Minimum energy multiple crack propagation. Part III: XFEM computer implementation and applications

Danas Sutula[a,b], Pierre Kerfriden[b], Tonie van Dam[a], and Stéphane P.A. Bordas[c,a,*]

[a]Institute of Computational Engineering, University of Luxembourg, 6 Avenue de la Fonte, 4362 Esch-sur-Alzette, Luxembourg
[b]School of Engineering, Cardiff University, Queen's Building, The Parade, Cardiff, CF24 3AA, UK
[c]Visiting Professor, Institute of Research and Development, Duy Tan University, K7/25 Quang Trung, Danang, Vietnam
[*]stephane.bordas@alum.northwestern.edu

July 17, 2017

Contents

Abstract

The three-part paper deals with energy-minimal multiple crack propagation in a linear elastic solid under quasi-static conditions. The principle of minimum total energy, i.e. the sum of the potential and fracture energies, which stems directly from the Griffith's theory of cracks, is applied to the problem of arbitrary crack growth in 2D. The proposed formulation enables minimisation of the total energy of the mechanical system with respect to the crack extension directions and crack extension lengths to solve for the evolution of the mechanical system over time. The three parts focus, in turn, on (I) the theory of multiple crack growth including competing cracks, (II) the discrete solution by the extended finite element method using the minimum-energy formulation, and (III) the aspects of computer implementation within the *Matlab* programming language. The key contributions of Part-III of the three-part paper are as follows: (1) implementation of XFEM in *Matlab* with emphasis on the design of the code to enable fast and efficient computational times of fracture problems involving multiple cracks and arbitrary crack intersections, (2) verification of the minimum energy criterion and comparison with the maximum tension criterion via multiple benchmark studies, and (3) we propose a numerical improvement to the crack growth direction criterion that gives significant gains in accuracy and convergence rates of the fracture paths, especially on coarse meshes. Finally, the open-source *Matlab* code, documentation, benchmarks and other example cases are included as supplementary material.

1 Introduction

In the previous two parts of this three-part paper we focussed on the theory and the formulation of energy-minimal crack growth in the context of linear elastic fracture mechanics (LEFM). We described three solution strategies within a discrete framework for the prediction of the fracture paths. We recommended a gradient-descent based solution approach due to its better robustness in tackling unstable crack growth problems. In the actual implementation of the proposed solution approach we discovered that a large amount of thought and planning was required to achieve a robust computer implementation. Indeed, such an implementation is a *sine qua non* for the success of simulations involving arbitrary multiple crack growth with intersects. Here, in Part-III of our three-part paper, we describe the lessons learnt from building such an implementation.

We apply the proposed energy minimisation routines within the framework of the extended finite element method (XFEM) [5, 29, 4, 19, 18]. We chose XFEM because it offers certain advantages in modelling problems with non-smooth solutions; in particular, cracks can be modelled independently of the underlying finite element mesh.[1] In XFEM, the classic finite element approximation is *enriched* with additional functions that can capture the essential features of discontinuities, high gradients and singularities associated with cracks in LEFM. In turn, XFEM can offer accurate and optimally convergent numerical solution for the elastic field [25, 22, 23, 2, 1]. The reader is referred to Part-II of our three-part paper for details on the XFEM approximation we use.

Even though XFEM offers mesh independent modelling of cracks, the enrichment, which is introduced via the partition of unity method [27, 12], needs to be updated as cracks evolve over time. The XFEM implementation poses some practical challenges, especially concerning crack growth with intersections or when the crack extension solution needs to be determined iteratively, such as in pursuing the energy-minimal fracture solution. To this end, a robust enrichment tracking strategy needs to be devised to facilitate the efficient updating of the discretised mechanical system with respect to any topological changes in the enrichment. The current XFEM implementation relies heavily on available computational memory for storing various enrichment data. The idea is to strive for faster computations at the expense of a higher memory demand.

Nowadays, many XFEM implementations exist in *C++* [8, 24, 31] and *Fortran* [7, 36, 35]. Since these are compiled programming languages, fast computations are generally possible. An interpreted language, such as *Matlab*, is generally slower (despite using many pre-combined libraries); however, faster development times can be achieved, which can be beneficial particularly for research purposes. Some example open-source research codes written in *Matlab* include an (extended) isogeometric analysis code [32] (available *here*) and an (enriched) meshless method library [33] (available *here*).

Concerning simulations of multiple crack growth within the XFEM framework, it can be noted that only small topological changes take place with each time-step in the enrichment topology. Provided the enriched part of the global system of equations is updated only where needed and not reassembled from scratch (for simplicity's sake), the bottleneck in every time-step will tend to lie in the solution to the linear system of equations as opposed to updating of the discrete system or post-processing of the solution. Our way of speeding up the computational times is to avoid recomputations of quantities that tend to be reused regularly. We consider a fair compromise to store each enriched element's shape functions, shape function derivatives and quadrature weights at Gauss points at the expense of a higher memory demand. This, in turn, can speed up the solution post-processing and updating times since certain expensive recomputations can be avoided (especially those that can not be easily expressed by vectorisation [20, 3, 26]). For example, the post-processing stage of the solution normally involves computing domain integrals to evaluate the crack tip stress intensity factors or the energy release rates.

[1] There are, however, some discretisation constraints related to XFEM, which will be addressed later.

Having the enriched elements' shape functions at one's disposal the domain integrals can be evaluated readily. Moreover, updating the global system of equations after each time-step requires certain enriched elements' contributions to be subtracted from the global system of equations. Since the enriched elements' shape functions are pre-computed, the subtraction of the elemental equations can be carried out faster. The benchmark problems of multi-crack growth that we solve clearly show such a performance profile of the current XFEM implementation within *Matlab*. Since *Matlab* uses built-in pre-compiled solvers, which are fast and robust, the total computational time of a large multi-crack growth simulation can be comparable to that of a compiled language.

The aim of this paper is to show the implementation of XFEM in *Matlab* for the solution of arbitrary multi-crack growth problems with the application of the minimum energy crack growth criterion. The salient features of this implementation are as follows:

1. arbitrary number of enrichment layers over an element,
2. efficient updating of the system of equations,
3. crack growth with arbitrary intersections,
4. minimum energy crack growth solution,
5. internal pressure driven crack growth.

A significant advantage of updating the enrichment topology only where the fracture geometry changes is that the non-linear problem of determining the crack extensions by the minimum energy criterion can be solved computationally faster. On the other hand, the limitation of the current XFEM implementation is that crack growth is not entirely mesh independent because the crack extension length needs to be at least as big as the tip enrichment radius. Although this is a general limitation of enriched discretisation strategies, some heuristic treatments can be found in the literature, e.g. mapping the crack tip enrichment functions along curved cracks [16, 5, 37]. The reader is referred to [34] for a review of the partition of unity treatments of crack growth in 2D and 3D.

2 Aspects of implementation

2.1 Crack intersections

Generally, a Heaviside enrichment for a crack is introduced at a node if the nodal support is completely cut by the crack. Care needs to be taken to determine if a node is enriched in the vicinity of a crack junction; as two Heaviside enrichments will exist at some nodes, it is important that the resulting enriched shape functions are unique and that there are enough of them to describe the displacements in the vicinity of the crack junction.

A crack junction enrichment was first developed by [13, 4] and used in [10] to model multi-crack growth with intersections. The enrichment took the form of a superposition

of two Heaviside enrichments – one for the intersecting (*slave*) crack and another for the intersected (*master*) crack. However, the Heaviside for the *slave* crack had to be modified; the enrichment function assumed the value of the Heaviside of the *slave* crack if a queried point lay on the side of the intersection and the Heaviside value of the *master* crack if the queried point lay on the other side of the intersection. The first downside of such an enrichment construct is the need to explicitly track the crack *junction* elements and the modified Heaviside enrichment function for the *slave* crack. The second downside is the lack of robustness of the enrichment method if intersection take place very close to the tip of the *master* crack. Depending on which end of the *master* crack is intersected, it may not be possible to enrich a node whose support is cut by both the *slave* and the *master* crack since the discontinuity from the enrichment would then extend past the crack tip of the *master* crack. However, the full enrichment of the nodes of a *junction* element with the Heaviside for the *slave* crack is essential in order to adequately reproduce the discontinuous displacements in the vicinity of the crack junction.

The present approach to the junction enrichment is in most cases equivalent to the one used in [10]. However, the present strategy does not require a conceptually different Heaviside enrichment and, consequently, the explicit tracking of the crack *junction* elements. Instead, the standard Heaviside function is used to simulate the junction enrichment for the *slave* crack. The key idea is to consider an *imaginary* branch of the *slave* crack that is deflected along the *master* crack. The problem is then to determine along which direction this imaginary branch should be deflected so that the junction enrichment can be constructed optimally, i.e. all the nodes of the *junction* elements can be enriched. Thus, the Heaviside-enriched nodes for the *slave* crack are selected as the nodes whose support is cut by both the *real* and the *imaginary* branches of the *slave* crack. On the other hand, the Heaviside enrichment for the *master* crack is unchanged. Figure 1 shows a schematic diagram of the crack junction enrichment. The deflection of the *slave* crack is normally chosen to be along the direction that yields an absolute change in the angle relative to the crack increment direction of less than $\pi/2$. An exception to this rule is when the crack intersection is close to the tip of the *master* crack, in which case the deflection of the *slave* crack is forced away from the tip of the *master* crack. This way, the junction enrichment can be properly constructed, i.e. with all the nodes of the crack *junction* elements enriched. On the other hand, the elements that are cut only by the imaginary branch and that are the nodes of the *junction* elements are the Heaviside *blending* elements for the *slave* crack since only some (not all) of these elements' nodes can be enriched. For the present purposes of junction enrichment, an element can be regarded as a *junction* element if it is simultaneously cut by both the *real* and the *imaginary* branches of the *slave* crack such that all of the elements nodes will be enriched. Note that this can mean that the crack intersection point may lie outside some of the *junction* elements (depending on the geometry of the crack intersection).

Concerning initially crossing cracks (i.e. 'X'-type crack intersections), the superposition of two Heaviside enrichments (for each of the two cracks) is insufficient to kinematically decouple the four quadrants of the intersected region. One crack in the 'X'-type intersec-

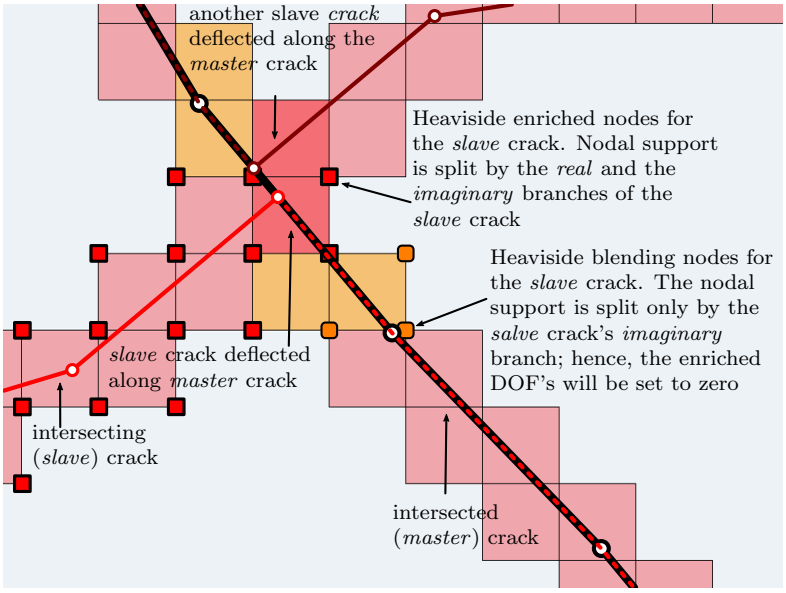


Figure 1: An example of two *minimum-distance* crack intersections. Consider the intersecting (*slave*) crack on the left. A junction enrichment is introduced by first deflecting the *slave* crack along the *master* crack. Subsequently, a Heaviside enrichment for the *slave* crack is introduced at the nodes whose support is cut by both the *real* (i.e. intersecting) and the *imaginary* (i.e. deflected) branches of the *slave* crack. The darker squares indicate the nodes whose supports are fully enriched whereas the lighter squares indicate the nodes whose supports are only partially enriched.

tion needs to be converted to two *slave* cracks, each of which needs to be deflected along the *master* crack as discussed previously. The Heaviside junction enrichment is then introduced for each of the two *slave* cracks. For propagating cracks, the following two criteria are applied for determining when a crack intersection needs to be created:

- **The minimum distance criterion:** intersection between crack A and crack B is created once the distance between A's tip and B's surface (including B's crack tip) becomes less than a prescribed tolerance (e.g. A's tip enrichment radius); crack A is then extended normal to B's surface.
- **The intersection criterion:** intersection between a propagating crack A and another crack B is created once it is detected that A's tip extension crosses B's surface thereby creating an 'X'-type intersection; crack A is then pulled back so that its tip lies on B's surface.

If an intersection point happens to lie very close to a crack node the intersection point is snapped to the crack vertex. If the crack intersection occurs very close to the tip of the *master* crack (e.g. the distance from the tip is smaller than the tip enrichment radius), the tip enrichment is annihilated since its benefit would otherwise be diminished.

