:
  ...
}
```

How to add a new policy (last slide of code for a while...)

```
class AssemblyPippo: public AssemblyStandard<  matrixDimension_Type >
{
public:

    typedef matrixDimension_Type                          tensorDimension_Type;
    typedef matrixDimension_Type::tensor_Type             tensor_Type;
    typedef boost::shared_ptr<tensor_Type>                tensorPtr_Type;
    typedef AssemblyPippo                                 assembly_Type;

    AssemblyPippo(){}

    template< typename GeometricElementType >
    void buildLocalTensor( matrixDimension_Type::tensor_Type& localMat,
                           const Real& coefficient,
                           const GeometricElementType& geoEle,
                           const std::vector< currentFEPtr_Type > & elementCFEVector,
                           const std::vector< UInt > &           fieldDimVector );

    static const UInt S_numFESpaces = ...;
    static const UInt S_whichUpdate[ S_numFESpaces ];
    static const currentFEFlag_Type S_updateFlag [ S_numFESpaces ];

private:

    AssemblyPippo ( const AssemblyPippo& );
};
```

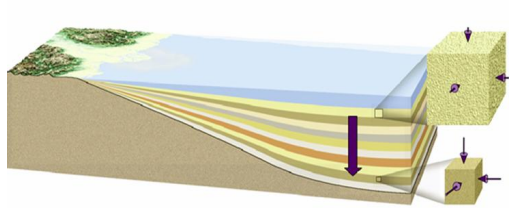
Parameter class

We aim to introduce a new class `Parameter` able to compute the values in the quadrature nodes of an element whatever function of field (`boost::function`, `FEField`, `FEFunction`, vector, xfem stuff...) is passed as input parameter.

The final goal is to have a common interface for the operators (stiffness, mass, gradient, massHdiv...) that takes as input a vector with just the values of the function or field in the quadrature nodes.

Handling discontinuities

Simulation of cracks, shear bands, dislocations, multi-phase problems, two-fluid flows and fluid-structure interaction needs advanced numerical techniques in order to handle the discontinuities of the mathematical model .



Many challenges...

- accuracy in proximity of the discontinuity
- complex geometries
- moving front
- simulation of different scenarios

Trick of the method

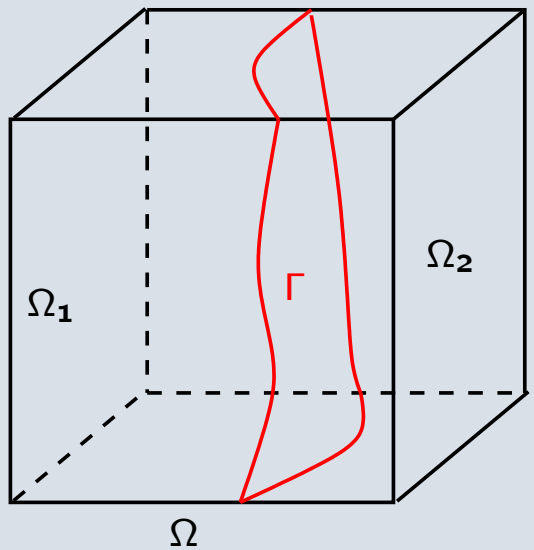
Weak and strong discontinuities are taken into account by an enrichment of the approximation space:

