General Assembler and XFEM in LifeV

Iori Guido
guido.iori@mail.polimi.it

Politecnico di Milano

Lausanne
January 7th, 2013
Outline

- **A general Assembler in LifeV**
  - Motivation
  - User interface and implementation
  - Todo

- **The XFEM branch**
  - The method
  - Implementation
  - Tests
  - In development

- **Conclusions**
A general Assembler in LifeV
Motivation
User interface and implementation
Todo

The XFEM branch
The method
Implementation
Tests
In development

Conclusions
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, OseenAssemble, 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
A general Assembler in LifeV
Motivation
User interface and implementation
Todo

The XFEM branch
The method
Implementation
Tests
In development

Conclusions
// 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 );
// 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 );
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 );
    }
}
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:
...
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& );
};
A general Assembler in LifeV
  Motivation
  User interface and implementation
  Todo

The XFEM branch
  The method
  Implementation
  Tests
  In development

Conclusions
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, FEFunctor, 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.
A general Assembler in LifeV
- Motivation
- User interface and implementation
- Todo

The XFEM branch
- The method
- Implementation
- Tests
- In development

Conclusions
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:

\[ U(x) = \sum_{j \in \mathcal{I}} U_j \phi_j(x) + \sum_{k} \sum_{i \in \mathcal{I}^*_k} q_i^k \mu_i(x) [\psi^k(x) - \psi^k(x_i)] \]

where \( \mathcal{I} \) is the set of the nodes and \( \mathcal{I}^*_k \subset \mathcal{I} \) is the subset on which the enriching functions \( \psi^k \) are localized [2].
An elliptic interface problem

Differential problem

Let $\Omega$ be a bounded domain in $\mathbb{R}^3$, with convex polygonal boundary $\partial \Omega$ and an internal smooth boundary $\Gamma$ dividing $\Omega$ into two open sets $\Omega_1$ and $\Omega_2$. We consider the following stationary problem

\[
\begin{aligned}
-\nabla \cdot (\alpha \nabla u) &= f & \text{in } \Omega_1 \cup \Omega_2 \\
u &= 0 & \text{on } \partial \Omega \\
[u] &= 0 & \text{on } \Gamma \\
[\alpha \nabla n u] &= g & \text{on } \Gamma
\end{aligned}
\]  

(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 $\Omega$. Let $V^h = V^h_1 \times V^h_2$ where

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

for $i = 1, 2$.

**Discrete problem [3]**

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

$$a_h(U, \phi) = L(\phi), \ \forall \phi \in V^h$$

where

$$a_h(U, \phi) = (\alpha_i \nabla U_i, \nabla \phi_i)_{\Omega_1 \cup \Omega_2} - ([U], \{ \alpha \nabla n \phi \})_\Gamma - ([\alpha \nabla n U], [\phi])_\Gamma + (\lambda [U], [\phi])_\Gamma$$

$$L(\phi) = (f, \phi)_\Omega + (\kappa_2 g_1, \phi)_\Gamma + (\kappa_1 g_2, \phi)_\Gamma,$$

and $\lambda$ is a penalty parameter.

$\{ \cdot \}$ denotes a weighted mean: $\{ v \} = \kappa_1 v_1 + \kappa_2 v_2$ where $\kappa_1, \kappa_2$ are appropriate weights.
We define $F_h = \{K \in T_h : K \cap \Gamma \neq \emptyset\}$. For each element $K \in F_h$, let $K_i \in \Omega_i$ denote the part of $K$ in $\Omega_i$.

**FE basis for $V^h$**

A local basis function $\phi$ on $K \in F_h$ must be discontinuous across $\Gamma$:

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

$\phi_1$ is represented in $K_1$ by its nodal values in $K$ and the same holds for $\phi_2$. Since $\phi_1$ and $\phi_2$ 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.
We now consider a domain crossed by two level-set function, $f(x)$, $g(x)$. $\Omega$ is generally partitioned in 3 or 4 sub-domains $\Omega_i$ 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 $\Omega_i$.

As a consequence, the number of degrees of freedom of an element $K \in (F_h \cap G_h)$ is dependent on the number of sub-domain $\Omega_i$ such that $K \cap \Omega_i \neq \emptyset$. 
A general Assembler in LifeV
Motivation
User interface and implementation
Todo

The XFEM branch
The method
Implementation
Tests
In development

Conclusions
Introducing a discontinuity in the domain

**Level-set**

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

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

$$\Omega_1 = \{x \in \Omega \text{ such that } f(x) > 0\}$$

$$\Omega_2 = \{x \in \Omega \text{ such that } f(x) < 0\}$$

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

```cpp
XfemMeshHandler< LinearTetra > xfemhandler( fullMeshPtr, surfaces );
xfemhandler.intersectMeshSurfaces();
```
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. \( f(x) > 0 \), 2 if \( f(x) < 0 \) and 1 if the element is crossed by the surface defined by the equation \( f(x) = 0 \).