2.2 Enrichment tracking

Different types of enriched elements are tracked separately in order to facilitate the enrichment updating process (e.g. extending or annihilating a certain enrichment) and to facilitate assembly routines routines that are specific to different element types, such as: tailoring element quadratures (e.g. for integrands that are: continuous/discontinuous, low/high order, singular/non-singular), and introducing the enrichment functions within an element in a particular way (e.g. using corrected branch enrichment functions [17]). Therefore, the different element types of each enrichment function are identified as:

- Heaviside enriched elements
  1. *standard* (all nodes are enriched) elements:
     `cHvStd_elm = cell(nCrack,1)`
  2. *blending* (not all nodes are enriched) elements:
     `cHvBln_elm = cell(nCrack,2)`
- Branch enriched elements
  1. standard *crack-tip* elements (partially cut by crack):
     `cBrStd_eTp = cell(nCrack,2)`
  2. standard *crack-split* elements (fully cut by crack):
     `cBrStd_eSp = cell(nCrack,2)`
  3. standard *non-split* (full) elements:
     `cBrBln_eFl = cell(nCrack,2)`
  4. blending crack-split elements:
     `cBrBln_eSp = cell(nCrack,2)`
  5. blending non-split (full) elements:
     `cBrBln_eFl = cell(nCrack,2)`

The element storage structure that we use is known as a *cell* in *Matlab*. We use the cell to store a vector (a 1D array) of enriched elements of a particular type. For example, the vector of *standard* Heaviside elements of crack `i_crk` is stored in `cHvStd_elm{i_crk}` whereas the blending branch split elements of crack `i_crk` and crack tip `i_tip` are stored in `cBrBln_eSp{i_crk,i_tip}`. In general, we refer to an element as an *enriched* element if any of the element's nodes is enriched. An enriched *standard* element is an element that has all of its nodes enriched, where as an (enriched) *blending* element has only some (not all) of its nodes enriched. In the proposed book-keeping of the different enriched element types there is no need to explicitly track the Heaviside crack *junction* elements; the crack junction enrichment is resolved naturally by considering that the *slave* crack has an *imaginary* branch that is deflected along the *master* crack on intersection.

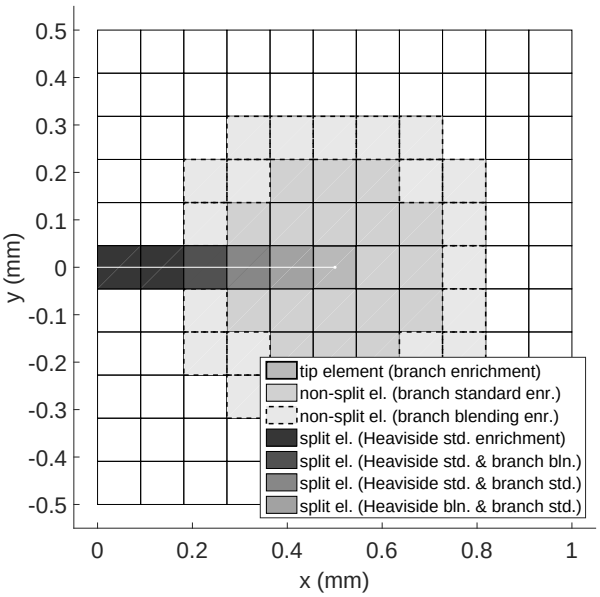


Figure 2: Different types of enriched elements. The disk-like patch of branch-enriched elements is centred at the crack tip. The branch enrichment is introduced at all the nodes of these elements. As in the corrected XFEM approximation [17], the branch enrichment is weighted by a ramp function that smoothly decreases over the last ring branch-enriched elements. The ramp equals to unity in the interior of the branch-enriched domain and zero on its boundary. Refer to Part-II for details on the XFEM approximation.

Of course, some elements will contain multiple overlapping enrichments. For instance, the Heaviside and the branch enrichments generally appear along some portion of the crack tip segment. Also, when two crack tips are in a sufficiently close proximity it is possible for an element to be simultaneously enriched with two sets of branch functions. Finally, when two cracks intersect, the intersected elements (i.e. the crack junction elements) will need to be enriched with two Heaviside functions, e.g. refer to Figure 1. In general, an allowance must be made for an element to be enriched an arbitrary number of times. Hence, it is necessary to track each enriched element's enrichment functions and the generalised enriched nodes (for the element's enriched shape functions). For this purpose, the cell variable `cLEnDt = cell(nElemn,1)` (`c`ell e`L`ement `EN`richment `Dt`a) is introduced. The enrichment information of a particular enriched finite element `i_elm` is tracked by the matrix `cLEnDt{i_elm} = [3-by-n]` where the number of columns in the matrix (`n`) corresponds to the number of enrichments in the finite element and where each column of the matrix lists the following information about the enrichment:

1. `cLEnDt{i_elm}[1,i_enr]` the crack identity of enrichment layer `i_enr`
2. `cLEnDt{i_elm}[2,i_enr]` the enrichment function identity
3. `cLEnDt{i_elm}[3,i_enr]` the preferred quadrature scheme identity

Another cell structure `cLNodE = cell(nElemn,1)` (eLements' `Nod`es `E`nriched) is used to store the generalised enriched nodes of the enriched shape functions of each finite element. An element of the cell is left *empty* if the finite element is not enriched. The enriched nodes are permanently tied to the enriched shape functions. Thus, when some enriched shape functions (and their enriched nodes) are annihilated during the updating of the discrete system, the remaining enriched nodes are unaffected. Also, the generalised enriched nodes are not recycled, i.e. new enriched shape functions are always tied to new enriched nodes. This is done for practical reasons in order to avoid having to globally update the cell `cLNodE`, which would be cumbersome in *Matlab*. The purpose of the enriched nodes is to serve as indices into the columns of the generalised nodal degrees of freedom matrix `mNDofE = [2-by-nNdEnr_max]`, where loosely speaking `nNdEnr_max` = max(`cLNodE{1:nElemn}`) is the largest enriched node number. The generalised DOFs (unlike the enriched nodes) have to be updated in response to the annihilation of any enriched shape functions (and their enriched nodes). This is because the enriched DOFs are indices into the columns/rows of the stiffness matrix (and into the elements of the force vector) and, as such, the DOFs must be numbered sequentially without gaps. Consequently, the enriched DOFs are recycled every time there is an update in the discrete system. Note that recycling old DOFs while introducing new enriched nodes means that the matrix `mNDofE` tends to become sparser with increasing number of time steps. However, the updating process of the matrix `mNDofE` is relatively simpler: the columns of `mNDofE` corresponding to the annihilated enriched nodes are zeroed out, `mNDofE` is enlarged by the number of columns equal to the number of new enriched nodes, the DOFs corresponding to all current enriched nodes are numbered sequentially from `nNdStd*2+1` to `nNdEnr*2`, where the factor 2 is the number of DOFs per node (which is 2 in 2D). In particular, if `vNdEnr_del` is the vector of deleted enriched nodes and `vNdEnr` is the vector of current enriched nodes (where the vector is ordered sequentially with `vNdEnr(end) = nNdEnr_max`), then updating `mNDofE` is basically as follows:

1. `mNDofE(:,vNdEnr_del) = 0;`
2. `mNDofE(1,vNdEnr) = [(nNdStd*2+1):2:(nNdEnr*2-1)];`
3. `mNDofE(2,vNdEnr) = [(nNdStd*2+2):2:(nNdEnr*2)];`

The computational times can be further improved in the updating of the system of equations and in the post-processing of the numerical solution by pre-computing the shape functions at quadrature points of the enriched finite elements. To this end, the following cells are used: (suppose `i_elm` is the enriched finite element, `nGauss` is the number of quadrature points in the element, `nLNodS` is the number of nodes in the element, and `nLNodE` is the number of generalised enriched nodes in the element.)

1. standard shape functions:
   `cGsEnr_omgShS = cell(nElemn,1)`, where
   `cGsEnr_omgShS{i_elm} = [nGauss-by-nLNodS]`
2. enriched shape functions:

`cGsEnr_omgShE = cell(nElemn,1)`, where
`cGsEnr_omgShE{i_elm} = [nGauss-by-nLNodE]`

3. derivatives of standard shape functions:
`cGsEnr_omgDvS = cell(nElemn,1)`, where
`cGsEnr_omgDvS{i_elm} = [2*nGauss-by-nLNodS]`

4. derivatives of enriched shape functions:
`cGsEnr_omgDvE = cell(nElemn,1)`, where
`cGsEnr_omgDvE{i_elm} = [2*nGauss-by-nLNodE]`

5. weights of quadrature points:
`cGsEnr_omgWgt = cell(nElemn,1)`, where
`cGsEnr_omgWgt{i_elm} = [nGauss-by-1]`

The shape functions of element `i_elm` are evaluated based on the quadrature-governing enrichment. The identities of the preferred quadrature rules for each of the occurring enrichments inside the finite element `i_elm` are stored in vector `cLEnDt{i_elm}[3,:]`. Among the preferred quadrature rules, which are defined for all enriched element types, the critical rule is selected as the one that will result in no (or, at worst, minimal) under-integration of the XFEM element stiffness matrix and force vector.

2.3 Numerical integration

Different types of elements of the enrichment functions (refer to Figure 2) require specialised quadrature schemes in order to perform the integration of the elemental equations adequately and efficiently [6, 38, 30] For example, integration over sub-cells should be used if an element is cut by a crack since there will be a discontinuity in the integrands across the crack interface. If the element contains a crack tip, an integration scheme that is better suited for the singular integrands should be used [25, 11], such as: *polar* [25], *almost-polar* [14] or the *parabolic* [28] integration scheme. Below is a summery of the quadrature schemes used in the current implementation for triangular and quadrilateral finite elements: (note that the schemes are enumerated in the order of precedence)

- Linear triangle (**T3**)
    1. Crack tip element with branch enrichment: use polar integration with the number of points per sub-cell in the radial and angular directions `nGsBrn_pol=[13,7]`
    2. Completely split element with branch enrichment: use a standard quadrature rule with the number of points per integration sub-cell `nGsBrn_sub=33`
    3. Completely split element with Heaviside enrichment: use a standard quadrature rule with the number of points per integration sub-cell `nGsHvi_sub=1`
    4. Non-split element with branch enrichment: use a standard quadrature rule with the number of points per element `nGsBrn_omg=33`

- Bi-linear quadrilateral (**Q4**)
    1. Crack tip element with branch enrichment: use polar integration with the number of points per sub-cell in the radial and angular directions `nGsBrn_pol=[13,7]`
    2. Completely split element with branch enrichment: use a standardquadrature rule with the number of points per integration sub-cell `nGsBrn_sub=33`
    3. Completely split element with Heaviside enrichment: use a standard quadrature rule with the number of points per integration sub-cell `nGsHvi_sub=3`
    4. Non-split element with branch enrichment: use a standard quadrature rule with the number of points per element `nGsBrn_omg=36`

Concerning the elements with branch enrichment, the above quadrature schemes showed good accuracy. Increasing the quadrature order did not notably change the strain energy for the discretisations used in the numerical benchmarks. Note that we apply the same quadrature scheme to both the *standard* and the *blending* branch-enriched elements. Concerning the elements with multiple overlapping enrichments, a single quadrature scheme is applied for computing all the enriched parts of the elemental equations. The enrichment tracking variable `cLEnDt` lists the preferred quadrature rules for each enrichment inside each element. Specifically, the vector `cLEnDt{i_elm}[3,:]` contains the preferred quadrature schemes for the element `i_elm`. When there are multiple quadrature schemes for an element with multiple layers of enrichment, the critical scheme is selected based on the most demanding enrichment function (see the enumerations above).

For every discontinuous enrichment the crack intersection points with an element need to be known in order to prepare the element for integration over sub-cells. Since, in general, an element can be intersected multiple times, all crack intersections with each element need to be found before the integration sub-cells can be computed. We use a cell variable `cXsElm = cell(nElemn,1)` to track the intersection points between elements and cracks. Recall that an intersection point is generalised to be any of these instances:

- a point on a boundary of an element that is crossed by a crack
- a point inside an element that is a crack vertex
- a point inside an element that is a crack tip

Note that a crack intersection point inside an element does not need to tracked explicitly because, in the current implementation, a *slave* (i.e. the intersecting) crack is deflected along a *master* (i.e. intersected) crack such that the point of deflection is actually a crack vertex point inside an element (i.e. the second bullet-point in the foregoing list).

Once all the intersection points between elements and cracks have been determined, the integration sub-cells can be obtained based on a Delaunay triangulation. An important requirement of a triangulation is to respect crack boundaries. *Matlab* (2015b) offers a built-in function that can perform a triangulation with respect to a set of edge constraints. These can be specified in terms of the crack segments themselves.

We store the quadrature weights of enriched elements in a cell variable `cGsEnr_omgWgt`. The weights are actually scaled by a factor that is equal to the ratio of the Jacobian at a quadrature point in the sub-cell to that in the element. The advantage is that the numerical integration only involves multiplying the (adjusted) quadrature weight by the determinant of the Jacobian of the element to obtained the discrete volume differential. Consequently, the integrals can be evaluated using the same algorithm for all types of enriched elements or quadrature schemes, which is very similar to the classic FEM.

3 Assembly of equations

3.1 Initiation of the discrete system

The elements to be enriched with the crack tip branch functions are relatively simple to identify; as the current implementation assumes the enrichment domain to be a disk of a fixed radius, the elements that have any of their nodes within this disk are enriched at all their nodes. The elements that lie on the periphery of the enrichment domain are classified as the *blending* elements (refer to Figure 2). The *blending* elements are different from the other (*standard*) branch-enriched elements in that the enrichment function is weighted by a ramp function [17]. The ramp function is a smooth function that is equal to unity at the nodes within the disk and zero outside the disk. Refer to Part-II Appendix A.1 for details on the construction of this XFEM approximation.

The robust selection of elements for Heaviside enrichment can be more challenging, especially in the presence of crack intersections. In general, a node is enriched with the Heaviside function if its support is completely split by a crack. On the contrary, if the nodal support covers the crack tip then the nodal support is not completely split by the crack; hence, Heaviside enrichment at that node should not be introduced. When two cracks intersect, the so-called *slave* (i.e. interesting) crack is deflected along the *master* (i.e. intersected) crack. The deflected branch is called the *imaginary* branch and the remainder of the crack is called the *real* branch. The *imaginary* branch is a fictitious part of the *slave* crack that serves to smoothly blend the Heaviside enrichment of the *slave* crack onto the *master* crack. A node is enriched with the Heaviside function of the *slave* crack if the nodal support is completely split by the *real* branch or by both the *real* and the *imaginary* branches of the *slave* crack. On the contrary, a node is not enriched if its support is split only by the *imaginary* branch or if the support of the node covers a crack tip.[2] Finally, the Heaviside enrichment for the *master* crack is unchanged.