$$U(\mathbf{x}) = \underbrace{\sum_{j \in \mathcal{I}} U_j \phi_j(\mathbf{x})}_{U^{FE}} + \underbrace{\sum_{k} \sum_{i \in \mathcal{I}_k^*} q_i^k \mu_i(\mathbf{x}) [\psi^k(\mathbf{x}) - \psi^k(\mathbf{x}_i)]}_{U^{enr}}$$

where $\mathcal{I}$ is the set of the nodes and $\mathcal{I}_k^* \subset \mathcal{I}$ is the subset on which the enriching functions ψ^k are localized [2].

Differential problem

Let Ω be a bounded domain in $\mathbb{R}^3$, with convex polygonal boundary $\partial\Omega$ and an internal smooth boundary Γ dividing Ω into two open sets Ω_1 and Ω_2. We consider the following stationary problem



$$\begin{cases} -\nabla \cdot (\alpha \nabla u) = f & \text{in } \Omega_1 \cup \Omega_2 \\ u = 0 & \text{on } \partial\Omega \\ [\![u]\!] = 0 & \text{on } \Gamma \\ [\![\alpha \nabla_n u]\!] = g & \text{on } \Gamma \end{cases} \tag{1}$$

where $[\![u]\!] = u_1|_\Gamma - u_2|_\Gamma$ and $u_i = u|_{\Omega_i}$.
(1) has a unique solution in $H^2(\Omega_i)$ for $i = 1, 2$ [1].

The Hansbo-Hansbo method

Let $\mathcal{T}_h$ be a conforming triangulation of Ω. Let $V^h = V_1^h \times V_2^h$ where

$$V_i^h = \{w \in [H^1(\Omega_i)]^d : w\,|_{K\cap\Omega_i} \in \mathbb{P}_1(K \cap \Omega_i), \forall K \in \mathcal{T}_h \text{ and } w\,|_{\partial\Omega} = 0\}$$

for $i = 1, 2$.

Discrete problem [3]

Find $U = (U_1, U_2) \in V^h$ such that

$$a_h(U, \phi) = L(\phi),\ \forall \phi \in V^h \tag{2}$$

where

$$\begin{aligned} a_h(U,\phi) =& (\alpha_i \nabla U_i, \nabla \phi_i)_{\Omega_1 \cup \Omega_2} - (\llbracket U \rrbracket, \{\!\{\alpha \nabla_n \phi\}\!\})_\Gamma \\ & - (\{\!\{\alpha \nabla_n U\}\!\}, \llbracket \phi \rrbracket)_\Gamma + (\lambda \llbracket U \rrbracket, \llbracket \phi \rrbracket)_\Gamma \\ L(\phi) =& (f, \phi)_\Omega + (\kappa_2 g, \phi_1)_\Gamma + (\kappa_1 g, \phi_2)_\Gamma, \end{aligned}$$

and λ is a penalty parameter.
$\{\!\{\cdot\}\!\}$ denotes a weighted mean: $\{\!\{v\}\!\} = \kappa_1 v_1 + \kappa_2 v_2$ where κ_1,κ_2 are appropriate weights.

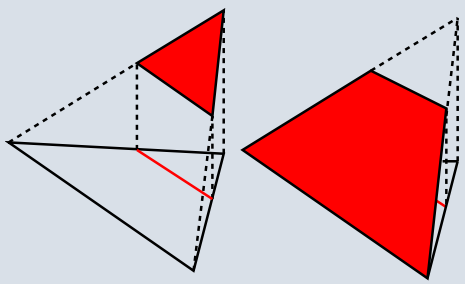
A local enrichment of the approximation space

