INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2006; **2**:1–33 *Prepared using* nmeauth.cls *[Version: 2002/09/18 v2.02]*

An extended finite element library

Stéphane Bordas [1,*], Phu Vinh Nguyen [2], Cyrille Dunant [3], Hung Nguyen-Dang [4] and Amor Guidoum [3]

[1] *Ecole Polytechnique Fédérale de Lausanne, Laboratory of Structural and Continuum Mechanics, EPFL-ENAC-IS-LSC, Station 18, CH-1015, Lausanne, Switzerland.*
[2] *Student of European Master in Engineering Science Mechanics of Constructions, EMMC, Hochiminh University of Technology, 268 Ly Thuong Kiet, Vietnam*
[3] *Ecole Polytechnique Fédérale de Lausanne, Laboratory of Construction Materials, EPFL-STI-IMX-LMC, MXG 241 CH-1015, Lausanne, Switzerland.*
[4] *Prof. Dr. Nguyen-Dang Hung Institut de Mécanique et Génie Civil, Bâtiment B52/3 Chemin des Chevreuils 1, B-4000 Liège 1, Belgique E-mail : H.NguyenDang@ulg.ac.be*

SUMMARY

This paper presents and exercises a general structure for an object-oriented enriched finite element code. The programming environment provides a robust tool for extended finite element (XFEM) computations and a modular and extensible system. The program structure has been designed to meet all natural requirements for modularity, extensibility, and robustness. To facilitate mesh-geometry interactions with hundreds of enrichment items, a mesh generator and mesh database are included. The salient features of the program are: flexibility in the integration schemes (subtriangles, subquadrilaterals, independent near-tip and discontinuous quadrature rules); domain integral methods for homogeneous and bi-material interface cracks arbitrarily oriented with respect to the mesh; geometry is described and updated by level sets, vector level sets or a standard method; standard and enriched approximations are independent; enrichment detection schemes: topological, geometrical, narrow-band, etc.; multi-material problem with an arbitrary number of interfaces and slip-interfaces; non-linear material models such as J2 plasticity with linear, isotropic and kinematic hardening. To illustrate the possible applications of our paradigm, we present two-dimensional linear elastic fracture mechanics for hundreds of cracks with local near-tip refinement, and crack propagation in two dimensions as well as complex three-dimensional industrial problems. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: Object-oriented programming, C++, partition of unity enrichment, extended finite element method, fracture mechanics, dense fissuration, crack propagation, industrial problems, open source

*Correspondence to: stephane.bordas@alumni.northwestern.edu until July 1st 2006: Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne, Switzerland, Laboratory of Structural and Continuum Mechanics, LSC-ENAC-EPFL, Station 18, Tel. +41(0)21 693 2403, Fax. +41(0)21 693 6340.; After July 1st, 2006: Department of Civil Engineering, University of Glasgow, GLASGOW, G12 8LT; Tel: +44(0)141 330 5204 Fax: +44(0)141 330 4557

Received January 4th, 2006
Revised April 13th, 2006
Accepted June ??th, 2006

1. Introduction

We present in this paper a novel open source architecture for the extended finite element method aiming at easing the development of this method by researchers. The Partition of Unity Method (PUM) [1] allows for the addition of a priori knowledge about the solution of a boundary value problem into the approximation space of the numerical solution through the addition of enrichment functions that may be exactly reproduced by the enriched numerical scheme. The extended finite element method (XFEM) uses a local partition of unity [2, 3], where some of the enrichment functions are chosen to be discontinuous thereby allowing for the reproduction of strong discontinuities within an element. Only a portion of the mesh is enriched. Another similar application of PUM is known as the Generalized Finite Element Method (GFEM) [4,5]. Very recently, another related method emerged with the work of Hansbo and Hansbo [6], based on Nitsche's method, in which the authors give a convergence proof. Further, in Reference [7], it is shown that the kinematics of the method described in [6] is in fact equivalent to that of the extended finite element method. Based on Nitsche's method, recent work has been performed in the area of discontinuity modeling within a finite element framework [8, 9, 10]. A review on computational modeling of cohesive cracks is given in [11].

Any function may be introduced in the approximation. In particular, singular functions may be added into the approximation to decrease the required mesh density close to singularities such as crack tips in linear elastic fracture mechanics (LEFM). Enriched finite element methods have been successfully applied to numerous solid mechanics problems such as 2D crack growth problems in linear elastic fracture mechanics with small displacements [2, 3, 12, 13], and large displacement [14, 15]. Extensions to 3D were presented in [16, 17, 18] and further improved to handle crack initiation and propagation in [19] and multiple cracks [20]. Other applications include multiple crack growth in brittle materials [21], crack growth in shells and plates [22], cohesive crack growth [23, 24, 25, 26, 27, 28, 29, 11], bi-material interface cracks [30], holes and inclusions [31,32], brittle fracture in Polycrystalline Microstructures [33], shear bands [34] and, finally, contact problems [35]. Multiscale work has been performed with micro-macro crack models based on the LATIN methods (LArge Time INcrement method) in [36]. The XFEM has also been used to model computational phenomena in areas such as fluids mechanics, phase transformations [37], material science and biofilm growth [38, 39], Chemically-induced swelling of hydrogels [40], among others. Fluid-structure interaction problems are also free-boundary problems, where the "free" boundary usually is the structure. Recent work has been performed in this area by several researchers [41]. Recent developments of enriched finite element methods in conjunction with discontinuous Galerkin in time, for dynamics and time dependent problems have been recently made [42, 43, 44, 45]. XFEM was also recently introduced in spectral elements [46]. A nice work on interface conditions is given in [47].

For a complete review on recent developments of both XFEM and GFEM, interested readers can refer to the papers of Q.Z. Xiao and B.L. Karihaloo at Cardiff University [48, 49].

The PUM may be applied in the context of meshfree methods such as the Element Free Galerkin method (EFG) [50, 51, 52, 53], yielding enriched methods where crack tip fields [54], discontinuous derivatives [55] can be incorporated into the meshfree approximation. A recent overview of meshfree methods may be found in [56, 57]. One of the drawbacks of the meshfree methods lies in the tricks needed to handle the essential boundary conditions due to the lack of the so-called Kronecker-Delta property of meshfree approximation functions. Another drawback of the meshfree methods is that it is difficult and awkward to convert

in-house finite element codes into a meshfree code. However, meshfree methods are very powerful tools that were successfully used to model fracture in concrete, for instance [58, 59, 60, 61, 62, 63, 64, 65, 66, 67].

Enriched finite element methods such as the XFEM, contrary to meshfree methods, can be implemented within a finite element code with relatively small modifications: variable number of degrees of freedom (dofs) per node; mesh geometry interaction (a procedure to detect elements intersecting with the geometry of the discontinuities); enriched stiffness matrices; numerical integration. The goal of this paper[†] is to present one way these additional features can be added into an existing finite element code, which is a topic of interest not only to finite element software companies desirous to include enrichment into an existing code, but also to academics who would like to extend their in-house program. Moreover, we tackle all known difficult and critical parts in the implementation and explain them in detail, so as to ease the understanding. We strongly believe that the proposed XFEM library[‡] is simple enough to give beginning graduate students an easy start.

In order to produce extendable code, in which new problem formulations may easily be added, the Object-Oriented Programming (OOP) approach gives the largest flexibility [69]. An object oriented enriched finite element library, named OpenXFEM++, which is based on FEMOBJ, an object oriented finite element code [69], has been built and used to solve various 2D LEFM problems with great success. The combination of an object-oriented XFEM code and a commercial FE software allows the solution of very complex three-dimensional industrial problems was presented in [70].

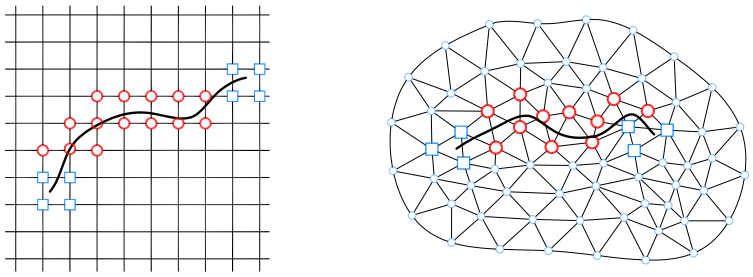
The paper is organized as follows. In section 2, a brief introduction to enriched FEM for crack modeling is given. Section 3 expresses how relevant enrichment concepts can be easily expressed in an Object-Oriented framework. In particular, the class diagram for the code OpenXFEM++ is presented, and the key classes described. Section 4 presents a possible extension of the code to new problems, in particular the addition of new enrichment functions is easily made possible due to the chosen design technique. Numerical examples are presented in Section 5. Finally, Section 6 provides a summary and some concluding remarks. The class tree is given in appendix A and a typical data file of OpenXFEM++ is presented in appendix B.

2. Crack modeling with partition of unity enrichment

The extended finite element method [3, 13] provides a simple and efficient treatment of cracks where the element topologies do not conform to the crack geometry. Some elements are split by the crack and others contain the crack tips. Nodes whose support is bisected by a crack are collected in $\mathcal{N}_{cr}$, while nodes whose support contains the tips are grouped in the set $\mathcal{N}_{tip}$. In Figure 1, these sets of nodes are shown.

The crack is represented in the XFEM by enriching the standard displacement approximation

[†]Details can be found in the Master's thesis [68], available upon request to the first author
[‡]provided on request



(a) (b)

Figure 1. Selection of enriched nodes for 2D crack problem. Circled nodes (set of nodes $\mathcal{N}_{cr}$) are enriched by the step function whereas the squared nodes (set of nodes $\mathcal{N}_{tip}$) are enriched by the crack tip functions. (a) on structured mesh; (b) on unstructured mesh.

as follows:

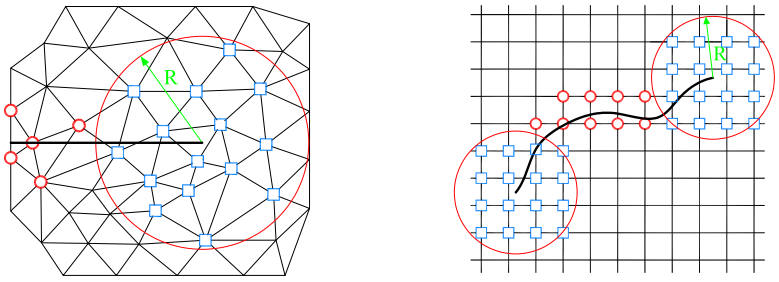
$$\begin{aligned}\mathbf{u}^h(\mathbf{x}) = &\sum_{I\in\mathcal{N}} N_I(\mathbf{x})\mathbf{u}_I + \sum_{J\in\mathcal{N}_{cr}} \widetilde{N}_J(\mathbf{x})(H(\mathbf{x}) - H(\mathbf{x}_J))\mathbf{a}_J \\ &+ \sum_{K\in\mathcal{N}_{tip}} \widetilde{N}_K(\mathbf{x}) \sum_{\alpha=1}^{4} (B_\alpha(\mathbf{x}) - B_\alpha(\mathbf{x}_K))\mathbf{b}_{\alpha K}\end{aligned} \tag{1}$$

where $N_I(\mathbf{x})$ and $\widetilde{N}_J(\mathbf{x})$ are finite element shape functions, while $\mathbf{u}_\mathrm{I}$, $\mathbf{a}_\mathrm{J}$ and $\mathbf{b}_{\alpha\mathrm{K}}$ are the displacement and enrichment nodal variables, respectively. Note that the shape functions $\widetilde{N}_J(\mathbf{x})$ localizing the enrichment can differ from the shape functions $N_I(\mathbf{x})$ for the displacement approximation (for instance, high order shape functions $N_I(\mathbf{x})$ but linear shape functions $\widetilde{N}_J(\mathbf{x})$ were used for the six-noded triangular elements [71]). $H(\mathbf{x})$ is the modified Heaviside function which takes on the value +1 above the crack and -1 below the crack and $B_\alpha(\mathbf{x})$ is a basis that spans the near tip asymptotic field:

$$\mathbf{B} \equiv [B_1, B_2, B_3, B_4] = \left[\sqrt{r}\sin\frac{\theta}{2}, \sqrt{r}\cos\frac{\theta}{2}, \sqrt{r}\sin\frac{\theta}{2}\cos\theta, \sqrt{r}\cos\frac{\theta}{2}\cos\theta\right] \tag{2}$$

It was shown [72, 73] that the rate of convergence of PU enriched finite element methods such as the XFEM is considerably improved by fixing the near-tip enrichment domain (area in 2D, volume in 3D) independently of the mesh size. Defining a circle $C(\mathbf{x_0}, R)$ with predefined radius R whose center is the crack tip $\mathbf{x_0}$, any node located inside this circle is enriched with the crack tip functions, i.e., belongs to the set of nodes $\mathcal{N}_{tip}$ (see Figure 2). At this point, no clear rule has been found that relates the optimal –from an accuracy standpoint– enrichment radius to the characteristic dimensions of the structure.

From this enriched approximation (1), using the Bubnov-Galerkin procedure, the standard discrete equation $\mathbf{Kd} = \mathbf{f}$ is obtained. The numerical integration for elements cut by the crack can be done by partitioning split elements into sub-triangles. Recently, a method requiring no



(a) (b)

Figure 2. Another scheme for selection of enriched nodes for 2D crack problem. Circled nodes (set of nodes $\mathcal{N}_{cr}$) are enriched by the step function whereas the squared nodes (set of nodes $\mathcal{N}_{tip}$) are enriched by the crack tip functions.

background cells was presented, but is restricted to linear elastic materials and cracks that do not kink inside elements [74]. When the solution of the standard discrete equations is obtained, the stress intensity factors are computed using the domain form of the interaction integral. The crack propagation direction is calculated from the maximum hoop stress criterion. Interested readers can refer to [68, 75] for details.

3. Object-oriented enriched finite element implementation

3.1. Additional classes used to describe an enriched finite element problem

The process of object-oriented design involves two main steps. First, the key concepts in the *application world* (for instance, the finite element world consisting of Nodes, Elements, Materials, Solvers, etc.) need to be isolated. Second, those concepts should be transformed into classes, i.e., building blocks that can be used to formulate a problem in the application world. The problem at hand requires handling *enrichment* of the standard Finite Element Method with *a priori* knowledge about the solution to the boundary value problem. Some of the key concepts involved in the XFEM are enrichment items (crack, hole, inclusion, material interface, biofilm, solid-fluid interface etc.), enrichment functions (Heaviside function and branch functions etc.) and geometry entities (polylines, points, circles etc.) which are implemented through the classes **EnrichmentItem**, **EnrichmentFunction** and **GeometryEntity**, respectively. Numerical integration needs special attention for enriched elements [3]. This is handled through the **IntegrationRule** class. To have a flexible technique for selection of enriched nodes, for example, the standard way (i.e. sub-triangle generation see Figure 1) or fixed enrichment area (Figure 2), the class **EnrichmentDetector** is designed. The crack growth law is implemented through class **CrackGrowthDirectionLaw** and its derived class **MaxHoopStress** whereas the rules determining the crack advance length is handled by class **CrackGrowthIncrementLaw** and its derived classes **FixedIncrement**

and **ParisLaw**.