Once the elements to be enriched with a particular enrichment are found, the generalised enriched nodes of these elements need to be determined. The first step is to reduce the standard nodal connectivity matrix of the set of elements to be enriched to a reference nodal connectivity matrix where the nodes are renumbered sequentially (without gaps)

[2] In practice, an element is enriched at all its nodes but the unwanted enriched DOFs are set to zero.

starting from one. The second step is to offset the nodes in the reference nodal connectivity matrix by the current largest enriched node number in the discrete system, which we call `nNdEnr_max`. This yields the enriched nodal connectivity matrix `mLNodE` for the patch of enriched elements. Finally, the vector of the generalised enriched nodes `mLNodE[i_elm,:]` for each element `i_elm` in the enriched patch needs to be appended to the element's global vector of all generalised enriched nodes `cLNodE{i_elm}`. This routine is repeated for any other enrichment functions over the same patch of elements, such as in the case of branch enrichment where four enrichment functions are used.

Apart from tracking the generalised enriched nodes of the enriched elements, the generalised enriched DOFs also need to be tracked. For this we use a matrix `mNDofE = [2-by-nNdEnr_max]` whose rows correspond to the DOFs in the two spatial dimensions. The nodal DOFs can be retrieved simply by using the generalised enriched nodes as column indices. In our implementation, the matrix `mNDofE` tends to become more sparse with increasing number of time-steps. This is because we do not recycle any enriched nodes once they are deleted during the enrichment updating stage; instead, when a new enrichment is introduced, the numbering of the new generalised enriched nodes is continued from the maximum enriched node number `mNdEnr_max`. This means there will be gaps in the numbering of the enriched nodes and that the nodes will need to be remapped to their corresponding DOFs. This is relatively easy to do in comparison to renumbering the generalised enriched nodes in the cell `cLNodE` for each enriched element.

The enrichment information of each element is stored in the cell variable `cLEnDt`. Every time a new layer of enrichment is introduced, the enriched element's enrichment tracking matrix `cLEnDt{i_elm}` is updated by appending a column vector that contains the following information about the new enrichment: (1) the crack identity, (2) the enrichment function identity, and (3) the preferred quadrature scheme identity.

If an enriched element is cut by a crack, each intersection point between the crack and the element needs to be stored. A cell variable `cXsElm` keeps track of the intersection points for each element. For an element `i_elm` the matrix `cXsElm{i_elm}` is updated for each new intersection point by appending the intersection point coordinates to the matrix. Recall that a valid intersection point is generalised to be: (1) an intersection point on a boundary of an element, (2) a crack vertex inside an element, or (3) a crack tip inside an element. In the end, the matrix `cXsElm{i_elm}` will be used to generate a discontinuity-conforming Delaunay triangulation for numerical integration purposes.

The preferred integration scheme for each enrichment occurring in an element is specified by an integer value. The integer value (also called the integration identity) is associated to the required order of quadrature and to either a continuous or a discontinuous type of integration. Once all enrichments over an element are known, the governing quadrature scheme is selected based on the precedence of the schemes (refer to Section 2.3).

After the governing numerical integration scheme is identified, the element's quadrature points and weights need to be determined. To apply the same algorithm for the numerical integration to elements that are either subdivided into sub-cells or not, all quadrature

points and weights need to be computed with respect to the parent coordinates of the element (not the sub-cell). Thus, if an element requires integration over sub-cells, the quadrature points and weights will need to be mapped from the parent coordinates of each sub-cell to the parent coordinates of the element. Concerning the quadrature point weights, the weights are computed by scaling each weight in the sub-cell by the ratio of the determinant of the Jacobian of the sub-cell to that of the element. The scaled weights are then stored in the cell variable `cGsEnr_omgWgt` for the enriched element.

The next step is to evaluate each enriched element's standard shape functions and shape function derivatives at the quadrate points and store them in cell variables `cGsStd_omgShS` and `cGsStd_omgDvS` respectively. Subsequently, the enriched shapes can be constructed by computing the products of the standard shapes with the value of the enrichment function for each quadrature point. This is repeated for any number of enrichment functions that occur inside an element. The assembled enriched shapes are stored in the cell variables `cGsEnr_omgShS` and `cGsEnr_omgDvS`. It is important to assemble the enriched shape functions in the order specified by the element's enrichment tracking matrix `cLEnDt{i_elm}`; specifically, the information provided by `cLEnDt{i_elm}[1:2,:]` gives a matrix whose column vectors denote the crack and the enrichment identities.

The final step is to assembly the stiffness matrix and the generalised force vector. Concerning the enriched elements, the assembly is similar to the classic FEM because each enriched element is accompanied by the quadrature point weights and all the shape functions and shape function derivatives pre-evaluated at the quadrature points.

3.2 Updating the discrete system

When a crack is extended, the previous tip enrichment needs to be annihilated. The procedure of clearing the former branch enrichment involves the following steps:

1. Deleting the branch-enriched elements previously stored (refer to Section 2.2).
2. Deleting the subsets of the generalised enriched nodes and the enriched shape functions of each element belonging to the former branch enrichment.
3. Deleting the rows and columns of the global equation system corresponding to the enrichment DOFs at the former branch-enriched nodes.
4. Renumbering all enrichment DOFs to match the remaining enriched nodes.

After the former branch enrichment is annihilated, the branch enrichment needs to be reintroduced considering the new crack tip configuration. This is essentially a repetition of the steps described in the previous subsection. Updating the Heaviside enrichment, on the other hand, is different since the new enrichment needs to be consistent with the existing Heaviside enrichment. More specifically, the Heaviside enrichment for a crack extension needs to be connected to the existing Heaviside enrichment along the remainder

of the crack. This is done by assigning the same generalised enriched nodes at the interface between the existing and the new Heaviside-enriched patches of elements.

If a crack tip extension cuts an element that is already cut by the same crack, the cut element's stiffness matrix and force vector contributions to the global system of equations need to be recomputed since the Heaviside enrichment is effectively different now. This can be done by first subtracting the enriched parts of the element's stiffness matrix and force vector from global system of equations; then, considering the updated crack geometry, recomputing the quadrature points and weights (based on the new integration sub-cells), recomputing the standard and the enriched shape functions, recomputing the enriched parts of element's stiffness matrix and force vector and, finally, adding them to the global system of equations. More generally, when a new enrichment needs to be introduced to an element that is already enriched, we adopt the simplest approach which is to recompute the entire enrichment from scratch for that element. Therefore, our procedure of enriching an already enriched element involves the following steps:

1. Subtracting the element's enriched parts of the stiffness matrix and and force vector from the global system of equations.
2. Determining (if necessary) the enriched nodes for the new/updated enrichment and appending them to the element's vector of all enriched nodes.
3. Determining the appropriate integration strategy for the element and (if necessary) recomputing the quadrature points and weights.
4. Re-evaluating the standard shape functions at the quadrature points and then recomputing the enriched shape functions for each of the enrichment functions.
5. Computing the new enriched parts of the element's stiffness matrix and force vector and adding them to the global system of equations.

Even though it would be more efficient to compute only what is required, specifically: the new enriched parts of the stiffness matrix and force vector due to the new enrichment, the entire computational routine is not so trivial. More often than not, introducing a new enrichment changes the preferred quadrature scheme; hence, it is usually necessary to recompute the quadrate points and weights, re-evaluate the standard shape functions and then all of the enriched shape functions. Both the standard and the enriched shape functions need to be recomputed because the interactions between the standard shape functions, the enriched shape functions of the existing enrichments, and the enriched shape functions of the new enrichment appear in the enriched parts of the element's stiffness matrix. Resolving these interactions can be cumbersome in the general case. Considering that a lot of computational work is necessary in updating the enriched parts of the stiffness matrix anyway, we can avoid the last bit of cumbersomeness in computing the interactions between the element's shape functions in the element's stiffness matrix by simply recomputing the entire enriched part of the element's stiffness matrix from scratch using *Matlab*'s efficient vectorisation for matrix multiplications [20, 3, 26].

4 Numerical benchmarks

The numerical benchmarks have two aims: to verify our implementation of the enrichment updating strategy, which we call the *efficient* implementation, and to assess its effectiveness in terms of the computational speed-up relative to reassembling the entire enrichment from scratch, which we call the *basic* implementation.

The so-called *efficient* approach updates the enrichment and the enriched part of the stiffness matrix only where there are changes in the enrichment topology, such as due to the growth and intersection of cracks; furthermore, it does so consistently with all existing enrichments. The so-called *basic* approach updates the enrichment by reassembling all enrichments and the enriched part of the stiffness matrix from scratch, as in the initial assembly (refer to Section 3.1). The *basic* approach is attractive mainly because achieving a robust implementation is much simpler in comparison to the *efficient* approach. However, the *basic* implementation can lead to a major bottleneck in the computational time when simulating multi-crack growth. Therefore, the *efficient* implementation is particularity desirable for multi-crack growth as it can offer a significant speed-up in the overall computational time even on moderate size meshes.

The benchmark cases assume a rectangular 10×1 plate with a roughly uniform distribution of cracks along its length. The domain is discretised using a uniform grid of bilinear quadrilateral elements (Q4). We consider 4 meshes: 300×30, 600×60, 900×90 and 1200×120, and 6 randomly generated crack distributions of: 5, 10, 20, 30, 40 and 50 cracks of length 0.25. The rectangular plate is subjected to fixed-grip vertical extension loading conditions. The maximum hoop stress crack growth criterion [15] is used.

4.1 Verification of enrichment updating

Firstly, the computer implementations of the *efficient* enrichment updating approach is verified by showing that the numerical solutions to the fracture paths are close (in terms of machine precision) to the fracture paths obtained using the *basic* enrichment updating approach. Figure 3 shows some example fracture paths that were obtained for the different initial crack distributions. Figure 4 shows the convergence of the fracture paths as the mesh is refined for the test case of 50 randomly distributed cracks.

The maximum distance $d_{\max}$ between the fracture profiles obtained by the *efficient* and the *basic* enrichment updating implementations is computed using equation (1):

$$d_{\max} = \sqrt{\max_{j \in \{1,2,\ldots,n_{\text{crk}}\}} \max_{i \in \{1,2,\ldots,n_{\text{vtx}}\}} \left((\Delta x_i^j)^2 + (\Delta y_i^j)^2 \right)} \tag{1}$$

where the term in the parenthesis represents the square of the separation distance between corresponding vertex points of each crack. Table 1 summarises the results for the

maximum separation distance $d_{\max}$ between the final fracture profiles of each test case. The discrepancies between the two implementations are close to machine precision.

Table 1: Maximum distance $d_{\max}$ between the fracture profiles obtained by the two enrichment updating strategies, namely: the *efficient* and the *basic* approaches.

mesh size	number of cracks					
	5	10	20	30	40	50
300×30	3.99E-15	1.95E-15	2.13E-15	8.00E-15	2.67E-15	3.74E-15
600×60	9.17E-15	1.31E-14	7.26E-15	2.17E-14	2.15E-14	6.63E-15
900×90	2.85E-14	2.02E-14	1.08E-14	6.75E-14	8.97E-14	4.77E-14
1200×120	1.77E-14	2.37E-14	1.33E-14	8.45E-14	2.54E-13	4.84E-14

4.2 Computational speed-up

The total computational time of a simulation is composed of 5 parts, namely: *pre-processing*, *initial assembly*, *updated assembly*, *solution of the linear system* and *post-processing*. The first two parts are done only once whereas the last three need to be repeated with each time-step. Therefore, in simulating multi-crack growth over many times-steps the total computational time will be primarily composed of the *solution*, *post-processing* and *updating* times. For the solution of the linear system we use *Matlab*'s built-in direct solver (by calling the backslash operator "\"), which uses a Cholesky decomposition that is optimised for sparse symmetric positive definite matrices. The post-processing step involves mainly the computations of the stress intensity factors using the domain-form interaction integral approach [40, 39, 21], identification of the crack tips to grow and the management of crack coalescence. Our aim is to assess the computational speed-up of the *efficient* implementation compared to the *basic* implementation.

Figure 5 shows the total computational time spent in updating the discrete system (t_{upd}) using the *basic* approach relative to the total computational time (t_{tot}) of the simulation. Similarly, Figure 6 shows the total time spent in updating the discrete system using the *efficient* approach. Figure 7 gives the speed-up factor of the total simulation time achieved by the *efficient* approach relative to the *basic* approach. Finally, Figure 8 gives the speed-up factor as a function of the number of time-steps (or crack increments).

4.3 Discussion and general remarks

The numerical results show that the computational benefits of the *efficient* enrichment updating approach can be significant relative to the *basic* approach, especially for coarser meshes or for larger numbers of cracks. While the *basic* implementation leads to a major computational bottleneck in the total simulation time (refer to Figure 5), the *efficient* implementation, in comparison, cuts down significantly on this time (refer to Figure 6)

giving a considerable speed-up factor in the total simulation time. In the studied cases, the speed-up factor was from around 2 to 5 (refer to Figure 7). Although the speed-up factor tends to diminish with mesh refinement, the computational benefit is considerable even for moderate size problems of around a few hundred thousand DOFs.