We define $\mathcal{F}_h = \{K \in \mathcal{T}_h : K \cap \Gamma \neq \emptyset\}$. For each element $K \in \mathcal{F}_h$, let $K_i \in \Omega_i$ denote the part of K in Ω_i.

FE basis for V^h

A local basis function ϕ on $K \in \mathcal{F}_h$ must be discontinuous across Γ:

$$\phi = \begin{cases} \phi_1 \text{ in } K_1 = K \cap \Omega_1 \\ \phi_2 \text{ in } K_2 = K \cap \Omega_2 \end{cases}$$

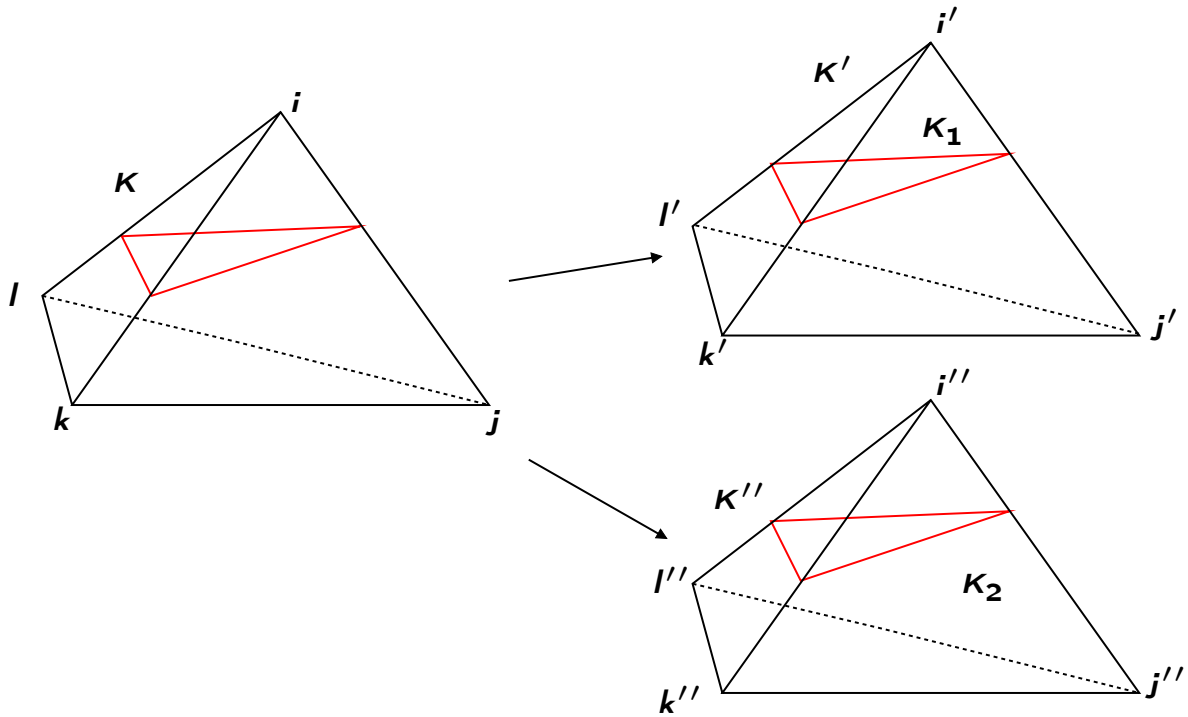


ϕ_1 is represented in K_1 by its nodal values in K and the same holds for ϕ_2.

Since ϕ_1 and ϕ_2 must be independent, we need to double the degrees of freedom on K.

This kind of enrichment leads to a splitting of the cut elements.

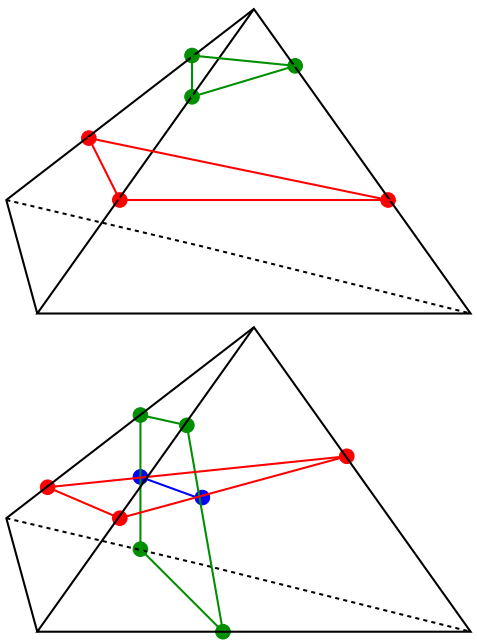
Splitting the cut elements



We assign for each $K \in \mathcal{F}_h$ two identical copies K' and K''. K' to the part K_1 and K'' to K_2.
K' and K'' are geometrically coincident, but they have different degrees of freedom.

Double cut for elements

We now consider a domain crossed by two level-set function, $f(\mathbf{x})$, $g(\mathbf{x})$. Ω is generally partitioned in 3 or 4 sub-domains Ω_i with respect to the sign of level-set functions.
When an element is cut by two surfaces, we need to enrich the approximation space in order to represent the solution u_i on each restriction of element K to Ω_i.



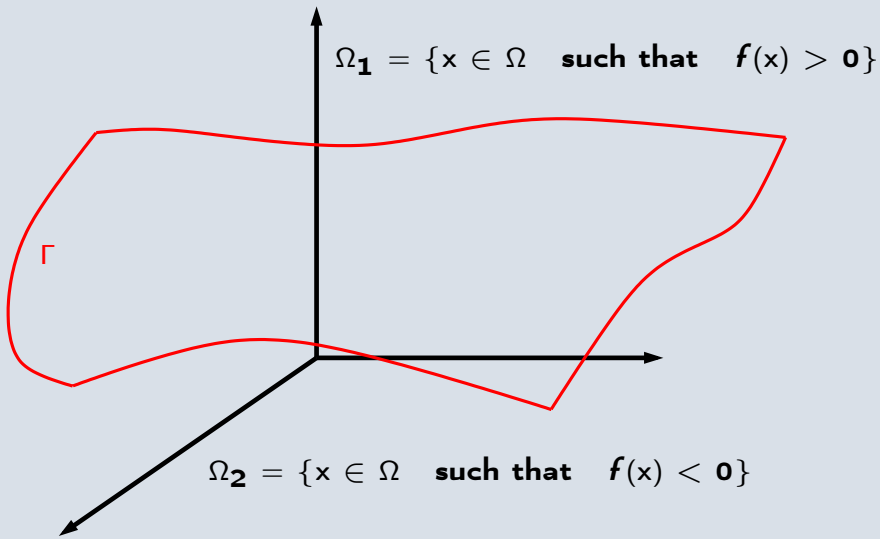
As a consequence, the number of degrees of freedom of an element $K \in (\mathcal{F}_h \cap \mathcal{G}_h)$ is dependent on the number of sub-domain Ω_i such that $K \cap \Omega_i \neq \emptyset$.

Introducing a discontinuity in the domain

Level-set

A *level-set* function is a scalar continuous function $f(\mathbf{x})$ such that its zero-level curve is interpreted as the surface of discontinuity:

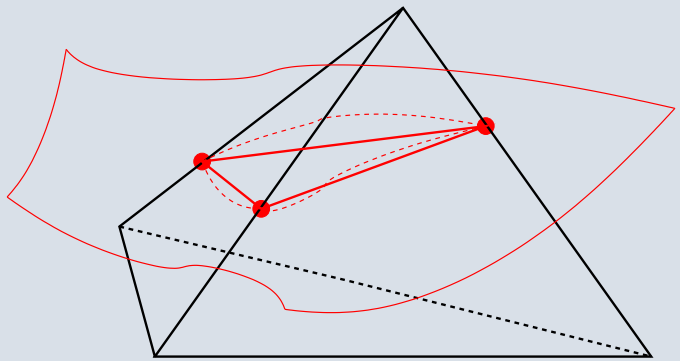
$$\Gamma = \{\mathbf{x} \in \Omega \text{ such that } f(\mathbf{x}) = 0\}$$



Cut information

The level-set method simplify the computation of informations about the intersection between surface and mesh:

- search of elements crossed by the surface
- edge-surface intersections



How to define a surface