In case of two interfaces, regions are defined as follows:

<table>
<thead>
<tr>
<th>( f_1(x) )</th>
<th>( f_2(x) &gt; 0 )</th>
<th>( f_2(x) = 0 )</th>
<th>( f_2(x) &lt; 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1(x) &gt; 0 )</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( f_1(x) = 0 )</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>( f_1(x) &lt; 0 )</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

This approach can be easily generalized.
Problems

Dof assignment 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?
A new DOF class

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:

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

```java
xfemhandler.modifiedMesh(); // refinement
meshPtr = xfemhandler.meshPartition(); // we take the new mesh
```
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 );
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 general Assembler in LifeV
- Motivation
- User interface and implementation
- Todo

The XFEM branch
- The method
- Implementation
- Tests
- In development

Conclusions
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 $\alpha$ jumps across the surface $\Gamma = \{x \text{ such that } x = 0.5\}$.

$$\alpha = \begin{cases} 
20 & \text{in } \Omega_1 \\
0.5 & \text{in } \Omega_2 
\end{cases}$$
Convergence and eigenvalues analysis (3D test)

**Error analysis: XFEM vs. FEM**

- Log(L2-error) vs. Log(h)

- XFEM
- Unfitted FEM

<table>
<thead>
<tr>
<th>( \min(K_1/K_2) )</th>
<th>( \lambda_{max}/\lambda_{min} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.015625</td>
<td>427.09</td>
</tr>
<tr>
<td>0.001</td>
<td>4105.4</td>
</tr>
<tr>
<td>1.25e-4</td>
<td>33960</td>
</tr>
<tr>
<td>1.56e-5</td>
<td>274603</td>
</tr>
<tr>
<td>1.25e-7</td>
<td>34529863</td>
</tr>
</tbody>
</table>

**Matrix conditioning**

**Figure: L_2-norm convergence**

**Figure: Sparsity pattern**
Performances of the code (3D test)

**Scalability (test based on the code of December 2011)**

<table>
<thead>
<tr>
<th># proc</th>
<th>time (no export.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>846.5</td>
</tr>
<tr>
<td>2</td>
<td>230.7</td>
</tr>
<tr>
<td>4</td>
<td>53.43</td>
</tr>
<tr>
<td>8</td>
<td>22.29</td>
</tr>
<tr>
<td>16</td>
<td>11.21</td>
</tr>
<tr>
<td>32</td>
<td>9.74</td>
</tr>
<tr>
<td>64</td>
<td>8.35</td>
</tr>
</tbody>
</table>

**Partitioning and load balancing**

<table>
<thead>
<tr>
<th></th>
<th>Weighted</th>
<th>Not weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Assembly operators</td>
<td>1.60</td>
<td>1.49</td>
</tr>
<tr>
<td>Global assemble</td>
<td>2.82</td>
<td>2.88</td>
</tr>
<tr>
<td>Mesh adaptation</td>
<td>49.8</td>
<td>45.1</td>
</tr>
</tbody>
</table>
Stokes problem

Test (artificial interface)

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

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

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

Analytic solution is:

\[
\begin{align*}
 u(x,y) &= \begin{cases} 
 20xy^3 \\
 5x^4 - 5y^4
\end{cases} \\
p(x,y) &= 60x^2 - 20y^3 - 5
\end{align*}
\]
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:

- \( \mathbb{P}_{1}^{\text{bubble}} - \mathbb{P}^{1} \)
- \( \mathbb{P}_{1}^{\text{bubble}} - \mathbb{P}^{1} \) 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:

\[
J(p_h, q_h) = \sum_{F \in \mathcal{F}_1} \int_F \frac{\gamma_p}{\mu_1} h_F [p_h, q_h] [q_1, h] + \sum_{F \in \mathcal{F}_2} \int_F \frac{\gamma_p}{\mu_2} h_F [p_h, q_h] [q_2, h]
\]
### Convergence (preliminary results)

#### $L^2$-norm error for pressure

<table>
<thead>
<tr>
<th></th>
<th>$h = 0.088$</th>
<th>$h = 0.044$</th>
<th>$h = 0.21$</th>
<th>$h = 0.11$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P^1_{bubble} - P^1$</td>
<td>0.598</td>
<td>0.196</td>
<td>0.066</td>
<td>0.022</td>
</tr>
<tr>
<td>$P^1_{bubble} - P^1$ with BP stab.</td>
<td>0.535</td>
<td>0.194</td>
<td>0.065</td>
<td>0.023</td>
</tr>
<tr>
<td>$P^1 - P^1$ with BP stab</td>
<td>0.247</td>
<td>0.085</td>
<td>0.027</td>
<td>0.009</td>
</tr>
</tbody>
</table>

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

**$P^1_{bubble} - P^1$**
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 ($RT_0 - P_0$)
- natural BC using flags and Assembler
- $H^1$-norm on the discontinuous space
A general Assembler in LifeV
Motivation
User interface and implementation
Todo

The XFEM branch
The method
Implementation
Tests
In development

Conclusions
...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!
Z. Chen and J. Zhou.
Finite element methods and thier convergence for elliptic and parabolic interface problems.

Thomas-Peter Fries and Ted Belytscho.
The extended/generalized finite element method: an overview of the method and its applications.

A. Hansbo and P. Hansbo.
An unfitted finite element method, based on nitsche’s method, for elliptic interface problems.