We considered benchmark problems of multi-crack growth within a relatively thin strip such that for the different mesh sizes and numbers of cracks that we studied there was a large number of enriched elements relative to the total number of elements. This, in turn, gave rise to more *dramatic* speed-up factors. However, with mesh refinement the number of enriched elements will increase at a slower rate than the total number of elements. Consequently, the times spent updating the enrichment and post-processing the solution will further diminish relative to the time spent solving the linear system of XFEM equations. In other words, with mesh refinement, the total computational time will be dominated by the solution to the linear system, regardless of the implementation of the enrichment updating. Nonetheless, for solving practical fracture mechanics problem the *efficient* implementation is still very much desirable despite the diminishing return.

Finally, recall that our implementation relies a lot on computational memory to achieve faster computations. Various enrichment related data are pre-computed to speed up the enrichment updating and solution post-processing times. Although excessive reliance on computational memory can create a bottleneck by itself, the proposed implementation scales quite favourably with mesh refinement in terms of the memory usage. The ratio of the number of enriched elements to the total number of elements will decrease as the mesh is refined and, thus, the stored enrichment data will not grow at an outstanding rate. For example, the memory required by large discrete system will be predominantly used to store the global stiffness matrix followed by the typical mesh related data.

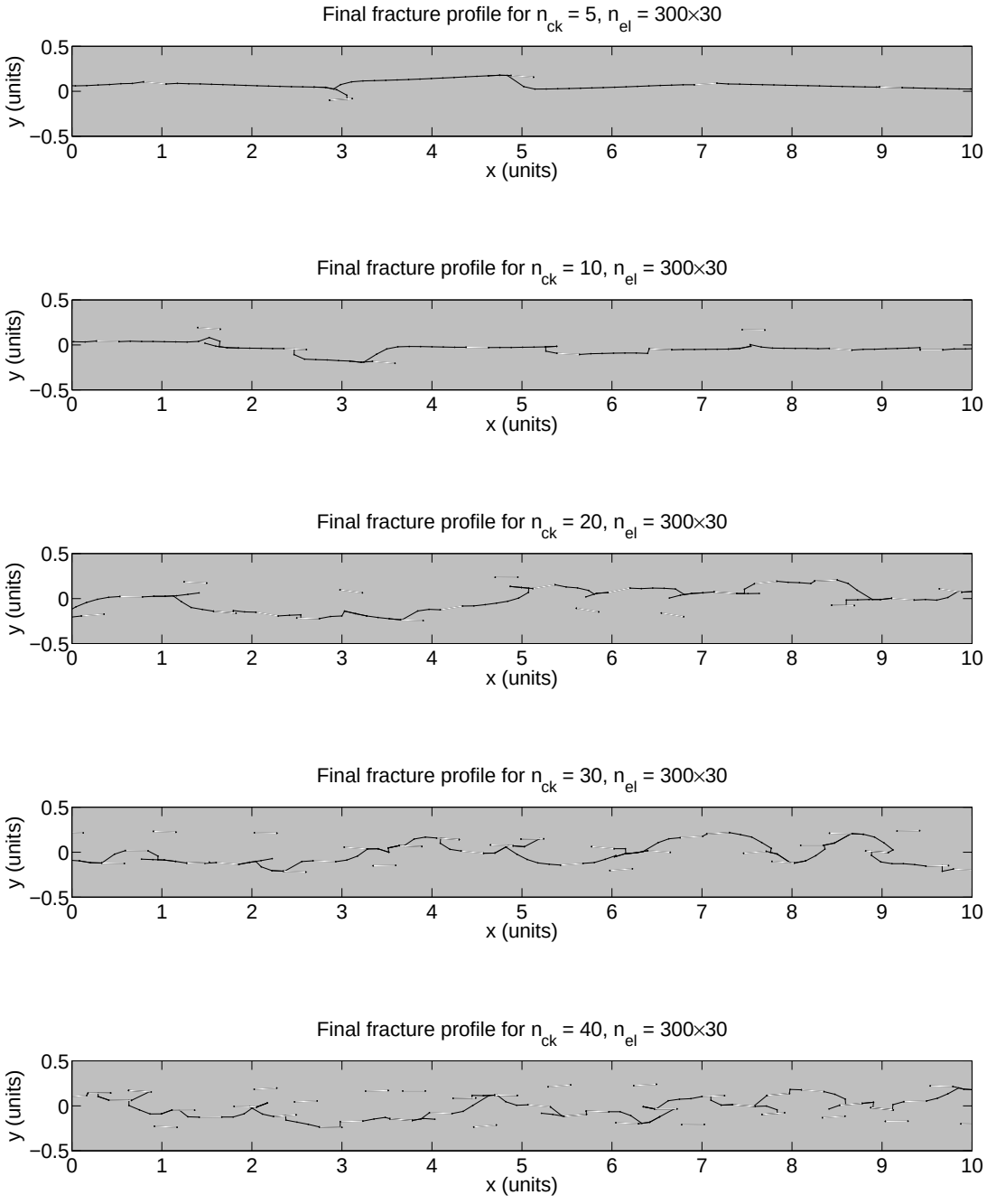


Figure 3: Fracture profiles for: $n_{\text{crk}} = \{5, 10, 20, 30, 40\}$, $n_{\text{elm}} = 300 \times 30$. The sub-figures show the type of fracture solutions obtained for each crack distribution. Note that the crack extension length is proportional to the mesh size.

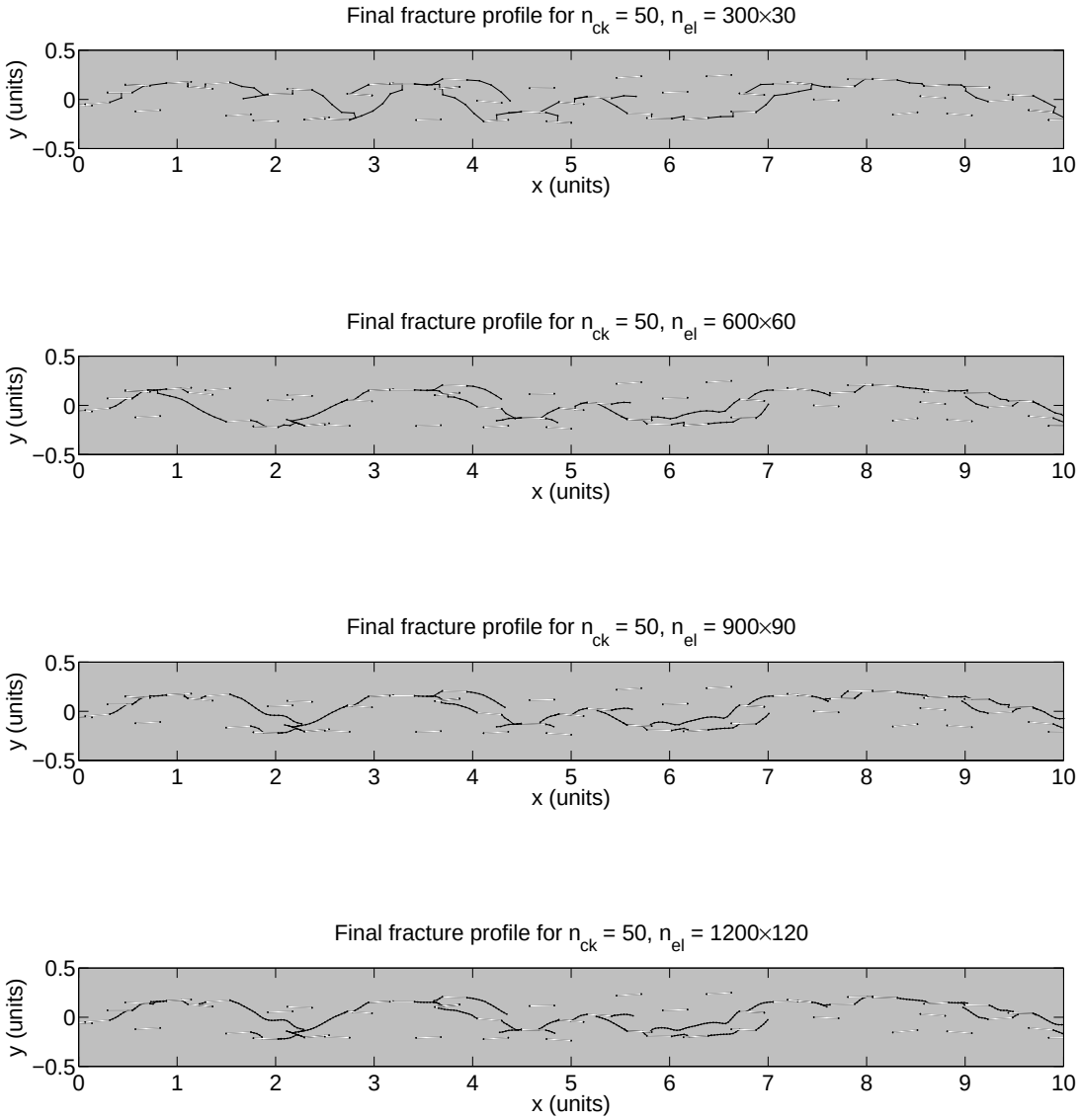


Figure 4: Illustration of the convergence of the fracture profiles for the case $n_{\mathrm{crk}} = 50$. Note that the crack extension length is proportional to the mesh size.

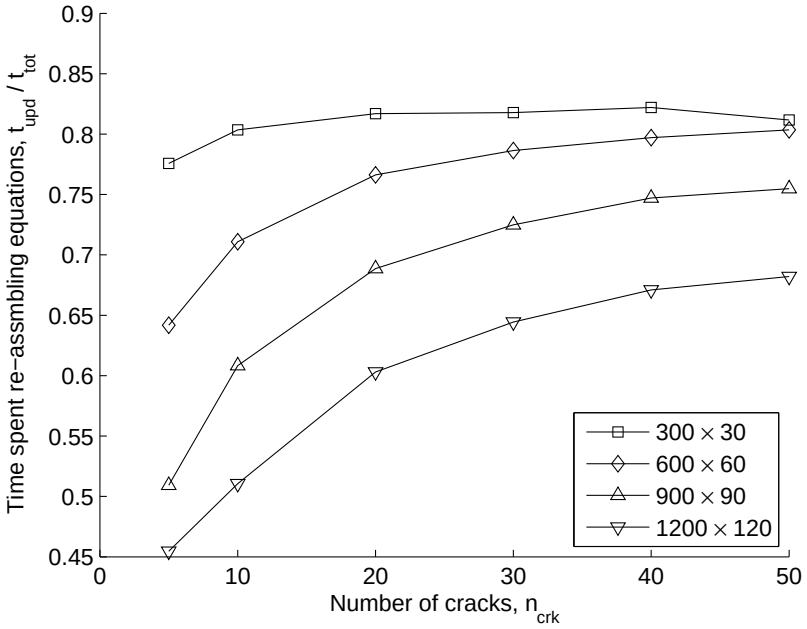


Figure 5: Total time spent updating the discrete system relative to the total computational time of the fracture simulation obtained by the *basic* implementation.

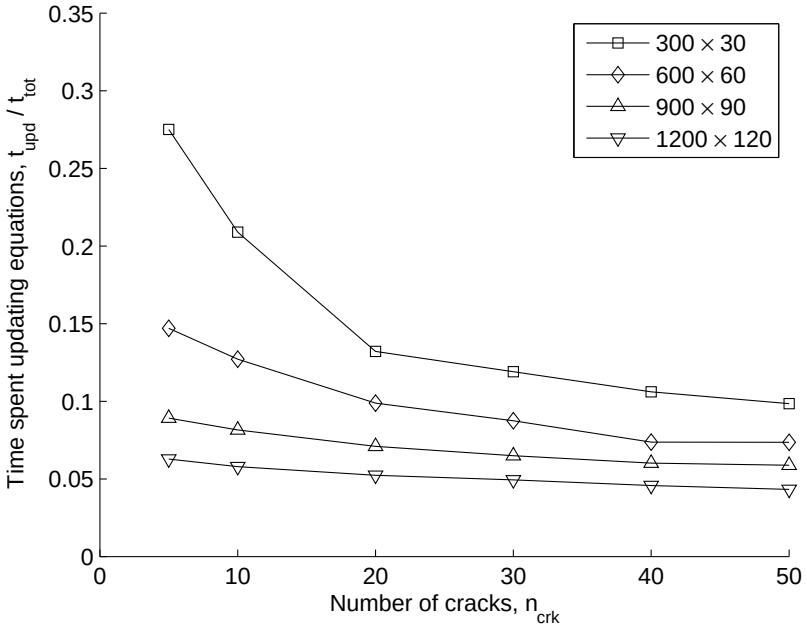


Figure 6: Total time spent updating the discrete system relative to the total computational time of the fracture simulation obtained by the *efficient* implementation.

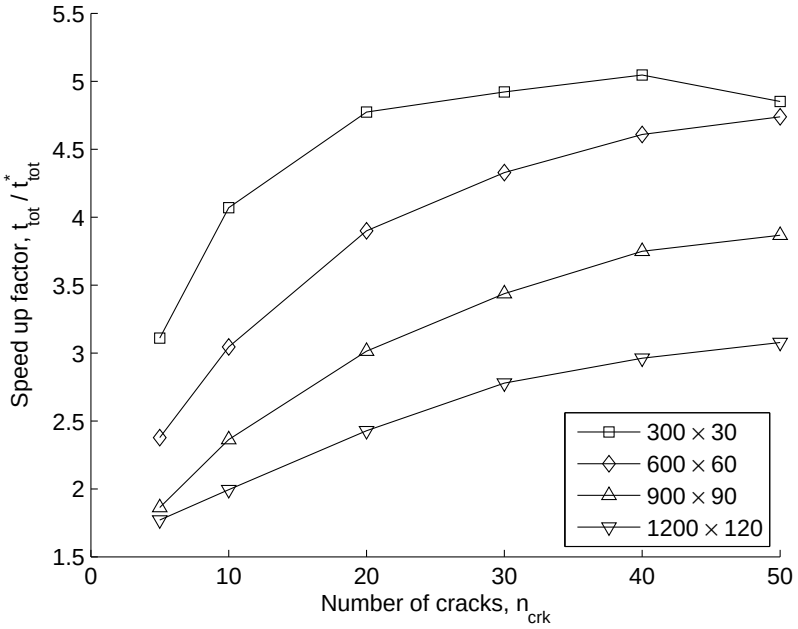


Figure 7: Speed-up in the total simulation time obtained by the *efficient* implementation relative to the *basic* one. (The superscript "$*$" denotes the *basic* implementation).