```
class NewSurface: public AnalyticSurface{

//...typedefs...

NewSurface( const vectorCoeff_Type& coefficients );
NewSurface( const NewSurface& surface);

// definition of abstract methods of class AnalyticSurface
virtual Real operator()( const point_Type& point ) const;
virtual VectorSmall<3>& normal( const Real& x,
                                const Real& y,
                                const Real& z ) const;
};
```

Handling the interface crossing the domain

XfemMeshHandler class

This class performs the loop on all the elements of the mesh in order to compute:

- position of the elements with respect to the interfaces
- intersections between edges and interfaces
- ri-triangulation of cut elements and cut facets (using `QHull`)
- triangulation of the planar approximation of the interface crossing the elements

Only meshes templetized by `xfemMarkerCommon_Type` are allowed.
It can handle linear (triangular and tetrahedric) elements.
For the computation of these informations we call:

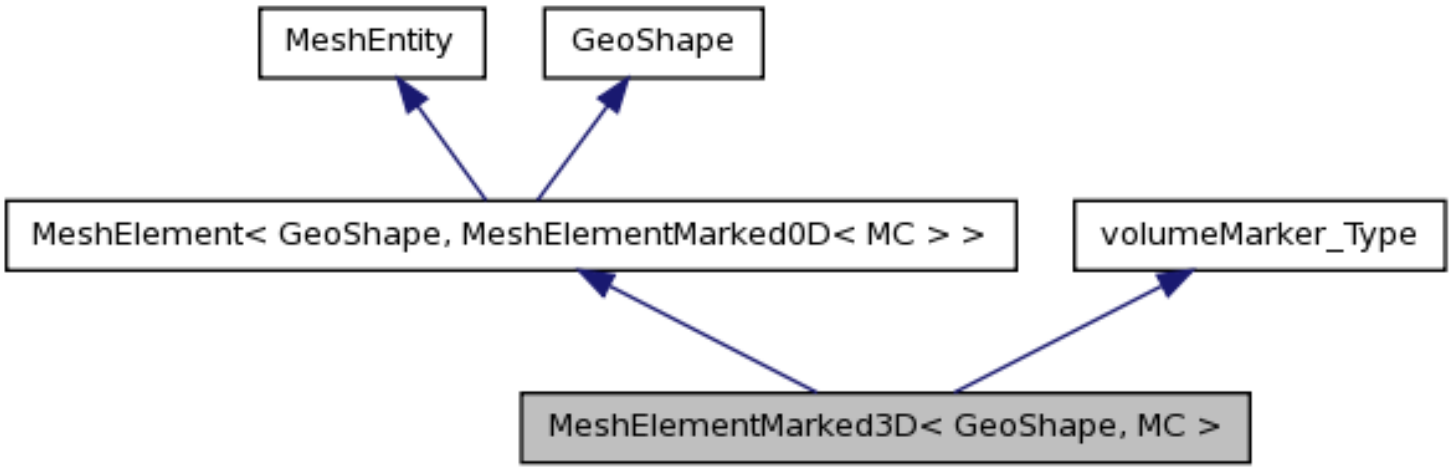
```
XfemMeshHandler< LinearTetra > xfemhandler( fullMeshPtr, surfaces );
xfemhandler.intersectMeshSurfaces();
```

Storing additional informations

We need to store the informations about :

- points of intersection between edges and surfaces
- sub-elements and sub-facet (for computation of local matrices of cut elements)

Where can I store them? The answer is the `Marker` class.



New markers for entities

```
template<class MT>
class MarkerCommonXfem
{
public:

    typedef MT markerTraits_Type;
    typedef MarkerXfem<MT>                  pointMarker_Type;
    typedef MarkerEdgeXfem<MT>              edgeMarker_Type;
    typedef MarkerFaceXfem<MT>              faceMarker_Type;
    typedef MarkerElementXfem<MT>           volumeMarker_Type;
    typedef Marker<MT>                      regionMarker_Type;
};