Besides these classes, some of the existing FE classes such as **Domain, Element, Node** also demanded modifications. The modification of such classes as well as the implementation of new ones are detailed next.

3.1.1. Enrichment items A general enriched finite element problem may hold a number of features such as cracks, holes, material interfaces, sliding interfaces, contact interfaces, fluid-solid interfaces etc. In our implementation, those features are objects of class **EnrichmentItem**. Such an object holds its geometry, an object of class **GeometryEntity** described in Section 3.1.4 –used to check whether the enrichment item interacts with a given element–, and its enrichment functions, object of class **EnrichmentFunction**, used to model it. The interface of the **EnrichmentItem** class is given in Figure 3.

```
class EnrichmentItem : public FEMComponent {
public:
   EnrichmentItem(int,Domain*);
   virtual ~EnrichmentItem();

   void                              getGeometry();
   vector< EnrichmentFunction* >*    giveEnrFuncVector();
   bool                              interactsWith(Element*);
   virtual void                      treatEnrichment();
   EnrichmentDetector*               defineMyEnrDetector();
   EnrichmentDetector*               giveMyEnrDetector();
   GeometryEntity*                   giveMyGeo();
   vector<Element*>*                 giveElementsInteracWithMe();
   void                              setListOfInteractedElements(Element*);
   virtual void                      resolveLinearDependency(){}
   virtual void                      updateYourGeometry(){;}
   virtual void                      updateEnrichment(){;}
protected:
   GeometryEntity*                   myGeometry;
   vector<EnrichmentFunction*>*      myEnrichFns ;
   EnrichmentDetector*               myEnrDetector;
   vector<Element*>*                 interactedElements ;
} ;
```

Figure 3. The interface of class **EnrichmentItem**

In the present implementation, three derived classes of the base class **EnrichmentItem** are designed as shown in Figure 4[§]. A 2D crack with two tips is modeled by an object of class **CrackInterior** (see Figure 5) and two objects of class **CrackTip** (see Figure 6). A **CrackInterior** object holds its tips through the data member *myTips*. The method *treatMeshGeoInteractionForMyTips* is used to find elements containing these tips.

The member *myAssociatedCrackInterior* stores the associated crack interior of this tip. A **CrackTip** object uses this member to update the geometry and the list of elements cut by its associated crack interior. The stress intensity factors (mode I, mode II and the equivalent

[§]generated by Doxygen [76]

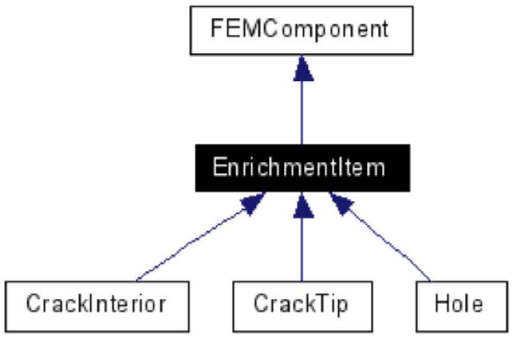


Figure 4. The inheritance tree of class **EnrichmentItem**

```
class CrackInterior : public EnrichmentItem
{
public:
   CrackInterior(int,Domain*);
   ~CrackInterior(){;}

   void                    getMyTips();
   std::vector<CrackTip*> giveMyTips();

   void     treatMeshGeoInteractionForMyTips() ;
   void     resolveLinearDependency();

   void     updateEnrichment(){;}
   void     updateMyGeometry();
private:
   std::vector<CrackTip*>  myTips;
} ;
```

Figure 5. C++ header file of a 2D crack interior

stress intensity factor) are stored in variables *K_i, K_ii, K_eq* , respectively.

For crack growth problems and especially multiple crack growth problems, it is necessary to be able to kill crack tips, when they reach free boundaries –domain boundary or another crack's interior. The data member *bool isActive* is implemented for this purpose, initially set to *true*, and changed to *false* when the tip should be killed.

The key method of class **CrackTip** is the interaction integral computation, *computeInteractionIntegral(TimeStep*)*, which allows to compute the stress intensity factors, from which the crack propagation direction and increment are deduced.

```
class CrackTip : public EnrichmentItem {
public:
  CrackTip(int,Domain*);
  ~CrackTip();

  void                  computeSIFs(TimeStep* stepN);
  void                  computeK_eq(TimeStep* stepN);
  std::list<Element*>   buildIntegrationDomain();
  std::valarray<double> computeInteractionIntegral(TimeStep* stepN);

  bool  giveState()const {return isActive;}
  void  kill(){isActive = false;}
  void  crackTypeInitialization();
  void  crackTypeUpdate();

  std::vector<Material*>*   giveMatArray();

  Mu::Segment*   giveTipSegment();
  FloatArray*    computeLocalCoordOf(Mu::Point* p);

  double         giveRadiusOfDomainIntegration();
  double         giveEnrichRadius();

  void           resolveLinearDependency(){}

  void                 updateMyGeometry();
  Mu::Circle*          DefineDomainForUpdatedEnrichment();
  std::list<Element*>  defineUpdatedElements();
  void                 updateEnrichment();

  void    setMyAssociatedCrackInterior(CrackInterior* crInt);

private:
    CrackType               tipID;
    FieldType               field;
    std::vector<Material*>* matArray;
    double                  K_i,K_ii,K_eq ;
    Mu::Segment*     tipSegment ;
    bool             isActive;
    CrackInterior*   myAssociatedCrackInterior ;
} ;
```

Figure 6. C++ header file of a 2D crack tip

3.1.2. Enrichment functions The class **EnrichmentFunction** implements specific functions holding a priori knowledge about the solution (branch functions for linear elastic fracture mechanics) or particular functions used to model the discontinuities (Heaviside function for displacement discontinuity and ramp function for strain discontinuity).

An object of class **EnrichmentFunction** should know how to evaluate itself (*EvaluateYourSelfAt(Point*)*) and its gradient (*EvaluateYourGradAt(Point*)*) at a given point in space. To do so, it requires the geometry and the nature of the enrichment item with which it is associated. A crack tip, for instance, knows which coordinate system to use to compute

```
class EnrichmentFunction : public FEMComponent {
public:
    EnrichmentFunction(Domain*,int);
    ~EnrichmentFunction();

    virtual double      EvaluateYourSelfAt(Mu::Point*){return NULL;}
    virtual FloatArray* EvaluateYourGradAt(Mu::Point*){return NULL;}

    void                setMyEnrichmentItem(EnrichmentItem*);
    void                findActiveEnrichmentItem(EnrichmentItem*);
protected:
    int number;
    std::vector<EnrichmentItem*> *myEnrItems;
    EnrichmentItem               *activeEnrItem;
} ;
```

Figure 7. Interface of class **EnrichmentFunction**

the asymptotic enrichment functions (Equation (2)) through its geometry. If several instances of the same enrichment item are to be modelled in the same problem (for instance, multiple crack problem), some nodes may require enrichment by two identical enrichment functions, belonging to two different enrichment items. Consequently, we chose to tell each enrichment item which enrichment function it is associated with, and vice versa. Therefore, an object of class **EnrichmentFunction** holds a list of objects of **EnrichmentItem** being modeled by this enrichment function. The data member *myEnrItems* (see Figure 7) is designed for this purpose. To know for which **EnrichmentItem** the enrichment function should be computed, we design the data member *activeEnrItem*.

The **EnrichmentFunction** class serves as an *interface* to the code, because it specifies all methods that should be implemented by an enrichment function, but actually implements none of them (abstract class). This is a safe method to make sure a new programmer implements *all* required methods, since this check is made at *compile time*. The inheritance tree of the **EnrichmentFunction** class is given in Figure 8¶.

An example of a derived class‖ is the class **HomoElastCrackAsymp**, which specializes into the computation of the asymptotic crack tip fields for an homogeneous linear elastic material. For a future programmer to add enrichment functions for, say, a Neo-Hookean material, it is sufficient to implement a **HomoNeoHookeanCrackAsym** class, which will only differ from the **HomoElastCrackAsymp** class, by the implementation of its evaluation methods.

3.1.3. Numerical integration For split elements, to exactly integrate the weak form, a widely used method is to partition the element into subelements**. The task of class **SplitGaussLegendreQuadrature** lies in finding the coordinates and weights of the quadrature points to be used to integrate the weak-form accurately. First, the intersection

¶generated by Doxygen [76]
‖specialized forms of enrichment functions
**or rather sub-integration cells since no extra dof is associated with the sub-triangles.

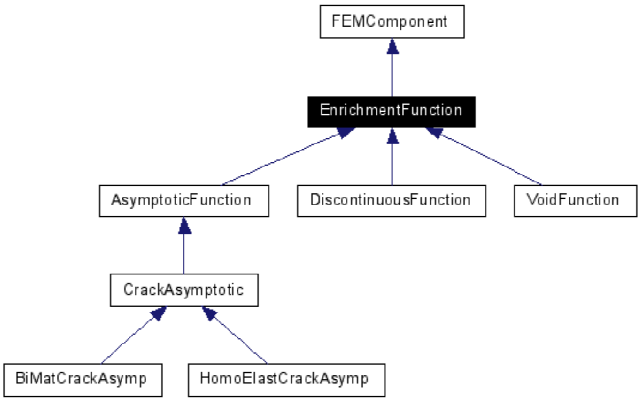


Figure 8. The inheritance tree of class **EnrichmentFunction**

points of the element edges with the geometry of the enrichment item are computed. Then, they are fed into a Delaunay mesher [77, 78] together with the element's nodes, yielding the sub-integration triangles. Finally, for each triangle, the integration points are obtained, and their coordinates transformed into the parent coordinate system of the split element. In order to make seamless the addition of new integration techniques, this class is derived from an interface abstract class: **IntegrationRule**.

Special integration techniques are required for near-tip enriched finite elements (since the integrand in the stiffness matrix is no longer polynomial). To enable this, suitable integration rules are derived from the **IntegrationRule** abstract class.

3.1.4. Geometry handling In our implementation, geometry can be handled a number of ways (Standard, Level Set, Vector Level Set, etc.). Similarly, the update of the geometry can easily be chosen. The XFEM is naturally coupled with the level set method [79,31] and, more recently, the vector level set was proposed, for crack growth problems, as a simpler alternative to the standard level set method. Duflot et al. [80] review the drawbacks of existing level set methods for which an improved version, leading to optimally accurate stress intensity factors, is proposed. Standard geometry handling for the **EnrichmentItem** is also possible, and used in the numerical examples, along with a Tree structure to manage both geometrical entities and finite elements. The advantages of using an integrated mesh generator are detailed in the paper by Dunant et al. [78].

Each **EnrichmentItem** knows its geometry as an object of class **GeometryEntity**, say *myGeo*. Through *myGeo*, the enrichment item knows whether it interacts with a given element. In turn, *myGeo*, through its **GeometryDescription** member, knows how to perform this interaction (by standard geometry or using level sets).

This piece of code is given as follows:

```
bool GeometryEntity :: interactsWith(Element* e) {
   return this->giveMyGeoDescription()->interactsWith(e,this);}
```

For example, for a **CrackInterior** represented by standard geometry (line segments), the usual geometry predicates are used as in [13] to check the intersection between an element and the crack. It is noted that the level sets are used only for description of the geometry of the discontinuities, not to generate the mesh. For complex structures or microstructures, however, using level sets to build the mesh is promising.

3.1.5. Classes specific to the enriched nodes detection A flexible XFEM code should allow the user to decide the criterion based on which enriched nodes are selected. To detect near-tip enriched nodes, we can name: nodes of elements containing the tip and inclusion in a ball of radius R centered at the crack tip as possible criterions. Similarly, only nodes belonging to an element split by a discontinuity should be enriched with the Heaviside function, while, for biofilm problems [39,38], elements within the biofilm, and below a given distance from the biofilm/water interface often need to be asymptotically enriched in order to track the boundary layer.

The abstract class **EnrichmentDetector** and its derived classes serve the purpose of making the selection of enriched nodes flexible. Abstract class **EnrichmentDetector** is designed with only one pure virtual method *void setEnrichedNodes(EnrichmentItem *enrItem)*. It is obvious that we can not have objects of class **EnrichmentDetector** since a general **EnrichmentDetector** does not know how to select enriched nodes. That is why we design this class as an abstract class. Therefore, its derived classes have to implement the method *setEnrichedNodes(EnrichmentItem*)*.

For linear elements, it is sufficient to enrich all of the nodes of a split element. However, for higher order elements, the choice of which nodes should be enriched depends on the partition of unity shape functions to be used. If linear shape functions are used for the enriched part, only the corner nodes need to be enriched –not the mid-side nodes. This is implemented in method *setEnrichmentForMyNodes(EnrichmentItem *enrItem)* of class **Tri6**.

3.2. Modification of standard finite element classes

3.2.1. Class Domain The domain [69] can be considered as the main object that contains all the problem's components: list of nodes, elements, materials, loads, etc. It serves as a link between the components needed to describe the physical and numerical problem. To account for the discontinuities, the following data members and methods are added as shown in Figure 9.

The main method of this class is the solution procedure *solveFractureMechanicsProblemAt(TimeStep* stepN)* as shown in Figure 10. Given the aforementioned classes of objects, solving an enriched finite element problem now reduces to handling the mesh geometry interaction to find out elements that interact with the **EnrichmentItem**s and enrich the corresponding nodes with the approriate **EnrichmentFunction**s. If there are conflicts, those are resolved by the method *resolveLinearDependencyForEnrichment()*. Finally, the **Domain** object asks its **NLSolver** object to *Solve()* the problem using the appropriate scheme [69].

```
class Domain {
 private :
  List*                 enrichmentFunctionList ;
  List*                 enrichmentItemList ;
  List*                 geoEntityList ;
  bool                  isFEM ;
  bool                  isXFEM ;
  CrackGrowthDirectionLaw* directionLaw ;
  CrackGrowthIncrementLaw* incrementLaw ;
 public :
  EnrichmentItem*       giveEnrichmentItem(int n);
  EnrichmentFunction*   giveEnrichmentFunction(int n)
  GeometryEntity*       giveGeoEntity(int n) ;

  void                  solveFractureMechanicsProblem ();
  void                  solveFractureMechanicsProblemAt(TimeStep*);

  void                  treatMeshGeoInteractionPhase1();
  void                  treatMeshGeoInteractionPhase2();
  void                  treatEnrichment() ;
  void                  resolveConflictsInEnrichment();
  void                  resolveLinearDependencyForEnrichment();
} ;
```

Figure 9. Added data and methods for class **Domain** to account discontinuities

```
void  Domain :: solveFractureMechanicsProblemAt(TimeStep* stepN)
{
    if (unknownArray) {
        delete unknownArray;
    }

    this -> treatMeshGeoInteractionPhase1();
    this -> treatMeshGeoInteractionPhase2();
    this -> treatEnrichment();
    this -> resolveConflictsInEnrichment();
    this -> resolveLinearDependencyForEnrichment();

    unknownArray = this -> giveNLSolver() -> Solve();

    this -> terminate(stepN) ;
}
```

Figure 10. Method *solveFractureMechanicsProblemAt(TimeStep* stepN)*

3.2.2. Class Element The data members that are added for the **Element** class include the following:

FEInterpolation* standardFEInterpolation used as interpolation functions for classical finite approximation.