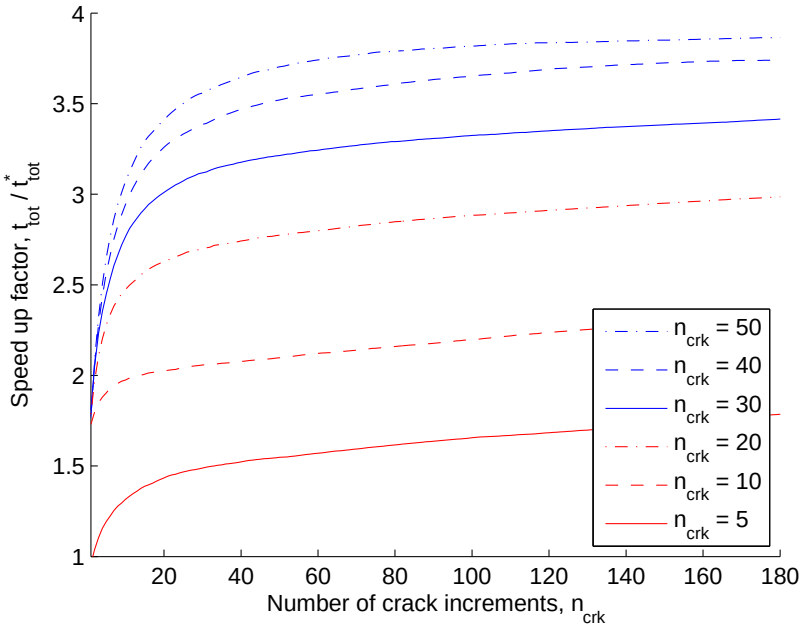


Figure 8: Speed-up in the simulation time as a function of the number of time-steps. The finest mesh (1200×120) is used. (The superscript "$*$" denotes the *basic* implementation).

5 Convergence of fracture paths

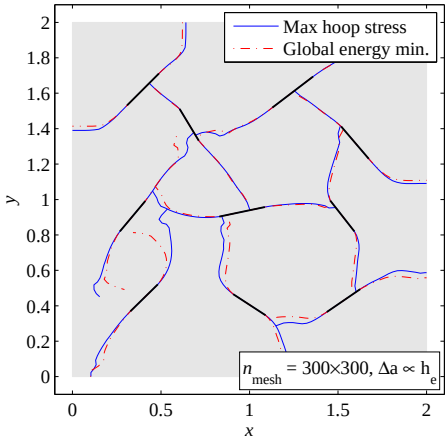
The maximum stress criterion is useful for solving linear-elastic fracture mechanics problems mainly because it is simple to apply. In Part-I and Part-II of this three-part paper we developed the minimum energy approach to multi-crack growth within the XFEM framework. It is of both theoretical and practical interest to compare the fracture paths by the two criteria and to assess the converge of the solutions with mesh refinement. This is the aim of the following benchmark cases involving multi-crack growth.

5.1 Square plate with 10 random cracks

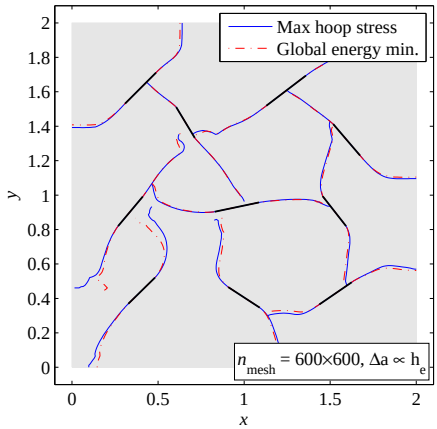
The present test case is a square plate with 10 randomly distributed cracks where the crack distribution is taken from [10]. Two types of boundary conditions are considered. The first case assumes a plate subjected to a bi-axial extension. The second case assumes internal pressure driven crack growth. In both cases, all cracks are allowed to grow at the same rate. The fracture paths obtained for the bi-axially loaded plate are shown in Figure 9. The apparent convergence of the fracture paths by the two criteria is plotted in Figure 11 in terms of the L_2-norm of the distance between the crack surfaces. Similarly, the fracture paths for the internal pressure driven cracks are shown in Figure 10. The apparent convergence of the fracture paths by the two criteria is plotted in Figure 12. Interestingly, the fracture paths obtained by two criteria appear to convergence towards the same solution with mesh refinement. The bi-axially loaded case yields an average convergence rate of 0.73 whereas the internal pressure loaded case yields 0.99.

5.2 Rectangular plate with 10 parallel cracks

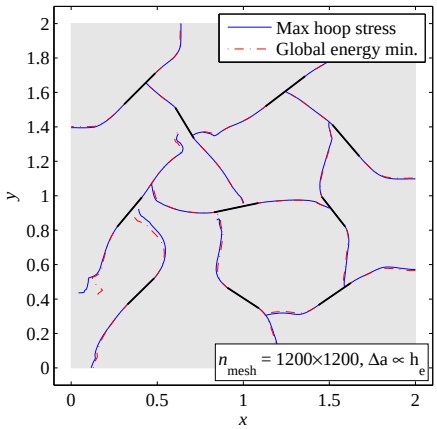
We consider a rectangular plate with 10 initially non-overlapping parallel cracks distributed randomly within a narrow width. The plate is simply supported and subjected to a uniform vertical tensile load. For the assumed crack distribution, crack growth is found to occur from left to right leading to complete horizontal splitting of the plate. The fracture solutions by the maximum hoop stress and the minimum energy criteria are computed for two types of finite element meshes, namely: quadrilateral (Q4) and triangular (T3) meshes. The fracture profiles obtained on Q4 and T3 meshes of similar numbers of DOFs are shown in Figures 13 and 14 respectively. The convergence plots of the fracture paths towards the same solution for Q4 and T3 meshes are shown in Figures 15 and 16, where the convergence rates are found to be 1.04 and 0.93 respectively.



(a) Q4 mesh 300 × 300.

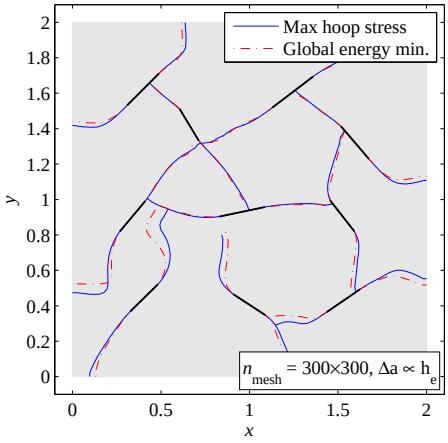


(b) Q4 mesh 600 × 600.

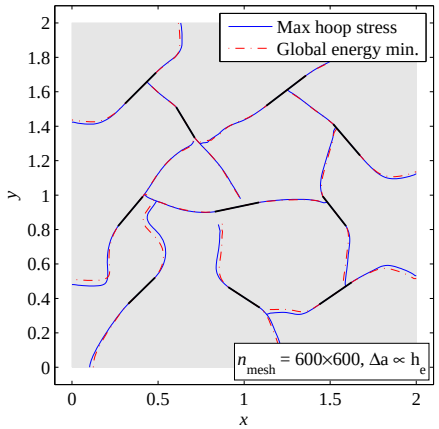


(c) Q4 mesh 1200 × 1200.

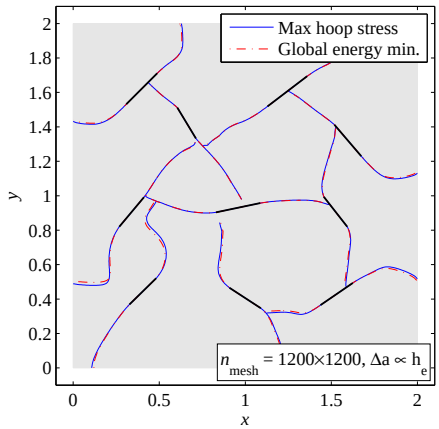
Figure 9: Fracture paths by the maximum stress and the minimum energy criteria. The plate is subjected to a bi-axial extension loading conditions. All cracks are allowed to grow at the same rate.



(a) Q4 mesh 300 × 300.



(b) Q4 mesh 600 × 600.



(c) Q4 mesh 1200 × 1200.

Figure 10: Fracture paths by the maximum stress and the minimum energy criteria. The cracks are subjected to a uniform pressure loading conditions. All cracks are allowed to grow at the same rate.

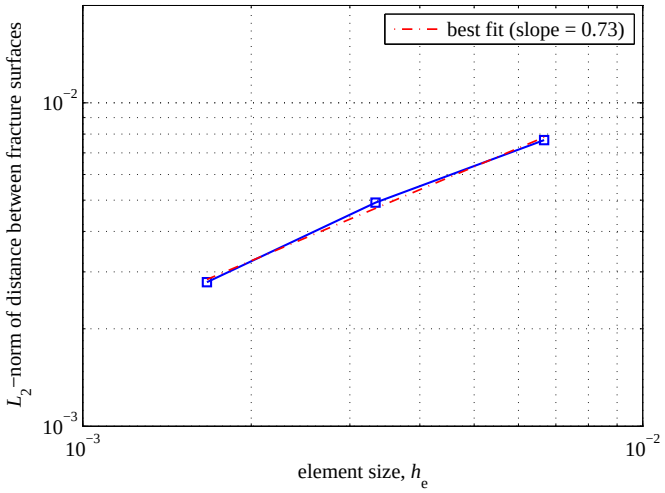


Figure 11: Convergence of the fracture paths by maximum stress and the minimum energy criteria towards the same solution. The test case is a square plate with 10 randomly distributed cracks. The plate is subjected to a bi-axial extension loading conditions. All cracks are allowed to grow at the same rate.

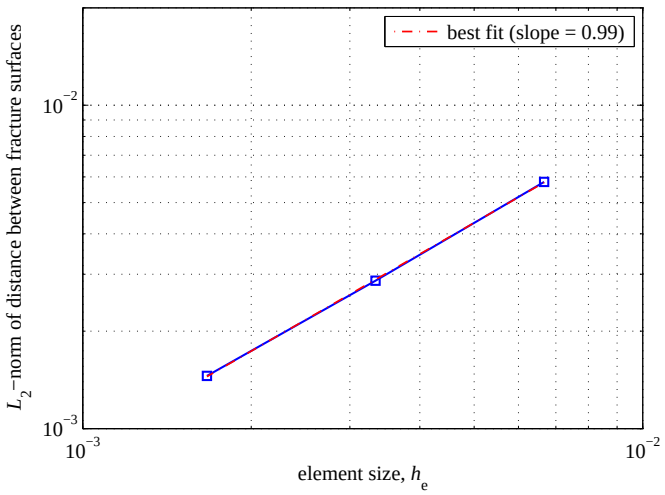
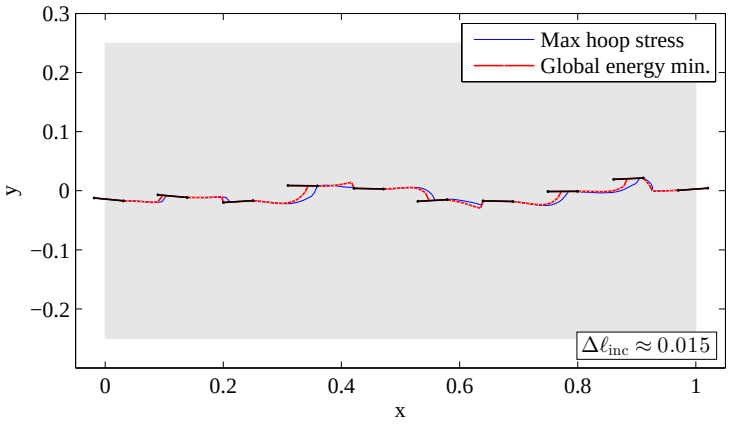
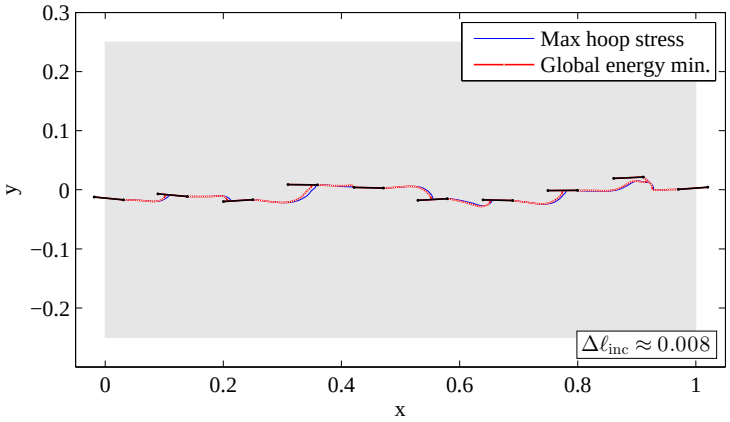


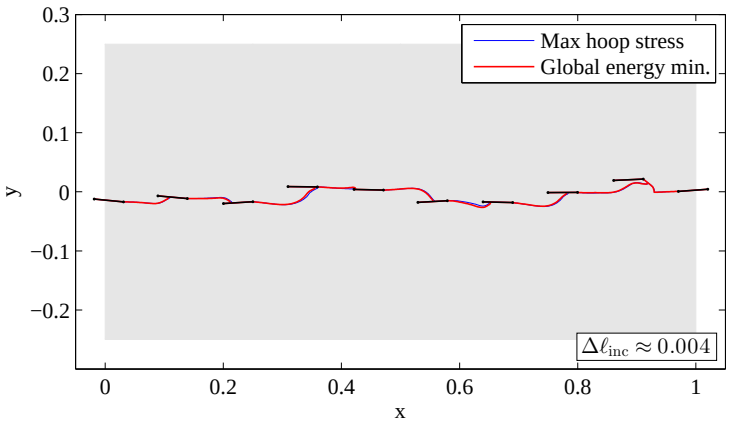
Figure 12: Convergence of the fracture paths by maximum stress and the minimum energy criteria towards the same solution. The test case is a square plate with 10 randomly distributed cracks. The cracks are subjected to a uniform pressure loading conditions. All cracks are allowed to grow at the same rate.