typedef MarkerCommonXfem<MarkerIDStandardPolicy> xfemMarkerCommon_Type;
} // Namespace LifeV
```

A flag to identify the sub-domains

If the domain is crossed by only one surface, we identify with 0 an element located in the part of domain s.t. $f(\mathbf{x}) > 0$, 2 if $f(\mathbf{x}) < 0$ and 1 if the element is crossed by the surface defined by the equation $f(\mathbf{x}) = 0$
In case of two interfaces, regions are defined as follows:

	$f_1(\mathbf{x}) > 0$	$f_1(\mathbf{x}) = 0$	$f_1(\mathbf{x}) < 0$
$f_2(\mathbf{x}) > 0$	0	1	2
$f_2(\mathbf{x}) = 0$	3	4	5
$f_2(\mathbf{x}) < 0$	6	7	8

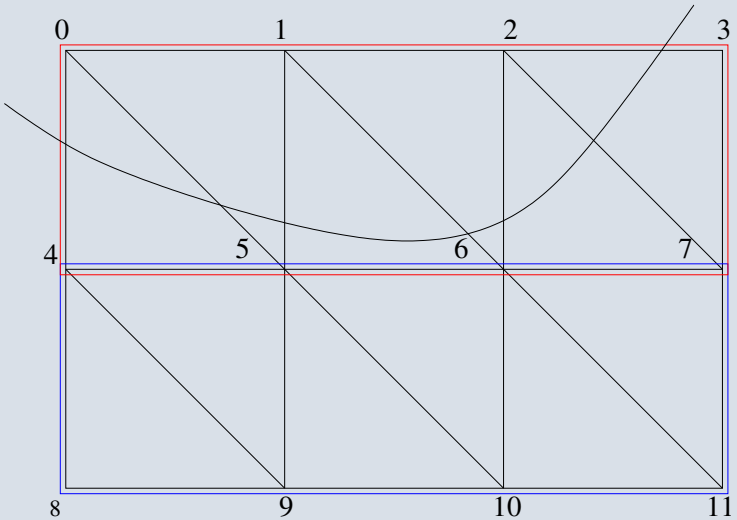
This approach can be easily generalized.

One geometric element, two finite elements!

Problems

Dof assignation in `LifeV` is based on the global ID of the related geometric entity. This implies two main problems:

- how can I assign two different dof IDs given only one global ID?
- how can i guarantee that the dof IDs given, for example, to a node (of a cut element) near to the boundary of the partition are the same on the two processors?



DOFXfem and global virtual IDs

The solution is

- assigning a number of virtual global IDs to a geometric entity equal to the number of sub-domains in which the entity must be represented. When assigning the dof of a node for a certain region I need to ask for the global virtual ID of the node for that region.
- `DOFXfem` class, that creates a map of DOF, where the number of elements is not equal to the number of geometric ones but to the number of finite elements on the partition. To know the dofs of an element, it is necessary to give the local ID of the element and the flag of the region considered.

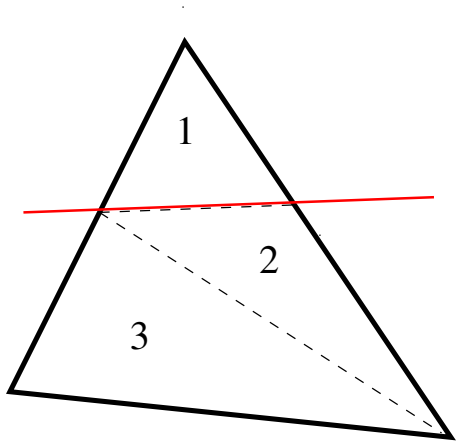
To set the virtual global IDs, we need to call, before partitioning:

```
MultipleIDHandler<LinearTetra> idHandler( fullMeshPtr );
idHandler.updateIDMaps( refFE );
```

Integration on sub-elements: a 2D example

The following tools have been implemented:

- integration on a sub-element (quadrature nodes change but the basis functions are the same of the entire element)
- integration on the triangulation on the cutting surface
- assembling of different contribution coherently with the dofs of the two finite elements (sub-element 1 for K' and sub-elements 2 and 3 for K'')

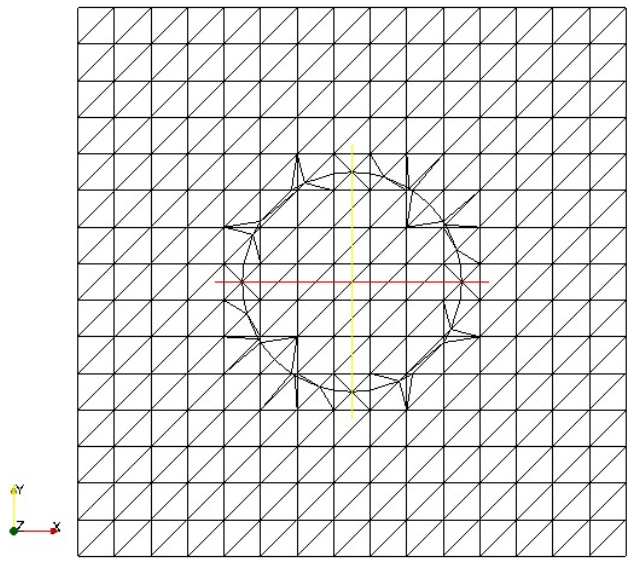


Integration and assembling of different contributions of a cut element is done by the class `Xfem`. We'll see later how to use this class with the `Assembler`.

Exporting the solution (1)

Mesh refinement

We use the ri-triangulation of the cut elements to refine the mesh in order to have the edges of the elements matching with the interface.



Refinement is done calling the method:

```
xfemhandler.modifiedMesh(  );              // refinement
meshPtr = xfemhandler.meshPartition(); // we take the new mesh
```

Exporting the solution (2)

Some new classes...

- `DiscontinuousDOF`, a sort of DG DOF
- `ExporterHDF5FromDOF`, an exporter for the discontinuous FE space
- `XfemInterpolation`, interpolator between old and new mesh

```
XfemInterpolation meshInterpolator;
meshInterpolator.initialize( meshPtr ); //meshPtr is the original mesh