FEInterpolation* enrichmentFEInterpolation shape functions multiplied by the enrichment functions to form the PUM shape funcions. It was shown that using higher order shape functions for this approximation decreases the error, but makes the convergence rate erratic, because of the loss of reproducing condition in partially enriched elements. [72, 81].

std::list<EnrichmentItem*>* enrichmentItemListOfElem Every **EnrichmentItem** objects interacting with one element are stored in this list. **Element** e uses this list to do the partitioning for numerical integration.

bool isUpdated a marker to know a given element is needed to recompute its stiffness matrix when the **EnrichmentItem** involves as time goes by.

New methods are added :

```
bool              isEnriched();
void              isEnrichedWith(EnrichmentItem *enrItem);
void              treatGeoMeshInteraction();
virtual void      setEnrichmentForMyNodes(EnrichmentItem*){}

virtual FEInterpolation*    giveFEInterpolation(){return NULL;}
virtual FEInterpolation*    giveXFEInterpolation(){return NULL;}

void    setUpIntegrationRule();
virtual vector<DelaunayTriangle*>* PartitionMySelf(){return NULL;}
virtual GaussPoint** setGaussQuadForJ_Integral(){return NULL;}

void    setStateOfElement(){isUpdated = true ;}
bool    isUpdatedElement(){return isUpdated ;}
```

The method *isEnriched()* consists in looping over the receiver's nodes (the current instance of the **Element** object), if at least one node is enriched, then the element is enriched. The **Element** needs to know if it is enriched or not to correctly compute the **B** matrix (equation (3)) and use the appropriate integration rule.

When an element is enriched by an **EnrichmentItem** then the method *isEnriched-With(EnrichmentItem *enrItem)* inserts *enrItem* into its enrichmentItemListOfElem.

The definition of the elemental stiffness matrix, $\mathbf{K}_e$, of element e occupying a volume Ω_e writes

$$\mathbf{K}_e = \int_{\Omega_e} \mathbf{B}^T \mathbf{D} \mathbf{B} d\Omega_e \tag{3}$$

where **D** is the constitutive matrix and **B** is the matrix of shape function derivatives including the enriched part. The method *computeBMatrixAt(Gausspoint *gp)* was modified as shown in Figure 11. The first advantage of this approach is that the code to compute the stiffness matrix of any element (enriched or not) is unchanged. Secondly, a node can be enriched with

any number of enrichment functions. For instance, an element may be split by a crack and also contain the tip of another crack. In this case, nodes of this element are enriched by both the step function (of the first crack) and the crack asymptotic functions (of the second one).

```
FloatMatrix*  Element :: ComputeBmatrixAt(GaussPoint *aGausspoint){
   Compute the standard part of B matrix, called Bu
   If this element is not enriched, return Bu and stop.
   Otherwise, loop on nodes, if node is enriched then
   Loop on enrichment items of this node, say enrItem
      Each enrItem knows which enrichment functions should be used.
      Loop on these enrichment functions and calculate their
      contribution to B matrix, say Ba
   The final result is B = [Bu Ba]
}
```

Figure 11. Method *ComputeBmatrixAt(GaussPoint*)*

To do the element partitioning for finding Gauss points for split and tip elements, method *PartitionMySelf()* is implemented. It is a pure virtual method: its derived classes such as **TriU, QuadU**, and **Tri6** have their own implementation.

Since the Gauss points used for the computation of the interaction integral are different from those used in the stiffness matrix computation (see also Section 3.1.3), the method *setGaussQuadForJIntegral()* is designed. It is a pure virtual method since each element type potentially requires different integration schemes.

It is obvious that when **EnrichmentItem** objects change, for example when cracks grow, only some nodes and elements around the crack tips should change status. The rest of the elements are unchanged and it is not efficient to recompute their stiffness matrix. The data member *isUpdated* is implemented to handle this problem. Initially, this member is set to false, after the crack growth step, if a given element is detected to be changed then its *isUpdated* is reset to true i.e., its matrix needs to be recomputed.

3.2.3. Class Node To handle the nodal enrichment, the following data members are added to this class

int isEnriched a marker to differentiate non-enriched and enriched nodes.

list<EnrichmentItem*>* enrichmentItemListOfNode This is list of all **EnrichmentItem** objects acting on the node.

and here are added methods

```
void              isEnrichedWith(EnrichmentItem* enrItem);
void              resolveConflictsInEnrichment();
void              resolveLinearDependency(EnrichmentItem*);

int                    getIsEnriched(){return isEnriched;}
list<EnrichmentItem*>* giveEnrItemListOfNode();
```

A node should not be enriched with both the Heaviside function and branch functions associated with the same crack. Therefore, whenever the data member *enrichmentItemListOfNode* contains objects of both class **CrackInterior** and class

CrackTip, one should remove the **CrackInterior** object so that this node is just enriched with branch functions[††]. This is performed by method *resolveConflictsInEnrichment()*.

For any node enriched by the Heaviside function $H(\mathbf{x})$, its support is fully cut into two disjoint pieces by the crack. If for a certain node n_I, one of the two pieces is very small compared to the other, then the function $H(\mathbf{x})$ is almost a constant over the support, leading to an ill-conditioned stiffness matrix [3]. In this case, node n_I is no longer enriched with the function $H(\mathbf{x})$ –method *resolveLinearDependency(EnrichmentItem*)*.

With enriched finite elements, the number and nature of the degrees of freedoms associated with a node may vary from node to node and, in addition, evolve with time. Therefore, the way to compute the number of dofs and the location of dofs in the global matrix must be modified. Below are modified and added methods implemented for this purpose :

```
size_t          computeNumberOfDofs () ;
size_t          giveNumberOfTrueDofs () ;
size_t          giveNumberOfDofs () ;

IntArray*       giveStandardLocationArray () ;
IntArray*       giveEnrichedLocationArray () ;
```

For plane elasticity problems, the number of dofs, n_{DOF}, of any node is determined by

$$n_{DOF} = \begin{cases} 2 & \text{if node is not enriched} \\ 2 + 2n_{enr} & \text{otherwise} \end{cases} \tag{4}$$

where n_{enr} is the number of enrichment functions used to enrich this node. n_{enr} is the total number of enrichment functions of all enrichment items acting on this node. It is important to note that Equation (4) must be addapted for the case of a slip interface[‡‡], in which only one additional degree of freedom per node is required since the discontinuity is only along one direction. The code permits the treatment of such cases with no additional care since enrichment is treated at the degree of freedom level. In other words, upon calculation of the number of degrees of freedom and their equation number in the global system of equations, it is checked whether the active degree of freedom is enriched.

4. Extension to new problems

This section explains how the object-oriented approach allows to easily extend the current code to include new problems.

Here, assume that we want to solve interfacial crack problems. First, the asymptotic functions associated with this problem need to be added [30]. To do so, it suffices to build a new class called **BiMaterialElastCrackAsymp** in which the near tip asymptotic functions for interfacial cracks are implemented. It is emphasized that this new class is completely similar to class **HomoElastCrackAsymp**.

[††]the $\theta \longmapsto \sin\theta/2$ branch function is discontinuous through $\theta = \pm\pi$ since $\sin\pi/2 = -\sin(-\pi/2)$, therefore, tip elements' nodes do not need to be enriched with the Heaviside function since the discontinuity is provided naturally by the branch functions.

[‡‡]allowing for a jump in the tangential displacement while maintaining continuity in the normal direction

Finally, to compute the stress intensity factors (SIF), we need to implement the auxiliary fields used in the domain integral computations. Here, there are two possibilities : (1) use inheritance, i.e., one would implement an abstract class, say **AuxiliaryFields**, with pure virtual methods to compute the components of any auxiliary field and two derived classes, one for homogeneous cracks and one for interfacial cracks. This approach is simple but leads to code duplication, thus is not effective; (2) use templates. Following this way, one implements a single template class. In the current implementation, we chose the template approach as given in Figure 12.

```
template<class M1, class M2, const FieldType field=PlaneStrain>
class AuxiliaryField
{
public:
   AuxiliaryField(){};
   virtual ~AuxiliaryField(){};
   void ComputeComponentsOfAuxField(CrackTip*,Point*,ModeType,
       FloatMatrix&,FloatArray&,FloatArray&);
   void ComputeComponentsOfOneMat(CrackTip* tip,Point*,ModeType,
       FloatMatrix&,FloatArray&,FloatArray&);
   void ComputeComponentsOfBiMat(CrackTip* tip,Point*,ModeType,
       FloatMatrix&,FloatArray&,FloatArray&);
protected:
   M1 *material1;
   M2 *material2;
   size_t giveNumberOfMaterials() const{
      return (size_t)(typeid(M2) != typeid(NullMaterial))+1 ;}
};
```

Figure 12. Interface of class **AuxiliaryField**

If the auxiliary fields of the homogeneous crack (under plane strain condition) are needed, the declaration goes as follows

```
AuxiliaryField<Material,NullMaterial,PlaneStrain> *auxFieldHomo
   = new AuxiliaryField<Material,NullMaterial,PlaneStrain>();
```

If the auxiliary fields for a bimaterial crack are needed, then one declares as follows

```
AuxiliaryField<Material,Material,PlaneStrain> *auxFieldBiMat
   = new AuxiliaryField<Material,Material,PlaneStrain>();
```

5. Numerical applications

We present numerical results for the computation of the stress intensity factors (SIFs) and study the crack growth simulations in isotropic homogeneous and heterogeneous media. First, the examples on static crack problems are considered with the following objectives:

1. to verify and test the OpenXFEM++ library;

2. to show the computational advantage of embedding a mesher into an extended finite element code;
3. to show the accuracy obtainable on unstructured as well as structured meshes which are relatively coarse;
4. to study the domain independence in the SIF computations;
5. to assess the accuracy of the SIF computations in three dimensions;
6. to show how XFEM can be used for industrial problems;

Various crack growth problems are then solved to show the strength of the XFEM. In particular, crack growth in multimaterial media is studied through two examples, namely the crack growth from a fillet and crack inclusion interaction. We also perform a simple computation concerning the influence of the coating of inclusions on crack propagation, where both the inclusion and the coating are handled through enrichment with a discontinuous derivative function –also known as ramp function, i.e. the absolute value of the signed distance function. To complete the numerical applications, we show how the XFEM may be used for a complex, industrial, three-dimensional crack growth problem.

For the 2D examples, plane strain conditions are assumed throughout. Linear elements (three-noded triangular element and four-noded quadrilateral) are used for all computations. The calculation of the stress intensity factors is performed with the domain form of the interaction integral. Quasi-static crack growth is governed by the maximum hoop stress criterion, and the crack growth increment is chosen in advance and constant for all steps in the 2D simulations, and governed by the Paris law in the 3D simulations.

5.1. Inclined crack in tension

Stress intensity factors are calculated for a plate with an angled center crack shown in Figure 13. The plate is subjected to a far field uniaxial stress as shown. It is emphasized that the same mesh (structured triangular mesh consisting of only 1520 elements) is used for all angles considered, with only three elements along the length of the crack.

The plate dimensions are taken to be $W = 10$in. with a half crack length of $a = 0.5$in. As the plate dimensions are large in comparison to the crack length, the numerical solution can be compared to the solution for an infinite plate. For the load shown, the exact stress intensity factors are given by:

$$K_I = \sigma\sqrt{\pi a}\cos^2\beta, \quad K_{II} = \sigma\sqrt{\pi a}\sin\beta\cos\beta \tag{5}$$

Numerical results for the SIFs are obtained for $\beta = 15^o$, 30^o, 45^o, 60^o, 70^o, and domain independence of the J integral computation is also studied. In Table I, the normalized SIFs are compared to the exact solution: r_d is the radius of the domain used for the interaction integral computation and h_{local} is the size of the tip element. Excellent agreement between the numerical solution and the exact solution is obtained for this coarse mesh with only three elements along the crack length.

Additionally, we observe that with $r_d/h_{local} = 3.0$ and 3.5, the obtained SIFs are incorrect. The reason for this is that the domain size is big enough to include the other tip element.

The SIFs are also computed for other angles with the domain size used in the interaction integral computation is $r_d = 2.5h_{local}$ and plotted in Figure 14. The result shows excellent agreement with exact solution for the entire range of β.

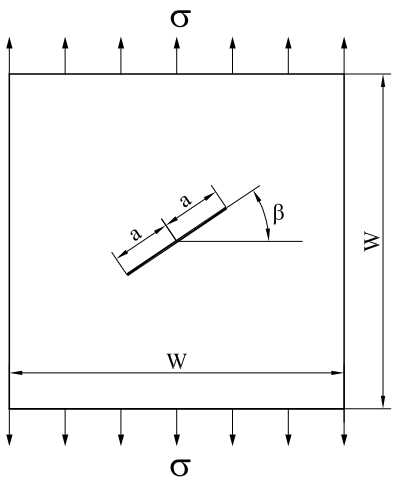


Figure 13. Inclined crack in tension

Table I. Normalized SIFs for the inclined crack problem. r_d is the radius of the domain used for the interaction integral computation and h_{local} is the size of the tip element.