(a) Q4 mesh 100×200.

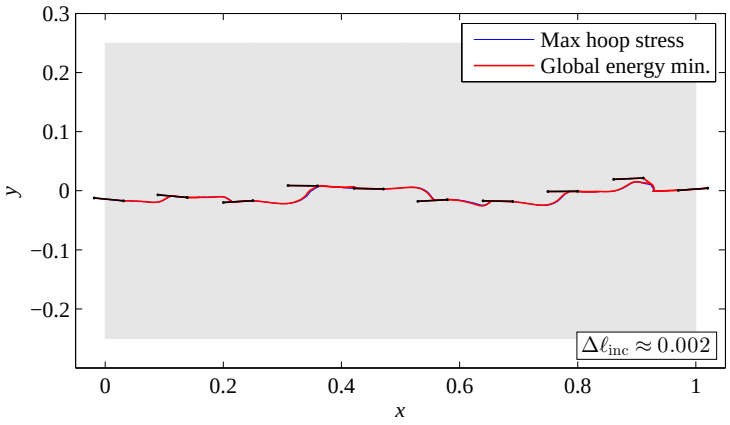


(b) Q4 mesh 200×400.

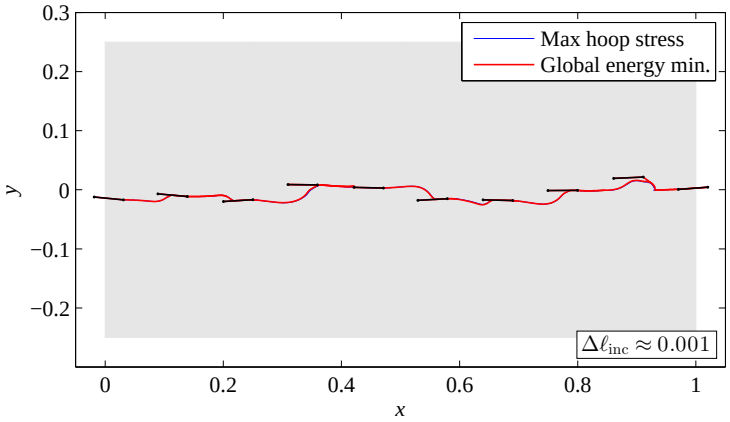


(c) Q4 mesh 400×800.

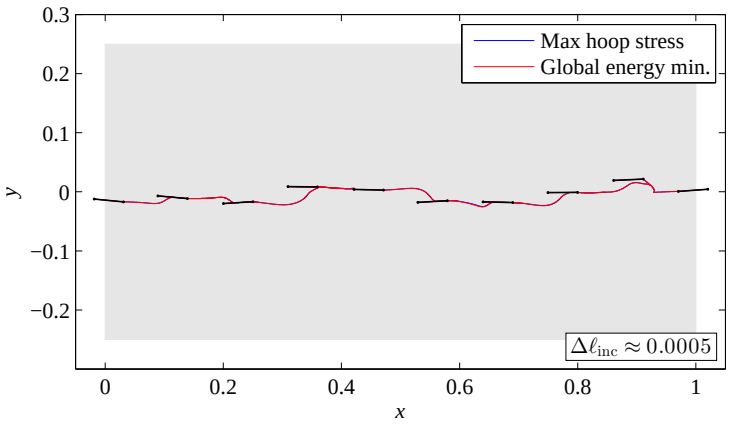
Figure 13: Fracture paths by the maximum stress and the minimum energy criteria. The plate is simply supported and subjected to a uniform vertical tension loading conditions. Q4 meshes are used.



(a) T3 mesh 100×200.



(b) T3 mesh 200×400.



(c) T3 mesh 400×800.

Figure 14: Fracture paths by the maximum stress and the minimum energy criteria. The plate is simply supported and subjected to a uniform vertical tension loading conditions. T3 meshes are used.

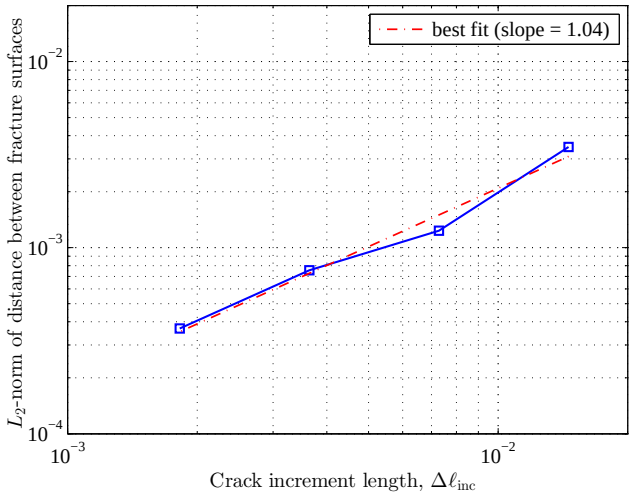


Figure 15: Convergence of the fracture paths by maximum stress and the minimum energy criteria towards the same solution. The test case is a simply supported plate in vertical tension with 10 narrowly distributed parallel and initially non-overlapping horizontal cracks. Q4 meshes were used.

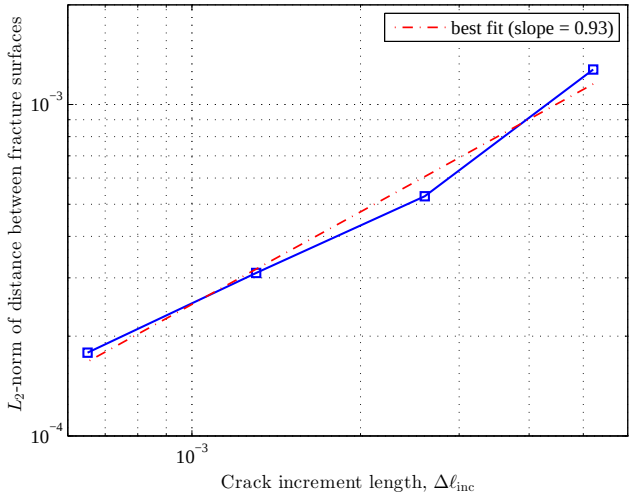


Figure 16: Convergence of the fracture paths by maximum stress and the minimum energy criteria towards the same solution. The test case is a simply supported plate in vertical tension with 10 narrowly distributed parallel and initially non-overlapping horizontal cracks. T3 meshes were used.

5.3 A numerical improvement to the growth direction

The preceding numerical results indicate that the solutions to the fracture path by the maximum stress and the minimum energy criteria converge towards very similar solutions when compared on the global length scale. For the range of XFEM discretions that were studied such convergence behaviours of the fracture paths were clearly observed. These results were obtained by the two criteria despite their inherent differences in the solutions to the incipient crack tip kink angles under general mixed-mode loading conditions, as demonstrated by the benchmark cases in Part-II of this three-part paper.

From inspection of the discrete solutions (including the solution to the benchmark cases from Part-II) it appears that the maximum stress criterion consistently underestimates the crack kink angles whereas the minimum energy solution tends to overestimate them. Refining the XFEM discretisation (and, thus, decreasing the crack extension length) leads the fracture solutions by the two criteria to convergence from opposite directions towards each other and towards a solution that lies close to the middle of the solutions obtained by the two criteria on coarser discretisations. Motivated by these observations, it may be possible to numerically improve the crack growth direction by assuming the average of the directions obtained by the stress and the energy based criteria.

It turns out that the accuracy and the convergence rate of the numerical solution to the fracture paths can be significantly improved by this approach. For the sake of brevity, we will refer to the proposed modification as the *bi-section* method. Satisfactory performance of this method was obtained in all the test cases that were attempted. The fracture solutions by the different criteria, namely: the maximum stress, the minimum energy and the *bi-section* method are presented in the subsequent benchmark studies.

5.4 Comparison of fracture paths by different criteria

Several 2D benchmark cases are proposed for the comparison the fracture paths by the three criteria, namely: maximum stress, minimum energy, and the proposed *bi-section* method which assumes the average direction obtained by maximum stress and minimum energy criteria at each time-step. The fracture paths for each test case are computed for different XFEM discretisation densities. It is found in every case that the maximum stress and minimum energy criteria converge towards very similar fracture paths; however, the *bi-section* method appears to converge to this solution the fastest.

The subsequent results of the benchmark cases verify that the *bi-section* method can be useful for improving the accuracy and the convergence rate of the fracture paths with mesh refinement. The *bi-section* method is most effective for fracture evolutions that do not involve crack intersections. The reason is that the position of a crack intersection tends to have a strong influence on the subsequent fracture paths (particularly on courser discretisations). The benefits of the *bi-section* approach are less apparent if the

discretisation is already well refined because the fracture paths by the stress and the energy based criteria tend to already be in a very close proximity, e.g. Figure 20.

6 Summary

It was shown via solutions to multiple benchmark problems of multi-crack growth with coalescence that the *efficient* enrichment updating approach was implemented within *Matlab* correctly as the fracture solutions coincided within machine precision with those obtained by the *basic* enrichment updating approach, which essentially involved reassembling the entire enrichment and recomputing the entire enriched part of the stiffness matrix from scratch at each time-step. With regard to the computational times, it was shown that the so-called *basic* implementation created a major computational bottleneck in the total simulation time, e.g. Figure 5. On the other hand, the so-called *efficient* implementation resulted in significantly faster computational times, e.g. Figure 6.

One of the contributions of this paper is the comparison of different crack growth criteria. The maximum stress and minimum energy criteria – despite their inherent differences in the incipient crack tip kink angles under general mixed-mode loading conditions – were found to yield very similar fracture solutions when compared on the global length scale. From the multiple benchmark cases that were studied, the fracture solutions converged to very similar fracture paths and at similar rates but from opposite directions.

Finally, a numerical improvement to the crack growth direction was proposed. The so-called *bi-section* method averaged the crack growth directions obtained by the maximum stress and minimum energy criteria. Despite the method's lack of a sound physical basis, it was found to be useful for numerically improving the accuracy of either criterion and for speeding-up the convergence of the fracture paths with discretisation refinement.

7 Supplementary material

The open-source code XFEM_Fracture2D and supporting material can be found here:

- XFEM_Fracture2D: https://figshare.com/s/0b4394e8fab7191d2692
- competing cracks: https://figshare.com/s/4a7dd5fb0a8634c9fae4
- demo screenshots: https://figshare.com/s/6397737c78beb59f3b58
- demo movies: https://figshare.com/s/73d7b50a7729070c2173

Double cantilever problem

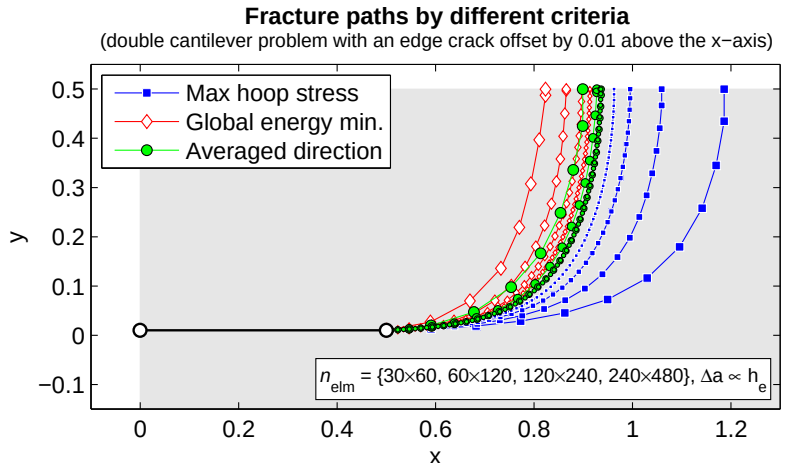


Figure 17: Fracture paths by different growth criteria for the double cantilever problem with the initial crack positioned 0.01 above the x-axis. The prying action is exerted by prescribed displacements on the left edge.

Two edge crack problem (simple tension loading)

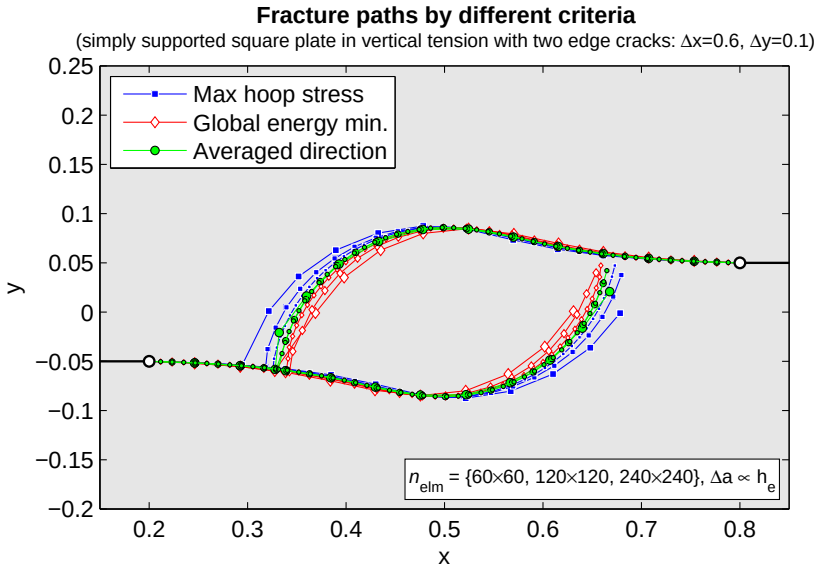


Figure 18: Fracture paths by different criteria for a simply supported square plate (1×1) in simple vertical tension with two initial edge cracks ($a_1 = a_2 = 0.2$). Crack-tip separation: $\Delta x = 0.6$, $\Delta y = 0.10$.