...refinement...

dFeSpacePtr_Type expFESpace( new dFeSpace_Type( meshPtr,
              refFE1, qR, bdQr, 1, comm ) ); //meshPtr is the new mesh

vectorPtr_Type solutionXfem( new vector_Type( expFESpace->map() ) );

meshInterpolator.interpolateSolutionToP1( *solutionRepeated,*solutionXfem,
                                          uFESpace, expFESpace );
```

User interface with Assembler

```
Assembler xfemAssembler;

// diffusion
xfemAssembler.addOperatorOnSelectedElements< AssemblyStiffness >(
                            uFESpace, uFESpace, systemMatrix, alpha,
    EntityFlags::CUTTED, Flag<meshEntityFlag_Type>::testOneNotSet );

xfemAssembler.addOperatorOnSelectedElements<Xfem <AssemblyStiffness> >(
                            uFESpace, uFESpace, systemMatrix, alpha,
        EntityFlags::CUTTED, Flag<meshEntityFlag_Type>::testOneSet );

// continuity of solution imposed by penalty method
xfemAssembler.addOperatorOnSelectedElements< XfemInterface <
                                              XFEM_AssemblyJumpOnSolution > >
            ( uFESpace, uFESpace, systemMatrix,lambda,
            EntityFlags::CUTTED, Flag<meshEntityFlag_Type>::testOneSet );