β	SIFs	Exact	XFEM				
			$r_d/h_{local}=1.5$	2.0	2.5	3.0	3.5
15^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.9330	0.9313	0.9316	0.9312	0.9791	0.9882
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.2500	0.2760	0.2512	0.2489	0.2607	0.2613
30^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.7500	0.7232	0.7486	0.7484	0.7787	0.7770
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.4330	0.4028	0.4413	0.4413	0.4455	0.4427
45^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.5000	0.4836	0.4897	0.5010	0.5159	0.5132
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.5000	0.4767	0.5009	0.5022	0.5132	0.5108
60^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.2500	0.2000	0.2581	0.2549	0.2582	0.2596
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.4330	0.3783	0.4406	0.4366	0.4563	0.4547
75^o	$\frac{K_I}{\sigma\sqrt{\pi a}}$	0.0670	0.0589	0.0673	0.0690	0.0692	0.0689
	$\frac{K_{II}}{\sigma\sqrt{\pi a}}$	0.2500	0.2115	0.2526	0.2535	0.2565	0.2560

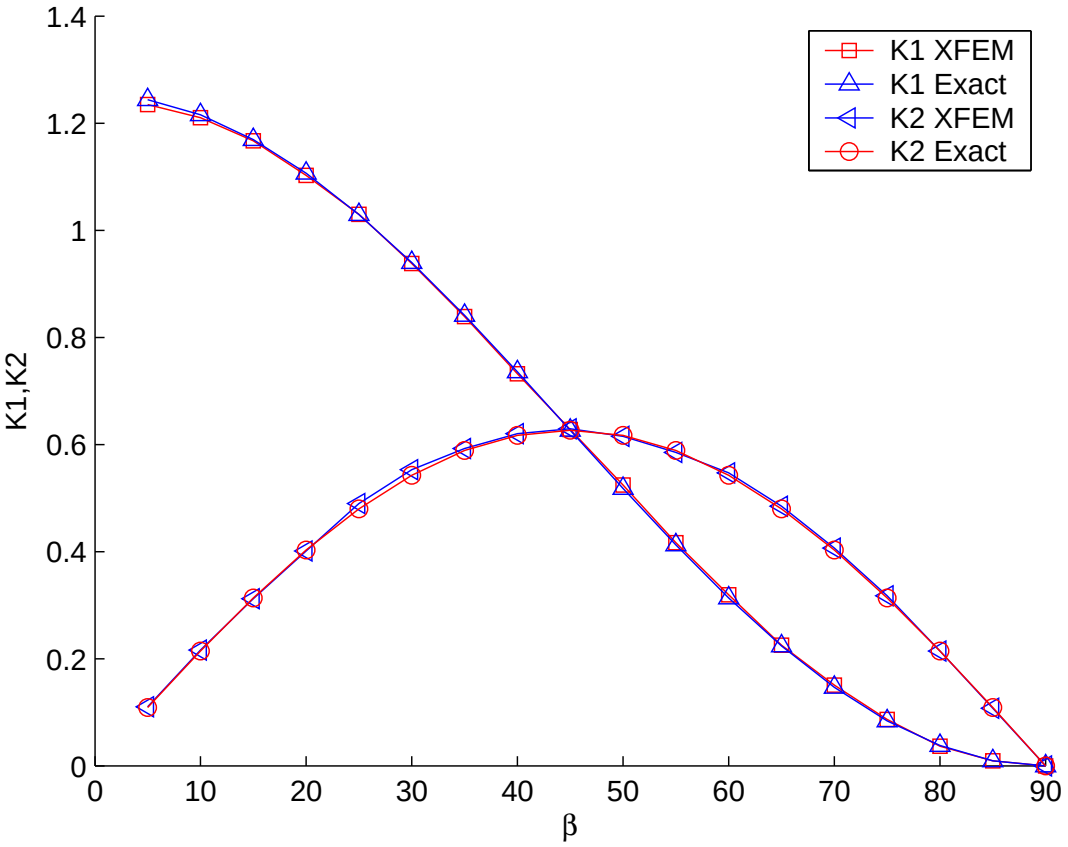


Figure 14. K_I and K_{II} vs. β for a plate with an angle center crack

5.2. Densely micro cracked solid

Although there are numerous publications on the XFEM, little attention has been cast onto efficient implementation. To the authors knowledge, Reference [78] is the only work assessing the CPU time consumption of various *mesh-geometry interaction* algorithms. In Dunant et al. [78], the authors show that by retaining mesh information from the mesh generator and storing it in tree structures allowing for fast access and retrieval, dramatic performance gains are attained. For cases where thousands of enrichment items are present, such as in dense micro fissuration problems or materials with complex micro structures, it is crucial to devise an efficient *mesh-geometry interaction* module. By integrating an internal mesher to XFEM codes [78], the enrichment detection step is optimal, whilst the ability to generate general, well adapted meshes is retained. As an illustrating example, consider a rectangular plate with four hundred (400) cracks of random length and orientation as shown in Figure 15. The displacements are fixed on the left and right ends of the sample, symmetrically whereas the top and bottom edges are left free.

In this example, the mesh-geometry computation step took less than a second. Such a

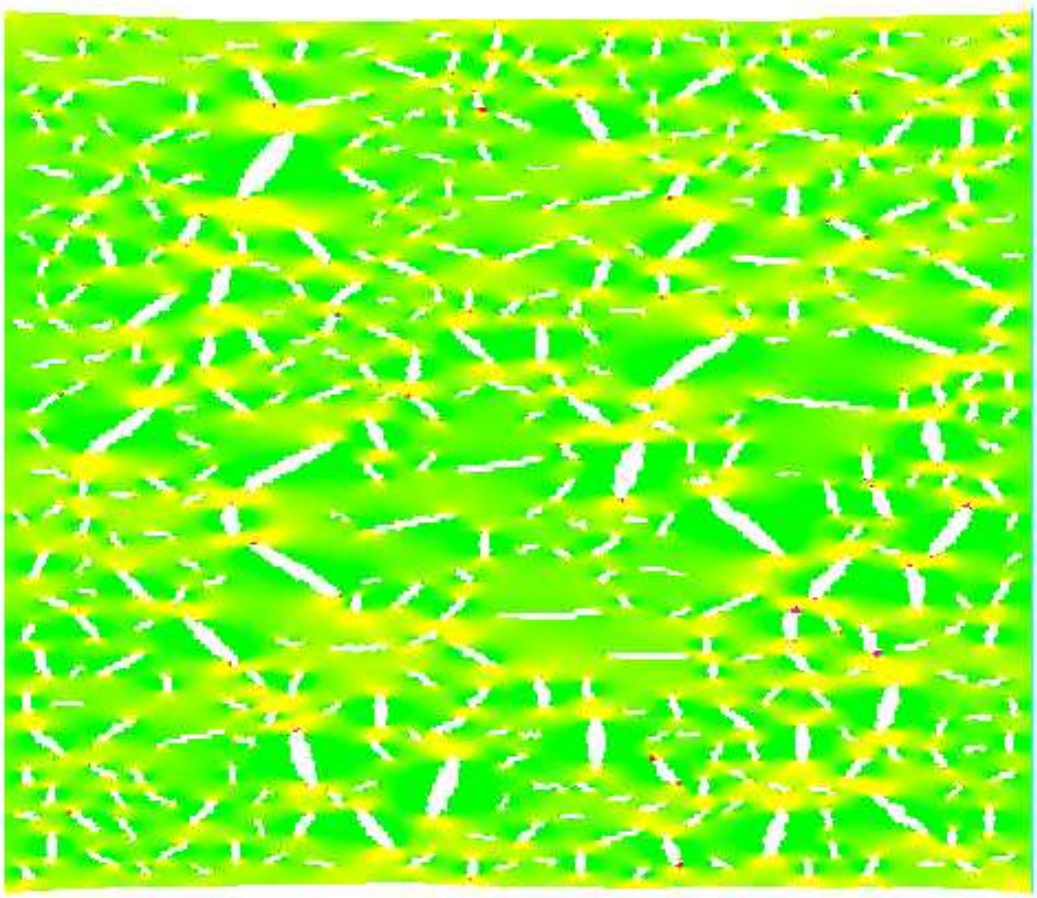


Figure 15. One of the principal stresses, computed around approximatively 400 cracks.

case uses approximatively 200,000 degrees of freedom, including enrichment. The longest step is solving the linear system itself and the total analysis time is about 12mn on a desktop computer. The assembly step takes 20% longer, than an equivalently sized assembly with no enrichment. To solve the same problem with conventional finite elements (without quarter point elements), we created a mesh holding 253,652 elements, 142,345 nodes and 1,024,998 unknowns, with sufficient refinement close to the eight hundred (800) crack tips in the domain. The total analysis time was thirty minutes (30mn), including meshing, assembly, and solve with a preconditioned conjugate gradient, on the same desktop computer. This comparison is only crude since we did not compare the results for both cases. Reference [78] provides details on the efficiency of the proposed approach.

5.3. Crack growth from a fillet

This example together with the next one are presented in order to demonstrate the influence of material heterogeneity on crack paths. Figure 16 shows the experimental configuration for crack growth from a fillet [82]. The crack path depends on the welding residual stresses and the bending stiffness of the structure. Here, residual stresses are not taken into account. The bending stiffness of the structure is modified by varying the thickness of the lower I-beam. Only limiting cases for the bottom I-beam of a rigid constraint (very thick beam) and flexible constraint (very thin beam) are examined.

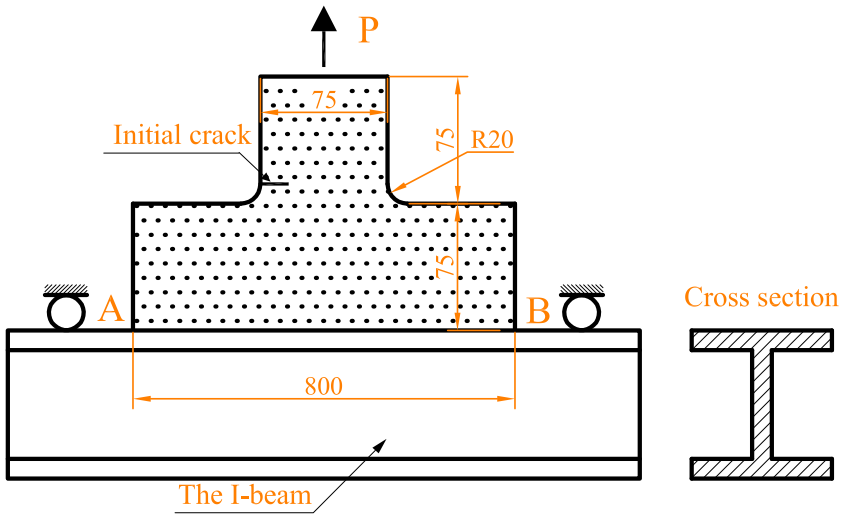


Figure 16. Experimental configuration for crack growth from a fillet. The dotted region is the computational domain and the boundary conditions are imposed along line AB.

The structure is loaded with a traction of $P = 20$kN, and the initial crack length is taken to be of $a = 5$mm. The computational domain is the dotted region and is discretized with 7108 three-noded triangular elements. The effects of the thickness are incorporated into the problem through the Dirichlet boundary conditions. For a rigid I-beam, the displacement in the vertical direction is fixed on the entire bottom edge. A flexible beam is idealized by fixing the vertical displacement at only both endpoints of the bottom edge. For both sets of boundary conditions, an additional degree of freedom is fixed to prevent rigid body motions.

Crack growth is simulated with a fixed crack increment length of 1mm at each step. Figure 17 shows the mesh in the vicinity of the fillet and compares the crack paths for the cases of a thick I-beam (upper crack) and thin I-beam (lower crack). The results are consistent with both experimental [82] and previous numerical results using the Element Free Galerkin method [54] and advanced remeshing technique [83].

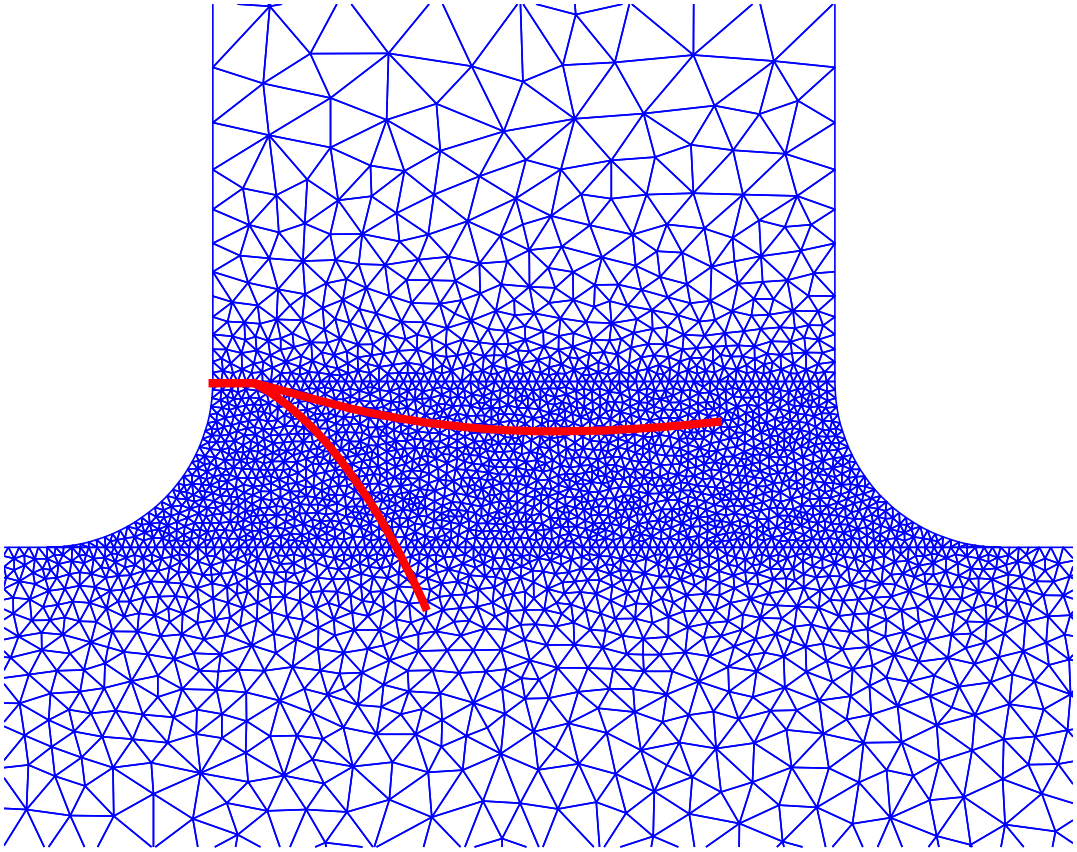


Figure 17. Zoom of crack paths for the case of a thick (top crack) and thin (bottom crack) I-beam.

5.4. Crack inclusion interaction

A rectangular plate with an off-centered inclusion is pre-cracked and subjected to a tensile stress as shown in Figure 18. Both cases of uncoated and coated inclusion are considered. Let R be the ratio of the matrix to the inclusion Young's moduli, $R = E_{matr}/E_{incl}$ and the Poisson's ratio is kept the same for the matrix, the inclusion and the coating.