Three crack problem (pressure loaded centre crack)

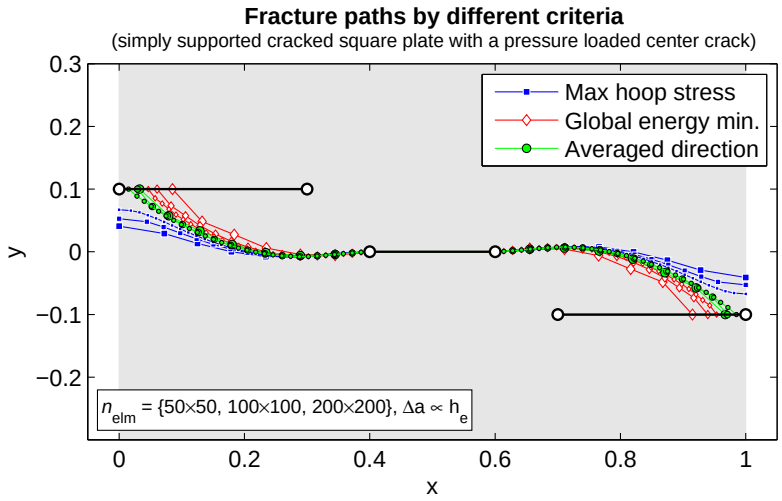


Figure 19: Fracture paths by different growth criteria for a simply supported square plate with three pre-existing cracks, where the centre crack is subjected to a pressure load acting normal to the crack surface.

Two cracks protruding from holes (vertical tension load)

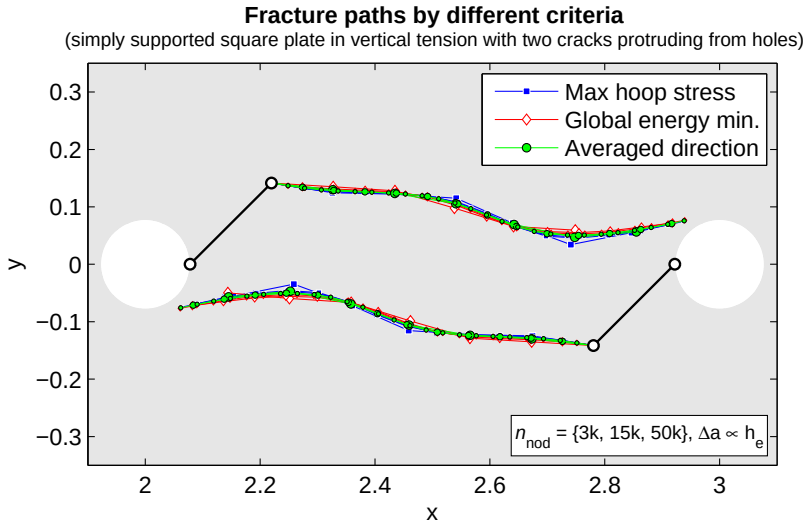


Figure 20: Fracture paths by different growth criteria for a simply supported square plate with a pair of initial cracks protruding from holes.

Two edge crack problem (crack pressure loading)

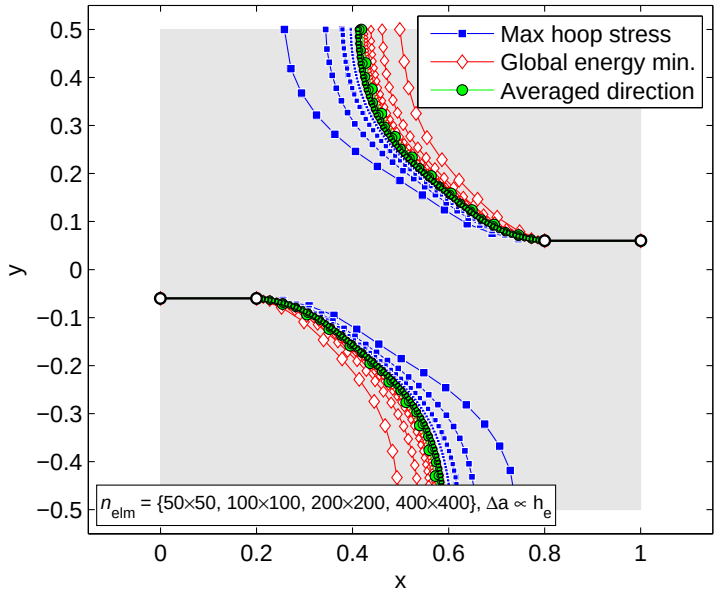


Figure 21: Fracture paths by different criteria for a simply supported square plate (1×1) with two initial edge cracks ($a_1 = a_2 = 0.2$) that are loaded by pressure. Crack-tip separation: $\Delta x = 0.6$, $\Delta y = 0.12$.

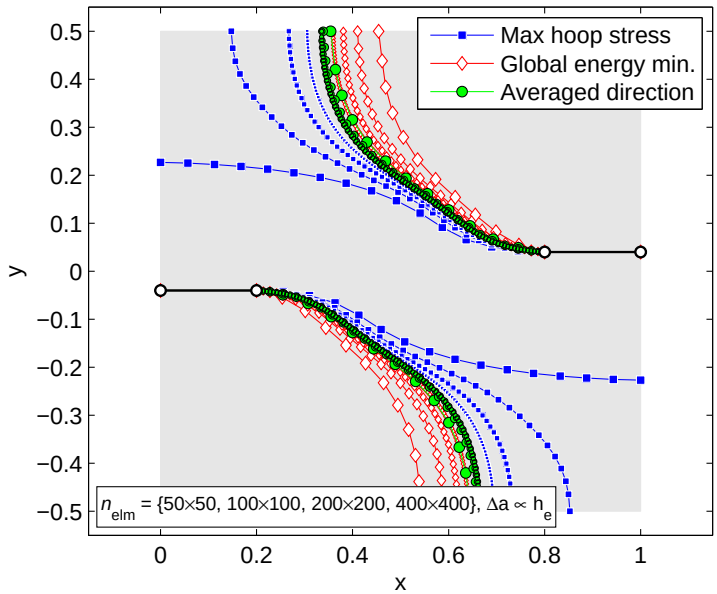


Figure 22: Fracture paths by different criteria for a simply supported square plate (1×1) with two initial edge cracks ($a_1 = a_2 = 0.2$) that are loaded by pressure. Crack-tip separation: $\Delta x = 0.6$, $\Delta y = 0.08$.

Two-edge crack problem (crack pressure loading) [cont.]

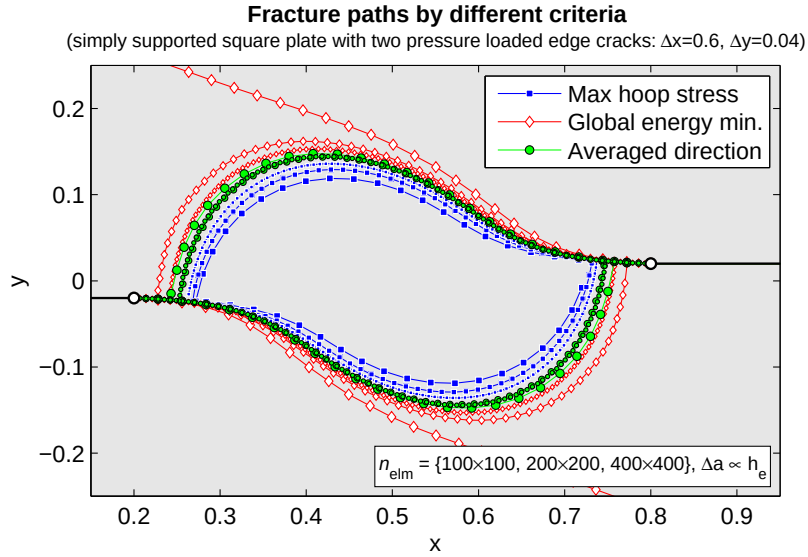


Figure 23: Fracture paths by different criteria for a simply supported square plate (1×1) with two initial edge cracks ($a_1 = a_2 = 0.2$) that are loaded by pressure. Crack-tip separation: $\Delta x = 0.6$, $\Delta y = 0.04$.

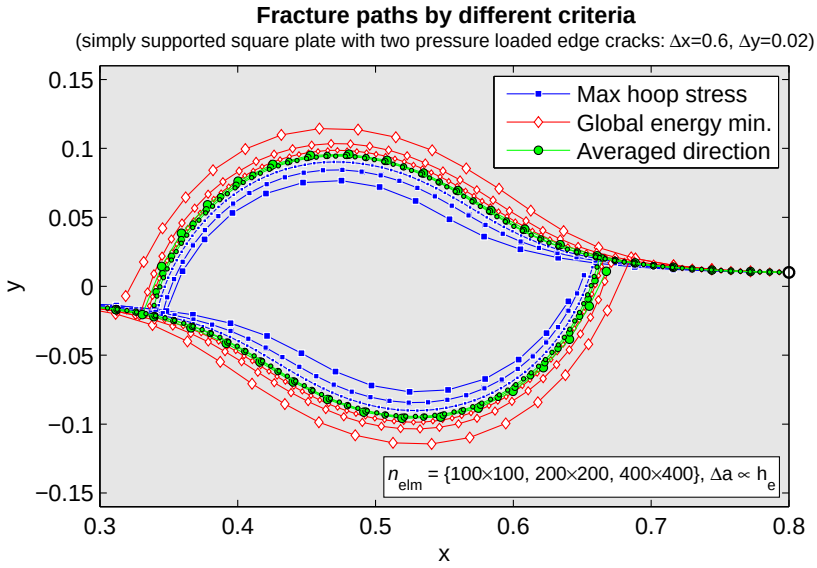
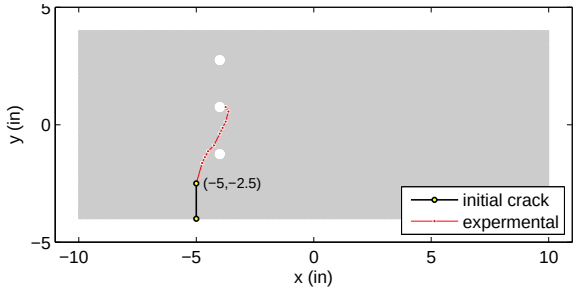
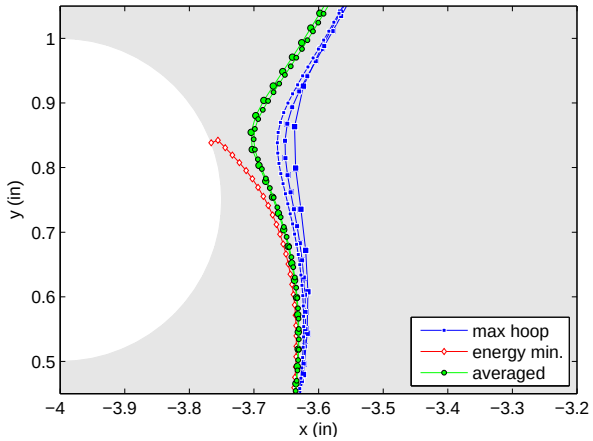


Figure 24: Fracture paths by different criteria for a simply supported square plate (1×1) with two initial edge cracks ($a_1 = a_2 = 0.2$) that are loaded by pressure. Crack-tip separation: $\Delta x = 0.6$, $\Delta y = 0.02$.

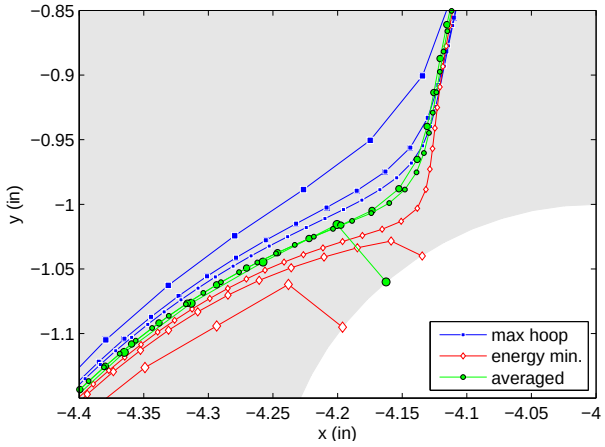
The PMMA beam with a bottom slit (case-1)



(a) Schematic of the PMMA beam with a bottom crack; A concentrated point load is applied in the middle of the top face.



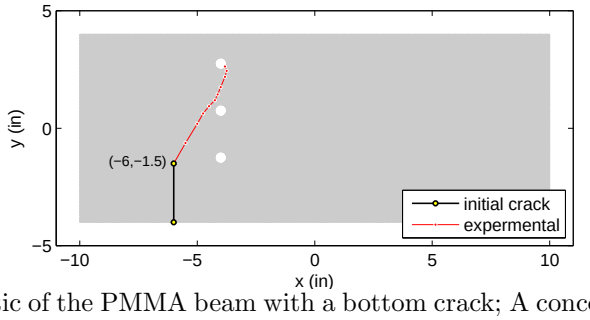
(b) A close-up view of the fracture paths around the middle hole of the beam.