...
```

`Xfem` and `XfemInterface` handle the integration and assembling for the cut elements and for the interface operators.

A diffusion problem in 2D and 3D domain

We solve on the domain $\Omega = (0,1)^d, d = 2,3$ this mono-dimensional problem:

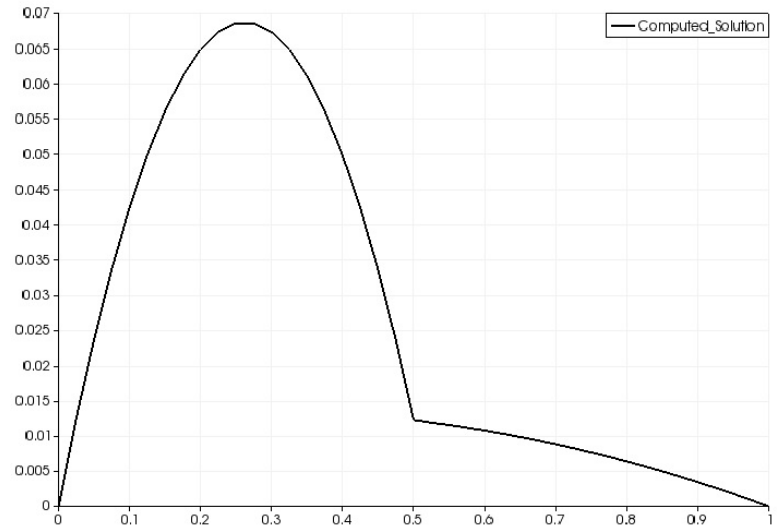
$$-\sum_i \frac{d}{dx}\left(\alpha_i \frac{du_i}{dx}\right) = 1$$

$$[\![u(1/2)]\!] = 0$$

$$\alpha_1 \frac{du_1}{dx}(1/2) - \alpha_2 \frac{du_2}{dx}(1/2) = 0$$

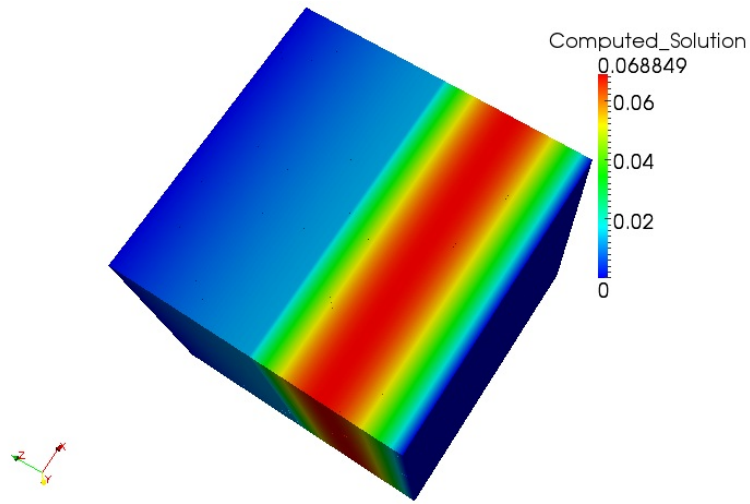
$$u_1(0) = 0$$

$$u_2(1) = 0$$



Diffusion coefficient α jumps across the surface $\Gamma = \{\mathbf{x} \quad \text{such that} \quad x = 0.5\}$.

$$\alpha = \begin{cases} 20 & \text{in } \Omega_1 \\ 0.5 & \text{in } \Omega_2 \end{cases}$$



Convergence and eigenvalues analysis (3D test)

Error analysis: XFEM vs. FEM

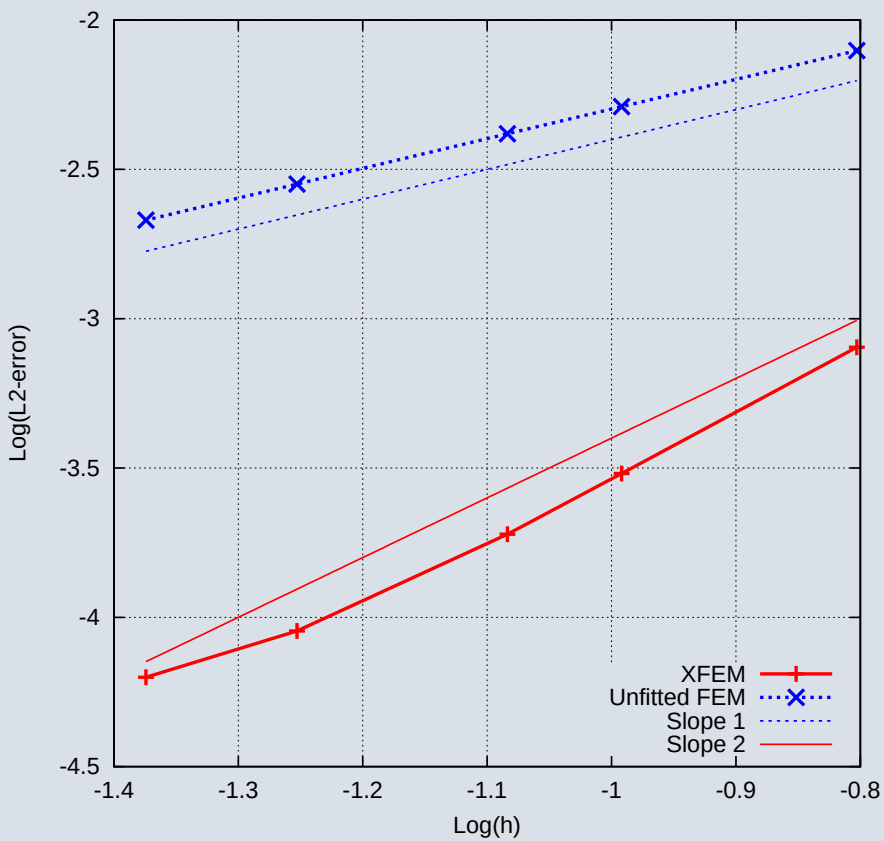


Figure : L_2-norm convergence

Matrix conditioning

$\min(K_1/K_2)$	$\lambda_{max}/\lambda_{min}$
0.015625	427.09
0.001	4105.4
1.25e-4	33960
1.56e-5	274603
1.25e-7	34529863

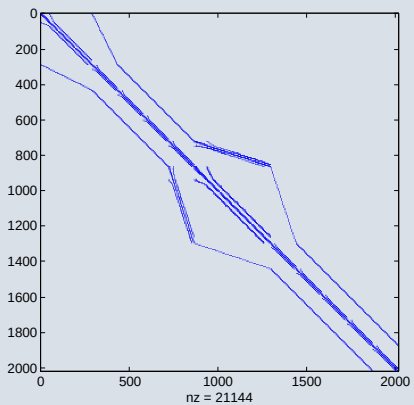
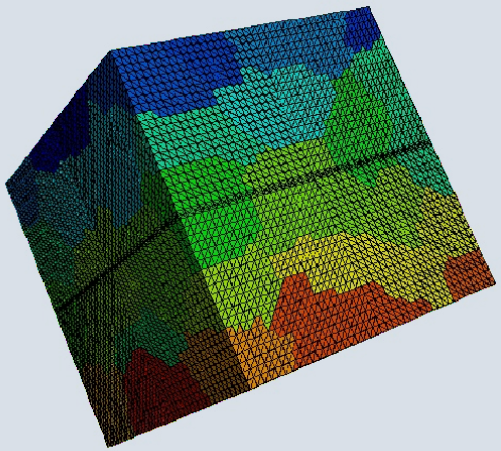


Figure : Sparsity pattern

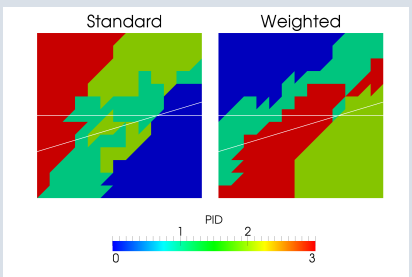
Performances of the code (3D test)

Scalability (test based on the code of December 2011)

# proc	time (no export.)
1	846.5
2	230.7
4	53.43
8	22.29
16	11.21
32	9.74
64	8.35



Partitioning and load balancing



	Weighted		Not weighted	
Processor	1	2	1	2
Assembly operators	1.60	1.49	0.89	2.26
Global assemble	2.82	2.88	3.79	3.76
Mesh adaptation	49.8	45.1	0.15	128

Test (artificial interface)

We consider the domain $\Omega = [0,1]^2$ and the following problem:

$$\begin{cases} -\mu\nabla^2(\mathbf{u}) + \nabla p = 0 & \text{in } \Omega \\ \nabla \cdot \mathbf{u} = 0 & \text{in } \Omega \\ [\![p\mathbf{n} - 2\mu\nabla\mathbf{u}\cdot\mathbf{n}]\!] = 0 & \text{su } \Gamma \\ + \text{ condizioni al bordo} \end{cases} \tag{3}$$

where $\Gamma = \{x, t | (x - \frac{1}{2})^2 + (y - \frac{1}{2})^2 = \frac{1}{4}\}$.
Analytic solution is:

$$\mathbf{u}(x,y) = \begin{cases} 20xy^3 \\ 5x^4 - 5y^4 \end{cases}$$

$$p(x,y) = 60x^2 - 20y^3 - 5$$

Stabilizations for the Stokes problem

The artificial interface can produce oscillation in the solution (in particular for the pressure) in its proximity.