The numerical simulations are performed for two cases, namely (1) hard inclusion ($R = 0.1$) and (2) soft inclusion ($R = 10$). Figure 19 shows the calculated crack paths using 4292 three-noded triangular elements and a crack increment length of 0.1. One observes that, for a soft inclusion (Figure 19(a)), the crack is attracted to the inclusion. Conversely, if the inclusion is

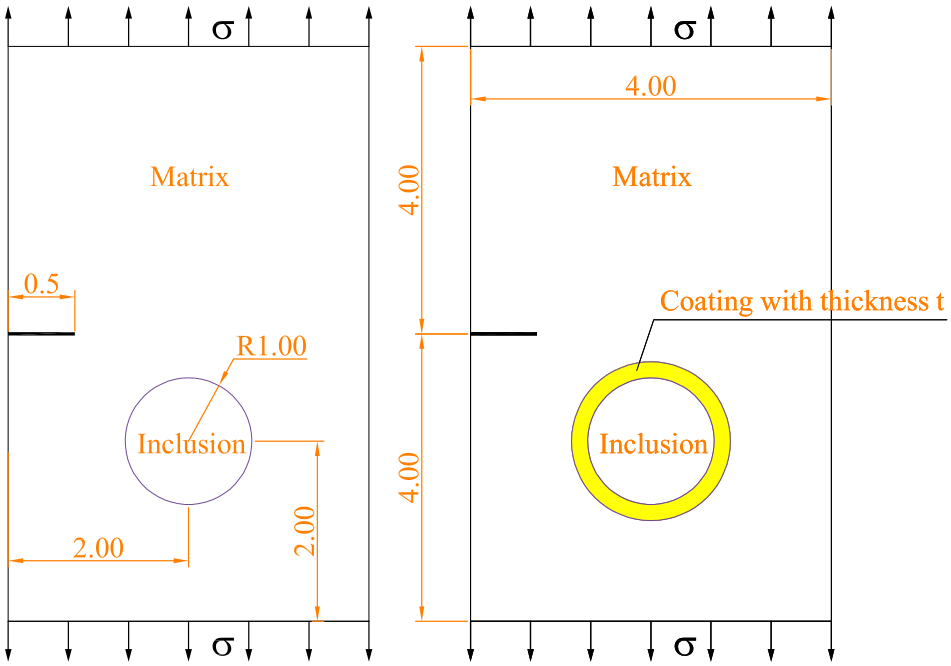
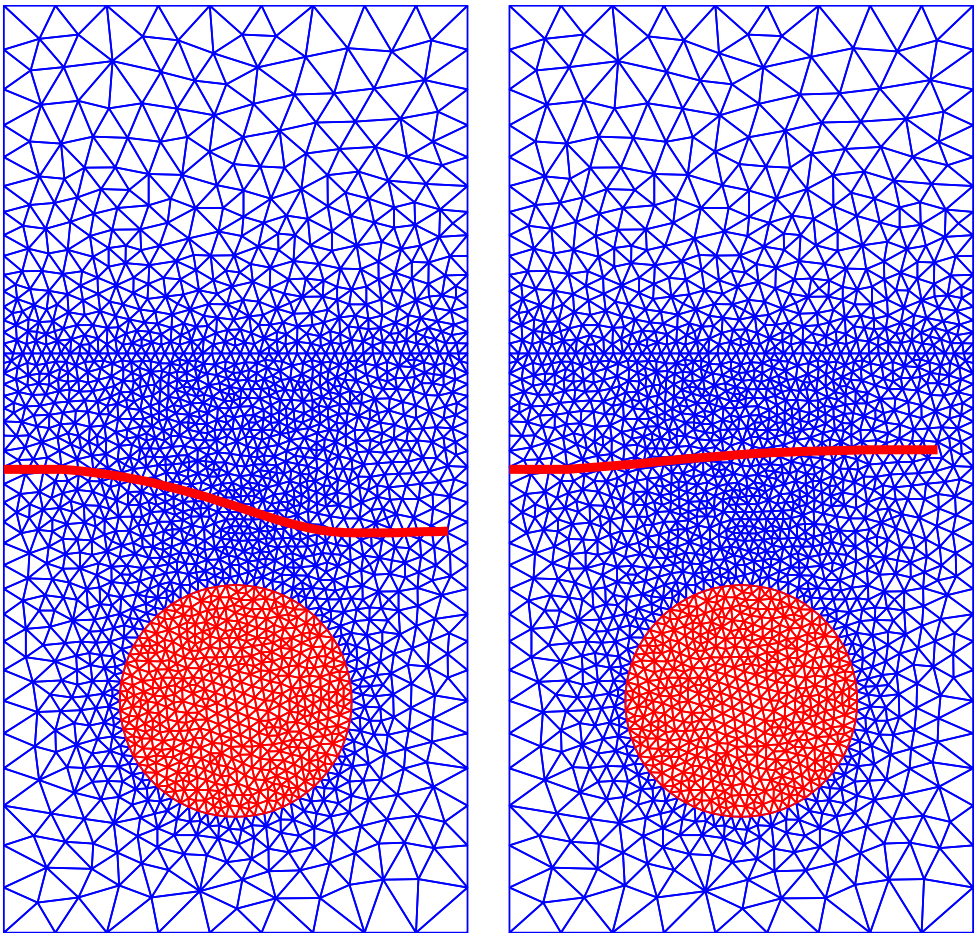


Figure 18. Crack inclusion interaction problem: uncoated inclusion (left figure) and coated inclusion (right figure)

more rigid than the matrix (Figure (19(b))), the crack is moving away from the inclusion. The obtained results are consistent with those using remeshing techniques [83].

In the following the influence of mesh densities on the simulated crack path is studied. Two unstructured meshes of 2018 and 4292 elements are considered. The crack increment lengths are always about the tip element's size. Results given in Figure 20 show that, for sufficiently fine meshes and an appropriate crack increment length (about the tip element's size), the simulated crack paths are almost identical.

When studying the influence of inclusion coating on crack propagation around an inclusion with the FEM, one mesh must be created for each coating thickness under consideration. Using the XFEM with material interface enrichment [31], this burden is lifted and only one mesh may be used for all coating thicknesses. In what follows, the ratio R is taken equal to 10, i.e. the coating is ten times harder than the matrix and of thickness $t = 0.2$. A fully non-conforming mesh is built and all discontinuities (crack and two material interfaces) are represented through enrichment. Although the coating thickness is only 0.2 and it is just ten times harder than the matrix, the crack path given in Figure 21 shows its considerable effect on crack path, as the coating prevents the crack from being attracted by the soft inclusion. The purpose of this analysis is to show the flexibility of the enriched finite elements. For more details on this subject, readers could refer to the work of Knight et al. [84].



(a) (b)

Figure 19. Simulated crack paths:(a) Soft inclusion; and (b) Hard inclusion

5.5. Penny crack in a finite cube under uniaxial tension

In this example, the geometry of the problem is a cube of side $b = 2$mm at the center of which a penny crack of radius a is embedded. The cube is subjected to uniaxial tension $\sigma_\infty = 1\text{mN/mm}^2$. The stress intensity factors along the front are computed for the initial crack. The crack is then grown under fatigue assuming the Paris law For an infinite domain, the closed form solution is known for this problem, namely

$$K_I = 2\sigma_\infty \sqrt{\frac{a}{\pi}} \tag{6}$$

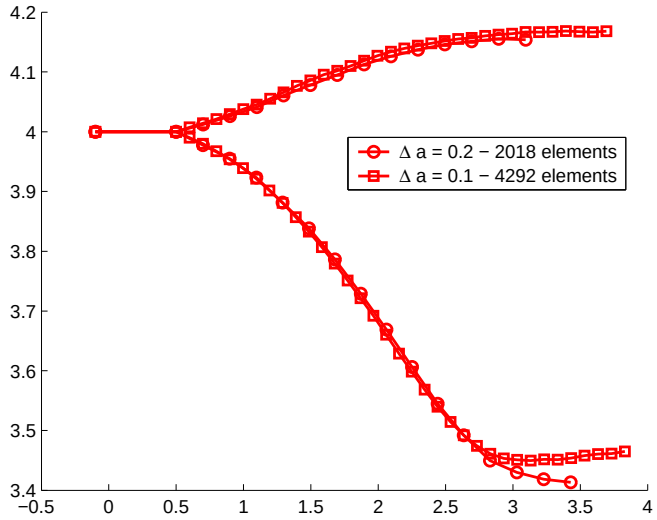


Figure 20. Calculated crack trajectories for two mesh densities. The top corresponds to the case of the hard inclusion whereas the lower curve is associated with the soft inclusion

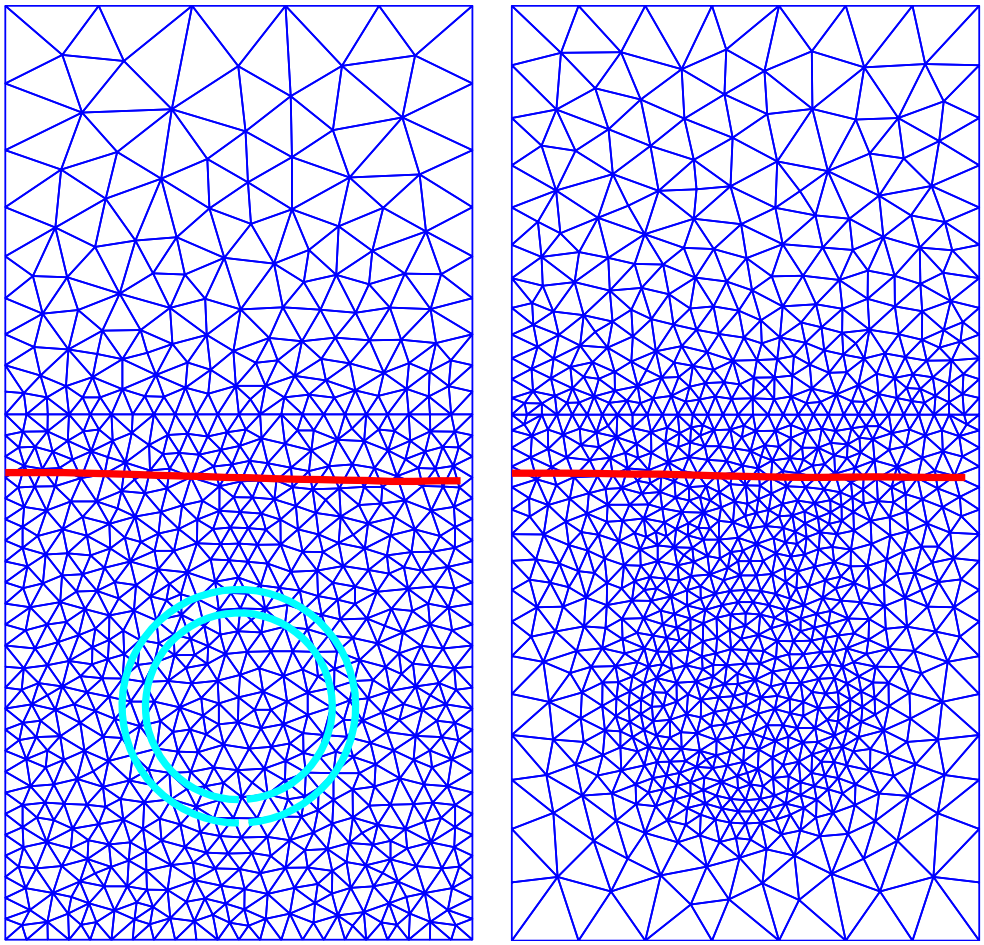
Mode I, II and III stress intensity factors are computed along the front of the penny crack. The maximum, minimum and average error results for different mesh sizes and initial crack size are tabulated in Table II.

Relative error on K_I	$a = 3.0$mm $h = 0.01$mm	$a = 0.4$mm $h = 0.01$mm	$a = 0.2$mm $h = 0.01$mm	$a = 0.1$mm $h = 0.01$mm
Max error (%)	7.569	6.1594	16.912	35.4873
Avg error (%)	1.9429	3.1807	6.6254	16.94
Min error (%)	0.0040957	0.074917	0.050829	0.22179

Table II. Error on the mode I stress intensity factor for the problem of a penny crack embedded in a finite cube for various crack radii and characteristic mesh sizes.

In Table II, h denotes the characteristic element length at the vicinity of the crack and a the penny crack radius. Note that for a ratio $a/h = 40$ (third column in Table II), the maximum error on the front is about 6%. This information is useful to help design optimized meshes for more complex problems. Note that even though no meshing of the crack is needed, it is still important to have sufficient mesh refinement in the vicinity of the crack in order to obtain accurate SIF values.

The penny crack described above is grown under fatigue governed by a Paris law with $C = 1$ and $m = 2$. The increment in crack length Δa is given as a function of material parameters C, m, the stress intensity factor range ΔK and the number of increment in loading cycles as $\Delta a = C\,(\Delta K)^m$. The initial crack configuration and velocity field on the front and the crack after five time steps are shown in Figure 23. Note that the velocity field on the front indicated by the arrows is almost uniform on the crack front and that the crack front remains circular



(a) (b)

Figure 21. Simulated crack paths:(a) Fully non-conforming mesh; and (b) Interfaces are explicitly meshed

with growth. Figure 22 shows the progression of the crack length with the number of fatigue cycles.