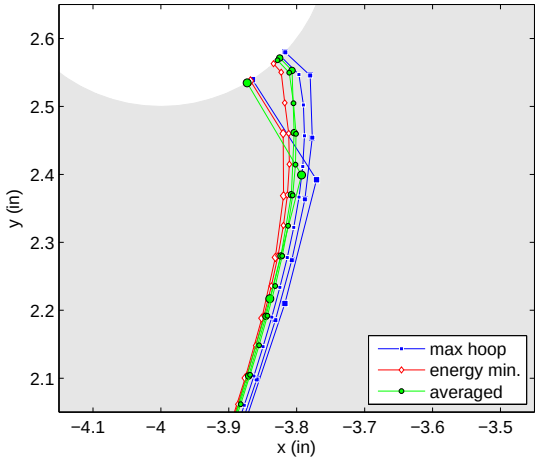
(c) A close-up view of the fracture paths around the bottom hole of the beam.

Figure 25: A simply supported 4×10 (in) PMMA beam with an initial vertical slit of length $a = 1.5$ (in) and a point load mid-way the top-edge. The fracture paths by different criteria are compared.

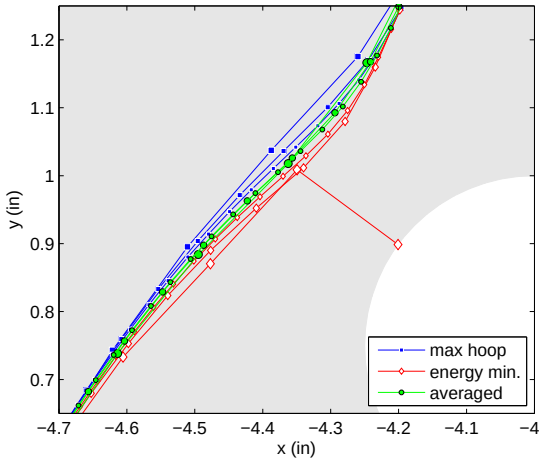
The PMMA beam with a bottom slit (case-2)



(a) Schematic of the PMMA beam with a bottom crack; A concentrated point load is applied in the middle of the top face.



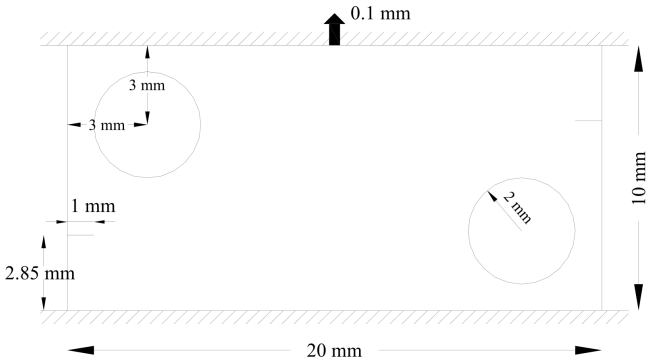
(b) A close-up view of the fracture paths around the top hole of the beam.



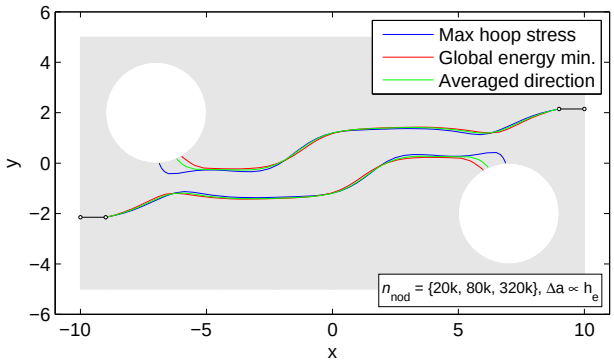
(c) A close-up view of the fracture paths around the middle hole of the beam.

Figure 26: A simply supported 4×10 (in) PMMA beam with an initial vertical slit of length $a = 2.5$ (in) and a point load mid-way the top-edge. The fracture paths by different criteria are compared.

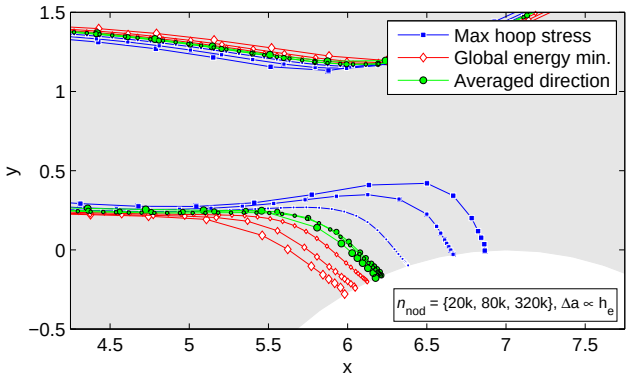
A pre-cracked plate with two holes



(a) Schematic diagram of the pre-cracked part. (Source: [9])



(b) Outline of fracture paths by different criteria.



(c) Close-up view of sub-figure (b) around the hole on the right.

Figure 27: Fracture paths by different criteria for a rectangular plate with two holes and two edge cracks subjected to a vertical extension.

References

[1] K. Agathos, E. Chatzi, and S. P. A. Bordas. "Stable 3D extended finite elements with higher order enrichment for accurate non planar fracture". In: *Computer Methods in Applied Mechanics and Engineering* 306.October 2015 (2016), pp. 19–46. ISSN: 00457825.

[2] K. Agathos et al. "A well-conditioned and optimally convergent XFEM for 3D linear elastic fracture". In: *International Journal for Numerical Methods in Engineering* 105.9 (2016), pp. 643–677. ISSN: 00295981.

[3] I. Anjam and J. Valdman. "Fast MATLAB assembly of FEM matrices in 2D and 3D: Edge elements". In: *Applied Mathematics and Computation* 267.13 (2015), pp. 252–263. ISSN: 00963003. arXiv: `arXiv:1409.4618v1`.

[4] T. Belytschko et al. "Arbitrary discontinuities in finite elements". In: *International Journal for Numerical Methods in Engineering* 50.4 (2001), pp. 993–1013. ISSN: 0029-5981.

[5] T. Belytschko and T. Black. "Elastic crack growth in finite elements with minimal remeshing". In: *International Journal for Numerical Methods in Engineering* 45.5 (1999), pp. 601–620. ISSN: 0029-5981.

[6] T. Belytschko, R. Gracie, and G. Ventura. "A review of extended/generalized finite element methods for material modeling". In: *Modelling and Simulation in Materials Science and Engineering* 17.4 (2009), p. 043001. ISSN: 0965-0393.

[7] E. Benvenuti et al. "Variationally consistent eXtended FE model for 3D planar and curved imperfect interfaces". In: *Computer Methods in Applied Mechanics and Engineering* 267 (2013), pp. 434–457. ISSN: 00457825.

[8] S. P. A. Bordas et al. "An extended finite element library". In: *International Journal for Numerical Methods in Engineering* 71.6 (2007), pp. 703–732. ISSN: 00295981.

[9] P. Bouchard, F. Bay, and Y. Chastel. "Numerical modelling of crack propagation: automatic remeshing and comparison of different criteria". In: *Computer Methods in Applied Mechanics and Engineering* 192.35-36 (2003), pp. 3887–3908. ISSN: 00457825.

[10] E. Budyn et al. "A method for multiple crack growth in brittle materials without remeshing". In: *International Journal for Numerical Methods in Engineering* 61.10 (2004), pp. 1741–1770. ISSN: 0029-5981.

[11] A. Cano and C. Moreno. "Transformation Methods for the Numerical Integration of Three-Dimensional Singular Functions". In: *Journal of Scientific Computing* (2016). ISSN: 0885-7474.

[12] J. Chessa, H. Wang, and T. Belytschko. "On the construction of blending elements for local partition of unity enriched finite elements". In: *International Journal for Numerical Methods in Engineering* 57.7 (2003), pp. 1015–1038. ISSN: 0029-5981.

[13] C. Daux et al. "Arbitrary branched and intersecting cracks with the extended finite element method". In: *International Journal for Numerical Methods in Engineering* 48.12 (2000), pp. 1741–1760. ISSN: 0029-5981.

[14] M. G. Duffy. “Quadrature Over a Pyramid or Cube of Integrands with a Singularity at a Vertex”. In: *SIAM Journal on Numerical Analysis* 19.6 (1982), pp. 1260–1262. ISSN: 0036-1429.

[15] F. Erdogan and G. C. Sih. “On the Crack Extension in Plates Under Plane Loading and Transverse Shear”. In: *Journal of Basic Engineering* 85.4 (1963), p. 519. ISSN: 00219223.

[16] M. Fleming and Y. A. Chu. “Enriched element-free Galerkin methods for crack tip fields”. In: *International Journal for Numerical Methods in Engineering* 40.January 1996 (1997), pp. 1483–1504.

[17] T. P. Fries. “A corrected XFEM approximation without problems in blending elements”. In: *International Journal for Numerical Methods in Engineering* 75.5 (2008), pp. 503–532. ISSN: 00295981.

[18] T. P. Fries. “Overview and comparison of different variants of the XFEM”. In: *PAMM* 14.1 (2014), pp. 27–30. ISSN: 16177061.

[19] T. P. Fries and T. Belytschko. “The extended/generalized finite element method: An overview of the method and its applications”. In: *International Journal for Numerical Methods in Engineering* (2010), n/a–n/a. ISSN: 00295981.

[20] E. Gekeler. *Mathematical Methods for Mechanics*. 2008, p. 636. ISBN: 9783540302674. arXiv: `arXiv:1011.1669v3`.

[21] V. F. González-Albuixech et al. “Domain integral formulation for 3-D curved and non-planar cracks with the extended finite element method”. In: *Computer Methods in Applied Mechanics and Engineering* 264 (2013), pp. 129–144. ISSN: 00457825.

[22] V. Gupta et al. “Stable GFEM (SGFEM): Improved conditioning and accuracy of GFEM/XFEM for three-dimensional fracture mechanics”. In: *Computer Methods in Applied Mechanics and Engineering* 289 (2015), pp. 355–386. ISSN: 00457825.

[23] V. Gupta and C. A. Duarte. “On the enrichment zone size for optimal convergence rate of the Generalized/Extended Finite Element Method”. In: *Computers & Mathematics with Applications* (2016). ISSN: 08981221.

[24] R. Huang, N. Sukumar, and J. H. Prévost. “Modeling quasi-static crack growth with the extended finite element method Part II: Numerical applications”. In: *International Journal of Solids and Structures* 40.26 (2003), pp. 7539–7552. ISSN: 00207683.

[25] P. Laborde et al. “High-order extended finite element method for cracked domains”. In: *International Journal for Numerical Methods in Engineering* 64.3 (2005), pp. 354–381. ISSN: 0029-5981.

[26] S. Linge and H. P. Langtangen. *Programming for Computations - MATLAB/Octave*. Vol. 14. Texts in Computational Science and Engineering. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-32451-7.

[27] J. Melenk and I. Babuška. “The partition of unity finite element method: Basic theory and applications”. In: *Computer Methods in Applied Mechanics and Engineering* 139.1-4 (1996), pp. 289–314. ISSN: 00457825.

[28] H. Minnebo. "Three-dimensional integration strategies of singular functions introduced by the XFEM in the LEFM". In: *International Journal for Numerical Methods in Engineering* 92.13 (2012), pp. 1117–1138. ISSN: 00295981.

[29] N. Moës, J. Dolbow, and T. Belytschko. "A finite element method for crack growth without remeshing". In: *Int. J. Numer. Meth. Engng* 150.February (1999), pp. 131–150. ISSN: 00295981.

[30] S. Natarajan. "Enriched finite element methods: advances & applications". PhD thesis. Cardiff University, 2011.

[31] V. P. Nguyen. "An object-oriented approach to the extended finite element method with applications to fracture mechanics". MSc Thesis. Hochiminh City University of Technology, 2005.

[32] V. P. Nguyen et al. "Isogeometric analysis: An overview and computer implementation aspects". In: *Mathematics and Computers in Simulation* 117 (2015), pp. 89–116. ISSN: 03784754.

[33] V. P. Nguyen et al. "Meshless methods: A review and computer implementation aspects". In: *Mathematics and Computers in Simulation* 79.3 (2008), pp. 763–813. ISSN: 03784754.

[34] T. Rabczuk, S. Bordas, and G. Zi. "On three-dimensional modelling of crack growth using partition of unity methods". In: *Computers & Structures* 88.23-24 (2010), pp. 1391–1411. ISSN: 00457949.

[35] H. Talebi, M. Silani, and T. Rabczuk. "Concurrent multiscale modeling of three dimensional crack and dislocation propagation". In: *Advances in Engineering Software* 80.C (2015), pp. 82–92. ISSN: 09659978.

[36] H. Talebi et al. "A computational library for multiscale modeling of material failure". In: *Computational Mechanics* 53.5 (2014), pp. 1047–1071. ISSN: 0178-7675.

[37] G. Ventura, E. Budyn, and T. Belytschko. "Vector level sets for description of propagating cracks in finite elements". In: *International Journal for Numerical Methods in Engineering* 58.10 (2003), pp. 1571–1592. ISSN: 0029-5981.

[38] G. Ventura, R. Gracie, and T. Belytschko. "Fast integration and weight function blending in the extended finite element method". In: *International Journal for Numerical Methods in Engineering* 77.1 (2009), pp. 1–29. ISSN: 00295981.

[39] M. C. Walters, G. H. Paulino, and R. H. Dodds. "Interaction integral procedures for 3-D curved cracks including surface tractions". In: *Engineering Fracture Mechanics* 72.11 (2005), pp. 1635–1663. ISSN: 00137944.

[40] J. F. Yau, S. S. Wang, and H. T. Corten. "A Mixed-Mode Crack Analysis of Isotropic Solids Using Conservation Laws of Elasticity". In: *Journal of Applied Mechanics* 47.2 (1980), p. 335. ISSN: 00218936.