We implemented and studied the behaviour of different kind of stabilizations:

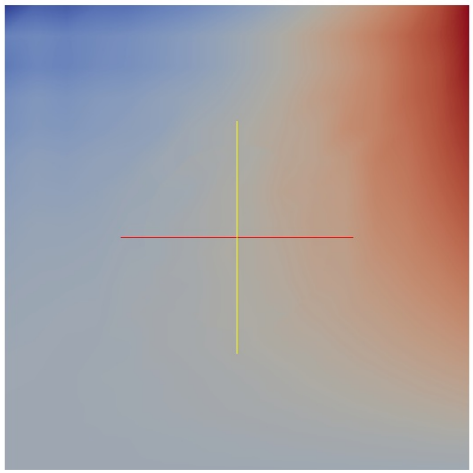
- $\mathbb{P}^1_{bubble} - \mathbb{P}^1$
- $\mathbb{P}^1_{bubble} - \mathbb{P}^1$ with Brezzi-Pitkaranta stabilization on the cut region
- $\mathbb{P}^1 - \mathbb{P}^1$ with Brezzi-Pitkaranta stabilization on all the domain
- $\mathbb{P}^1 - \mathbb{P}^0$ with a term of stabilization proposed by Burman-Becker and Hansbo:

$$J(p_h, q_h) = \sum_{F \in \mathcal{F}_1} \int_F \frac{\gamma_p}{\mu_1} h_F [\![p_{1,h}]\!][\![q_{1,h}]\!] + \sum_{F \in \mathcal{F}_2} \int_F \frac{\gamma_p}{\mu_2} h_F [\![p_{2,h}]\!][\![q_{2,h}]\!]$$

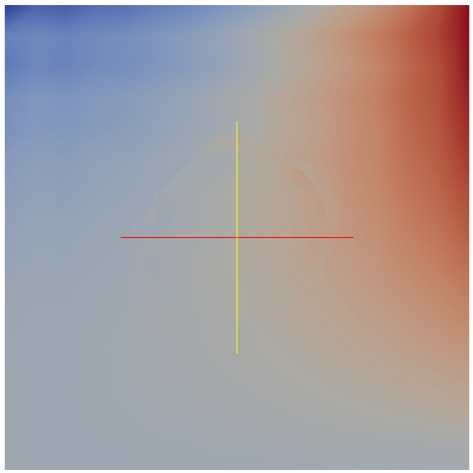
Convergence (preliminary results)

L^2 -norm error for pressure

	$h = 0.088$	$h = 0.044$	$h = 0,21$	$h = 0,11$
$\mathbb{P}^1_{bubble} - \mathbb{P}^1$	0.598	0.196	0.066	0.022
$\mathbb{P}^1_{bubble} - \mathbb{P}^1$ with BP stab.	0.535	0.194	0.065	0.023
$\mathbb{P}^1 - \mathbb{P}^1$ with BP stab	0.247	0.085	0.027	0.009



$\mathbb{P}^1_{bubble} - \mathbb{P}^1$ with BP stabilization



$\mathbb{P}^1_{bubble} - \mathbb{P}^1$

- A general `Assembler` in `LifeV`
  - Motivation
  - User interface and implementation
  - Todo
- The XFEM branch
  - The method
  - Implementation
  - Tests
  - In development
- Conclusions

Todo list for the XFEM branch

Before review...

- boundary conditions for xfem
- documentation
- warning cleaning

...and also...

- mixed elements Darcy ($\mathbb{RT}_0 - \mathbb{P}_0$)
- natural BC using flags and `Assembler`
- H^1-norm on the discontinuous space

...not really a conclusion...

Thanks for the attention, any questions?

If you don't have, no problem, don't be forced to ask something!

Enjoy the 

branch!

References

Z. Chen and J. Zhou.

Finite element methods and thier convergence for elliptic and parabolic interface problems.

Numer. Math., (79/175-202), 1998.

Thomas-Peter Fries and Ted Belytscho.

The extended/generalized finite element method: an overview of the method and its applications.

International Journal for Numerical Methods in Engineering, (84/253-034), 2010.

A. Hansbo and P. Hansbo.

An unfitted finite element method, based on nitsche's method, for elliptic interface problems.

Comput. Methods Appl. Mech. Engrg., (191/47-48), 2002.