Remark 1: Note that although the stress intensity factors at each step are computed with a fair accuracy, the predicted crack growth curve drifts away from the analytical solution as time evolves. This phenomenon is all the more acute when the Paris exponent is high. This idea is summarized in Table III, which shows the error on the crack growth rate induced by an initial error on the stress intensity factor for three values of the Paris exponent. The Moving Least Squares Element Free Galerkin method (MLS-EFG) is particularly attractive to

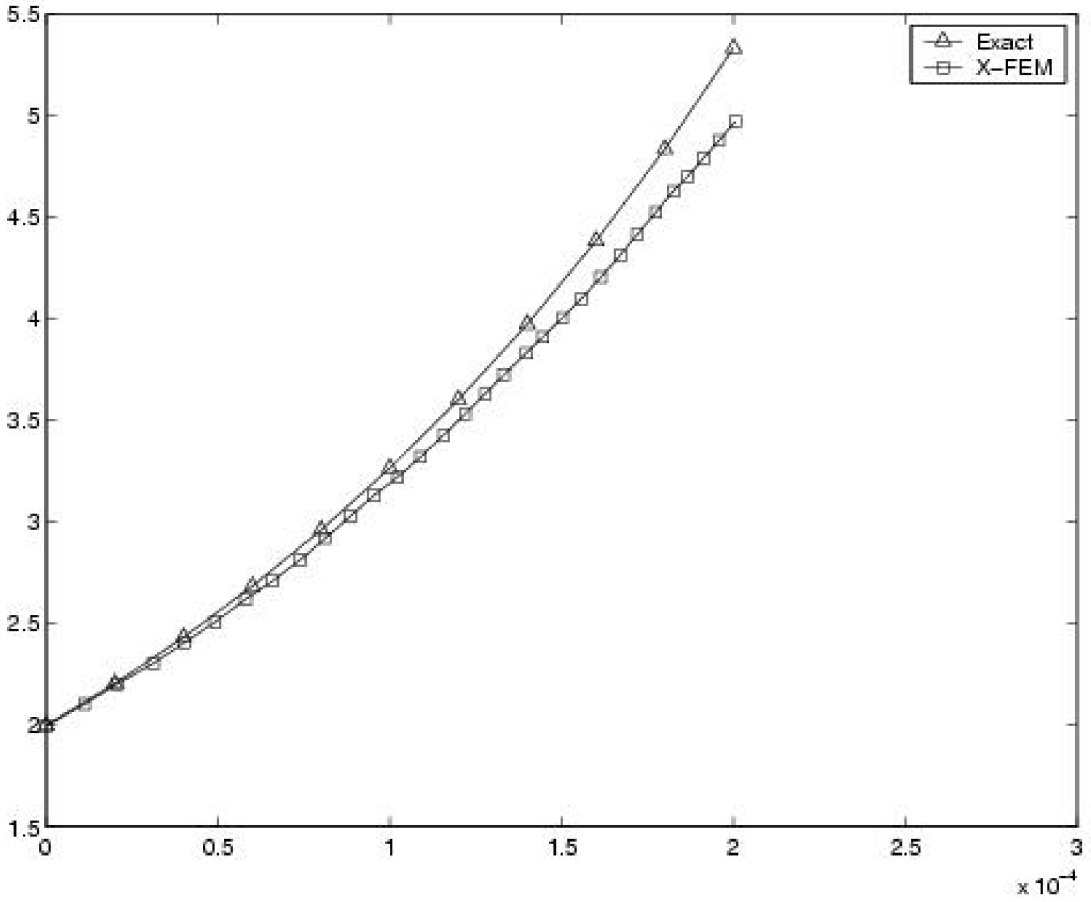


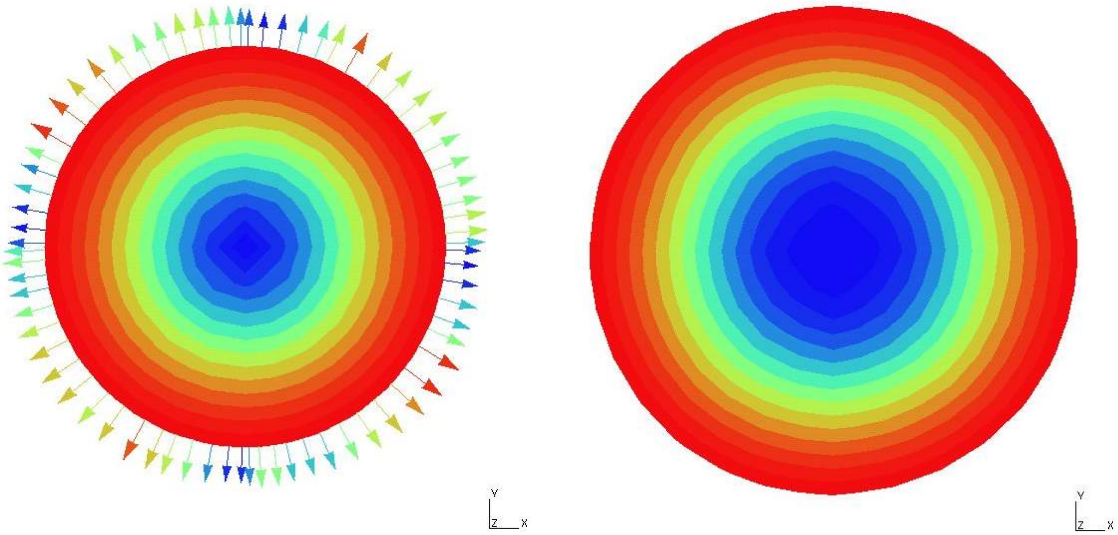
Figure 22. Crack length (mm) versus number of cycles for a penny crack in a finite cube

compute stress intensity factors. In Reference [58], for instance, it is shown that the accuracy of the EFG with extrinsic partition of unity enrichment is much higher than that of the XFEM presented here. This is certainly a by-product of the high degree of continuity of the MLS approximation, which allows reducing the maximum error on stress intensity factor along the front to only 0.2% for a 81,000 dof discretization.

Remark 2: A similar reasoning leads to the conclusion that, in order to achieve an accuracy of 1% on crack growth increments, the stress intensity factors need to be known with less than 0.2% error. As shown in [58], this level of accuracy can easily be achieved using an MLS-EFG method with extrinsic enrichment.

Paris exponent	Error on SIF	Error on crack growth rate
2	5%	10.25%
3	5%	15.76%
4.98	5%	27.5%

Table III. Effect of the Paris exponent on the error on the crack growth rate.



(a) Initial crack (b) Crack after 5 steps

Figure 23. Growth of a penny crack under uniaxial tension: initial configuration with crack growth field distribution on front and crack after five steps.

5.6. Industrial example, the Boeing 757 EE Access Door

To illustrate the applicability of the XFEM to industrial problems, we present some damage tolerance assessment results for a complex aerospace component –the *Boeing 757 EE Access Door*. The material of the door is *A356-T60*. The material constants of this aluminium grade are taken as $E = 72.4GPa$ and Poisson's ratio $\nu = 0.33$. Constant amplitude load cycles are assumed, and fatigue crack propagation is governed by the Paris law. Fatigue properties for the aluminium were given by the NASA program NASGRO [85]:

- $m = 4.98$
- $C = 2.25 \times 10^{-12}(\frac{\text{m}}{\text{cycles}})(\frac{1}{\text{MPa}\sqrt{\text{m}}})^{4.98}$

obtained with the super-element-XFEM method of References [70,86], FAA reports [87,88,89] and Ph.D. thesis [38]. The model of the door mixes Bathe and Dvorkin shell elements

with solid elements, linked using Multiple Point Constraint (MPC) elements. The door is subjected to a cyclic pressure loading of amplitude 9psi applied on the outer skin, to simulate compression/decompression cycles of the cabin. A superelement technique depicted in Figure 24 is used.

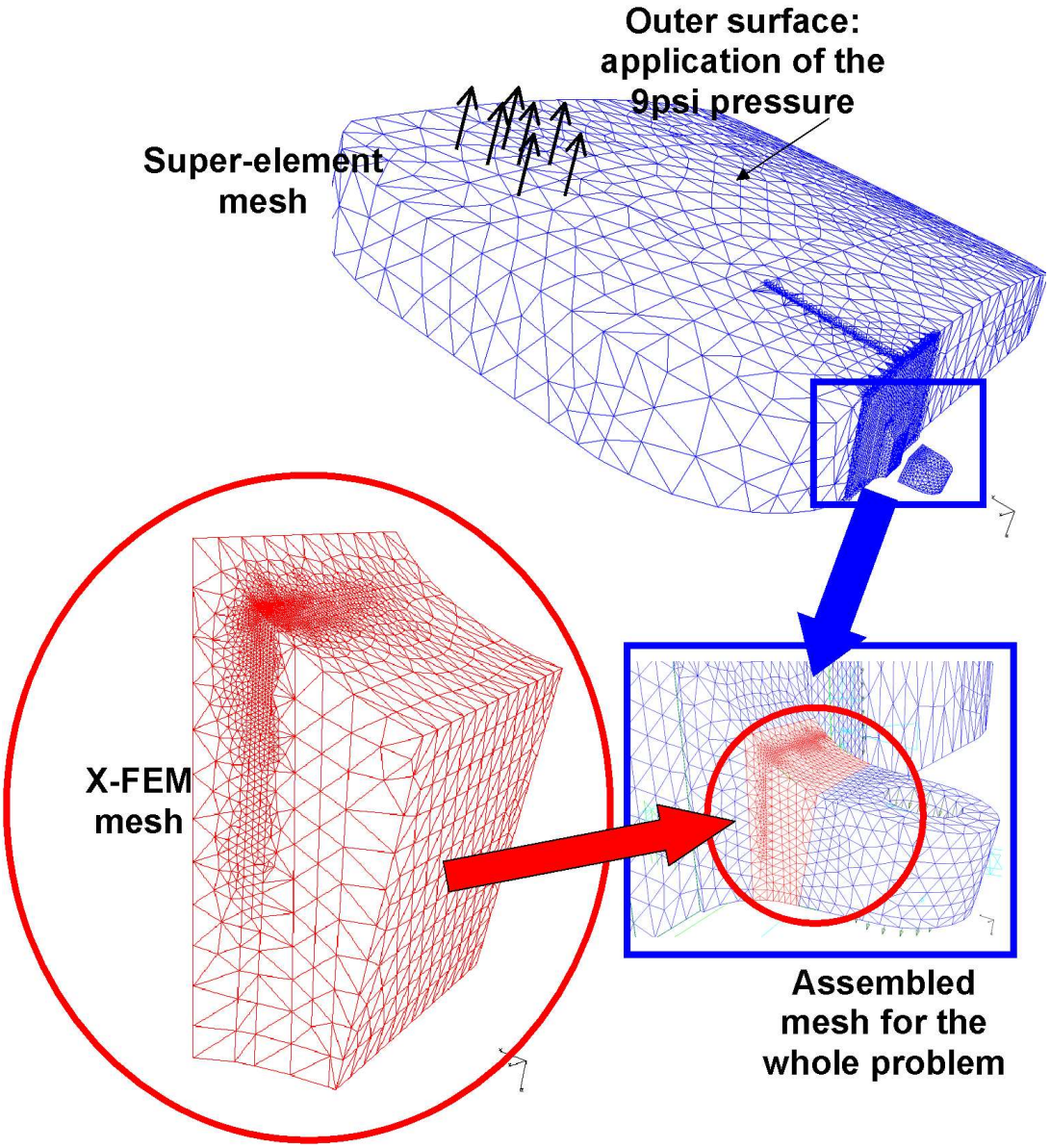
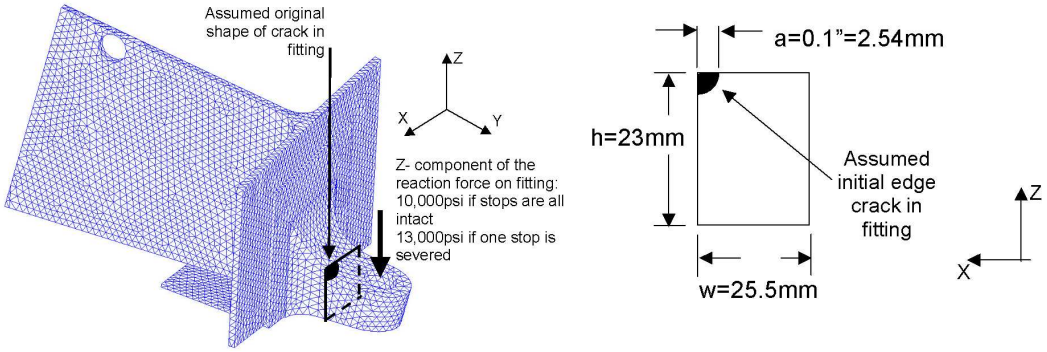
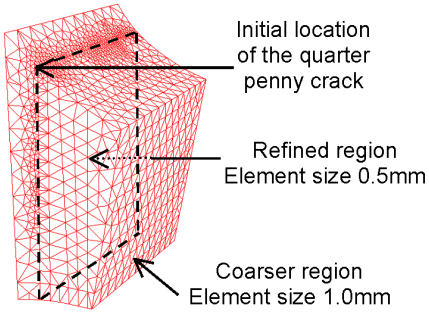


Figure 24. Superelement model and XFEM model

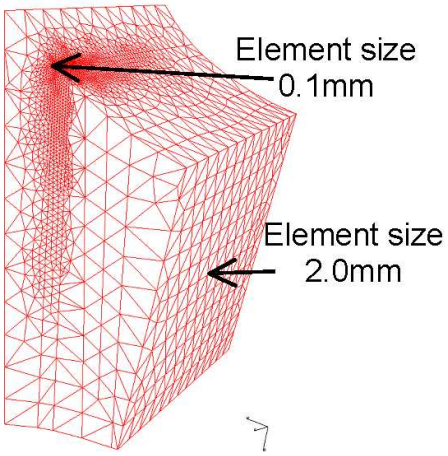
The advantage of the method proposed here is to permit multiple analyses to be carried out with minimum human intervention. The purpose of this paper is not to fully describe the damage tolerance assessment, as in Reference [86] nor the superelement-XFEM methodology of Reference [70].



(a) Continuum model and initial location of the crack



(b) XFEM model: refinement in the crack region



(c) XFEM model with 0.1mm elements in the crack region

Figure 25. XFEM models of the Boeing 757 EE Access door stop and location of the initial crack

Consequently, one single scenario of an edge crack of radius 2.54mm is used here to estimate the safe time intervals between two inspections The results summarized here are given in detail in Reference [70], to which the interested reader is referred for more details.

Remark 3: In the crack growth plots, the minimum, average and maximum distance from the crack front to the center of the initial flaw are given. Initially, for a circular flaw, all points

on the front are equidistant from the center of the flaw. If the SIFs is not constant on the front, some points on this front may move at different speed.

In this example, we assume that the stress intensity factor on the crack is constant and equal to the maximum mode I SIF on the whole front, which is a conservative design assumption. The corresponding crack length versus time curve is given in Figure 26. In Figure 27, crack growth curves for two time steps are depicted and show the insensitivity of the method to the choice of the time step used for the Paris' crack growth law.

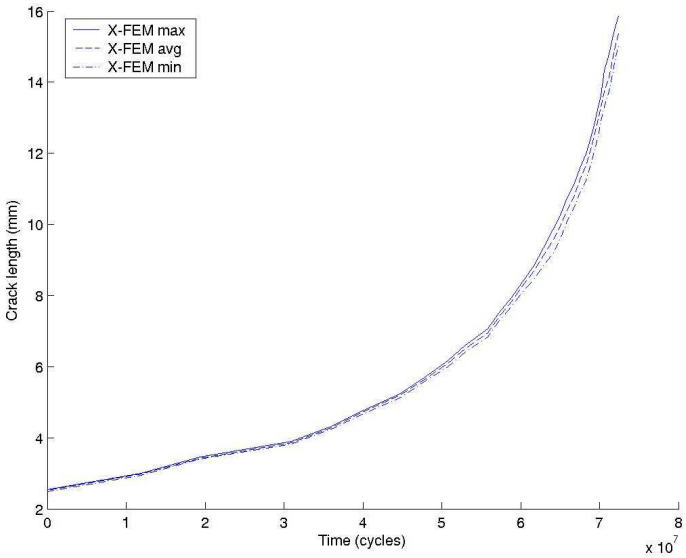
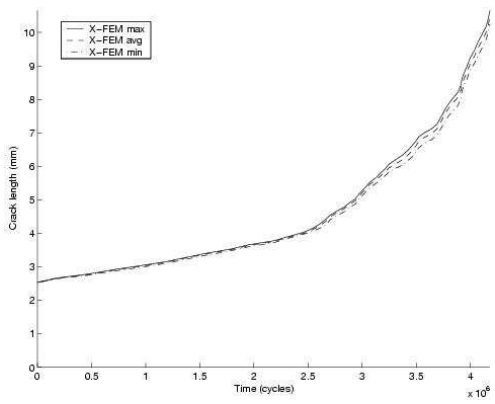


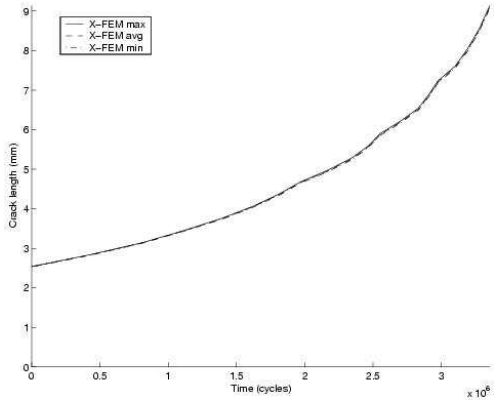
Figure 26. Crack length (mm) versus number of cycles for a corner edge crack in a stop with the constant mode I maximum SIF assumed on the whole front for growth.

6. Conclusions

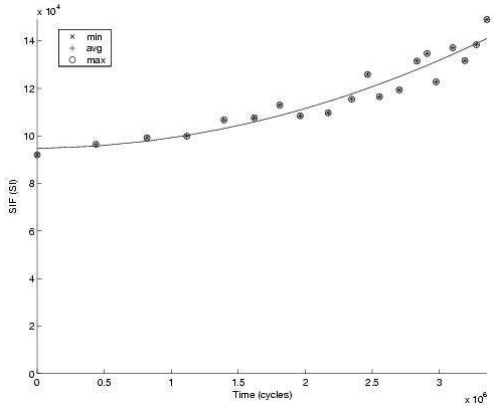
In this paper, an evolutive, versatile and easy-to-use C++ object-oriented –the C++ code has been compiled with Microsoft Visual Studio .NET2003– approach to the computer implementation of enriched finite element methods is presented. Object-oriented design allows easy code expansion. Hence, the incorporation of new enrichment functions, for instance, is straightforward. The code is coupled with an integrated mesh generator whose advantages are described Reference [78] where it is shown that problems with thousands of enrichment features, such as cracks, can be solved very efficiently, while a more naive approach makes this class of problems prohibitive. Basically, coupling a mesh generator to the extended finite element code permits performing mesh-geometry interaction, geometric enrichment, i.e. enrichment of fixed-area around the crack tip, local refinement, J-integral computations, for an optimal computational cost, by using tree structures, retaining the mesh generator information



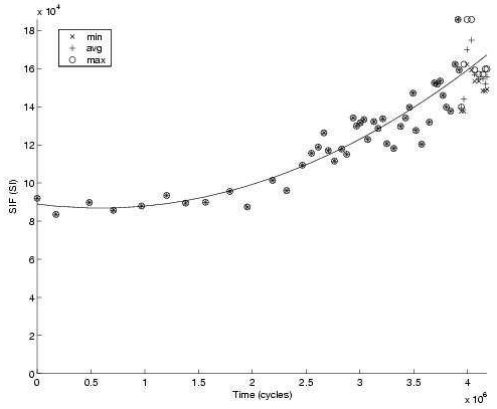
(a) Crack length (mm) versus time (cycles) with crack advances on the order of 2.5 elements per step



(b) Crack length (mm) versus time (cycles) with crack advances on the order of 1 element per step



(c) SIF ($\mathrm{mN/mm^2}\sqrt{\mathrm{mm}}$) with crack advances on the order of 2.5 elements per step



(d) SIF ($\mathrm{mN/mm^2}\sqrt{\mathrm{mm}}$) with crack advances on the order of 1 element per step

Figure 27. Severed stop, quarter corner edge crack growth for two time steps

and making it available to the solver.

The focus of the numerical examples are on computational fracture applications in isotropic homogeneous and heterogeneous media as well as industrial crack growth problems. The code also handles arbitrary material interfaces and voids, interface cracks as well as non-linear material laws and multiple fatigue crack propagation. Enrichment features can be treated using standard geometry, level sets and vector level sets. Quadrature rules are adaptive, and include sub-triangulation, and rectangular background grids. High order elements are supported, and the finite element space used for the standard degrees of freedom is independent from that

used for the enriched part of the approximation, allowing for a greater flexibility.

Issues pertaining to the modification of classical finite element classes as well as the implementation of new classes are addressed. Numerical examples in 2D and complex 3D situations show the accuracy and efficiency of the method and how enriched finite elements can be applied to industrial crack growth problems.

Excellent domain independence in the SIF computations are realized for straight cracks. A preliminary parameter study on the influence of the radius of the domain used for stress intensity computation is performed. A more complete study may be found in [68], where it is concluded that for fixed enrichment area, the interaction integral domains should *not* be smaller than the enrichment radius. However, no conclusion are drawn concerning the optimal value of the asymptotic enrichment radius and its relationship with the structure's characteristic dimensions, nor has a general methodology been devised to estimate the optimal value for the enrichment radius. This is subject to on-going work.

This paper points out the advantages of partition of unity enriched finite element methods over their meshfree counterparts. Enriched FEM being finite element based can exploit the large body of available finite element technology and pieces of software. This is reflected in the relatively small modifications of an available finite element package required for their implementation. The discontinuities being unmeshed, an open issue resides in devising an optimal technique to track the discontinuities in enriched numerical methods, especially if such methods need to be coupled with existing commercial software. A large step in this direction is detailed in the paper by Duflot et al. [80], which shows how the usual level set techniques may be improved to yield robustness and accuracy to the crack-tracking algorithm. In Reference [58], in the context of an adaptive moving least squares element free Galerkin method with extrinsic partition of unity enrichment, we present a possible explicit treatment of the crack geometry in quite complex settings including multiple cracks, initiation, growth and coalescence of arbitrary cracks in non-linear materials, statics and dynamics. Useful references for 3D crack growth in LEFM in the context of the EFG method are [51, 52, 53].

This work is intended to serve as a starting package for further developments of enriched finite element methods such as the XFEM or the GFEM. The proposed library can be obtained on request by contacting the corresponding author. We believe that it can serve as a very efficient base for further developments, especially for beginning graduate students.

ACKNOWLEDGEMENTS

The first author thanks the LSC at the EPFL, Lausanne for its support. The support for the first author's work at Northwestern University from the FAA through grant DTFA03- 98-F-IA025 Design and Quality Assurance of Premium Quality Aerospace Castings is gratefully acknowledged as is the support of Professors Brian Moran, James Conley and David Chopp. The first author wishes to thank Professor Ted Belytschko for his advice. The support of the EMMC program in Ho Chi Minh City University of Technology should be emphasized. Helpful comments of the late Richard Topp from the Boeing Company are also gratefully acknowledged.

The authors wish to thank deeply the reviewers from the journal for their very honest and thorough work in reviewing this work, which greatly contributed to the enhancement of this contribution.

APPENDIX

I. The class hierarchy

This appendix introduces the class hierarchy of the OpenXFEM++ library. Classes in **bold** font are new ones, while the classes in *italic* font are classes of FEMOBJ which are modified to include the X-FEM.

AuxiliaryFields
Dictionary
Dof
Domain
FEInterpolation
- **FEInterpolation2d**
  - **FEI2dQuadLin**
  - **FEI2dTriLin**

IntegrationRule
- **SplitGaussQuadrature**
- **StandardGaussQuadrature**

FEMComponent
- **CrackGrowthIncrementLaw**
  - **ParisLaw**
  - **FixedIncrement**
- **CrackGrowthDirectionLaw**
  - **MaxHoopStress**
  - **MaxEnergyReleaseRate**
- *Element*
  - QuadU
  - **TriU**
  - **Tri6**
- **EnrichmentFunction**
  - **DiscontinuousFunction**
  - **AsymptoticFunction**
    - **CrackAsymptotic**
      - **HomogElastCrackAsymp**
      - **BiMaterialElastCrackAsymp**
- **EnrichmentItem**
  - **CrackInterior**
  - **CrackTip**
  - **Hole**
  - **MaterialInterface**
- **GeometryDescription**

LevelSetDescription
VectorLevelSetDescription
StandardDescription
GeometryEntity
Circle
PiecewiseLinear
Vertex
Load
BodyLoad
DeadWeight
BoundaryCondition
InitialCondition
NodalLoad
LoadTimeFunction
ConstantFunction
PeakFunction
PiecewiseLinFunction
Material
ElasticMaterial
VonMisesMaterial
VonMisesMaterialH
NullMaterial
NLSolver
ConstantStiffness
ModNewtonRapson
NewtonRapson
Node
TimeIntegrationScheme
Newmark
Static
TimeStep
FileReader
FloatArray
Column
GaussPoint
IntArray
LHS
SkyLine
LinearSystem
List
MathUtil
Matrix
FloatMatrix
DiagonalMatrix
PolynomialMatrix
Pair

Polynomial
PolynomialXY

II. Typical data file of OpenXFEM++

This appendix presents the data file of the OpenXFEM++ package. For the complete explanation on sections of the data file, the interested reader can refer to [68].

```
TimeIntegrationScheme
1 class Static   *
**
Material 1
1 E 3.e7 n 0.25 *
**
Element 234
1 class T3U  mat 1 nodes 1 3 4     *
2 class T3U  mat 1 nodes 10 33 41  *
**
Node 1234
1 coord 2 0.0 0.0 nDofs 2 bcOnDof1 1  bcOnDof2 1   *
2 coord 2 2.0 0.0 nDofs 2 bcOnDof1 1  bcOnDof2 1   *
**
Load 2
1 class BoundaryCondition  loadTimeFunction  1  conditions 1  d  0.    *
2 class NodalLoad          loadTimeFunction  2  components 2 4000.  0. *
**
EnrichmentItem 3
1 class CrackInterior myTips 2 2 3 geometry 1
EnrichmentFunctions 1 1 enrichScheme 3 *
2 class CrackTip Type HomoElast Mat 1 geometry 2 EnrichmentFuncs 4 2 3 4 5
enrichScheme 1  domainIntFac 2.5 *
3 class CrackTip TypeHomoElast  Mat 1 geometry 2 EnrichmentFuncs 4 2 3 4 5
enrichScheme 1  domainIntFac 2.5 *
**
GeometryEntity 3
1 class PiecewiseLinear numOfVertices 2 vertices 2 3 geoDescription 1*
2 class Vertex coord 2 0.75 3.0   *
3 class Vertex coord 2 1.25 3.0   *
**
EnrichmentFunction 5
1 class DiscontinuousField       *
2 class HomoElastCrackAsymp1     *
3 class HomoElastCrackAsymp2     *
4 class HomoElastCrackAsymp3     *
5 class HomoElastCrackAsymp4     *
**

CrackGrowthDirectionLaw
1 class MaxHoopStress *
**
CrackGrowthIncrementLaw
1 class FixedIncrement delta 0.2 *
```

**

references

[1] I. Babuška and I. Melenk. Partition of unity method. *International Journal for Numerical Methods in Engineering*, 40(4):727–758, 1997.

[2] T. Belytschko and T. Black. Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45(5):601–620, 1999.

[3] N. Moës, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150, 1999.

[4] T. Strouboulis, I. Babuška, and K. Copps. The design and analysis of the generalized finite element method. *Computer Methods in Applied Mechanics and Engineering*, 181(1-3):43–69, 2000.

[5] T. Strouboulis, K. Copps, and I. Babuška. The generalized finite element method : An example of its implementation and illustration of its performance. *International Journal for Numerical Methods in Engineering*, 47(8):1401–1417, 2000.

[6] A. Hansbo and P. Hansbo. A finite element method for the simulation of strong and weak discontinuities in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 193(33-35):3523–3540, 2004.

[7] P.M.A. Areias and T. Belytschko. A comment on the article "A finite element method for simulation of strong and weak discontinuities in solid mechanics". *Computer Methods in Applied Mechanics and Engineering*, 195(9-12):1275–1276, 2006.

[8] J. Mergheim, E. Kuhl, and P. Steinmann. Towards the algorithmic treatment of 3D strong discontinuities. *Communications in Numerical Methods in Engineering*, 2006. submitted.

[9] J. Mergheim, E. Kuhl, and P. Steinmann. A finite element method for the computational modelling of cohesive cracks at large strains. *International Journal for Numerical Methods in Engineering*, 2006. submitted.

[10] J. Mergheim, E. Kuhl, and P. Steinmann. A finite element method for the computational modelling of cohesive cracks. *International Journal for Numerical Methods in Engineering*, 63(2):276–289, 2005.

[11] R. de Borst, J.J.C. Remmers, and A. Needleman. Mesh-independent discrete numerical representations of cohesive-zone models. *Engineering Fracture Mechanics*, 73(2):160–177, 2006.

[12] T. Belytschko, N. Moës, S. Usui, and C. Parimi. Arbitrary discontinuities in finite elements. *Internation Journal of Numerical Methods in Engineering*, 50(4):993–1013, 2001.

[13] N. Sukumar and J.-H. Prévost. Modeling quasi-static crack growth with the extended finite element method. part I: Computer implementation. *International Journal of Solids and Structures*, 40(26):7513–7537, 2003.

[14] G. Legrain, N. Moës, and E. Verron. Stress analysis around crack tips in finite strain problems using the extended finite element method. *International Journal for Numerical Methods in Engineering*, 63(2):290—314, 2005.

[15] P.M.A. Areias, J.M.A. Cesar de Sa, C.A. Conceicao Antonio, J.A.S.A.O. Carneiro, and V.M.P. Teixeira. Strong displacement discontinuity and lagrange multipliers in the analysis of finite displacement fracture problems. *Computational Mechanics*, 35(1):54–71, 2004.

[16] N. Sukumar, D. L. Chopp, and B. Moran. Extended finite element method and fast marching method for three-dimensional fatigue crack propagation. *Engineering Fracture Mechanics*, 70(1):29–48, 2003.

[17] N. Moës, A. Gravouil, and T. Belytschko. Non-planar 3D crack growth by the extended finite element and level sets. part I: Mechanical model. *International Journal for Numerical Methods in Engineering*, 53(11):2549–2568, 2002.

[18] A. Gravouil, N. Moës, and T. Belytschko. Non-planar 3D crack growth by the extended finite element and level sets. part II: level set update. *International Journal for Numerical Methods in Engineering*, 53(11):2569–2586, 2002.

[19] P.M.A. Areias and T. Belytschko. Analysis of three-dimensional crack initiation and propagation using the extended finite element method. *International Journal for Numerical Methods in Engineering*, 63(5):760–788, 2005.

[20] D. L. Chopp and N. Sukumar. Fatigue crack propagation of multiple coplanar cracks with the coupled extended finite element/fast marching method. *International Journal of Engineering Science*, 41(8):845–869, 2003.

[21] E. Budyn. *Multiple Crack Growth by the eXtended Finite Element Method*. PhD thesis, Northwestern University, June 2004.

[22] J. Dolbow, N. Moës, and T. Belytschko. Modeling fracture in Mindlin-Reissner plates with the eXtended finite element method. *International Journal of Solids and Structures*, 37(48-50):7161–7183, 2000.

[23] N. Moës and T. Belytschko. Extended finite element method for cohesive crack growth. *Engineering Fracture Mechanics*, 69(2):813–833, 2002.

[24] J. J. C. Remmers, R. de Borst, and A. Needleman. A cohesive segments method for the simulation of crack growth. *Computational Mechanics*, 31(1-2):69–77, 2003.

[25] G. Zi and T. Belytschko. New crack-tip elements for XFEM and applications to cohesive cracks. *International Journal for Numerical Methods in Engineering*, 57(15):2221–2240, 2003.

[26] R. de Borst. Numerical aspects of cohesive-zone models. *Engineering Fracture Mechanics*, 70(14):1743–1757, 2003.

[27] G. N. Wells and L. J. Sluys. A new method for modelling cohesive cracks using finite elements. *International Journal for Numerical Methods in Engineering*, 50(12):2667–2682, 2001.

[28] R. de Borst, M.A. Gutiérrez, G.N. Wells, J.J.C. Remmers, and H. Askes. Cohesive-zone models, higher-order continuum theories and reliability methods for computational failure analysis. *International Journal for Numerical Methods in Engineering*, 60(1):289–315, 2004.

[29] M.G.A. Tijssens, B.L.J. Sluys, and E. van der Giessen. Numerical simulation of quasi-brittle fracture and damaging cohesive surfaces. *European journal of mechanics. A, solids*, 19(5):761–779, 2000.

[30] N. Sukumar, Z. Y. Huang, J.-H. Prévost, and Z. Suo. Partition of unity enrichment for bimaterial interface cracks. *International Journal for Numerical Methods in Engineering*, 59(8):1075–1102, June 2003.

[31] N. Sukumar, D.L. Chopp, N. Moës, and T. Belytschko. Modeling holes and inclusions by level sets in the extended finite element method. *International Journal for Numerical Methods in Engineering*, 190(47):6183–6200, 2001.

[32] C. Daux, N. Moës, J. Dolbow, N. Sukumar, and T. Belytschko. Arbitrary branched and intersecting cracks with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 48(12):1741—1760, 1999.

[33] N. Sukumar, D. J. Srolovitz, T. J. Baker, and J.-H. Prévost. Brittle fracture in polycrystalline microstructures with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 56(14):2015–2037, 2003.

[34] P. Areias and T. Belytschko. Two-scale shear band evolution by local partition of unity. *International Journal for Numerical Methods in Engineering*, 66(5):878–910, 2006.

[35] J. Dolbow, N. Moës, and T. Belytschko. An extended finite element method for modeling crack growth with frictional contact. *Computer Methods in Applied Mechanics and Engineering*, 19:6825—6846, 2001.

[36] P.A. Guidault. *Une stratégie de calcul pour les structures fissurées: analyse locale-globale et approche multiéchelle pour la fissuration.* Phd, ENS-Cachan, 2005.

[37] J. Chessa, P. Smolinski, and T. Belytschko. The extended finite element method (X-FEM) for solidification problems. *International Journal of Numerical Methods in Engineering*, 53(7):1957–1977, 2002.

[38] S. Bordas. *Extended Finite Element and level set methods with applications to growth of cracks and biofilms*. PhD thesis, Northwestern University, December 2003.

[39] R. Duddu, S. Bordas, B. Moran, and D. Chopp. Extended finite elements and level sets for biofilm growth. *International Journal for Numerical Methods in Engineering*, 2006. submitted.

[40] J. E. Dolbow, E. Fried, and H. Ji. Chemically induced swelling of hydrogels. *Mechanics and Physics of Solids*, 52(1):51–84, 2003.

[41] A. Legay, J. Chessa, and T. Belytschko. An Eulerian-Lagrangian method for fluid-structure interaction based on level sets. *Computer Methods in Applied Mechanics and Engineering*, 195(17-18):2070–2087, 2006.

[42] P. Cavin, A. Gravouil, A. A. Lubrecht, and A. Combescure. Automatic energy conserving space-time refinement for linear dynamic structural problems. *International Journal for Numerical Methods in Engineering*, 64(3):304–321, 2005.

[43] J. Réthoré, A. Gravouil, and A. Combescure. A combined space-time extended finite element method. *International Journal for Numerical Methods in Engineering*, 64(2):260–284, 2005.

[44] J. Réthoré, A. Gravouil, and A. Combescure. An energy-conserving scheme for dynamic crack growth using the extended finite element method. *International Journal for Numerical Methods in Engineering*, 63(5):631–659, 2005.

[45] E. Walhorn, A. Kölke, B. Hübner, and D. Dinkler. Fluid-structure coupling within a monolithic model involving free surface flows. *Computers and Structures*, 83(25-26):2100–2111, 2005.

[46] A. Legay, H.W. Wang, and T. Belytschko. Strong and weak arbitrary discontinuities in spectral finite elements. *International Journal for Numerical Methods in Engineering*, 64(8):991–1008, 2005.

[47] H. Ji and J. Dolbow. On strategies for enforcing interfacial constraints and evaluating jump conditions with the extended finite element method. *International Journal for Numerical Methods in Engineering*, 61(14):2508–2535, 2004.

[48] Q.Z. Xiao and B.L. Karihaloo. Recent developments of the extended/generalized FEM and a comparison with the FEM. *In Development and Applications of Solid Mechanics , University of Science and Technology of China Press, Hefei, China*, ISBN 7-312-01842-4:303–324, 2005.

[49] B.L. Karihaloo and Q.Z. Xiao. Modelling of stationary and growing cracks in FE framework without remeshing: a state-of-the-art review. *Computers and Structures*, 81(3):119–129, 2003.

[50] T. Belytschko, Y. Y. Lu, and L. Gu. Element-free Galerkin methods. *International Journal for Numerical Methods in Engineering*, 37(2):229–256, 1994.

[51] M. Duflot and H. Nguyen-Dang. A meshless method with enriched weight functions for fatigue crack growth. *International Journal for Numerical Methods in Engineering*, 59(14):1945–1961, 2004.

[52] M. Duflot and H. Nguyen-Dang. A truly meshless Galerkin method based on a moving least squares quadrature. *Communications in Numerical Methods in Engineering*, 18(6):441–449, 2002.

[53] M. Duflot. A meshless method with enriched weight functions for three-dimensional crack propagation. *International Journal for Numerical Methods in Engineering*, 65(12):1970–2006, 2006.

[54] Fleming M., Y.A. Chu, B. Moran, and T. Belytschko. Enriched element-free galerkin methods for crack tip fields. *International Journal for Numerical Methods in Engineering*, 40(8):1483—1504, 1997.

[55] Y. Krongauz and T. Belytschko. EFG approximation with discontinuous derivatives. *International Journal for Numerical Methods in Engineering*, 41(7):1215–1233, 1998.

[56] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl. Meshless methods: An overview and recent developments. *Computer Methods in Applied Mechanics and Engineering*, 139(1-4):3–47, 1996.

[57] T-P. Fries and H. G. Matthies. Classification and overview of meshfree methods. Informatikbericht report, Institute of Scientific Computing, Technical University Braunschweig, Hans-Sommer-Strasse 65, D38106 Braunschweig, July 2004.

[58] T. Rabczuk, S. Bordas, and G. Zi. A three-dimensional meshfree method for continuous crack initiation, nucleation and propagation with crack path continuity. *Computational Mechanics*, 2006. in press.

[59] T. Rabczuk and J. Eibl. Simulation of high velocity concrete fragmentation using SPH/MLSPH. *International Journal for Numerical Methods in Engineering*, 56(10):1421–1444, 2003.

[60] T. Rabczuk and T. Belytschko. Adaptivity for structured meshfree particle methods in 2D and 3D. *International Journal for Numerical Methods in Engineering*, 63(11):1559–1582, 2005.

[61] T. Rabczuk and T. Belytschko. An adaptive continuum/discrete crack approach for meshfree particle methods. *Latin American Journal of Solids and Structures*, 1:141–166, 2003.

[62] T. Rabczuk and T. Belytschko. Cracking particles: A simplified meshfree method for arbitrary evolving cracks. *International Journal for Numerical Methods in Engineering*, 61(13):2316–2343, 2004.

[63] T. Rabczuk and T. Belytschko. A three dimensional large deformation meshfree method for arbitrary evolving cracks. *Computer Methods in Applied Mechanics and Engineering*, in progress.

[64] T. Rabczuk. *Numerische Untersuchungen zum Fragmentierungsverhalten von Beton*. PhD thesis, Institut fuer Massivbau und Baustofftechnologie, Universitaet Karlsruhe, 2002.

[65] T. Rabczuk and T. Belytschko. Application of particle methods to static fracture of reinforced concrete structures. *International Journal of Fracture*, 137(1-4):19–49, 2006.

[66] T. Belytschko, T. Rabczuk, E. Samaniego, and P.M.A. Areias. A simplified meshfree method for shear bands with cohesive surfaces. *International Journal for Numerical Methods in Engineering*, submitted.

[67] T. Rabczuk, P.M.A. Areias, and T. Belytschko. A meshfree thin shell for large deformation, finite strain and arbitrary evolving cracks. *International Journal for Numerical Methods in Engineering*, submitted.

[68] V. P. Nguyen. An object oriented approach to the XFEM with applications to fracture mechanics. Master's thesis, EMMC-Hochiminh University of Technology, Vietnam, November 2005. available on request.

[69] T. Zimmermann, Y. Dubois Pelerin, and P. Bomme. Object oriented finite element programming: I. governing principles. *Computer Methods in Applied Mechanics and Engineering*, 93(8):291–303, 1992.

[70] S. Bordas and B. Moran. Enriched finite element and level set method for damage tolerance assessment of complex structures. *Engineering Fracture Mechanics*, 73(9):1176–1201, 2006.

[71] F. Stazi, E. Budyn, J. Chessa, and T. Belytschko. An extended finite element method with higher-order elements for curved cracks. *Computational Mechanics*, 31(1-2):38–48, 2003.

[72] P. Laborde, J. Pommier, Y. Renard, and M. Salaun. High-order extended finite element method for cracked domains. *International Journal for Numerical Methods in Engineering*, 190(47):6183—6200, 2004.

[73] E. Béchet, H. Minnebo, N. Moës, and B. Burgardt. Improved implementation and robustness study of the X-FEM for stress analysis around cracks. *International Journal for Numerical Methods in Engineering*, 64(8):1033–1056, 2005.

[74] G. Ventura. On the elimination of quadrature subcells for discontinuous functions in the extended finite-element method (p n/a). *International Journal for Numerical Methods in Engineering*, 66(5):761–795,, 2006.

[75] S. Bordas and A. Legay. Enriched finite element short course: class notes. In *The extended finite element method, a new approach to numerical analysis in mechanics: course notes*. Organized by S. Bordas and A. Legay through the EPFL school of continuing education, Lausanne, Switzerland, December 7–9, 2005.

[76] Dimitri van Heesch. Doxygen, a documentation system for C++, C, Java, Objective-C, Python, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D, 1997–2006. http://www.stack.nl/ dimitri/doxygen/.

[77] O. Devillers, S. Meiser, and M. Teillaud. Fully dynamic Delaunay triangulation in logarithmic expected time per operation. *Computational Geometry: Theory and Applications*, 2(2):55–80, 1992.

[78] C. Dunant, S. Bordas, V.P. Nguyen, A. Guidoum, and Nguyen-Dang H. Architecture trade-offs of including a mesher in an enriched finite element program. *European journal of computational mechanics, special issue on partition of unity enrichment in finite elements*, 2006. in press.

[79] M. Stolarska, D. L. Chopp, N. Moës, and T. Belytschko. Modelling crack growth by level sets and the extended finite element method. *International Journal for Numerical Methods in Engineering*, 51(8):943–960, 2001.

[80] M. Duflot. A study of the representation of cracks with level sets. *International Journal for Numerical Methods in Engineering*, 2006. submitted.

[81] J. Chessa, H. Wang, and T. Belytschko. On the construction of blending elements for local partition of unity enriched finite elements. *International Journal of Numerical Methods in Engineering*, 57:1015–1038, 2003.

[82] Y. Sumi, C. Yang, and Z. Wang. Morphological aspects of fatigue crack propagation. Part II - effects of stress biaxiality and welding residual stresses. Technical report, Department of Naval Architecture and Ocean Engineering, Yokohama National University, Japan, 1995.

[83] P.O. Bouchard, F. Bay, and Y. Chastel. Numerical modeling of crack propogation: automatic remeshing and comparison of different criteria. *Computer Methods in Applied Mechanics and Engineering*, 192(35-36):3887—3908, 2003.

[84] M.G. Knight, L.C. Wrobel, J.L. Henshall, and L.A. De Lacerda. A study of interaction between a propagating crack and an uncoated/coated elastic inclusion using the BE technique. *International Journal of Fracture*, 114:47–61, 2002.

[85] NASA. Nasgro, 2003. NASA Johnson Space Center and Southwest Research Institute (SwRI), information available at http://www.nasgro.swri.org/.

[86] S. Bordas, J. Conley, B. Moran, J. Gray, and E. Nichols. Design paradigm for castings integrating non-destructive evaluation, casting and damage tolerance simulations. *Engineering with Computers*, 2006. in press.

[87] B. Moran, S. Bordas, and J. G. Conley. Static Strength Analysis of Aerospace Castings. *Design and Quality Assurance of Premium Quality Aerospace Castings, FAA contract DTFA03-98-F-IA025*, 2003. Northwestern University, Center for Quality Engineering and Failure Prevention, McCormick School of Engineering and Applied Science.

[88] B. Moran, S. Bordas, and J. G. Conley. Damage Tolerance Assessment of Complex Aerospace Structures. *Design and Quality Assurance of Premium Quality Aerospace Castings, FAA contract DTFA03-98-F-IA025*, 2003. Northwestern University, Center for Quality Engineering and Failure Prevention, McCormick School of Engineering and Applied Science.

[89] S. Bordas, J. G. Conley, and B. Moran. Integrated design approach of aerospace castings. *Design and Quality Assurance of Premium Quality Aerospace Castings, FAA contract DTFA03-98-F-IA025*, 2003. Northwestern University, Center for Quality Engineering and Failure Prevention, McCormick School of Engineering and Applied Science.